Esercizi e progetti di programmazione

Gli esercizi relativi alla grafica e al collaudo (*testing*) sono evidenziati mediante, rispettivamente, la lettera G o T.

Gli asterischi indicano il grado di difficoltà.

Capitolo 1

Esercizi di programmazione

★ **Esercizio P1.1.** Scrivete un programma `NamePrinter` che visualizzi nella finestra di comandi il vostro nome all'interno di un rettangolo, come nell'esempio seguente:

```
+----+
¦Dave¦
+----+
```

Fate quanto possibile per comporre i lati del rettangolo con i caratteri ¦ - +.

★★★ **Esercizio P1.2.** Scrivete un programma che scriva il vostro nome con lettere molto grandi, in questo modo:

```
*   *    **    ****    ****    *   *
*   *   *  *   *   *   *   *   *   *
*****  *    *  ****    ****     * *
*   *  ******  *   *   *   *     *
*   *  *    *  *   *   *   *     *
```

★ **Esercizio P1.3.** Scrivete un programma `FacePrinter` che, usando caratteri, visualizzi un viso, possibilmente migliore di questo:

```
  /////
 ¦ o o ¦
(¦  ^  ¦)
 ¦ \_/ ¦
  -----
```

Usate *commenti* per indicare quando vengono visualizzati i capelli, le orecchie, la bocca, e così via.

★★★ **Esercizio P1.4.** Scrivete un programma che visualizzi un animale mentre pronuncia un saluto, simile (ma diverso) da questo:

```
 /\_/\     -----
( ' ' )  / Hello \
(  -  ) <  Junior ¦
 ¦ ¦ ¦   \ Coder!/
(__¦__)    -----
```

★ **Esercizio P1.5.** Scrivete un programma `TicTacToeBoardPrinter` che visualizzi una scacchiera per giocare a tris (*tic-tac-toe*):

```
+---+---+---+
¦   ¦   ¦   ¦
+---+---+---+
¦   ¦   ¦   ¦
+---+---+---+
¦   ¦   ¦   ¦
+---+---+---+
```

★ **Esercizio P1.6.** Scrivete un programma `StaircasePrinter` che visualizzi una scala:

```
            +---+
            ¦   ¦
        +---+---+
        ¦   ¦   ¦
    +---+---+---+
    ¦   ¦   ¦   ¦
+---+---+---+---+
¦   ¦   ¦   ¦   ¦
+---+---+---+---+
```

★ **Esercizio P1.7.** Scrivete un programma che, su tre righe consecutive, visualizzi tre stringhe, ad esempio i nomi dei vostri migliori amici o i vostri film preferiti.

★★ **Esercizio P1.8.** Scrivete un programma che calcoli e visualizzi la somma dei primi 10 numeri interi positivi: 1 + 2 + … + 10. *Suggerimento:* Utilizzate un programma con questa struttura.

```
public class Sum10
{
   public static void main(String[] args)
   {
       System.out.println(        );
   }
}
```

★★ **Esercizio P1.9.** Copiate ed eseguite il programma seguente:

```
import javax.swing.JOptionPane;

public class DialogViewer
{
```

```
   public static void main(String[] args)
   {
      JOptionPane.showMessageDialog(null, "Hello, World!");
      System.exit(0);
   }
}
```

Modificate poi il programma in modo che visualizzi il messaggio "Hello, *vostro nome*!".

** **Esercizio P1.10.** Copiate ed eseguite il programma seguente:

```
import javax.swing.JOptionPane;

public class DialogViewer
{
   public static void main(String[] args)
   {
      String name = JOptionPane.showInputDialog("What is your name?");
      System.out.println(name);
      System.exit(0);
   }
}
```

Modificate poi il programma in modo che visualizzi il messaggio "Hello, *nome*!", essendo *nome* ciò che è stato digitato dall'utente nella finestra di dialogo.

** **Esercizio P1.11.** Eseguite il programma seguente:

```
import java.net.URL;
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;

public class ImageGreeter
{
   public static void main(String[] args)
   {
      URL imageLocation = new URL(
         "http://horstmann.com/bigjava/duke.gif");
      JOptionPane.showMessageDialog(null, "Hello", "Title",
         JOptionPane.PLAIN_MESSAGE, new ImageIcon(imageLocation));
      System.out.println(name);
      System.exit(0);
   }
}
```

Modificate poi il programma in modo che visualizzi un diverso saluto e una diversa immagine.

Progetti di programmazione

Progetto 1.1. Questo progetto si basa sugli Esercizi P1.9 e P1.10. Il vostro programma deve leggere il nome dell'utente, poi visualizzare due finestre di dialogo in sequenza:

- Per prima cosa, una finestra con inserimento dati che chieda: "Cosa vorresti che facessi?"
- Poi, una finestra che visualizzi il messaggio: "Mi dispiace, *nome*. Non posso farlo."

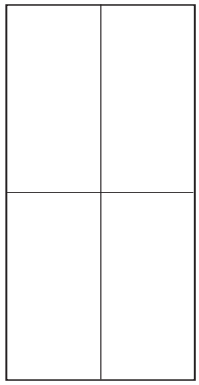
Capitolo 2

Esercizi di programmazione

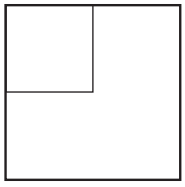
*T **Esercizio P2.1.** Scrivete un programma `AreaTester` che costruisca un oggetto `Rectangle`, ne calcoli l'area e la visualizzi. Usate i metodi `getWidth` e `getHeight` e visualizzate anche il valore previsto.

*T **Esercizio P2.2.** Scrivete un programma `PerimeterTester` che costruisca un oggetto `Rectangle`, ne calcoli il perimetro e lo visualizzi. Usate i metodi `getWidth` e `getHeight` e visualizzate anche il valore previsto.

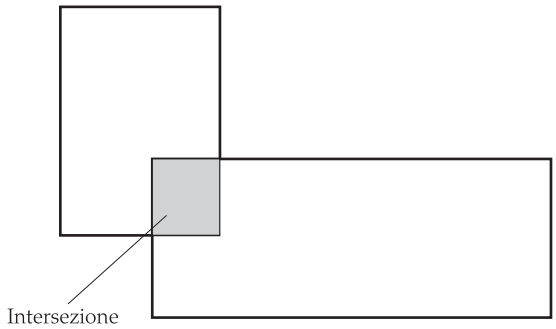
** **Esercizio P2.3.** Scrivete un programma `FourRectanglePrinter` che costruisca un oggetto `Rectangle`, visualizzi la sua posizione invocando `System.out.println(box)` e, quindi, lo sposti e ne visualizzi la posizione per altre tre volte, in modo che, se i rettangoli fossero disegnati, formerebbero un unico grande rettangolo, qui visibile:



** **Esercizio P2.4.** Scrivete un programma `GrowSquarePrinter` che costruisca un oggetto `Rectangle`, memorizzato nella variabile `square`: un quadrato con i lati di lunghezza 50 e con l'angolo superiore sinistro nel punto (100, 100). Il programma deve, poi, visualizzare la posizione del quadrato invocando `System.out.println(square)` e, quindi, invocare i metodi `translate` e `grow`; infine, va invocato di nuovo `System.out.println(square)`. Le invocazioni di `translate` e `grow` devono modificare il quadrato in modo che la dimensione del suo lato raddoppi, senza spostare la posizione del suo angolo superiore sinistro. Se i quadrati venissero disegnati, darebbero luogo alla figura qui visibile:



*** **Esercizio P2.5.** Il metodo `intersection` calcola l'*intersezione* di due rettangoli, ovvero il rettangolo formato dalla sovrapposizione parziale di altri due rettangoli.



Il metodo viene invocato in questo modo:

```
Rectangle r3 = r1.intersection(r2);
```

Scrivete un programma `IntersectionPrinter` che costruisca due rettangoli, li visualizzi come descritto nell'Esercizio P2.3 e, quindi, visualizzi allo stesso modo il rettangolo che rappresenta la loro intersezione. Il programma visualizza il risultato anche quando i rettangoli non si sovrappongono: aggiungete un commento al codice che spieghi come si può capire se tale rettangolo risultante è vuoto.

*** **Esercizio P2.6.** In questo esercizio scoprirete una semplice modalità di visualizzazione di un oggetto di tipo `Rectangle`. Il metodo `setBounds` della classe `JFrame` sposta un frame in modo che posizioni il proprio bordo in un rettangolo assegnato. Completate il programma che segue in modo che illustri visivamente il funzionamento del metodo `translate` della classe `Rectangle`.

```
import java.awt.Rectangle;
import javax.swing.JFrame;
import javax.swing.JOptionPane:

public class TranslateDemo
{
   public static void main(String[] args)
   {
      // costruisce un frame e lo visualizza
      JFrame frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);

      // qui dovete scrivere voi:
      // costruite un rettangolo e impostate la posizione del bordo del frame

      JOptionPane.showMessageDialog(frame, "Click OK to continue");

      // qui dovete scrivere voi:
      // spostate il rettangolo e reimpostate la posizione del bordo del frame
   }
}
```

** **Esercizio P2.7.** Nella libreria Java, un colore viene specificato mediante le sue tre componenti (rosso, verde e blu), con valori numerici compresi tra 0 e 255, come visto nella Tabella 4. Scrivete un programma `BrighterDemo` che costruisca un oggetto di tipo `Color` con i valori di rosso, verde e blu rispettivamente uguali a 50, 100 e 150. Successivamente, applicate il metodo `brighter` e

visualizzate i valori delle tre componenti del colore risultante (non vedrete veramente il colore: per visualizzare i colori, svolgete il prossimo esercizio).

** **Esercizio P2.8.** Risolvete nuovamente l'Esercizio precedente, ma inserite il vostro codice nella classe qui riportata: in questo modo il colore verrà visualizzato veramente.

```
import java.awt.Color;
import javax.swing.JFrame;

public class BrighterDemo
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();
      frame.setSize(200, 200);
      Color myColor = …;
      frame.getContentPane().setBackground(myColor);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
   }
}
```

** **Esercizio P2.9.** Risolvete nuovamente l'Esercizio P2.7, ma applicate due volte il metodo `darken` al colore predefinito `Color.RED`. Chiamate il vostro programma `DarkerDemo`.

** **Esercizio P2.10.** La classe `Random` realizza un *generatore di numeri casuali*, cioè genera sequenze di numeri che appaiono essere casuali. Per generare numeri interi casuali, dovete costruire un oggetto della classe `Random`, a cui applicare poi il metodo `nextInt`. Ad esempio, l'invocazione `generator.nextInt(6)` fornisce un numero casuale compreso tra 0 e 5.

Scrivete un programma `DieSimulator` 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.

*** **Esercizio P2.11.** Scrivete un programma `LotteryPrinter` che generi una combinazione vincente per una lotteria, nella quale i giocatori possono scegliere 6 numeri (eventualmente ripetuti) compresi fra 1 e 49 (in una vera lotteria le ripetizioni non sono ammesse, ma non abbiamo ancora presentato gli strumenti di programmazione che sarebbero necessari per gestire questo problema). Il vostro programma deve visualizzare una frase beneaugurante, del tipo "Ecco la combinazione che ti farà ricco", seguita da una combinazione per la lotteria.

T **Esercizio P2.12. Scrivete un programma `ReplaceTester` che codifichi una stringa sostituendo, mediante il metodo `replace`, tutte le lettere `"i"` con `"!"` e tutte le lettere `"s"` con `"$"`. Fate vedere che riuscite a codificare correttamente la stringa `"Mississippi"`, visualizzando il risultato prodotto e quello previsto.

*** **Esercizio P2.13.** Scrivete un programma `HollePrinter` che scambi tra loro le lettere `"e"` e `"o"` in una stringa, usando ripetutamente il metodo `replace`. Fate vedere che la stringa `"Hello, World!"` si trasforma in `"Holle, Werld!"`.

G **Esercizio P2.14. Scrivete un programma grafico per disegnare il vostro nome in rosso, centrandolo all'interno di un rettangolo blu. Progettate le classi `NameViewer` e `NameComponent`.

G **Esercizio P2.15. Scrivete un programma grafico che disegni dodici stringhe, una per ciascuno dei dodici colori predefiniti (eccetto `Color.WHITE`): ciascuna stringa sia uguale al nome di un colore e venga visualizzata nel proprio colore. Progettate le classi `ColorNameViewer` e `ColorNameComponent`.

****G** **Esercizio P2.16.** Scrivete un programma che disegni due quadrati colorati internamente, uno rosa e uno viola. Usate un colore standard per il primo e un colore personalizzato per l'altro. Progettate le classi `TwoSquareViewer` e `TwoSquareComponent`.

*****G** **Esercizio P2.17.** Scrivete un programma che riempia una finestra con una grande ellisse, che tocchi i bordi della finestra e che sia riempita con il vostro colore preferito, avendo però il contorno nero. L'ellisse deve ridimensionarsi automaticamente quando si ridimensiona la finestra.

****G** **Esercizio P2.18. P**rogettando le classi `FaceViewer` e `FaceComponent`, scrivete un programma per tracciare la faccia qui visibile.

Progetti di programmazione

Progetto 2.1. La classe `GregorianCalendar` descrive un istante nel tempo, misurato secondo il calendario gregoriano, che è il calendario adottato oggi come standard in tutto il mondo. Si costruisce un oggetto di tipo `GregorianCalendar` usando come parametri un anno, un mese e un giorno del mese, in questo modo:

```
GregorianCalendar cal = new GregorianCalendar(); // la data di oggi
GregorianCalendar eckertsBirthday = new GregorianCalendar(1919,
      Calendar.APRIL, 9);
```

Per specificare il mese usate le costanti `Calendar.JANUARY` ... `Calendar.DECEMBER`.
Per aggiungere un certo numero di giorni a una data rappresentata da un oggetto di tipo `GregorianCalendar` si può usare il metodo `add`:

```
cal.add(Calendar.DAY_OF_MONTH, 10);
      // ora cal rappresenta il decimo giorno nel futuro a partire da oggi
```

Si tratta di un metodo modificatore: modifica l'oggetto `cal`.
Per ottenere informazioni da un oggetto di tipo `GregorianCalendar` si può usare il metodo `get`:

```
int dayOfMonth = cal.get(Calendar.DAY_OF_MONTH);
int month = cal.get(Calendar.MONTH);
int year = cal.get(Calendar.YEAR);
int weekday = cal.get(Calendar.DAY_OF_WEEK);
      // 1 rappresenta domenica, 2 lunedì, ..., 7 sabato
```

Il vostro compito consiste nella realizzazione di un programma che visualizzi le seguenti informazioni:

- La data e il giorno della settimana che dista 100 giorni da oggi nel futuro.
- Il giorno della settimana della vostra data di nascita.
- La data che dista 10 000 giorni nel futuro dalla vostra data di nascita.

Se non volete rendere nota la vostra data di nascita, usate quella di uno scienziato informatico.

Progetto 2.2. Eseguite il programma seguente:

```
import java.awt.Color;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class FrameViewer
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();
      frame.setSize(200, 200);
      JLabel label = new JLabel("Hello, World!");
      Label.setOpaque(true);
      label.setBackground(Color.PINK);
      frame.add(label);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
   }
}
```

Ora, modificatelo nel modo seguente:

- Raddoppiate la dimensione del frame
- Fate in modo che il messaggio di saluto diventi "Hello, *vostro nome*!"
- Fate in modo che il colore dello sfondo ("background") diventi verde chiaro (rivedete l'Esercizio P2.7)
- Per ottenere una valutazione migliore, aggiungete un'immagine di voi stessi. *Suggerimento*: costruite un oggetto di tipo `ImageIcon`.

Capitolo 3

Esercizi di programmazione

*T **Esercizio P3.1.** Scrivete una classe `BankAccountTester` il cui metodo `main` costruisca un conto bancario, effettui un versamento di $ 1000 seguito dai prelievi di $ 500 e $ 400 e, infine, visualizzi il saldo rimanente, seguito dal suo valore previsto.

* **Esercizio P3.2.** Aggiungete alla classe `BankAccount` un metodo

```
public void addInterest(double rate)
```

che aggiunga al saldo del conto gli interessi, calcolati al tasso specificato. Ad esempio, dopo l'esecuzione di questi enunciati

```
BankAccount momsSavings = new BankAccount(1000);
momsSavings.addInterest(10); // interessi al 10%
```

il saldo di `momsSavings` è $ 1100. Progettate anche una classe `BankAccountTester` che visualizzi il saldo finale e il suo valore previsto.

** **Esercizio P3.3.** Scrivete una classe `SavingsAccount` (*conto di risparmio*), del tutto simile alla classe `BankAccount`, tranne per la presenza di un ulteriore variabile di esemplare, `interest`. Fornite un costruttore che assegni un valore sia al saldo iniziale sia al tasso di interesse. Fornite anche un

metodo `addInterest` (privo di parametri espliciti) che aggiunga gli interessi al conto. Scrivete poi una classe `SavingsAccountTester` che costruisca uno di tali "conti di risparmio" con saldo iniziale di $ 1000 e tasso di interesse del 10%. Applicate, infine, per cinque volte il metodo `addInterest`, visualizzando il saldo finale e il suo valore previsto, dopo averlo calcolato manualmente.

*** **Esercizio P3.4.** Dotate la classe `CashRegister` della capacità di calcolare le tasse relative alla vendita effettuata, con il tasso di interesse fornito al costruttore della classe. Aggiungete i metodi `recordTaxablePurchase` e `getTotalTax` (quanto viene aggiunto con `recordPurchase` non è soggetto a tassazione). Il metodo `giveChange` deve tener conto in modo corretto delle tasse applicate agli acquisti soggetti a tassazione.

** **Esercizio P3.5.** Dopo la chiusura, il gestore del negozio vorrebbe conoscere il volume totale di vendite effettuate nella giornata: modificate la classe `CashRegister` in modo che lo possa fare, aggiungendo i metodi `getSalesTotal` e `getSalesCount`, che restituiscono l'incasso totale e il numero totale di vendite effettuate, oltre al metodo `reset` che azzera tutto, in modo che le operazioni funzionino correttamente il giorno successivo.

** **Esercizio P3.6.** Realizzate una classe `Employee` (*dipendente*). Ogni dipendente ha un nome (una stringa) e uno stipendio (di tipo `double`). Scrivete un costruttore con due parametri

```
public Employee(String employeeName, double currentSalary)
```

e i metodi

```
public String getName()
public double getSalary()
public void raiseSalary(double byPercent)
```

Tali metodi forniscono il nome e lo stipendio del dipendente e ne aumentano il salario della percentuale indicata. Ecco un esempio di utilizzo:

```
Employee harry = new Employee("Hacker, Harry", 50000);
harry.raiseSalary(10); // Harry ottiene un aumento del 10%
```

Progettate anche una classe `EmployeeTester` che collaudi tutti i metodi.

** **Esercizio P3.7.** Realizzate una classe `Car` (*automobile*). Un'automobile è caratterizzata da un consumo di carburante (misurato in miglia/gallone o in litri/chilometro, a vostra scelta) e una certa quantità di carburante nel serbatoio. Il consumo è specificato nel costruttore e inizialmente il serbatoio è vuoto. Progettate: un metodo `drive` per simulare il percorso di un'automobile per una determinata distanza, riducendo conseguentemente il livello di carburante nel suo serbatoio; un metodo `getGasInTank`, per ispezionare il livello del carburante; un metodo `addGas`, per fare rifornimento. Ecco un esempio di utilizzo:

```
Car myHybrid = new Car(50); // 50 miglia per gallone
myHybrid.addGas(20); // aggiungi 20 galloni di carburante
myHybrid.drive(100); // viaggia per 100 miglia
double gasLeft = myHybrid.getGasInTank()); // quantità di carburante rimasto
```

Potete ipotizzare che il metodo `drive` non venga mai invocato per una distanza maggiore di quella percorribile con il carburante disponibile. Progettate anche una classe `CarTester` che collaudi tutti i metodi.

** **Esercizio P3.8.** Realizzate una classe `Student` (*studente*). Ai fini di questo esercizio, ciascuno studente ha un nome e un punteggio totale per i questionari a cui ha risposto. Progettate un costruttore appropriato e i metodi seguenti: `getName()`, per conoscere il nome dello studente; `addQuiz(int score)`, per registrare il punteggio di un nuovo questionario; `getTotalScore()`, per conoscere il punteggio totale; `getAverageScore()`, per ottenere la media dei punteggi. Per calcolare tale media dovete registrare anche il *numero dei questionari* a cui lo studente ha dovuto rispondere. Progettate anche una classe `StudentTester` che collaudi tutti i metodi.

* **Esercizio P3.9.** Realizzate una classe `Product` (*prodotto*). Ciascun prodotto ha un nome e un prezzo, descritti nel costruttore: `new Product("Tostapane", 29.95)`. Progettate i seguenti metodi: `getName()`, per conoscere il nome del prodotto; `getPrice()`, per conoscerne il prezzo; `reducePrice()`, per scontarne il prezzo. Scrivete un programma che crea due prodotti e ne stampa il nome e il prezzo, per poi scontarne i prezzi di $ 5.00 e stamparli nuovamente.

** **Esercizio P3.10.** Realizzate una classe che impagini una semplice lettera. Il costruttore riceve come parametri il nome del mittente (*from*) e quello del destinatario (*to*):

```
public Letter(String from, String to)
```

Progettate un metodo per aggiungere una riga di testo al contenuto della lettera, in fondo:

```
public void addLine(String line)
```

Progettate un altro metodo che restituisca l'intero testo della lettera:

```
public String getText()
```

Il testo della lettera ha il seguente formato:

Dear *destinatario*:
riga vuota
prima riga del contenuto della lettera
seconda riga del contenuto della lettera
…
ultima riga del contenuto della lettera
riga vuota
Sincerely,
riga vuota
mittente

Progettate anche un programma `LetterPrinter` che visualizzi questa lettera:

```
Dear John:

I am sorry we must part.
I wish you all the best.

Sincerely,

Mary
```

costruendo un esemplare della classe `Letter` e invocando due volte il metodo `addLine`.

Suggerimenti: (1) Usate il metodo `concat` per costruire una stringa più lunga a partire da due stringhe più corte. (2) La stringa speciale `"\n"` rappresenta il carattere speciale che si usa per andare a capo. Ad esempio, il seguente enunciato

```
body = body.concat("Sincerely,").concat("\n");
```

aggiunge al contenuto della lettera la riga `"Sincerely,"`.

** **Esercizio P3.11.** Realizzate una classe `Bug` che rappresenti un insetto che si sposta lungo una linea orizzontale, verso sinistra o verso destra. Inizialmente si sposta verso destra, ma può cambiare direzione; ogni volta che si sposta, la sua posizione lungo la linea cambia di un'unità verso la direzione più recente. Dotate la classe di un costruttore

```
public Bug(int initialPosition)
```

e dei metodi

```
public void turn()
public void move()
public int getPosition()
```

Ecco un esempio di utilizzo:

```
Bug bugsy = new Bug(10);
bugsy.move(); // ora si trova nella posizione 11
bugsy.turn(); // cambia direzione
bugsy.move(); // ora si trova nella posizione 10
```

La classe `BugTester` deve costruire un insetto, farlo muovere e girare alcune volte, poi visualizzarne la posizione effettiva e quella prevista.

** **Esercizio P3.12.** Realizzate una classe `Moth` che rappresenti una falena che si sposta lungo una linea retta, ricordando la propria posizione e la distanza da un'origine prefissata. Quando si sposta verso una sorgente luminosa, la sua nuova posizione viene a trovarsi a metà strada tra quella precedente e la posizione della sorgente luminosa. Dotate la classe di un costruttore

```
public Moth(double initialPosition)
```

e dei metodi

```
public void moveToLight(double lightPosition)
public double getPosition()
```

La classe `MothTester` deve costruire una falena, farla muovere verso un paio di sorgenti luminose, poi visualizzarne la posizione effettiva e quella prevista.

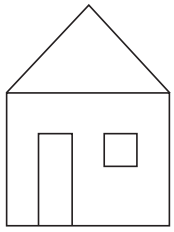
** **Esercizio P3.13.** Realizzate una classe `RoachPopulation` che simuli la crescita di una popolazione di scarafaggi. Il costruttore riceve la dimensione della popolazione iniziale di scarafaggi. Il metodo `breed` simula un periodo di tempo in cui la popolazione raddoppia. Il metodo `spray` simula una spruzzata di insetticida, che riduce la popolazione del 10%. Il metodo `getRoaches` restituisce il numero attuale di scarafaggi. Realizzate un programma di collaudo `RoachSimulation` che simuli una popolazione che inizia con 10 scarafaggi. Raddoppiate la popolazione, spruzzate l'insetticida e stampate il numero di scarafaggi. Ripetete la procedura altre tre volte.

****** **Esercizio P3.14.** Realizzate una classe `VotingMachine` che possa essere utilizzata per una semplice elezione, con i metodi per azzerare inizialmente il conteggio dei voti, per assegnare un voto ai Democratici, per assegnare un voto ai Repubblicani e per conoscere il numero totale di voti per ciascuno dei due partiti. Otterrete una valutazione migliore se il vostro programma, nel caso in cui la situazione sia di parità dopo le ore 20 del primo martedì di Novembre, è in grado di assegnare la vittoria al partito che preferite, pur funzionando correttamente in ogni altro momento dell'anno. *Suggerimento*: usate la classe `GregorianCalendar`, già vista nel Progetto di programmazione 2.1.

****G** **Esercizio P3.15.** Disegnate un bersaglio ("bull's eye"), vale a dire una serie di cerchi concentrici, alternando i colori bianco e nero. *Suggerimento:* Colorate un cerchio nero, quindi sovrapponete un cerchio bianco più piccolo, e così via.

Il programma dovrebbe essere costituito dalle classi `BullsEye`, `BullsEyeComponent` e `BullsEyeViewer`.

****G** **Esercizio P3.16.** Scrivete un programma che tracci lo schizzo di una casa, semplice come la figura qui riportata oppure più elaborata, se preferite (potete utilizzare una vista prospettica, disegnare un grattacielo, colonne di marmo nell'ingresso o qualsiasi cosa).



Realizzate una classe `House` dotata di un metodo `draw(Graphics2D g2)` che disegni la casa.

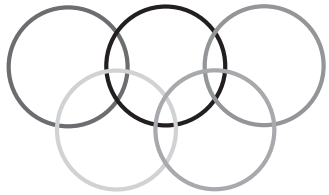
****G** **Esercizio P3.17.** Migliorate l'esercizio precedente dotando la classe `House` di un costruttore che consenta di specificarne posizione e dimensione, quindi popolate lo schermo con un po' di case di varie dimensioni.

****G** **Esercizio P3.18.** Modificate il programma che disegna automobili visto nel Paragrafo 3.9 in modo che le automobili vengano visualizzate in colori diversi. Ogni oggetto di tipo `Car` dovrebbe memorizzare il proprio colore. Scrivete le classi `Car` e `CarComponent` modificate.

****G** **Esercizio P3.19.** Modificate la classe `Car` in modo che la dimensione di un'automobile possa essere specificata nel costruttore. Modificate la classe `CarComponent` in modo che una delle automobili abbia dimensioni doppie rispetto alla versione originale.

****G** **Esercizio P3.20.** Scrivete un programma che disegni la stringa "HELLO" utilizzando solo linee e cerchi. Non invocate il metodo `drawString` e non usate `System.out`. Progettate le classi `LetterH`, `LetterE`, `LetterL` e `LetterO`.

****G** **Esercizio P3.21.** Scrivete un programma per visualizzare gli anelli olimpici. Colorate gli anelli con i colori olimpici (blu, nero, rosso, giallo e verde, procedendo da sinistra a destra e dall'alto in basso), progettando le classi `OlympicRingViewer` e `OlympicRingComponent`.



****G Esercizio P3.22.** Costruite un grafico a barre per rappresentare i dati seguenti:

Nome del ponte	Campata massima (in piedi)
Golden Gate	4200
Brooklyn	1595
Delaware Memorial	2150
Mackinac	3800

Disponete le barre orizzontalmente, per facilitare l'inserimento delle etichette in corrispondenza di ciascuna barra, e progettate le classi `BarChartViewer` e `BarChartComponent`.

Progetti di programmazione

Progetto 3.1. Con questo progetto migliorerete la classe `BankAccount` e vedrete come l'astrazione e l'incapsulamento possano portare un cambiamento rivoluzionario nel mondo del software.

Iniziate con un miglioramento molto semplice: addebitate una commissione per ogni versamento e per ogni prelievo. Consentite l'impostazione del valore della commissione e modificate i metodi `deposit` e `withdraw` in modo che venga applicata tale commissione. Collaudate la classe risultante e verificate che i costi delle commissioni vengano gestiti correttamente.

Fate ora una modifica più complessa. La banca consentirà ogni mese un numero fisso di operazioni gratuite (versamenti o prelievi) e applicherà la commissione soltanto alle operazioni eccedenti. Le commissioni non vengono prelevate all'atto della singola operazione, ma al termine di ogni mese.

Progettate e realizzate un nuovo metodo per la classe `BankAccount`, `deductMonthlyCharge`, che sottragga al saldo del conto le commissioni del mese appena trascorso, azzerando il conteggio delle operazioni. *Suggerimento*: usate il metodo `Math.max(`*`effettuate, gratuite`*`)`.

Scrivete un programma di collaudo che verifichi la correttezza del funzionamento della classe simulando operazioni per alcuni mesi.

Progetto 3.2. Con questo progetto realizzerete un'alternativa orientata agli oggetti per il programma "Hello, World!" che avete visto nel Capitolo 1.

Iniziate con una semplice classe `Greeter` dotata di un solo metodo, `sayHello`, che deve *restituire* una stringa, senza visualizzarla. Usate BlueJ per creare due oggetti di tale classe, invocandone i relativi metodi `sayHello`.

Ovviamente si tratta di un'attività alquanto noiosa, dal momento che entrambi gli oggetti generano il medesimo saluto.

Migliorate la classe `Greeter` in modo che ciascun oggetto produca un saluto personalizzabile. Ad esempio, l'oggetto costruito con `new Greeter("Dave")` dovrebbe dire `"Hello, Dave!"` (usate il metodo `concat` per unire due stringhe in modo da formare una stringa più lunga, oppure date un'occhiata al Paragrafo 4.6 per vedere come si possa usare l'operatore `+` per ottenere lo stesso risultato).

Aggiungere alla classe `Greeter` anche un metodo `sayGoodbye`.

Infine, aggiungete alla classe `Greeter` un metodo `refuseHelp`, che restituisca una stringa di questo tipo: `"I am sorry, Dave. I am afraid I can't do that."` Collaudate la vostra classe con BlueJ: costruite oggetti che salutino il mondo e Dave, invocando poi i loro metodi.

Capitolo 4

Esercizi di programmazione

★ **Esercizio P4.1.** Migliorate la classe `CashRegister` aggiungendo i metodi `enterDollars`, `enterQuarters`, `enterDimes`, `enterNickels` e `enterPennies`. Usate questa classe di collaudo:

```
public class CashRegisterTester
{
   public static void main(String[] args)
   {
      CashRegister register = new CashRegister();
      register.recordPurchase(20.37);
      register.enterDollars(20);
      register.enterQuarters(2);
      System.out.println("Change: " + register.giveChange());
      System.out.println("Expected: 0.13");
   }
}
```

★ **Esercizio P4.2.** Migliorate la classe `CashRegister` in modo che tenga traccia del numero complessivo di articoli che compongono un acquisto. Contate tutti gli articoli acquistati e progettate un metodo

```
int getItemCount()
```

che restituisca il numero di articoli che compongono l'acquisto in corso. Ricordatevi di azzerare il conteggio al termine di ogni acquisto.

★★ **Esercizio P4.3.** Realizzate una classe `IceCreamCone` (*cono gelato*) con i metodi `getSurfaceArea()` e `getVolume()`. Nel costruttore specificate l'altezza e il raggio del cono. Fate attenzione nell'utilizzare la formula che calcola l'area superficiale: dovreste considerare solamente l'area esterna laterale del cono, che è aperto in cima per contenere il gelato.

★★ **Esercizio P4.4.** Scrivete un programma che chieda all'utente di fornire due numeri e che poi ne stampi:

- La somma
- La differenza
- Il prodotto
- La media
- La distanza (cioè il valore assoluto della differenza)
- Il massimo (cioè il valore più grande dei due)
- Il minimo (cioè il valore più piccolo dei due)

Per farlo, realizzate questa classe:

```
public class Pair
{
```

```
   /**
      Costruisce una coppia.
      @param aFirst il primo valore della coppia
      @param aSecond il secondo valore della coppia
   */
   public Pair(double aFirst, double aSecond) { ... }

   /**
      Calcola la somma dei valori di questa coppia.
      @return la somma del primo e del secondo valore
   */
   public double getSum() { ... }
   ...
}
```

Realizzate, poi, una classe di collaudo `PairTester` che costruisca un oggetto di tipo `Pair`, ne invochi i metodi e visualizzi i risultati.

★ **Esercizio P4.5.** Dichiarate una classe `DataSet` che calcoli la somma e la media di una sequenza di numeri interi, dotata dei metodi

- `void addValue(int x)`
- `int getSum()`
- `double getAverage()`

Suggerimento: memorizzate la somma progressiva e il numero di valori introdotti.

Scrivete poi un programma di collaudo, `DataSetTester`, che invochi `addValue` quattro volte e visualizzi i valori previsti e i risultati ottenuti.

★★ **Esercizio P4.6.** Scrivete una classe `DataSet` che calcoli il valore maggiore e il valore minore presenti in una sequenza di numeri, dotata dei metodi

- `void addValue(int x)`
- `int getLargest()`
- `int getSmallest()`

Tenete traccia, istante per istante, del valore maggiore e del valore minore presente nella sequenza, usando all'interno del metodo `addValue` i metodi `Math.min` e `Math.max` per aggiornarli. Quali dovrebbero essere i valori iniziali per le variabili di esemplare utilizzate in questo problema? *Suggerimento*: `Integer.MIN_VALUE`, `Integer.MAX_VALUE`.

Scrivete un programma di collaudo, `DataSetTester`, che invochi `addValue` quattro volte e visualizzi i valori previsti e i risultati ottenuti.

★ **Esercizio P4.7.** Scrivete un programma che chieda all'utente una misura in metri e che poi la converta in miglia, piedi e pollici. Definite questa classe

```
public class Converter
{
   /**
      Costruisce un convertitore fra due unità di misura.
      @param aConversionFactor il fattore moltiplicatore per convertire
                               nell'unità desiderata
   */
   public Converter(double aConversionFactor) { ... }
```

```
   /**
      Converte una misura da un'unità a un'altra.
      @param fromMeasurement la misura da convertire
      @return il valore convertito
   */
   public double convertTo(double fromMeasurement) { ... }
}
```

Nella classe `ConverterTester` costruite e collaudate il seguente oggetto:

```
final double MILE_TO_KM = 1.609;
Converter milesToMeters = new Converter(1000 * MILE_TO_KM);
```

* **Esercizio P4.8.** Scrivete una classe `Square` il cui costruttore riceve la lunghezza dei lati di un quadrato. Progettate i metodi necessari a calcolare:

- l'area e il perimetro del quadrato
- la lunghezza della diagonale (usando il teorema di Pitagora)

** **Esercizio P4.9.** Realizzate una classe `SodaCan` (*lattina di bibita*) il cui costruttore riceve l'altezza e il diametro della lattina. Fornite i metodi `getVolume` e `getSurfaceArea`, nonché una classe `SodaCanTester` per il collaudo.

*** **Esercizio P4.10.** Realizzate una classe `Balloon` che rappresenti il modello di un pallone di forma sferica riempito d'aria, il cui costruttore costruisce un pallone vuoto. Fornite questi metodi:

- `void addAir(double amount)` aggiunge la quantità d'aria specificata
- `double getVolume()` restituisce il volume attuale
- `double getSurfaceArea()` restituisce l'area superficiale attuale
- `double getRadius()` restituisce il raggio attuale

Progettate, poi, una classe `BalloonTester` che costruisca un pallone, vi aggiunga 100 cm^3 di aria, collaudi i tre metodi di accesso, aggiunga altri 100 cm^3 di aria e collaudi nuovamente i tre metodi di accesso.

** **Esercizio P4.11.** *Calcolare il resto in monete.* Migliorate la classe `CashRegister` in modo che possa assistere un cassiere nelle operazioni di calcolo del resto. Il registratore di cassa calcola, in centesimi, il resto che deve essere restituito al cliente. Aggiungete alla classe `CashRegister` i metodi seguenti:

- `int giveDollars()`
- `int giveQuarters()`
- `int giveDimes()`
- `int giveNickels()`
- `int givePennies()`

Ciascuno di tali metodi calcola il numero di monete di un certo tipo che vanno restituite al cliente e riduce della quantità opportuna il resto ancora dovuto. Potete ipotizzare che i metodi vengano invocati nell'ordine sopra indicato, come in questa classe di collaudo:

```
public class CashRegisterTester
{
   public static void main(String[] args)
   {
      CashRegister register = new CashRegister();
```

```
      register.recordPurchase(8.37);
      register.enterPayment(10, 0, 0, 0, 0);
      System.out.println("Dollars: " + register.giveDollars());
      System.out.println("Expected: 1");
      System.out.println("Quarters: " + register.giveQuarters());
      System.out.println("Expected: 2");
      System.out.println("Dimes: " + register.giveDimes());
      System.out.println("Expected: 1");
      System.out.println("Nickels: " + register.giveNickels());
      System.out.println("Expected: 0");
      System.out.println("Pennies: " + register.givePennies());
      System.out.println("Expected: 3");
   }
}
```

** **Esercizio P4.12.** Nei Consigli pratici 4.1 abbiamo rappresentato lo stato del distributore automatico di francobolli mediante il saldo in centesimi: una scelta astuta, ma forse non quella più ovvia. Diversamente, si può memorizzare la quantità introdotta dall'utente, in dollari, e il saldo rimanente dopo aver emesso i francobolli di tipo "first class". Realizzare di nuovo il distributore usando questa strategia, senza, ovviamente, modificare la sua interfaccia pubblica.

*** **Esercizio P4.13.** Scrivete un programma che legge un numero intero e lo scompone in una serie di cifre singole, in ordine inverso. Per esempio, il numero 16384 verrà visualizzato come:

```
4
8
3
6
1
```

Potete ipotizzare che il numero non abbia più di cinque cifre e che non sia negativo. Progettate la classe `DigitExtractor`:

```
public class DigitExtractor
{
   /**
      Costruisce un estrattore di cifre che restituisce le cifre
      di un numero intero in ordine inverso.
      @param anInteger il numero intero da scomporre in cifre
   */
   public DigitExtractor(int anInteger) { ... }

   /**
      Restituisce la prossima cifra estratta.
      @return la prossima cifra
   */
   public double nextDigit() { ... }
}
```

Nella classe di collaudo, `DigitPrinter`, scrivete cinque volte l'enunciato `System.out.println(myExtractor.nextDigit())`.

** **Esercizio P4.14.** Realizzate una classe `QuadraticEquation` il cui costruttore riceve i coefficienti `a`, `b` e `c` dell'equazione di secondo grado $ax^2 + bx + c = 0$. Progettate i metodi `getSolution1` e `getSolution2` che restituiscono le soluzioni, usando la formula nota. Scrivete una classe di collaudo `QuadraticEquationTester` che costruisce un oggetto di tipo `QuadraticEquation` e visualizza le due soluzioni.

*** **Esercizio P4.15.** Scrivete un programma che legge due indicazioni orarie in formato militare (0900, 1730) e stampa il numero di ore e di minuti che separano i due orari. Ecco un esempio di esecuzione (i dati inseriti dall'utente sono in grassetto)

```
Please enter the first time: 0900
Please enter the second time: 1730
8 hours 30 minutes
```

Otterrete una valutazione migliore se riuscite a gestire il caso in cui il primo orario è successivo al secondo orario

```
Please enter the first time: 1730
Please enter the second time: 0900
15 hours 30 minutes
```

Realizzate una classe `TimeInterval` il cui costruttore riceve due orari in formato militare. La classe dovrebbe avere due metodi, `getHours` e `getMinutes`.

* **Esercizio P4.16.** *Scrivere lettere giganti*. Una lettera `H` gigante si può visualizzare in questo modo:

```
*   *
*   *
*****
*   *
*   *
```

Definite la classe `LetterH`:

```
public class LetterH
{
   public String toString()
   {
      return "*   *\n*   *\n*****\n*   *\n*   *\n";
   }
}
```

Definite classi simili per le lettere `E`, `L` e `O`. Poi, scrivete il seguente messaggio con lettere giganti:

```
H
E
L
L
O
```

** **Esercizio P4.17.** Scrivete una classe `ChristmasTree` il cui metodo `toString` restituisce una stringa così fatta, per rappresentare un albero di Natale (ricordatevi di usare le sequenze di escape).

```
   /\
  /  \
 /    \
/      \
--------
  "  "
  "  "
  "  "
```

** **Esercizio P4.18.** Scrivete un programma per tradurre i numeri da 1 a 12 nei nomi dei mesi corrispondenti, in inglese: `January`, `February`, `March`, …, `December`. Realizzate una classe `Month`, con un costruttore che riceve come parametro il numero del mese e un metodo, `getName`, che restituisce il nome del mese corrispondente. *Suggerimento:* create una stringa molto lunga per contenere i nomi di tutti i mesi (`"January February March..."`), nella quale inserirete spazi in modo che ciascun nome di mese abbia la stessa lunghezza. Poi, usate la funzione `substring` per estrarre il mese richiesto.

** **Esercizio P4.19.** Scrivete una classe che calcoli la data della domenica di Pasqua, che è la prima domenica dopo la prima luna piena di primavera. Usate questo algoritmo, ideato dal matematico Carl Friedrich Gauss nel 1800:

1. Sia `y` l'anno (ad esempio, 1800 o 2001).
2. Dividi `y` per 19, ottenendo il resto `a`. Ignora il quoziente.
3. Dividi `y` per 100, ottenendo quoziente `b` e resto `c`.
4. Dividi `b` per 4, ottenendo quoziente `d` e resto `e`.
5. Dividi `8 * b + 13` per 25, ottenendo il quoziente `g`. Ignora il resto.
6. Dividi `19 * a + b - d - g + 15` per 30, ottenendo il resto `h`. Ignora il quoziente.
7. Dividi `c` per 4, ottenendo quoziente `j` e resto `k`.
8. Dividi `a + 11 * h` per 319, ottenendo il quoziente `m`. Ignora il resto.
9. Dividi `2 * e + 2 * j - k - h + m + 32` per 7, ottenendo il resto `r`. Ignora il quoziente.
10. Dividi `h - m + r + 90` per 25, ottenendo il quoziente `n`. Ignora il resto.
11. Dividi `h - m + r + n + 19` per 32, ottenendo il resto `p`. Ignora il quoziente.

Infine, Pasqua cade il giorno `p` del mese `n`. Ad esempio, se `y` è 2001:

```
a = 6
b = 20
c = 1
d = 5, e = 0
g = 6
h = 18
j = 0, k = 1
m = 0
r = 6
n = 4
p = 15
```

Quindi, nel 2001 la domenica di Pasqua è caduta il 15 aprile. Scrivete una classe `Easter` con i metodi `getEasterSundayMonth` e `getEasterSundayDay`.

Progetti di programmazione

Progetto 4.1. In questo progetto elaborerete triangoli. Un triangolo è definito dalle coordinate x e y dei propri tre vertici.

Il vostro compito consiste nel calcolare le seguenti proprietà di un dato triangolo:

- le lunghezze di tutti i lati
- gli angoli in tutti i vertici
- il perimetro
- l'area

Naturalmente dovete realizzare una classe `Triangle` con i metodi appropriati. Realizzate, poi, un programma che chieda all'utente le coordinate dei vertici e generi una tabella ben impaginata che contenga tutte le sopraelencate proprietà del triangolo.

Questo progetto è particolarmente adatto a essere suddiviso tra due studenti, che dovrebbero concordare le specifiche dell'interfaccia pubblica della classe `Triangle`. Uno di essi, poi, dovrà realizzare la classe, mentre l'altro realizzerà simultaneamente il programma di interazione con l'utente e la presentazione dei risultati.

Progetto 4.2. La classe `CashRegister` ha una sfortunata limitazione: è strettamente connessa al sistema monetario degli Stati Uniti e del Canada. Il vostro obiettivo è quello di progettare un registratore di cassa che sia in grado di funzionare, invece, con euro e centesimi di euro. Però, invece di realizzare per il mercato europeo un'altra versione di `CashRegister`, altrettanto limitata, dovreste progettare una classe separata, `Coin`, che rappresenti un valore monetario, e un registratore di cassa che possa funzionare con monete di ogni tipo.

Capitolo 5

Esercizi di programmazione

** **Esercizio P5.1.** Scrivete un programma che stampi tutte le soluzioni reali dell'equazione di secondo grado $ax^2 + bx + c = 0$. Richiedete in ingresso i valori di a, b e c (verificando che a sia diverso da zero) e usate la formula dell'equazione di secondo grado. Se il *discriminante* $b^2 - 4ac$ è negativo, visualizzate un messaggio per segnalare che non esistono soluzioni reali.

Realizzate una classe `QuadraticEquation`, il cui costruttore riceva i coefficienti `a`, `b` e `c` dell'equazione quadratica. Dotatela dei metodi `getSolution1` e `getSolution2` che, usando la formula quadratica, restituiscono le soluzioni, oppure 0 se non vi sono soluzioni. Il metodo `getSolution1` deve restituire la soluzione di valore minore. Progettate, infine, il metodo

```
boolean hasSolutions()
```

che restituisce `false` se il discriminante è negativo.

** **Esercizio P5.2.** Scrivete un programma che riceva dall'utente un dato che descrive una carta da gioco, usando le abbreviazioni seguenti :

`A`	Asso
`2 ... 10`	Punteggi delle carte
`J`	Fante (*Jack*)
`Q`	Donna (*Queen*)
`K`	Re (*King*)
`D`	Quadri (*Diamonds*)
`H`	Cuori (*Hearts*)
`S`	Picche (*Spades*)
`C`	Fiori (*Clubs*)

Il programma deve visualizzare la descrizione completa della carta, come in questo esempio:

```
Enter the card notation:
4S
Four of spades
```

Realizzate una classe `Card` il cui costruttore riceve le lettere che descrivono la carta e il cui metodo `getDescription` restituisce una stringa che descrive compiutamente la carta. Se la stringa inserita non ha il formato corretto, il metodo `getDescription` deve restituire la stringa `"Unknown"`.

** **Esercizio P5.3.** Scrivete un programma che riceva come dati in ingresso tre numeri in virgola mobile, per poi stamparli in ordine crescente. Ecco un esempio:

```
Please enter three numbers:
4
9
2.5
The inputs in sorted order are
2.5
4
9
```

* **Esercizio P5.4.** Scrivete un programma che converta nel numero corrispondente un voto scolastico espresso mediante una lettera, come avviene comunemente nel sistema anglosassone. Le lettere sono `A`, `B`, `C`, `D` e `F`, eventualmente seguite dai segni `+` o `-`. I loro valori numerici sono, rispettivamente, 4, 3, 2, 1 e 0. I voti `F+` e `F-` non esistono. Un segno `+` o `-` incrementa o decrementa il valore numerico di 0.3, ma `A+` è uguale a 4.0. Qualunque altro voto, non riconosciuto, ha valore -1.

```
Enter a letter grade:
B-
Numeric value: 2.7.
```

Progettate una classe `Grade` dotata di un metodo `getNumericGrade`.

* **Esercizio P5.5.** Scrivete un programma che traduca un numero, compreso fra 0 e 4, nella lettera corrispondente al voto scolastico più simile, usando le convenzioni viste nell'Esercizio precedente. Per esempio, il numero 2.8 (che potrebbe essere la media di più voti) va convertito in `B-`. Arrotondate in favore del voto migliore: per esempio, 2.85, che è equidistante da `B-` e da `B`, diventa `B`.

Progettate una classe `Grade` dotata di un metodo `getLetterGrade`.

* **Esercizio P5.6.** Scrivete un programma che legga tre stringhe, per poi stamparle secondo l'ordinamento lessicografico crescente.

```
Please enter three strings:
Tom
Diana
Harry
The inputs in sorted order are:
Diana
Harry
Tom
```

** **Esercizio P5.7.** Modificate il metodo `getTax` della classe `TaxReturn`, usando una variabile `rate1_limit` il cui valore dipende dallo stato civile del contribuente. Scrivete, quindi, una formula unica per calcolare le imposte, in funzione del reddito e della soglia tra i due scaglioni. Verificate che i risultati siano identici a quelli forniti dalla classe `TaxReturn` vista in precedenza.

*** **Esercizio P5.8.** Il sistema fiscale originario degli Stati Uniti, nel 1913, era assai semplice e le imposte erano così definite:

- 1% dei primi $ 50 000 di reddito.
- 2% della porzione di reddito compresa tra $ 50 000 e $ 75 000.
- 3% della porzione di reddito compresa tra $ 75 000 e $ 100 000.
- 4% della porzione di reddito compresa tra $ 100 000 e $ 250 000.
- 5% della porzione di reddito compresa tra $ 250 000 e $ 500 000.
- 6% della porzione di reddito superiore a $ 500 000.

Non c'era alcuna distinzione tra contribuenti coniugati e non coniugati. Scrivete un programma che calcoli le imposte usando questo schema.

★★ **Esercizio P5.9.** Scrivete un programma che chieda all'utente il mese e il giorno del suo compleanno, visualizzando poi l'oroscopo corrispondente. Usate frasi che siano adatte a programmatori, come questa:

```
Please enter your birthday (month and day): 6 16
Gemini are experts at figuring out the behavior of complicated programs.
You feel where bugs are coming from and then stay one step ahead.
Tonight, your style wins approval from a tough critic.
```

Ogni frase deve contenere il nome del segno zodiacale corretto: per scoprire i nomi dei segni e le date che ne determinano l'inizio e la fine, consultate Internet e troverete un numero talmente elevato di siti che ci si potrebbe preoccupare…

★ **Esercizio P5.10.** Nel confrontare due istanti di tempo, espressi come ore e minuti in formato militare (cioè con le ore che vanno da 0 a 23), si può usare questo pseudocodice per determinare quale istante venga prima dell'altro.

```
Se hour1 < hour2
    time1 viene prima.
Altrimenti se hour1 e hour2 sono uguali
    Se minute1 < minute2
        time1 viene prima.
    Altrimenti se minute1 e minute2 sono uguali
        time1 e time2 sono uguali.
    Altrimenti
        time2 viene prima.
Altrimenti
    time2 viene prima.
```

Scrivete un programma che chieda all'utente di fornire due istanti di tempo e visualizzi quello che viene prima, seguito dall'altro.

★ **Esercizio P5.11.** Questo algoritmo calcola la stagione (Spring, *primavera*, Summer, *estate*, Fall, *autunno* o Winter, *inverno*) corrispondente a un giorno (*day*) e mese (*month*).

```
Se month è 1, 2 o 3, season = "Winter"
Altrimenti se month è 4, 5 o 6, season = "Spring"
Altrimenti se month è 7, 8 o 9, season = "Summer"
Altrimenti se month è 10, 11 o 12, season = "Fall"
Se month è divisibile per 3 e day >= 21
    Se season è "Winter", season = "Spring"
    Altrimenti se season è "Spring", season = "Summer"
    Altrimenti se season è "Summer", season = "Fall"
    Altrimenti season = "Winter"
```

Scrivete un programma che chieda all'utente di fornire un giorno e un mese, per poi visualizzare la stagione corrispondente.

★ **Esercizio P5.12.** Un anno di 366 giorni è detto bisestile (*leap year*). 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 è bisestile. Scrivere una classe `Year` con un metodo predicativo `boolean isLeapYear()`.

* **Esercizio P5.13.** Scrivete un programma che chieda all'utente di inserire il numero corrispondente a un mese (1 per gennaio, 2 per febbraio, e così via), e che quindi stampi il numero di giorni del mese. Nel caso di febbraio, stampate comunque "28 days".

```
Enter a month(1-12):
5
31 days
```

Realizzate una classe `Month` con un metodo `int getDays()`. Non usate un diverso enunciato `if` o `else` per ogni mese, ma fate uso di operatori booleani.

*** **Esercizio P5.14.** Scrivete un programma che legga due numeri in virgola mobile, per poi dichiararli uguali se, quando vengono arrotondati con due cifre decimali dopo la virgola, differiscono per meno di 0.01. Ecco due esempi di esecuzione:

```
Enter two floating-point numbers:
2.0
1.99998
They are the same when rounded to two decimal places.
They differ by less than 0.01.

Enter two floating-point numbers:
0.999
0.991
They are different when rounded to two decimal places.
They don't differ by less than 0.01.
```

* **Esercizio P5.15.** Migliorate la classe `BankAccount` del Capitolo 3, in modo da:

1. Rifiutare importi negativi nei metodi `deposit` e `withdraw`.
2. Rifiutare prelievi che portino a un saldo negativo.

* **Esercizio P5.16.** Scrivete un programma che riceva la retribuzione oraria di un dipendente. Quindi, chiedete quante ore ha lavorato il dipendente la scorsa settimana. Assicuratevi di accettare le frazioni di ora e calcolate la retribuzione. Eventuali straordinari (oltre le 40 ore settimanali) vanno pagati nella misura del 150 per cento rispetto alla retribuzione normale. Risolvete il problema scrivendo una classe `Paycheck`.

** **Esercizio P5.17.** Scrivete un programma per la conversione di unità di misura. Chiedete all'utente da quale unità vuole convertire e qual è l'unità di destinazione (*in* per pollice, *ft* per piede, *mi* per miglio, *mm* per millimetro, *cm* per centimetro, *m* per metro e *km* per chilometro). Usate due oggetti di una classe `UnitConverter`, uno che converta in metri e uno che converta da metri.

```
Convert from:
in
Convert to:
mm
Value:
10
10 in = 254 mm
```

*** **Esercizio P5.18.** Una retta giacente in un piano può essere specificata in diversi modi:

- mediante un punto (x, y) e una pendenza m

- mediante due punti $(x_1, y_1), (x_2, y_2)$
- mediante un'equazione nella forma $y = mx + b$
- mediante un'equazione nella forma $x = a$, se la retta è verticale

Realizzate una classe `Line` dotata di quattro costruttori, corrispondenti ai quattro casi menzionati, e realizzate i metodi:

```
boolean intersects(Line other)
boolean equals(Line other)
boolean isParallel(Line other)
```

****G** **Esercizio P5.19.** Scrivete un programma che disegni un cerchio di raggio 100 e centro nel punto di coordinate (200, 200) e che chieda all'utente di specificare le coordinate x e y di un punto. Disegnate il punto usando un piccolo cerchio, di colore verde se si trova all'interno del cerchio, di colore rosso in caso contrario. Per risolvere l'esercizio, dichiarate una classe `Circle` dotata del metodo `boolean isInside(Point2D.Double p)`.

*****G** **Esercizio P5.20.** Scrivete un programma grafico che chieda all'utente di specificare i raggi di due cerchi, il primo centrato nel punto di coordinate (100, 200), il secondo nel punto (200, 100). Disegnate i cerchi, colorandoli di verde se si intersecano, di rosso in caso contrario. Se l'utente fornisce un raggio negativo, il programma non deve disegnare niente. Per risolvere l'esercizio, dichiarate una classe `Circle` dotata del metodo `boolean intersects(Circle other)`. *Suggerimento*: calcolate la distanza tra i centri dei cerchi e confrontatela con i raggi.

Progetti di programmazione

Progetto 5.1. Realizzate una classe per rappresentare una *serratura a combinazione*, che ha una ruota con 26 posizioni etichettate con le lettere dell'alfabeto inglese, dalla A alla Z. La ruota deve essere impostata tre volte e la serratura si apre se la combinazione è corretta. Nel momento in cui la serratura viene richiusa, si può introdurre una nuova combinazione. Se la ruota viene impostata più di tre volte, le ultime tre impostazioni sono quelle che determinano se aprire o meno la serratura. Una parte importante del progetto consiste nella realizzazione di un'interfaccia pubblica appropriata per la classe `CombinationLock`.

Progetto 5.2. Recuperate dal sito `http://www.irs.ustreas.gov` le istruzioni per il calcolo delle tasse dello scorso anno (*form 1040*). Individuate gli scaglioni di reddito usati lo scorso anno per tutte le categorie di contribuenti (non coniugato, coniugato che presenta dichiarazione congiunta, coniugato che presenta dichiarazione separata, capofamiglia). Scrivete un programma che calcoli le tasse seguendo tali regole, ignorando deduzioni, esenzioni e crediti d'imposta: applicate semplicemente l'aliquota di tassazione al reddito imponibile.

Capitolo 6

Esercizi di programmazione

***** **Esercizio P6.1.** Completate il programma visto nei Consigli pratici 6.1, in modo che legga dodici valori di temperatura e visualizzi il mese che ha registrato la temperatura massima.

******* **Esercizio P6.2.** *Verifica di un numero di carta di credito.* L'ultima cifra di un numero di carta di credito è una *cifra di controllo*, che protegge dagli errori di trascrizione, come un errore che alteri

un'unica cifra oppure uno che scambi due cifre tra loro. L'algoritmo qui descritto viene effettivamente utilizzato per la verifica di numeri di carta di credito, ma, per semplicità, lo descriviamo per numeri di 8 cifre anziché di 16 cifre.

- Partendo dalla cifra più a destra, in posizione uno, calcolate la somma di tutte le cifre aventi posto dispari. Ad esempio, se il numero della carta di credito è 43589795, si calcola 5 + 7 + 8 + 3 = 23.
- Raddoppiate il valore di tutte le cifre che non sono state sommate al punto precedente e sommate tutte le singole cifre dei numeri così ottenuti. Ad esempio, con il numero precedente, il raddoppio delle cifre di posto pari, iniziando dalla penultima, genera la sequenza 18 18 10 8. Sommando tutte le singole cifre di questi numeri, si ottiene 1 + 8 + 1 + 8 + 1 + 0 + 8 = 27.
- Sommate i risultati ottenuti nei due punti precedenti: se l'ultima cifra del risultato è zero, il numero è valido. Nel nostro caso, 23 + 27 = 50, quindi il numero è valido.

Scrivete un programma che realizzi questo algoritmo. L'utente deve fornire un numero di otto cifre e il programma deve dire se è valido oppure no. Se non risulta essere valido, il programma deve anche visualizzare il valore della cifra di controllo che lo renderebbe valido.

★ **Esercizio P6.3.** *Conversione di valuta.* Scrivete il programma `CurrencyConverter` che chieda all'utente di inserire il tasso di cambio odierno fra dollari statunitensi (USD) ed euro (EUR). Quindi, il programma legge alcuni valori in dollari e ne visualizza il rispettivo valore convertito in euro. Terminate quando l'utente scrive `Q`.

★★★ **Esercizio P6.4.** *Traiettoria di un proiettile.* Immaginate di lanciare una palla di cannone verticalmente nell'aria, con una velocità iniziale v_0. Qualsiasi formulario vi dirà che la posizione della palla, dopo t secondi, è pari a $s(t) = -0.5 \cdot g \cdot t^2 + v_0 \cdot t$, dove g, uguale a 9.81 m/s^2, rappresenta la forza di gravità della Terra. Peraltro, nessun manuale di formule spiega perché qualcuno dovrebbe voler eseguire un esperimento così chiaramente pericoloso, quindi lo simuleremo in tutta sicurezza al computer.

Di fatto, confermeremo mediante una simulazione il teorema espresso dal calcolo. Nella nostra prova, esamineremo come la palla si sposta in intervalli di tempo molto brevi, chiamati Δt. In un breve intervallo di tempo, la velocità v rimane pressoché costante e possiamo calcolare la distanza percorsa dal proiettile come segue: $\Delta s = v \cdot \Delta t$. Nel nostro programma, imposteremo semplicemente questo valore:

```
double deltaT = 0.01;
```

e aggiorneremo la posizione mediante l'enunciato:

```
s = s + v * deltaT;
```

La velocità cambia continuamente, perché di fatto si riduce a causa della forza di gravità terrestre. In un breve intervallo di tempo, la variazione di velocità corrisponde a $-g \cdot \Delta t$ e dobbiamo continuare ad aggiornare la velocità in questo modo:

```
v = v - g * deltaT;
```

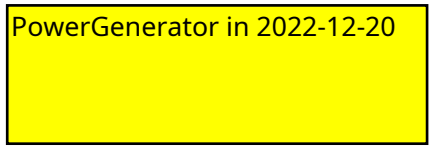
Nell'iterazione successiva, si utilizza la nuova velocità per aggiornare la distanza.

Ora eseguite la simulazione, finché la palla di cannone ricade al suolo. Considerate la velocità iniziale come dato d'ingresso (100 m/s è un buon valore). Aggiornate la posizione e la velocità 100 volte al secondo, ma stampate la posizione solamente una volta al secondo. Stampate anche i valori esatti della formula $s(t) = -0.5 \cdot g \cdot t^2 + v_0 \cdot t$, per confrontarli. Usate una classe `Cannonball`.

Se vi chiedete quale sia il vantaggio di questo tipo di simulazioni, quando è disponibile una formula esatta, osserviamo che la formula del manuale *non* è esatta. In realtà, la forza gravitazionale diminuisce a mano a mano che la palla di cannone si allontana dalla superficie della Terra. Questo complica il calcolo algebrico in modo tale da rendere impossibile la definizione di una formula esatta per il moto reale. Per contro, la simulazione al computer si può migliorare facilmente, applicando una forza gravitazionale variabile. Per le palle di cannone, in realtà, la formula del manuale è abbastanza precisa, ma i computer sono indispensabili per calcolare accuratamente le traiettorie di oggetti che volano a grandi altezze, come i missili balistici.

** **Esercizio P6.5.** Scrivete un programma che visualizzi le potenze di dieci, in questo modo:

```
1.0
10.0
100.0
1000.0
10000.0
100000.0
1000000.0
1.0E7
1.0E8
1.0E9
1.0E10
1.0E11
```



Realizzate una classe:

```
public class PowerGenerator
{
   /**
      Costruisce un generatore di potenze.
      @param aFactor il numero che verrà moltiplicato per se stesso
   */
   public PowerGenerator(double aFactor) { ... }

   /**
      Calcola la successiva potenza.
   */
   public double nextPower() { ... }
   ...
}
```

Infine, fornite una classe di collaudo, `PowerGeneratorRunner`, che invochi dodici volte `System.out.println(myGenerator.nextPower())`.

** **Esercizio P6.6.** La *sequenza di Fibonacci* è definita dalla regola seguente: i primi due valori della sequenza sono 1 e 1; ciascun valore successivo è costituito dalla somma dei due valori che lo precedono. Per esempio, il terzo valore è dato dalla somma 1 + 1 = 2, il quarto valore è 1 + 2 = 3, e il quinto è 2 + 3 = 5. Se f_n indica l'*enne*simo valore nella sequenza di Fibonacci, allora abbiamo:

$$f_1 = 1$$
$$f_2 = 1$$
$$f_n = f_{n-1} + f_{n-2} \text{ se } n > 2$$

Scrivete un programma che chieda all'utente di inserire il valore di `n` e che stampi l'`n`-esimo valore nella sequenza di Fibonacci. Usate una classe `FibonacciGenerator` con un metodo `nextNumber`.

Suggerimento: non occorre memorizzare tutti i valori di f_n. Vi servono solamente i due valori calcolati più recentemente, per poi calcolare quello successivo nella serie:

```
fold1 = 1;
fold2 = 1;
fnew = fold1 + fold2;
```

Poi, eliminate `fold2`, che non è più necessario, e impostate `fold2` al valore di `fold1` e `fold1` al valore di `fnew`. Ripetete il calcolo di `fnew` per il numero appropriato di volte.

La vostra classe generatrice sarà poi collaudata da questo programma:

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

      System.out.println("Enter n: ");
      int n = in.nextInt();

      FibonacciGenerator fg = new FibonacciGenerator();

      for (int i = 1; i <= n; i++)
         System.out.println(fg.nextNumber());
   }
}
```

★★ **Esercizio P6.7**. *Media e deviazione standard*. Scrivete un programma che legga un insieme di dati in virgola mobile. Al termine, stampate il conteggio dei valori, la media e la deviazione standard. La media di un insieme di valori $x_1, ..., x_n$ è data da

$$\overline{x} = \frac{\sum x_i}{n}$$

dove $\Sigma x_i = x_1 + ... + x_n$ è la somma dei valori di ingresso. La deviazione standard è espressa da questa formula

$$S = \sqrt{\frac{\sum (x_i - \overline{x})^2}{n - 1}}$$

che, però, non è adatta per la nostra attività, perché, nel momento in cui calcolate la deviazione standard, i singoli valori x_i se ne sono andati da un pezzo. Finché non sapete come memorizzare questi valori, utilizzate questa formula, anche se è numericamente meno stabile:

$$S = \sqrt{\frac{\sum x_i^2 - \frac{1}{n}\left(\sum x_i\right)^2}{n - 1}}$$

Potete calcolare quest'ultimo valore tenendo traccia, nella classe `DataSet`, del conteggio dei valori in ingresso, della loro somma e della somma dei loro quadrati, mano a mano che li ricevete.

★★ **Esercizio P6.8.** *Scomposizione di numeri interi in fattori*. Scrivete un programma che chieda all'utente un numero intero e che visualizzi tutti i suoi fattori. Per esempio, se l'utente immette 150, il programma stamperà:

```
2
3
5
5
```

Usate una classe `FactorGenerator` con il costruttore `FactorGenerator(int numberToFactor)` e i metodi `nextFactor` e `hasMoreFactors`. Progettate la classe `FactorPrinter` il cui metodo `main` legga un numero in ingresso, costruisca un esemplare di `FactorGenerator` e visualizzi i fattori.

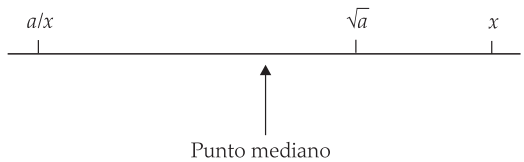
** **Esercizio P6.9.** *Numeri primi*. Scrivete un programma che chieda all'utente un numero intero e che poi stampi tutti i numeri primi fino al numero stesso. Per esempio, se l'utente immette 20, il programma stamperà:

```
2
3
5
7
11
13
17
19
```

In java-scuola/6.x/6.9

Ricordate 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`.

** **Esercizio P6.10.** Il *metodo di Erone* è un metodo per calcolare le radici quadrate noto fin dagli antichi greci. Se x è un'approssimazione del valore della radice quadrata di a, allora la media fra x e a/x è un'approssimazione migliore.



Realizzate una classe `RootApproximator` che usi 1 come prima approssimazione e il cui metodo `nextGuess` generi una sequenza di approssimazioni sempre migliori. Fornite un metodo `hasMoreGuesses` che restituisca `false` se due tentativi successivi sono sufficientemente simili tra loro (cioè se differiscono tra loro per meno di un piccolo valore ε). Collaudate quindi la vostra classe in questo modo:

```
RootApproximator approx = new RootApproximator(a, EPSILON);
while (approx.hasMoreGuesses())
   System.out.println(approx.nextGuess());
```

** **Esercizio P6.11.** Il metodo iterativo più famoso per calcolare le radici di una funzione $f(x)$ (vale a dire i valori x per i quali $f(x)$ vale 0) è l'*approssimazione di Newton-Raphson*. Per trovare lo zero di una funzione, le cui derivate siano già note, calcolate ripetutamente:

$$x_{new} = x_{old} - f(x_{old})/f'(x_{old}).$$

Per questo esercizio, scrivete un programma che calcoli le radici *enne*sime di numeri in virgola mobile. Chiedete all'utente di inserire i valori per a e per n, quindi ottenete il valore della radice *enne*sima di a mediante il calcolo di uno zero della funzione $f(x) = x^n - a$. Usate l'approccio dell'esercizio precedente.

****** **Esercizio P6.12.** Il valore di e^x si può calcolare con la serie di potenze

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

dove $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$ e $0! = 1$.

Scrivete un programma che calcoli e^x usando questa formula. Ovviamente non potete calcolare una sommatoria infinita: continuate semplicemente a sommare termini finché un singolo addendo non sia minore di una determinata soglia. A ogni passo dovete calcolare il nuovo termine e aggiungerlo al totale. Aggiornate tali termini in questo modo:

```
term = term * x / n;
```

Seguite l'approccio dei due esercizi precedenti, realizzando una classe `ExpApproximator` che usi il valore 1 come prima approssimazione

***** **Esercizio P6.13.** Scrivete un programma `RandomDataAnalyzer` che generi 100 numeri casuali compresi tra 0 e 1000 e li aggiunga a un oggetto di tipo `DataSet`. Visualizzate poi la loro media e il valore massimo.

****** **Esercizio P6.14.** Programmate la simulazione seguente. Si lanciano alcune freccette in punti casuali all'interno di un quadrato avente vertici nelle posizioni (1, 1) e (–1, –1). Si fa centro quando la freccetta cade all'interno del cerchio unitario (avente centro (0, 0) e raggio uguale a 1), altrimenti è un colpo mancato. Eseguite la simulazione e usatela per determinare un valore approssimato di π. Otterrete una valutazione migliore se spiegate perché questo è un metodo migliore per stimare π, rispetto al programma dell'ago di Buffon.

*****G** **Esercizio P6.15.** *Itinerario casuale.* Simulate il girovagare di un ubriaco in un reticolo quadrato di strade. Disegnate una griglia con venti strade orizzontali e venti strade verticali. Rappresentate l'ubriaco simulato mediante un punto e collocatelo inizialmente al centro della griglia. Per 100 volte, fate prendere all'ubriaco una direzione casuale (est, ovest, nord, sud), spostatelo di un isolato nella direzione scelta e ridisegnate il punto. Ci si aspetterebbe che, in media, la persona non possa andare da nessuna parte e che, a lungo andare, gli spostamenti casuali in direzioni diverse impediscano di uscire dal reticolo, ma si può dimostrare effettivamente che, con probabilità 1, alla fine la persona esce da qualsiasi regione circoscritta. Usate una classe per la griglia e un'altra per l'ubriaco.

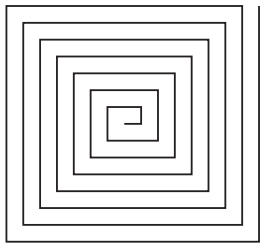
*****G** **Esercizio P6.16.** Questo esercizio è la prosecuzione dell'Esercizio P6.4. La maggior parte delle palle di cannone non viene sparata in verticale, ma secondo un certo angolo. Se la velocità iniziale è v e l'angolo iniziale è α, in pratica la velocità viene espressa da un vettore, con componenti $v_x = v \cos(\alpha)$, e $v_y = v \operatorname{sen}(\alpha)$. Nella direzione dell'asse x la velocità non cambia, mentre in quella dell'asse y la forza gravitazionale impone il suo pedaggio. Ripetete la simulazione descritta in quell'esercizio, ma aggiornate le posizioni x e y separatamente, aggiornando separatamente anche le componenti x e y della velocità. Per ciascun secondo di tempo trascorso completamente, tracciate in una visualizzazione grafica la posizione del proiettile mediante un piccolo cerchio e ripetete l'operazione finché il proiettile ricade a terra.

Questo genere di problema è di interesse storico: i primi computer furono progettati proprio per eseguire calcoli balistici di questo tipo, tenendo conto, per proiettili volanti ad altezza elevata, della diminuzione della forza gravitazionale e della forza del vento.

***G** **Esercizio P6.17**. Scrivete un'applicazione grafica che visualizzi una scacchiera con 64 riquadri, alternativamente bianchi e neri.

****G** **Esercizio P6.18**. Scrivete un'applicazione grafica che chieda all'utente di inserire un numero `n` e che poi tracci `n` cerchi, con raggio e con centro casuali. Tutti i cerchi devono essere completamente contenuti all'interno della finestra

*****G** **Esercizio P6.19**. Scrivete un'applicazione grafica che disegni una spirale come questa:



****G** **Esercizio P6.20.** Tracciare grafici di curve mediante la libreria grafica di Java è facile e divertente. Disegnate semplicemente un centinaio di segmenti di linea per collegare i punti $(x, f(x))$ e $(x + d, f(x + d))$, dove x varia da x_{min} a x_{max} e dove $d = (x_{max} - x_{min})/100$.

Disegnate la curva $f(x) = 0.00005\ x^3 - 0.03\ x^2 + 4x + 200$, dove x varia da 0 a 400 nel modo appena descritto.

*****G** **Esercizio P6.21.** Disegnate un'immagine della "rosa con quattro petali", la cui equazione in coordinate polari è $r = \cos(2\theta)$. Aumentate il valore di θ da zero a 2π, mediante 100 incrementi successivi. Ogni volta calcolate r e, conseguentemente, le coordinate (x, y) a partire dai valori in coordinate polari, mediante questa formula:

$$x = r \cos \theta, \quad y = r \operatorname{sen} \theta$$

Progetti di programmazione

Progetto 6.1. *Indice di leggibilità di Flesch*. L'indice è stato inventato da Flesch quale semplice strumento per misurare la leggibilità di un documento senza effettuare analisi linguistiche.

- Contate tutte le parole nel file. Una *parola* è una qualunque sequenza di caratteri delimitata da spazi vuoti, indipendentemente dal fatto che si tratti di un vocabolo realmente esistente.
- Contate tutte le sillabe in ciascuna parola. Per semplificare l'operazione, applicate le seguenti regole. Ciascun *gruppo* di vocali adiacenti (a, e, i, o, u, y) fa parte di un'unica sillaba (per esempio, il gruppo "ea" di "real" forma una sillaba, mentre le vocali "e .. a", in "regal", danno luogo a due sillabe). Tuttavia, una "e" alla fine di una parola non costituisce una sillaba a sé stante. Inoltre, ciascuna parola contiene almeno una sillaba, anche nel caso in cui le regole precedenti diano un conteggio uguale a zero.
- Contate tutte le frasi. Una frase è terminata da un punto, un punto e virgola, due punti, un punto interrogativo o esclamativo.
- L'indice si calcola mediante la formula seguente, arrotondando il risultato all'intero più vicino.

$$\begin{aligned}\text{Indice} = {} & 206.835 \\ & - 84.6 \times (\text{Numero di sillabe/Numero di parole}) \\ & - 1.015 \times (\text{Numero di parole/Numero di frasi})\end{aligned}$$

Lo scopo dell'indice è quello di obbligare gli autori a riscrivere i loro testi finché l'indice non è sufficientemente alto, un risultato che si ottiene riducendo la lunghezza delle frasi ed eliminando le parole lunghe. Per esempio, la frase seguente:

> *The following index was invented by Flesch as a simple tool to estimate the legibility of a document without linguistic analysis.*

si può riscrivere in questo modo:

> *Flesch invented an index to check whether a text is easy to read. To compute the index, you need not look at the meaning of the words.*

L'indice è espresso da un numero, solitamente compreso fra 0 e 100, che indica la facilità di lettura del testo. Ecco alcuni esempi di materiale scelto a caso fra varie pubblicazioni:

Fumetti	95
Pubblicità diretta al consumatore finale	82
Rivista *Sports Illustrated*	65
Rivista *Time*	57
Quotidiano *New York Times*	39
Polizza di assicurazione auto	10
Codice fiscale	–6

Gli indici, tradotti in funzione del livello scolastico statunitense, sono:

91-100	Studente della 5ª classe della scuola primaria
81-90	Studente della 6ª classe della scuola primaria
71-80	Studente della 7ª classe della scuola primaria
66-70	Studente della 8ª classe della scuola primaria
61-65	Studente della 9ª classe della scuola primaria
51-60	Studente di scuola secondaria
31-50	Studente universitario di primo livello
0-30	Laureato di primo livello
Inferiore a 0	Laureato in legge

Il vostro programma deve leggere un file di testo, calcolarne l'indice di leggibilità e stampare il livello scolastico corrispondente. Usate le classi `Word` e `Document`.

Progetto 6.2. *Il gioco di Nim.* Si tratta di un gioco ben conosciuto, con un certo numero di varianti: vedremo qui una versione che ha una strategia interessante per arrivare alla vittoria. Due giocatori prelevano a turno biglie da un mucchio. In ciascun turno, il giocatore sceglie quante biglie prendere: deve prenderne almeno una, ma non più della metà del mucchio. Quindi tocca all'altro giocatore. Perde chi è costretto a prendere l'ultima biglia.

Scrivete un programma in cui il computer gioca contro un avversario umano. Generate un numero intero casuale, compreso fra 10 e 100, per indicare il numero iniziale di biglie. Generate un altro numero intero casuale, zero o uno, per decidere se la prima mossa tocca al computer o al giocatore umano. Generate un ultimo numero intero casuale, zero o uno, per stabilire se il computer giocherà in modo *intelligente* o *stupido.* Nel modo stupido, quando è il suo turno, il computer si limita a sottrarre dal mucchio un numero casuale di biglie, purché sia una quantità ammessa (compresa, quindi, fra 1 e $n/2$). Nel modo intelligente, il computer preleva il numero di biglie sufficiente affinché il numero di quelle rimanenti sia uguale a una potenza di due, meno uno: 3, 7, 15, 31 o 63. Questa è sempre una mossa valida, eccetto quando il numero delle biglie presenti è uguale a una potenza di due, meno uno: in questo caso, il computer preleverà una quantità casuale, purché ammessa.

Noterete che, nel modo intelligente, se ha la prima mossa il computer non può essere sconfitto, a meno che il mucchio non contenga inizialmente 15, 31 o 63 biglie. Analogamente, un giocatore umano che abbia la prima mossa e che conosca la strategia vincente può vincere contro il computer.

Per realizzare questo programma, usate le classi `Pile` ("mucchio"), `Player` ("giocatore") e `Game` ("partita"). Un giocatore può essere inesperto, esperto o umano: gli oggetti di tipo `Player` che rappresentano giocatori umani chiedono dati in ingresso all'utente.

Capitolo 7

Esercizi di programmazione

* **Esercizio P7.1.** Realizzate una classe `Purse` ("borsellino"), i cui esemplari siano adatti a contenere un insieme di monete. Per semplicità, memorizzeremo semplicemente i nomi delle monete in un oggetto di tipo `ArrayList<String>` (nel Capitolo 8 vedrete una migliore rappresentazione dei dati). Fornite un metodo

```
void addCoin(String coinName)
```

Aggiungete poi alla classe `Purse` un metodo `toString` che visualizzi le monete presenti nel borsellino, in questo formato:

```
Purse[Quarter,Dime,Nickel,Dime]
```

* **Esercizio P7.2.** Scrivete un metodo `reverse` che inverta la sequenza di monete presenti in un borsellino. Usate il metodo `toString` dell'esercizio precedente per collaudare il vostro codice. Ad esempio, se si invoca `reverse` con questo borsellino

```
Purse[Quarter,Dime,Nickel,Dime]
```

si ottiene il seguente borsellino

```
Purse[Dime,Nickel,Dime,Quarter]
```

* **Esercizio P7.3.** Aggiungete alla classe `Purse` il metodo

```
public void transfer(Purse other)
```

che trasferisca nel borsellino con cui viene invocato il contenuto di un altro borsellino. Ad esempio, se `a` è

```
Purse[Quarter,Dime,Nickel,Dime]
```

e `b` è

```
Purse[Dime,Nickel]
```

allora, dopo l'invocazione di `a.transfer(b)`, `a` diventa:

```
Purse[Quarter,Dime,Nickel,Dime,Dime,Nickel]
```

mentre `b` rimane vuoto.

* **Esercizio P7.4.** Aggiungete alla classe `Purse` il metodo

```
public boolean sameContents(Purse other)
```

che controlli se due borsellini contengono le stesse monete nello stesso ordine.

** **Esercizio P7.5.** Aggiungete alla classe `Purse` il metodo

```
public boolean sameCoins(Purse other)
```

che controlli se due borsellini contengono le stesse monete, eventualmente in ordine diverso. Ad esempio, i borsellini

```
Purse[Quarter,Dime,Nickel,Dime]
```

e

```
Purse[Nickel,Dime,Dime,Quarter]
```

devono considerarsi uguali. Avrete probabilmente bisogno di uno o più metodi ausiliari.

** **Esercizio P7.6.** Un poligono è una curva chiusa formata da segmenti che ne congiungono i vertici. Realizzate una classe `Polygon` i cui metodi

```
public double perimeter()
```

e

```
public double area()
```

calcolino, rispettivamente, il perimetro e l'area di un poligono. Per calcolare il perimetro, valutate la distanza fra punti adiacenti e sommate tali distanze. L'area di un poligono con vertici (x_0, y_0), ..., (x_{n-1}, y_{n-1}) è

$$(x_0y_1 + x_1y_2 + \ldots + x_{n-1}y_0 - y_0x_1 - y_1x_2 - \ldots - y_{n-1}x_0) / 2$$

Per prova, calcolate il perimetro e l'area di un rettangolo e di un esagono regolare. *Nota*: non c'è bisogno di visualizzare graficamente il poligono, lo farete nell'Esercizio 7.18.

* **Esercizio P7.7.** Scrivete un programma che legga una sequenza di numeri interi, memorizzandola in un array, e ne calcoli la *somma a elementi alterni*. Per esempio, se il programma viene eseguito fornendo questi dati

1 4 9 16 9 7 4 9 11

allora calcola

$$1 - 4 + 9 - 16 + 9 - 7 + 4 - 9 + 11 = -2$$

** **Esercizio P7.8.** Scrivete un programma che produca 10 permutazioni casuali dei numeri da 1 a 10. Per generare una permutazione casuale, riempite un array con i numeri da 1 a 10, facendo in modo che non ve ne siano due uguali. Potreste farlo in modo brutale, invocando `Random.nextInt` fino a quando 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, rimuovetelo e accodatelo all'array che contiene la permutazione. Ripetete dieci volte. Realizzate una classe `PermutationGenerator` che abbia il metodo `int[] nextPermutation`.

** **Esercizio P7.9.** Una *ripetizione* ("run") è definita come una sequenza di valori adiacenti ripetuti. Scrivete un programma che generi una sequenza di 20 lanci casuali di un dado e che visualizzi i valori ottenuti, identificando le ripetizioni racchiudendole tra parentesi tonde, con questo formato:

```
1 2 (5 5) 3 1 2 4 3 (2 2 2 2) 3 6 (5 5) 6 3 1
```

Usate questo pseudocodice:

```
Imposta la variabile booleana inRun a false.
Per ogni indice i valido nel vettore
   Se inRun
      Se values[i] è diverso dal valore precedente
         Visualizza )
         inRun = false
   Se non inRun
      Se values[i] è uguale al valore seguente
         Visualizza (
         inRun = true
   Visualizza values[i]
Se inRun, visualizza )
```

★★★ **Esercizio P7.10.** Scrivete un programma che generi una sequenza di 20 lanci casuali di un dado e che visualizzi i valori ottenuti, identificando soltanto la ripetizione più lunga, racchiudendola tra parentesi tonde come in questo esempio:

```
1 2 5 5 3 1 2 4 3 (2 2 2 2) 3 6 5 5 6 3 1
```

Se sono presenti più ripetizioni della stessa lunghezza massima, contrassegnate la prima.

★★ **Esercizio P7.11.** È scientificamente accertato che, in un bagno pubblico, gli uomini generalmente preferiscono rendere massima la distanza tra sé e i servizi già occupati, occupando il servizio che si trova al centro della più lunga sequenza di servizi liberi.

Ad esempio, esaminiamo questa situazione, in cui sono presenti dieci servizi, tutti liberi.

```
_ _ _ _ _ _ _ _ _ _
```

Il primo avventore occuperà una delle due posizioni centrali:

```
_ _ _ _ _ X _ _ _ _
```

Il successivo sceglierà il servizio che si trova al centro dell'area libera sulla sinistra:

```
_ _ X _ _ X _ _ _ _
```

Scrivete un programma che legga inizialmente il numero di servizi disponibili e visualizzi diagrammi nel formato appena descritto, uno per volta, man mano che i servizi vengono occupati. *Suggerimento*: usate un array di valori `boolean` per indicare se un servizio è occupato oppure no.

★★★ **Esercizio P7.12.** In questo progetto vi occuperete del *solitario bulgaro*. Il gioco inizia con 45 carte (non è necessario che siano carte da gioco, bastano dei cartoncini impilabili senza segni di distinzione), che vanno divise in un certo numero di pile di dimensione casuale: potreste, ad esempio, iniziare con pile di dimensioni pari a 20, 5, 1, 9 e 10. Ad ogni turno prendete una carta da ciascuna pila, formando con esse una nuova pila: la configurazione iniziale che abbiamo usato prima come esempio verrebbe trasformata in un insieme di pile aventi dimensioni pari a 19, 4, 8, 9 e 5.

Il solitario termina quando sono presenti, in qualsiasi ordine, le pile aventi dimensioni pari a 1, 2, 3, 4, 5, 6, 7, 8 e 9 (e si può dimostrare che ciò accade sempre).

Nel vostro programma, generate una configurazione iniziale casuale e visualizzatela. Poi, eseguite turni successivi del solitario, visualizzando il risultato dopo ogni turno. Fermatevi quando è stata raggiunta la configurazione finale del solitario.

** **Esercizio P7.13.** Aggiungete alla classe `TicTacToe` del Paragrafo 7.8 il metodo `getWinner`, che restituisca `"x"` oppure `"o"` per indicare un vincitore, oppure `" "` se non c'è ancora un vincitore. Ricordate che una configurazione vincente consiste di tre segni identici su una riga, su una colonna o su una diagonale.

*** **Esercizio P7.14.** Scrivete un'applicazione per giocare a tic-tac-toe. Il vostro programma deve visualizzare la scacchiera, passare all'altro giocatore dopo ogni mossa e dichiarare il vincitore.

** **Esercizio P7.15.** *Quadrati magici*. Una matrice $n \times n$ riempita con i numeri 1, 2, 3, ..., n^2 è un quadrato magico se la somma degli elementi di ogni riga, di ogni colonna e delle due diagonali ha lo stesso valore. Per esempio, questo è un quadrato magico:

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

Scrivete un programma che legga n^2 valori dalla tastiera e verifichi se, disposti in una matrice, formano un quadrato magico.

Dovete verificare tre caratteristiche:

- L'utente ha inserito un numero n^2 di valori, per qualche valore di n?
- Ciascuno dei numeri 1, 2, ..., n^2 è veramente presente una e una sola volta nei dati inseriti dall'utente?
- Quando i numeri vengono disposti in un quadrato, la somma degli elementi di ogni riga, di ogni colonna e delle due diagonali ha lo stesso valore?

Se il numero di valori in ingresso è il quadrato di un numero intero n, verificate che tutti i numeri compresi fra 1 e n^2 siano presenti. Poi, calcolate la somma degli elementi di ogni riga, di ogni colonna e delle due diagonali. Realizzate una classe `Square` con i metodi seguenti:

```
public void add(int i)
public boolean isMagic()
```

** **Esercizio P7.16.** Implementate il seguente algoritmo per costruire quadrati magici n per n, che funziona solo se n è dispari. Posizionate il numero 1 al centro dell'ultima riga. Dopo aver messo il numero k nella posizione (i, j), mettete il numero $k + 1$ nel riquadro in basso a destra rispetto alla posizione di k, andando a capo sui bordi. Se, però, il riquadro in basso a destra è già occupato, dovete passare al riquadro che si trova immediatamente sopra al numero che avete appena inserito. Ecco il quadrato 5×5 che si ottiene seguendo questo metodo:

11	18	25	2	9
10	12	19	21	3
4	6	13	20	22
23	5	7	14	16
17	24	1	8	15

Scrivete un programma il cui dato in ingresso sia il numero n e il cui prodotto in uscita sia il quadrato magico di ordine n, se n è dispari. Realizzate una classe `MagicSquare` con un costruttore che costruisca il quadrato e con il metodo `toString` che restituisca una rappresentazione del quadrato.

*G **Esercizio P7.17.** Realizzate una classe `Cloud` ("nuvola") che contenga un vettore di oggetti di tipo `Point2D.Double`. Progettate i metodi:

```
public void add(Point2D.Double aPoint)
public void draw(Graphics2D g2)
```

Scrivete un'applicazione grafica che visualizzi una nuvola di 100 punti disposti casualmente, disegnando ciascun punto come un piccolo cerchietto.

G **Esercizio P7.18. Realizzate una classe `Polygon` che contenga un vettore di oggetti di tipo `Point2D.Double`. Progettate i metodi:

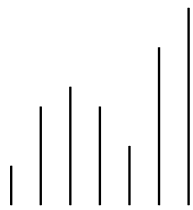
```
public void add(Point2D.Double aPoint)
public void draw(Graphics2D g2)
```

Disegnate il poligono congiungendo punti adiacenti con una linea, per poi chiuderlo congiungendo i punti finale e iniziale. Scrivete un'applicazione grafica che disegni un quadrato e un pentagono usando due oggetti di tipo `Polygon`.

*G **Esercizio P7.19.** Scrivete una classe `Chart` dotata dei metodi

```
public void add(int value)
public void draw(Graphics2D g2)
```

che visualizzi un diagramma a bastoncini con i valori che sono stati inseriti, in modo simile a quanto si può qui vedere.



Potete ipotizzare che i valori rappresentino posizioni misurate in pixel.

G **Esercizio P7.20. Scrivete una classe `BarChart` dotata dei metodi

```
public void add(double value)
public void draw(Graphics2D g2)
```

che visualizzi un diagramma a barre dei valori che sono stati inseriti. Potete ipotizzare che tutti i valori siano positivi. Estendete proporzionalmente il diagramma nelle due direzioni in modo che riempia l'intero schermo, predeterminando il valore massimo dei dati e scalando opportunamente le barre.

***G **Esercizio P7.21.** Perfezionate la classe `BarChart` dell'esercizio precedente in modo che funzioni correttamente anche con valori negativi.

G **Esercizio P7.22. Scrivete una classe `PieChart` che visualizzi un diagramma a torta dei valori che sono stati inseriti e che sia dotata dei metodi:

```
public void add(double value)
public void draw(Graphics2D g2)
```

Potete ipotizzare che tutti i valori siano positivi.

Progetti di programmazione

Progetto 7.1. *Simulatore per il poker.* In questo progetto realizzerete un simulatore del popolare gioco d'azzardo solitamente chiamato "video poker". Il mazzo di carte ne contiene 52, 13 per ciascun seme, e viene mescolato all'inizio del gioco. Dovete individuare una modalità di mescolamento che sia equa, anche se non è necessario che sia efficiente. Successivamente, vengono mostrate le prime cinque carte del mazzo al giocatore, che ne può rifiutare alcune, anche tutte, o nessuna. Le carte rifiutate vengono sostituite con altre, prelevate ordinatamente dal mazzo. A questo punto viene assegnato alla mano, costituita dalle cinque carte, un punteggio, che deve essere uno dei seguenti:

- "Niente". La mano più bassa, che contiene cinque carte spaiate che non compongono alcuna delle mani elencate nel seguito.
- "Coppia". Due carte dello stesso valore, ad esempio due regine.
- "Doppia coppia". Due coppie, ad esempio due regine e due cinque.
- "Tris". Tre carte dello stesso valore, ad esempio tre regine.
- "Scala". Cinque carte con valori consecutivi, non del medesimo seme, come 4, 5, 6, 7 e 8. L'asso può precedere il 2 oppure seguire il re.
- "Colore". Cinque carte dello stesso seme, con valori non consecutivi.
- "Full". Un tris e una coppia, ad esempio tre regine e due 5.
- "Poker". Quattro carte con lo stesso valore, ad esempio quattro regine.
- "Scala colore". Una scala e, contemporaneamente, un colore: cinque carte con valori consecutivi e dello stesso seme.
- "Scala reale". La mano migliore possibile: 10, fante, regina, re e asso, tutti del medesimo seme.

Se vi piace l'azzardo, potete anche realizzare un sistema che consenta di effettuare puntate. Il giocatore paga un JavaDollaro per ogni partita e ne vince secondo questo schema di pagamento:

Mano	Vincita	Mano	Vincita
Scala reale	250	Scala	4
Scala colore	50	Tris	3
Poker	25	Doppia coppia	2
Full	6	Coppia almeno di fanti	1
Colore	5		

Figura 1
Cellule "vicine" di una cellula

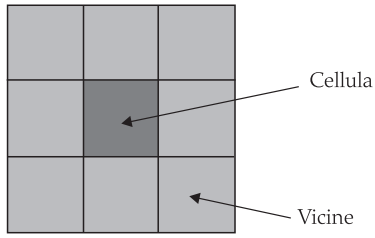
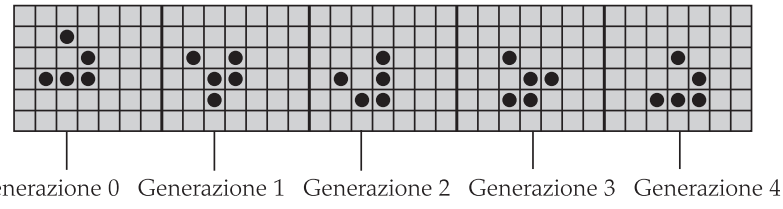


Figura 2
Configurazione "aliante" (*glider*)



Progetto 7.2. Il *Gioco della Vita* (*Game of Life*) è un gioco matematico molto conosciuto che genera un comportamento incredibilmente complesso, sebbene sia definibile con poche e semplici regole (in realtà, non è un vero e proprio gioco in senso tradizionale, perché non ha giocatori che gareggiano per vincere). Il gioco si svolge su una scacchiera rettangolare e ogni suo riquadro può essere vuoto o occupato da una cellula. All'inizio potete specificare in qualche modo quali siano i riquadri vuoti e quali siano, invece, occupati da una cellula; poi, il gioco procede in modo autonomo. A ogni *generazione* viene calcolata la generazione successiva di cellule: viene creata una nuova cellula in un riquadro vuoto solo se è circondato da esattamente tre riquadri occupati; una cellula muore per sovraffollamento se è circondata da quattro o più vicine, mentre muore

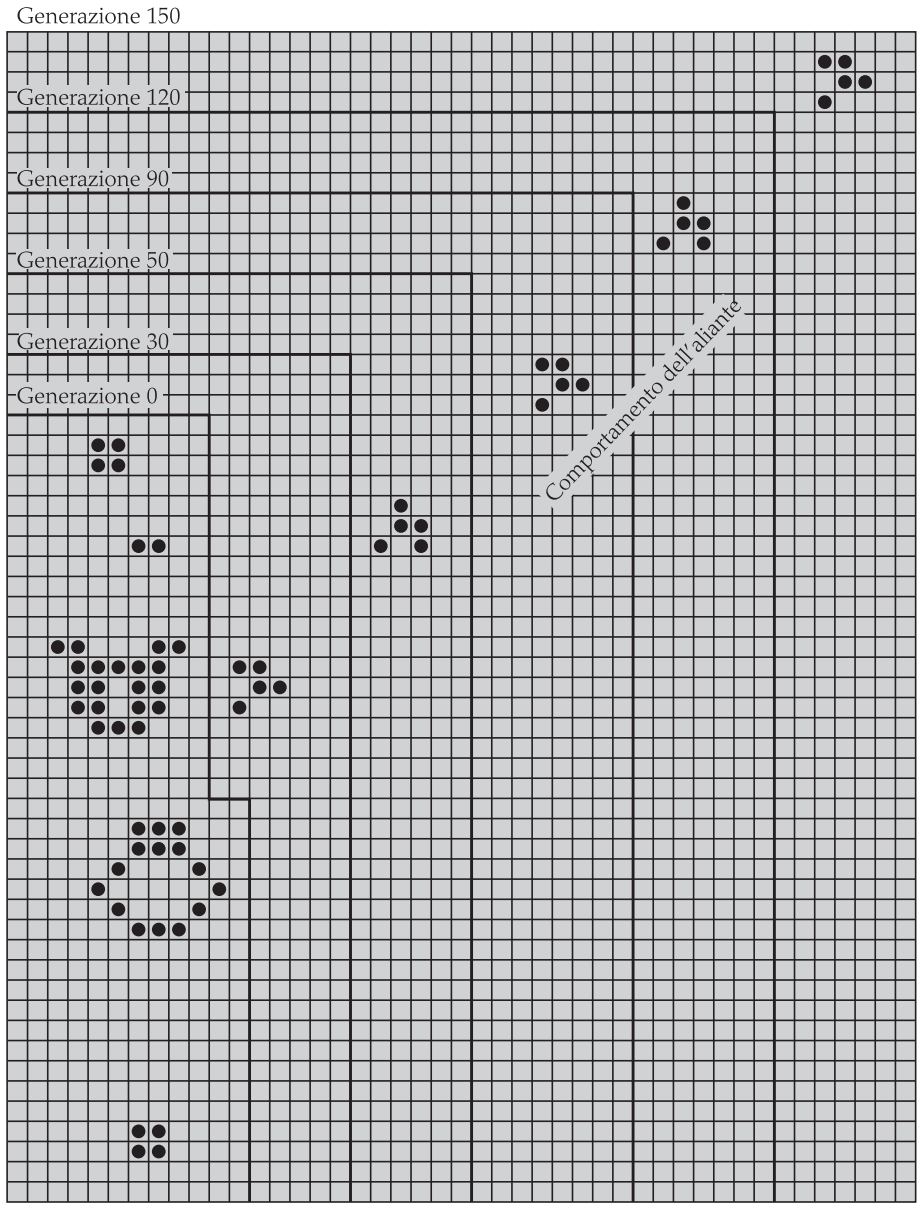


Figura 3
Configurazione "spara-alianti" (*glider gun*)

di solitudine se è circondata al più da una sola vicina. Una vicina è una cellula che si trova in un riquadro adiacente, a sinistra, a destra, sopra, sotto o in direzione diagonale. La Figura 1 mostra una cellula e le sue vicine.

Molte configurazioni, quando vengono assoggettate a queste regole, presentano un comportamento interessante. Nella Figura 2 si vede un *aliante* (*glider*), osservato in una successione di cinque generazioni: ogni quattro generazioni riprende la stessa forma, spostata di un riquadro verso il basso e verso destra.

Una delle configurazioni più sorprendenti è lo *spara-alianti* (*glider gun*), un complesso insieme di cellule che dopo 30 mosse ritorna alla sua forma iniziale, visibile nella Figura 3.

Scrivete un programma di gioco che elimini l'ingrato calcolo manuale delle generazioni che si susseguono, mostrandole sullo schermo in modo grafico. Usate un array bidimensionale per memorizzare la configurazione rettangolare. Otterrete un punteggio extra se realizzerete un'applicazione grafica in cui l'utente possa aggiungere o rimuovere cellule con una selezione del mouse.

Capitolo 8

Esercizi di programmazione

** **Esercizio P8.1.** Realizzate la classe `Coin` descritta nel Paragrafo 8.2 e modificate la classe `CashRegister` in modo che si possano aggiungere monete al registratore di cassa mediante il metodo

```
void enterPayment(int coinCount, Coin coinType)
```

Tale metodo va invocato più volte, una volta per ogni tipo di moneta presente nel pagamento.

** **Esercizio P8.2.** Modificate il metodo `giveChange` della classe `CashRegister` in modo che restituisca il numero di monete di un particolare tipo che vanno fornite come resto

```
void giveChange(Coin coinType)
```

Tale metodo va invocato una volta per ogni tipo di moneta, in ordine di valore decrescente.

* **Esercizio P8.3.** I veri registratori di cassa possono gestire monete e banconote. Progettate un'unica classe che rappresenti ciò che accomuna questi due concetti, poi riprogettate la classe `CashRegister` e dotatela di un metodo che consenta la registrazione di pagamenti descritti da tale classe. Uno degli obiettivi principali è l'identificazione di un nome significativo per la classe.

* **Esercizio P8.4.** Migliorate la classe `BankAccount` aggiungendo pre-condizioni per il metodo `deposit`, che richiede che il parametro `amount` non sia negativo, per il metodo `withdraw`, che richiede che `amount` sia non negativo e al massimo uguale al saldo attuale, e per il costruttore. Per verificare le pre-condizioni usate le asserzioni.

** **Esercizio P8.5.** Progettate i seguenti metodi, per calcolare il volume e l'area della superficie, rispettivamente, di una sfera con raggio `r`, di un cilindro con altezza `h` e base circolare di raggio `r` e di un cono con altezza `h` e base circolare di raggio `r`. Inseriteli nella classe `Geometry` e scrivete un programma che chieda all'utente di inserire i valori per `r` e per `h`, che invochi i sei metodi e che stampi i risultati.

- `public static double sphereVolume(double r)`
- `public static double sphereSurface(double r)`
- `public static double cylinderVolume(double r, double h)`
- `public static double cylinderSurface(double r, double h)`

- `public static double coneVolume(double r, double h)`
- `public static double coneSurface(double r, double h)`

★★ **Esercizio P8.6.** Risolvete l'esercizio precedente realizzando le classi `Sphere`, `Cylinder` e `Cone`. Quale approccio è maggiormente orientato agli oggetti?

★★ **Esercizio P8.7.** Modificate l'applicazione vista nei Consigli pratici 7.1, relativa a un registro scolastico, in modo che gestisca più studenti: per prima cosa chiedete all'utente di fornire i nomi di tutti gli studenti; poi, leggete i voti di tutte le singole prove d'esame, una dopo l'altra, chiedendo il punteggio di ciascuno studente (visualizzandone il nome); infine, visualizzate i nomi di tutti gli studenti, con il relativo voto finale. Usate un'unica classe, avente soltanto metodi statici.

★★★ **Esercizio P8.8.** Risolvete l'esercizio precedente usando più classi: modificate la classe `GradeBook` in modo che memorizzi un insieme di oggetti di tipo `Student`, ciascuno dei quali deve essere caratterizzato da un elenco di voti.

★★★ **Esercizio P8.9.** Progettate questi due metodi che calcolano il perimetro e l'area dell'ellisse `e`, aggiungendoli alla classe `Geometry`:

```
public static double perimeter(Ellipse2D.Double e)
public static double area(Ellipse2D.Double e)
```

La parte più difficile dell'esercizio è l'identificazione e la realizzazione di una formula corretta per il calcolo del perimetro. Perché in questo caso è sensato usare un metodo statico?

★★ **Esercizio P8.10.** Progettate questi due metodi che calcolano (in gradi) l'angolo compreso tra l'asse x e la retta che passa per due punti, e la pendenza (cioè il coefficiente angolare) di tale retta:

```
public static double angle(Point2D.Double p, Point2D.Double q)
public static double slope(Point2D.Double p, Point2D.Double q)
```

Aggiungete i metodi alla classe `Geometry` e descrivete pre-condizioni ragionevoli. Perché in questo caso è sensato usare un metodo statico?

★★★ **Esercizio P8.11.** Progettate questi due metodi che verificano se un punto si trova all'interno o sul contorno dell'ellisse `e`, aggiungendoli alla classe `Geometry`:

```
public static boolean isInside(Point2D.Double p, Ellipse2D.Double e)
public static boolean isOnBoundary(Point2D.Double p, Ellipse2D.Double e)
```

★ **Esercizio P8.12.** Progettate il metodo

```
public static int readInt(
      Scanner in, String prompt, String error, int min, int max)
```

che visualizza il messaggio (`prompt`), legge un numero intero e verifica se si trova tra il valore minimo e il valore massimo specificati. In caso negativo, stampa il messaggio d'errore (`error`) e ripete la lettura del dato d'ingresso. Inserite il metodo nella classe `Input`.

★★ **Esercizio P8.13.** Esaminate il seguente algoritmo per calcolare x^n, con n intero. Se $n < 0$, x^n è uguale a $1/x^{-n}$. Se n è positivo e pari, allora $x^n = (x^{n/2})^2$. Se n è positivo e dispari, allora $x^n = x^{n-1} \cdot x$. Progettate il metodo statico `double intPower(double x, int n)` che utilizzi questo algoritmo, inserendolo nella classe `Numeric`.

★★ **Esercizio P8.14.** Migliorate la classe `Needle` vista nel Capitolo 6, rendendo statica la variabile `generator`, in modo che tutti gli aghi condividano un unico generatore di numeri casuali.

★★ **Esercizio P8.15.** Realizzate le classi `CashRegister` e `Coin` descritte nell'Esercizio P8.1, inse-

rendole nel pacchetto `money` e lasciando, invece, la classe `CashRegisterTester` nel pacchetto predefinito.

* **Esercizio P8.16.** Inserite la classe `BankAccount` in un pacchetto il cui nome sia derivato dal vostro indirizzo di posta elettronica, come descritto nel Paragrafo 8.9, lasciando invece la classe `BankAccountTester` nel pacchetto predefinito.

T **Esercizio P8.17. Progettate con JUnit una classe di collaudo `BankTest` avente tre metodi, ciascuno dei quali collauda un diverso metodo della classe `Bank` vista nel Capitolo 7.

T **Esercizio P8.18. Progettate con JUnit una classe di collaudo `TaxReturnTest` avente tre metodi, ciascuno dei quali collauda una diversa situazione fiscale della classe `TaxReturn` vista nel Capitolo 5.

*G **Esercizio P8.19.** Scrivete i metodi seguenti per disegnare le lettere H, E, L e O in una finestra grafica; il parametro `p` indica il punto superiore sinistro del rettangolo che racchiude la lettera. Tracciate linee ed ellissi, ma non usate il metodo `drawString` né `System.out`. Poi, invocate ripetutamente tali metodi per disegnare le parole "HELLO" e "HOLE".

- `public static void drawH(Graphics2D g2, Point2D.Double p);`
- `public static void drawE(Graphics2D g2, Point2D.Double p);`
- `public static void drawL(Graphics2D g2, Point2D.Double p);`
- `public static void drawO(Graphics2D g2, Point2D.Double p);`

G **Esercizio P8.20. Ripetete l'esercizio precedente progettando le classi `LetterH`, `LetterE`, `LetterL` e `LetterO`, ciascuna con un costruttore che riceve un parametro di tipo `Point2D.Double` (il punto superiore sinistro del rettangolo che racchiude la lettera) e un metodo `draw(Graphics2D g2)`. Quale soluzione è più orientata agli oggetti?

Progetti di programmazione

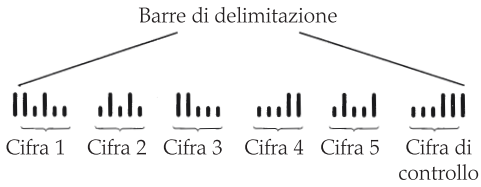
Progetto 8.1. Realizzate un programma che visualizzi le buste paga per un gruppo di tutori didattici, applicando le appropriate deduzioni corrispondenti alle tasse federali e per il servizio sanitario statunitense (per il calcolo delle tasse potete usare lo schema visto nel Capitolo 5, mentre per il servizio sanitario consultate Internet, cercando *Social Security Tax*). Per ciascun tutore il programma deve chiedere, come dati in ingresso, il nome, la tariffa oraria e il numero di ore lavorate.

Progetto 8.2. Per ordinare più velocemente le lettere, il Servizio Postale degli Stati Uniti incoraggia le aziende che spediscono grossi volumi di posta a usare un codice a barre per indicare il codice ZIP (il codice di avviamento postale , come si può vedere in Figura 4).

Lo schema di codifica per un codice ZIP di cinque cifre è illustrato nella Figura 5. A ciascuna estremità si trova una barra di delimitazione ad altezza intera. Le cinque cifre codificate sono seguite da una cifra di controllo, calcolata nel modo seguente: si sommano tutte le cifre e si sceglie la cifra di controllo che, sommata al totale, restituisca un multiplo di dieci. Per esempio, la somma di tutte le cifre del codice ZIP 95014 è uguale a 19, quindi la cifra di controllo è uno, perché porta il totale a 20.

Figura 4
Codice postale a barre

Figura 5
Codifica di cinque cifre mediante codice a barre



Ciascuna cifra del codice ZIP, così come la cifra di controllo, è codificata secondo lo schema della tabella seguente, in cui zero indica una mezza barra e uno indica una barra ad altezza intera:

	7	**4**	**2**	**1**	**0**
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	1	0	0	0	1
8	1	0	0	1	0
9	1	0	1	0	0
0	1	1	0	0	0

Notate che queste codifiche sono tutte le dieci combinazioni possibili di due barre intere e di tre mezze barre. Risalire a una cifra così codificata è semplice, utilizzando i pesi attribuiti alle colonne: 7, 4, 2, 1 e 0. Per esempio, la codifica 01100 equivale a $0 \times 7 + 1 \times 4 + 1 \times 2 + 0 \times 1 + 0 \times 0 = 6$. L'unica eccezione è la cifra zero, che, decodificata mediante la formula dei pesi, restituirebbe 11.

Scrivete un programma che chieda all'utente di inserire un codice ZIP e che stampi il relativo codice a barre. Usate il carattere "due punti" per indicare le mezze barre e il carattere ¦ per le barre intere. Per esempio, il codice 95014 diventa:

```
¦¦:¦:::¦:¦:¦¦:::::::¦¦:¦::¦:::¦¦¦
```

(in alternativa, progettate un'applicazione grafica che visualizzi barre vere).

Il programma deve anche poter leggere un codice a barre (costruito mediante i "due punti" per le mezze barre e il carattere ¦ per le barre intere) e stampare il codice ZIP corrispondente, visualizzando un messaggio di errore se la sequenza di caratteri in ingresso non è valida.

Capitolo 9

Esercizi di programmazione

★ **Esercizio P9.1.** Modificate la classe `Die` del Capitolo 6 in modo che realizzi l'interfaccia `Measurable`. Create alcuni dadi, effettuate i cast necessari e aggiungeteli alla prima realizzazione della classe `DataSet` (Paragrafo 9.1). Visualizzate la media.

★ **Esercizio P9.2.** Progettate una classe `Quiz` che realizzi l'interfaccia `Measurable`. Un quiz ha un punteggio e un voto in lettere (come `B+`). Usate la prima realizzazione della classe `DataSet`

(Paragrafo 9.1) per elaborare un insieme di quiz, visualizzando il punteggio medio e il quiz con il punteggio massimo, sia in lettere che in numeri.

* **Esercizio P9.3.** Progettate una classe `Person`: una persona è caratterizzata da un nome e un'altezza in centimetri. Usate la seconda realizzazione della classe `DataSet` (Paragrafo 9.4) per elaborare un insieme di oggetti di tipo `Person`, visualizzando l'altezza media delle persone e il nome della persona più alta.

* **Esercizio P9.4.** Modificate la prima realizzazione della classe `DataSet` (quella che elabora oggetti di tipo `Measurable` vista nel Paragrafo 9.1) in modo che calcoli anche l'elemento di misura minima.

* **Esercizio P9.5.** Modificate la seconda realizzazione della classe `DataSet` (quella che usa oggetti di tipo `Measurer` vista nel Paragrafo 9.4) in modo che calcoli anche l'elemento di misura minima.

* **Esercizio P9.6.** Usando un diverso oggetto di tipo `Measurer`, elaborate un insieme di oggetti di tipo `Rectangle`, trovando il rettangolo con il perimetro maggiore.

*** **Esercizio P9.7.** Migliorate la classe `DataSet` in modo che possa essere usata con un oggetto di tipo `Measurer` oppure per elaborare oggetti di tipo `Measurable`. *Suggerimento*: fornite un costruttore senza parametri che crei un oggetto di tipo `Measurer`, il quale elabori oggetti di tipo `Measurable`.

* **Esercizio P9.8.** Modificate il metodo `display` della classe `LastDigitDistribution` vista in Esempi completi 9.1 in modo che produca un istogramma, come questo:

```
0: *************
1: ******************
2: *************
```

Dimensionate le barre in scala, in modo che la più lunga sia composta da 40 caratteri.

** **Esercizio P9.9.** Progettate una classe `PrimeSequence` che realizzi l'interfaccia `Sequence` vista in Esempi completi 9.1, generando la sequenza dei numeri primi.

* **Esercizio P9.10.** Aggiungete all'interfaccia `Sequence` di Esempi completi 9.1 il metodo `hasNext`, che restituisca `false` se la sequenza non ha più valori. Progettate una classe `MySequence` che produca una sequenza di dati reali, a vostra scelta, come la popolazione di città o nazioni, valori di temperatura o quotazioni di borsa. Ricavate i dati da Internet e inseriteli in un array, restituendo un valore diverso ad ogni invocazione di `next`, fino all'esaurimento dei dati stessi. La vostra classe `SequenceDemo` deve visualizzare la distribuzione dell'ultima cifra di tutti i valori della sequenza.

* **Esercizio P9.11.** Progettate una classe `FirstDigitDistribution` che funzioni come la classe `LastDigitDistribution` vista in Esempi completi 9.1, tranne per il fatto di prendere in esame la prima cifra di ciascun valore, anziché l'ultima. È ben noto che le prime cifre di valori casuali *non* hanno una distribuzione uniforme: questo fatto è stato usato per rilevare frodi bancarie, evidenziate da distribuzioni anomale delle prime cifre di importi relativi a sequenze di transazioni.

** **Esercizio P9.12.** Definite un'interfaccia `Filter` in questo modo:

```
public interface Filter
{
   boolean accept(Object x);
}
```

Modificate la seconda realizzazione della classe `DataSet` (Paragrafo 9.4) in modo che usi sia un oggetto di tipo `Measurer` sia un oggetto di tipo `Filter`, elaborando soltanto oggetti accettati dal

filtro. Mostrate il corretto funzionamento della vostra modifica usando un insieme di conti bancari, nel quale vengano ignorati tutti i conti con saldo minore di $ 1000.

** **Esercizio P9.13.** La libreria standard di Java contiene l'interfaccia `Comparable`:

```
public interface Comparable
{
   /**
      Confronta questo oggetto con un altro.
      @param other l'oggetto da confrontare
      @return un numero intero negativo, zero o positivo se, rispettivamente,
              questo oggetto è minore, uguale o maggiore dell'altro
   */
   int compareTo(Object other);
}
```

Modificate la classe `DataSet` del Paragrafo 9.1 in modo che accetti oggetti di tipo `Comparable`. Con questa interfaccia non ha più senso calcolare la media: la classe `DataSet` deve ora memorizzare il valore minimo e il valore massimo. Collaudate la vostra classe `DataSet`, così modificata, aggiungendo a un suo esemplare un certo numero di oggetti di tipo `String` (la classe `String` realizza l'interfaccia `Comparable`).

* **Esercizio P9.14.** Modificate la classe `Coin` in modo che realizzi l'interfaccia `Comparable` descritta nell'esercizio precedente.

** **Esercizio P9.15.** Il metodo `System.out.printf` dispone di formati predefiniti per visualizzare numeri interi, numeri in virgola mobile e altri tipi di dati, ma può anche essere esteso: usando il formato `S`, potete visualizzare qualunque classe che realizzi l'interfaccia `Formattable`, dotata di un unico metodo:

```
void formatTo(Formatter formatter, int flags, int width, int precision)
```

In questo esercizio dovete modificare la classe `BankAccount` in modo che realizzi l'interfaccia `Formattable`. Ignorate i parametri `flags` e `precision`, visualizzando semplicemente il saldo del conto usando la larghezza specificata da `width`. Per assolvere a questo compito avete bisogno di un riferimento di tipo `Appendable`, come questo:

```
Appendable a = formatter.out();
```

`Appendable` è un'altra interfaccia, dotata dell'unico metodo

```
void append(CharSequence sequence)
```

`CharSequence` è, a sua volta, un'interfaccia, realizzata da molte classi, tra cui la classe `String`. Per prima cosa costruite una stringa convertendo il saldo del conto, per poi aggiungere spazi fino al raggiungimento della larghezza desiderata. Infine, passate tale stringa al metodo `append`.

*** **Esercizio P9.16.** Migliorate il metodo `formatTo` dell'esercizio precedente, in modo che prenda in considerazione anche il parametro `precision`.

*T **Esercizio P9.17.** Avete il compito di scrivere un programma che giochi a TicTacToe ("tris") contro un giocatore umano. Una classe `TicTacToeUI`, che realizza l'interfaccia grafica, legge le mosse dell'utente e visualizza quelle del computer su una scacchiera. La classe `TicTacToeStrategy` determina la mossa successiva del computer, mentre la classe `TicTacToeBoard` rappresenta lo stato attuale della scacchiera. Completate tutte le classi, tranne quella che si occupa della strategia, per la quale usate una classe semplificata che sceglie la prima casella libera.

****T** **Esercizio P9.18.** Dovete tradurre in HTML un testo disponibile nel sito del Progetto Gutenberg (`http://gutenberg.org`). Ecco, ad esempio, l'inizio del primo capitolo di Anna Karenina, di Tolstoy:

```
Chapter 1

Happy families are all alike; every unhappy family is unhappy in
its own way.

Everything was in confusion in the Oblonskys' house. The wife
had discovered that the husband was carrying on an intrigue with
a French girl, who had been a governess in their family, and she
had announced to her husband that she could not go on living in
the same house with him ...
```

L'equivalente in HTML è:

```
<h1>Chapter 1</h1>
<p>Happy families are all alike; every unhappy family is unhappy in
its own way.</p>
<p>Everything was in confusion in the Oblonskys&rsquo house. The wife
had discovered that the husband was carrying on an intrigue with
a French girl, who had been a governess in their family, and she
had announced to her husband that she could not go on living in
the same house with him ...</p>
```

La conversione in HTML può essere portata a termine in due fasi. Dapprima il testo viene decomposto in *segmenti*, cioè blocchi di testo dello stesso tipo (intestazione, paragrafo, e così via), poi ciascun segmento viene convertito, racchudendolo tra i corretti marcatori HTML e convertendo i caratteri speciali presenti.

Recuperare il testo da Internet e scomporlo in segmenti è un compito arduo, per cui occorre definire un'interfaccia e progettare un oggetto semplificato, con cui collaudare la classe che si occupa di portare a termine la conversione in HTML.

Testo originario	HTML
“ ”	`&ldquo;` (a sinistra) oppure `&rdquo;` (a destra)
‘ ’	`&lsquo;` (a sinistra) oppure `&rsquo;` (a destra)
—	`&emdash;`
<	`<`
>	`>`
&	`&`

*****G** **Esercizio P9.19.** Scrivete un metodo `randomShape` che generi casualmente oggetti che realizzano l'interfaccia `Shape`: un miscuglio di rettangoli, ellissi e linee, con posizioni casuali. Invocatelo 10 volte e disegnate tutte le forme.

***G** **Esercizio P9.20.** Migliorate il programma `ButtonViewer` in modo che visualizzi il messaggio "I was clicked *n* times!" ogni volta che viene premuto il pulsante. Il valore di *n* deve aumentare a ogni pressione.

****G** **Esercizio P9.21.** Migliorate il programma `ButtonViewer` in modo che abbia due pulsanti, ciascuno dei quali visualizzi il messaggio "I was clicked *n* times!" ogni volta che viene premuto. Ogni pulsante deve avere il proprio contatore.

****G** **Esercizio P9.22.** Migliorate il programma `ButtonViewer` in modo che abbia due pulsanti etichettati come A e B, ciascuno dei quali, quando premuto, visualizzi il messaggio "Button *x* was clicked!", dove *x* è A o B.

****G** **Esercizio P9.23.** Realizzate il programma `ButtonViewer` dell'esercizio precedente usando una sola classe come ricevitore di eventi.

***G** **Esercizio P9.24.** Migliorate il programma `ButtonViewer` in modo che visualizzi l'istante di tempo in cui è avvenuta la pressione del pulsante.

*****G** **Esercizio P9.25.** Realizzate la classe `AddInterestListener` del programma `InvestmentViewer1` in modo che sia una classe normale, anziché una classe interna. *Suggerimento*: memorizzate un riferimento al conto bancario, aggiungendo al ricevitore un costruttore che imposti tale riferimento.

*****G** **Esercizio P9.26.** Realizzate la classe `AddInterestListener` del programma `InvestmentViewer2` in modo che sia una classe normale, anziché una classe interna. *Suggerimento*: memorizzate riferimenti al conto bancario e all'etichetta, aggiungendo al ricevitore un costruttore che imposti tali riferimenti.

****G** **Esercizio P9.27.** Scrivete un programma che visualizzi la crescita di una popolazione di pesci. Iniziate con due pesci e raddoppiate il numero di pesci presenti a ogni "click" su un pulsante del mouse.

****G** **Esercizio P9.28.** Scrivete un programma che usi un temporizzatore per stampare l'ora esatta una volta ogni secondo. *Suggerimento*: il codice seguente stampa l'ora esatta e la classe `Date` si trova nel pacchetto `java.util`:

```
Date now = new Date();
System.out.println(now);
```

*****G** **Esercizio P9.29.** Modificate la classe `RectangleComponent` del programma di animazione visto nel Paragrafo 9.10 in modo che il rettangolo rimbalzi ai bordi del componente, invece di uscire.

****G** **Esercizio P9.30.** Scrivete un programma che faccia muovere un'automobile all'interno di una finestra frame.

*****G** **Esercizio P9.31.** Scrivete un programma che faccia muovere all'interno di una finestra frame due automobili, in direzioni opposte ma ad altezze diverse, in modo che non si scontrino.

***G** **Esercizio P9.32.** Modificate la classe `RectangleComponent` del programma che gestisce eventi del mouse visto nel Paragrafo 9.11, in modo che ogni "click" del mouse provochi l'aggiunta di un nuovo rettangolo al componente visualizzatore. *Suggerimento*: usate un `ArrayList<Rectangle>` e disegnate tutti i rettangoli all'interno del metodo `paintComponent`.

***G** **Esercizio P9.33.** Scrivete un programma che chieda all'utente di fornire le coordinate *x* e *y* del centro di un cerchio e la lunghezza del suo raggio, mediante `JOptionPane`. Quando l'utente preme il pulsante "Draw", il programma deve chiedere i dati all'utente e disegnare all'interno di un componente grafico un cerchio avente tale centro e raggio.

****G** **Esercizio P9.34.** Scrivete un programma che chieda all'utente di specificare il raggio di un cerchio mediante `JOptionPane`, indicandone poi il centro con un "click" del mouse all'interno del componente grafico. In questo caso non serve un pulsante "Draw".

*****G** **Esercizio P9.35.** Scrivete un programma che consenta all'utente di specificare posizione e dimensione di un cerchio mediante due "click" del mouse, il primo che identifica la posizione del centro del cerchio e il secondo che si trova sulla sua circonferenza. *Suggerimento*: nel gestore di

eventi di pressione del pulsante del mouse, dovete tenere traccia del fatto che sia già stata registrata la posizione del centro del cerchio in un evento precedente.

Progetti di programmazione

Progetto 9.1. Progettate un'interfaccia `MoveableShape` che possa essere utilizzata come meccanismo generale per l'animazione di una forma geometrica. Una tale forma mobile deve avere due metodi: `move` e `draw`. Scrivete una generica classe `AnimationPanel` che disegni e sposti qualsiasi oggetto di tipo `MoveableShape` (oppure un vettore di oggetti di tipo `MoveableShape`, se avete studiato il Capitolo 7). Progettate poi le sagome mobili di un rettangolo e di un'automobile.

Progetto 9.2. Avete il compito di progettare un programma generico che sia in grado di gestire giochi da tavolo con due giocatori. Il programma dovrebbe essere sufficientemente flessibile da poter gestire giochi come il tic-tac-toe, gli scacchi o il Gioco di Nim visto nel Progetto 6.2.

Progettate un'interfaccia `Game` che descriva un gioco da tavolo, pensando a ciò che deve fare il vostro programma. Ad esempio, deve chiedere al primo giocatore di fare una mossa, cioè di fornire in ingresso una stringa che rispetti un formato specifico di ciascun gioco, che per gli scacchi potrebbe essere `Be3`. Il programma non ha alcuna conoscenza dei singoli giochi, per cui l'interfaccia `Game` deve avere un metodo come

```
boolean isValidMove(String move)
```

Dopo che una mossa è stata ritenuta valida, deve essere eseguita: l'interfaccia ha quindi bisogno di un altro metodo, che possiamo chiamare `executeMove`. Il programma deve poi verificare se il gioco è terminato e, in caso contrario, deve elaborare la mossa dell'altro giocatore. Dovete anche fornire un meccanismo che consenta la visualizzazione dello stato della tavola da gioco.

Progettate l'interfaccia `Game` e fornitene due realizzazioni, che potrebbero essere `Nim` e `Chess` (oppure `TicTacToe` se siete meno ambiziosi). La classe `GamePlayer` dovrebbe essere in grado di gestire un riferimento di tipo `Game` senza sapere quale gioco si sta giocando, elaborando le mosse di entrambi i giocatori. Scrivete poi due programmi che differiscano solamente per la fase di inizializzazione del riferimento di tipo `Game`.

Capitolo 10

Esercizi di programmazione

* **Esercizio P10.1.** Migliorate il metodo `addInterest` della classe `SavingsAccount` in modo che calcoli gli interessi in base al saldo *minimo* che è stato raggiunto nel periodo intercorso dalla sua ultima invocazione. *Suggerimento*: dovete modificare anche il metodo `withdraw` e dovete aggiungere una variabile di esemplare che memorizzi il saldo minimo.

** **Esercizio P10.2.** Aggiungete la classe `TimeDepositAccount` alla gerarchia dei conti bancari. Tale "conto di deposito vincolato" è identico a un conto di risparmio, a parte il fatto che si promette di lasciare il denaro nel conto per un prefissato numero di mesi ed è prevista una penale di $ 20 in caso di prelievo prematuro. Nella costruzione di un conto si specificano il tasso di interesse e il numero di mesi in cui agisce il vincolo. Nel metodo `addInterest` decrementate il conteggio dei mesi. Se il conteggio è positivo nel corso di un prelievo, addebitate la penale.

* **Esercizio P10.3.** Aggiungete alla gerarchia di domande vista in Consigli pratici 10.1 la classe `NumericQuestion`. Se la differenza tra la risposta e il valore previsto non è maggiore di 0.01, accettate la risposta come corretta.

** **Esercizio P10.4.** Aggiungete alla gerarchia di domande vista in Consigli pratici 10.1 la classe `FillInQuestion`. Un oggetto di questa classe viene costruito fornendo una stringa che contiene la risposta corretta preceduta e seguita da un carattere di sottolineatura, come `"The inventor of Java was _James Gosling_"`. La domanda va visualizzata così:

```
The inventor of Java was_____
```

* **Esercizio P10.5.** Modificate il metodo `checkAnswer` della classe `Question` vista in Consigli pratici 10.1 in modo che ignori eventuali spazi vuoti e la differenza tra lettere maiuscole e minuscole. Ad esempio, la risposta `" JAMES gosling"` dovrebbe essere ritenuta equivalente a `"James Gosling"`.

* **Esercizio P10.6.** Aggiungete alla gerarchia di domande vista in Consigli pratici 10.1 la classe `MultiChoiceQuestion`, che consenta domande con più risposte corrette, scelte tra quelle proposte. L'esaminato deve fornire tutte (e sole) le risposte corrette, separate da spazi. Fornite istruzioni adeguate nel testo della domanda.

* **Esercizio P10.7.** Aggiungete alla gerarchia di domande vista in Consigli pratici 10.1 la classe `AnyCorrectChoiceQuestion`, che consenta domande con più risposte corrette, scelte tra quelle proposte. L'esaminato deve fornire una qualsiasi delle risposte corrette. La stringa che memorizza le risposte corrette deve contenerle tutte, separate da spazi.

* **Esercizio P10.8.** Aggiungete il metodo `addText` alla classe `Question` vista in Consigli pratici 10.1 e progettate una versione diversa di `ChoiceQuestion`, che invochi `addText` invece di memorizzare al proprio interno un vettore con le risposte disponibili.

** **Esercizio P10.9.** Aggiungete i metodi `toString` e `equals` alle classi `Question` e `ChoiceQuestion` viste in Consigli pratici 10.1.

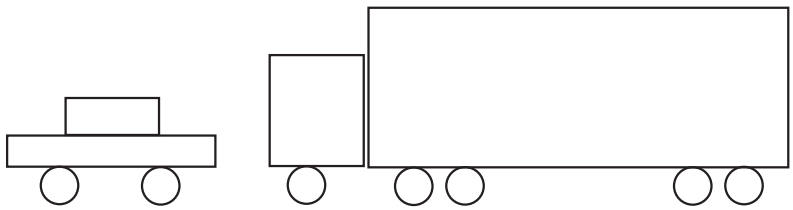
* **Esercizio P10.10.** Realizzate una classe `Square` che estenda la classe `Rectangle`. Nel costruttore ricevete le coordinate x e y del *centro* e la lunghezza del lato del quadrato. Invocate i metodi `setLocation` e `setSize` della classe `Rectangle`, dopo averli cercati nella relativa documentazione. Fornite anche un metodo `getArea` che calcoli e restituisca l'area del quadrato. Scrivete un programma di esempio che chieda all'utente il centro e la lunghezza del lato, per poi stampare il quadrato (usando il metodo `toString` ereditato da `Rectangle`) e la sua area.

* **Esercizio P10.11.** Realizzate una classe `Person` e due classi, `Student` e `Instructor`, che ereditino da essa. Una persona ha un nome e un anno di nascita, uno studente ha una disciplina di specializzazione e un istruttore ha un salario. Scrivete le dichiarazioni delle classi, i costruttori e il metodo `toString` per tutte le classi. Fornite un programma di prova per collaudare classi e metodi.

** **Esercizio P10.12.** Progettate una classe `Employee` in cui ciascun dipendente abbia un nome e una retribuzione. Progettate, quindi, una classe `Manager` che erediti da `Employee`: aggiungete una variabile di esemplare di tipo `String`, chiamata `department`, e scrivete un metodo `toString` che consenta di stampare il nome, il reparto e la retribuzione. Progettate, infine, una classe `Executive` che erediti da `Manager`. Fornite un metodo `toString` appropriato in tutte le classi e scrivete un programma di prova per collaudare classi e metodi.

*** **Esercizio P10.13.** Riorganizzate le classi dei conti bancari nel modo seguente. Nella classe `BankAccount`, inserite un metodo astratto `endOfMonth`, privo di implementazione. Nelle sottoclassi, cambiate i nomi dei metodi `addInterest` e `deductFees`, facendoli diventare `endOfMonth`. Quali classi sono ora astratte e quali sono concrete? Scrivete un metodo statico `void test(BankAccount account)` che esegua cinque transazioni e invochi `endOfMonth`. Collaudatelo con esemplari di tutte le classi concrete di conti bancari.

***G **Esercizio P10.14.** Realizzate la classe astratta `Vehicle` e le sue sottoclassi concrete `Car` e `Truck`. Un veicolo occupa una posizione sullo schermo. Scrivete i metodi `draw` necessari per disegnare automobili e autocarri come segue:



Car Truck

Scrivete, poi, un metodo `randomVehicle` che generi a caso riferimenti a oggetti di tipo `Vehicle`, metà dei quali siano automobili e l'altra metà autocarri, con posizioni casuali. Invocate il metodo dieci volte e disegnate tutti gli oggetti restituiti.

G **Esercizio P10.15. Scrivete un programma che, usando `JOptionPane`, chieda all'utente un numero intero e che, poi, disegni tanti rettangoli quanti ne sono stati richiesti dall'utente. I rettangoli devono comparire in posizioni casuali all'interno di un componente grafico. Progettate la classe del frame usando l'ereditarietà.

G **Esercizio P10.16. Scrivete un programma che chieda all'utente di inserire un numero intero n e, poi, disegni una scacchiera $n \times n$. Progettate la classe del frame usando l'ereditarietà.

Progetti di programmazione

Progetto 10.1. Avete il compito di programmare robot con vari comportamenti, che cerchino di uscire da un labirinto (*maze*) di questo tipo:

```
* *******
*     * *
* ***** *
* * *   *
* * *** *
*   *   *
*** * * *
*     * *
******* *
```

Un robot ha una posizione e un metodo `void move(Maze m)` che la modifica. Progettate una superclasse comune, `Robot`, il cui metodo `move` non fa nulla. Progettate le sottoclassi `RandomRobot`, `RightHandRuleRobot` e `MemoryRobot`, che rappresentino robot che agiscono con differenti strategie per uscire dai labirinti. Un `RandomRobot` effettua semplicemente movimenti casuali. Un `RightHandRuleRobot` si sposta nel labirinto in modo che la sua mano destra sia sempre in contatto con una parete. Un `MemoryRobot` si ricorda di tutte le posizioni in cui si è già trovato in precedenza e non imbocca mai per la seconda volta una strada che sa essere senza uscita.

Progetto 10.2. Realizzate i metodi `toString`, `equals` e `clone` per tutte le sottoclassi della classe `BankAccount`, oltre che per la classe `Bank` vista nel Capitolo 7. Scrivete collaudi di unità che verifichino il corretto funzionamento dei metodi. Siate certi di collaudare un esemplare di `Bank` che contenga esemplari di conti di tipo diverso.

Capitolo 11

Esercizi di programmazione

** **Esercizio P11.1.** Scrivete un programma che chieda all'utente il nome di un file e ne visualizzi il numero di caratteri, parole e righe.

** **Esercizio P11.2.** Scrivete un programma che chieda all'utente il nome di un file e ne conti il numero di caratteri, parole e righe, poi il programma chiederà il nome del file successivo; quando l'utente inserirà il nome di un file che non esiste (come, per esempio, una stringa vuota), il programma terminerà la propria esecuzione visualizzando il conteggio totale, per tutti i file letti, del numero di caratteri, parole e righe.

** **Esercizio P11.3.** Scrivete un programma `CopyFile` che copia un file in un altro. I nomi dei file sono specificati sulla riga dei comandi, per esempio così:

```
java CopyFile report.txt report.sav
```

** **Esercizio P11.4.** Scrivete un programma che *concateni* i contenuti di più file, generando un unico file. Per esempio

```
java CatFiles chapter1.txt chapter2.txt chapter3.txt book.txt
```

deve creare un lungo file, `book.txt`, che comprende, ordinatamente, i contenuti dei file `chapter1.txt`, `chapter2.txt` e `chapter3.txt`. Il file di destinazione è sempre l'ultimo indicato sulla riga dei comandi.

** **Esercizio P11.5.** Scrivete un programma `Find` che cerchi in tutti i file specificati sulla riga dei comandi e visualizzi tutte le righe contenenti una parola specifica, fornita come primo argomento sulla riga dei comandi. Per esempio, se eseguite

```
java Find ring report.txt address.txt Homework.java
```

il programma potrebbe stampare

```
report.txt: has broken an international ring of DVD bootleggers that
address.txt: Kris Kringle, North Pole
address.txt: Homer Simpson, Springfield
Homework.java: String filename;
```

** **Esercizio P11.6.** Scrivete un programma che controlli l'ortografia: deve leggere ogni parola di un file e controllare se essa è presente in un elenco di parole, disponibile sulla maggior parte dei sistemi UNIX nel file `/usr/dict/words` (e facilmente reperibile anche in Internet). Il programma dovrebbe stampare tutte le parole che non riesce a trovare nell'elenco.

** **Esercizio P11.7.** Scrivete un programma che inverta ogni riga di un file. Per esempio, se eseguite

```
java Reverse HelloPrinter.java
```

il contenuto di `HelloPrinter.java` diventa

```
retnirPolleH ssalc cilbup
{
)sgra ][gnirtS(niam diov citats cilbup
{
;)"n\!dlroW ,olleH"(tnirp.tuo.metsyS
}
}
```

Naturalmente, se eseguite due volte `Reverse` sullo stesso file, ottenete di nuovo l'originale.

* **Esercizio P11.8.** Recuperate i dati relativi ai nomi usati nei decenni scorsi (Social Security Administration), inserendoli in file di nome `babynames80s.txt`, ecc. Modificate il programma `BabyNames.java` in modo che chieda all'utente il nome di un file. I numeri all'interno dei file sono separati da virgole, quindi dovete modificare il programma in modo che gestisca questo formato. Siete in grado di individuare una tendenza, osservando le frequenze?

* **Esercizio P11.9.** Scrivete un programma che legga un file nel formato di `babynames.txt` e generi due file, `boynames.txt` e `girlnames.txt`, separando i dati relativi a maschi e femmine.

** **Esercizio P11.10.** Scrivete un programma che legga un file nel formato di `babynames.txt` e visualizzi tutti i nomi che sono sia maschili sia femminili (come Alexis o Morgan).

*** **Esercizio P11.11.** Scrivete un programma che sostituisca in un file tutti i caratteri di tabulazione `'\t'` con l'*adeguato* numero di spazi. La distanza predefinita fra colonne di tabulazione è costante e pari a 3, il valore che usiamo in questo libro per i programmi Java, ma l'utente può specificare un valore diverso. Sostituite i caratteri di tabulazione con il numero di spazi necessari per passare alla successiva colonna di tabulazione: potrebbe trattarsi di *meno* di tre spazi. Per esempio, considerate la riga `"\t¦\t¦¦\t¦"`: il primo carattere di tabulazione viene sostituito da tre spazi, il secondo da due spazi e il terzo da uno spazio. Il programma viene eseguito in questo modo:

```
java TabExpander nomeFile
```

oppure:

```
java TabExpander –t ampiezzaDiTabulazione nomeFile
```

* **Esercizio P11.12.** Modificate la classe `BankAccount` in modo che lanci un'eccezione di tipo `IllegalArgumentException` quando viene costruito un conto con saldo negativo, quando viene versata una somma negativa o quando viene prelevata una somma che non sia compresa tra 0 e il saldo del conto. Scrivete un programma di prova che provochi il lancio di tutte e tre le eccezioni, catturandole.

** **Esercizio P11.13.** Ripetete l'esercizio precedente, ma lanciate eccezioni di tre tipi definiti da voi.

** **Esercizio P11.14.** Scrivete un programma che chieda all'utente di inserire un insieme di valori in virgola mobile. Quando viene inserito un valore che non è un numero, date all'utente una seconda possibilità di introdurre un valore corretto e, dopo due tentativi, fate terminare il programma. Sommate tutti i valori specificati in modo corretto e, quando l'utente ha terminato di inserire dati, visualizzate il totale. Usate la gestione delle eccezioni per identificare i valori di ingresso non validi.

** **Esercizio P11.15.** Ripetete l'esercizio precedente, ma consentite all'utente tutti i tentativi necessari per inserire un valore corretto. Terminate il programma soltanto quando l'utente inserisce una riga vuota.

* **Esercizio P11.16.** Modificate la classe `DataSetReader` in modo che non invochi `hasNextInt` né `hasNextDouble`. Semplicemente, lasciate che `nextInt` e `nextDouble` lancino un'eccezione di tipo `NoSuchElementException`, catturandola nel metodo `main`.

** **Esercizio P11.17.** Scrivete un programma che legga un insieme di descrizioni di monete da un file avente il formato seguente:

```
nomeMoneta1 valoreMoneta1
nomeMoneta2 valoreMoneta2
...
```

Aggiungete alla classe `Coin` il metodo

```
void read(Scanner in) throws FileNotFoundException
```

che lanci un'eccezione se la riga letta non ha il formato corretto. Quindi, realizzate il metodo

```
static ArrayList<Coin> readFile(String filename)
   throws FileNotFoundException
```

Nel metodo `main` invocate `readFile`. Se viene lanciata un'eccezione, date all'utente la possibilità di selezionare un altro file. Se tutte le monete vengono lette correttamente, visualizzate il loro valore totale.

*** **Esercizio P11.18.** Progettate una classe `Bank` che contenga un certo numero di conti bancari. Ciascun conto ha un numero di conto e un saldo. Aggiungete una variabile di esemplare `accountNumber` alla classe `BankAccount` e memorizzate i conti bancari in un vettore. Scrivete un metodo `readFile` per la classe `Bank` che legga un file scritto con il seguente formato:

```
numeroDiConto1 saldo1
numeroDiConto2 saldo2
...
```

Realizzate metodi `read` nelle classi `Bank` e `BankAccount`. Scrivete un programma di prova che legga un file di conti correnti, per poi visualizzare il conto con il saldo maggiore. Se il file non contiene dati corretti, date all'utente la possibilità di selezionare un diverso file.

Progetti di programmazione

Progetto 11.1. Con la seguente sequenza di comandi potete leggere i contenuti di una pagina web:

```
String address = "http://java.sun.com/index.html";
URL u = new URL(address);
Scanner in = new Scanner(u.openStream());
...
```

Alcuni di questi metodi possono lanciare eccezioni: consultate la documentazione della libreria standard. Progettate una classe `LinkFinder` che cerchi tutti i collegamenti ipertestuali (*hyperlink*), aventi la forma:

```
<a href="collegamento">testo del collegamento</a>
```

Se identificate un collegamento espresso in modo errato, lanciate un'eccezione. Otterrete una valutazione migliore se il vostro programma è in grado di seguire i collegamenti che ha identificato, trovando collegamenti anche in quelle pagine (i motori di ricerca, come Google, usano un meccanismo simile a questo per scoprire i siti web).

Capitolo 12

Esercizi di programmazione

* **Esercizio P12.1.** Scrivete un metodo ricorsivo `void reverse()` che inverte il contenuto di una frase. Ad esempio, questo frammento di codice

```
Sentence greeting = new Sentence("Hello!");
greeting.reverse();
System.out.println(greeting.getText());
```

visualizza la stringa `"!olleH"`. Progettate una soluzione ricorsiva eliminando il primo carattere, invertendo la frase costituita dal testo rimanente e combinando le due parti.

** **Esercizio P12.2.** Risolvete di nuovo l'esercizio precedente, usando però un metodo ausiliario ricorsivo che inverte una sottostringa di un messaggio di testo.

* **Esercizio P12.3.** Realizzate iterativamente il metodo `reverse` dell'Esercizio P12.1.

** **Esercizio P12.4.** Usate la ricorsione per realizzare un metodo `boolean find(String t)` che verifica se una stringa è contenuta in una frase:

```
Sentence s = new Sentence("Mississippi!");
boolean b = s.find("sip"); // restituisce true
```

Suggerimento: se il testo inizia con la stringa che state cercando, avete finito; altrimenti, considerate la frase che si ottiene eliminando il primo carattere.

** **Esercizio P12.5.** Usate la ricorsione per realizzare un metodo `int indexOf(String t)` che restituisce la posizione iniziale della prima sottostringa del testo che sia uguale alla stringa `t`. Restituite –1 se `t` non è una sottostringa di `s`. Ad esempio:

```
Sentence s = new Sentence("Mississippi!");
int n = s.indexOf("sip"); // restituisce 6
```

Suggerimento: questo è un po' più difficile del problema precedente, perché dovete tenere traccia di quanto sia lontana dall'inizio della frase la corrispondenza che state cercando; inserite tale valore come parametro di un metodo ausiliario.

* **Esercizio P12.6.** Usando la ricorsione, trovate l'elemento maggiore in un array.

```
public class DataSet
{
   public DataSet(int[] values, int first, int last) {...}
   public int getMaximum() {...}
   ...
}
```

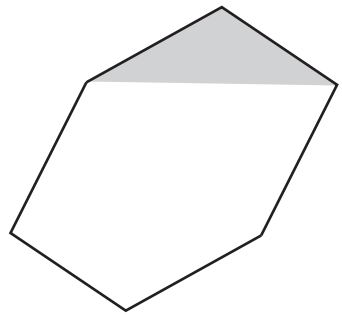
Suggerimento: trovate l'elemento maggiore nel sottoinsieme che contiene tutti gli elementi tranne l'ultimo; quindi, confrontate tale massimo con il valore dell'ultimo elemento.

★ **Esercizio P12.7.** Usando la ricorsione, calcolate la somma di tutti i valori presenti in un array.

```
public class DataSet
{
   public DataSet(int[] values, int first, int last) {...}
   public int getSum() {...}
   ...
}
```

★★ **Esercizio P12.8.** Usando la ricorsione, calcolate l'area di un poligono. Identificatene una porzione triangolare e sfruttate il fatto che un triangolo con vertici (x_1, y_1), (x_2, y_2) e (x_3, y_3) ha area:

$$\frac{\left|x_1y_2 + x_2y_3 + x_2y_1 - y_1x_2 - y_2x_3 - y_3x_1\right|}{2}$$



★★★ **Esercizio P12.9.** Realizzate una classe `SubstringGenerator` che generi tutte le sottostringhe di una stringa. Ad esempio, le sottostringhe della stringa `"rum"` sono queste sette stringhe:

```
"r", "ru", "rum", "u", "um", "m", ""
```

Suggerimento: dapprima trovate tutte le sottostringhe che iniziano con il primo carattere, che sono *n* se la stringa ha lunghezza *n*; poi, trovate le sottostringhe della stringa che si ottiene eliminando il primo carattere.

★★★ **Esercizio P12.10.** Realizzate una classe `SubsetGenerator` che generi tutti i sottoinsiemi di caratteri di una stringa. Ad esempio, i sottoinsiemi di caratteri della stringa `"rum"` sono queste otto stringhe:

```
"rum", "ru", "rm", "r", "um", "u", "m", ""
```

Notate che non è necessario che i sottoinsiemi siano sottostringhe: ad esempio, `"rm"` non è una sottostringa di `"rum"`.

★★★ **Esercizio P12.11.** In questo esercizio modificherete la classe `PermutationGenerator` vista nel Paragrafo 12.4 (che aveva un metodo per il calcolo di tutte le permutazioni di una stringa), trasformandola nella classe `PermutationIterator` (che calcola una permutazione per volta).

```
public class PermutationIterator
{
```

```
   public PermutationIterator(String s) {...}
   public String nextPermutation() {...}
   public boolean hasMorePermutations() {...}
}
```

Ecco come si usa tale classe per visualizzare tutte le permutazioni della stringa `"eat"`:

```
PermutationIterator iter = new PermutationIterator("eat");
while (iter.hasMorePermutations())
{
   System.out.println(iter.nextPermutation());
}
```

Dobbiamo, quindi, identificare un modo per calcolare ricorsivamente le permutazioni, restituendole però una per volta. Consideriamo la stringa `"eat"`. Come già visto, genereremo tutte le permutazioni che iniziano con la lettera `'e'`, poi quelle che iniziano con la lettera `'a'` e, infine, quelle che iniziano con la lettera `'t'`. Come generiamo le permutazioni che iniziano con `'e'`? Costruiamo un altro esemplare di `PermutationIterator` (che chiamiamo `tailIterator`) che itera all'interno dell'insieme delle permutazioni della sottostringa `"at"`. Nel metodo `nextPermutation`, chiediamo semplicemente a `tailIterator` quale sia la *sua* successiva permutazione, alla quale aggiungiamo `'e'` come prima lettera. C'è, però, un caso speciale: quando `tailIterator` esaurisce le proprie permutazioni, significa che sono già state elencate tutte le permutazioni che iniziano con la lettera che stiamo esaminando. Di conseguenza:

- Facciamo avanzare la posizione che stiamo esaminando.
- Calcoliamo la nuova stringa contenente tutte le lettere tranne quella in esame.
- Creiamo un nuovo generatore iterativo di permutazioni per tale nuova stringa.

Quando la posizione da esaminare giunge al termine della stringa, il lavoro è terminato.

*** **Esercizio P12.12.** La classe seguente genera tutte le permutazioni dei numeri 0, 1, 2, ..., $n - 1$, senza usare la ricorsione.

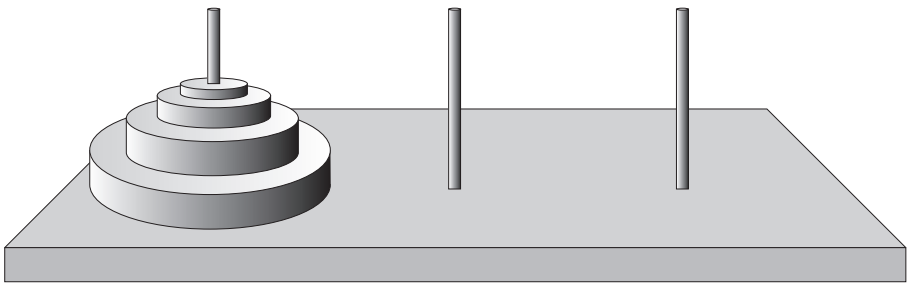
```
public class NumberPermutationIterator
{
   private int[] a;

   public NumberPermutationIterator(int n)
   {
      a = new int[n];
      for (int i = 0; i < n; i++) a[i] = i;
   }

   public int[] nextPermutation()
   {
      if (a.length <= 1) { return a; }

      for (int i = a.length - 1; i > 0; i--)
      {
         if (a[i - 1] < a[i])
         {
            int j = a.length - 1;
            while (a[i - 1] > a[j]) j--;
            swap(i - 1, j);
            reverse(i, a.length - 1);
            return a;
         }
```

```
      }
      return a;
   }

   public boolean hasMorePermutations()
   {
      if (a.length <= 1) { return false; }
      for (int i = a.length - 1; i > 0; i--)
      {
         if (a[i - 1] < a[i]) { return true; }
      }
      return false;
   }

   public void swap(int i, int j)
   {
      int temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }

   public void reverse(int i, int j)
   {
      while (i < j) { swap(i, j); i++; j--; }
   }
}
```

L'algoritmo sfrutta una proprietà dell'insieme da permutare, che contiene numeri distinti: non potete, quindi, usare lo stesso algoritmo per calcolare le permutazioni dei caratteri in una stringa. Potete, però, usare questa classe per ottenere tutte le permutazioni delle posizioni dei caratteri, per poi comporre una stringa il cui `i`-esimo carattere sia `word.charAt(a[i])`. Usate questo approccio per realizzare nuovamente, senza ricorsione, la classe `PermutationIterator` dell'esercizio precedente

★★ **Esercizio P12.13.** *Torri di Hanoi*. Si tratta di un famoso rompicapo. Bisogna spostare una pila di dischi di grandezza decrescente dal piolo più a sinistra a quello più a destra. Il piolo centrale si può usare per riporre i dischi temporaneamente (osservate la Figura 6). Si può muovere un disco alla volta, da un piolo a un altro, e si possono sovrapporre solamente dischi di misura inferiore su dischi più grandi, ma non viceversa.

Figura 6
Torri di Hanoi



Scrivete un programma che stampi le mosse necessarie per risolvere il rompicapo per un numero n di dischi (all'inizio del programma chiedete all'utente il valore di n). Stampate le mosse da eseguire in questa forma:

```
Move disk from peg 1 to peg 3
```

Suggerimento: realizzate una classe `DiskMover` (trasportatore di dischi), il cui costruttore riceve:

- il piolo sorgente, `source`, da cui prelevare i dischi (1, 2 o 3)
- il piolo destinazione, `target`, in cui inserire i dischi (1, 2 o 3)
- il numero di dischi da spostare, `disks`

Un trasportatore di dischi che sposta un solo disco da un piolo a un altro ha un metodo `nextMove` che restituisce semplicemente la stringa:

```
Move disk from peg source to peg target
```

Un trasportatore di dischi che deve spostare più dischi deve faticare di più e ha bisogno di un altro oggetto di tipo `DiskMover` che lo aiuti. Nel costruttore, costruite l'altro oggetto in questo modo: `DiskMover(source, other, disks - 1)`, dove `other` è il piolo diverso da `source` e da `target`.

Il metodo `nextMove` chiede a tale trasportatore di dischi "ausiliario" quale sia la sua successiva mossa, finché non ha terminato. L'effetto è quello di spostare `disks - 1` dischi sul piolo `other`. Quindi, il metodo `nextMove` scrive il comando per spostare un disco dal piolo `source` al piolo `target`. Infine, costruisce un altro trasportatore di dischi, `DiskMover(other, target, disks - 1)`, che genera le mosse necessarie a spostare i dischi dal piolo `other` al piolo `target`.

Suggerimento: è utile tenere traccia dello stato del trasportatore di dischi:

- `BEFORE_LARGEST`: il trasportatore ausiliario sposta la pila più piccola sul piolo `other`
- `LARGEST`: sposta il disco più grande dall'origine alla destinazione
- `AFTER_LARGEST`: il trasportatore ausiliario sposta la pila più piccola dal piolo `other` alla destinazione
- `DONE`: tutte le mosse sono state eseguite

Collaudate il vostro programma in questo modo:

```
DiskMover mover = new DiskMover(1, 3, n);
while (mover.hasMoreMoves())
{
   System.out.println(mover.nextMove());
}
```

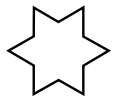
*** **Esercizio P12.14.** *Uscire da un labirinto.* Vi trovate all'interno di un labirinto, nella posizione indicata da un punto (`.`); le mura del labirinto sono rappresentate da asterischi (`*`).

```
* *******
*     * *
* ***** *
* * *   *
* * *** *
*   *   *
*** * * *
*     * *
******* *
```

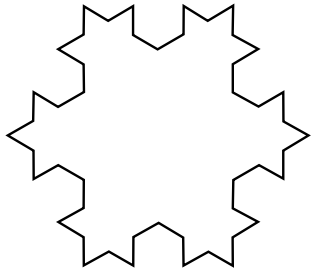
Per uscire dal labirinto, usate la seguente strategia ricorsiva. Se vi trovate accanto a un'uscita, restituite `true`. Controllate ricorsivamente se potete uscire da una delle caselle vuote adiacenti, senza tornare nella posizione attuale. Tale metodo verifica, semplicemente, se esiste un percorso per uscire dal labirinto: riceverete una valutazione migliore se siete in grado di visualizzare il percorso che vi conduce all'uscita.

***G **Esercizio P12.15.** *Il fiocco di neve di Koch.* Nel seguito definiamo ricorsivamente una forma simile a un fiocco di neve. Iniziate con un triangolo equilatero:

Poi, aumentatene la dimensione di un fattore 3 e sostituite ciascun segmento con quattro segmenti, in questo modo:



Ripetete il procedimento.



Scrivete un programma che disegni la quinta iterazione di questa curva. Riceverete una valutazione migliore se aggiungete un pulsante che, quando premuto, genera l'iterazione successiva.

** **Esercizio P12.16.** Il calcolo ricorsivo dei numeri di Fibonacci può essere accelerato in modo significativo tenendo traccia dei valori che sono già stati calcolati: realizzate una nuova versione del metodo `fib` usando questa strategia. Ogni volta che restituite un nuovo valore, memorizzatelo anche in un array ausiliario; prima di iniziare il calcolo di un valore, consultate l'array per vedere se tale calcolo sia già stato eseguito. Confrontate con le realizzazioni originali, ricorsiva e iterativa, il tempo di esecuzione della vostra versione, così migliorata.

Progetti di programmazione

Progetto P12.1. Migliorate l'analizzatore sintattico di espressioni del Paragrafo 12.5 in modo che possa gestire espressioni più complesse, con esponenti e funzioni matematiche, come `sqrt` o `sin`.

Progetto P12.2. Realizzate una versione grafica del programma per le Torri di Hanoi visto nell'Esercizio P12.13. Ogni volta che l'utente preme un pulsante con l'etichetta "Next", disegnate la mossa successiva.

Capitolo 13

Esercizi di programmazione

* **Esercizio P13.1.** Modificate l'algoritmo di ordinamento per selezione in modo da ordinare un array di numeri interi in ordine decrescente.

* **Esercizio P13.2.** Modificate l'algoritmo di ordinamento per selezione in modo da ordinare un array di monete in base al loro valore.

** **Esercizio P13.3.** Scrivete un programma che generi automaticamente la tabella dei tempi delle esecuzioni di prova dell'ordinamento per selezione. Il programma deve chiedere i valori minimo e massimo di `n` e il numero di misurazioni da effettuare, per poi attivare tutte le esecuzioni di prova.

* **Esercizio P13.4.** Modificate l'algoritmo di ordinamento per fusione in modo da ordinare un array di stringhe in ordine lessicografico.

*** **Esercizio P13.5.** Scrivete un programma per consultare l'elenco del telefono. Leggete un insieme di dati contenente 1000 nomi e i relativi numeri di telefono, memorizzati in un file che contiene i dati in ordine casuale. Gestite la ricerca sia in base al nome sia in base al numero di telefono, utilizzando una ricerca binaria per entrambe le modalità di consultazione.

** **Esercizio P13.6.** Scrivete un programma che misuri le prestazioni dell'algoritmo di ordinamento per inserimento descritto in Argomenti avanzati 13.1.

*** **Esercizio P13.7.** Scrivete un programma che ordini un esemplare di `ArrayList<Coin>` in ordine decrescente, in modo che la moneta di valore maggiore si trovi all'inizio del vettore. Usate una classe che implementa `Comparator`.

** **Esercizio P13.8.** Considerate l'algoritmo di ricerca binaria del Paragrafo 13.7: se non trova l'elemento cercato, il metodo `search` restituisce –1. Modificatelo in modo che, se `a` non viene trovato, venga invece restituito il valore $-k-1$, dove k è la posizione prima della quale l'elemento dovrebbe essere inserito (che è il comportamento di `Arrays.binarySearch`).

** **Esercizio P13.9.** Realizzate senza ricorsione il metodo `sort` dell'algoritmo di ordinamento per fusione, nell'ipotesi che la lunghezza dell'array sia una potenza di 2. Prima fondete le regioni adiacenti di dimensione 1, poi le regioni adiacenti di dimensione 2, quindi le regioni adiacenti di dimensione 4 e così via.

*** **Esercizio P13.10.** Realizzate senza ricorsione il metodo `sort` dell'algoritmo di ordinamento per fusione, nell'ipotesi che la lunghezza dell'array sia un numero arbitrario. Procedete fondendo regioni adiacenti la cui dimensione sia una potenza di 2 e fate attenzione all'ultima regione, che può avere dimensione inferiore.

*** **Esercizio P13.11.** Usate l'ordinamento per inserimento e la ricerca binaria dell'Esercizio P13.8 per ordinare un array secondo quanto descritto nell'Esercizio R13.14. Realizzate tale algoritmo e misuratene le prestazioni.

* **Esercizio P13.12.** Scrivete una classe `Person` che realizzi l'interfaccia `Comparable`, confrontando persone in base al loro nome. Chiedete all'utente di inserire dieci nomi e generate dieci oggetti di tipo `Person`. Usando il metodo `compareTo`, determinate la prima e l'ultima persona dell'insieme, e stampatele.

** **Esercizio P13.13.** Ordinate un vettore di stringhe in ordine crescente di *lunghezza*. *Suggerimento*: progettate un apposito `Comparator`.

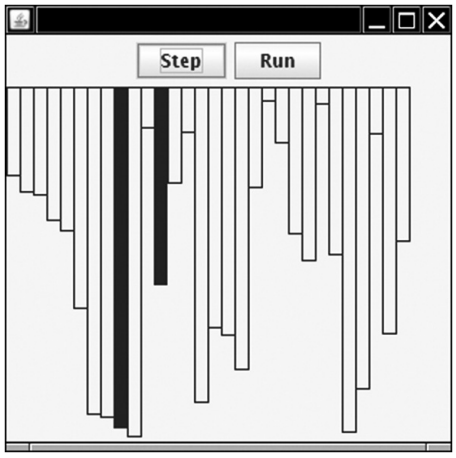
*** **Esercizio P13.14.** Ordinate un vettore di stringhe in ordine crescente di lunghezza, in modo che stringhe della stessa lunghezza vengano disposte in ordine lessicografico. *Suggerimento*: progettate un apposito `Comparator`.

Progetti di programmazione

Progetto P13.1. Scrivete un programma per gestire un'agenda di appuntamenti. Create una classe `Appointment` che memorizzi una descrizione dell'appuntamento, il giorno dell'appuntamento, l'ora di inizio e l'ora di fine. Il programma deve conservare gli appuntamenti in un vettore ordinato. Gli utenti possono aggiungere appuntamenti e visualizzare tutti gli appuntamenti di un determinato giorno. Quando viene aggiunto un nuovo appuntamento, utilizzate una ricerca binaria per stabilire in quale posizione del vettore va inserito: non aggiungetelo se è in conflitto con altri appuntamenti.

Progetto P13.2. Realizzate una *animazione grafica* degli algoritmi di ordinamento e di ricerca. Costruite un array e inseritevi numeri casuali compresi tra 1 e 100, poi disegnate ciascun elemento dell'array sotto forma di barretta verticale, come nella Figura 7. Ogni volta che l'algoritmo modifica l'array, mostrate una finestra di dialogo e attendete che l'utente prema il pulsante "Step", poi invocate il metodo `repaint`. Premendo il pulsante "Run" si fa procedere l'animazione, fino al termine oppure fino a quando l'utente preme il pulsante "Step".

Figura 7
Animazione grafica



Realizzate le animazioni dell'ordinamento per selezione, dell'ordinamento per fusione e della ricerca binaria. Nell'animazione della ricerca binaria evidenziate l'elemento in fase di ispezione e i valori di `from` e `to`.

Capitolo 14

Esercizi di programmazione

** **Esercizio P14.1.** Utilizzando soltanto l'interfaccia pubblica della classe lista concatenata, scrivete un metodo

```
public static void downsize(LinkedList<String> staff)
```

che elimini un impiegato sì e uno no da una lista concatenata.

** **Esercizio P14.2.** Utilizzando soltanto l'interfaccia pubblica della classe lista concatenata, scrivete un metodo

```
public static void reverse(LinkedList<String> staff)
```

che inverta i dati presenti in una lista concatenata.

*** **Esercizio P14.3.** Aggiungete alla nostra realizzazione della classe `LinkedList` un metodo `reverse` che inverta i collegamenti presenti nella lista. Realizzate questo metodo modificando direttamente i collegamenti, non usando un iteratore.

* **Esercizio P14.4.** Aggiungete alla nostra implementazione della classe `LinkedList` un metodo `size` che calcoli il numero di elementi presenti nella lista, seguendo i collegamenti e conteggiando gli elementi fino a quando si arriva alla fine della lista.

* **Esercizio P14.5.** Aggiungete alla nostra implementazione della classe `LinkedList` un campo `currentSize` e modificate i metodi `add` e `remove` sia della lista concatenata sia dell'iteratore per aggiornare il campo `currentSize` in modo che contenga sempre la dimensione corretta. Modificate il metodo `size` dell'esercizio precedente in modo che restituisca, semplicemente, il valore di questo campo di esemplare.

** **Esercizio P14.6.** La classe della libreria standard che realizza una lista concatenata ha un metodo `add` che consente inserimenti efficienti all'estremità terminale della lista. Realizzate tale metodo anche nella classe `LinkedList` vista nel Paragrafo 14.2, aggiungendovi una variabile di esemplare che si riferisca all'ultimo nodo della lista. Accertatevi che gli altri metodi modificatori aggiornino tale variabile.

*** **Esercizio P14.7.** Ripetete l'esercizio precedente usando una diversa strategia. Eliminate dalla classe `LinkedList` il riferimento al primo nodo e fate in modo che il riferimento `next` dell'ultimo nodo punti al primo nodo, in modo che i nodi concatenati formino una catena chiusa. Tale realizzazione viene chiamata *lista concatenata circolare*.

*** **Esercizio P14.8.** Realizzate nuovamente la classe `LinkedList` vista nel Paragrafo 14.2 facendo in modo che le classi `Node` e `LinkedListIterator` non siano classi interne.

*** **Esercizio P14.9.** Aggiungete alla classe `Node` vista nel Paragrafo 14.2 una variabile di esemplare `previous`, dotando l'iteratore dei metodi `previous` e `hasPrevious`.

** **Esercizio P14.10.** Il linguaggio di programmazione LISP, progettato nel 1960, realizza le liste concatenate in modo molto elegante: in questo e nei prossimi tre esercizi vedrete una soluzione analoga in Java. L'osservazione chiave è questa: la *coda* di una lista, cioè la lista privata del suo elemento iniziale, è a sua volta una lista, e la coda di tale lista è ancora una lista, e così via, fino a quando si ottiene una lista vuota. Ecco un'interfaccia Java che descrive una lista di questo tipo:

```
public interface LispList
{
   boolean isEmpty();
   Object head();
   LispList tail();
   ...
}
```

Esistono due tipi di queste liste, quelle vuote (*empty*) e quelle non vuote:

```
public class EmptyList implements LispList { ... }
public class NonEmptyList implements LispList { ... }
```

Si tratta di classi banali. La classe `EmptyList` non ha variabili di esemplare, i suoi metodi `head` e `tail` lanciano semplicemente `UnsupportedOperationException` e il suo metodo `isEmpty` restituisce `true`. La classe `NonEmptyList` ha, invece, due variabili di esemplare, per memorizzare la testa e la coda.

Ecco un modo per costruire una lista con tre elementi:

```
LispList list = new NonEmptyList("A",
                   new NonEmptyList("B",
                   new NonEmptyList("C",
                   new EmptyList())));
```

L'operazione è un po' noiosa, quindi è bene progettare un metodo di utilità, `cons`, che invochi il costruttore in modo opportuno, oltre a una variabile statica, `NIL`, che memorizza un esemplare di lista vuota. In questo modo la costruzione della lista che abbiamo usato come esempio diventa:

```
LispList list = LispList.NIL.cons("C").cons("B").cons("A");
```

Osservate che occorre costruire la lista partendo da una coda vuota.

Per poter apprezzare l'eleganza di questo approccio, considerate la realizzazione di un metodo, `toString`, che generi una stringa contenente tutti gli elementi presenti in una lista. Il metodo deve essere realizzato in entrambe le sottoclassi:

```
public class EmptyList ...
{
   ...
   public String toString() { return ""; }
}

public class NonEmptyList ...
{
   ...
   public String toString() { return head().toString() + " " + tail().toString(); }
}
```

Osservate che non occorre alcun enunciato `if`. Una lista può essere vuota o non vuota e, grazie al polimorfismo, viene invocato il metodo `toString` corretto.

In questo esercizio, completate l'interfaccia `LispList` e le classi `EmptyList` e `NonEmptyList`. Scrivete, poi, un programma di collaudo che costruisca una lista e la visualizzi.

* **Esercizio P14.11.** Aggiungete all'interfaccia `LispList` dell'esercizio precedente il metodo `length`, che restituisca la lunghezza della lista, realizzandolo poi nelle classi `EmptyList` e `NonEmptyList`.

** **Esercizio P14.12.** Aggiungete all'interfaccia `LispList` dell'esercizio P14.10 il metodo

```
LispList merge(LispList other)
```

che fonda due liste, realizzandolo poi nelle classi `EmptyList` e `NonEmptyList`. Per effettuare la fusione di due liste, prendete un elemento alternativamente da una delle due liste, aggiungendo poi gli elementi rimasti nella lista più lunga. Ad esempio, fondendo le liste che contengono `1 2 3 4` e `5 6`, si ottiene la lista che contiene `1 5 2 6 3 4`.

* **Esercizio P14.13.** Aggiungete all'interfaccia `LispList` dell'esercizio P14.10 il metodo

```
boolean contains(Object obj)
```

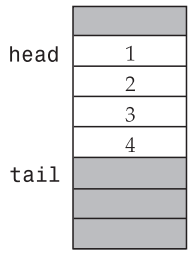
che restituisca `true` se la lista contiene un elemento uguale a `obj`.

*** **Esercizio P14.14.** La libreria standard di Java contiene una classe `Stack` che realizza una pila, ma in questo esercizio vi viene chiesto di fornire una vostra realizzazione di tale tipo di dati astratto. Non realizzate una classe parametrica, usate un array di tipo `Object[]` per memorizzare gli elementi presenti nella pila e, quando l'array si riempie, predisponetene uno di dimensioni doppie, copiando i valori dal vecchio al nuovo array.

★ **Esercizio P14.15.** Realizzate una classe `Stack` usando una lista concatenata per memorizzarne gli elementi. Non realizzate una classe parametrica.

★★ **Esercizio P14.16.** Realizzate una coda come *array circolare*, nel modo seguente. Usate due variabili, `head` e `tail`, che contengono l'indice del prossimo elemento da eliminare e del prossimo elemento da aggiungere: dopo che un elemento viene eliminato o aggiunto, il relativo indice viene incrementato (si veda la Figura 8).

Figura 8
Aggiungere e rimuovere elementi in una coda

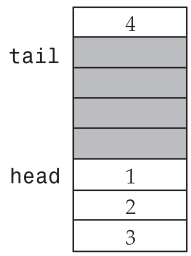


Dopo alcuni passi di funzionamento, l'elemento `tail` raggiungerà la fine dell'array, dopodiché "si riavvolge" e riparte da 0 (si veda la Figura 9): per tale ragione, l'array viene chiamato "circolare".

```
public class CircularArrayQueue
{
   private int head;
   private int tail;
   private int theSize;
   private Object[] elements;

   public CircularArrayQueue(int capacity) {...}
   public void add(Object element) {...}
   public Object remove() {...}
   public int size() {...}
}
```

Questa realizzazione fornisce una coda di dimensioni *limitate*, che può anche riempirsi. Nel prossimo esercizio vedrete come rimuovere questo vincolo.

Figura 9
Coda che si "riavvolge" attorno alla fine dell'array



★★★ **Esercizio P14.17.** La coda dell'esercizio precedente può riempirsi, se vengono aggiunti più elementi di quanti possono essere contenuti nell'array. Migliorate la realizzazione nel modo seguente: quando l'array si riempie, costruite un array più grande, copiate i valori nel nuovo array e assegnatelo alla variabie di esemplare `elements`. *Suggerimento*: Non potete semplicemente copiare gli elementi nel nuovo array nella stessa posizione che occupano nel vecchio; spostate, invece, l'elemento iniziale della coda nella posizione 0, e così via.

** **Esercizio P14.18.** Modificate l'algoritmo di ordinamento per inserimento visto in Argomenti avanzati 13.1 in modo che ordini una lista concatenata.

Esercizio P14.19. *Esercizio relativo a una parte del testo che non è stata tradotta.*

* **Esercizio P14.20.** In un programma per disegnare, l'operazione di "riempimento per inondazione" (*flood fill*) assegna un determinato colore a tutti i pixel vuoti di un disegno, fermandosi quando raggiunge pixel occupati. In questo esercizio svilupperete una semplice variante di questo algoritmo, riempiendo per inondazione un array di numeri interi di dimensione 10 ×10, inizialmente riempito di zeri. Chiedete all'utente la posizione iniziale: riga e colonna. Inserite in una pila la coppia `<riga, colonna>` (vi servirà una semplice classe `Pair`).

Quindi, ripetete queste operazioni finché la pila non è vuota.

- La coppia `<riga, colonna>` presente in cima alla pila viene estratta.
- Se la posizione corrispondente è ancora vuota, viene riempita, usando via via i numeri 1, 2, 3, ecc., in modo da visualizzare l'ordine di riempimento delle celle.
- Le coordinate delle celle adiacenti in direzione nord, est, sud e ovest, che non siano state riempite, vengono inserite nella pila.

Al termine, visualizzate l'intero array.

* **Esercizio P14.21.** Ripetete l'esercizio precedente usando una coda.

** **Esercizio P14.22.** Usate una pila per generare tutte le permutazioni di una stringa. Immaginate di voler trovare tutte le permutazioni della stringa `meat`: impilate la stringa `+meat`. Quindi, ripetete queste operazioni finché la pila non è vuota.

- La stringa presente in cima alla pila viene estratta.
- Se la stringa termina con il carattere `+` (come `tame+`), eliminatelo e aggiungete la stringa alla lista delle permutazioni.
- Altrimenti, rimuovete una delle lettere che si trovano a destra del carattere `+`, inseritela subito prima del carattere `+` e impilate la stringa risultante. Ripetete la procedura per tutte le lettere. Ad esempio, dopo aver estratto dalla pila la stringa `e+mta`, impilate `em+ta`, `et+ma` e `ea+mt`.

** **Esercizio P14.23.** Ripetete l'esercizio precedente usando una coda.

G **Esercizio P14.24. Scrivete un programma che rappresenti graficamente una lista concatenata. Disegnate ciascun elemento della lista con un riquadro e indicate i collegamenti con segmenti di retta. Disegnate un iteratore come nella Figura 3. Inserite pulsanti per spostare l'iteratore e per aggiungere e rimuovere elementi dalla lista.

Progetti di programmazione

Progetto 14.1. Progettate una classe `Polynomial` che descriva un polinomio del tipo

$$p(x) = 5x^{10} + 9x^7 - x - 10$$

Memorizzate un polinomio come lista concatenata di termini. Un termine contiene il coefficiente e l'esponente di x. Per esempio, $p(x)$ verrebbe memorizzato come

$$(5,10), (9,7), (-1,1), (-10,0)$$

Fornite metodi per sommare, moltiplicare e visualizzare polinomi, oltre che per calcolare la derivata di un polinomio.

Progetto 14.2. Fate in modo che la realizzazione di lista vista in questo capitolo abbia le stesse funzionalità di quella realizzata nella libreria di Java (pur senza usare un tipo parametrico).

- Fornite iteratori bidirezionali.
- Fate in modo che `Node` sia una classe interna statica.
- Realizzate le interfacce standard `List` e `ListIterator`, scrivendo i metodi mancanti. *Suggerimento*: può essere più semplice estendere `AbstractList` piuttosto che partire da zero nella realizzazione di tutti i metodi di `List`.

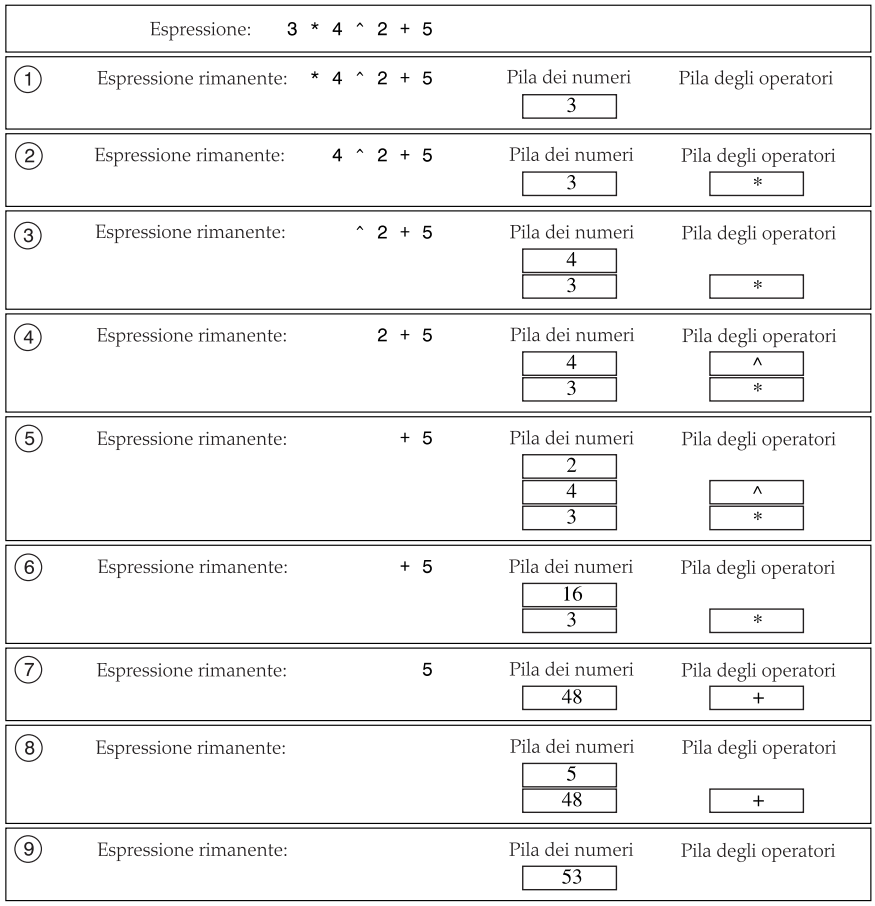
Progetto 14.3. Realizzate il seguente algoritmo per la valutazione di espressioni aritmetiche. Ogni operatore ha una *precedenza*: gli operatori `+` e `-` hanno la precedenza più bassa, `*` e `/` hanno una precedenza più elevata (e tra loro uguale) e `^` (che, in questo esercizio, indica l'elevamento a potenza) ha la precedenza più alta. Ad esempio:

```
3 * 4 ^ 2 + 5
```

deve avere il significato seguente:

```
(3 * (4 ^ 2)) + 5
```

fornendo, quindi, il valore 53 come risultato della valutazione.



Usate due pile, una che contiene numeri e un'altra che contiene operatori. Quando trovate un numero, inseritelo nella pila dei numeri. Quando trovate un operatore, se ha precedenza più elevata dell'operatore che si trova in cima alla pila degli operatori, inseritelo nella pila degli operatori, altrimenti estraete un operatore dalla relativa pila, estraete due numeri dalla pila dei numeri e inserite il risultato dell'operazione nella pila dei numeri. Ripetete la procedura finché l'operatore che si trova in cima alla pila non ha precedenza inferiore a quello che stavate esaminando. Al termine dell'analisi dell'espressione, svuotate la pila nello stesso modo. La figura mostra come viene valutata l'espressione `3 * 4 ^ 2 + 5`.

Dovreste poi migliorare questo algoritmo in modo che sia in grado di gestire le parentesi. Accertatevi anche che le sottrazioni e le divisioni vengano svolte nell'ordine corretto: ad esempio, la valutazione di `12 - 5 - 3` deve avere come risultato 4, non 10.

Capitolo 15

Esercizi di programmazione

★ **Esercizio P15.1.** Scrivete un programma che legga un testo dal flusso `System.in` e lo suddivida in singole parole. Inserite le parole in un insieme realizzato mediante un albero. Dopo aver letto tutti i dati in ingresso, visualizzate tutte le parole, seguite dalla dimensione dell'insieme risultante. Questo programma determina, quindi, quante parole diverse sono presenti in un testo.

★ **Esercizio P15.2.** Inserite in un insieme i 13 colori standard predefiniti nella classe `Color` (cioè `Color.PINK`, `Color.GREEN` e così via). Chiedete all'utente di inserire un colore specificandone i valori delle componenti di rosso, verde e blu con numeri interi compresi tra 0 e 255. Comunicate poi all'utente se il colore che ne risulta è nell'insieme.

★★ **Esercizio P15.3.** Realizzate il *crivello di Eratostene*, un metodo per calcolare i numeri primi noto agli antichi greci. Scegliete un numero n: questo metodo calcolerà tutti i numeri primi fino a n. Come prima cosa inserite in un insieme tutti i numeri da 2 a n. Poi, cancellate tutti i multipli di 2 (eccetto 2); vale a dire 4, 6, 8, 10, 12, ... Dopodiché, cancellate tutti i multipli di 3; cioè, 6, 9, 12, 15, ... Arrivate fino a $\sqrt{n}$, quindi visualizzate l'insieme.

★ **Esercizio P15.4.** Leggendo tutte le parole di un file di testo di grandi dimensioni (come il romanzo "War and Peace", *Guerra e pace*, disponibile in Internet), inseritele in due insiemi, uno realizzato mediante tabella di hash e uno realizzato mediante albero. Misurate i due tempi di esecuzione: quale struttura agisce più velocemente?

★★★ **Esercizio P15.5.** Scrivete un programma che legga un file di codice sorgente Java e generi un elenco di tutti gli identificatori presenti, visualizzando, accanto a ciascuno di essi, i numeri delle righe in cui compare. *Suggerimento*: Invocate il metodo `in.useDelimiter("[^A-Za-z0-9_]+")`, in modo che ogni invocazione di `next` restituisca una stringa composta soltanto da lettere, cifre e caratteri di sottolineatura.

★★ **Esercizio P15.6.** Leggendo tutte le parole di un file di testo di grandi dimensioni (come `/usr/share/dict/words`, disponibile in un sistema Linux), cercatene due che abbiamo lo stesso codice di hash. Usate una mappa di tipo `Map<Integer, HashSet<String>>`: ogni volta che leggete una parola, calcolate il suo codice di hash, h, e inserite la parola nell'insieme associato alla chiave h; alla fine, scandite tutte le chiavi e visualizzate gli insiemi aventi dimensione maggiore di uno.

★★ **Esercizio P15.7.** Scrivete un programma che usi una mappa in cui sia le chiavi sia i valori sono stringhe: rispettivamente, i nomi degli studenti e i loro voti in un esame. Chiedete all'utente del

programma se vuole inserire o rimuovere studenti, modificarne il voto o stampare tutti i voti. La visualizzazione dovrebbe essere ordinata per nome e avere un aspetto simile a questo:

```
Carl: B+
Joe: C
Sarah: A
```

*** **Esercizio P15.8.** Risolvete nuovamente l'esercizio precedente in modo che le chiavi della mappa siano esemplari della classe `Student`; uno studente dovrebbe avere un nome, un cognome e un numero intero, ID, usato come identificativo univoco (in analogia con il numero di matricola usato nelle università italiane). La visualizzazione dovrebbe essere ordinata per cognome; se due studenti hanno lo stesso cognome, usate il nome per risolvere la parità; se anche i nomi sono uguali, usate il numero ID. *Suggerimento:* usate due mappe.

*** **Esercizio P15.9.** Aggiungete alla realizzazione di `HashSet` vista nel Paragrafo 15.3 un metodo `debug` che visualizzi i bucket non vuoti presenti nella tabella hash. Eseguite il programma di collaudo presentato alla fine del Paragrafo 15.3, invocando il metodo `debug` dopo l'esecuzione di tutti gli inserimenti e rimozioni e verificate che la Figura 5 rappresenti in modo esatto lo stato della tabella hash.

** **Esercizio P15.10.** Realizzate, nella classe `Student` descritta nell'Esercizio P15.8, metodi `hashCode` e `equals` che siano fra loro compatibili. Verificate la correttezza dell'implementazione della funzione di hash aggiungendo oggetti di tipo `Student` a un insieme realizzato con tabella hash.

* **Esercizio P15.11.** Realizzate, nella classe `BankAccount` del Capitolo 7, metodi `hashCode` e `equals` che siano fra loro compatibili. Verificate la correttezza dell'implementazione del metodo `hashCode` visualizzando codici di hash e aggiungendo oggetti di tipo `BankAccount` a un insieme realizzato con tabella hash.

* **Esercizio P15.12.** Un punto geometrico dotato di etichetta è caratterizzato da coordinate x e y, oltre che dall'etichetta, sotto forma di stringa. Progettate la classe `LabeledPoint` dotata del costruttore `LabeledPoint(int x, int y, String label)` e dei metodi `hashCode` e `equals`; due punti sono considerati uguali quando si trovano nella stessa posizione e hanno la stessa etichetta.

* **Esercizio P15.13.** Realizzate una diversa versione della classe `LabeledPoint` vista nell'esercizio precedente, memorizzando la posizione del punto in un oggetto di tipo `java.awt.Point`. I metodi `hashCode` e `equals` devono invocare gli omonimi metodi della classe `Point`.

* **Esercizio P15.14.** Modificate la classe `LabeledPoint` dell'Esercizio P15.12 in modo che realizzi l'interfaccia `Comparable`. Fate in modo che i punti vengano ordinati innanzitutto in base alle loro coordinate x; se due punti hanno la stessa coordinata x, ordinateli in base alla loro coordinata y; se due punti hanno le stesse coordinate, ordinateli in base alla loro etichetta. Scrivete un programma di collaudo che verifichi tutti i casi.

** **Esercizio P15.15.** Progettate una struttura di dati, `IntSet`, che possa contenere un insieme di numeri interi. Nascondete l'implementazione privata, costituita da un albero di ricerca binario che contiene oggetti di tipo `Integer`. Fornite i seguenti metodi:

- Un costruttore per creare un insieme vuoto
- `void add(int x)` per aggiungere `x` se non è presente
- `void remove(int x)` per eliminare `x` se è presente
- `void print()` per visualizzare tutti gli elementi presenti nell'insieme
- `boolean find(int x)` per verificare se `x` è presente nell'insieme

*** **Esercizio P15.16.** Realizzate nuovamente la classe dell'esercizio precedente usando un esemplare di `TreeSet<Integer>`. Oltre ai metodi già visti, progettate un metodo `iterator` che restituisca un oggetto dotato dei *soli* metodi `hasNext` e `next`. Il metodo `next` deve restituire un `int`, non un oggetto, per cui non potete semplicemente restituire l'iteratore fornito dalla classe `TreeSet`.

* **Esercizio P15.17.** Realizzate nuovamente la classe dell'Esercizio P15.15 usando un esemplare di `TreeSet<Integer>`. Oltre ai metodi già visti nell'Esercizio P15.15, progettate i metodi seguenti, che calcolano l'unione e l'intersezione di due insiemi:

```
IntSet union(IntSet other)
IntSet intersection(IntSet other)
```

* **Esercizio P15.18.** Scrivete un metodo della classe `BinarySearchTree`

```
Comparable smallest()
```

che restituisca l'elemento minimo presente in un albero. Avrete bisogno di aggiungere un metodo anche alla classe `Node`.

*** **Esercizio P15.19.** Modificate il metodo `BinarySearchTree.print` in modo che visualizzi la forma grafica di un albero, facendolo estendere orizzontalmente, da sinistra a destra. Otterrete una valutazione migliore se, invece, visualizzate l'albero che si estende verso il basso, con il nodo radice in alto, centrato orizzontalmente.

* **Esercizio P15.20.** Realizzate metodi che usino la visita in ordine anticipato e in ordine posticipato per visualizzare gli elementi presenti in un albero di ricerca binario.

*** **Esercizio P15.21.** Nella classe `BinarySearchTree`, modificate il metodo `remove` in modo che un nodo avente due figli venga sostituito dal figlio contenente il dato di valore maggiore presente nel suo sottoalbero di sinistra.

** **Esercizio P15.22.** Definite un'interfaccia, `Visitor`, dotata dell'unico metodo:

```
void visit(Object obj)
```

Aggiungete, poi, alla classe `BinarySearchTree` i metodi:

```
void inOrder(Visitor v)
void preOrder(Visitor v)
void postOrder(Visitor v)
```

Tali metodi devono visitare i nodi dell'albero secondo l'ordine specifico di ciascun attraversamento, applicando il metodo `visit` ai dati presenti nei nodi visitati.

** **Esercizio P15.23.** Usate i metodi definiti nell'esercizio precedente per calcolare il valor medio degli elementi presenti in un albero di ricerca binario contenente oggetti di tipo `Integer`, fornendo un oggetto di una classe che realizzi l'interfaccia `Visitor`.

** **Esercizio P15.24.** Modificate la realizzazione della classe `MinHeap` in modo che gli indici corrispondenti alle posizioni dei nodi genitore e figlio vengano calcolati direttamente, senza invocare metodi ausiliari.

*** **Esercizio P15.25.** Modificate la realizzazione della classe `MinHeap` in modo che la cella di indice 0 dell'array non sia sprecata.

* **Esercizio P15.26.** Misurate il tempo necessario all'esecuzione degli algoritmi *heapsort* e *mergesort*: nella realtà, quale algoritmo ha le prestazioni migliori?

Progetti di programmazione

Progetto 15.1. Realizzate una classe, `BinaryTreeSet`, che usi un esemplare di `TreeSet` per memorizzare i propri elementi. Dovrete realizzare un iteratore che scandisca i nodi secondo l'ordine dei valori in essi contenuti. Tale iteratore è piuttosto complesso, perché a volte ha la necessità di tornare indietro: potete aggiungere in ciascun esemplare di `Node` un riferimento al nodo genitore, oppure potete fare in modo che un oggetto iteratore memorizzi una pila dei nodi visitati.

Progetto 15.2. Realizzate un valutatore di espressioni che usi un analizzatore sintattico (*parser*) per costruire un albero di espressione, come visto nel Paragrafo 15.6 (notate che l'albero che ne risulta è un albero binario ma non è un albero di ricerca binario). Eseguite, poi, una visita in ordine posticipato per valutare l'espressione, usando una pila per la memorizzazione dei risultati intermedi.

Progetto 15.3. Scrivete un programma che mostri un'animazione del funzionamento dell'algoritmo *heapsort*, visualizzando l'albero graficamente e fermandosi dopo ogni invocazione di `fixHeap`.

Capitolo 16

Esercizi di programmazione

* **Esercizio P16.1.** Modificate la classe generica `Pair` in modo che i due valori siano dello stesso tipo.

* **Esercizio P16.2.** Aggiungete alla classe `Pair` dell'esercizio precedente un metodo `swap` che scambi tra loro il primo e il secondo elemento della coppia.

** **Esercizio P16.3.** Realizzate un metodo statico generico `PairUtil.swap`, il cui parametro sia un oggetto di tipo `Pair`, usando la classe generica definita nel Paragrafo 16.2. Il metodo deve restituire una nuova coppia avente il primo e il secondo elemento scambiati rispetto alla coppia originaria.

** **Esercizio P16.4.** Scrivete un metodo statico generico `PairUtil.minmax` che identifichi gli elementi minimo e massimo presenti in un array di tipo `T` e restituisca una coppia contenente tali valori minimo e massimo. Esprimete il fatto che gli elementi dell'array devono realizzare l'interfaccia `Measurable` vista nel Capitolo 9.

** **Esercizio P16.5.** Risolvete nuovamente il problema dell'esercizio precedente, richiedendo però che gli elementi dell'array realizzino l'interfaccia `Comparable`.

*** **Esercizio P16.6.** Risolvete nuovamente il problema dell'Esercizio P16.4, richiedendo però che il tipo parametrico estenda il tipo generico `Comparable`.

** **Esercizio P16.7.** Realizzate una versione generica dell'algoritmo di ricerca binaria.

*** **Esercizio P16.8.** Realizzate una versione generica dell'algoritmo di ordinamento per fusione. Il programma deve essere compilato senza che vengano segnalati *warning*.

** **Esercizio P16.9.** Realizzate una versione generica della classe `LinkedList` vista nel Capitolo 14.

** **Esercizio P16.10.** Realizzate una versione generica della classe `BinarySearchTree` vista nel Capitolo 15.

** **Esercizio P16.11.** Rendete generica la classe `HashSet` vista nel Capitolo 15. Per memorizzare i bucket, usate un vettore invece di un array.

** **Esercizio P16.12.** Dotate di un appropriato metodo `hashCode` la classe `Pair` vista nel Paragrafo 16.2 e realizzate una classe `HashMap` che usi un esemplare di `HashSet<Pair<K, V>>`.

*** **Esercizio P16.13.** Realizzate una versione generica del generatore di permutazioni visto nel Paragrafo 12.2, in modo che generi tutte le permutazioni degli elementi presenti in un esemplare di `List<E>`.

** **Esercizio P16.14.** Scrivete un metodo statico generico, `print`, che visualizzi l'elenco degli elementi presenti in qualsiasi esemplare di una classe che implementi l'interfaccia `Iterable<E>`, inserendolo in un'appropriata classe di utilità. Gli elementi visualizzati devono essere separati da una virgola.

Progetti di programmazione

Progetto 16.1. Progettate e realizzate una versione generica della classe `DataSet` vista nel Capitolo 9, che possa essere utilizzata per analizzare dati di qualsiasi classe che realizzi l'interfaccia `Measurable`. Rendete generica anche la stessa interfaccia `Measurable`. Dotate la classe di un metodo `addAll` che consenta di aggiungere a un suo esemplare tutti i valori presenti in un altro esemplare di tipo compatibile. Definite anche l'interfaccia generica `Measurer<T>` che consenta l'analisi di dati che siano esemplari di classi che non realizzano l'interfaccia `Measurable`.

Progetto 16.2. Rendete generica la classe `MinHeap` vista nel Capitolo 15. Come nella classe `TreeSet` della libreria standard, consentite l'utilizzo di un esemplare di `Comparator` per confrontare elementi. Se non viene fornito alcun comparatore, ipotizzate che gli elementi siano esemplari di un tipo che realizza l'interfaccia `Comparable`.