FONDAMENTI DI INFORMATICA-JAVA

ESERCIZIARIO

Ingegneria Gestionale
Canale II

2006-2007

Presentazione

Per i neofiti della materia, la programmazione, ovvero la scrittura di programmi in un linguaggio di programmazione, può essere tanto ostica quanto di difficile comprensione: la mentalità e la logica con cui un programmatore affronta un problema è, spesso, molto diversa da quella usata nella realtà.
Concetti come classi, metodi, costruttori o variabili possono sembrare oscuri ai più, ma un buon metodo di studio e, soprattutto, una continua pratica sulla materia possono facilitare il superamento delle difficoltà iniziali. E' con quest'intento che nasce il seguente eserciziario: introdurre una materia le cui potenzialità sono facilmente riscontrabili nella vita odierna, in cui la tecnologia la fa da padrona. Soprattutto il linguaggio Java: disponibile gratuitamente sulla Rete, esso permette la creazione di applicazioni eseguibili su un'ampia gamma di computer e di sistemi operativi diversi e, soprattutto, è divenuto lo standard per lo sviluppo delle applicazioni integrate nelle pagine del World Wide Web, o meglio conosciuto come Internet, o di dispositivi quali cellulari o qualsiasi sistema che si basi su un controllo informatico, come i GPS e sensori.
Molti degli esercizi presenti in codesto testo sono stati ricavati dal libro adottato a lezione:
"Concetti di informatica e fondamenti di Java" (Cay H. Horstmann, Apogeo Edizioni). Altri, invece, sono esercitazioni proposte in laboratorio dal docente Giorgio Satta (per maggiori informazioni, si rimanda alla sua pagina personale:http://www.dei.unipd.it/~satta/) o nel forum da lui gestito (http://forum.dei.unipd.it/, sezione Fond. di Informatica JAVA (VICENZA)).
L'eserciziario, in ogni caso, non potrà mai sostituire le nozioni espresse in classe o nel libro di testo: esso è una guida che, come Virgilio nella Divina Commedia, accompagna gli studenti all'approfondimento del linguaggio Java.

Struttura:

L'eserciziario è suddiviso in tre parti fondamentali:

- Esercizi del libro: risoluzione di parte degli esercizi presenti nel testo ufficiale. Essa è ulteriormente suddivisa in capitoli, all'inizio dei quali si riassumono i concetti fondamentali
- Esercizi stile esame: raccolta di esercizi svolti in classe o a lezione
- Esercizi d'esame: esercizi assegnati durante gli esami

Ogni esercizio, inoltre, è strutturato nel seguente modo:

- Specchietto: concetti non spiegati a lezione ma introdotti dagli autori dei programmi
- Testo: consegna dell'esercizio
- Consigli: suggerimenti sulle modalità di sviluppo dei programmi. La loro utilità è puramente soggettiva.

Autori:

Bressan Marco (Brex)
Da Giau Alessandro(Frollo)
Dal Maso Alberto
Fioretto Giulio (700hz)
Gallo Amanda (Fly)
Gecchele Stefano (Crystal)
Obradovic Tijana (One)
Ronchi Fabrizio
Zulian Marco (Zuzi)

Suggerimenti

JOptionPane: la classe JOptionPane, un sistema utilizzato in qualche esercizio di questa raccolta è sostanzialmente simile alla classe Scanner ma, per informazioni più esaurienti, s'invita a leggere il libro a pag. 27 (Capitolo 1), pag. 126 (cap. 4), pag. 299 (cap. 9).

JCreator e JDK: Il primo è un ambiente di sviluppo per Java, il secondo è la Java Virtual Machine (JVM). Ecco i link da dove è possibile scaricarli:

JCreator: http://www.jcreator.com/download.htm , si consiglia il download nella versione LE (light edition)

JDK: http://java.sun.com/j2se/1.5.0/download.jsp (versione 1.5 o più recente)

Documentazione API: è l'insieme di tutte le librerie di Java.
È reperibile al link:
http://java.sun.com/j2se/1.5/docs/api/index.html e scaricabile dal link: http://java.sun.com/j2se/downloads/index.html

CAPITOLO 1 Introduzione

REMIND:

(Paragrafo 1.3)

- Java Virtual Machine: → CPU ideale, simulata da un programma in esecuzione sulla CPU effettiva. La JVM perché i codici macchina delle varie CPU (Pentium, Sparc, etc.) hanno linguaggi diversi per le istruzioni macchina. Così facendo tutte le macchine utilizzano lo stesso linguaggio (JVM) → Java può essere codificato indistintamente su qualsiasi macchina.
- Difficile scrivere in linguaggio macchina → codificato come numeri.
- Compilatore → traduce enunciati logici del linguaggio ad alto livello (Java) in sequenze di calcoli del codice macchina.

(Paragrafo 1.4)

- Java pensato per internet, due qualità:
  - sicurezza → caratteristiche si sicurezza per garantire che non si possa scrivere applet[1] nocivi;
  - trasferibilità → Java opera senza bisogno di modifiche su tutti i sistemi operativi (JVM).

(Paragrafo 1.7) Errori

[1] APPLET: Applicazioni java che vengono eseguite all'interno di broswer web. Servono per visualizzare oggetti dinamici che altrimenti con l'html sarebbero difficili da mostrare. Per visualizzarli serve la JVM.

- ERRORE DI SINTASSI → vi è qualcosa di sbagliato secondo le regole del linguaggio e il compilatore lo trova.
- ERRORE DI ESECUZIONE (o LOGICO) → il programma è corretto sintatticamente e fa qualcosa, ma non quello che ci si aspettava

Esercizio 1.1 - HelloTester.java

Testo:
Scrivere un programma che stampi sullo schermo il messaggio "Hello, World!".

Consigli:
/

```
public class HelloTester
{
	public static void main(String[] args)
	{
		// sullo schermo, viene stampato un messaggio di saluti
		System.out.println("Hello, World");
	}
}
```

Esercizio 1.2 - Dave.java (Esercizio P1.1 pag. 25)

Testo:
Scrivere un programma che visualizzi sullo schermo del terminale il vostro nome all'interno di un rettangolo, come nell'esempio seguente:

```
+ -  - +
| Dave |
+ -  - +
```

Consigli:
/

```
public class Dave
{
	public static void main(String[] args)
	{
		// si stampa la cornice
		System.out.println("+ -  - +");
		// si stampa il nome
		System.out.println("| Dave |");
		// si stampa la cornice
		System.out.println("+ -  - +");
	}
}
```

Esercizio 1.3 - Fcentrata.java

Testo:
Scrivere un programma che visualizzi sullo schermo del terminale la vostra iniziale grande e centrata composta da molti caratteri uguali.

```
public class Fcentrata
{
	public static void main(String[] args)
	{
		System.out.println("              FFFFFFFFFFFFF");
		System.out.println("              FFFFFFFFFFFFF");
		System.out.println("              FFF");
		System.out.println("              FFF");
```

```
            System.out.println("              FFFFFFFFFF");
            System.out.println("              FFFFFFFFFF");
            System.out.println("              FFF");
            System.out.println("              FFF");
            System.out.println("              FFF");
            System.out.println("              FFF");
    }
}
```

CAPITOLO 2 Utilizzare oggetti

REMIND:

(Paragrafo 2.1) Tipi di variabili:

- A ogni valore è assegnato un tipo (es. integer).
- Regole per gli identificatori:
  - possono contenere lettere, cifre, "$", underscore ("_"), ma non possono iniziare con una cifra.
  - No altri simboli (es. "?" o "!").
  - NO SPAZI.
  - Non si possono usare parole riservate.
  - greeting ≠ Greeting.
  - Le variabili iniziano con la lettera minuscola, le Classi iniziano con la lettera maiuscola.

(Paragrafo 2.2) Operatore di assegnazione

- Si usa "=" (es. int n = 7 → assegna a n il valore 7).

(Paragrafo 2.4) Parametri e valori restituiti dai metodi

- Parametri → dati in ingresso
  Es. System.out.println(greeting)
  greeting = parametro ESPLICITO
  System.out = parametro IMPLICITO → oggetto con cui si invoca il metodo.

- replace → esegue operazioni di ricerca/sostituzione (river.replace("issi", "our"))
- Quando un metodo non restituisce un valore → void
- Nome di un metodo SOVRACCARICO → si riferisce a più di un metodo.
  Es. System.out.println(int n), System.out.println(String s)

(Paragrafo2.7) Metodi di accesso e metodi modificatori

- ACCESSO → restituisce informazioni relative all'oggetto senza modificarlo (es. get()).
- MODIFICATORI → modificano lo stato di un oggetto (es. set()).

(Paragrafo 2.10) Riferimenti a oggetti

- RIFERIMENTO A OGGETTO → la variabile contiene la posizione in memoria di un oggetto: si riferisce a quell'oggetto.
- Le variabili numeriche memorizzano i numeri stessi → RIFERIMENTO A VALORE
- - Quando si copia un valore di tipo primitivo, la copia e l'originale sono indipendenti
  - Quando si copia un riferimento a un oggetto l'originale e la copia sono riferimenti al medesimo oggetto.

Esercizio 2.1 - Rettangolo.java

Testo:
Si costruisce un rettangolo partendo da una base, un'altezza e dalle coordinate del piano.

Consigli:
Per chi si affaccia per la prima volta al panorama della programmazione d'informatica, può risultare difficile la risoluzione di tale problema. Si cerchi, quindi, di comprendere ogni passaggio:

- costruttore: ne sono presenti due. Il primo crea un rettangolo con le dimensioni definite, il secondo, invece, permette all'utente di assegnare i valori
- metodi: possono essere suddivisi in più categorie. I metodi contenenti la dicitura "get" (getBase, getAltezza…) restituiscono il valore della dimensione richiesta o eseguono un'operazione specifica (getArea). Quelli contenenti il prefisso "set", invece, permettono di assegnare un nuovo valore alla variabile d'istanza.

```
public class Rettangolo
{
	private int b;
	private int h;
	private int x;
	private int y;
	// si costruisce un rettangolo con i parametri predefiniti
	public Rettangolo()
	{
		b = 1;
		h = 1;
		x = 0;
		y = 0;
	}
	/* si costruisce un rettangolo con i parametri acquisiti dall'esterno
	@param base la base del rettangolo
	@param altezza l'altezza del rettangolo
	@param ascissa l'ascissa del rettangolo
	@param ordinata l'ordinata del rettangolo */
	public Rettangolo(int base, int altezza, int ascissa, int ordinata)
	{
		b = base;
		h = altezza;
		x = ascissa;
		y = ordinata;
	}
	/* si acquisisce la base
	@return la base del rettangolo */
	public int getBase()
	{
		return b;
	}
	/* si acquisisce l'altezza
	@return l'altezza del rettangolo */
	public int getAltezza()
	{
		return h;
	}
	/* si acquisisce l'ascissa
	@return l'ascissa del rettangolo */
	public int getAscissa()
	{
		return x;
	}
	/* si acquisisce l'ordinata
	 @return l'ordinata del rettangolo */
	public int getOrdinata()
	{
		return y;
```

```
	}
	/* si modifica la base
	@param nuovaBase la nuova misura della base*/
	public void setBase(int nuovaBase)
	{
		b = nuovaBase;
	}
	/* si modifica l'altezza
	@param nuovaAltezza la nuova misura dell'altezza*/
	public void setAltezza(int nuovaAltezza)
	{
		h = nuovaAltezza;
	}
	/* si modifica l'ascissa
	@param nuovaAscissa la nuova ascissa*/
	public void setAscissa(int nuovaAscissa)
	{
		x = nuovaAscissa;
	}
	/* si modifica l'ordinata
	@param nuovaOrdinata la nuova ordinata*/
	public void setOrdinata(int nuovaOrdinata)
	{
		y = nuovaOrdinata;
	}
	/* si traslano le coordinate nel piano
	@param trX lo spostamento in ascissa
	@param trY lo spostamento in ordinata*/
	public void traslazione(int trX, int trY)
	{
		x = x + trX;
		y = y + trY;
	}
	/* si calcola il perimetro
	@return il perimetro */
	public int getPerimetro()
	{
		return (b + h)*2;
	}
	/* si calcola l'area
	@return l'area */
	public int getArea()
	{
		return b * h;
	}
}
```

RettangoloTester.java

```
public class RettangoloTester
{
	public static void main(String[] args)
	{
		Rettangolo r = new Rettangolo();
		System.out.println("Perimetro: " + r.getPerimetro());
		System.out.println("Area: " + r.getArea());
		System.out.println("Expected perimetro: 4");
		System.out.println("Expected area: 1");
		Rettangolo rr = new Rettangolo(5, 3, 9, 2);
		System.out.println("Perimetro: " + rr.getPerimetro());
		System.out.println("Area: " + rr.getArea());
		System.out.println("Expected perimetro: 16");
		System.out.println("Expected area: 15");
```

```
	}
}
```

Esercizio 2.2 – Dado.java (Esercizio P2.7 pag. 53)

Testo:
Scrivere un programma che usi la classe Random per simulare il lancio di un dado, visualizzando un numero casuale compreso tra 1 e 6 ogni volta che viene eseguito.

Consigli:
Il seguente programma può essere di difficile comprensione ai neofiti della materia perché viene introdotta la classe Random. Essa, come molte altre classi (Scanner, Rectangle), deve essere importata dalla libreria virtuale di Java, tramite la dicitura "import nomePacchetto.nomeClasse". Si rimanda al ripasso del capitolo 2 l'approfondimento dell'importazione di classi.

Dado.java

```
import java.util.Random[2];

public class Dado
{
	private int numeroFacce;
	private Random gen;

	//costruisce un Dado a 6 facce
	public Dado()
	{
		numeroFacce = 6;
		// si inizializza l'oggetto gen della classe Random
		gen = new Random();
	}

	/*simula il lancio del dado
	@return il risultato del lancio*/
	public int lancia()
	{
		int result = gen.nextInt(numeroFacce) + 1[3];
		return result;
	}
}
```

DadoTester.java

```
public class DadoTester
{
	public static void main(String[] args)
	{
		Dado d = new Dado();
		int risultato = d.lancia();
		System.out.println("Lancio: " + risultato);
	}
}
```

CAPITOLO 3 Realizzare classi

[2] Un oggetto *Random* genera numeri in maniera casuale

[3] "gen.nextInt(numeroFacce)+1" perché il metodo nextInt(n) della classe Random restituisce numeri interi casuali da 0 a n-1. Aggiungendo 1 ottengo i numeri da 1 a n (in questo caso da 1 a 6 come un dado!).

REMIND:

(Paragrafo 3.4) Campi di Esemplare

- Un oggetto memorizza i propri dati dentro a ***variabili d'istanza*** (o ***campi di esemplare***).
- VARIABILE D'ISTANZA → zona di memorizzazione presente in ogni oggetto della classe.

(Paragrafo 3.7) Categorie di variabili

- Categorie:
  - Campi di esemplare (o d'istanza)
  - Variabili locali
  - Variabili parametro
- DIFFERENZA → tempo di vita

Esercizio 3.1 – BankAccount.java (Esercizio P3.1 – P3.2 pag. 86)

Testo:
Es. 3.1: scrivere un programma che costruisca un conto bancario, versi in esso $1000, prelevi da esso $500, prelevi altri $400 e infine visualizzi il saldo rimanente

Consigli:
Il programma non presenta particolari difficoltà. Si invita a rivedere i consigli dell'esercizio sul rettangolo.

```
public class BankAccount
{
	private double balance;
	// si costruisce un conto bancario con saldo uguale a zero
	public BankAccount()
	{
		balance = 0;
	}
	/* si costruisce un conto bancario con un saldo assegnato
	@param initialBalance il saldo iniziale*/
	public BankAccount(double initialBalance)
	{
		balance = initialBalance;
	}
	/* si versa denaro nel conto bancario
	@param amount l'importo da versare*/
	public void deposit(double amount)
	{
		balance = balance + amount;
	}
	/* si preleva denaro dal conto bancario
	@param amount l'importo da prelevare*/
	public void withdraw(double amount)
	{
		balance = balance - amount;
	}
	/* s'ispeziona il valore del saldo attuale del conto bancario
	@return il saldo attuale*/
	public double getBalance()
	{
		return balance;
	}
	/* si calcola un interesse sul conto
	@param rate il tasso d'interesse*/
	public void addInterest(double rate)
	{
```

```
            balance = balance + ((balance * rate) / 100);
      }
}
```

BankAccountTester.java

```
public class BankAccountTester
{
      public static void main(String[] args)
      {
            BankAccount harrysChecking = new BankAccount();
            harrysChecking.deposit(1000);
            harrysChecking.withdraw(500);
            harrysChecking.withdraw(400);
            System.out.println(harrysChecking.getBalance());
            BankAccount momsSaving = new BankAccount(1000);
            momsSaving.addInterest(10); //Interessi al 10%
            System.out.println("Dollars: " + momsSaving.getBalance());
      }
}
```

Esercizio 3.2 - BankWithPassword

Testo:
Lo scopo del programma è riadattare il programma presente nel libro di testo, introducendo il metodo boolean controlloPassword, che restituisce true se la password inserita corrisponde a quella del conto corrente cercato, e il metodo boolean controlloPrelievo, che restituisce true se la somma inserita è disponibile nel conto corrente.

Consigli:
L'esercizio non presenta particolari difficoltà.

BankWithPassword.java

```
public class BankWithPassword
{
      private double balance;
      private int password;

      /*si costruisce un conto con saldo uguale a zero e password di accesso
      numerica
      @param pass password numerica di accesso al conto*/
      public BankWithPassword(int pass)
      {
            balance = 0;
            password = pass;
      }

      /*si costruisce un conto con un saldo assegnato e password di accesso
      @param initialBalance il saldo iniziale
      @param pass password di accesso al conto*/
      public BankWithPassword(double initialBalance, int pass)
      {
            balance = initialBalance;
            password = pass;
      }

      /* si versa denaro nel conto bancario
      @param amount l'importo da versare*/
      public void deposit(double amount)
      {
            balance = balance + amount;
```

```
	}

	/* si preleva denaro dal conto bancario
	@param amount l'importo da prelevare*/
	public void withdraw(double amount)
	{
		balance = balance - amount;
	}

	/* si applica un interesse sul conto
	@param rate il tasso d'interesse*/
	public void addInterest(double rate)
	{
		balance = balance + ((balance * rate) / 100);
	}

	/* ispeziona il valore del saldo attuale del conto bancario
	@return il saldo attuale*/
	public double getBalance()
	{
		return balance;
	}

	/* restituisce la password del conto
	@return la password del conto*/
	public int getPassword()
	{
		return password;
	}

	/* verifica la validità della password immessa
	@param pass la password da verificare
	@return true se corretta, false se errata*/
	public boolean controlloPassword(int pass)
	{
		if (pass == password)
			return true;
		return false;
	}

	/* verifica che la somma da prelevare sia disponibile nel conto
	@param amount la somma da prelevare
	@return true se disponibile, false se non disponibile*/
	public boolean controlloPrelievo(double amount)
	{
		if (amount <= balance)
			return true;
		return false;
	}
}
```

BankWithPasswordTester.java

```
import java.util.Scanner;

public class BankWithPasswordTester
{
	public static void main(String[] args)
	{
		//si inizializza la variabile che mi servirà per inserire I dati
		//dalla tastiera
		Scanner in=new Scanner(System.in);
```

```
            //si costruisce un primo conto con saldo iniziale pari a zero e
            //password definita dall'utente
            System.out.println("Inserire un numero che sarà password del conto
                  1");
            BankWithPassword conto1 = new BankWithPassword(in.nextInt());
            //stampa di verifica dei dati inseriti
            System.out.println("Il saldo del conto 1 è: " + conto1.getBalance()
                  + "\nla password del conto 1 è: " + conto1.getPassword());
            //si costruisce un secondo conto con saldo iniziale e
            //password definiti dall'utente
            System.out.println("Inserire un numero che sarà password del conto
                  2");
            int temp = in.nextInt();
            System.out.println("Inserire il saldo iniziale del conto 2");
            BankWithPassword conto2 = new BankWithPassword(in.nextDouble(),
                  temp);
            //stampa di verifica dei dati inseriti
            System.out.println("Il saldo del conto 2 è: " + conto2.getBalance()
                  + "\nla password del conto 2 è: " + conto2.getPassword());
            //si preleva una somma dal conto 2
            System.out.println("Prelievo dal conto 2:\nInserire la password del
                  conto 2");
            if(conto2.controlloPassword(in.nextInt()))//controlla la password
                  {
                        System.out.println("Inserire la somma da prelevare:");
                        double amount = in.nextDouble();
                        //controlla che la somma sia disponibile e in caso
                        //affermativo la preleva
                        if (conto2.controlloPrelievo(amount))
                              conto2.withdraw(amount);
                        else
                              System.out.println("Somma non disponibile nel
                                    conto");
                  }
            else
                  System.out.println("Password errata. Impossibile effettuare il
                        prelievo.");
            //si versa una somma sul conto 1
            System.out.println("Versamento sul conto 1:\nInserire la password
                  del conto 1");
            if(conto1.controlloPassword(in.nextInt()))//controlla la password
                  {
                        System.out.println("Inserire la somma da versare:");
                        conto1.deposit(in.nextDouble());
                  }
            else
                  System.out.println("Password errata. Impossibile effettuare il
                        versamento.");
            //stampa un resoconto dei saldi
            System.out.println("Nuovi saldi:\nconto 1: " + conto1.getBalance() +
                  "\nconto 2: " + conto2.getBalance());
            //applica l'interesse sui due conti e stampa i saldi aggiornati
            conto1.addInterest(3);
            conto2.addInterest(3);
            System.out.println("Nuovi saldi con interesse applicato del 3%:");
            System.out.println("conto 1: " + conto1.getBalance() + "\nconto 2: "
                  + conto2.getBalance());
      }
}
```

Esercizio 3.3 - Car.java (Esercizio P3.6 pag. 87)

Testo:

Progettare e realizzare una classe Car(automobile) con le proprietà seguenti. Un'automobile ha una determinata resa del carburante (misurata in miglia/galloni o in litri/chilometri: scegliete il sistema che preferite) e una certa quantità di carburante nel serbatoio. La resa è specificata dal costruttore e il livello iniziale del carburante è a zero. Fornire questi metodi: un metodo drive per simulare il percorso di un'automobile per una certa distanza, riducendo il livello di carburante nel serbatoio; un metodo getGas, per ispezionare il livello corrente del carburante; un metodo addGas per far rifornimento.

Consigli:
S'invita a porre particolare attenzione al metodo drive(double km), che calcola il livello di carburante dopo un certo percorso.

```
public class Car
{
	// quantità di carburante nel serbatoio
	private double gas;
	// prestazioni della vettura
	private double kmL;
	// si costruisce un'automobile con carburante uguale a zero
	public Car(double resa)
	{
		kmL = resa;
		gas = 0;
	}
	/* Calcola il livello di carburante rimasto dopo un certo percorso
	@param km i chilometri percorsi*/
	public void drive(double km)
	{
		gas = gas - (km / kmL);
	}
	/* ispeziona il livello di carburante rimasto
	@return il carburante rimasta*/
	public double getGas()
	{
		return gas;
	}
	/* aggiunge carburante nel serbatoio
	@param rifornimento il carburante da aggiungere*/
	public void addGas(double rifornimento)
	{
		gas = gas + rifornimento;
	}
}
```

CarTester.java

```
public class CarTester
{
	public static void main(String[] args)
	{
		Car ibiza = new Car(20);
		ibiza.addGas(20);
		ibiza.drive(100);
		System.out.println(ibiza.getGas());
	}
}
```

Esercizio 3.4 - Employee.java (Esercizio P3.4 - Esercizio P3.5 pag. 87)

Testo:
Esercizio 3.4: Progettare e realizzare una classe Employee(dipendente). Ciascun dipendente ha un nome (di tipo stringa) e uno stipendio (di tipo double). Scrivere un costruttore senza parametri, un costruttore con due parametri (nome

e stipendio), e i metodi per conoscere nome e stipendio. Scrivere un breve programma per collaudare la classe.
Esercizio 3.5: Sviluppare la classe dell'esercizio precedente, aggiungendo un metodo raiseSalary(double byPercent), che incrementi lo stipendio del dipendente secondo una certa percentuale

Consigli:
Il libro vorrebbe che si facessero due costruttori: uno con i parametri e uno senza parametri. È preferibile fare un costruttore senza parametri e in seguito impostare i valori dei campi con dei metodi (setNome(), setStipendio()).
Il seguente esercizio è simile a quelli precedenti. Lo scopo risulta semplicemente acquisire una metodologia di lavoro.

```
public class Employee
{
	private String nome;
	private double stipendio;
	public Employee()
	{
		nome = null;
		stipendio = 0;
	}
	/* si restituisce il nome del dipendente
	@return il nome del del dipendente*/
	public String getNome()
	{
		return nome;
	}
	/* si restituisce lo stipendio del dipendente
	@return lo stipendio*/
	public double getStipendio()
	{
		return stipendio;
	}
	/* si modifica il valore di stipendio
	@param s il nuovo valore dello stipendio*/
	public void setStipendio(double s)
	{
		stipendio = s;
	}
	/* si modifica il valore di nome
	@param s il nuovo nome*/
	public void setNome(String n)
	{
		nome = n;
	}
	/* s'incrementa lo stipendio del dipendente
	@param byPercent il tasso di aumento dello stipendio*/
	public void raiseSalary (double byPercent)
	{
		stipendio = stipendio + ((stipendio * byPercent) / 100);
	}
}

public class EmployeeTester
{
	public static void main(String[] args)
	{
		// si crea un oggetto di classe Employee
		Employee d = new Employee();
		// si crea un nuovo dipendente
		d.setNome("Rossi");
		// s'inserisce il valore dello stipendio
```

```
            d.setStipendio(1000);
            // s'incrementa del 10% lo stipendio
            d.raiseSalary(10);
            // si restituiscono i valori richiesti
            System.out.println(d.getNome() + " = " + d.getStipendio());
      }
}
```

CAPITOLO 4 Tipi di dati fondamentali

REMIND:

(Paragrafo 4.1) Tipi di numeri

- ***BigInteger*** → classe con rispettivi metodi: non è un tipo primitivo, si usa con numeri molto grandi.
- ***BigDecimal*** → come BigInteger. Consente di effettuare calcoli in virgola mobile senza errori di approssimazione.
- Con valori in virgola mobile bisogna tenere presente il problema degli errori di arrotondamento
- (int) → cast, converte in numero intero ignorandone la parte frazionaria (es. 13.75 → 13)
- Usate long per arrotondare un numero in virgola mobile: se il valore assoluto è elevato int potrebbe non essere sufficiente:
  long n = Math.round(d); (13.75 → 14)

(Paragrafo 4.2) Costanti

- final → indica le costanti in Java
- Si usano le lettere maiuscole (es. MAX_VALUE)
- Se si usano in più metodi si devono dichiarare insieme alle variabili d'istanza
- Non c'è pericolo nel dichiararle come "public" perché non possono essere modificate

(Paragrafo 4.4) Aritmetica e funzioni matematiche

- "/" → divisione (es. 15 / 4)
- Se si dividono numeri interi, il risultato sarà un intero (es. 15 / 4 = 3)
- "%" (percentuale) → resto (es. 15 % 4 = 3)

(Paragrafo 4.5) Invocare metodi statici

- Non agiscono su alcun oggetto
- Sono sempre definiti all'interno di classi → specificare la classe a cui appartiene il metodo (es. Math.sqrt(x);)

(Paragrafo 4.6) Stringhe

- Studia in questo paragrafo i vari metodi per la classe String
- Con l'operatore "+" si possono concatenare stringhe
- Integer.parseInt(s) → converte la stringa s in un numero intero (esiste anche parseDouble())
- Integer.toString(n) → converte il numero n in una stringa

(Paragrafo 4.7) Leggere dati in ingresso

- Scanner → consente la lettura dei dati inseriti in ingresso dalla tastiera
- Studia in questo paragrafo i vari metodi per la classe Scanner
- ATTENZIONE: PRECISAZIONE SULLA LETTURA DATI IN INGRESSO
  nextInt(), next(), nextDouble() e compagnia leggono tutto quello che devono leggere e si fermano al primo carattere separatore incontrato (bianco, tab, a capo) lasciando tale carattere nel canale di ingresso (cioe` non viene prelevato). Per contro, nextLine() prende tutta la riga

di caratteri sino al prossimo carattere a capo compreso.
Allora se digitiamo ad es. 231 seguito da a capo, nextInt() preleva 231 e lascia a capo nel canale. Quanto arriva nextLine() legge una riga vuota, rimuove a capo e si ferma subito. Quindi non bisognerebbe mescolare i due tipi di istruzioni di input, a meno che non si sappia bene quello che si sta facendo.

Esercizio 4.1 – PowerGenerator.java (Esercizio P4.3 pag. 130)

Testo:

Scrivere un programma che stampi i seguenti valori:

```
1
10
100
1000
10000
100000
1000000
10000000
100000000
1000000000
10000000000
100000000000
```

Consigli:

Si rammenta, come già enunciato negli esercizi del capitolo 2, la necessità di importare una libreria Java per poter eseguire talune operazioni: in questo caso, sarà la classe Math. Nel caso specifico, essa sarà utile per creare l'elevamento a potenza. Si ricorda, inoltre, di incrementare la variabile d'istanza "exp".

```
// s'importa la classe Math
import java.lang.Math;
public class PowerGenerator
{
	private double base;
	private int exp;
	public PowerGenerator()
	{
		base = 0.0;
		exp = 0;
	}
	/* s'imposta il valore della base
	@param b il valore da impostare*/
	public void setBase(double b)
	{
		base = b;
	}
	/* si calcola la potenza del numero
```

```
	@return il numero elevato a exp*/
	public double nextPow()
	{
		double result = Math.pow(base, exp);
		/* l'esponente viene incrementato di uno ogni volta che viene
		richiamato il seguente metodo*/
		exp = exp + 1;
		return result;
	}
}
```

PowerGeneratorTester.java

```
public class PowerGeneratorTester
{
	public static void main(String[] args)
	{
		PowerGenerator p = new PowerGenerator();
		p.setBase(10);
		System.out.println(p.nextPow());
		System.out.println(p.nextPow());
		System.out.println(p.nextPow());
	}
}
```

Esercizio 4.2 - TimeConverter.java

Testo:
Scrivere un programma che, dato un certo tempo in giorni, ore, minuti e secondi, restituisca il numero totale di secondi.

Consigli:
Si prega di osservare che un giorno ha 86400 secondi, un'ora ha 3600 secondi e un minuto 60 secondi.

TimeConverter.java

```
public class TimeConverter
{
	private int s, m, h, d;

	/* si costruisce un covertitore da giorni, ore, minuti, secondi, in
	secondi totali
	@param dd giorni
	@param hours ore
	@param min minuti
	@param sec secondi*/
	public TimeConverter(int dd, int hours, int min, int sec)
	{
		d = dd;
		h = hours;
		m = min;
		s = sec;
	}

	/* si restituisce i secondi totali a partire da gg, h, min, sec inseriti
	@return SecTot secondi totali*/
	public int getSecTot()
	{
		int sTot = d*86400 + h*3600 + m*60 + s;
		return sTot;
	}
}
```

TimeConverterTester.java

```
import java.util.Scanner;

public class TimeConverterTester
{
	public static void main(String[] args)
	{
		Scanner in = new Scanner(System.in);
		System.out.print("Inserisci il numero dei giorni ");
		int dd = in.nextInt();
		System.out.print("\nInserisci il numero delle ore ");
		int hours = in.nextInt();
		System.out.print("\nInserisci il numero dei minuti ");
		int min = in.nextInt();
		System.out.print("\nInserisci il numero dei secondi ");
		int sec = in.nextInt();
		TimeConverter conv = new TimeConverter(dd, hours, min, sec);
		System.out.println("\nSecondi totali : " +  conv.getSecTot());
	}
}
```

Esercizio 4.3 - SecondConverter.java

Testo:
Scrivere un programma che, dato un certo tempo in secondi restituisca il numero di giorni, ore, minuti e secondi.

Consigli:
Si preferisce implementare quattro metodi separati per ottenere il numero di giorni, ore, minuti o secondi. Le uniche difficoltà riscontrabili nell'esercizio riguardano la serie di calcoli da dover svolgere per rendere efficace la conversione. Si prega di osservare che un giorno ha 86400 secondi, un'ora ha 3600 secondi e un minuto 60 secondi.

SecondConverter.java

```
public class SecondConverter
{
	private int sTot;
	private final int SECONDI_AL_MINUTO = 60;
	private final int SECONDI_ALL_ORA = 3600;
	private final int SECONDI_AL_GIORNO = 86400;

	/* si costruisce un convertitore da secondi a gg, h, min, sec
	@param secondi i secondi totali*/
	public SecondConverter(int secondi)
	{
		sTot = secondi;
	}

	/* si restituisce il numero dei giorni
	@return dLocal giorni*/
	public int getDays()
	{
		int dLocal = sTot / SECONDI_AL_GIORNO;
		return dLocal;
	}

	/* si restituisce il numero delle ore
	@return hLocal ore*/
	public int getHours()
	{
```

```
            int sLocal = sTot % SECONDI_AL_GIORNO;
            int hLocal = sLocal / SECONDI_ALL_ORA;
            return hLocal;
      }

      /* si restituisce il numero dei minuti
      @return mLocal minuti*/
      public int getMin()
      {
            int sLocal = (sTot % SECONDI_AL_GIORNO)% SECONDI_ALL_ORA;
            int mLocal = sLocal / SECONDI_AL_MINUTO;
            return mLocal;
      }
      /* si restituisce il numero dei secondi
      @return sLocal secondi*/
      public int getSec()
      {
            int sLocal = ((sTot % SECONDI_AL_GIORNO)% SECONDI_ALL_ORA)%
                  SECONDI_AL_MINUTO;
            return sLocal;
      }

}
```

SecondConverterTester.java

```
import java.util.Scanner;

public class SecondConverterTester
{
      public static void main(String[] args)
      {
            Scanner in = new Scanner(System.in);
            System.out.print("Iserisci il numero dei secondi totali ");
            int secTot = in.nextInt();
            SecondConverter conv = new SecondConverter(secTot);
            System.out.println("Giorni: " + conv.getDays());
            System.out.println("Ore: " + conv.getHours());
            System.out.println("Minuti: " + conv.getMin());
            System.out.println("Secondi: " + conv.getSec());
      }
}
```

CAPITOLO 6 Decisioni

REMIND:

(Paragrafo 6.4) Utilizzare espressioni booleani

- && → and
- || → or
- ! → not

Esercizio 6.1 - InputChecker.java (Esercizio P6.7 pag. 208)

Testo:

Scrivere un programma che stampi la domanda "Vuoi continuare?" e che attenda dati dall'utente. Se l'utente immette "S", "Sì", "Ok", "Certo" o "Perché no?", stampare "OK". Se l'utente scrive "N" o "No", stampare "Fine". Negli altri casi, stampare "Dato non corretto". Non considerare differenze tra maiuscolo e minuscolo, quindi anche "s" e "sì" sono validi.

Consigli:
Per la prima volta, l'esercizio introduce l'espressione decisionale "if", cioè definisce il comportamento di un programma, se e solo se sono rispettate determinate condizioni. Nel metodo String risposta(), è consigliabile concatenare tutte le condizioni richieste dall'esercizio.

```
public class InputChecker
{
	private String answer;
	/* si costruisce un confrontatore di risposte
	@param la risposta da confrontare*/
	public InputChecker(String aAnswer)
	{
		answer = aAnswer;
	}
	/* si restituisce il risultato della risposta
	@return il risultato della risposta*/
	public String risposta()
	{
		if (answer.equalsIgnoreCase("S") ||
		    answer.equalsIgnoreCase("Sì") ||
		    answer.equalsIgnoreCase("Certo") ||
		    answer.equalsIgnoreCase("OK") ||
		    answer.equalsIgnoreCase("Perché no?"))
		    // fine condizione, || = or
		return "OK";
		else
		if(answer.equalsIgnoreCase("N")|| answer.equalsIgnoreCase("No"))
			return "Fine";
		else
		return "Dato non corretto!";
	}
}
```

InputCheckerTester.java

```
import java.util.Scanner;
public class InputCheckerTester
{
	public static void main(String[] args)
	{
		Scanner in = new Scanner(System.in);
		System.out.println("Vuoi continuare ? (s/n)");
		String risp = in.nextLine();
		InputChecker input = new InputChecker(risp);
		System.out.println(input.risposta());
	}
}
```

Esercizio 6.2 - Sort.java (Esercizio P6.4 pag. 208)

Testo:
Scrivere un programma che riceva tre numeri in virgola mobile come dati in ingresso, per poi stamparli in ordine crescente.

Consigli:
L'esercizio non presenta difficoltà importanti di nota. Si consiglia, però, di fare un piccolo schema di tutti i casi possibili richiesti dall'esercizio, in modo da evitare errori logici.

```
public class Sort
{
	private double a, b, c;
	public Sort()
```

```
{
	a = 0.0;
	b = 0.0;
	c = 0.0;
}
/* s'ordinano i numeri
@param n,m,d i numeri inseriti*/
public void sortNum(double n, double m, double d)
{
	double temp = 0.0;
	a = n;
	b = m;
	c = d;
	if (a < b)
		if (c < b)
			if (c < a)
			{
				temp = a;
				a = c;
				c = b;
				b = temp;
			}
			else
			{
				temp = b;
				b = c;
				c = temp;
			}
	else
	if (b < a)
		if (c < a)
			if (b < c)
			{
				temp = a;
				a = b;
				b = c;
				c = temp;
			}
			else
			{
				temp = a;
				a = c;
				c = temp;
			}
		else
		{
			temp = a;
			a = b;
			b = temp;
		}
}
public double getA()
{
	return a;
}
public double getB()
{
	return b;
}
public double getC()
{
	return c;
}
```

```
}
```

SortTester.java

```
import java.util.Scanner;
public class SortTester
{
	public static void main(String[] args)
	{
		Scanner in = new Scanner(System.in);
		System.out.println("Please enter three numbers ");
		double a = in.nextDouble();
		double b = in.nextDouble();
		double c = in.nextDouble();
		Sort s = new Sort();
		s.sortNum(a, b, c);
		System.out.println("The inputs in sorted order are ");
		System.out.println(s.getA());
		System.out.println(s.getB());
		System.out.println(s.getC());
	}
}
```

Esercizio 6.3 - Anno Bisestile (Esercizio P6.12 pag209)

Testo:
Un'anno con 366 giorni è detto anno bisestile. Un anno è bisestile se è divisibile per quattro (per esempio, il 1980), ma, dopo l'adozione del calendario gregoriano avvenuta il 15 ottobre 1582, un anno non è bisestile se è divisibile per 100 (per esempio, il 1900), mentre è bisestile se è divisibile per 400 (per esempio, il 2000). Scrivete un programma che chieda all'utente di inserire un anno e che calcoli se l'anno è bisestile. Scrivere una classe Year con un metodo predicativo boolean isLeapYear().

Consigli:
Come nell'esercizio precedente, si invita a controllare attentamente le condizioni per cui un anno è bisestile.

Year.java

```
// Classe  che controlla se un anno è bisestile o meno.
public class Year
{
	private int anno;

	public Year(int unAnno)
	{
		anno = unAnno;
	}

	/*Un anno è bisestile se è divisibile per 400 oppure per 4 ma non per 100.
	Questo metodo controlla tale proprietà.
	@return true se è bisestile, false se non lo è*/
	public boolean isLeapYear()
	{
		int resto1 = anno % 400;
		int resto2 = anno % 4;
		int resto3 = anno % 100;
		if (resto1 == 0 || (resto2 == 0 && resto3 != 0))
			return true;
		else
			return false;
	}

	/*restituisce il valore della variabile anno
```

```
	@return l'anno*/
	public int getYear()
	{
		return anno;
	}
}
```

YearTester.java

```
import java.util.Scanner;
public class YearTester
{
public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);
		System.out.println("Inserisci un anno ");
		Year anno = new Year(input.nextInt());
		if(anno.isLeapYear())
			System.out.println("L'anno " + anno.getYear() + " è
				bisestile.");
		else
			System.out.println("L'anno " + anno.getYear() + " non è
				bisestile.");
	}
}
```

Esercizio 6.4 - Voti (Esercizio 6.8 pag. 208)

Testo:

Scrivete un programma per convertire la lettera di un voto scolastico nel numero corrispondente. Le lettere sono A, B, C, D e F, eventualmente seguite dai segni + o -. I loro valori numerici sono 4, 3, 2, 1 e 0. F+ e F- non esistono. Un segno + o – incrementa o decrementa il valore numerico di 0.3. Tuttavia, A+ è uguale a 4.0. Usate una classe Grade con un metodo getNumericGrade.

Consigli:

Si consiglia di intendere il voto come un double, anziché una stringa. Ciò semplifica l'esecuzione del programma: in tal modo, il voto può essere modificato tramite semplici operazioni aritmetiche.

Grade.java

```
// Classe che converte i voti in lettere in voti in numeri
public class Grade
{
	private String votoL;
	private double votoN;
	private String segno;
	// per il segno + si aggiunge,per il segno - si sottrae
	final static double SEGNO = 0.3;
	public Grade()
	{}
	public double getNumericGrade(String input)
	{
		// si separa il voto(lettera) dal segno
		votoL = input.substring(0,1);
		segno = input.substring(1);
		if (votoL.equalsIgnoreCase("A"))
		{
			votoN = 4;
			if (segno.equals("-"))
				votoN = votoN - SEGNO;
		}
		else if (votoL.equalsIgnoreCase("B"))
		votoN = 3;
```

```
            else if (votoL.equalsIgnoreCase("C"))
            votoN = 2;
            else if (votoL.equalsIgnoreCase("D"))
            votoN = 1;
            else if (votoL.equalsIgnoreCase("F"))
            votoN = 0;
            if (votoL.equalsIgnoreCase("B")|| votoL.equalsIgnoreCase("C") ||
            votoL.equalsIgnoreCase("D"))
            {
                  if (segno.equals("+"))
                        votoN = votoN + SEGNO;
                  else if (segno.equals("-"))
                        votoN = votoN - SEGNO;
            }
            return votoN;
   }
}
```

GradeTester.java

```
import java.util.Scanner;
public class GradeTester
{
      public static void main(String[] args)
      {
            Scanner in = new Scanner(System.in);
            Grade voto1 = new Grade();
            boolean c = true;
            while (c)
            {
                  System.out.println("Scrivi il voto da A a F da convertire.
                        Premi Q per uscire");
                  String input = in.next();
                  if (input.equalsIgnoreCase("Q"))
                        c = false;
                  else
                        System.out.println(voto1.getNumericGrade(input));
            }
   }
}
```

Esercizio 6.5 - NumberConverter

Testo:

Si scriva un metodo arabicToString che converta un numero nell'intervallo [1, 999.999] in una stringa con il nome del numero in italiano. Ad es 247.247 viene convertito nella stringa duecentoquarantasettemiladuecentoquarantasette.

Consigli:

Si scomponga il problema in casi più semplici considerando la simmetria e la ripetitività di alcuni eventi. Si devono creare nuovi oggetti della classe NumberConverter da cui invocare i metodi per convertire le centinaia, le decine e le unità.

NumberConverter.java

```
public class NumberConverter
{
      private int number = 0;

      public NumberConverter(int aNumber)
      {
```

```
	number = aNumber;
}

/*converte un numero compreso tra 0 e 999.999 dalla forma in cifre a
quella in caratteri
@return il numero convertito*/
public String arabicToString()
{
	if (number == 0)
		return "zero";
	String result = new String("");
	/*divido il numero in due composti al più di 3 cifre e per ognuno
	creo un nuovo oggetto*/
	int sx = number / 1000;
	int dx = number % 1000;
	NumberConverter s = new NumberConverter(sx);
	NumberConverter d = new NumberConverter(dx);
	if (sx == 1)
		result = "mille";
	else if (sx > 1)
		result = s.centinaiaToString() + "mila";
	result = result + d.centinaiaToString();
	return result;
}

/*converte un numero compreso tra 0 e 999 dalla forma in cifre a quella in
caratteri
@return il numero convertito*/
public String centinaiaToString()
{
	String result = new String("");
	//separo le centinaia dal resto del numero e creo due nuovi oggetti
	int sx = number / 100;
	int dx = number % 100;
	NumberConverter s = new NumberConverter(sx);
	NumberConverter d = new NumberConverter(dx);
	if (sx > 1)
		result = s.unitàToString() + "cento";
	else if (sx == 1)
		result = "cento";
	result = result + d.decineToString();
	return result;
}

/*converte un numero compreso tra 0 e 99 dalla forma in cifre a quella in
caratteri
@return il numero convertito*/
public String decineToString()
{
	String result = new String("");
	/*separo le decine dalle unità e creo un nuovo oggetto solamente per
	le unità: i multipli di dieci in italiano devono essere definiti uno
	per uno perchè non seguono uno schema riproducibile*/
	int sx = number / 10;
	int dx = number % 10;
	NumberConverter d = new NumberConverter(dx);
	if(sx == 9)	result = "novanta" + d.unitàToString();
	else if(sx == 8)  result = "ottanta" + d.unitàToString();
	else if(sx == 7)  result = "settanta" + d.unitàToString();
	else if(sx == 6)  result = "sessanta" + d.unitàToString();
	else if(sx == 5)  result = "cinquanta" + d.unitàToString();
	else if(sx == 4)  result = "quaranta" + d.unitàToString();
	else if(sx == 3)  result = "trenta" + d.unitàToString();
```

```
            else if(sx == 2)  result = "venti" + d.unitàToString();
            else if(sx == 0)  result = d.unitàToString();
            else
            {
                  /*anche i numeri compresi tra 10 e 19 vanno definiti uno
                  per uno*/
                  if (dx == 0)      result = "dieci";
                  else if (dx == 1) result = "undici";
                  else if (dx == 2) result = "dodici";
                  else if (dx == 3) result = "tredici";
                  else if (dx == 4) result = "quattordici";
                  else if (dx == 5) result = "quindici";
                  else if (dx == 6) result = "sedici";
                  else if (dx == 7) result = "diciassette";
                  else if (dx == 8) result = "diciotto";
                  else if (dx == 9) result = "diciannove";
            }
            return result;
      }

      /*converte un numero compreso tra 0 e 9 dalla forma in cifre a quella in
      caratteri
      @return il numero convertito*/
      public String unitàToString()
      {
            String result = new String("");
            if (number == 0)        result = "";
            else if (number == 1)   result = "uno";
            else if (number == 2)   result = "due";
            else if (number == 3)   result = "tre";
            else if (number == 4)   result = "quattro";
            else if (number == 5)   result = "cinque";
            else if (number == 6)   result = "sei";
            else if (number == 7)   result = "sette";
            else if (number == 8)   result = "otto";
            else if (number == 9)   result = "nove";
            return result;
      }
}
```

NumberConverterTester.java

```
import java.util.Scanner;

public class NumberConverterTester
{
      public static void main(String[] args)
      {
            Scanner in = new Scanner(System.in);
            System.out.println("Inserire un numero compreso tra 0 e 999999
                  da convertire in caratteri:");
            NumberConverter numero = new NumberConverter(in.nextInt());
            System.out.println("Il numero è: " + numero.arabicToString());
      }
}
```

Esercizio 6.6 - NumberConverter - due

Testo:

Come nell'esercizio precedente si deve scrivere un programma che converta un numero nell'intervallo [1, 999.999] in una stringa con il nome del numero in italiano. In questo caso però si preferisce non creare l'oggetto della classe NumberConverter, ma tradurre il numero fornendolo come parametro esplicito.

Consigli:
La peculiarità di questo programma è che tutti i metodi sono static.

NumberConverter.java

```
public class NumberConverter
{
	/*Si converte il numero dividendo il problema in casi più semplici
	 *invocando metodi ausiliari.
	 *@param il numero tra 0 e 999.999 da convertire in lettere
	 *@return il numero convertito in lettere*/
	public static String arabicToString(int n)
	{
		if (n == 0) return "zero";
		String result = "";
		int sin = n / 1000;
		int dex = n % 1000;
		if (sin == 0) result = centinaiaToString(dex);
		else if (sin == 1) result = "mille" + centinaiaToString(dex);
		else result = centinaiaToString(sin) + "mila" +
			centinaiaToString(dex);
		return result;
	}

	/*Si converte il numero dividendo il problema in casi più semplici ed
	 *invocando metodi ausiliari.
	 *@param il numero tra 1 e 999 da convertire in lettere
	 *@return il numero convertito in lettere*/
	public static String centinaiaToString (int n)
	{
		String result = "";
		int sin = n / 100;
		int dex = n % 100;
		if (sin == 0) result = decineToString(dex);
		else if (sin == 1) result = "cento" + decineToString(dex);
		else result = unitàToString(sin) + "cento" + decineToString(dex);
		return result;
	}

	/*Si converte il numero dividendo il problema in casi più semplici ed
	 *invocando metodi ausiliari.
	 *@param il numero tra 1 e 99 da convertire in lettere
	 *@return il numero convertito in lettere*/
	public static String decineToString (int n)
	{
		String result = "";
		int sin = n / 10;
		int dex = n % 10;
		if (sin == 0) result = unitàToString(dex);
		else if ( sin == 1)
		{
			if(dex == 0) result = "dieci";
			else if(dex == 1) result = "undici";
			else if(dex == 2) result = "dodici";
			else if(dex == 3) result = "tredici";
			else if(dex == 4) result = "quattordici";
			else if(dex == 5) result = "quindici";
			else if(dex == 6) result = "sedici";
			else if(dex == 7) result = "diciassette";
			else if(dex == 8) result = "diciotto";
			else if(dex == 9) result = "diciannove";
		}
```

```
            else if(sin == 2) result = "venti" + unitàToString(dex);
            else if(sin == 3) result = "trenta" + unitàToString(dex);
            else if(sin == 4) result = "quaranta" + unitàToString(dex);
            else if(sin == 5) result = "cinquanta" + unitàToString(dex);
            else if(sin == 6) result = "sessanta" + unitàToString(dex);
            else if(sin == 7) result = "settanta" + unitàToString(dex);
            else if(sin == 8) result = "ottanta" + unitàToString(dex);
            else if(sin == 9) result = "novanta" + unitàToString(dex);
            return result;
      }

      /*Si converte il numero dividendo il problema in casi più semplici ed
       *invocando metodi ausiliari.
       *@param il numero tra 1 e 9 da convertire in lettere
       *@return il numero convertito in lettere*/
      public static String unitàToString(int n)
      {
            String result = "";
            if(n == 1) result = "uno";
            else if(n == 2) result = "due";
            else if(n == 3) result = "tre";
            else if(n == 4) result = "quattro";
            else if(n == 5) result = "cinque";
            else if(n == 6) result = "sei";
            else if(n == 7) result = "sette";
            else if(n == 8) result = "otto";
            else if(n == 9) result = "nove";
            return result;
      }
}
```

NumberConverterTester.java

```
import java.util.Scanner;

public class NumberConverterTester
{
      public static void main(String[] args)
      {
            Scanner in = new Scanner(System.in);
            System.out.println("dammi un numero intero tra 0 e 999.999 da
                  scrivere in lettere");
            int numero = in.nextInt();
            if (numero < 0 || numero > 999999) System.out.println("Voglio un
                  numero tra 0 e 999.999");
            else System.out.println("Il numero è: " +
                  NumberConverter.arabicToString(numero));
    }
}
```

Esercizio 6.7 - ArabicToRoman.java

Testo:

Realizzare una classe con un metodo che converta un numero intero compreso tra 1 e 100 in un numero romano (es: 57 diventa LVII).

Consigli:

L'esercizio non presenta particolari difficoltà. Si trattano separatamente le centinaia, le decine e le unità, e si associano al relativo simbolo romano.

ArabicToRoman.java

```
public class ArabicToRoman
{
	private int arabic;

	/*costruisce un oggetto che contiene il numero da convertire
	@param aNumber il numero da convertire*/
	public ArabicToRoman(int aNumber)
	{
		arabic = aNumber;
	}

	/*restituisce il numero convertito in simboli romani
	@return il numero convertito in simboli romani*/
	public String converti()
	{
		String numeroRomano = "";
		int numeroArabo = arabic;
		int resto1 = numeroArabo % 10;/*resto1 rappresenta le unità del
					numero arabo*/
		numeroArabo = numeroArabo / 10;//elimino la cifra delle unità
		int resto2 = numeroArabo % 10;//resto2 rappresenta le decine
		numeroArabo = numeroArabo / 10;//elimino la cifra delle decine
		if (resto1 == 1 ) numeroRomano = "I";
		else if (resto1 == 2 ) numeroRomano = "II";
		else if (resto1 == 3 ) numeroRomano = "III";
		else if (resto1 == 4 ) numeroRomano = "IV";
		else if (resto1 == 5 ) numeroRomano = "V";
		else if (resto1 == 6 ) numeroRomano = "VI";
		else if (resto1 == 7 ) numeroRomano = "VII";
		else if (resto1 == 8 ) numeroRomano = "VIII";
		else if (resto1 == 9 ) numeroRomano = "IX";
		if (resto2 == 1 ) numeroRomano = "X" + numeroRomano;
		else if (resto2 == 2 ) numeroRomano = "XX" + numeroRomano;
		else if (resto2 == 3 ) numeroRomano = "XXX" + numeroRomano;
		else if (resto2 == 4 ) numeroRomano = "XL" + numeroRomano;
		else if (resto2 == 5 ) numeroRomano = "L" + numeroRomano;
		else if (resto2 == 6 ) numeroRomano = "LX" + numeroRomano;
		else if (resto2 == 7 ) numeroRomano = "LXX" + numeroRomano;
		else if (resto2 == 8 ) numeroRomano = "LXXX" + numeroRomano;
		else if (resto2 == 9 ) numeroRomano = "XC" + numeroRomano;
		if (numeroArabo == 0)return numeroRomano;
		else return "C" + numeroRomano;/*con questa operazione, in
			realtà, siamo in grado di convertire numeri fino al 199*/
	}
}
```

ArabicToRomanTester.java

```
import java.util.Scanner;

public class ArabicToRomanTester
{
	public static void main(String[] args)
	{
		Scanner in = new Scanner(System.in);
		System.out.println("Inserire un numero tra 1 e 100 da convertire");
		int numero = in.nextInt();
		if (numero <= 0 || numero > 100)
			System.out.println("Numero inserito non valido!");
		else
		{
			ArabicToRoman arabic = new ArabicToRoman(numero);
```

```
                System.out.println("Il numero richiesto è: " +
                        arabic.converti());
            }
        }
}
```

Esercizio 6.8 – ArabicToRomanAvanzato.java

Testo:

Realizzare un convertitore arabo – romano che sfrutti tutti i simboli romani disponibili nel codice ASCII (I – V – X – L – C – D – M). Con questi simboli si possono scrivere tutti i numeri compresi tra 1 e 3999.

Consigli:

Il valore del numero è la somma dei valori dei caratteri. I è 1, II è 2, e III è 3. VI è 6 ("5 e 1"), VII è 7 e VIII è 8. I "caratteri di decina" (I, X, C, e M) possono esere ripetuti fino a tre volte. Alla quarta, si deve sottrarre uno dal più vicino "carattere di quintina" (V, L, D). Non si può rappresentare 4 come IIII, lo si deve rappresentare con IV ("1 in meno di 5"). 40 è scritto come XL, 41 come XLI, 42 come XLII, 43 come XLIII ed infine 44 come XLIV ("10 in meno di 50, più uno in meno di 5"). Similmente, arrivati al 9, si deve sottrarre dal "carattere di decina" immediatamente superiore. Ovvero una cifra inferiore scritta a sinistra di una cifra con il valore immediatamente maggiore si sottrae. 8 è VIII, ma 9 è IX ("uno in meno di dieci"), non VIIII (in quanto il carattere I non può essere ripetuto quattro volte). 90 è XC, 900 è CM. I "caratteri di quintina" non possono essere ripetuti. 10 è sempre rappresentato come X, mai come VV. 100 è sempre C, mai LL. Le cifre dei numeri romani sono sempre scritte dal più grande al più piccolo (ordine decrescente) e letti da sinistra a destra, per cui l'ordine dei caratteri è molto importante. DC è 600; CD è un numero completamente diverso (400, "100 meno di 500"). CI è 101; IC non è un numero romano valido (perché non si può sottrarre 1 direttamente da 100; 99 si deve scrivere XCIX, "10 in meno di 100 e poi 1 in meno di 10").

ArabicToRomanAvanzato.java

```
public class ArabicToRomanAvanzato
{
      /*converte il numero arabo in numero romano
      @param numeroArabo il numero da convertire
      @return la conversione in numero romano*/
      public static String converti(int numeroArabo)
      {
            String numeroRomano = "";
            int i = 1;
            int cinquina = 0;
            while (numeroArabo != 0)
            {
                  int resto = numeroArabo % 5;
                  numeroArabo -= resto;
                  if (numeroArabo % 10 == 5)
                  {
                        cinquina = 1;
                        numeroArabo -= 5;
                  }
                  if (resto >= 0 && resto <= 3)
                  {
                        int n = resto;
                        while (n > 0)
                        {
                        numeroRomano = getSimbolo(i) + numeroRomano;
                        n -- ;
                        }
                        if (cinquina == 1)
```

```
                        numeroRomano = getSimbolo(i + cinquina) +
                        numeroRomano;
                }
                if (resto == 4)
                {
                      if (cinquina == 1)
                        numeroRomano = getSimbolo(i) + getSimbolo(i + 2) +
                        numeroRomano;
                      else
                        numeroRomano = getSimbolo(i) + getSimbolo(i + 1) +
                        numeroRomano;
                }
                cinquina = 0;
                i = i + 2;
                numeroArabo = Math. round(numeroArabo / 10);
          }
      return numeroRomano;
      }

      /*metodo ausiliario che restituisce i simboli romani
      @param posizione il riferimento al simbolo
      @return il simbolo da inserire*/
      public static String getSimbolo(int posizione)
      {
            if (posizione == 1) return "I";
            else if (posizione == 2) return "V";
            else if (posizione == 3) return "X";
            else if (posizione == 4) return "L";
            else if (posizione == 5) return "C";
            else if (posizione == 6) return "D";
            else if (posizione == 7) return "M";
            return "errore in getSimbolo";
      }
}
```

ArabicToRomanAvanzatoTester.java

```
import java.util.Scanner;

public class ArabicToRomanAvanzatoTester
{
      public static void main(String[] args)
      {
            Scanner in = new Scanner(System.in);
            System.out.println("dammi un numero tra 1 e 3999 da convertire");
            int numero = in.nextInt();
            if (numero < 1 || numero > 3999)
                  System.out.println("Numero inserito non valido!");
            else
                  System.out.println("Il numero richiesto è: " +
                        ArabicToRomanAvanzato.converti(numero));
      }
}
```

CAPITOLO 7 Iterazioni

REMIND:

(Paragrafo 7.1) WHILE

- while → esegue l'enunciato finché la condizione è vera
- do…while → esegue l'enunciato fintanto che non si verifica la condizione
- boolean done = false; while(!done) {} → done è una 'flag' (bandierina)

Esercizio 7.1 - Fact.java

Testo:

Scrivere un programma che calcoli il fattoriale di un numero assegnato n.

Consigli:

La difficoltà maggiore riguarda l'implementazione in codice del fattoriale, ma questa difficoltà può esser risolta con il seguente ragionamento. Si crea un metodo, conv(), nel quale si creano due variabili locali: i (valore temporaneo) e result (il risultato finale dell'operazione). Fintanto che i è minore o uguale al valore assegnato dall'utente, si moltiplica i per il risultato, e la variabile i viene incrementata. L'operazione si concluderà solo quando i sarà pari a n, ed il metodo stamperà il risultato finale.

Fact.java

```
public class Fact
{
	private int n;

	/* si costruisce un calcolatore per il fattoriale
	@param k il numero di partenza*/
	public Fact(int k)
	{
		n = k;
	}

	/* si calcola il fattoriale
	@return il risultato*/
	public long conv()
	{
		int i = 1;
		long result = 1;
		while (i <= n)
		{
			result = result * i;
			i++;
		}
		return result;
    }
}
```

FactTester.java

```
import java.util.Scanner;
public class FactTester
{
	public static void main(String[] args)
	{
		Scanner in = new Scanner(System.in);
		System.out.println("Inserisci il numero del quale vuoi
			calcolare il fattoriale:”);
		int a = in.nextInt();
		Fact n = new Fact(a);
		System.out.println(n.conv());
	}
}
```

Esercizio 7.2 - PrimeGenerator.java (Esercizio P7.11 pag. 250 - 251)

Testo:

Scrivere un programma che chieda all'utente un numero intero e che poi stampi tutti i numeri primi fino al numero stesso. Per esempio, se l'utente immette 20, il programma stamperà:

2
3
5
7
11
13
17
19

Ricordarsi che un numero è primo se non è divisibile per nessun altro numero, salvo 1 e se stesso. Usate una classe PrimeGenerator con un metodo nextPrime.

Consigli:

Le maggiori difficoltà dell'esercizio possono essere riscontrate nel metodo boolean isPrime(int a), il quale deve definire se un dato numero in ingresso è primo. Si creino due variabili locali: una boolean, definita primo, che restituisce true se il numero è primo (funziona da sentinella), l'altra int, indicata con i, che serve da denominatore per verificare se la divisione tra il numero dato in ingresso ed i risulta zero. Fintanto che i è minore o pari al dato in ingresso e la divisione tra il numero stesso e i dà resto, allora s'incrementa la variabile i. Se risulta che i è pari al numero dato, ciò significa che è primo (un numero si definisce primo se è divisibile solo per se stesso e per 1), e quindi il programma restituisce true, altrimenti false.
Il metodo String getFirstPrimes(int input), invece, stampa tutti i numeri primi compresi tra 1 e il dato iniziale: ciò viene eseguito con un ciclo for, che restituisce i valori sottoforma di stringhe.

PrimeGenerator.java

```
public class PrimeGenerator
{
	private int n;

	/* si costruisce un generatore di numeri primi
	@param n il numero fino a cui devo trovare i primi */
	public PrimeGenerator(int numero)
	{
		n = numero;
	}

	/* si verifica se un numero è primo
	@param a il numero su cui eseguo il test
	@return il risultato del test */
	public boolean isPrime(int a)
	{
		if (a <= 1)
			return false;
		int i = 2;
		while (i <= a / 2) // si può dimostrare che il limite di i
				   // può essere ridotto ad a/2 anzichè a
			{
			if (a % i == 0)
				return false;
			i++;
			}
		return true;
	}

	public String getFirstPrimes()
	{
		String primi = "";
		for (int i=2; i <= n; i++)
		{
			if (isPrime(i))
```

```
                        primi = primi + i + "\n";
            }
            return primi;
      }
}
```

PrimeGeneratorTester.java

```
import java.util.Scanner;

public class PrimeGeneratorTester
{
public static void main(String[] args)
      {
            Scanner console = new Scanner(System.in);
            System.out.println("Inserisci il numero fino a cui devo trovare i
            primi");
            int n = console.nextInt();
            PrimeGenerator max = new PrimeGenerator(n);
            System.out.println(max.getFirstPrimes());
      }
}
```

Esercizio 7.3 - TextStatistic.java

Testo:
Scrivere un programma che sia in grado di verificare la presenza di una parola all'interno di un testo.

Consigli:
Il metodo stat(String word) funziona da "motore di ricerca": se nel testo compare la parola ricercata, s'incrementa la variabile locale result, che indica il numero di ripetizioni del termine dato in ingresso. Alla fine, esso restituirà il numero totale di volte con cui la parola cercata è presente nel testo.

TextStatistic.java

```
public class TextStatistic
{
      private String text;

      /* si costruisce un oggetto che cerca in una stringa
      @param unTesto la stringa su cui cercare */
      public TextStatistic(String unTesto)
      {
            text = unTesto;
      }

      /* si conta quante volte una parola compare in un testo
      @param word La parola da cercare
      @return result Quante volte la parola cercata compare */
      public int stat(String word)
      {
            int result = 0;
            for (int i = 0; i <= (text.length() - word.length()); i++)
              if (text.substring(i, i +
                   word.length()).toUpperCase().equals(word.toUpperCase()))
                result++;
            return result;
      }
}
```

TextStatisticTester.java

```
import java.util.Scanner;

public class TextStatisticTester
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);
		System.out.println("Immettere il testo sul quale
		si desidera effettuare la ricerca");
		String text = input.nextLine();
		TextStatistic times = new TextStatistic(text);
		System.out.println("Immettere la parola che si desidera cercare");
		String word = input.nextLine();
		System.out.println(times.stat(word));
	}
}
```

Esercizio 7.4 - FibonacciGenerator.java

Testo:

Scrivere un programma che calcoli l'n-esimo numero della serie di Fibonacci.

Consigli:

L'esercizio non presenta particolari difficoltà. Si realizza il programma creando un convertitore non universale, in cui il numero n specificato in ingresso dal tester viene incorporato nella classe sotto forma di variabile d'istanza non successivamente modificabile.

FibonacciGenerator.java

```
/*  Creo una classe con un metodo che prende un numero intero n
    e restituisce l'n-esimo numero della serie di Fibonacci       */
public class FibonacciGenerator
{
	private int indice;

	public FibonacciGenerator(int unNumero)
	{
		indice = unNumero;
	}

	public long getFibonacci()
	{
		long result = 0;
		long temp1 = 0;
		long temp2 = 1;
		for (int i = 0; i < indice; i++)
			{
				result = temp1 + temp2;
				temp1 = temp2;
				temp2 = result;
			}
		return result;
	}
```

FibonacciGeneratorTester.java

```
import java.util.Scanner;

public class FibonacciGeneratorTester
{
	public static void main (String[] args)
	{
```

```
            Scanner in = new Scanner(System.in);
            System.out.println("Inserire un numero intero positivo n
            per ottenere l'n-esimo numero della serie di Fibonacci");
            if (in.hasNextInt())
            {
                  int indice = in.nextInt();
                  if (indice > 0)
                  {
                        FibonacciGenerator fib = new FibonacciGenerator(indice);
                        System.out.println("Il numero di Fibonacci cercato
                        è " + fib.getFibonacci());
                  }
                  else
                        System.out.println("Numero inserito non valido!");
            }
            else
                  System.out.println("Numero inserito non valido!");
      }
}
```

Esercizio 7.5 - FibonacciGeneratorStatic.java

Testo:

Scrivere un programma che calcoli l'n-esimo numero della serie di Fibonacci.

Consigli:

In questa seconda versione si realizza il programma creando un metodo statico. In questo modo, il programma puo' essere utilizzato per diverse conversioni.

FibonacciGeneratorStatic.java

```
/*Creo una classe con un metodo statico che prende un numero intero n
 * e restituisce l'n-esimo numero della serie di Fibonacci*/
public class FibonacciGeneratorStatic
{
      public static long getFibonacciAt(int aNumber)
      {
            long result = 0;
            long temp1 = 0;
            long temp2 = 1;
            for (int i = 0; i < aNumber; i++)

                  {
                        result = temp1 + temp2;
                        temp1 = temp2;
                        temp2 = result;
                  }
            return result;
      }
}
```

FibonacciGeneratorStaticTester.java

```
import java.util.Scanner;

public class FibonacciGeneratorStaticTester
{
      public static void main (String[] args)
      {
            Scanner in = new Scanner(System.in);
            System.out.println("Inserire un numero intero positivo n
            per ottenere l'n-esimo numero della serie di Fibonacci");
            int n = in.nextInt();
            System.out.println("Il numero di Fibonacci cercato
```

```
            è " + FibonacciGeneratorStatic.getFibonacciAt(n));
      }
}
```

CAPITOLO 8 Vettori e Array

REMIND:

(Paragrafo 8.1) ARRAY

- array → sequenza di valori del medesimo tipo.
- i valori degli indici iniziano da 0 (zero)
- data.length → lunghezza dell'array data (notare: length non è seguito da parentesi).
- la loro lunghezza è fissa.

(paragrafo 8.2) VETTORI

- ArrayList, *vantaggi*:
  - un ArrayList può crescere e calare di dimensione
  - ci sono metodi per svolgere le operazioni più comuni (inserimento, rimozione…)
- *Metodi*:
  - .add(a) → aggiunge un oggetto a alla fine della sequenza
  - .add(i, c) → aggiunge nella posizione i l'oggetto c e sposta i successivi in avanti di una posizione
  - .remove(i) → elimina l'elemento nella posizione i
  - .size() → restituisce la dimensione dell'array
  - .get(i) → restituisce l'oggetto nella posizione i
  - .set(i, a) → assegna il valore a alla posizione i

(paragrafo 8.3) INVOLUCRI E AUTOIMPACCHETTAMENTO (wrapping)

- autoimpacchettamento → conversione tra tipi primitivi e le corrispondenti classi involucro: avviene automaticamente (es. pag. 265)
- auto-unboxing → gli oggetti involucro vengono tolti dalla scatola per generare valori di tipo primitivo

(paragrafo 8.3) IL CICLO "FOR" GENERALIZZATO

- scandire tutti gli oggetti di una sequenza

```
es: double[] data = ...;
 double sum = 0;
 for (double e: data)
 { sum = sum + e }
```

- utilizzato per una scansione dall'inizio alla fine di tutti gli elementi di una raccolta

(paragrafo 8.7) COPIARE ARRAY

- clone → vera copia di un array
  `double[] prices = (double[]) data.clone();`
- copiare elementi da un array a un altro
  → `System.arraycopy(from, fromStart, to , toStart, count);`
- aggiungere un nuovo elemento nella posizione i nell'array data
  → `System.arraycopy(data, i, data, i+1, data.length-i-1); data[i] = x;`
- rimuovere un elemento nella posizione i nell'array data
  → `System.arraycopy(data, i, data, i+1, data.length-i-1);`

Esercizio 8.1 – Purse.java (Esercizio P8.2 pag. 288)

Testo:

Realizzare una classe Purse ("borsellino"), i cui esemplari siano adatti a contenere uan raccolta di monete. Per semplicità, si memorizzeranno semplicemente i nomi delle monete in un ArrayList<String>.
Fornine un metodo void addCoin(String coinName) e un metodo toString che visualizzi le monete presenti nel borsellino, nel formato seguente:

Purse[Quarter, Dime, Nickel, Dime].

Consigli:

Il metodo non presenta particolari difficoltà. Si raccomanda d'importare solamente la classe ArrayList, altrimenti Java non riesce ad individuare il comando "add" del metodo void addCoin (String coinName).

Purse.java

```
import java.util.ArrayList;

public class Purse
{
	private ArrayList<String> coins;

	/*crea un oggetto della classe Purse, viene inizializzata la variabile
	coins*/
	public Purse()
	{
		coins = new ArrayList<String>();
	}

	/*aggiunge una moneta al purse
	@param coinName il nome della moneta da aggiungere*/
	public void addCoin(String coinName)
	{
		coins.add(coinName);
	}

	/*traduce in stringa il contenuto del purse
	@return il contenuto nel formato richiesto nel testo dell'esercizio*/
	public String toString()
	{
		if (coins.size() == 0) return "Empty purse!";
		String result = "Purse[";
		for (int i = 0; i < coins.size(); i++)
		{
			result = result + coins.get(i);
			if (i < coins.size() -1) result = result  + ", ";
		}
		return result + "]";
	}
}
```

PurseTester.java

```
import java.util.Scanner;

public class PurseTester
```

```
{
	public static void main(String[] args)
	{
		Scanner in = new Scanner(System.in);
		Purse p = new Purse();
		boolean add = true;
		System.out.println("Inserire una moneta nel borsellino, oppure 0
			(zero) per terminare.");
		while (add)
		{
			String s = in.next();
			if(s.equals("0"))
				add = false;
			else
				p.addCoin(s);
		}
		System.out.println(p.toString());
	}
}
```

Esercizio 8.2 - Ascensore

Testo:

Si desidera simulare un ascensore in funzione in un palazzo. L'escensore ospita persone ed esegue le loro prenotazioni di spostamento ad un certo piano.
Costruire una classe Prenotazioni, con informazione su numero clienti prenotanti e numero piano prenotato. Sviluppare metodi di accesso ed un metodo toString che descriva la prenotazione stessa.
Costruire inoltre una classe Ascensore con le seguenti info: max numero piani, piano corrente, max numero persone, numero persone corrente, e lista prenotazioni. La lista delle prenotazioni ha una capienza massima prefissata, usare un array di Prenotazioni. le prenotazioni vengono servite con la politica primo arrivato primo servito, usare una sentinella come piu` volte spiegato in classe. Sviluppare i seguenti metodi.
entra: incrementa il numero di persone nell'escensore e mette in coda la relativa prenotazione. muovi: porta l'ascensore al piano specificato dalla prima prenotazione trovata, fa uscire le persone relative ed aggiorna la lista delle prenotazioni. toString: restituisce una stringa con la descrizione dello stato attuale dell'escensore. In tutti i casi in cui venga violata una condizione (troppe persone in ascensore, troppe prenotazioni, ecc.) stampare un messaggio di errore ed uscire dal metodo relativo.

Consigli:

La classe Prenotazioni non presenta difficoltà particolari. Può risultare di più difficile risoluzione la classe Ascensore, in particolare il metodo muovi(), il quale presuppone di: individuare il piano cercato, sottrarre le persone che sono uscite dall'ascensore alla "fermata" richiesta ed, infine, aggiornare l'array delle prenotazioni.

Prenotazioni.java

```
public class Prenotazioni
{
	private int persone;
	private int piano;

	public Prenotazioni(int nPersone,int nPiano)
	{
		persone = nPersone;
		piano = nPiano;
	}

	public int getPersone()
	{
```

```
            return persone;
      }

      public int getPiano()
      {
            return piano;
      }

      public String toString()
      {
            String s = "il piano " +piano +" è stato prenotato da " + persone+"
                  persone" +"\n";
            return s;
      }
}
```

PrenotazioniTester.java

```
import java.util.Scanner;
public class PrenotazioniTester
{
      public static void main(String[] args)
      {
            Scanner input = new Scanner(System.in);
            System.out.println("Inserire numero di piano");
            int nPiano = input.nextInt();
            System.out.println("Inserire numero di persone che vogliono recarsi
            a quel piano");
            int nPersone = input.nextInt();
            Prenotazioni tester=new Prenotazioni(nPersone,nPiano);
            System.out.println(tester.toString());
      }
}
```

Ascensore.java

```
public class Ascensore
{
      private final int MAXPIANI = 4;
      private final int MAXPERSONE = 5;
      private Prenotazioni[] prenota;
      private int sentinella;
      private int pianoCorr;
      private int personeCorr;

      /*si costruisce l'ascensore inizializzando le sue variabili d'istanza
      @param unMax il numero massimo di prenotazioni che si possono accodare*/
      public Ascensore(int unMax)
      {
            prenota = new Prenotazioni[unMax];
            sentinella = 0;
            pianoCorr = 0;
            personeCorr = 0;
      }

      /* s'incrementa il numero di persone nell'ascensore e
      mette in coda la relativa prenotazione quando ciò è possibile.
      @param persone le persone che vogliono entrare in ascensore
      @param piano il piano cui queste persone vogliono andare*/
      public void entra(int persone,int piano)
      {
            if (personeCorr + persone > MAXPERSONE)
            {
```

```
                System.out.println("Errore: capacità carico superata");
            }
            else if (piano > MAXPIANI || piano < 0)
            {
                System.out.println("Errore: piano non esistente");
            }
            else if (sentinella == prenota.length)
            {
                System.out.println("Errore: capacità prenotazioni superata");
            }
            else
            {
                personeCorr = personeCorr+persone;
                prenota[sentinella] = new Prenotazioni(persone,piano);
                sentinella++;
            }
        }

        /* porta l'ascensore al piano specificato dalla prima prenotazione
        trovata, fa uscire le persone relative ed aggiorna la lista delle
        prenotazioni */
        public void muovi()
        {
            if (sentinella == 0) return;
            pianoCorr = prenota[0].getPiano();
            personeCorr = personeCorr - prenota[0].getPersone();
            System.arraycopy(prenota,1,prenota,0,prenota.length-1);
            sentinella --;
        }

        /* restituisce una stringa con la descrizione
        dello stato attuale dell'escensore, comprese prenotazioni in coda
        @return lo stato attuale dell'ascensore */
        public String toString()
        {
            String r = "";
            for (int i = 0; i < sentinella ; i++) r = r + prenota[i].toString()
                + "\n";
            String s = "";
            return s = "Piano corrente: " + pianoCorr + "." + "\n" +  "Persone
                in ascensore: " + personeCorr + "\n" + r;
        }
}
```

AscensoreTester.java

```
import java.util.Scanner;
public class AscensoreTester
{
        public static void main(String [] args)
        {
            Scanner in = new Scanner(System.in);
            System.out.println("Inserisci il massimo delle prenotazioni
                possibili");
            int max = in.nextInt();
            Ascensore elevator = new Ascensore(max);
            String t = "";
            int contatore = 0;
            while(!t.equalsIgnoreCase("q"))
            {
                System.out.println("inserire numero passeggeri");
                int n = in.nextInt();
```

```
                System.out.println("inserire il piano che si vuole
                      raggiungere");
                int p = in.nextInt();
                elevator.entra(n,p);
                contatore ++;
                System.out.println("Info: per termirare le prenotazioni,
                      digitare Q, altrimenti una lettera qualsiasi.");
                t=in.next();
                System.out.println("fatto");
                System.out.println(elevator.toString());
          }
          for (int i=0; i < contatore; i++)
          {
                elevator.muovi();
                System.out.println(elevator.toString());
          }
    }
}
```

Esercizio 8.3 - MinMaxArray.java

Testo:

Scrivere un programma che individui il valore massimo e minimo all'interno di un' array.

Consigli:

I metodi getMin() e getMax(), dal punto di vista dell'implementazione, sono praticamente identici: entrambi definiscono una variabile locale intera, a cui viene assegnata la prima posizione dell'array temporanea. Con un ciclo for, si passano in rassegna tutti i valori dell'array: se se il valore della posizione successiva è maggiore o minore (a seconda del caso) di quello contenuto nella posizione 0, allora lo si sostituisce. Alla fine, il metodo restituisce il valore massimo o minimo.

MinMaxArray.java

```
public class MinMaxArray
{
      private int[] pippo;

      /*costruisce un oggetto della classe MinMaxArray
      @param unVettore il vettore che l'oggetto deve contenere*/
      public MinMaxArray(int[] unVettore)
      {
            pippo = new int[unVettore.length];
            System.arraycopy(unVettore, 0, pippo, 0, unVettore.length);
      }

      /*restituisce il valore massimo contenuto nell'array
      @return il valore massimo contenuto nell'array*/
      public int getMax()
      {
            int massimo = pippo[0];
            for(int i= 1; i < pippo.length; i++)
            {
                  if(massimo <= pippo[i])
                  massimo = pippo[i];
            }
            return massimo;
      }

      /*restituisce il valore minimo contenuto nell'array
      @return il valore minimo contenuto nell'array*/
      public int getMin()
```

```
	{
		int minimo = pippo[0];
		for(int i= 1; i< pippo.length; i++)
		{
			if(minimo>= pippo[i])
			minimo = pippo[i];
		}
		return minimo;
	}
}
```

MinMaxArrayTester.java

```
import java.util.Scanner;
public class MinMaxArrayTester
{
	public static void main(String[] args)
	{
		Scanner in = new Scanner(System.in);
		System.out.println("Inserire la lunghezza del vettore da
			costruire:");
		int lunghezza = in.nextInt();
		if (lunghezza < 1)
			System.out.println("Errore: numero inserito non valido!");
		else
		{
			int[] myArray = new int[lunghezza];
			for(int i = 0; i < myArray.length; i++)
			{
				System.out.println("Inserire un intero da aggiungere
					nella posizione " + i);
				myArray[i] = in.nextInt();
			}
			MinMaxArray pluto = new MinMaxArray(myArray);
			System.out.println("numero minimo: ");
			System.out.println(pluto.getMin());
			System.out.println("numero massimo: ");
			System.out.println(pluto.getMax());
		}
	}
}
```

Esercizio 8.4 – AlterniArray.java (Esercizio P8.10 pag. 291 – 292)

Testo:

Scrivere un programma che legga una sequenza di numeri interi, memorizzandola in un array, e ne calcoli la somma algebrica degli elementi alternando i segni (il primo – il secondo + il terzo – il quarto ecc…).

Consigli:

L'esercizio non presenta particolari difficoltà. Si pone l'attenzione esclusivamente sul metodo sommaAlterna(), che restituisce un intero: se la posizione dell'array è pari, si aggiunge il valore alla somma, altrimenti lo si sottrae (si assume che la posizione 0 sia pari).

AlterniArray.java

```
import java.util.ArrayList;

public class AlterniArray
{
	ArrayList<Integer> lista;

	/*costruisce un oggetto della classe AlterniArray*/
```

```
	public AlterniArray()
	{
		lista = new ArrayList<Integer>();
	}

	/*aggiunge un valore all'ArrayList lista
	@param n il valore da aggiungere*/
	public void inserisciVal(int n)
	{
		lista.add(n);
	}

	/*converte il contenuto della lista in stringa
	@return gli elementi della lista*/
	public String toString()
	{
		String s = "";
		for (int i = 0; i < lista.size(); i++)
			s = s + lista.get(i) + " ";
		return s;
	}

	/*effettua la somma a segni alterni
	@return somma alterna*/
	public int sommaAlterna()
	{
		int somma = 0;
		for (int i = 0; i < lista.size(); i++)
		{
		if (i % 2 == 0)
			somma = somma + lista.get(i);
		else
			somma = somma - lista.get(i);
		}
	return somma;
	}
}
```

AlterniArrayTester.java

```
import java.util.Scanner;

public class AlterniArrayTester
{
	public static void main(String[] args)
	{
		Scanner in = new Scanner(System.in);
		AlterniArray vettore = new AlterniArray();
		String scelta = "Y";
		while(scelta.toUpperCase ().equals("Y"))
		{
			System.out.print("Inserisci un numero intero: ");
			int i = in.nextInt();
			vettore.inserisciVal(i);
			System.out.print("Se vuoi inserire un altro numero digita 'Y'
				(yes), altrimenti premi un tasto: ");
			scelta = in.next();
		}
		System.out.println("La lista inserita è: " + vettore.toString());
		System.out.println("La somma a elementi alterni e': " +
			vettore.sommaAlterna());
	}
}
```

Esercizio 8.5 - Permutazioni (Esercizio 8.11 pag. 290)

Testo:

Scrivete un programma che produca 10 permutazioni casuali dei numeri da 1 a 10. Per generare una permutazione casuale dovete riempire un array con i numeri da 1 a 10 in modo che non ve ne siano 2 uguali. Potreste farlo in modo brutale, chiamando Random.nextInt finché produce un valore non ancora presente nell'array, ma dovreste, invece, usare un metodo più intelligente: create un secondo array e inseritevi i numeri da 1 a 10; poi, prendetene uno a caso, rimovetelo e accordatelo all'array che contiene la permutazione. Ripetete 10 volte. Realizzate una classe PermutationGenerator che abbia il metodo int[] nextPermutation.

Consigli:

Suggerimenti sullo sviluppo dell'esercizio sono già contenuti nel testo del programma

PermutationGenerator.java

```
// Una classe che produce 10 permutazioni casuali sei numeri da 1 a 10
import java.util.Random;
public class PermutationGenerator
{
	private int[] permutazione;
	private int[] numeri;

	public PermutationGenerator()
	{
		// si costruisce un array per metterci la sequenza permutata
		permutazione = new int[10];
		// si costruisce un array con i numeri da 1 a 10
		numeri = new int[10];
		for (int i=0; i<10; i++)
		{
			numeri[i]=i + 1;
		}
	}
	// si compila l'array permutazioni con i numeri casuali
	public int[] nextPermutation()
	{
		Random genera = new Random();
		// il contatore si riferisce all'array permutazione
		int cont = 0;
		while (cont < 10)
		{
			// variabile temporanea, contiene un numero casuale
			int temp1 = genera.nextInt(10);
			// per evitare ripetizioni
			if ((numeri[temp1] != 0))
			{
				int temp2 = numeri[temp1];
				permutazione[cont] = temp2;
				numeri[temp1] = 0;
				cont++;
			}
		}
		return permutazione;
	}
	// si trasforma l'array in una stringa
	public String toString()
	{
		String s = "";
```

```
            for (int i=0; i<10; i++)
            {
                  s = s + permutazione[i] + " ";
            }
            return s;
      }
}

PermutationGeneratorTester.java
public class PermutationGeneratorTester
{
      public static void main(String[] args)
      {
            final int NUM_PERM = 10;
            for (int i = 0; i < NUM_PERM; i++)
            {
                  PermutationGenerator p = new PermutationGenerator();
                  int[] aPerm = p.nextPermutation();
                  for (int j = 0; j < 10; j++)
                  System.out.print(aPerm[j] + " ");
                  System.out.println("\n");
            }
      }
}
```

CAPITOLO 9 Progettazione di Classi

REMIND:

(Paragrafo 9.4) EFFETTI COLLATERALI

- *effetto collaterale* → qualunque tipo di comportamento al di fuori del metodo stesso.

(Paragrafo 9.4) AMBITO DI VISIBILITÀ

- *ambito di visibilità* → porzione di programma all'interno della quale si può accedere alla variabile.
  Ambito di visibilità di:
  - *variabile locale* → dal punto in cui viene creata fino alla fine del blocco ({…}) che la contiene.
  - *parametro esplicito* → dall'inizio alla fine del metodo che lo contiene.
  - *variabili d'istanza* → visibilità di classe (in tutta la classe).

- Se si utilizza una variabile d'istanza nel campo i visibilità di una variabile locale con lo stesso nome, prevale la variabile locale, che mette in ombra il campo d'istanza. Per riferirsi alla variabile d'esemplare si può utilizzare la forma this.variabileEsemplare

Nota: Gli esercizi di questo capitolo non sono molto significativi, ragion per cui non verranno messi in questa dispensa.

CAPITOLO 16 Ricorsione

Esercizio 16.1 - Find.java (Es. P16.4 pag. 556)

Testo:
Usare la ricorsione per realizzare un metodo boolean che verifica se un intero è contenuto in un array.

Consigli:
Il metodo ricorsivo può risultare alquanto utile per poter snellire e semplificare soluzioni che, spesso, possono essere lunghe e ripetitive: esso, infatti, prevede la scomposizione del problema in parti sempre più piccole, fino a ridurlo al "nocciolo", si consenta l'espressione, ovvero individuare il caso base. Per poter applicare correttamente la ricorsione, però, bisogna rispettare due regole:

1. Evitare l'uso dei cicli for e while: essi vengono usati per creare programmi iterativi.
2. Evitare assolutamente di interpretare la soluzione del programma come la esegue Java: il rischio è non riuscire a districare alcuna soluzione in modo ricorsivo.

Si consiglia vivamente, per risolvere un programma ricorsivamente, di seguire il dato schema:

- leggere più volte la consegna dell'esercizio
- scomporre il problema e risolverlo per i casi più semplici, definiti base
- applicare la stessa soluzione ai casi più importanti.

InArrayFinder.java

```
public class InArrayFinder
{
	private int[] array;

	//costruisce un oggetto contenente un array di interi
	public InArrayFinder(int[] anArray)
	{
		array = anArray;
	}

	//verifica se un numero intero è contenuto nell'array
	public boolean test(int cercato)
	{
		if (array.length == 0)
			return false;
		if (array[0] == cercato)
			return true;
		int[] provvisorio = new int[array.length - 1];
		System.arraycopy(array, 1, provvisorio, 0, array.length -1);
		InArrayFinder shorter = new InArrayFinder(provvisorio);
		return shorter.test(cercato);
	}

	//restituisce il contenuto dell'array in formato stringa
	public String toString()
	{
		String s = "";
		for(int i = 0; i < array.length; i++)
			s = s + array[i] + "  ";
		return s;
	}
}
```

FindTester.java

```
import java.util.Random;
import java.util.Scanner;

public class InArrayFinderTester
{
	public static void main(String[] args)
	{
		int MAX = 10;
		int[] mioVettore = new int[MAX];
		Random gen = new Random();
		for (int i = 0; i < MAX; i++)
			mioVettore[i] = gen.nextInt(MAX) + 1;
		Scanner in = new Scanner(System.in);
		System.out.println("Inserisci il numero da trovare");
		int elemento = in.nextInt();
		InArrayFinder found = new InArrayFinder(mioVettore);
		if (found.test(elemento))
			System.out.println(elemento + " compare nell'array");
		else
			System.out.println(elemento + " non compare nell'array");
		System.out.println("Puoi controllare se non ti fidi: " +
			found.toString());
	}
}
```

Esercizio 16.2 - FindAux.java

Testo:

Vedere consegna esercizio precedente

Consigli:

La consegna del programma è identica a quella precedente, solo che s'invita l'utente ad utilizzare un metodo ausiliario nella soluzione del problema, ovvero creare un metodo parallelo a quello principale, allo scopo di snellire e semplificare il codice. Come si vedrà in seguito, i metodi ausiliari saranno molto utilizzati, proprio a tal fine.

```
import java.util.Scanner;
public class FindAux
{
	private int[] array;
	private int n;
	public FindAux (int[]aa, int nn)
	{
		array = aa;
		n = nn;
	}
	public boolean test()
	{
		// si richiama un metodo privato che effettivamente fa la verifica
		return testAux(0, array.length - 1);
	}
	private boolean testAux(int start, int end)
	{
		// l'array ha lunghezza zero
		if (start > end)
			return false;
		else if (array[start] == n)
			return true;
		else
			return testAux(start + 1, end);
	}
}
```

Esercizio 16.3 - Concatenazione di due stringhe

Testo:
Il seguente esercizio si propone di risolvere la concatenazione di due stringhe, fondere cioè una stringa con un'altra. Per far ciò, si crea un oggetto di classe Concatenazione avente come parametri espliciti due stringhe, disposte nell'ordine con cui verranno unite. Poi si crea un metodo che restituisce una nuova stringa frutto della concatenazione delle due stringhe.

Consigli:
La consegna dell'esercizio richiederebbe l'esecuzione dell'esercizione solo ricorsivamente, ma si è deciso, comunque, di proporre lo stesso esercizio in diverse modalità di sviluppo.

Primo sviluppo: Concatenazione ricorsiva
La ricorsione consiste nel togliere il primo carattere a dx e aggiungerlo a sx, fare questo finchè dx diventa vuota.

Concatena2stringhe.java

```
public class Concatena2stringhe
{
	private String sinistra;
	private String destra;

	public Concatena2stringhe(String unaStringaSx,String unaStringaDx)
	{
		sinistra=unaStringaSx;
		destra=unaStringaDx;
	}

	public String concatena()
	{
		/* caso base:la stringa di dx è vuota, vuol dire che tutte le
		lettere sono a sx e viene restituita la stringa sx*/
		if(destra.length()==0)
			return sinistra;
		/* si crea un oggetto che ha il primo carattere di dx spostato dopo
		l'ultimo di sx*/
		Concatena2stringhe shorter = new Concatena2stringhe(sinistra +
			destra.substring(0,1),destra.substring(1));
		//si richiama la ricorsione in shorter
		return shorter.concatena();
	}
}
```

Concatena2stringheTester.java

```
public class Concatena2stringheTester
{
	public static void main(String[] args)
	{
		Concatena2stringhe zuzi = new Concatena2stringhe("ciao"," belli");
		System.out.println(zuzi.concatena());
	}
}
```

Secondo sviluppo: Concatenazione di 2 stringhe in modo complicato (metodo A)
Si hanno due variabili d'istanza, una stringa destra e una sinistra. Si definiscono due metodi sovraccarichi. Il primo ha un parametro implicito, un int: se il numero dato in ingresso è maggiore della lunghezza della stringa di destra, si restituisce la stringa sinistra, altrimenti si assegna a sinistra la stringa stessa e si passano in rassegna tutti i caratteri della stringa destra.

Il secondo metodo restituisce il risultato del metodo precedente, applicandolo al caso di int n=0.

Concatena2Stringhe.java

```
public class Concatena2stringhe
{
	private String sinistra;
	private String destra;

	public Concatena2stringhe(String unaStringaSx,String unaStringaDx)
	{
		sinistra=unaStringaSx;
		destra=unaStringaDx;
	}

	public String concatena()
	{
		return concatena(0);
	}

	private String concatena(int n)
	{
		if(n>=destra.length())
			return sinistra;
		sinistra = sinistra + destra.substring(n, n + 1);
		n++;
		return concatena(n);
	}
}
```

Concatena2StringheTester.java

```
public class Concatena2stringheTest
{
	public static void main(String[] args)
	{
		Concatena2stringhe brex = new Concatena2stringhe("ciao"," belli");
		System.out.println(brex.concatena());
	}
}
```

Terzo sviluppo: Concatenazione di 2 stringhe complicato (Metodo B)
In questo caso, si usa una terza variabile int che indica la posizione del carattere da trasferire. Si aggiunge a sx il carattere indicato da n senza toglierlo alla stringa dx, incrementando successivamente n in modo che esso punti al prossimo carattere.

```
public class Concatena2stringhe
{
	private String sinistra;
	private String destra;
	private int n;
	public Concatena2stringhe(String unaStringaSx,String unaStringaDx)
	{
		sinistra=unaStringaSx;
		destra=unaStringaDx;
		n=0;
	}
	public String concatena()
	{
		/* caso base: n è più grande della lunghezza della stringa, cioè
		punterebbe a un carattere dopo il termine della stringa*/
```

```
            if(n>=destra.length())
                  return sinistra;
            // si aggiunge a sx il carattere indicato da n della stringa dx
            sinistra=sinistra+destra.substring(n,n+1);
            // si incrementa n in modo che punti al prossimo carattere
            n++;
            // si richiama la ricorsione
            return concatena();
   }
}
```

Concatena2Stringhe.java

```
public class Concatena2stringheTest
{
      public static void main(String[] args)
      {
            Concatena2stringhe zuzi = new Concatena2stringhe("ciao"," belli");
            System.out.println(zuzi.concatena());
      }
}
```

Quarto sviluppo: concatenazione di 2 stringhe statico

Simile alla classe precedente, ma questa non usa variabili d'istanza, quindi non ha bisogno di un metodo costruttore per inizializzarle. è presente solo un metodo statico(capitolo 4).

```
public class Concatena2stringhe
{
      static public String concatenaStatic(String prima,String seconda)
      {
            if(seconda.length()==0)
            return prima;
            /* in mancanza del costruttore, è necessario spostare "manualmente"
            ciascun carattere dalla seconda stringa e aggiungerlo alla prima*/
            prima=prima+seconda.substring(0,1);
            seconda=seconda.substring(1);
            // si richiama la ricorsione
            return concatenaStatic(prima,seconda);
      }
}
```

Concatena2StringheTest.java

```
public class Concatena2stringheTest
{
      public static void main(String[] args)
      {
            System.out.println(Concatena2stringhe.concatenaStatic(new
            String("Java e' "),new String("Bellissimo")));
      }
}
```

Esercizio 16.4 - ContaDispariRic.java

Testo:

Scrivere un programma che restituisca il valore corrispondente alla totalità di numeri dispari presenti all'interno di un array.

Consigli:

Il seguente esercizio non presenta particolari difficoltà, in quanto è necessario solo semplificare il programma in due casi base: se il contenuto di una posizione dell'array è un numero dispari, allora si incrementa la variabile fungente da sentinella, altrimenti quest'ultima rimane invariata.

```
public class ContaDispariRic
{
	private int[] array;

	public ContaDispariRic(int[] n)
	{
		array = n;
	}

	public int contaDispari()
	{
		int c = 0;
		if(array.length == 0) return c;
		int[] a1 = new int[array.length - 1];
		System.arraycopy(array, 1, a1, 0, array.length - 1);
		ContaDispariRic shorter = new ContaDispariRic(a1);
		c = shorter.contaDispari();
		if(array[0] % 2 == 0)
			return c;
		else
			return c + 1;
	}
}
```

ConcatenaDispariRicTester.java

```
public class ContaDispariRicTester
{
	public static void main(String[] args)
	{
		int[] a = new int[5];
		a[0] = 5;
		a[1] = 7;
		a[2] = 6;
		a[3] = 45;
		a[4] = 50;
		ContaDispariRic c = new ContaDispariRic(a);
		System.out.println("I dispari sono " + c.contaDispari());
	}
}
```

Esercizio 16.5 - InvertiStringa.java (Es. P16.1 pag. 556)

Testo:
Scrivere un metodo ricorsivo void reverse() che inverte il contenuto di una frase.

Consigli:
L'esercizio è stato risolto seguendo questo meccanismo: dopo l'individuazione dei casi base, si ricava la prima lettera della parola, si costruisce un oggetto della stessa classe con la stessa parola, escludendo il primo carattere di questa, e si restituisce il nuovo termine invertito.
Nota: per semplificare il codice, il metodo reverse() è stato sviluppato con la possibilità di restituire un valore, nel caso specifico una Stringa.

```
public class InvertiStringa
{
	private String text;

	/* si costruisce un invertitore di stringhe
	@param aText la stringa da invertire */
	public InvertiStringa(String aText)
```

```
	{
		text = aText;
	}

	/* si costruisce un invertitore di stringhe
	@return la stringa invertita */
	public String reverse()
	{
		// casi base
		if (text.length() == 0) return text;
		if (text.length() == 1) return text;
		// caso ricorsivo
		else
		{
			char ch = text.charAt(0);
			InvertiStringa shorter = new InvertiStringa(text.substring(1));
			return shorter.reverse() + ch;
		}
	}
}
```

InvertiStringaTester.java

```
import java.util.Scanner;
public class InvertiStringaTester
{
	public static void main(String[] args)
	{
		System.out.println("Inserisci il testo da invertire ");
		Scanner in = new Scanner(System.in);
		String t = in.nextLine();
		InvertiStringa inv = new InvertiStringa(t);
		System.out.println("");
		System.out.println("Inverto il testo ");
		System.out.println(inv.reverse());
		System.out.println("");
	}
}
```

Esercizio 16.6 - SommaRic.java

Testo:
Scrivere un programma che restituisca la somma tra due valori. Usare il metodo ricorsivo.

Consigli:
La ricorsione prevede il seguente meccanismo: dopo aver individuato i casi base, si definisce un costruttore avente come parametri impliciti due numeri e un metodo somma(), nel quale si costruisce un nuovo oggetto della stessa classe avente come parametri impliciti le variabili d'istanza rispettivamente incrementate e decrementate. Alla fine, esso restituirà il nuovo valore di n1, in quanto n2 diventa zero (vedi casi base).

```
public class SommaRic
{
	private int n1,n2;

	public SommaRic(int num1, int num2)
	{
		n1 = num1;
		n2 = num2;
	}

	public int somma()
```

```
	{
		// casi base
		if (n2 == 0) return n1;
		if (n1 == 0) return n2;
		else
		{
			SommaRic s = new SommaRic(n1+1, n2-1);
			int res = s.somma();
			return res;
		}
	}
}
```

SommaRicTester.java

```
public class SommaRicTester
{
	public static void main(String[] args)
	{
		SommaRic som = new SommaRic(5,4);
		System.out.println(som.somma());
	}
}
```

Esercizio 16.7 - RicercaStringa.java (Esercizio P16.4 pag. 556)

Testo:
Usare la ricorsione per realizzare un metodo boolean find(String t) che verifica se una stringa è contenuta in un testo.

Consigli:
La ricorsione viene risolta eseguendo un semplice algoritmo di ricerca.

RicercaStringa.java

```
public class RicercaStringa
{
	private String testo;

	public RicercaStringa(String s)
	{
		testo = s;
	}

	public boolean find(String t)
	{
		if(t.length() > testo.length())
			return false;
		if (t.equalsIgnoreCase(testo.substring(0, t.length())))
			return true;
		RicercaStringa corto = new RicercaStringa(testo.substring(1));
		return corto.find(t);
	}
}
```

RicercaStringaTester.java

```
import java.util.Scanner;
public class RicercaStringaTester
{
	public static void main(String[] args)
	{
		Scanner in = new Scanner(System.in);
		System.out.print("Scrivi la frase: ");
```

```
            String s = in.nextLine();
            RicercaStringa oggetto = new RicercaStringa(s);
            System.out.print("Inserisci la parola: ");
            String p = in.next();
            if (oggetto.find(p))
                  System.out.println("Si c'e'");
            else
                  System.out.println("No non c'e'");
      }
}
```

Esercizio 16.8 - MaxFinder.java (Esercizio P16.6 pag. 556)

Testo:

Usando la ricorsione, trovare l'elemento maggiore in un array.

Consigli:

Il programma non presenta particolari difficoltà: come in ogni esercizio ricorsivo, si raccomanda di impostare, per prima cosa, i casi base, e poi lo sviluppo dell'esercizio stesso. Si consiglia di trovare l'elemento maggiore nel sottoinsieme che contiene tutti gli elementi tranne il primo. Quindi, si confronti tale massimo con il valore del primo elemento.

MaxFinder.java

```
public class MaxFinder
{
      private int[] numeri;

      // si crea un oggetto contenente un array di numeri.
      public MaxFinder(int[] anArray)
      {
            numeri = anArray;
      }

      //trova il massimo all'interno dell'array
      public int getMax()
      {
            if(numeri.length == 1)
            {
                  int max = numeri[0];
                  return max;
            }
            int[] newNumeri = new int[numeri.length - 1];
            System.arraycopy(numeri, 1, newNumeri, 0, numeri.length - 1);
            MaxFinder nuovo = new MaxFinder(newNumeri);
            int max = nuovo.getMax();
            if(numeri[0] > max)
                  return numeri [0];
            else
                  return max;
      }
}
```

MaxFinderTester.java

```
import java.util.Scanner;

public class MaxFinderTester
{
      public static void main(String[] args)
      {
            Scanner in = new Scanner(System.in);
            int dim = 5;
```

```
            int[] a = new int[dim];
            System.out.println("Costruire un array di " + dim + " numeri
                  interi.");
            for (int i = 0; i < dim; i++)
            {
                  System.out.println("Dammi un numero da inserire nell'array:");
                  a[i] = in.nextInt();
            }
            MaxFinder g = new MaxFinder(a);
            System.out.println("\nNumero max " + g.getMax());
      }
}
```

Esercizio 16.9 - PermutationGenerator.java

Testo:

Scrivere un programma che calcola le permutazioni di una parola.

Consigli:

Per semplificare l'esecuzione del programma, si è utilizzato sia il metodo iterativo che quello ricorsivo. Si veda, a proposito, il metodo getPermutations(): si crea un'arraylist temporanea, di nome result, e si definiscono i casi base. Dopo aver imposto un ciclo for, da 0 alla fine della lunghezza della parola, si creano tre nuovi variabili: una stringa, un oggetto di classe e un'arraylist di stringhe. L'ultima è fondamentale: si aggiungono gli elementi dell'ultimo vettore in result, combinando ogni singolo carattere del termine dato in ingresso con il resto della parola. In tal modo, si ottengono tutte le possibili permutazioni della parola cercata.

```
import java.util.ArrayList[4];
// Questa classe genera le permutazioni di una parola
public class PermutationGenerator
{
      /* si costruisce un generatore di permutazioni.
      @param aWord la parola da permutare */
      private String word;
      public PermutationGenerator(String aWord)
      {
            word = aWord;
      }
      /* si forniscono tutte le permutazioni della parola
      @return un contenente tutte le permutazioni */
      public ArrayList<String> getPermutations()
      {
            ArrayList<String> result = new ArrayList<String>();
            // la stringa vuota ha un'unica permutazione: se stessa
            if (word.length() == 0)
            {
                  result.add(word);
                  return result;
            }
            // s'effettua un ciclo sui caratteri della stringa
            for (int i = 0; i < word.length(); i++)
            {
                  String shorterWord = word.substring(0,i)+word.substring(i+1);
                  PermutationGenerator shorterPermutationGenerator = new
                  PermutationGenerator(shorterWord);
                  ArrayList<String> shorterWordPermutation =
                   shorterPermutationGenerator.getPermutations();
                  for (String s: shorterWordPermutation)
                  {
```

[4] ATTENZIONE: Ogni volta che viene usata un'ArrayList, si ricorda d'importare la specifica libreria

```
                        result.add(word.charAt(i) + s);
                  }
            }
            return result;
      }
}
```

PermutationGeneratorTester.Java

```
import java.util.ArrayList;
public class PermutationGeneratorTester
{
      public static void main(String[] args)
      {
            PermutationGenerator generator = new PermutationGenerator("eat");
            ArrayList<String> permutations = generator.getPermutations();
            for (String s: permutations)
            {
                  System.out.println(s);
            }
      }
}
```

Esercizio 16.10 – PermutationGeneratorDue.java

Testo:

Come per l'esercizio precedente scrivere un programma che calcola le permutazioni di una parola.

Consigli:

A differenza di quanto fatto nell'esercizio precedente, in questo si preleva la prima lettera della parola, si generano le permutazioni della parola più corta in modo ricorsivo ed infine si inserisce la lettera prelevata in tutte le posizioni possibili (dalla prima all'ultima) con un ciclo for.

PermutationGeneratorDue.java

```
import java.util.ArrayList;

public class PermutationGeneratorDue
{
      public String parola;

      /* si costruisce un generatore di permutazioni.
      @param aWord la parola da permutare */
      public PermutationGeneratorDue(String unaParola)
      {
            parola = unaParola;
      }

      /*genera un'ArrayList con tutte le permutazioni della parola
      @return un ArrayList di stringhe*/
      public ArrayList<String> permuta()
      {
            ArrayList<String> permutazioni = new ArrayList<String>();
            //caso base: la stringa vuota che ha un'unica permutazione
            if(parola.length() == 0)
            {
                  permutazioni.add(parola);
                  return permutazioni;
            }
            /*si riduce il caso a uno più semplice e si applica la ricorsione*/
            char lettera = parola.charAt(0);
```

```
            PermutationGeneratorDue piùCorta = new
                  PermutationGeneratorDue(parola.substring(1));
            ArrayList<String> permutazioniPiùCorta = piùCorta.permuta();
            /*si inserisce la lettera in tutte le posizioni posibili di ogni
            elemento di permutazioniPiùCorta*/
            for(String a : permutazioniPiùCorta)
            {
                  for(int i = 0; i <= a.length(); i++)
                        permutazioni.add(a.substring(0,i) + lettera +
                              a.substring(i));
            }
            return permutazioni;
      }
}
```

PermutationGeneratorDueTester.java

```
import java.util.Scanner;
import java.util.ArrayList;

public class PermutationGeneratorDueTester {

      public static void main(String[] args)
      {
            Scanner in = new Scanner(System.in);
            System.out.println("Dammi una parola da permutare");
            String parola = in.next();
            PermutationGeneratorDue permutanda = new
                  PermutationGeneratorDue(parola);
            ArrayList<String> permutazioni = new ArrayList<String>();
            permutazioni = permutanda.permuta();
            for(String a : permutazioni) System.out.println(a + "\n");
      }
}
```

Esercizio 16.11 – Document.java

Specchietto:

La classe JOptionPane si può utilizzare in sostituzione di System.out.print() o println(). Tale classe va sempre importata all'inizio del documento con un import javax.swing.JOptionPane.
I due principali metodi della classe JOptionPane sono showMessageDialog() e showInputDialog():
- showMessageDialog() si usa quando si deve far comparire un messaggio non interattivo (che non richiede dati in ingresso); la forma da utilizzare all'interno delle parentesi è la seguente: showMessageDialog(null, "ciò che si vuole scrivere tra virgolette o un metodo che non sia void").
- showInputDialog() si usa con i messaggi interattivi (che richiedono inserimenti di dati); si fornisce come parametro la stringa che chiede all'utente il dato in ingresso. Il metodo restituisce un oggetto di tipo String (ecco perchè se si vogliono numeri o altri tipi è sempre necessario convertire il tipo String in quello desiderato).

Testo:

Scrivere un programma che calcoli il numero di spazi bianchi presenti all'interno di un testo.

Consigli:

Il metodo ricorsivo qui proposto risulta di facile comprensione: si crea una sottostringa di lunghezza 1; se essa equivale ad uno spazio bianco, viene incrementato il contatore di spazi vuoti.

```
public class Document
{
```

```
	private String mystring;

	/*crea un oggetto della classe Document
	@param aMystring la stringa contenuta nell'oggetto*/
	public Document(String aMystring)
	{
		mystring = aMystring;
	}

	/*conta gli spazi bianchi presenti in una stringa
	@return il numero di spazi trovati*/
	public int space_count()
	{
		int cont = 0;
		if (mystring.length() == 0)
			return cont;
		if (mystring.substring(0,1).equals(" "))
			cont ++;
		Document mystringshorter = new Document(mystring.substring(1));
		return cont + mystringshorter.space_count();
	}
}
```

DocumentTester.java

```
import javax.swing.*;
public class DocumentTester
{
	public static void main(String[] args)
	{
		String a = JOptionPane.showInputDialog("inserisci una stringa");
		Document stringa = new Document(a);
		JOptionPane.showMessageDialog(null, stringa.space_count());
	}
}
```

Esercizio 16.12 – **InvertiArray.java**

Testo:

Scrivere un programma che, dato in ingresso un'array, sia in grado di invertirlo.

Consigli:

La principale difficoltà dell'esercizio può esser riscontrata nel metodo rovescia(int left, int right): si definisce una variabile locale int tmp che serve come supporto allo scambio dei valori delle posizioni left e right dell'array. Si prosegue allo stesso modo incrementando la prima posizione e decrementando l'ultima, fino al completamento dell'array.

InvertiArray.java

```
public class InvertiArray
{
	private int[] array;

	/*costruisce un oggetto della classe InvertiArray
	@param anArray l'array contenuto nell'oggetto*/
	public InvertiArray(int[] anArray)
	{
		array = anArray;
	}

	/*metodo che serve a invocare il metodo ausiliario rovescia(left, right)*/
```

```
	public void rovescia()
	{
		rovescia(0, array.length-1);
	}

	/*metodo ausiliario che rovescia il contenuto di un array
	@param left indice dell'elemento sinistro da scambiare
	@param right indice dell'elemento destro da scambiare*/
	public void rovescia (int left, int right)
	{
		if(left>=right)
			return;
		int tmp = array[left];
		array[left] = array[right];
		array[right] = tmp;
		rovescia(left + 1, right - 1);
	}

	/*restituisce l'array contenuto nell'oggetto
	@return l'array dell'oggetto*/
	public int[] getArray()
	{
		return array;
	}

	/*restituisce il contenuto dell'array in formato stringa
	@return una stringa con tutti gli elementi dell'array*/
	public String toString()
	{
		String s = "";
		for(int i = 0; i < array.length; i++)
			s = s + array[i] + "  ";
		return s;
	}
}
```

InvertyArrayTester.java

```
import javax.swing.*;

public class InvertiArrayTester
{
	public static void main(String[] args)
	{
		String lunghezza = JOptionPane.showInputDialog("Inserisci la
			lunghezza dell'array da invertire");
		int n = Integer.parseInt(lunghezza);
		int[] mioVettore = new int[n];
		for(int i = 0; i < n; i++)
		{
			String valore = JOptionPane.showInputDialog("inserisci un
				valore");
			int numero = Integer.parseInt(valore);
			mioVettore[i] = numero;
		}
		InvertiArray rovesciabile = new InvertiArray(mioVettore);
		rovesciabile.rovescia();
		JOptionPane.showMessageDialog(null, rovesciabile.toString());
		System.exit(0);
	}
}
```

Esercizio 16.13 – SommaAlterniIndex.java

Testo:
Scrivere un programma che calcoli la somma degli elementi di un array alternativamente. Per esempio, trova il totale di tutti i numeri contenuti nella posizione pari, e la sottrae a quella delle posizioni dispari.

Consigli:
Viene creato un metodo ausiliario che tiene conto dell'indice di posizione dell'elemento dell'array: se l'indice è pari somma il suo contenuto al risultato fornito dal metodo ricorsivo per il caso inferiore, se dispari lo sottrae (0 è considerato pari per praticità).

SommaAlterniIndex.java

```
public class SommaAlterniIndex
{
	int[] myArray;

	public SommaAlterniIndex(int[] unVettore)
	{
		myArray = unVettore;
	}

	public int somma()
	{
		return somma(0);
	}

	public int somma(int i)
	{
		if(i >= myArray.length) return 0;
		if(i % 2 == 0) return myArray[i] + somma(i + 1);
		else return -1 * myArray[i] + somma(i + 1);
	}
}
```

SommaAlterniIndexTester.java

```
import java.util.Scanner;

public class SommaAlterniIndexTester
{
	public static void main(String[] args)
	{
		Scanner in = new Scanner(System.in);
		System.out.println("Quanti numeri vuoi inserire?");
		int n = in.nextInt();
		int[] unVettore = new int[n];
		for(int i = 0; i < n; i++)
		{
			System.out.println("Dammi il numero da inserire");
			int a = in.nextInt();
			unVettore[i] = a;
		}
		SommaAlterniIndex alternatore = new SommaAlterniIndex(unVettore);
		System.out.println("la somma a segni alternati dei numeri inseriti
			è: " + alternatore.somma());
	}
}
```

Esercizio 16.14 – SommaAlterniBoolean.java
Testo:
Vedi esercizio 16.13

Consigli:
Viene creato un metodo ausiliario che attraverso un parametro boolean sa se l'elemento si trova in una posizione pari o dispari.

SommaAlterniBoolean.java

```
public class SommaAlterniBoolean
{
	int[] myArray;

	public SommaAlterniBoolean(int[] unVettore)
	{
		myArray = unVettore;
	}

	public int somma()
	{
		return somma(false);
	}

	public int somma(Boolean posizionePari)
	{
		/*caso base*/
		if(myArray.length == 0) return 0;
		/*costruisco un oggetto più corto da passare in ricorsione*/
		int[] provvisorio = new int[myArray.length - 1];
		System.arraycopy(myArray, 1, provvisorio, 0, myArray.length - 1);
		SommaAlterniBoolean shorter = new SommaAlterniBoolean(provvisorio);
		/*applico un metodo ricorsivo*/
		if(posizionePari) return myArray[0] + shorter.somma(false);
		else return -1 * myArray[0] + shorter.somma(true);
	}
}
```

SommaAlterniBooleanTester.java

```
import java.util.Scanner;

public class SommaAlterniBooleanTester
{
	public static void main(String[] args)
	{
		Scanner in = new Scanner(System.in);
		System.out.println("Quanti numeri vuoi inserire?");
		int n = in.nextInt();
		int[] unVettore = new int[n];
		for(int i = 0; i < n; i++)
		{
			System.out.println("Dammi il numero da inserire");
			int a = in.nextInt();
			unVettore[i] = a;
		}
		SommaAlterniBoolean alternatore = new
			SommaAlterniBoolean(unVettore);
		System.out.println("la somma a segni alternati dei numeri inseriti
			è: " + alternatore.somma());
	}
}
```

Esercizio 16.15 – Fattoriale.java
Specchietto:
Il tipo long ha un intervallo che va da -9223372036854775808 (19 cifre) a +9223372036854775807 (19 cifre). Nel calcolo del fattoriale si riesce ad

arrivare a 31! che ha esattamente 19 cifre. Il numero 32! ha venti cifre significative e supera l'intervallo del tipo long: per questo motivo nell'esercizio si riesce a calcolare il fattoriale di numeri minori di 32.

Testo:
Scrivere un programma che calcoli ricorsivamente il fattoriale di un numero.

Consigli:
La maggior difficoltà può esser rilevata nella comprensione del concetto di fattoriale: si indica con n! ed è il prodotto dei primi *n* numeri interi positivi, come ad esempio:
Dato in ingresso: 5
Fattoriale: 5*4*3*2*1 = 120
Per calcolare il fattoriale, quindi, è sufficiente diminuire costantemente di 1 il valore dato in parametro.

```
public class Fattoriale
{
	private int n;

	public Fattoriale(int aN)
	{
		n = aN;
	}

	public long getFatt()
	{
		// casi base
		if (n == 0) return 1;
		if (n == 1) return 1;
		// si crea un nuovo oggetto Fattoriale
		Fattoriale fat = new Fattoriale(n-1);
		// si invoca il metodo getFatt() sul nuovo oggetto
		return n * fat.getFatt();
	}
}
```

FattorialeTester.java

```
import javax.swing.JOptionPane;
public class FattorialeTester
{
	public static void main(String[] args)
	{
		String n = JOptionPane.showInputDialog("Inserisci il numero: ");
		int num = Integer.parseInt(n);
		Fattoriale risultato = new Fattoriale(num);
		JOptionPane.showMessageDialog(null,"Il fattoriale di "+num+" e'
		"+risultato.getFatt());
	}
}
```

ESERCIZI STILE ESAME[5]

Gestione di un'assicurazione

Specchietto:

L'enunciato switch è una forma abbreviata in sostituzione di una serie di if/else. Si può utilizzare solo con numeri interi di tipo int
- Viene esaminato il primo "case", se è vero ne esegue l'enunciato altrimenti esamina il "case" successivo
- E' necessario il "break" alla fine di ogni enunciato, altrimenti il compilatore esamina anche il "case" successivo.

Testo:

Un'assicurazione desidera creare un archivio elettronico in grado di raccogliere informazioni sulle automobili e sui loro proprietari. Si implementi una classe Cliente, avente il nominativo (stringa) come variabile d'istanza; una classe Automobile avente come variabili d'istanza il numero di targa della vettura (intero) e un riferimento al proprietario della classe Cliente. Si implementi, infine, la classe Archivio.

Consigli:

Le indicazioni sono molto simili a quelle della classe AziendaSanitaria.
I problemi maggiori possono essere riscontrati nella classe Archivio: si consiglia l'uso di metodi ausiliari. Essi non sono richiesti dal testo, ma sono utili per accorciare il codice nei metodi successivi. Altra difficoltà può essere riscontrata in tutti i metodi della classe che richiedono, come parametro implicito, il nome del cliente di tipo Stringa: il suggerimento è seguire il procedimento già enunciato nella classe AziendaSanitaria.

Cliente.java

```
public class Cliente
{
	private String nome;
	public Cliente(String unNome)
	{
		nome=unNome;
	}
	public String getNome()
	{
		return nome;
	}
}
```

Automobile.java

```
public class Automobile
{
	private int targa;
	private Cliente proprietario;
	public Automobile (int unaTarga, Cliente unCliente)
	{
		targa = unaTarga;
		proprietario = unCliente;
	}
	public int getTarga()
	{
		return targa;
	}
```

[5] Altri simili esercizi e commenti li trovate al link http://forum.dei.unipd.it/ sotto la sezione Fond. di Informatica JAVA (VICENZA)

```
	public Cliente getProprietario()
	{
		return proprietario;
	}
}
```

Archivio.java

```
import java.util.ArrayList;

public class Archivio
{
	private ArrayList<Cliente> clienti;
	private ArrayList<Automobile> automobili;

	public Archivio ()
	{
		clienti = new ArrayList<Cliente>();
		automobili = new ArrayList<Automobile>();
	}

	// restituisce, se esiste, l'indice del cliente nell'arraylist clienti
	private int indiceCliente (String unNome)
	{
		for (int i = 0; i < clienti.size(); i++)
			if (clienti.get(i).getNome().equalsIgnoreCase(unNome))
				return i;
		return -1;
	}

	// restituisce, se esiste, l'indice della targa nell'arraylist automobili
	private int indiceTarga (int unaTarga)
	{
		for (int i = 0; i<automobili.size(); i++)
			if (automobili.get(i).getTarga() == unaTarga)
				return i;
		return -1;
	}

	// aggiunge un nuovo cliente all'arraylist clienti
	public void addCliente(String unNome)
	{
		if(indiceCliente(unNome) == -1)
			clienti.add(new Cliente(unNome));
	}

	// aggiunge una nuova auto all'arraylist automobile
	public void addAutomobile (int unaTarga, String unNome)
	{
		if(indiceTarga(unaTarga) != -1) return;
		addCliente(unNome);
		int index = indiceCliente(unNome);
		automobili.add(new Automobile(unaTarga,clienti.get(index)));
	}

	/* restituisce le targhe delle automobili aventi unNome come
	proprietario*/
	public ArrayList<Integer> returnTarga (String unNome)
	{
		ArrayList<Integer> temp = new ArrayList<Integer>();
		for (int i = 0; i < automobili.size(); i++)
			if(automobili.get(i).getProprietario().getNome().
				equalsIgnoreCase(unNome))
```

```
                  temp.add(automobili.get(i).getTarga());
      return temp;
}

// restituisce il nome del proprietario di una targa
public String clienteTarga (int unaTarga)
{
      for (Automobile auto: automobili)
            if ((auto.getTarga()) == unaTarga)
                  return auto.getProprietario().getNome();
      return null;
}

/* data una targa di un automobile, se presente, la rimuove
dall'arraylist*/
public void removeAutomobile (int unaTarga)
{
      if(indiceTarga(unaTarga) != -1)
            automobili.remove(indiceTarga(unaTarga));
}

/* toglie unNome dall'arraylist clienti e le sue eventuali auto da
automobili*/
public void removeCliente (String unNome)
{
      if(indiceCliente(unNome) == -1)
            return;
      boolean hoFinito = false;
      while(!hoFinito)
      {
            hoFinito = true;
            for(int i = 0; i < automobili.size(); i++)
                  if(automobili.get(i).getProprietario().getNome().
                        equalsIgnoreCase(unNome))
                  {
                        automobili.remove(i);
                        hoFinito = false;
                  }
      }
      clienti.remove(indiceCliente(unNome));
}

// conta quante automobili ha un cliente
public int contaAuto(String unNome)
{
      int temp = 0;
      for (Automobile auto: automobili)
            if (auto.getProprietario().getNome().equalsIgnoreCase(unNome))
                  temp++;
      return temp;
}

// ritorna un array con il numero di auto possedute da ogni cliente
private int[] listaAutoClienti()
{
      int[] temp = new int[clienti.size()];
      int i=0;
      for (Cliente c : clienti)
      {
            temp[i] = contaAuto(c.getNome());
            i++;
      }
      return temp;
```

```
	}

	// ritorna il cliente con il maggior numero di auto
	public Cliente maxAuto()
	{
		int indexMax = 0;
		int max = 0;
		int[] temp = listaAutoClienti();
		for (int i = 0; i < clienti.size(); i++)
			if (temp[i] > max)
			{
				indexMax = i;
				max = temp[i];
			}
		return clienti.get(indexMax);
	}
}
```

Assicurazione.java

```
import javax.swing.JOptionPane;
import java.util.ArrayList;

public class Assicurazione
{
	public static void main(String args[])
	{
		Archivio archivio = new Archivio();
		String scelta,nome,targa;
		int numScelta,numTarga;
		String continua = "s";
		while (continua.equalsIgnoreCase("s"))
		{
			scelta = JOptionPane.showInputDialog("Operazioni eseguibili:\n
				1) Aggiungi Cliente\n
				2) Aggiungi Automobile\n
				3) Ricerca Proprietario\n
				4) Ricerca automobili di un cliente\n
				5) Cliente con più automobili\n
				6) Cancella Cliente\n
				7) Cancella Automobile");
			numScelta = Integer.parseInt(scelta);
			switch(numScelta)
			{
			case 1:nome = JOptionPane.showInputDialog("Inserisci nuovo
				cliente:");
				archivio.addCliente(nome);break;
			case 2:targa = JOptionPane.showInputDialog("Inserisci nuova
				targa:");
				numTarga = Integer.parseInt(targa);
				nome = JOptionPane.showInputDialog("Inserisci nome
				cliente:");
				archivio.addAutomobile(numTarga, nome);break;
			case 3:targa = JOptionPane.showInputDialog("Inserisci numero
				targa:");
				numTarga = Integer.parseInt(targa);
				JOptionPane.showMessageDialog(null,"Targa " + numTarga +
				" appartiene al cliente: " +
				archivio.clienteTarga(numTarga));break;
			case 4:nome = JOptionPane.showInputDialog("Inserisci nome
				cliente:");
				ArrayList<Integer> targhe = new ArrayList();
				targhe = archivio.returnTarga(nome);
				String targheCliente = "";
```

```
                        for (Integer a : targhe)
                              targheCliente = targheCliente + "\n" + a;
                        JOptionPane.showMessageDialog(null,targheCliente);break;
                  case 5:JOptionPane.showMessageDialog(null,"Il cliente con più
                        auto è: " + archivio.maxAuto().getNome());break;
                  case 6:nome = JOptionPane.showInputDialog("Inserisci nome
                        cliente:");
                        archivio.removeCliente(nome);break;
                  case 7:targa = JOptionPane.showInputDialog("Inserisci numero
                        targa:");
                        numTarga = Integer.parseInt(targa);
                        archivio.removeAutomobile(numTarga);break;
                  }
                  continua = JOptionPane.showInputDialog("Vuoi effettuare nuove
                        operazioni? (s/n)");
            }
      }
}
```

Gestione di una biblioteca

Testo:

Una biblioteca ha dato l'incarico di implementare un software per la gestione dei prestiti di libri. Si costruiscano le seguenti classi:

- Data: serve per costruire "l'orologio" virtuale. Implementare tre variabili d'istanza e i metodi: int getDay(), int getMonth(), int getYear(), void setDay(int aDay), void setMonth(int aMonth), void setYear(int aYear), void nextDay(), Boolean isEqual(Data date).
- Abbonato: costruisce l'oggetto utente. Essa ha due variabili d'istanza e i seguenti metodi: String getCognome(), String getNome()
- Libro: costruisce l'oggetto libro. Ha tre variabili d'istanza: titolo, utente a cui è prestato, data scadenza del prestito. Ha inoltre i relativi metodi accessori.
- Archivio: essa implementa due arrayList, una contenente gli abbonati alla biblioteca e l'altra i libri presenti, oltre ad un riferimento alla data, e i seguenti metodi: void nuovoLibro(String unTitolo), void nuovoUtente(String unNome, String unCognome), int trovaUtente(String unNome, String unCognome), Data scadenza(Data date), int presta(Libro unLibro, Abbonato unUtente), void aggiorna() [si riferisce all'aggiornamento dell'ora], int numLibri(Abbonato anAbb) [restituisce il numero di libri posseduti da un dato utente]

Consigli:

Le classi Data, Abbonato e Libro non presentano particolari difficoltà d'esecuzione. La classe Archivio, invece, può risultare di difficile implementazione, in particolare:

- presta(Libro unLibro, Abbonato unUtente): si passa in rassegna l'arraylist contenente i libri, se una copia del libro cercato esiste e non è in prestito, viene assegnata una data di scadenza per la restituzione e l'utente che l'ha preso in prestito;
- aggiorna(): aggiorna la data, se un libro deve essere restituito nel medesimo giorno, allora si resettano i dati sull'utente possessore.
- numLibri(Abbonato anAbb): restituisce quanti libri possiede un dato utente

Data.java

```
public class Data
{
      private int d, m, y;
      public Data()
      {
            d = 0;
            m = 0;
```

```
            y = 0;
      }

      public int getDay()
      {
            return d;
      }

      public int getMonth()
      {
            return m;
      }

      public int getYear()
      {
            return y;
      }

      public void setDay(int aDay)
      {
            y = aDay;
      }

      public void setMonth(int aMonth)
      {
            y = aMonth;
      }

      public void setYear(int anYear)
      {
            y = anYear;
      }

      public void nextDay()
      {
            if (d < 30) d++;
            else if (m < 12)
            {
                  d = 1;
                  m++;
            }
            else
            {
                  d = 1;
                  m = 1;
                  y++;
            }
      }

      public boolean isEqual(Data date)
      {
           if(date.getDay() == d && date.getMonth() == m && date.getYear() == y)
                  return true;
            else
                  return false;
      }
}
```

Abbonato.java

```
public class Abbonato
{
      private String cognome, nome;
```

```
	public Abbonato(String unNome, String unCognome)
	{
		nome = unNome;
		cognome = unCognome;
	}

	public String getCognome()
	{
		return cognome;
	}

	public String getNome()
	{
		return nome;
	}
}
```

Libro.java

```
public class Libro
{
	private String titolo;
	private Abbonato utente;
	private Data scadenza;

	public Libro(String unTitolo)
	{
		titolo = unTitolo;
		utente = null;
		scadenza = null;
	}

	public String getTitolo()
	{
		return titolo;
	}

	public Abbonato getUtente()
	{
		return utente;
	}

	public Data getScadenza()
	{
		return scadenza;
	}

	public void setUtente(Abbonato unUtente)
	{
		utente = unUtente;
	}

	public void setScadenza(Data unaData)
	{
		scadenza = unaData;
	}
}
```

Biblioteca.java

```
import java.util.ArrayList;
```

```
public class Biblioteca
{
	private ArrayList<Abbonato> utenti;
	private ArrayList<Libro> libri;
	private Data dataCorr;

	public Biblioteca(Data unaData)
	{
		utenti = new ArrayList<Abbonato>();
		libri = new ArrayList<Libro>();
		dataCorr = unaData;
	}

	public void nuovoLibro(String unTitolo)
	{
		Libro l = new Libro(unTitolo);
		libri.add(l);
	}

	public void nuovoUtente(String unNome, String unCognome)
	{
		if(trovaUtente(unNome, unCognome) != -1)
			return;
		Abbonato nuovoAbb = new Abbonato(unNome, unCognome);
		utenti.add(nuovoAbb);
	}

	private int trovaUtente(String unNome, String unCognome)
	{
		for(int i = 0; i < utenti.size(); i++)
			if(utenti.get(i).getNome().equalsIgnoreCase(unNome) &&
				utenti.get(i).getCognome().equalsIgnoreCase(unCognome))
				return i;
		return -1;
	}

	private Data scadenza(Data date)
	{
		Data d = new Data();
		d.setDay(date.getDay());
		if (date.getMonth() > 10)
		{
			d.setMonth(date.getMonth() - 10);
			d.setYear(date.getYear() + 1);
		}
		else
		{
			d.setMonth(date.getMonth() + 2);
			d.setYear(date.getYear());
		}
		return d;
	}

	public int presta(Libro unLibro, Abbonato unUtente)
	{
		if(trovaUtente(unUtente.getNome(), unUtente.getCognome()) == -1)
			return -1;
		for (Libro lib : libri)
			if (lib.getTitolo().equalsIgnoreCase(unLibro.getTitolo()) &&
				lib.getUtente() == null)
			{
				lib.setScadenza(this.scadenza(dataCorr));
```

```
                        /*notare che: abbiamo verificato la presenza dell'utente
                        nell'ArrayList utenti (vedi inizio metodo), ora il
                        metodo setUtente riceve come parametro l'elemento
                        dell'ArrayList utenti e non la variabile parametro
                        unUtente*/
                        lib.setUtente(utenti.get(trovaUtente(unUtente.getNome(),
                              unUtente.getCognome())));
                        return 0;
                  }
            return -1;
      }

      public void aggiorna()
      {
            dataCorr.nextDay();
            for (Libro lib : libri)
                  if (lib.getScadenza().isEqual(dataCorr))
                  {
                        lib.setUtente(null);
                        lib.setScadenza(null);
                  }
      }

      public int numLibri(Abbonato unUtente)
      {
         int nLib = 0;
         for (Libro a: libri)
            if (a.getUtente() != null &&
            a.getUtente().getNome().equalsIgnoreCase(unUtente.getNome()) &&
            a.getUtente().getCognome().equalsIgnoreCase(unUtente.getCognome()))
                  nLib = nLib + 1;
         return nLib;
      }
}
```

Gestione di un campo da tennis

Testo:

Fare una classe Prenotazione (di un campo da tennis) contenente il nome del cliente e l'ora della sua prenotazione. Implementare una classe Campo in cui ci sono i seguenti metodi: public boolean addPren(int inizio, int fine, String unNomeCliente), per prenotare il campo (controlla se i dati inseriti sono giusti e se il campo è disponibile dopodichè salva la prenotazione e restituisce true se il campo è stato prenotato); public boolean removePren (int inizio, int fine, String unNomeCliente), per cancellare una prenotazione (controlla se il campo è stato prenotato dal cliente che vuole cancellare la prenotazione dopodichè la cancella e restituisce true se la prenotazione è stata cancellata); public String toString(); public double utilizzo(), per trovare la percentuale dell'utilizzo del campo.

Consigli:

Nello sviluppo di alcuni metodi, si consiglia il seguente procedimento:

- addPren (int inizio, int fine, String unNomeCliente): creare un metodo ausiliario che individua se il campo da gioco è disponibile.
- removePren (int inizio, int fine, String unNomeCliente): anche in questo caso s'invita ad utilizzare un metodo ausiliario per verificare se esiste la prenotazione da disdire.

Prenotazione.java

```
public class Prenotazione
{
      private int ora;
```

```
	private String nomeCliente;

	public Prenotazione (int o, String c)
	{
		ora=o;
		nomeCliente=c;
	}

	public int getOra()
	{
		return ora;
	}

	public String getCliente()
	{
		return nomeCliente;
	}

	public String toString()
	{
		return "Ore: " + ora + " campo prenotato dal Sig. " + nomeCliente;
	}
}
```

Campo.java

```
public class Campo
{
	public final int APERTURA = 9;
	public final int CHIUSURA = 21;
	// contiene gli oggetti della classe Prenotazione
	private Prenotazione[] prenotazioni;

	public Campo()
	{
		prenotazioni = new Prenotazione[CHIUSURA-APERTURA];
	}

	/* Metodo per prenotare il campo.
	Controlla se i dati inseriti sono giusti e se il campo è disponibile,
	salvando la prenotazione
	@return true se la prenotazione è andata a buon fine*/
	public boolean addPren(int inizio, int fine, String unNomeCliente)
	{
		if (fine - inizio <= 0 || inizio < APERTURA || fine > CHIUSURA)
			return false;
		if (isDisp(inizio,fine))
		{
			for (int i = inizio - APERTURA; i < fine - APERTURA; i++)
				prenotazioni[i] = new Prenotazione(APERTURA + i,
					unNomeCliente);
			return true;
		}
		else
			return false;
	}

	/*metodo ausiliario per controllare se il campo è disponibile in un dato
	intervallo*/
	private boolean isDisp (int inizio, int fine)
	{
		for (int i = inizio - APERTURA; i < fine - APERTURA; i++)
			if (prenotazioni[i] != null)
```

```
                        return false;
                return true;
        }

        /*metodo per cancellare una prenotazione.
        Controlla se il campo è stato prenotato dal cliente che vuole cancellare
        la prenotazione dopodichè la cancella.
        @return true se la cancellazione è andata a buon fine*/
        public boolean removePren (int inizio, int fine, String unNomeCliente)
        {
                if (fine - inizio <= 0 || inizio < APERTURA || fine > CHIUSURA)
                        return false;
                if (!isPren(inizio, fine, unNomeCliente))
                        return false;
                for (int i = inizio - APERTURA; i < fine - APERTURA; i++)
                        prenotazioni[i] = null;
                return true;
        }

        /*metodo ausiliario per controllare se il campo è prenotato in un dato
        intervallo da un cliente (unNomeCliente)*/
        private boolean isPren (int inizio, int fine, String unNomeCliente)
        {
                for (int i = inizio - APERTURA; i < fine - APERTURA; i++)
                        if (prenotazioni[i] == null || !prenotazioni[i].getCliente().
                                equalsIgnoreCase(unNomeCliente))
                                return false;
                return true;
        }

        public String toString()
        {
                String temp="";
                for (int i = 0; i < prenotazioni.length; i++)
                {
                        if (prenotazioni[i] == null)
                                temp = temp + "Ore: " + (i + APERTURA) + " campo libero"
                                        + "\n";
                        if (prenotazioni[i] != null)
                                temp = temp + prenotazioni[i].toString() + "\n";
                }
                return temp;
        }

        // metodo per trovare la percentuale dell'utilizzo del campo
        public double utilizzo()
        {
                int conta = 0;
                for (int i = 0; i < CHIUSURA - APERTURA; i++)
                        if (prenotazioni[i] != null) conta++;
                /*nella riga seguente si usa un trucco per ottenere un double con
                sole due cifre decimali*/
                return conta * 10000 / (CHIUSURA - APERTURA) / 100.0;
        }
}
```

CampoTester.java

```
import javax.swing.JOptionPane;
public class CampoTester
{
   public static void main(String args[])
   {
```

```
	int i = 0;
	Campo campo1 = new Campo();

	while(i != 5)
	{
		String scegli = JOptionPane.showInputDialog("---CAMPO DA TENNIS---\n
		Operazioni eseguibili:\n 1)Prenota il campo\n 2)Disdici il campo\n
		3)Lista prenotazioni\n 4)Stato utilizzo campo\n 5)Esci");
		i=Integer.parseInt(scegli);
		switch(i)
		{
		case 1:
		{
			String in = JOptionPane.showInputDialog("Ora inizio:");
			int inizio= Integer.parseInt(in);
			String fin = JOptionPane.showInputDialog("Ora fine:");
			int fine= Integer.parseInt(fin);
			String cli = JOptionPane.showInputDialog("Nome cliente:");
			 if (campo1.addPren(inizio,fine,cli))
			JOptionPane.showMessageDialog(null,"Prenotazione
			effettuata!");
			else
			JOptionPane.showMessageDialog(null,"Prenotazione non
			effettuata!");
			break;
		}
		case 2:
		{
			String in = JOptionPane.showInputDialog("Ora inizio:");
			int inizio= Integer.parseInt(in);
			String fin = JOptionPane.showInputDialog("Ora fine:");
			int fine= Integer.parseInt(fin);
			String cli = JOptionPane.showInputDialog("Nome cliente:");
			if (campo1.removePren(inizio,fine,cli))
			JOptionPane.showMessageDialog(null,"Prenotazione
			cancellata!");
			else
			JOptionPane.showMessageDialog(null,"Prenotazione non
			cancellata!");
			break;
		}
		case 3:
		{
			if(campo1.toString().equals(""))
			JOptionPane.showMessageDialog(null,"Non ci sono
			prenotazioni.");
			else
			JOptionPane.showMessageDialog(null,campo1.toString());
			break;
		}
		case 4:
		{
			JOptionPane.showMessageDialog(null "Il campo è utilizzato al: "
			+ campo1.utilizzo() + " %.");
			break;
		}
		}
	}
	}
}
```

Corso di laurea

Testo:

La segreteria di un corso di laurea desidera sviluppare un archivio elettronico per la gestione di tutti gli studenti, di tutti gli appelli d'esame, e delle sessioni di laurea. Implementare le seguenti classi con i relativi metodi prescritti. Attenzione: non esistono esami diversi con lo stesso nome. Inoltre, potreste avere bisogno di metodi ausiliari che non ho indicato, per semplificare il codice: definite tali metodi ogniqualvolta lo ritenete opportuno.

```
public class Esame {
private String nome;
private int crediti;
private int voto;

public Esame(String unNomeEsame, int numCrediti){...}
public String getNome(){...}
public String getCrediti(){...}
public String getVoto(){...}
public void setVoto(int unVoto){...}
public String toString(){...} }

public class Studente {
private String nome;
private int matricola;
private ArrayList<Esame> esamiSostenuti;
private ArrayList<Esame> pianoStudio;
public Studente(String unNomeStudente, int unaMatricola){...}
public String getNome(){...}
public String getMatricola(){...}
public void aggiungiEsame(String unNomeEsame, int numCrediti){...}
public void promosso(String unNomeEsame, int unVoto){...}
public void cancella(String unNomeEsame){...}
public double media(){...}
public double crediti(){...}
public String toString(){...} }

public class CorsoLaurea {
private String nome;
private ArrayList<Studente> immatricolati;
public static int prossimaMatricola;
public static final int MIN_CREDITI;
public CorsoLaurea(String unNomeStudente){...}
public String getNome(){...}
public void iscrivi(String unNomeStudente){...}
public void ritira(String unNomeStudente){...}
public void appello(String unNomeEsame, ArrayList<Studente> iscritti){...}
public void laurea(){...} }
```

Consigli:
Il programma può esser considerato di media difficoltà. Per facilitarne la comprensione, verrà analizzata ogni singola classe:

- Classe Esame: risulta di semplice implementazione
- Classe Studente: tra le variabili d'istanza, sono implementate due arrayList di oggetti Esame, uno contenente il piano di studi dello studente, cioè il numero di esami che uno studente deve sostenere per conseguire la laurea, e l'altra EsamiSostenuti, cioè test il cui voto è superiore a 18. Si crea un metodo ausiliario cercaIndiceEsamePiano / cercaIndiceEsamiSostenuti(String unNomeEsame), che verificano la presenza di un dato esame nelle arrayList: esso sarà necessario per la costruzione del metodo aggiungiEsame(String unNomeEsame). Altra difficoltà può esser riscontrata nel metodo promosso(String unNomeEsame, int unVoto): se l'esito della ricerca di un dato esame è stato positivo, si aggiunge un nuovo oggetto nell'arrayList EsamiSostenuti, fissando un voto, ed eliminando il corrispondente dall'arrayList PianoStudi. Quest'ultimo

algoritmo, comunque, verrà ripreso nel metodo cancella(String unNomeEsame).

- Classe Archivio: si ricorda che è presente una variabile d'istanza prossimaMatricola, in quale assegna un numero ad ogni studente, e che essa deve essere incrementata o decrementata in base alle necessità (aggiungi o rimuovi un nuovo studente). Il metodo più complicato è, sicuramente, appello(String unNomeEsame, ArrayList<Studente> iscritti), in cui è necessario creare un oggetto di classe Random, che assegna un voto casuale ad ogni studente (ciò, ovviamente, non succede nella realtà…). Gli studenti che totalizzano un punteggio superiore a 18, sono promossi, altrimenti vengono rimandati.

Esame.java

```
public class Esame
{
	private String nomeEsame;
	private int crediti;
	private int voto;
	// un esame non ancora sostenuto ha voto fittizio 0
	public Esame(String unNomeEsame , int numCrediti)
	{
		nomeEsame = unNomeEsame;
		crediti = numCrediti;
		voto = 0;
	}
	public String getNomeEsame()
	{
		return nomeEsame;
	}
	public int getCrediti()
	{
		return crediti;
	}
	public int getVoto()
	{
		return voto;
	}
	public void setVoto(int unVoto)
	{
		voto = unVoto;
	}
	public String toString()
	{
		String r = "";
			r = r + "Nome esame: " + getNomeEsame() + "\n";
				r = r + "Crediti: " + getCrediti() + "\n";
					r = r + "Voto: " + getVoto() + "\n";
		return r;
	}
}
```

EsameTester.java

```
import java.util.Scanner;
public class EsameTester
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);
		System.out.println("Inserisci il nome di un esame");
		String nome = input.next();
```

```
            System.out.println("Inserisci il numero di crediti dell'esame");
            int crediti = input.nextInt();
            Esame esame1= new Esame(nome,crediti);
            System.out.println("Inserisci ora il voto ottenuto in quell'esame");
            int voto = input.nextInt();
            esame1.setVoto(voto);
            System.out.println(esame1.toString());
      }
}
```

Studente.java

```
import java.util.ArrayList;
public class Studente
{
      private String nome;
      private int matricola;
      private ArrayList<Esame>esamiSostenuti;
      // un esame nel piano di studio ha voto fittizio 0
      private ArrayList<Esame>pianoStudio;
      public Studente(String unNomeStudente, int unaMatricola)
      {
            pianoStudio = new ArrayList<Esame>();
            esamiSostenuti = new ArrayList<Esame>();
            nome = unNomeStudente;
            matricola = unaMatricola;
      }
      public String getNome()
      {
            return nome;
      }
      public int getMatricola()
      {
            return matricola;
      }
      private boolean cercaEsame(String unNomeEsame)
      {
            boolean flag = false;
            if (pianoStudio.size()==0)
                  return flag = false;
            else
            {
                  for (Esame a: pianoStudio)
                  {
                        if (a.getNomeEsame().equalsIgnoreCase(unNomeEsame))
                        return flag = true;
                  }
            }
            return flag;
      }
      // aggiunge l'esame al piano di studi,se non è già presente
      public void aggiungiEsame(String unNomeEsame, int numCrediti)
      {
            String nomeEsame = unNomeEsame;
            int crediti = numCrediti;
            if (!cercaEsame(nomeEsame))
                  pianoStudio.add(new Esame(nomeEsame,crediti));
      }
      /* metodo privato che serve per ottenere in che posizione
      si trova l'esame cercato nell'array list Piano.
      param@ a: indice che indica la posizione dell'esame nell'array piano*/
      private int cercaIndiceEsamePiano(String unNomeEsame)
      {
```

```
	int i=0;
	while (i<pianoStudio.size())
	{
		Esame esame1= pianoStudio.get(i);
		String nEsame= esame1.getNomeEsame();
		if (nEsame.equalsIgnoreCase(unNomeEsame))
		return i;
		i++;
	}
	return i=(-1);
}
/* metodo privato che serve per ottenere in che posizione
si trova l'esame cercato nell'array list EsamiSostenuti.
param@ a: indice che indica la posizione dell'esame nell'array. */
private int cercaIndiceEsameSostenuti(String unNomeEsame)
{
	int i=0;
	while (i<esamiSostenuti.size())
	{
		Esame esame1= esamiSostenuti.get(i);
			String nEsame= esame1.getNomeEsame();
		if (nEsame.equalsIgnoreCase(unNomeEsame))
		return i;
		i++;
	}
	return i=(-1);
}
/* quando uno studente viene promosso l'esame viene tolto dal piano studi
e inserito nell'array list esami sostenuti. */
public void promosso(String unNomeEsame,int unVoto)
{
	String nomeEsame = unNomeEsame;
	int voto = unVoto;
	int indiceEsame= cercaIndiceEsamePiano(nomeEsame);
	if (indiceEsame==-1)
	return;
	else
	{
		Esame esameSostenuto= new
		Esame(nomeEsame,pianoStudio.get(indiceEsame).getCrediti());
		esamiSostenuti.add(esameSostenuto);
		esamiSostenuti.get(esamiSostenuti.size()-1).setVoto(voto);
		pianoStudio.remove(indiceEsame);
	}
}
/* elimina un esame sia se esso è presente nel piano studi che negli esami
sostenuti */
public void cancella(String unNomeEsame)
{
	String nomeEsame = unNomeEsame;
	int indiceEsame1= cercaIndiceEsamePiano(nomeEsame);
	int indiceEsame2= cercaIndiceEsameSostenuti(nomeEsame);
	if (indiceEsame1==-1)
	{
		if (indiceEsame2==-1)
		return;
		else
		esamiSostenuti.remove(indiceEsame2);
	}
	else
	pianoStudio.remove(indiceEsame1);
}
// calcola la media dei voti
```

```
	public double media()
	{
		int somma = 0;
		for(Esame a: esamiSostenuti)
			somma = somma + a.getVoto();
			double media = somma/esamiSostenuti.size();
		return media;
	}
	public int crediti()
	{
		int sommaCrediti = 0;
		for(Esame a: esamiSostenuti)
			sommaCrediti = sommaCrediti + a.getCrediti();
		return sommaCrediti;
	}
	public String toString()
	{
		String s = "";
		s = s + "Studente: " + getNome() + "\n";
		s = s + "Matricola: " + getMatricola() + "\n";
		s = s + "Media voti: " + media() + "\n";
		s = s + "Crediti: " + crediti() + "\n";
		return s;
	}
}
```

StudenteTester.java

```
import java.util.Scanner;
public class StudenteTester
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);
		System.out.println("Inserisci il nome studente");
		String nome = input.next();
		System.out.println("Inserisci il numero di matricola");
		int matricola = input.nextInt();
		Studente stud= new Studente(nome,matricola);
		System.out.println("Inserisci quanti esami dovrà sostenere lo
		studente nel suo piano studi");
		int n = input.nextInt();
		for(int i=0; i<n; i++)
		{
			System.out.println("Inserisci il nome di un esame");
			String nomeEsame = input.next();
			System.out.println("Inserisci il numero di crediti
			dell'esame");
			int crediti = input.nextInt();
			stud.aggiungiEsame(nomeEsame,crediti);
		}
		for(int j=0; j<2; j++)
		{
			System.out.println("Inserisci il nome dell'esame in cui lo
			studente è stato promosso");
			String nomeEsameP = input.next();
			System.out.println("Inserisci ora il voto ottenuto
			all'esame");
			int votoEsameP = input.nextInt();
			stud.promosso(nomeEsameP,votoEsameP);
		}
		System.out.println("Inserisci il nome dell'esame che vuoi eliminare
		da entrambi gli elenchi");
```

```
            String nomeEsameElim = input.next();
            stud.cancella(nomeEsameElim);
            System.out.println(stud.toString());
      }
}
```

CorsoLaurea.java

```
import java.util.ArrayList;
import java.util.Random;
public class CorsoLaurea
{
      private String nome;
      private ArrayList<Studente> immatricolati;
      // prossimo numero matricola non ancora assegnato
      public static int prossimaMatricola;
      // numero crediti richiesto per la laurea
      public static final int MIN_CREDITI=30;

      public CorsoLaurea(String unNomeCorso)
      {
            immatricolati = new ArrayList<Studente>();
            nome = unNomeCorso;
            prossimaMatricola=0;
      }
      public String getNome()
      {
            return nome;
      }
      // numeri di matricola assegnati consecutivamente
      public void iscrivi(String unNomeStudente)
      {
            prossimaMatricola++;
            immatricolati.add(new Studente(nome,prossimaMatricola));
      }
      /* cerca in che posizione del vettore immatricolati si trova lo studente
      e restituisce l'indice. Se lo studente non è presente, restiuisce -1 */
      public int cercaIndiceStudente(String unNomeStudente)
      {
            int i=0;
            while (i<immatricolati.size())
            {
                  Studente studente1= immatricolati.get(i);
                  String nStudente= studente1.getNome();
                  if (nStudente.equalsIgnoreCase(unNomeStudente))
                        return i;
                              i++;
            }
            return i=(-1);
      }
      // ritira studente
      public void ritira(String unNomeStudente)
      {
            String nomeStudente = unNomeStudente;
                  int indiceStudente= cercaIndiceStudente(nomeStudente);
            if (indiceStudente==-1)
                  return;
            else
                  immatricolati.remove(indiceStudente);
      }
      // assegna agli studenti iscritti voti random da 1 a 30
      // ed aggiorna conseguentemente i curricula degli studenti promossi
      public void appello(String unNomeEsame,ArrayList<Studente> iscritti)
```

```
	{
		Random generator = new Random ();
		int i = 0;
		while (i<iscritti.size())
		{
			int rVoto = 1 + generator.nextInt(30);
			if (rVoto>=18)
			iscritti.get(i).promosso(unNomeEsame,rVoto);
			i++;
		}
	}
	// rimuovi tutti gli studenti che hanno totalizzato almeno MIN_CREDITI
	public void laurea()
	{
		int i = 0;
		while (i<immatricolati.size())
		{
			if (immatricolati.get(i).crediti() >= MIN_CREDITI)
				immatricolati.remove(i);
					i++;
		}
	}
}
```

CorsoLaureaTester.java

```
import java.util.ArrayList;
import java.util.Scanner;
public class CorsoLaureaTester
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);
		System.out.println("Inserisci nome corso laurea");
		String nomeCorso = input.next();
		CorsoLaurea corso = new CorsoLaurea(nomeCorso);
		for (int i=0; i<3; i++)
		{
			System.out.println("Inserisci il nome dello studente da
			iscrivere");
			String nomeStud = input.next();
			corso.iscrivi(nomeStud);
		}
		System.out.println("Inserisci il nome dello studente da ritirare");
			String nomeStudRit = input.next();
				corso.ritira(nomeStudRit);
		System.out.println("Inserisci il nome dell'esame che gli studenti
		dovranno sostenere");
		String appEsame = input.next();
		ArrayList<Studente> iscritti = new ArrayList<Studente>();
		for (int j=0; j<3;j++)
		{
			System.out.println("Inserisci il nome dello studente da
			iscrivere all'appello");
			String nomeStudIscr = input.next();
			if (corso.cercaIndiceStudente(nomeStudIscr)==(-1))
				System.out.println("lo studente non è iscritto");
			else
			{
				System.out.println("Inserisci il numero di matricola");
				int matricola = input.nextInt();
				iscritti.add(new Studente(nomeStudIscr,matricola));
			}
```

```
            }
            corso.appello(appEsame,iscritti);
            corso.laurea();
      }
}
```

Archivio MP3

Testo:

Uno studente desidera implementare un archivio elettronico con tutti i brani musicali in formato mp3 contenuti nel suo pc.
L'archivio tiene traccia di tutti i cd, artisti e generi di pertinenza dei brani catalogati. Ad esempio, due brani dello stesso cd avranno come variabile d'istanza album due riferimenti allo stesso cd, contenuto nella variabile d'istanza cds specificata per la classe Archivio.
Allo stesso modo, due cd dello stesso genere avranno come variabile d'istanza genere due riferimenti allo stesso genere, contenuto nella variabile d'istanza generi specificata per la classe Archivio.
Non devono dunque mai essere duplicati i cd, oppure i generi.
Bisogna invece "condividere" questi dati ogni volta che sia possibile.
Quando si deve aggiungere un nuovo brano nell'archivio, vengono specificate tutte le coordinate: artista, cd e genere.
Se viene trovato il cd a cui il brano appartiene, ci si deve riferire a quell'oggetto e non duplicarlo.
Se invece tale oggetto non esiste nell'archivio (ovvero si tratta del primo brano acquisito di questo cd)si deve allora creare anche il nuovo oggetto che rappresenti il cd.
Stesso discorso vale per il genere e l'artista: si deve creare una nuova istanza solamente se non ne esiste alcuna.

Consigli:

Le classi Genere, Artista, Cd e Brano non risultano complicate.
S'analizzi, invece, la classe Archivio: si nota come alcuni metodi, come quelli di ricerca, sono identici dal punto di vista strutturale (cercare un elemento tramite ciclo for, se esso corrisponde alla variabile parametro, allora viene restituita la posizione dell'oggetto cercato), ma sono diverse dal punto di vista dei parametri. Si consiglia di fare attenzione. Le maggiori difficoltà riguardano i seguenti metodi:

- cdToBrani(String unTitoloCd) e artistaToCd(String unNomeArtista): si crea un'arrayList temporanea, si scandisce l'arrayList di arrivo (Brani o Cd) e se la ricerca da esito positivo, si aggiunge l'elemento all'arrayList. Alla fine, il metodo restituirà quel risultato.
- maxCd(): si crea un oggetto nullo di classe Genere e un contatore o valore sentinella. Per ogni elemento dell'arrayList Genere, se il cd è del genere ricercato, allora si incrementa il secondo contatore, creato all'interno del ciclo for. Alla fine si confrontano i valori: il metodo restituisce il genere con il maggior numero di Cd
- crea(int n, String unNomeGenere): è necessario creare un oggetto della classe Random e, successivamente, due arrayList temporanee contenente brani(una di appoggio e la vera playlist). In seguito, si passano i brani selezionati dalla prima alla seconda arrayList.

Genere.java

```
public class Genere
{
      private String nomeGenere;
      public Genere (String unNomeGenere)
      {
            nomeGenere = unNomeGenere;
      }
      public String getNomeGenere ()
```

```
	{
		return nomeGenere;
	}
	public String toString()
	{
		return "Genere: " + nomeGenere;
	}
}
```

Artista.java

```
public class Artista
{
	private String nomeArtista;
	public Artista (String unNomeArtista)
	{
		nomeArtista = unNomeArtista;
	}

	public String getNomeArtista ()
	{
		return nomeArtista;
	}
	public String toString()
	{
		return "Nome artista: " + nomeArtista;
	}
}
```

Cd.java

```
public class Cd
{
	private String titoloCd;
	private Artista artista;
	private Genere genere;
	public Cd (String unTitoloCd, Artista unArtista, Genere unGenere)
	{
		titoloCd = unTitoloCd;
		artista = unArtista;
		genere = unGenere;
	}
	public String getTitoloCd ()
	{
		return titoloCd;
	}
	public Artista getArtista ()
	{
		return artista;
	}
	public Genere getGenere ()
	{
		return genere;
	}
	public String toString()
	{
		return "Titolo Cd: " +titoloCd+ "\n" +artista.toString()+ "\n" +
		genere.toString();
	}
}
```

Brano.java

```
public class Brano
{
	private String titolo;
	private Cd album;
	private double durata;
	public Brano (String unTitoloBrano, Cd unAlbum, double unaDurata)
	{
		titolo = unTitoloBrano;
		album = unAlbum;
		durata = unaDurata;
	}
	public String getTitolo ()
	{
		return titolo;
	}
	public Cd getAlbum ()
	{
		return album;
	}
	public double getDurata ()
	{
		return durata;
	}
	public Genere getGenere ()
	{
		return album.getGenere();
	}
	public Artista getArtista()
	{
		return album.getArtista();
	}
	public String toString()
	{
		return "Brano: " + titolo + "\nAlbum: " + album + "\nDurata: " +
		durata;
	}
}
```

Archivio.java

```
import java.util.*;
public class Archivio
{
	public ArrayList<Genere> generi;
	public ArrayList<Artista> artisti;
	public ArrayList<Cd> cds;
	public ArrayList<Brano> brani;
	public Archivio()
	{
		generi = new ArrayList<Genere>();
		artisti = new ArrayList<Artista>();
		cds = new ArrayList<Cd>();
		brani = new ArrayList<Brano>();
	}
	// restituisce l'indice del genere, se esiste, -1 altrimenti
	public int cercaGenere (String unNomeGenere)
	{
		for (int i = 0; i<generi.size(); i++)
		{
			if(generi.get(i).getNomeGenere().equalsIgnoreCase
			(unNomeGenere))
			return i;
		}
```

```
	return -1;
}
// restituisce l'indice dell'artista, se esiste, -1 altrimenti
public int cercaArtista (String unNomeArtista)
{
	for (int i = 0; i<artisti.size(); i++)
	{
		if(artisti.get(i).getNomeArtista().equals(unNomeArtista))
		return i;
	}
	return -1;
}
// restituisce l'indice del cd, se esiste, -1 altrimenti
public int cercaCd (String unTitoloCd)
{
	for (int i = 0; i<cds.size(); i++)
	{
		if(cds.get(i).getTitoloCd().equals(unTitoloCd))
		return i;
	}
	return -1;
}
// restituisce l'indice del brano, se esiste, -1 altrimenti
public int cercaBrano (String unTitoloBrano)
{
	for (int i = 0; i<brani.size(); i++)
	{
		if(brani.get(i).getTitolo().equals(unTitoloBrano))
		return i;
	}
	return -1;
}
// aggiunge un nuovo genere, se non esiste gia'
public void inserisciGenere (String unNomeGenere)
{
	for (int i = 0; i<generi.size(); i++)
	{
		if (cercaGenere(unNomeGenere) == -1)
		generi.add(new Genere(unNomeGenere));
	}
}
// aggiunge un nuovo artista, se non esiste gia'
public void inserisciArtista (String unNomeArtista)
{
	for (int i = 0; i<artisti.size(); i++)
	{
		if (cercaArtista(unNomeArtista) == -1)
		artisti.add(new Artista(unNomeArtista));
	}
}
//aggiunge un nuovo cd, se non esiste gia'
public void inserisciCd (String unTitoloCd, String unNomeArtista, String
unNomeGenere)
{
	inserisciArtista(unNomeArtista);
	inserisciGenere(unNomeGenere);

	for (int i = 0; i<cds.size(); i++)
	{
		if (cercaCd(unTitoloCd) == -1)
		cds.add(new Cd(unTitoloCd,
		artisti.get(cercaArtista(unNomeArtista)),
		generi.get(cercaGenere(unNomeGenere))));
```

```
	}
}
//aggiunge un nuovo brano, se non esiste gia'
public void inserisciBrano (String unTitoloBrano, String unTitoloCd,
String unNomeArtista, String unNomeGenere, double durata)
{
	inserisciCd(unTitoloCd, unNomeArtista, unNomeGenere);
	for (int i = 0; i<brani.size(); i++)
	{
		if (cercaBrano(unTitoloBrano) == -1)
		{
			brani.add(new Brano(unTitoloBrano,
			cds.get(cercaCd(unTitoloCd)), durata));
		}
	}
}
// restituisce tutti i brani di un cd presenti nell'archivio
public ArrayList<Brano> cdToBrani (String unTitoloCd)
{
	ArrayList<Brano> risultato = new ArrayList<Brano>();
	for(Brano a : brani)
	{
		if (a.getAlbum().getTitoloCd().equals(unTitoloCd))
		risultato.add(a);
	}
	return risultato;
}
// restituisce tutti i cd di un artista presenti nell'archivio
public ArrayList<Cd> artistaToCd (String unNomeArtista)
{
	ArrayList<Cd> risultato = new ArrayList<Cd>();
	for (Cd a : cds)
	{
		if (a.getArtista().getNomeArtista().equals(unNomeArtista))
		risultato.add(a);
	}
	return risultato;
}
// rimuove un brano
public void rimuoviBrano (String unTitoloBrano)
{
	if(cercaBrano(unTitoloBrano)!= -1)
	brani.remove(cercaBrano(unTitoloBrano));
}
// rimuove un cd e tutti i brani ad esso associati
public void rimuoviCd (String unTitoloCd)
{
	if(cercaCd(unTitoloCd) != -1)
	cds.remove(cercaCd(unTitoloCd));
	for (int i = 0; i < brani.size(); i++)
	{
		if(brani.get(i).getAlbum().getTitoloCd().equals(unTitoloCd))
		brani.remove(i);
	}
}
// restituisce il genere con il maggior numero di cd
public Genere maxCd ()
{
	Genere genmax = null;
	int cont = 0;
	for(int i = 0; i < generi.size(); i++)
	{
		int newcont = 0;
```

```
                for (int j = 0; j<cds.size(); j++)
                if(cds.get(j).getGenere().getNomeGenere().equals(generi.get(i)
                .getNomeGenere()))
                {
                      newcont++;
                }
                if (newcont>cont)
                {
                      cont = newcont;
                      genmax = generi.get(i);
                }
          }
          return genmax;
    }
    // compila una playlist di n brani casuali di un determinato genere
    public ArrayList<Brano> crea (int n, String unNomeGenere)
    {
          ArrayList<Brano> appoggio = new ArrayList<Brano>();
          Random destino = new Random();
          for(int i = 0; i < brani.size(); i++)
          {
                if(brani.get(i).getGenere().getNomeGenere().
                equals(unNomeGenere))
                appoggio.add(brani.get(i));
          }
          ArrayList<Brano> playlist = new ArrayList<Brano>();
          for (int j = 0; j<n; j++)
          {
                playlist.add(appoggio.remove(destino.nextInt
                (appoggio.size())));
          }
          return playlist;
    }
}
```

Gestione di una compagnia aerea

Testo:

Una compagnia aerea desidera gestire elettronicamente le prenotazioni sui singoli voli. Abbiamo una classe Cliente, con nome unico (si assuma che non ci siano clienti omonimi). Poi una classe Volo, che contiene due var d'istanza denominate posti e attesa. Ogni volo ha un numero max di passeggeri (passato come parametro al costruttore). Quando un cliente prenota il volo, lo mettiamo in posti se c'e` ancora disponibilita`, altrimenti lo mettiamo in attesa. Quando un cliente in posti disdice il suo volo, si libera un posto e prendiamo il primo elemento di attesa e lo trasferiamo su posti.
Suggerimento: posti e` un array di dimensione pari al max numero di passeggeri; attesa e` un ArrayList senza limitazioni sulla dimensione. Per l'array posti, usate la tecnica della sentinella vista nel cap 8. La sentinella deve essere anch'essa una variabile d'istanza.

Consigli:

Il metodo che risulta più complicato è disdici(String cognome): si devono introdurre due cicli for, il primo che scandisce gli elementi dell'array Cliente[] posti, eliminando il cliente cercato e spostandone uno dalla lista delle attese a quella delle prenotazioni, il secondo che verifica la presenza di un utente nell'arrayList <Cliente>() attesa, rimuovendolo in caso di esito positivo. L'unico problema, invece, che riguarda il metodo statistica(char ch) riguarda l'introduzione del tipo char: per facilitarne la comprensione, s'invita a rivedere il capitolo 4. Esso, comunque, può esser utile per individuare un carattere specifico di una stringa.

Cliente.java

```
public class Cliente
```

```
{
	private String cognome;
	public Cliente(String unCognome)
	{
		cognome = unCognome;
	}
	public String getCognome()
	{
		return cognome;
	}
}
```

Volo.java

```
import java.util.ArrayList;
public class Volo
{
	private int numero;
	private Cliente[] posti;
	private ArrayList<Cliente> attesa;

	public Volo (int numPass)
	{
		numero = 0 ;
		posti = new Cliente[numPass];
		attesa = new ArrayList<Cliente>();
	}

	public boolean prenota (String unCognome)
	{
		Cliente cognome = new Cliente(unCognome);
		if (numero < posti.length)
		{
			posti[numero] = cognome;
				numero++;
			return true;
		}
		else
		{
			attesa.add(cognome);
			return false;
		}
	}

	public void disdici(String unCognome)
	{
		for (int i = 0 ; i < numero ; i ++)
		{
			if (posti[i].getCognome().equalsIgnoreCase(unCognome))
			{
				System.arraycopy(posti,i+1,posti,i,posti.length-i-1);
				if(attesa.size() >= 1)
				{
					posti[numero-1] = attesa.get(0);
					attesa.remove(0);
				}
				else
					numero --;
				return;
			}
		}
		for (int k = 0; k < attesa.size(); k++)
		{
			if (attesa.get(k).getCognome().equalsIgnoreCase(unCognome))
```

```
                        attesa.remove(k);
                }
        }

        public String toString()
        {
                return "Sono prenotati per il volo " + numero + " clienti.\nSono in
                lista d'attesa " + attesa.size() + " clienti.";
        }


        /* il metodo statistica restituisce il numero di clienti aventi cognome
        che inizia con la lettera ch. Il tipo char contiene un solo carattere di
        una stringa, per questo il suo utilizzo è più immediato rispetto a quello
        del metodo substring(0,1) */
        public int statistica(char ch)
        {
                int cont = 0;
                for(int i = 0;i < numero; i++)
                        if(posti[i].getCognome().charAt(0) == ch)
                                cont++;
                for(int k = 0; k < attesa.size(); k++)
                        if(attesa.get(k).getCognome().charAt(0) == ch)
                                cont++;
                return cont;
        }
}
```

VoloTester.java

```
import javax.swing.*;
public class VoloTester
{
        public static void main(String[] argv)
        {
                String numPosti = JOptionPane.showInputDialog("Inserire il numero di
                        posti del volo: ");
                int numeroPosti = Integer.parseInt(numPosti);
                Volo volo = new Volo(numeroPosti);
                boolean done = false;
                while (!done)
                {
                        int n = Integer.parseInt(JOptionPane.showInputDialog("Scegli
                        l'opzione  digitando il numero corrispondente\n1. Inserisci
                        prenotazione\n2. Elimina prenotazione\n3. Statistica
                        cognome\n4. Statistica volo\n 5. Uscire"));
                        switch (n)
                        {
                        case 1:
                        {
                                String cogn1 = JOptionPane.showInputDialog("Cognome: ");
                                boolean prenota = volo.prenota(cogn1);
                                if (prenota)
                                JOptionPane.showMessageDialog(null, "Prenotazione
                                accettata");
                                else
                                JOptionPane.showMessageDialog(null, "Cliente inserito in
                                attesa");
                                        break;
                        }
                        case 2:
                        {
```

```
                        String cogn2 = JOptionPane.showInputDialog("Inserire il
                        cognome: ");
                        volo.disdici(cogn2);
                        JOptionPane.showMessageDialog(null, "Il cliente " +
                        cogn2 + " ha disdetto la prenotazione");
                              break;
                  }
                  case 3:
                  {
                        char lettera = JOptionPane.showInputDialog("Inserire
                        l'iniziale: ").charAt(0);
                        JOptionPane.showMessageDialog(null, "Sono presenti " +
                        volo.statistica(lettera) + " clienti il cui cognome
                        inizia con la lettera " + lettera);
                              break;
                  }
                  case 4:
                  {
                        JOptionPane.showMessageDialog(null, volo.toString());
                              break;
                  }
                  case 5:
                        done = true;
                              break;
                  default:
                  JOptionPane.showMessageDialog(null, "Scelta non valida");
                        break;
                  }
            }
      }
}
```

ESERCIZI DAGLI ESAMI

Tandem (appello dicembre 2005)
http://www.dei.unipd.it/~satta/teach/java/page/temi/dic1.pdf, **punto 4.**

Testo:
Una stringa è un tandem se è composta da due sequenze esattamente uguali. Ad esempio, la stringa "stringstring" `e un tandem. Si assuma una classe Tandem con una sola variabile d'istanza private String mystring. Utilizzando la ricorsione, si implementi un metodo boolean tandem(){...} che restituisce true se mystring è un tandem, e false altrimenti. Non usare alcuna istruzione di iterazione.

Consigli:
Si consiglia, in caso di dubbi, di rivedere gli esercizi del capitolo 16, precedentemente illustrati e spiegati.

Tandem.java

```
// Restituisce TRUE se una stringa è un tandem, altrimenti FALSE
public class Tandem
{
	private String myString;
	public Tandem(String s)
	{
		myString = s;
	}
	public boolean tandem()
	{
		if (myString.length() == 0) return true;
		if (myString.length() == 1) return false;

		if (myString.substring(0,1).equals(myString.substring
						(myString.length()/2,
						 myString.length()/2 + 1)))
		{
			Tandem s = new Tandem(myString.substring(1,
			myString.length()/2)+
			myString.substring(myString.length()/2 + 1));
			boolean res = s.tandem();
				return res;
		}
		else
		return false;
	}
}
```

TandemTester.java

```
public class TandemTester
{
	public static void main(String[] args)
	{
		Tandem t = new Tandem("stringstring");
		System.out.println(t.tandem());
	}
}
```

Gestione di un concerto (appello dicembre 2005)
http://www.dei.unipd.it/~satta/teach/java/page/temi/dic1.pdf, **punto 5**

Testo:
Un'agenzia per il turismo desidera gestire automaticamente la prenotazione dei posti per un concerto. Si sviluppi una classe Cliente avente come variabili d'istanza il nominativo ed numero telefonico del cliente, rappresentati entrambi come stringhe, con i relativi metodi di accesso ed un costruttore. Si sviluppi inoltre la seguente classe:

```
public class Spettacolo {
private Cliente[] prenotazioni;
private int n_prenotazione;
private ArrayList<Cliente> attesa;
public Spettacolo(int n){...}
public boolean libero(){...}
public int trova(String nome, String tel){...}
public void prenota(String nome, String tel){...}
public void disdici(String nome, String tel){...}
public boolean incompleto(){...}
}
```

L'array prenotazioni contiene i clienti che hanno il posto, la lista attesa contiene i clienti in lista d'attesa. La lista d'attesa deve essere gestita con la politica primo arrivato, primo servito. Il costruttore inizializza una classe con un array prenotazioni (inizialmente vuoto) avente lunghezza n specificata come parametro. Il metodo libero restituisce true solo se vi sono posti ancora liberi. Il metodo trova restituisce 0 se il cliente specificato ha il posto, 1 se il cliente è in attesa e -1 altrimenti. Il metodo prenota inserisce il cliente specificato nell'oggetto (eventualmente in attesa). Il metodo disdici rimuove il cliente specificato dall'oggetto. Nel caso venga liberato un posto in prenotazioni, viene trasferito il primo cliente da attesa. Infine, il metodo incompleto restituisce true se esiste almeno un cliente che abbia almeno un posto ed almeno una prenotazione in attesa; il metodo restituisce false in caso contrario.

Consigli:
L'esercizio è molto simile, come modello e come metodi, a quelli già spiegati nella sezione dei modelli d'esame. Per come viene specificata la funzione del metodo incompleto() si assume che un cliente possa avere più prenotazioni che possono trovarsi sia nell'array prenotazioni, sia nell'ArrayList attesa. Sulla base di questa considerazione si implementa il metodo disdici in modo che venga cancellata una prenotazione per volta a partire dalla lista delle attese. A questo scopo si è preferito invertire l'ordine delle istruzioni nel metodo trova() in modo da analizzare prima la lista d'attesa.

Cliente.java

```
public class Cliente
{
	private String nome;
	private String numTel;

	public Cliente(String unNome, String unNumero)
	{
		nome = unNome;
```

```
            numTel = unNumero;
      }

      public String getNome()
      {
            return nome;
      }

      public String getNumero()
      {
            return numTel;
      }
}
```

Spettacolo.java

```
import java.util.ArrayList;

public class Spettacolo
{
      private Cliente[] prenotazioni;
      private int n_pren;
      private ArrayList<Cliente> attesa;

      public Spettacolo(int n)
      {
            prenotazioni = new Cliente[n];
            n_pren = 0;
            attesa = new ArrayList<Cliente>();
      }

      public boolean libero()
      {
            if(n_pren >= prenotazioni.length) return false;
            return true;
      }

      public int trova(String nome, String tel)
      {
            for(Cliente a : attesa)
                  if(a.getNome().equalsIgnoreCase(nome) &&
                        a.getNumero().equalsIgnoreCase(tel))
                        return 1;
            for(int i = 0; i < n_pren; i++)
                  if(prenotazioni[i].getNome().equalsIgnoreCase(nome) &&
                        prenotazioni[i].getNumero().equalsIgnoreCase(tel))
                        return 0;
            return -1;
      }

      public void prenota(String nome, String tel)
      {
            if(libero())
            {
                  prenotazioni[n_pren] = new Cliente(nome, tel);
                  n_pren++;
            }
            else
                  attesa.add(new Cliente(nome, tel));
      }

      public void disdici(String nome, String tel)
      {
```

```
            if (trova(nome, tel) == 1)
                  for (int i = 0; i < attesa.size(); i++)
                  {
                        if(attesa.get(i).getNome().equalsIgnoreCase(nome) &&
                              attesa.get(i).getNumero().equalsIgnoreCase(tel))
                        {
                              attesa.remove(i);
                              return;
                        }
                  }
            else if(trova(nome, tel) == 0)
                  for (int i = 0; i < n_pren; i++)
                  {
                        if(prenotazioni[i].getNome().equalsIgnoreCase(nome) &&
                              prenotazioni[i].getNumero().equalsIgnoreCase(tel))
                        {
                              System.arraycopy(prenotazioni,i + 1, prenotazioni,
                                    i, prenotazioni.length - i - 1);
                              if(attesa.size() != 0)
                              {
                                    prenotazioni[n_pren - 1] = attesa.get(0);
                                    attesa.remove(0);
                              }
                              else n_pren--;
                              return;
                        }
                  }
      }

      public boolean incompleto()
      {
            for(int i = 0; i < n_pren; i++)
                  if(trova(prenotazioni[i].getNome(),
                        prenotazioni[i].getNumero()) == 1)
                        return true;
            return false;
      }
}
```

Shuffle (appello gennaio 2006)
http://www.dei.unipd.it/~satta/teach/java/page/temi/genn1.pdf, **punto 4**

Testo:
Date due stringhe, la stringa shuffle delle due si ottiene alternando i caratteri della prima stringa con i caratteri della seconda, e copiando i rimanenti caratteri nel caso le due stringhe abbiano diversa lunghezza. Ad esempio, date le stringhe "casa" e "scatola", la stringa shuffle delle due `e "csacsaatola". Si implementi un metodo statico String shuffle(String s1, String s2){...} che restituisce la stringa shuffle di s1 e s2.

Consigli:
L'esercizio è di facile implementazione: è fondamentale, però, riuscire ad definire con accortezza i casi base (se una delle due o entrambe le stringhe hanno lunghezza pari a zero, quali sono le conseguenze).

Shuffle.java

```
// Classe che mescola due stringhe in modo ricorsivo
public class Shuffle
{
      public static String shuffle(String s1, String s2)
      {
```

```
            // casi base
            if (s1.length()==0 && s2.length()==0)
                  return s1 + s2;
            if (s1.length()==0 && s2.length()>0)
                  return s2;
            if (s2.length()==0 && s1.length()>0)
                  return s1;
            // si creano delle sottostringhe
            String shorterS1 = s1.substring(1);
            String shorterS2 = s2.substring(1);
            return s1.substring(0,1)+s2.substring(0,1)+
            shuffle(shorterS1, shorterS2);
      }
}
```

Classe ShuffleTester

```
import java.util.Scanner;
public class ShuffleTester
{
      public static void main (String[] args)
      {
            Scanner input = new Scanner(System.in);
            System.out.println("Inserisci la prima stringa");
            String s1= input.next();
            System.out.println("Inserisci la seconda stringa");
            String s2= input.next();
            System.out.println(Shuffle.shuffle(s1,s2));
      }
}
```

Assegnazione delle tesi (appello settembre 2006)
http://www.dei.unipd.it/~satta/teach/java/page/temi/sett1.pdf, **punto 5**

Testo:
Un corso di laurea desidera gestire automaticamente l'assegnazione delle tesi ai propri studenti. Sviluppare una classe Tesi, avente come variabile d'istanza un titolo rappresentato come stringa, ed una classe Studente, avente come variabili d'istanza il nominativo (stringa) ed un puntatore ad un oggetto di classe Tesi. Per entrambe le classi definire i relativi metodi di accesso ed un costruttore. Si sviluppi inoltre la seguente classe:

```
public class Assegnazioni {
private ArrayList<Tesi> tesi;
private ArrayList<Studente> laureandi;
public Assegnazioni(){...}
public void aggiungiTesi(String titoloTesi){...}
public void aggiungiStudente(String nomeStudente, String titoloTesi){...}
public void laureato(String nomeStudente){...}
public void liberaTesi(String titoloTesi){...}
public int disponibili(){...}}
```

La lista tesi contiene le tesi proposte dai docenti, la lista laureandi contiene gli studenti che hanno avuto una assegnazione di tesi. Ciascun studente avr`a il proprio campo puntatore collegato ad un oggetto nella lista tesi, rappresentante la tesi a lui assegnata. Il costruttore inizializza un oggetto Assegnazioni con entrambe le liste vuote. I metodi aggiungiTesi e aggiungiStudente inseriscono una nuova tesi ed un nuovo studente, rispettivamente, nel sistema. Se la tesi associata allo studente non `e gi`a presente nel sistema, questa dovr`a essere creata. Il metodo laureato elimina dal sistema uno studente e la propria tesi. Il metodo liberaTesi elimina lo studente associato alla tesi specificata

come argomento, rendendo di nuovo disponibile la tesi stessa. Il metodo disponibili restituisce il numero di tesi presenti nel sistema non ancora assegnate ad un laureando. Sviluppare tutti i metodi della classe Assegnazioni.

Consigli:
Le classi Tesi e Studente risultano di facile implementazione. Le maggiori difficoltà sono riscontrabili nella classe Assegnazioni.
Come si è già affermato negli esercizi precedenti, s'invitano ad usare metodi ausiliari (si vedano, ad esempio, i metodi trovaTesi(String titoloTesi) e trovaLaureando(String nomeStudente)): in tal modo, compilare il codice risulta più facile. E' stato aggiunto un metodo toString(), non richiesto dalla consegna, solo per facilitare la verifica delle classi tramite il Tester.

Tesi.java

```
public class Tesi
{
	private String titolo;

	public Tesi(String unTitolo)
	{
		titolo = unTitolo;
	}

	public String getTitolo()
	{
		return titolo;
	}
}
```

Studente.java

```
public class Studente
{
	private String nome;
	private Tesi miaTesi;

	public Studente(String unNome, Tesi unaTesi)
	{
		nome = unNome;
		miaTesi = unaTesi;
	}

	public String getNome()
	{
		return nome;
	}

	public Tesi getTesi()
	{
		return miaTesi;
	}
}
```

Assegnazioni.java

```
import java.util.ArrayList;

public class Assegnazioni
{
	private ArrayList<Tesi> tesi;
	private ArrayList<Studente> laureandi;
```

```
public Assegnazioni()
{
	tesi = new ArrayList<Tesi>();
	laureandi = new ArrayList<Studente>();
}

/*metodo ausiliario che restituisce l'indice di posizione di una tesi
nell'ArrayList tesi*/
private int trovaTesi(String titoloTesi)
{
	for(int i = 0; i < tesi.size(); i++)
		if(tesi.get(i).getTitolo().equalsIgnoreCase(titoloTesi))
			return i;
	return -1;
}

/*aggiunge una nuova tesi dopo aver verificato che non sia già presente in
tesi*/
public void aggiungiTesi(String titoloTesi)
{
	if(trovaTesi(titoloTesi) == -1)
		tesi.add(new Tesi(titoloTesi));
}

/*metodo ausiliario che restituisce l'indice di posizione di un laureando
nell'ArrayList laureandi*/
private int trovaLaureando(String nomeStudente)
{
	for(int i = 0; i < laureandi.size(); i++)
		if(laureandi.get(i).getNome().equalsIgnoreCase(nomeStudente))
			return i;
	return -1;
}

/*Verifica che la tesi da assegnare non sia già assegnata ad un altro
studente, altrimenti esce dal metodo. Aggiunge uno studente a laureandi
dopo aver verificato che non sia già presente. Gli attribuisce la tesi
dall'ArrayList tesi dopo averla eventualmente creata*/
public void aggiungiStudente (String nomeStudente, String titoloTesi)
{
	for(Studente a : laureandi)
		if(a.getTesi().getTitolo().equalsIgnoreCase(titoloTesi))
			return;
	if(trovaLaureando(nomeStudente) == -1)
	{
		/*la verifica che la tesi non sia già presente nell'ArrayList
		avviene nel metodo aggiungiTesi()*/
		aggiungiTesi(titoloTesi);
		laureandi.add(new Studente(nomeStudente,
			tesi.get(trovaTesi(titoloTesi))));
	}
}

/*rimuove uno studente dall'ArrayList laureandi e la relativa tesi
dall'ArrayList tesi dopo le dovute verifiche*/
public void laureato(String nomeStudente)
{
	if(trovaLaureando(nomeStudente) == -1) return;
	int i = trovaLaureando(nomeStudente);
	String suaTesi = laureandi.get(i).getTesi().getTitolo();
	tesi.remove(trovaTesi(suaTesi));
	laureandi.remove(i);
}
```

```
	/*rimuove uno studente dall'ArrayList laureandi rendendo disponibile la
	tesi*/
	public void liberaTesi(String titoloTesi)
	{
		for(int i = 0; i < laureandi.size(); i++)
			if(laureandi.get(i).getTesi().getTitolo()
				.equalsIgnoreCase(titoloTesi))
				laureandi.remove(i);
	}

	public int disponibili()
	{
		return tesi.size() - laureandi.size();
	}

	public String toString()
	{
		String r = "I laureandi sono:";
		String s = "\nLe tesi libere sono:";
		for(Studente a : laureandi)
			r = r + "\n" + a.getNome() + " la cui tesi é " +
				a.getTesi().getTitolo();
		for(Tesi a : tesi)
		{
			boolean assegnata = false;
			for(Studente b : laureandi)
				if(b.getTesi().getTitolo()
					.equalsIgnoreCase(a.getTitolo()))
					assegnata = true;
			if(!assegnata)
				s = s + "\n" + a.getTitolo();
		}
		return r + s;
	}
}
```

AssegnazioniTester.java

```
import javax.swing.JOptionPane;
public class AssegnazioniTester
{
	public static void main(String args[])
	{
		int i = 0;
		Assegnazioni assegnazioni1 = new Assegnazioni();
		while(i != 7)
		{
			String scegli = JOptionPane.showInputDialog("
				---TESI DI LAUREA---\n
				Operazioni eseguibili:\n
				1)Aggiungi tesi\n
				2)Assegna tesi allo studente\n
				3)Libera tesi\n
				4)Cancella studenti laureati\n
				5)Vedi numero delle tesi disponibili\n
				6)vedi situazione assegnazioni\n
				7)Esci");
			i = Integer.parseInt(scegli);
			switch(i)
			{
			case 1:
			{
```

```
                    String titolo = JOptionPane.showInputDialog("Titolo tesi:");
                    assegnazioni1.aggiungiTesi(titolo);
                    break;
                    }
                    case 2:
                    {
                    String studente = JOptionPane.showInputDialog("Nome
                          studente:");
                    String tesi = JOptionPane.showInputDialog("Titolo tesi:");
                    assegnazioni1.aggiungiStudente(studente,tesi);
                    break;
                    }
                    case 3:
                    {
                    String titolo = JOptionPane.showInputDialog("Titolo tesi:");
                    assegnazioni1.liberaTesi(titolo);
                    break;
                    }
                    case 4:
                    {
                    String studente = JOptionPane.showInputDialog("Nome
                          studente:");
                    assegnazioni1.laureato(studente);
                    break;
                    }
                    case 5:
                    {
                    JOptionPane.showMessageDialog(null,"Numero di tesi disponibili
                          è: " + assegnazioni1.disponibili());
                    break;
                    }
                    case 6:
                    {
                    JOptionPane.showMessageDialog(null, assegnazioni1.toString());
                    break;
                    }
                    }
              }
        }
}
```

Bilanciato (appello dicembre 2006)
http://www.dei.unipd.it/~satta/teach/java/page/temi/dic2.pdf, **punto 4.**

Testo:
Questa classe serve a controllare se un array di integer è bilanciato. Un array di interi si dice bilanciato se la sua lunghezza è un numero pari maggiore di zero, e se la somma del primo e dell'ultimo elemento dell'array è uguale alla somma del secondo e del penultimo elemento ed è anche uguale alla somma del terzo e del terzultimo elemento, se esistono, e così via. Ad esempio, l'array [12, -3, 22, 7] è bilanciato.
Infatti 12+7=-3+22.

Consigli:
Si ricorda che, come enunciato nel capitolo 16, la ricorsione NON prevede cicli for e while. Si costruisce un metodo, boolean bilanciato. Si definiscono i casi base: se la lunghezza dell'array è dispari o contiene un numero di elementi minore a 4, restituisce false; se, invece, la lunghezza dell'array è pari a 4 e la somma degli estremi è uguale a quella dei medi, restituisci vero. Se l'array non corrisponde a nessuno dei casi appena citati, si crea un array più piccolo, eliminando gli estremi, e si applica il metodo a tale vettore.

```
Bilanciato_normale.java
public class Bilanciato_normale
{
	int[] myArray;
	public Bilanciato_normale(int[] aMyArray)
	{
		myArray=aMyArray;
	}
	public boolean bilanciato()
	{
		// casi base: se l'array è di lunghezza dispari ritorna false
		if (myArray.length%2!=0)
			return false;
		/* se la lunghezza dell'array è pari a 2 e i due numeri sono uguali,
		restituisce true
		if (myArray.length==2 && myArray[0]==myArray[1])
		return true;
		/* bisogna che l'array sia almeno di 4 elementi per essere
		bilanciato, se è più piccolo ritorna false*/
		if (myArray.length<4)
			return false;
		/* se il primo+l'ultimo=secondo+penultimo e l'array è lungo 4
		elementi, restituisce true*/
		if (myArray[0]+myArray[myArray.length-1]
		==myArray[1]+myArray[myArray.length-2])
			if (myArray.length==4)
				return true;


		else
		/* altrimenti crea un array più piccolo togliendo il primo e
		l'ultimo elemento e ripete il metodo ricorsivo*/
		{
			int[] small = new int[myArray.length - 2];
			System.arraycopy(myArray,1,small,0,myArray.length-2);
			Bilanciato_normale piccolo=new Bilanciato_normale(small);
				return piccolo.bilanciato();
		}
		return false;
	}
}

BilanciatoTester.java
public class BilanciatoTester
{
	public static void main(String[] args)
	{
		// si crea un array e lo si riempie con dei numeri interi
		int[] intArray={1,1,2,2};
		/* si crea un oggetto di classe Bilanciato e lo si usa intArray come
		parametro esplicito*/
		Bilanciato_normale brex= new Bilanciato_normale(intArray);
		/* se l'array è bilanciato stampare sullo schermo "bilanciato",
		altrimenti "non bilanciato"*/
		if (brex.bilanciato())
			System.out.println("bilanciato");
		else
			System.out.println("non bilanciato");
	}
}
```

Azienda Sanitaria (appello dicembre 2006)

http://www.dei.unipd.it/~satta/teach/java/page/temi/dic2.pdf, **punto 5.**

Testo:
Un'azienda sanitaria desidera creare un archivio elettronico per la gestione dei propri medici di base e delle liste dei relativi pazienti. Si sviluppi una classe Medico avente il nominativo (stringa) come variabili d'istanza, un metodo d'accesso ed un costruttore. Si sviluppi inoltre una classe Paziente avente come variabili d'istanza il numero (intero) di tessera sanitaria ed un riferimento al proprio medico curante, con i relativi metodi di accesso ed un costruttore. Si sviluppi la classe AziendaSanitaria, coi seguenti metodi: Il metodo aggPaziente inserisce un oggetto paziente nella arraylist pazienti, con un riferimento al proprio medico curante nella arraylist medici. Non devono mai essere duplicati pazienti o medici. Il metodo listaMedico restituisce una arraylist con tutti e soli i pazienti che hanno il medico specificato dal parametro esplicito come medico curante. Il metodo statMedico restituisce un riferimento al medico nella arraylist medici avente il maggior numero di pazienti. Sviluppare tutti i metodi della classe AziendaSanitaria.

Consigli:
Le maggiori difficoltà possono essere incontrate nello sviluppo della classe AziendaSanitaria.
Si consiglia l'uso di metodi ausiliari. Essi non sono richiesti dal testo, ma sono utili per accorciare il codice nei metodi successivi, come il metodo int trovaPaziente. Inoltre, esso è private, in quando non deve essere accessibile dall'utilizzatore futuro di questa classe. Altra difficoltà può essere riscontrata nel metodo void aggiungiPaziente: tra i suoi parametri impliciti, non compare un riferimento ad un oggetto della classe Medico, ma ad una stringa del nome del dottore. E' necessario, quindi, ricercare il nome del medico nell'ArrayList<Medico> e, se l'esito della ricerca è positivo, fare il collegamento tra il medico cercato e la posizione dell'oggetto nella lista medici.

Medico.java
```
public class Medico
{
	// variabili d'istanza
	private String nome;
	// costruttore
	public Medico(String unNome)
	{
		nome=unNome;
	}
	// metodi
	public String getNome()
	{
		return nome;
	}
}
```

Paziente.java
```
public class Paziente
{
	// variabili d'istanza
	int numeroTessera;
	Medico medicoCurante;
	// costruttore
	public Paziente(int unNumeroTessera,Medico unMedicoCurante)
	{
		numeroTessera=unNumeroTessera;
		medicoCurante=unMedicoCurante;
	}
	// metodi
```

```
	public int getTessera()
	{
		return numeroTessera;
	}
	public Medico getMedico()
	{
		return medicoCurante;
	}
}

AziendaSanitaria.java
import java.util.ArrayList;
public class AziendaSanitaria
{
	private ArrayList<Medico> medici;
	private ArrayList<Paziente> pazienti;
	public AziendaSanitaria()
	{
		medici=new ArrayList<Medico>();
		pazienti=new ArrayList<Paziente>();
	}
	/* cerca nell'array medici un medico con lo stesso nome di unNomeMedico,
	se lo trova ritorna true altrimenti false*/
	public boolean trovaMedico(String unNomeMedico)
	{
		for(Medico med: medici)
			if (med.getNome().equals(unNomeMedico))
				return true;
		return false;
	}
	// aggiunge un medico, se non già presente nell'arraylist Medico
	public void aggMedico(String unNomeMedico)
	{
		// controlla che il medico non sia già presente
		if (!trovaMedico(unNomeMedico))
			medici.add(new Medico(unNomeMedico));
	}
	/* cerca un paziente restituendomi la sua posizione nell'arraylist,
	altrimenti -1*/
	private int trovaPaziente(int unNumeroTessera)
	{
		// n è la variabile che rappresenta la posizione del paziente
		int n=0;
		for (Paziente paz:pazienti)
		{
			if (paz.getTessera()==unNumeroTessera)
			return n;
			n++;
		}
		return -1;
	}
	/* cerca un paziente e, dopo aver individuato la sua posizione, lo toglie
	dall'arrayList*/
	public void eliminaPaziente(int unNumeroTessera)
	{
		/* si assegna a una variabile temporanea il valore di trovaPaziente
		per non calcolarlo più volte*/
		int posizione=trovaPaziente(unNumeroTessera);
		// se il paziente non c'è, il metodo termina la sua esecuzione
		if (posizione==-1)
			return;
		// toglie dall'array il paziente con posizione nota
		pazienti.remove(posizione);
```

```
	}
	// aggiunge un nuovo paziente all'arraylist Paziente, se non esiste già
	public void aggPaziente(int unaTessera, String unNomeMedico)
	{
		if (trovaPaziente(unaTessera)!=-1)
		if (trovaMedico(unNomeMedico))
		{
			int posMed=0;
			for (Medico med: medici)
			if (med.getNome().equals(unNomeMedico))
			{
			pazienti.add(new Paziente(unaTessera,medici.get(posMed)));
				return;
			}
			else posMed++;
		}
		else
		{
		medici.add(new Medico(unNomeMedico));
		pazienti.add(new Paziente(unaTessera,medici.get(medici.size()-1)));
		}
	}
	/* dato unNomeMedico, restituisce la lista di tutti i pazienti in cura a
	tale dottore*/
	public ArrayList<Paziente> listaMedico(String unNomeMedico)
	{
		// temp è l'ArrayList che conterrà i pazienti del dottore
		ArrayList<Paziente> temp=new ArrayList<Paziente>();
		for (Paziente paz: pazienti)
			/* se il nome del medico coincide viene aggiunto un nuovo
			paziente a temp*/
			if (paz.getMedico().getNome().equals(unNomeMedico))
				temp.add(paz);
					return temp;
	}
	// ritorna un riferimento al medico con più pazienti
	public Medico statMedico()
	{
		/* maxPax rappresenta il numero massimo di pazienti conteggiati a un
		medico*/
		int maxPaz=0;
		// temp è il medico con più paziente
		Medico temp=null;
		for (Medico med: medici)
		{
			int numPaz=0;
			// si contano i pazienti di un medico
			for (Paziente paz: pazienti)
			if (paz.getMedico().getNome().equals(med.getNome()))
				numPaz++;


			/* se il suo numero è maggiore del max finora conteggiato,
			si sostituiscono i vecchi dati coi nuovi*/
			if (numPaz>maxPaz)
			{
				maxPaz=numPaz;
					temp=med;
			}
		}
		return temp;
	}
}
```

Bilanciato (appello gennaio 2007)
http://www.dei.unipd.it/~satta/teach/java/page/temi/genn2.pdf , **punto 4.**

Testo:
Un array di interi di lunghezza n si dice bilanciato se esiste un indice i tale che la somma dei primi i elementi dell'array sia uguale alla somma dei rimanenti n – i elementi. Ad esempio, l'array [20, -4, 11, 1, 4] è bilanciato, perchè i primi due elementi hanno somma uguale ai rimanenti elementi, mentre l'array [7, 13, 6, 8] non è bilanciato, perché nessun indice verifica la proprietà richiesta. Si assuma una classe Bilanciato con una sola variabile d'istanza private int[] myArray. Si implementi un metodo public boolean bilanciato(){...} che restituisce true se myArray è bilanciato, e false altrimenti.

Consigli:
Quest'esercizio è stato svolto in due modalità diverse. Si è tentato di enunciare ciascuna metodologia nel modo più chiaro e preciso.

Primo sviluppo: bilanciato ricorsivo
Si costruisce un metodo ausiliario, avente tre parametri impliciti: due posizioni(a destra e a sinistra dell'array) e un valore sentinella n, che punta nella posizione da "spostare" da destra a sinistra.
Casi base: se sono già state controllate tutte le posizioni, restituisce false; se sinistra è uguale a destra, restituisce true. Si incrementa la sentinella n e, dopo aver ridefinito la posizione destra e sinistra, si richiama lo stesso metodo con i nuovi parametri. Infine, si crea il metodo richiesto dall'esercizio, in cui si assegna a sinistra il primo valore e a destra il secondo valore dell'array. Senza usare la ricorsione, si sfrutta un ciclo for per assegnare a destra la somma di tutti i valori dell'array dal secondo in poi, ricordandosi poi di riapplicare il metodo ausiliario.

Bilanciato.java

```
public class Bilanciato
{
	int[] myArray;
	public Bilanciato(int[] unArray)
	{
		myArray=unArray;
	}
	public boolean bilanciato()
	{
		// si assegna a sx il primo valore dell'array
		int sx=myArray[0];
		/* si assegna a dx la somma di tutti i valori dell array dal secondo
		in poi*/
		int dx=myArray[1];
		for (int i=2;i<myArray.length;i++)
			dx=dx+myArray[i];
		// richiamo al metodo ausiliario
		return bilanciato(sx,dx,0);
	}
	private boolean bilanciato(int unSx,int unDx,int n)
	{
		/* s'incrementa n, che punta nella posizione da "spostare" da dx a
		sx*/
		n++;
		/* se sono già state controllate tutte le posizioni e la risposta è
		stata negativa, si restituisce false*/
		if (n>=myArray.length)
			return false;
		// se sx è uguale a dx, si restituisce true
		if (unSx==unDx)
```

```
                return true;
            // si spostano le posizioni dei valori sx e dx
            unSx=unSx+myArray[n];
            unDx=unDx-myArray[n];
                // richiamo alla ricorsione
                return bilanciato(unSx,unDx,n);
      }
}
BilanciatoTester.java
public class BilanciatoTester
{
      public static void main(String[] args)
      {
            int[] intArray={20, -4, 11, 1, 4};
            Bilanciato brex= new Bilanciato(intArray);
            if (brex.bilanciato())
            System.out.println("bilanciato");
            else System.out.println("non bilanciato");
      }
}
```

Secondo sviluppo: bilanciato statico

Si definisce il metodo boolean bilanciato come statico (capitolo 4), avente come parametro implicito un array di interi di posizioni non definite. Si usa il metodo iterativo: si assegna a sx il primo valore dell'array, e a dx la somma di tutti i valori dell'array dal secondo in poi tramite un ciclo for. Per tutti i valori definiti da i fino alla fine dell'array, si confrontano sx e dx: se sono uguali, il metodo restituisce vero, altrimenti si decrementa la posizione i da dx, la si aggiung a sx e poi si riconfronta. Se sx non risulta mai essere uguale a dx, allora bilanciato(int []unArray) restituisce false.

```
Bilanciato.java
public class Bilanciato
{
      public static boolean bilanciato(int[] unArray)
      {
            // si assegna a sx il primo valore dell'array
            int sx=unArray[0];
            /* si assegna a dx la somma di tutti i valori dell array dal secondo
            in poi*/
            int dx=unArray[1];
            for (int i=2;i<unArray.length;i++)
                  dx=dx+unArray[i];
            /* con un ciclo for, si confrontano sx e dx da 2 alla fine
            dell'array, se sono uguali restituisce true*/
            for (int i=1;i<unArray.length;i++)
                  if (sx==dx)
                        return true;
                  // se sono diversi, si decrementa i da dx
                  else
                  {
                        /* lo si incrementa a sx, confrontando ulteriormente i
                        valori*/
                        dx=dx-unArray[i];
                        sx=sx+unArray[i];
                  }
                  // se sx non è mai uguale a dx, restituisce false
                  return false;
      }
}

BilanciatoTester.java
public class BilanciatoTester
```

```
{
	public static void main(String[] args)
	{
		int[] zuzi={20, -4, 11, 1, 4};
		if (Bilanciato.bilanciato(zuzi))
			System.out.println("bilanciato");
		else
			System.out.println("non bilanciato");
	}
}
```

Parcheggio (appello gennaio 2007)
http://www.dei.unipd.it/~satta/teach/java/page/temi/genn2.pdf , **punto 5**

Testo:
Si desidera simulare un parcheggio a pagamento per autovetture. Si sviluppi la classe Vettura, avente le variabili d'istanza private int targa, private short oraArrivo, private short oraScadenza, dove le ore sono interi compresi tra 0 e 23. Implementare i relativi metodi di accesso e riscrittura per tali variabili, più un costruttore. Il parcheggio è rappresentato mediante la classe Parcheggio. Il costruttore inizializza un parcheggio con massima capienza numPosti e 0 vetture. I metodi entraVettura ed esceVettura aggiungono e rimuovono, rispettivamente, le vetture dei clienti del parcheggio. Viene restituito il valore true solo nel caso l'operazione possa essere eseguita. Il metodo aggiornaOra incrementa di una unità l'ora corrente, e sposta nella zona rimozione tutte le vetture il cui tempo sia scaduto. Il metodo promozione offre gratuitamente oreGratuite ore a tutte le vetture nel parcheggio il cui numero di targa termini con le cifre codice, dove codice deve essere un numero di due cifre. Il metodo statVetture restituisce una arraylist con tutte le vetture che hanno il più alto numero di ore di parcheggio pagato. Sviluppare tutti i metodi della classe.

Consigli:
E' introdotto, per la prima volta in un esame, il tipo short (capitolo 4). Anch'esso è un valore numerico, e quindi va trattato come un int o un double: a differenza dei precedenti, però, il tipo short ha un intervallo numerico minore. Per questo sono stati introdotti dei cast nelle operazioni, in modo da evitare la segnalazione del compilatore "possible loss of precision". Nella classe Vettura, è necessario creare un metodo ausiliario void modifyOraScadenza(), che serve ad incrementare le ore di sosta pagate.

Vettura.java

```
public class Vettura
{
	private int targa;
	private short oraArrivo;
	private short oraScadenza;

	public Vettura(int unaTarga, short unOraArrivo, short unOraScadenza)
	{
		targa = unaTarga;
		oraArrivo = unOraArrivo;
		oraScadenza = unOraScadenza;
	}

	public int getTarga()
	{
		return targa;
	}

	public short getOraArrivo()
```

```
	{
		return oraArrivo;
	}

	public short getOraScadenza()
	{
		return oraScadenza;
	}

	public void modifyOraScadenza(short oreDaAggiungere)
	{
		oraScadenza = (short)(oraScadenza + oreDaAggiungere);
		if(oraScadenza > 23) oraScadenza = (short)(oraScadenza - 24);
	}

	public short getOrePagate()
	{
		if(oraArrivo > oraScadenza) return (short)(oraScadenza + 24 -
			oraArrivo);
		return (short)(oraScadenza - oraArrivo);
	}
}
```

Parcheggio.java

```
import java.util.ArrayList;

public class Parcheggio
{
	private short oraCorrente;
	private Vettura[] vetture;
	private int numVetture;
	private ArrayList<Vettura> rimozione;

	public Parcheggio(int numPosti)
	{
		vetture = new Vettura[numPosti];
		numVetture = 0;
		rimozione = new ArrayList<Vettura>();
		oraCorrente = 0;
	}

	public boolean entraVettura(int targa, short orePagate)
	{
		if(numVetture == vetture.length) return false;
		short oraScadenza = (short)(oraCorrente + orePagate);
		if(oraScadenza > 23) oraScadenza = (short)(oraScadenza -24);
		Vettura inEntrata = new Vettura(targa, oraCorrente, oraScadenza);
		vetture[numVetture] = inEntrata;
		numVetture++;
		return true;
	}

	public boolean esceVettura(int targa)
	{
		for(int i = 0; i < numVetture; i++)
		{
			if(vetture[i].getTarga() == targa)
			{
				System.arraycopy(vetture, i + 1, vetture, i,
					vetture.length - i - 1);
				numVetture--;
				return true;
			}
```

```
            }
            return false;
      }

      public void aggiornaOra()
      {
            oraCorrente++;
            if(oraCorrente == 24) oraCorrente = 0;
            boolean hoFinito = false;
            while(!hoFinito)
            {
                  hoFinito = true;
                  for(int i = 0; i < numVetture ; i++)
                  {
                        if(vetture[i].getOraScadenza() == oraCorrente)
                        {
                              Vettura inRimozione = vetture[i];
                              rimozione.add(inRimozione);
                              System.arraycopy(vetture, i + 1, vetture, i,
                                    vetture.length - i - 1);
                              numVetture--;
                              hoFinito = false;
                        }
                  }
            }
      }

      public void promozione(short codice, short oreGratuite)
      {
            for(int i = 0; i < numVetture; i++)
                  if((vetture[i].getTarga() % 100) == codice)
                        vetture[i].modifyOraScadenza(oreGratuite);
      }

      public ArrayList<Vettura> statVetture()
      {
            ArrayList<Vettura> maggiori = new ArrayList<Vettura>();
            short maxOrePagate = 0;
            for(int i = 0; i < numVetture; i++)
                  if(vetture[i].getOrePagate() > maxOrePagate) maxOrePagate =
                        vetture[i].getOrePagate();
            for(int i = 0; i < numVetture; i++)
                  if(vetture[i].getOrePagate() == maxOrePagate)
                        maggiori.add(vetture[i]);
            return maggiori;
      }
}

ParcheggioTester.java
import javax.swing.*;
import java.util.ArrayList;

public class ParcheggioTester
{
      public static void main(String args[])
      {
            int numPosti = Integer.parseInt(JOptionPane.showInputDialog
                  ("Inserisci la capienza del parcheggio:"));
            Parcheggio park = new Parcheggio(numPosti);
            int numScelta,numTarga;
            short numOre,numPromo,orePromo;
            numScelta = 0;
            while(numScelta != 6)
```

```
            {
                  numScelta = Integer.parseInt(JOptionPane.showInputDialog
                        ("Operazioni eseguibili:
                        \n1) Ingresso auto
                        \n2) Uscita auto
                        \n3) Aggiorna ora
                        \n4) Nuova promozione!
                        \n5) Lista auto con sosta più lunga
                        \n6) Esci"));
                  switch(numScelta)
                  {
                  case 1: numTarga = Integer.parseInt
                        (JOptionPane.showInputDialog("Inserire la targa
                        dell'auto:"));
                        numOre = Short.parseShort
                        (JOptionPane.showInputDialog("Inserire la durata della
                        sosta:"));
                        if(!park.entraVettura(numTarga, numOre))
                              JOptionPane.showMessageDialog
                              (null,"Spiacente, il parcheggio è pieno.");
                        else
                              JOptionPane.showMessageDialog
                              (null,"Auto parcheggiata.");break;
                  case 2: numTarga = Integer.parseInt
                        (JOptionPane.showInputDialog("Inserisci la targa
                        dell'auto:"));
                        if(!park.esceVettura(numTarga))
                              JOptionPane.showMessageDialog
                              (null,"Errore: targa non presente nel
                              parcheggio!");
                        else
                              JOptionPane.showMessageDialog(null,"Arrivederci e
                              grazie!");break;
                  case 3: park.aggiornaOra();break;
                  case 4: numPromo = Short.parseShort
                        (JOptionPane.showInputDialog("Inserisci il codice
                        vincitore:"));
                        orePromo = Short.parseShort
                        (JOptionPane.showInputDialog("Inserisci il numero di ore
                        da regalare:"));
                        park.promozione(numPromo, orePromo);break;
                  case 5: String listaAuto = "Auto con prenotazione più
                        lunga:";
                        ArrayList<Vettura> lista = new ArrayList<Vettura>();
                        lista = park.statVetture();
                        for(Vettura a : lista)
                        {
                              listaAuto = listaAuto + "\n" + a.getTarga() +
                              " con " + a.getOrePagate() + " ore pagate.";
                        }
                        JOptionPane.showMessageDialog(null, listaAuto);break;
                  }
            }
      }
}
```

Distribuzione (appello settembre 2007)
http://www.dei.unipd.it/~satta/teach/java/page/temi/sett3.pdf
punto 5

Una società commerciale desidera gestire in modo elettronico i propri magazzini sul territorio nazionale. Si consideri una classe Articolo con variabili d'istanza codice (stringa) e quantitativo (intero). Si assuma siano già

disponibili i metodi d'accesso ed un costruttore per tale classe (non sviluppare il codice). Si considerinno le seguenti classi:

```
public class Magazzino {
private String citta;
private ArrayList<Articolo> articoli;
public void aggiungiArticolo(String unCodice, int quant);
public boolean rimuoviArticolo(String unCodice, int quant);
}
public class Distribuzione {
private Magazzino[] magazzini;
private int numeroMagazzini;
public Distribuzione(int maxNumeroMagazzini){...}
public boolean aggiungiMagazzino(String unaCitta){...}
public boolean eliminaMagazzino(String unaCitta){...}
public int totaleArticolo(String unCodice){...}
public ArrayList<Magazzino> daRifornire(){...}
public void rifornisciMagazzino(String unaCitta, String unCodice, int
quant){...}
public Articolo stat(){...}
}
```

Nella classe Distribuzione, il metodo totaleArticolo conteggia la quantità totale dell'articolo indicato su tutti i magazzini. Il metodo daRifornire restituisce una arraylist con tutti i magazzini che hanno almeno un articolo con quantitativo zero. Il metodo rifornisciMagazzino rifornisce il magazzino indicato, prelevando la quantità necessaria dell'articolo indicato dagli altri magazzini. Il metodo stat restituisce un riferimento all'oggetto articolo che ha il massimo quantitativo totale su tutti i magazzini (risolvere arbitrariamente i casi di parità). Sviluppare tutti i metodi delle classi Magazzino e Distribuzione.

Consigli:

Nello sviluppo proposto sono stati implementati alcuni metodi non esplicitamente richiesti dalla consegna: nella classe Magazzino il metodo trovaArticolo() è utile come metodo ausiliario per semplificare gli altri metodi, mentre i metodi accessori getArticoli() e getCitta() permettono di sviluppare i metodi della classe Distribuzione. In quest'ultima classe è stato aggiunto trovaMagazzino(), sempre come metodo ausiliario, mentre gli ultimi due metodi sono stati sviluppati solo per facilitare la stesura del tester (vedi nota sopra ai metodi).

Articolo.java

```
public class Articolo
{
	private String codice;
	private int quantitativo;

	public Articolo(String unCodice, int unQuantitativo)
	{
		codice = unCodice;
		quantitativo = unQuantitativo;
	}

	public void aggiungiQuantitativo(int unQuantitativo)
	{
		quantitativo = quantitativo + unQuantitativo;
	}

	public int prelevaQuantitativo(int unQuantitativo)
	{
		if(quantitativo > unQuantitativo)
		{
			quantitativo = quantitativo - unQuantitativo;
```

```
                return unQuantitativo;
            }
            else
            {
                int prelevato = quantitativo;
                quantitativo = 0;
                return prelevato;
            }
        }

        public String getCodice()
        {
            return codice;
        }

        public int getQuantitativo()
        {
            return quantitativo;
        }
}
```

Magazzino.java

```
import java.util.ArrayList;

public class Magazzino
{
        private String citta;
        private ArrayList<Articolo> articoli;

        public Magazzino(String unaCitta)
        {
            citta = unaCitta;
            articoli = new ArrayList<Articolo>();
        }

        public void aggiungiArticolo(String unCodice, int unQuantitativo)
        {
            if(trovaArticolo(unCodice) != -1)
                articoli.get(trovaArticolo(unCodice)).
                aggiungiQuantitativo(unQuantitativo);
            else
                articoli.add(new Articolo(unCodice, unQuantitativo));
        }

        public int trovaArticolo(String unCodice)
        {
            for(int i = 0; i < articoli.size(); i++)
                if(articoli.get(i).getCodice().equalsIgnoreCase(unCodice))
                    return i;
            return -1;
        }

        public boolean rimuoviArticolo(String unCodice, int unQuantitativo)
        {
            if(trovaArticolo(unCodice) == -1) return false;
            else if(articoli.get(trovaArticolo(unCodice)).getQuantitativo() <
                    unQuantitativo)
                return false;
            articoli.get(trovaArticolo(unCodice)).
                prelevaQuantitativo(unQuantitativo);
            return true;
        }
```

```
	public String getCitta()
	{
		return citta;
	}

	public ArrayList<Articolo> getArticoli()
	{
		return articoli;
	}
}
```

Distribuzione.java

```
import java.util.ArrayList;

public class Distribuzione
{
	private Magazzino[] magazzini;
	private int numMag;

	public Distribuzione(int n)
	{
		numMag = 0;
		magazzini = new Magazzino[n];
	}

	public boolean aggiungiMagazzino(String unaCitta)
	{
		if(numMag == magazzini.length || trovaMagazzino(unaCitta) != -1)
			return false;
		magazzini[numMag] = new Magazzino(unaCitta);
		numMag++;
		return true;
	}

	private int trovaMagazzino(String unaCitta)
	{
		for(int i = 0; i < numMag; i++)
			if(magazzini[i].getCitta().equalsIgnoreCase(unaCitta))
				return i;
		return -1;
	}

	public boolean eliminaMagazzino(String unaCitta)
	{
		int i = trovaMagazzino(unaCitta);
		if(i == -1) return false;
		System.arraycopy(magazzini, i + 1, magazzini, i, magazzini.length -
			i - 1);
		numMag--;
		return true;
	}

	public int totaleArticolo(String unCodice)
	{
		int result = 0;
		for(int i = 0; i < numMag; i++)
		{
			int indiceArt = magazzini[i].trovaArticolo(unCodice);
			if(indiceArt != -1)
				result = result + magazzini[i].getArticoli().
```

```
                        get(indiceArt).getQuantitativo();
        }
        return result;
}

public ArrayList<Magazzino> daRifornire()
{
        ArrayList<Magazzino> result = new ArrayList<Magazzino>();
        for(int i = 0; i < numMag; i++)
        {
                boolean rifornibile = false;
                ArrayList<Articolo> daControllare = magazzini[i].
                        getArticoli();
                for(Articolo a : daControllare)
                        if(a.getQuantitativo() == 0)
                                rifornibile = true;
                if(rifornibile)
                        result.add(magazzini[i]);
        }
        return result;
}

public void rifornisciMagazzino(String unaCitta, String unCodice, int
        unQuantitativo)
{
        int daRifornire = trovaMagazzino(unaCitta);
        int daPrelevare = unQuantitativo;
        if(daRifornire == -1) return;
        for(int i = 0; i < numMag; i++)
        {
                if(i != daRifornire)
                {
                        int trovArt = magazzini[i].trovaArticolo(unCodice);
                        if(trovArt != -1 && daPrelevare > 0)
                        {
                                int prelevato = magazzini[i].getArticoli().
                                    get(trovArt).prelevaQuantitativo(daPrelevare);
                                daPrelevare = daPrelevare - prelevato;
                                magazzini[daRifornire].aggiungiArticolo(unCodice,
                                    prelevato);
                        }
                }
        }
}

public Articolo stat()
{
        Articolo massimo = null;
        int max = 0;
        for(int i = 0; i < numMag; i++)
        {
                ArrayList<Articolo> provv = magazzini[i].getArticoli();
                for(Articolo a : provv)
                {
                        int n = totaleArticolo(a.getCodice());
                        if(n > max)
                        {
                                max = n;
                                massimo = a;
                        }
                }
        }
        return massimo;
```

```
	}

	/*I due metodi seguenti non sono richiesti dalla consegna dell'esercizio,
	ma sono stati sviluppati esclusivamente per facilitare la stesura del
	tester. Infatti le operazioni che eseguono si possono ottenere anche con i
	metodi delle classi Magazzino e Articolo direttamente dal tester.*/
	public boolean aggiungiArticolo(String unaCitta, String unCodice, int
		unQuantitativo)
	{
		int indiceMag = trovaMagazzino(unaCitta);
		if(indiceMag != -1)
		{
			magazzini[indiceMag].aggiungiArticolo(unCodice,
				unQuantitativo);
			return true;
		}
		else
			return false;
	}

	public boolean rimuoviArticolo(String unaCitta, String unCodice, int
		unQuantitativo)
	{
		int indiceMag = trovaMagazzino(unaCitta);
		if(indiceMag != -1)
		{
			if(magazzini[indiceMag].rimuoviArticolo(unCodice,
				unQuantitativo))
				return true;
			else
				return false;
		}
		else
			return false;
	}
}
```

DistribuzioneTester.java

```
import javax.swing.*;
import java.util.ArrayList;

public class DistribuzioneTester
{
	public static void main(String args[])
	{
		int numMag = Integer.parseInt(JOptionPane.showInputDialog
			("Inserisci il numero massimo di magazzini da gestire:"));
		Distribuzione dist = new Distribuzione(numMag);
		int numScelta, quantita;
		String nomeCitta, codiceArt;
		numScelta = 0;
		while(numScelta != 9)
		{
			numScelta = Integer.parseInt(JOptionPane.showInputDialog
				("Operazioni eseguibili:\n
				1) Aggiungi magazzino\n
				2) Elimina magazzino\n
				3) Aggiungi articolo a un magazzino\n
				4) Rimuovi articlo da un magazzino\n
				5) Quantità totale di un articolo\n
				6) Visualizza elenco magazzini da rifornire\n
				7) Rifornisci un magazzino prelevando dagli altri\n
```

```
      8) Visualizza articolo con quantitativo più alto\n
      9) Esci"));
switch(numScelta)
{
case 1: nomeCitta = JOptionPane.showInputDialog("Inserire il
      nome della sede del nuovo magazzino:");
      if(!dist.aggiungiMagazzino(nomeCitta))
            JOptionPane.showMessageDialog(null,"Attenzione!
            Impossibile aggiungere nuovo magazzino.");
      else
            JOptionPane.showMessageDialog(null,"Nuovo
            magazzino aperto a " + nomeCitta);break;
case 2: nomeCitta = JOptionPane.showInputDialog("Inserire il
      nome della sede del magazzino:");
      if(!dist.eliminaMagazzino(nomeCitta))
            JOptionPane.showMessageDialog(null,"Attenzione!
            Impossibile eliminare il magazzino.");
      else
            JOptionPane.showMessageDialog(null,"Il magazzino
            di " + nomeCitta + " è stato eliminato.");break;
case 3: nomeCitta = JOptionPane.showInputDialog("Inserire il
      nome della sede del magazzino:");
      codiceArt = JOptionPane.showInputDialog("Inserire il
      codice del prodotto:");
      quantita = Integer.parseInt(JOptionPane.showInputDialog
            ("Inserire la quantità fornita:"));
      if(!dist.aggiungiArticolo(nomeCitta,codiceArt,quantita))
            JOptionPane.showMessageDialog(null,"Attenzione!
            Impossibile aggiungere articolo.");
      else
            JOptionPane.showMessageDialog(null,"Articolo
            aggiunto");break;
case 4: nomeCitta = JOptionPane.showInputDialog("Inserire il
      nome della sede del magazzino:");
      codiceArt = JOptionPane.showInputDialog("Inserire il
      codice del prodotto:");
      quantita = Integer.parseInt(JOptionPane.showInputDialog
            ("Inserire la quantità da rimuovere:"));
      if(!dist.rimuoviArticolo(nomeCitta,codiceArt,quantita))
            JOptionPane.showMessageDialog(null,"Attenzione!
            Impossibile rimuovere articolo.");
      else
            JOptionPane.showMessageDialog(null,"Articolo
            rimosso");break;
case 5: codiceArt = JOptionPane.showInputDialog("Inserire il
      codice del prodotto:");
      JOptionPane.showMessageDialog(null, "Sono presenti " +
      dist.totaleArticolo(codiceArt) + " pezzi.");break;
case 6: String elenco = "Magazzini da rifornire:";
      for(Magazzino m : dist.daRifornire())
            elenco = elenco + "\n" + m.getCitta();
      JOptionPane.showMessageDialog(null, elenco);break;
case 7: nomeCitta = JOptionPane.showInputDialog("Inserire il
      nome della sede del magazzino:");
      codiceArt = JOptionPane.showInputDialog("Inserire il
      codice del prodotto:");
      quantita = Integer.parseInt(JOptionPane.showInputDialog
      ("Inserire la quantità da rifornire:"));
      dist.rifornisciMagazzino(nomeCitta,codiceArt,quantita);
      JOptionPane.showMessageDialog(null,"Magazzino
      rifornito");break;
case 8: JOptionPane.showMessageDialog(null,"L'articolo con
      quantitativo maggiore è il cod." + dist.stat().
```

```
                        getCodice());break;
                }
            }
        }
    }
```

Indice: