Progetti di programmazione

Gli asterischi indicano il grado di difficoltà.

Capitolo 1

** **P1.1.** Volete decidere se andare al lavoro in auto o in treno. Conoscete la distanza di sola andata da casa al luogo di lavoro e il consumo di carburante della vostra automobile (in miglia percorse per gallone di carburante), oltre al costo del biglietto del treno per un viaggio di sola andata. Ipotizzate che il carburante costi $4 al gallone e che i costi di manutenzione dell'automobile siano pari a 5 centesimi di dollaro per miglio percorso. Scrivete un algoritmo che consenta di individuare la modalità di viaggio più economica.

** **P1.2.** Volete scoprire quale frazione dell'utilizzo della vostra auto è dedicata ad andare al lavoro e quale, invece, ai viaggi dovuti ad altri motivi. Conoscete la distanza di sola andata da casa al luogo di lavoro e, durante un determinato periodo di tempo, avete registrato il chilometraggio iniziale e finale riportato dal contachilometri dell'auto, oltre al numero di giorni lavorativi. Scrivete un algoritmo che risolva il problema.

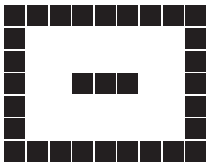
*** **P1.3.** Il valore di π può essere calcolato usando la formula seguente:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

Scrivete un algoritmo che calcoli π. Dato che la formula è una sommatoria di infiniti addendi e un algoritmo deve terminare dopo aver eseguito un numero finito di passi, fate in modo da concludere l'esecuzione quando il risultato è stato determinato con sei cifre significative.

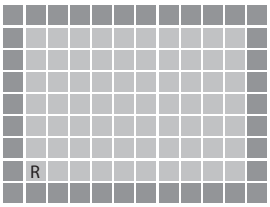
* **P1.4 (economia).** Immaginate di andare a cena in un ristorante di lusso con alcuni amici e che, dopo aver chiesto il conto, vogliate pagare dividendo equamente tra tutti l'importo totale e la mancia (che è il 15% del totale del conto). Scrivete pseudocodice che calcoli quanto dovuto da ciascuno. Il programma deve visualizzare l'importo totale del conto (esclusa la mancia), l'importo della mancia totale, la somma delle due quantità e quanto dovuto da ciascuno. In quest'ultimo importo devono essere evidenziate le quote per il conto totale e per la mancia.

** **P1.5.** Scrivete un algoritmo che crei una disposizione geometrica di piastrelle bianche e nere, con un bordo perimetrale di piastrelle nere e due o tre piastrelle nere al centro, equidistanti dai bordi. I valori forniti in ingresso all'algoritmo sono il numero totale di righe e di colonne che costituiscono lo schema.



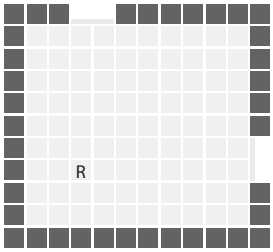
*** **P1.6.** Scrivete un algoritmo che guidi un robot nel falciare un prato rettangolare, partendo da un angolo, considerando che il robot può:

- Spostarsi in avanti di un riquadro nella griglia idealmente sovrapposta al terreno.
- Ruotare a sinistra o a destra, cioè, rispettivamente, in senso antiorario oppure orario.
- Percepire con un sensore il colore del terreno presente nel riquadro di spazio che si trova nella direzione in cui si sposterebbe se facesse uno spostamento.



*** **P1.7.** Immaginate un robot all'interno di una stanza, considerando che il robot può:

- Spostarsi in avanti di un riquadro nella griglia idealmente sovrapposta al pavimento.
- Ruotare a sinistra o a destra, cioè, rispettivamente, in senso antiorario oppure orario.
- Percepire con un sensore ciò che si trova nella direzione in cui si sposterebbe se facesse uno spostamento: un muro, una finestra o niente.



Scrivete un algoritmo che guidi il robot nel conteggio del numero di finestre presenti nella stanza, a partire da una posizione iniziale qualsiasi. Nell'esempio qui raffigurato, il robot,

partendo dalla posizione contrassegnata con la lettera R, deve scoprire che nella stanza sono presenti due finestre.

★★★ **P1.8.** Immaginate un robot all'interno di un labirinto. La regola della mano destra afferma che si può uscire da un labirinto tenendo sempre la mano destra appoggiata a un muro del labirinto: prima o poi si arriva a un'uscita. Ipotizzate che il robot possa:

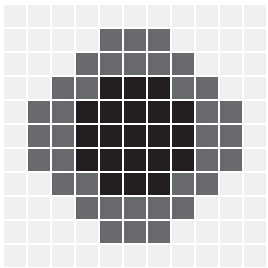
- Spostarsi in avanti di un riquadro nella griglia idealmente sovrapposta al labirinto.
- Ruotare a sinistra o a destra, cioè, rispettivamente, in senso antiorario oppure orario.
- Percepire con un sensore ciò che si trova nella direzione in cui si sposterebbe se facesse uno spostamento: un muro, un'uscita o niente.

Scrivete un algoritmo che guidi il robot verso un'uscita del labirinto. Potete fare l'ipotesi che esista un'uscita raggiungibile seguendo la regola della mano destra. La cosa difficile è gestire le situazioni in cui il percorso nel labirinto fa una curva, perché il robot non vede le curve: può solo vedere ciò che si trova nella direzione di marcia.

★★★ **P1.9 (economia).** Immaginate di essere destinatari di una offerta-fedeltà che vi conceda il diritto di acquistare gratuitamente un articolo il cui prezzo non superi $100, scegliendolo da un catalogo, e volete fare la scelta migliore, scegliendo l'articolo che più si avvicina al prezzo massimo (il catalogo contiene articoli di prezzo inferiore o superiore a $100). Scrivete un algoritmo che individui l'articolo il cui prezzo si avvicina maggiormente a $100, senza superare tale limite; se più articoli hanno tale stessa caratteristica, elencateli tutti. Ricordate che un calcolatore può soltanto analizzare un articolo alla volta: non può semplicemente dare un'occhiata all'elenco e trovare il migliore, come faremmo noi.

★★ **P1.10 (scienze).** La pubblicità di un televisore che vi interessa riporta la dimensione della diagonale dello schermo e vi chiedete se troverà posto nel vostro salotto. Scrivere un algoritmo che ricavi le dimensioni orizzontale e verticale del televisore, ricevendo come dato in ingresso la dimensione della diagonale e il fattore di forma (che è il rapporto tra la larghezza e l'altezza, oggi solitamente uguale a 16:9).

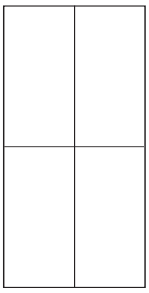
★★★ **P1.11 (scienze).** Le fotocamere più moderne sono in grado di correggere l'effetto "occhi rossi" provocato dall'uso del flash. Scrivete lo pseudocodice per un algoritmo che individui gli occhi rossi, a partire da una matrice quadrettata e colorata fornita in ingresso.



Sono dati il numero di righe e il numero di colonne della matrice e il colore di ciascun quadratino (tecnicamente chiamato *pixel*), che può essere nero (la pupilla dell'occhio), rosso (*grigio scuro nella figura*, attorno alla pupilla) o di altri colori (*grigio chiaro nella figura*). Quando il centro di una zona di pixel neri coincide con il centro di una zona di pixel rossi si è individuato un occhio rosso e l'algoritmo deve fornire "yes" (*si*) come risposta, altrimenti "no".

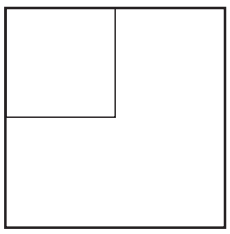
Capitolo 2

** **P2.1.** Scrivete il programma `FourRectanglePrinter` che costruisca un oggetto `box` di tipo `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.



Il programma non deve visualizzare alcunché di grafico, deve semplicemente visualizzare le posizioni dei quattro rettangoli.

** **P2.2.** Scrivete il programma `GrowSquarePrinter` che costruisca un oggetto di tipo `Rectangle`, memorizzato nella variabile `square`, che rappresenti un quadrato con i lati di lunghezza 50 e il vertice 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 vertice superiore sinistro. Se i quadrati venissero disegnati, darebbero luogo alla figura qui visibile.

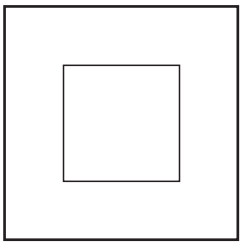


Il programma non deve visualizzare alcunché di grafico, deve semplicemente visualizzare le posizioni di `square` prima e dopo l'invocazione dei metodi modificatori.

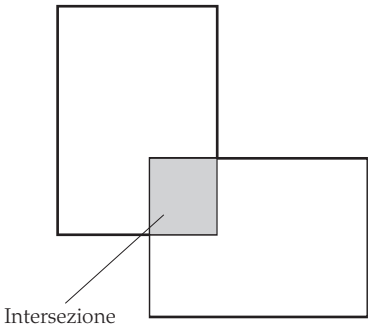
Consultate la documentazione API del metodo `grow`.

*** **P2.3.** Scrivete il programma `CenteredSquaresPrinter` che costruisca un oggetto `square` di tipo `Rectangle` che rappresenti un quadrato con i lati di lunghezza 200 e il vertice 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 `grow` e `translate`; infine, va invocato di nuovo `System.out.println(square)`. Le invocazioni di `grow` e `translate` devono modificare il quadrato in modo che la dimensione del suo lato venga dimezzata e che la sua posizione sia centrata rispetto a quella originale. Se i quadrati venissero disegnati, darebbero luogo alla figura qui visibile. Il programma non deve visualizzare alcunché di grafico, deve semplicemente visualizzare le posizioni di `square` prima e dopo l'invocazione dei metodi modificatori. Consultate la documentazione API del metodo `grow`.



★★★ **P2.4.** Il metodo `intersection` calcola l'*intersezione* di due rettangoli, ovvero il rettangolo formato dalla sovrapposizione parziale di altri due rettangoli, come si può vedere nell'esempio in figura.



Il metodo viene invocato in questo modo:

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

Scrivete il programma `IntersectionPrinter` che costruisca due rettangoli, li visualizzi come descritto nell'esercizio P2.1 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.

★★★ **P2.5 (grafica).** In questo esercizio scoprirete una semplice modalità di visualizzazione di un oggetto di tipo `Rectangle`. Il metodo `setBounds` della classe `JFrame` sposta e ridimensiona un frame in modo che abbia come bordo 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 usatelo come bordo del frame

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

      // qui dovete scrivere voi:
      // spostate il rettangolo e usatelo come nuovo bordo del frame
   }
}
```

** **P2.6.** La classe `BigInteger` rappresenta numeri interi aventi un numero di cifre arbitrario (come vedrete nel Capitolo 4, il tipo di dato `int` non è in grado di rappresentare numeri interi molto grandi). Si può costruire un oggetto `BigInteger` fornendo al costruttore una stringa contenente le sue cifre, in questo modo:

```
BigInteger a = new BigInteger(""12345678987654321");
```

Scrivete un programma che visualizzi le potenze di `a` con esponente due, quattro e otto, usando uno dei metodi della classe `BigInteger`.

*** **P2.7.** Scrivete il 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). Costruite un oggetto di tipo `Random` (una classe vista nell'esercizio E2.13) e invocate il suo metodo più appropriato per generare ciascun numero. Il programma deve visualizzare una frase beneaugurante, del tipo "Ecco la combinazione che ti farà ricco", seguita da una combinazione per la lotteria.

** **P2.8.** Usando la classe `Day` vista nella sezione Esempi completi 2.1, scrivete un programma che costruisca un oggetto di tipo `Day` rappresentante il 28 febbraio di quest'anno e altri tre oggetti che rappresentino il 28 febbraio dei tre anni successivi. Aggiungete un giorno a ciascuno degli oggetti e visualizzateli, insieme ai valori previsti:

```
2019-03-01
Expected: 2019-03-01
2020-03-01
Expected: 2020-03-01
2021-03-01
Expected: 2021-03-01
2022-02-29
Expected: 2022-02-29
```

*** **P2.9.** 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 si usano 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 un noto informatico.

Suggerimento: La classe `GregorianCalendar` è molto articolata ed è bene scrivere alcuni piccoli programmi d'esempio per comprenderne appieno la API prima di affrontare il problema nel suo complesso. Iniziate con un programma che costruisca la data di oggi, vi aggiunga dieci giorni e visualizzi il giorno risultante sotto forma di giorno del mese e giorno della settimana.

** **P2.10.** La classe `LocalDate` descrive una data in modo indipendente dalla posizione geografica (cioè dal fuso orario). Si costruisce un oggetto di tipo `LocalDate` in questo modo:

```
LocalDate today = LocalDate.now(); // la data di oggi
LocalDate eckertsBirthday = LocalDate(1919, 4, 9);
```

Per aggiungere un certo numero di giorni a una data rappresentata da un oggetto di tipo `LocalDate` si può usare il metodo `plusDays`:

```
LocalDate later = today.plusDays(10); // dieci giorni nel futuro da oggi
```

Questo metodo non modifica l'oggetto `today`, bensì restituisce un nuovo oggetto che rappresenta una data distante un dato numero di giorni da `today`.

Per conoscere l'anno di una data, si invoca:

```
int year = today.getYear();
```

Per conoscere il giorno della settimana di una data, si invoca:

```
String weekday = today.getDayOfWeek().toString();
```

Dovete scrivere un programma che visualizzi

- Il giorno della settimana del "Giorno del pi greco" (*Pi day*, il 14 marzo) di quest'anno.
- La data e il giorno della settimana del "Giorno del programmatore" di quest'anno (è il 256-esimo giorno dell'anno, perché il numero $256 = 2^8$ è usato nella programmazione).
- La data e il giorno della settimana del giorno che dista da oggi 10 000 giorni nel futuro.

*** **P2.11 (collaudo).** Scrivete il programma `LineDistanceTester` che costruisca un segmento avente come estremi i punti (100, 100) e (200, 200), per poi costruire anche i punti (100, 200), (150, 150)

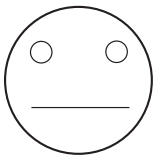
e (250, 50), visualizzando la distanza del segmento da ciascuno dei tre punti, calcolata usando il metodo `ptSegDist` della classe `Line2D`, oltre ai corrispondenti valori previsti (eventualmente facendo una figura su un foglio di carta per calcolarli).

** **P2.12 (grafica).** Risolvete nuovamente l'esercizio precedente, scrivendo un'applicazione grafica che mostri il segmento e i punti, ciascuno dei quali va disegnato mediante un piccolo cerchio. Usate il metodo `drawString` per scrivere il valore di ciascuna distanza in una posizione vicina al punto corrispondente, usando invocazioni come questa:

```
g2.drawString("Distance: " + distance, p.getX(), p.getY());
```

** **P2.13 (grafica).** Scrivete un programma grafico che disegni 12 stringhe, una per ciascuno dei 12 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`.

** **P2.14 (grafica).** Progettando le classi `FaceViewer` e `FaceComponent`, scrivete un programma per tracciare la faccia qui visibile.



** **P2.15 (grafica).** Scrivete un programma grafico che disegni un semaforo.

** **P2.16 (grafica).** 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 E2.11)
- Per ottenere una valutazione migliore, aggiungete un'immagine di voi stessi. *Suggerimento*: costruite un oggetto di tipo `ImageIcon`.

Capitolo 3

★ **P3.1.** Migliorate la classe `CashRegister` in modo che conti gli articoli acquistati, dotandola di un metodo `getItemCount` che restituisca tale conteggio.

★★★ **P3.2.** Migliorate la classe `CashRegister` in modo che calcoli le tasse che gravano sulle vendite. La percentuale di tassazione viene fornita al costruttore. Aggiungete i metodi `recordTaxablePurchase` e `getTotalTax:` il primo registra l'acquisto di un articolo sottoposto a tassazione (mentre `recordPurchase` compie la stessa azione per articoli esenti) e il secondo restituisce l'ammontare complessivo delle tasse. Il metodo `giveChange` deve gestire correttamente le tasse.

★★ **P3.3.** Realizzate la classe `Balloon` che rappresenti un pallone (sferico) gonfiabile, che viene costruito con raggio uguale a zero. Progettate il metodo:

```
public void inflate(double amount)
```

che gonfi il pallone, aumentandone il raggio della quantità indicata, e il metodo:

```
public double getVolume()
```

che restituisca il volume raggiunto dal pallone. Usate `Math.PI` come valore di π e, per elevare al cubo un valore `r`, usate semplicemente l'espressione `r * r * r`.

★★ **P3.4.** Con l'eccezione di Canada, Colombia, Repubblica Dominicana, Messico e Stati Uniti d'America, per le dimensioni dei fogli di carta la maggior parte degli stati segue lo standard ISO 216. Un foglio A0 misura 841 × 1189 millimetri. Un foglio A1 si ottiene tagliando un foglio A0 a metà lungo la sua dimensione maggiore, ottenendo un foglio 594 × 841. Di nuovo, un foglio A2 si ottiene compiendo la stessa operazione a partire da un foglio A1, ottenendo un foglio 420 × 594, e così via. Progettate la classe `Sheet`, il cui costruttore generi un foglio A0 e il cui metodo `cutInHalf` dimezzi la dimensione del foglio a cui viene applicato. Dotate la classe anche dei metodi `width`, `height` e `name`, che restituiscano, rispettivamente, la larghezza e l'altezza del foglio (in millimetri) e il suo nome (ad esempio, `"A2"`).

★★ **P3.5.** Il pannello di controllo di un forno a microonde ha quattro pulsanti: uno per aumentare di 30 secondi il tempo di funzionamento, uno per cambiare il livello di potenza (che può avere i valori 1 o 2), uno per riportare i valori a uno stato predefinito (pulsante di *reset*) e uno per far funzionare il forno (pulsante *start*). Realizzate una classe che simuli il funzionamento del microonde, con un metodo per ciascun pulsante. Il metodo corrispondente al pulsante *start* deve visualizzare il messaggio "Cooking for ... seconds at level ..." (*in funzione per … secondi al livello di potenza …*").

★★ **P3.6.** Un oggetto di tipo `Person` (*persona*) ha un nome (usiamo soltanto il primo nome, per semplicità) e alcuni amici, il cui nome è memorizzato in un'unica stringa, con uno spazio come carattere di separazione tra due nomi. Progettate un costruttore che riceva come parametro il nome della persona: l'oggetto corrispondente viene costruito privo di amici. La classe deve avere i metodi:

```
public void befriend(Person p) // la persona e p diventano amici
public void unfriend(Person p) // la persona e p non sono più amici
public String getFriendNames() // restituisce i nomi degli amici
```

★ **P3.7.** Aggiungete alla classe dell'esercizio precedente un metodo che restituisca il numero di amici:

```
public int getFriendCount()
```

★★★ **P3.8.** Realizzate una classe `Student` (*studente*). In 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 il punteggio medio. Per calcolare tale valore medio dovete registrare anche il *numero dei questionari* a cui lo studente ha risposto. Progettate anche la classe `StudentTester` che collaudi tutti i metodi.

★ **P3.9.** Scrivete la classe `Battery` che rappresenti una batteria ricaricabile. Progettate il costruttore:

```
public Battery(double capacity)
```

dove la capacità della batteria è misurata in milliampere per ora (mAh). Una tipica batteria AA ha una capacità compresa tra 2000 e 3000 mAh. Il metodo:

```
public void drain(double amount)
```

diminuisce della quantità specificata la capacità della batteria, mentre il metodo:

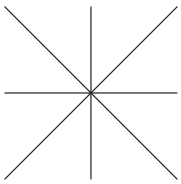
```
public void charge()
```

riporta la batteria alla sua capacità iniziale. Infine, il metodo:

```
public double getRemainingCapacity()
```

consente di ispezionare la capacità residua della batteria.

★★ **P3.10 (grafica).** Scrivete un programma che disegni tre stelle come quella qui visibile, usando le classi `Star`, `StarComponent` e `StarViewer`.



★★ **P3.11.** Realizzate la classe `RoachPopulation` che simuli la crescita di una popolazione di scarafaggi. Il costruttore riceve la dimensione della popolazione iniziale. Il metodo `breed` simula un periodo di tempo in cui la popolazione raddoppia. Il metodo `spray(double percent)` simula una spruzzata di insetticida, che riduce la popolazione della percentuale indicata. 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 riducendola del 10% e visualizzate il numero di scarafaggi. Ripetete la procedura altre tre volte.

★★ **P3.12.** Realizzate la classe `VotingMachine` che possa essere utilizzata per una semplice elezione, con i metodi per azzerare il conteggio dei voti, per assegnare un voto ai Democratici, per assegnare un voto ai Repubblicani e per conoscere il numero totale di voti ricevuti da ciascuno dei due partiti.

★★★ **P3.13.** 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 semplice: addebitate una commissione per ogni versamento e per ogni prelievo. Consentite l'impostazione dell'importo della commissione e modificate i metodi

`deposit` e `withdraw` in modo che tale commissione venga prelevata dal saldo. Collaudate la classe risultante e verificate che le commissioni vengano gestite correttamente.

Fate ora una modifica più complessa. La banca consentirà di effettuare un numero fisso di operazioni gratuite (indifferentemente versamenti o prelievi) ogni mese 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, a questo scopo, 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, infine, un programma di collaudo che verifichi la correttezza del funzionamento della classe simulando operazioni per alcuni mesi.

★★★ **P3.14.** 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. Create due oggetti di tale classe, invocandone i relativi metodi `sayHello:` ovviamente i due oggetti restituiscono il medesimo saluto.

Migliorate la classe `Greeter` in modo che ciascun oggetto produca un saluto personalizzabile. Ad esempio, il metodo `sayHello` dell'oggetto costruito con `new Greeter("Dave")` dovrebbe restituire `"Hello, Dave!"` (usate il metodo `concat` per unire due stringhe in modo da formare una stringa più lunga, oppure date un'occhiata al Paragrafo 4.5 per vedere come si possa usare l'operatore `+` per ottenere lo stesso risultato).

Aggiungete alla classe `Greeter` anche un metodo `sayGoodbye`, che generi un saluto analogo, con "Goodbye" al posto di "Hello".

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."

Se usate BlueJ, posizionate nello spazio di lavoro (*workbench*) due esemplari di Greeter (uno che saluti il mondo, *world*, e uno che saluti Dave) e invocate i loro metodi; altrimenti, scrivete un programma di collaudo che costruisca tali oggetti, ne invochi i metodi e visualizzi i risultati.

Capitolo 4

★★★ **P4.1.** Scrivete un programma che aiuti l'utente a decidere se convenga acquistare un'automobile con motore ibrido. I dati acquisiti dal programma sono:

- Il costo di una nuova automobile
- Una stima delle miglia percorse in un anno
- Una stima del costo del carburante
- L'efficienza del motore in miglia percorse per gallone
- Una stima del prezzo di vendita dell'automobile usata dopo 5 anni

Calcolate il costo totale relativo al possesso e alla gestione dell'auto per cinque anni (senza tener conto, per semplicità, degli oneri finanziari). Cercate di ottenere prezzi realistici per un'auto ibrida nuova e usata, oltre a quelli di un'auto simile con motore a benzina. Eseguite il programma due volte, usando il prezzo attuale del carburante e una percorrenza annuale di 15000 miglia. Iniziate dallo pseudocodice.

** **P4.2.** Scrivete una classe che chieda all'utente un anno e calcoli per quell'anno 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.

** **P4.3.** Scrivete un programma che legga un numero fornito dall'utente e faccia, nell'ordine indicato, quanto segue: eliminare tutte le cifre tranne le ultime tre; invertire le cifre, sottraendo le tre cifre originali da quelle invertite (ignorando eventuali segni negativi); invertire le cifre della differenza ottenuta e sommare tra loro la differenza e la differenza invertita; infine, visualizzare la somma così ottenuta. Ad esempio:

Dato iniziale: 371
Dopo l'inversione: 173
Differenza: 198
Differenza invertita: 891
Somma: 1089

Il procedimento illustrato viene a volte chiamato "rompicapo del 1089", perché nella maggior parte dei casi il risultato è 1089.

*** **P4.4.** In questo progetto elaborerete triangoli. Un triangolo è definito dalle coordinate x e y dei suoi 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

Progettate la 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.

** **P4.5.** Una barca naviga in un oceano bidimensionale e ha una posizione e una direzione. Si può spostare di una data distanza lungo la sua direzione attuale, oppure può ruotare su se stessa di un dato angolo. Progettate una classe che ne simuli il funzionamento, con i metodi seguenti:

```
public double getX()
public double getY()
public double getDirection()
public void turn(double degrees)
public void move(double distance)
```

*** **P4.6.** 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, dovete progettare una classe separata, `Coin`, che rappresenti un valore monetario, e un registratore di cassa che possa funzionare con monete di ogni tipo.

** **P4.7 (economia).** Lo pseudocodice seguente descrive come, in una libreria, viene calcolato l'importo di un ordine a partire dal costo totale dei libri ordinati e dal loro numero: traducetelo in un programma Java.

```
Leggere il costo totale dei libri e il numero di libri.
Calcolare le tasse (il 7.5 per cento del costo totale dei libri).
Calcolare i costi di spedizione ($2 per ciascun libro).
Il prezzo totale dell'ordine è la somma del costo totale dei libri,  delle tasse e dei costi di spedizione.
Visualizzare l'importo dell'ordine.
```

** **P4.8 (economia).** Lo pseudocodice seguente descrive come trasformare una stringa contenente un numero telefonico a dieci cifre (come `"4155551212"`) in una stringa più facilmente leggibile, con parentesi e trattini, secondo lo stile statunitense, come `"(415) 555-1212"`.

```
Prendere la sottostringa costituita dai primi tre caratteri e circondarla
    con parentesi tonde (questo è il prefisso, area code).
Concatenare il prefisso con la sottostringa contenente i tre caratteri
    successivi, seguita da un trattino e dalla sottostringa costituita
    dagli ultimi quattro caratteri, ottenendo il numero nel formato richiesto.
```

Traducete questo pseudocodice in un programma Java che acquisisca un numero telefonico in ingresso, memorizzandolo in una stringa, per poi visualizzarlo nel formato appena descritto.

** **P4.9 (economia).** Lo pseudocodice seguente descrive come estrarre, da un prezzo espresso mediante un numero in virgola mobile, il numero di dollari e il numero di centesimi. Ad esempio, dato un prezzo di 2.95 dollari, si ottengono i valori 2 e 95, rispettivamente per i dollari e i centesimi.

```
Assegnare il prezzo alla variabile numerica intera "dollari".
Moltiplicare per 100 la differenza prezzo – dollari e aggiungervi 0.5.
Assegnare il risultato alla variabile numerica intera "centesimi".
```

Traducete questo pseudocodice in un programma Java che acquisisca un prezzo in ingresso e ne visualizzi i dollari e i centesimi. Collaudate il programma con i prezzi 2.95 e 4.35.

** **P4.10 (economia).** *Dare il resto*. Realizzate un programma che aiuti un cassiere a dare il resto. Il programma riceve due dati in ingresso: la somma da pagare e la quantità di denaro pagata dal cliente. Visualizzate il resto dovuto sotto forma di quantità di monete da un dollaro, da un quarto di dollaro (*quarter*), da dieci centesimi (*dime*), da cinque centesimi (*nickel*) e da un centesimo (*penny*). Per evitare errori di arrotondamento, l'utente del programma deve fornire entrambi i valori in centesimi, scrivendo, ad esempio, 274 invece di 2.74.

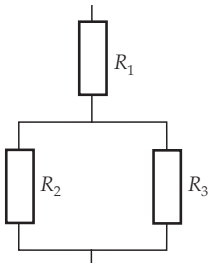
* **P4.11 (economia).** Una banca *online* vi ha chiesto di progettare un programma che mostri ai potenziali clienti come aumenteranno i loro depositi. Il programma deve acquisire in ingresso il saldo iniziale e il tasso di interesse annuo (ma gli interessi vengono calcolati e accreditati mensilmente), per poi visualizzare il saldo dopo i primi tre mesi, come in questo esempio di esecuzione:

```
Initial balance: 1000
Annual interest rate in percent: 6.0
After first month:     1005.00
After second month:   1010.03
After third month:    1015.08
```

** **P4.12 (economia).** Un negozio di videonoleggio vuole premiare i propri clienti migliori con uno sconto basato sul numero di noleggi di film e sul numero di nuovi clienti che si sono registrati presso il negozio in seguito al suggerimento del cliente stesso. Lo sconto agisce come percentuale ed è uguale alla somma dei noleggi effettuati e dei nuovi clienti presentati, ma non può eccedere il 75 per cento (*suggerimento*: `Math.min`). Scrivete un programma che calcoli il valore dello sconto, come in questo esempio di esecuzione:

```
Enter the number of movie rentals: 56
Enter the number of members referred to the video club: 3
The discount is equal to: 59.00 percent.
```

* **P4.13 (scienze).** In relazione al circuito qui raffigurato, scrivete un programma che acquisisca come dati in ingresso le resistenze dei tre resistori e calcoli la resistenza totale, usando la legge di Ohm.



** **P4.14 (scienze).** Il punto di rugiada T_d può essere approssimativamente calcolato a partire dall'umidità relativa, RH, e dalla temperatura effettiva, T, con queste formule, dove $a = 17.27$ e $b = 237.7$ °C:

$$T_d = \frac{b \cdot f(T, RH)}{a - f(T, RH)}$$

$$f(T, RH) = \frac{a \cdot T}{b + T} + \ln(RH)$$

Scrivete un programma che acquisisca in ingresso l'umidità relativa (un numero compreso tra 0 e 1) e la temperatura (in °C), visualizzando il corrispondente punto di rugiada. Per calcolare il logaritmo naturale (ln) presente nella formula, usate `Math.log`.

★★★ **P4.15 (scienze).** Alcuni sensori di temperatura possono essere collegati direttamente ai tubi in rame per misurare la temperatura dei liquidi al loro interno. Ciascun sensore contiene un dispositivo a semiconduttore, detto *termistore*, che è caratterizzato da una resistenza dipendente dalla temperatura secondo la legge seguente:

$$R = R_0 \, e^{\beta\left(\frac{1}{T} - \frac{1}{T_0}\right)}$$

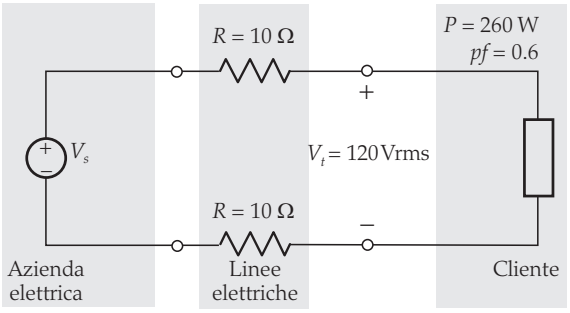
dove R è la resistenza (in Ω) alla temperatura T (in °K) e R_0 è la resistenza (in Ω) alla temperatura T_0 (in °K), mentre β è una costante che dipende dal materiale usato per fabbricare il termistore. In pratica, quindi, un termistore viene specificato mediante i suoi tre parametri, R_0, T_0 e β.

I termistori usati per i sensori di questo esercizio hanno $R_0 = 1075\ \Omega$, $T_0 = 85$ °C e $\beta = 3969$ °K (si noti che il coefficiente β è espresso in °K e si ricordi che la temperatura in °K si ottiene aggiungendo 273 alla temperatura espressa in °C). La temperatura del liquido, in °C, è determinata dalla resistenza R, espressa in Ω, usando la formula seguente:

$$T = \frac{\beta T_0}{T_0 \ln\left(\frac{R}{R_0}\right) + \beta} - 273$$

Scrivete un programma Java che chieda all'utente la resistenza R del termistore e visualizzi un messaggio che riporti la temperatura del liquido, in °C.

★★★ **P4.16 (scienze).** Il circuito qui raffigurato illustra gli aspetti più rilevanti della connessione esistente tra un'azienda che fornisce energia elettrica e uno dei suoi clienti. Il cliente viene rappresentato da tre parametri: V_t, P e *pf*. Il parametro V_t è la tensione disponibile collegandosi a una presa a muro: i clienti si aspettano di ottenere una tensione V_t affidabile, affinché i propri apparecchi elettrici funzionino correttamente, per cui le aziende distributrici regolano con particolare cura tale parametro. Il parametro P è la potenza usata dal cliente ed è il fattore principale nella determinazione della sua bolletta elettrica. Il fattore di potenza, *pf*, è meno noto (viene calcolato come coseno di un angolo, per cui il suo valore è compreso tra zero e uno). In questo esercizio vi si chiede di scrivere un programma Java che indaghi sul significato del fattore di potenza.

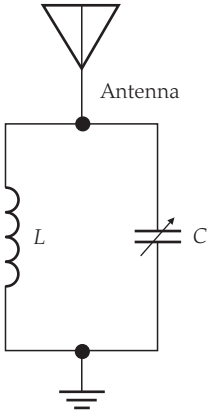


Nella figura le linee elettriche sono rappresentate, in modo piuttosto semplificato, mediante resistenze, misurate in Ohm. Il modello dell'azienda elettrica è una sorgente di tensione in corrente alternata e la tensione V_s necessaria per fornire al cliente la potenza P alla tensione V_t può essere calcolata usando questa formula (dove la tensione V_s ha Vrms come unità di misura):

$$V_s = \sqrt{\left(V_t + \frac{2RP}{V_t}\right)^2 + \left(\frac{2RP}{pfV_t}\right)^2 \left(1 - pf^2\right)}$$

Da questa formula si evince che il valore di V_s dipende dal valore di *pf*. Scrivete un programma Java che chieda all'utente il valore del fattore di potenza e, poi, visualizzi un messaggio contenente il valore di V_s corrispondente, usando i valori di P, R e V_t presenti nella figura.

★★★ **P4.17 (scienze).** Il circuito di sintonizzazione rappresentato in figura è connesso a un'antenna e C è un valore di capacità variabile tra C_{min} e C_{max}.



Il circuito di sintonizzazione seleziona la frequenza $f = \frac{1}{2\pi\sqrt{LC}}$. Per progettare il circuito adatto a una determinata frequenza, si calcola per prima cosa $C = \sqrt{C_{\text{min}}C_{\text{max}}}$, per poi calcolare l'induttanza L necessaria sulla base dei valori di f e C. A questo punto il circuito può essere sintonizzato su qualsiasi frequenza appartenente all'intervallo delimitato $f_{\text{min}} = \frac{1}{2\pi\sqrt{LC_{\text{max}}}}$ da e da $f_{\text{max}} = \frac{1}{2\pi\sqrt{LC_{\text{min}}}}$.

Scrivete un programma Java che aiuti a progettare il circuito di sintonizzazione per una specifica frequenza, usando un condensatore di capacità C variabile tra C_{min} e C_{max} (valori di ingresso tipici potrebbero essere f = 16.7 MHz, C_{min} = 14 pF e C_{max} = 365 pF). Il programma deve acquisire, come dati in ingresso, f (in Hz), C_{min} e C_{max} (in F), visualizzando il valore di induttanza richiesto e l'intervallo di frequenze sulle quali può essere sintonizzato il circuito variando la capacità.

★ **P4.18 (scienze).** In base alla legge di Coulomb, la forza elettrica (misurata in N) tra due particelle elettricamente cariche, con carica pari a Q_1 e Q_2 (misurate in C) e poste a una distanza di r metri, è uguale a $F = Q_1Q_2/(4\pi\varepsilon r^2)$, con $\varepsilon = 8.854 \times 10^{-12}$ F/m. Scrivete un programma che calcoli e visualizzi la forza agente su una coppia di particelle cariche, basandosi sui dati acquisiti dall'utente: Q_1, Q_2 e r.

Capitolo 5

** **P5.1.** L'algoritmo seguente individua la stagione (Spring, Summer, Fall o Winter, cioè, rispettivamente, primavera, estate, autunno o inverno) a cui appartiene una data, fornita come mese e giorno, due numeri interi.

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

Scrivete un programma che chieda all'utente un mese e un giorno e, poi, visualizzi la stagione determinata da questo algoritmo. Progettate la classe `Date` dotata del metodo `getSeason`.

** **P5.2.** Scrivete un programma che chieda all'utente il giorno e il mese del suo compleanno, per poi visualizzare un oroscopo. Inserite frasi dedicate ai programmatori, come questa:

```
Please enter your birthday (month and day): 6 16
I Gemelli sono esperti nella comprensione del comportamento di
programmi complicati. Sei in grado di intuire la causa degli errori
e, quindi, sei sempre un passo avanti. Questa sera il tuo stile di
programmazione troverà l'approvazione di una persona fortemente
critica.
```

Ogni frase deve contenere il nome del segno astrologico (potete trovare i nomi e gli intervalli di date per ciascun segno in un numero sorprendentemente elevato di siti Internet). Progettate la classe `Date` dotata del metodo `getFortune`.

** **P5.3.** Scrivete un programma che calcoli le tasse secondo questo schema.

Stato civile è "non coniugato" e reddito imponibile superiore a	**ma non superiore a**	**le tasse sono**	**della somma superiore a**
$ 0	$ 8000	10%	$ 0
$ 8000	$ 32 000	$ 800 + 15%	$ 8000
$ 32 000		$ 4400 + 25%	$ 32 000
Stato civile "coniugato" e reddito imponibile superiore a	**ma non superiore a**	**le tasse sono**	**della somma superiore a**
$ 0	$ 16 000	10%	$ 0
$ 16 000	$ 64 000	$ 1600 + 15%	$ 16 000
$ 64 000		$ 8800 + 25%	$ 64 000

*** **P5.4.** Il programma `TaxReturn.java` visto nel Paragrafo 5.4 usa una versione semplificata dello schema di tassazione sul reddito in vigore negli Stati Uniti nel 2008. Cercate gli scaglioni di reddito

e le relative percentuali per l'anno in corso, tanto per persone coniugate quanto per persone non coniugate, e realizzate un programma che calcoli l'importo della tassazione sul reddito con lo schema attuale.

*** **P5.5.** *Conversione di unità di misura*. Scrivete un programma per la conversione di unità di misura che chieda all'utente da quale unità (scegliendo tra: fl. oz, gal, oz, lb, in, ft, mi) e verso quale unità (scegliendo tra: ml, l, g, kg, mm, cm, m, km) vuole effettuare una conversione, rifiutando conversioni incompatibili (come, ad esempio, da gal a km). Chiedete, poi, il valore da convertire e, infine, visualizzate il risultato:

```
Convert from? gal
Convert to? ml
Value? 2.5
2.5 gal = 9462.5 ml
```

* **P5.6.** Scrivete un programma che legga le coordinate *x* e *y* di due vertici opposti di un rettangolo e visualizzi un messaggio che indichi se il rettangolo è un quadrato ("square"), oppure se è orientato in senso orizzontale ("landscape", cioè con l'altezza inferiore alla larghezza) o, al contrario, verticale ("portrait").

** **P5.7.** Scrivete un programma che legga le coordinate *x* e *y* dei tre vertici di un triangolo e visualizzi un messaggio che indichi se ha un angolo ottuso, un angolo retto o soltanto angoli acuti.

*** **P5.8.** Scrivete un programma che legga le coordinate *x* e *y* dei quattro vertici di un quadrilatero e visualizzi un messaggio che indichi se è un quadrato, un rettangolo, un trapezio, un rombo oppure non ha nessuna di queste forme.

*** **P5.9.** Un anno di 366 giorni viene detto bisestile (*leap year*) e serve a mantenere il calendario sincronizzato con il Sole, dal momento che la Terra vi ruota attorno una volta ogni 365.25 giorni. In realtà, questo numero non è esatto e per tutte le date successive al 1582 si applica la *correzione gregoriana*: gli anni divisibili per 4, come il 1996, sono bisestili, ma gli anni divisibili anche per 100, come il 1900, non lo sono; come eccezione all'eccezione, gli anni divisibili anche per 400, come il 2000, sono bisestili. Scrivete un programma che chieda all'utente un anno e determini se si tratta di un anno bisestile. Progettate la classe `Year` dotata del metodo `isLeapYear` che usi un unico enunciato `if` (con gli opportuni operatori booleani).

*** **P510.** *Numeri romani*. Scrivete un programma che converta un numero intero positivo nel corrispondente valore espresso nel sistema di numerazione romano. I numeri romani sono costituiti da sette diverse cifre, con questi valori:

I	1
V	5
X	10
L	50
C	100
D	500
M	1000

e i numeri vengono composti seguendo queste regole:

a. Si possono rappresentare soltanto i numeri fino a 3999.
b. Come nel sistema decimale, le migliaia, le centinaia, le decine e le unità vengono espresse separatamente.

c. I numeri da 1 a 9 sono rappresentati, in ordine crescente, da I, II, III, IV, V, VI, VII, VIII, IX; come potete notare, una lettera I che precede la lettera V o X rappresenta un'unità che viene *sottratta* dal valore; inoltre, non si possono avere più di tre I consecutive.
d. Le decine e le centinaia sono gestite allo stesso modo, tranne per il fatto che, al posto delle lettere I, V e X, si usano le lettere X, L e C e, rispettivamente, le lettere C, D e M.

Il programma deve acquisire un numero in ingresso, come 1978, e convertirlo secondo la numerazione romana, visualizzando, in questo esempio, MCMLXXVIII.

★★★ **P5.11.** In francese i nomi delle nazioni sono femminili quando terminano con la lettera e, altrimenti sono maschili, con l'eccezione dei nomi seguenti, che sono maschili anche se terminano con la e:

- le Belize
- le Cambodge
- le Mexique
- le Mozambique
- le Zaïre
- le Zimbabwe

Scrivete un programma che acquisisca come dato in ingresso il nome di una nazione in francese e vi aggiunga l'articolo: "le" per i nomi maschili e "la" per quelli femminili, come "le Canada" o "la Belgique". Se, però, il nome della nazione inizia con una vocale, l'articolo diventa "l", seguito da apostrofo (ad esempio, l'Afghanistan). Infine, per i paesi qui elencati, che hanno un nome plurale, si usa l'articolo "les":

- les Etats-Unis
- les Pays-Bas

★★★ **P5.12 (economia).** Scrivete un programma che simuli una transazione bancaria. Occorre gestire due conti bancari: un conto corrente (*checking account*) e un conto di risparmio (*savings account*). Per prima cosa chiedete all'utente i saldi iniziali dei due conti, rifiutando saldi negativi. Poi chiedete quale sia il tipo di operazione da eseguire: versamento (*deposit*), prelievo (*withdrawal*) o bonifico (*transfer*). Ancora, chiedete il tipo di conto: *checking* o *savings*. Rifiutate transazioni che rendano negativo il saldo di un conto. Infine, visualizzate i saldi dei due conti dopo l'operazione richiesta.

★★ **P5.13 (economia).** Quando con la vostra carta utilizzate uno sportello bancario automatico (ATM, *automatic teller machine*), dovete usare un numero identificativo personale (PIN, *personal identification number*) per poter accedere al vostro conto. Se un utente sbaglia per tre volte l'inserimento del PIN, la macchina trattiene la carta e la blocca. Nell'ipotesi che il PIN dell'utente sia "1234", scrivete un programma che chiede all'utente di digitare il PIN, consentendo al massimo tre tentativi e agendo in questo modo:

- se l'utente inserisce il numero corretto, visualizzate il messaggio "Your PIN is correct" e terminate il programma;
- se l'utente inserisce un numero sbagliato, visualizzate il messaggio "Your PIN is incorrect" e, se avete chiesto il PIN meno di tre volte, chiedetelo di nuovo;
- se l'utente inserisce un numero sbagliato per tre volte, visualizzate il messaggio "Your bank card is blocked" e terminate il programma.

★ **P5.14 (economia).** Calcolare la mancia da lasciare al ristorante non è difficile, ma il ristorante di vostra proprietà vuole suggerire una mancia basata sulla qualità del servizio ricevuto dai clienti. Scrivete un programma che calcoli la mancia sulla base della soddisfazione del cliente, in questo modo:

- chiede al cliente di descrivere il proprio livello di soddisfazione, usando queste valutazioni: 1 = completamente soddisfatto; 2 = soddisfatto; 3 = insoddisfatto;
- se il cliente è totalmente soddisfatto, calcola una mancia pari al 20%;
- se il cliente è soddisfatto, calcola una mancia pari al 15%;
- se il cliente è insoddisfatto, calcola una mancia pari al 10%;
- visualizza il livello di soddisfazione del cliente e l'importo della mancia suggerita, in dollari e centesimi.

★ **P5.15 (scienze).** Scrivete un programma che chieda all'utente il valore di una lunghezza d'onda e visualizzi una descrizione della porzione dello spettro elettromagnetico a cui appartiene, coerentemente con la tabella seguente.

Spettro elettromagnetico

Descrizione	Lunghezza d'onda (m)	Frequenza (Hz)
Onde radio	$> 10^{-1}$	$< 3 \times 10^{9}$
Microonde	$10^{-3} \div 10^{-1}$	$3 \times 10^{9} \div 3 \times 10^{11}$
Infrarosso	$7 \times 10^{-7} \div 10^{-3}$	$3 \times 10^{11} \div 4 \times 10^{14}$
Luce visibile	$4 \times 10^{-7} \div 7 \times 10^{-7}$	$4 \times 10^{14} \div 7.5 \times 10^{14}$
Ultravioletto	$10^{-8} \div 4 \times 10^{-7}$	$7.5 \times 10^{14} \div 3 \times 10^{16}$
Raggi X	$10^{-11} \div 10^{-8}$	$3 \times 10^{16} \div 3 \times 10^{19}$
Raggi gamma	$< 10^{-11}$	$> 3 \times 10^{19}$

★ **P5.16 (scienze).** Ripetete l'esercizio precedente, modificando il programma in modo che venga chiesto all'utente un valore di frequenza invece che di lunghezza d'onda.

★★ **P5.17 (scienze).** Ripetete l'Esercizio P5.14, modificando il programma in modo che per prima cosa chieda all'utente se vuole fornire un valore di lunghezza d'onda o di frequenza.

★★★ **P5.18 (scienze).** Un'autovettura di tipo minivan ha due porte scorrevoli, ciascuna delle quali può essere aperta da un interruttore posto sul cruscotto oppure da una delle proprie maniglie (una interna e una esterna). Le maniglie interne, però, non funzionano quando viene attivato l'apposito blocco per la sicurezza dei bambini. Perché le porte si possano aprire, la leva del cambio (automatico) deve essere in posizione di stazionamento *e* l'interruttore generale di sblocco deve essere attivo (l'autore di questo libro è da molto tempo un sofferente proprietario di uno di questi veicoli).

Il vostro compito è quello di simulare una parte del software di controllo del veicolo. I dati in ingresso sono costituiti da una sequenza di valori che descrivono lo stato degli interruttori e la posizione della leva del cambio, in questo ordine:

- interruttori sul cruscotto per la porta sinistra e destra, blocco bambini e interruttore generale di sblocco (0 significa spento e 1 significa acceso);
- maniglie interna e esterna delle porte sinistra e destra (0 = maniglia in chiusura, 1 = maniglia in apertura);
- posizione della leva del cambio, che può essere uno dei caratteri P, N, D, 1, 2, 3, R; la posizione di stazionamento è rappresentata dal carattere P (*park*).

Un esempio di valori in ingresso può essere: `0 0 0 1 0 1 0 0 P`. Il programma deve visualizzare il messaggio appropriato: "left door opens" (la porta sinistra si apre), "right door opens" (la porta destra si apre), "both doors stay closed" (entrambe le porte rimangono chiuse).

★ **P5.19 (scienze).** Il livello sonoro L misurato in decibel (dB) è determinato dalla formula seguente:

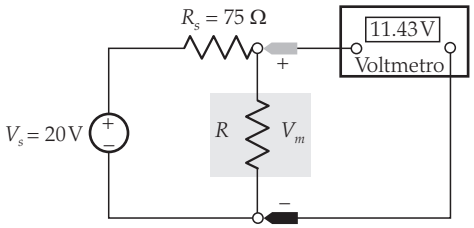
$$L = 20 \log_{10}(p/p_0)$$

dove p è la pressione sonora (misurata in Pascal, simbolo Pa) e p_0 è una pressione sonora di riferimento pari a 20×10^{-6} Pa (quando L vale 0 dB). La tabella seguente descrive alcuni livelli sonori:

130 dB	Soglia del dolore
120 dB	Possibili danni uditivi
100 dB	Martello pneumatico a 1 m di distanza
90 dB	Strada molto trafficata a 10 m di distanza
60 dB	Conversazione normale
30 dB	Biblioteca silenziosa
0 dB	Lieve stormire di foglie

Scrivete un programma che acquisisca un valore e un'unità di misura (dB o Pa) e, poi, visualizzi la descrizione sonora più adeguata, secondo l'elenco precedente.

★★ **P5.20 (scienze).** Il circuito elettrico qui disegnato è stato progettato per misurare la temperatura del gas in un laboratorio.



Il resistore R rappresenta un sensore di temperatura presente nel laboratorio. La sua resistenza, R, misurata in Ω, è legata alla temperatura T, misurata in °C, dall'equazione:

$$R = R_0 + kT$$

In questo dispositivo ipotizziamo che sia $R_0 = 100\ \Omega$ e $k = 0.5$. Il voltmetro visualizza il valore della tensione V_m ai capi del sensore, che è legata alla temperatura T del gas dall'equazione:

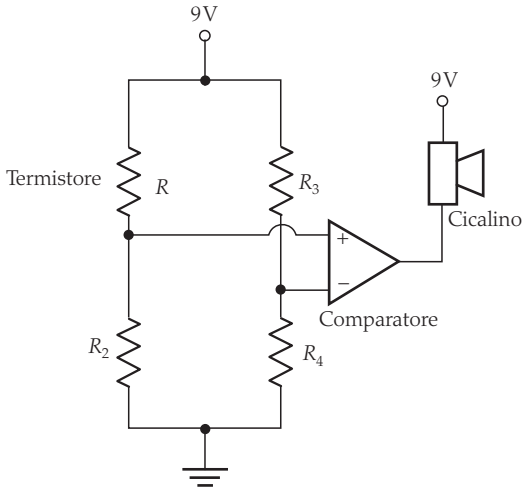
$$T = \frac{R}{k} - \frac{R_0}{k} = \frac{R_s}{k}\frac{V_m}{V_s - V_m} - \frac{R_0}{k}$$

Sapendo che la tensione misurabile dal voltmetro deve appartenere all'intervallo $V_{\text{min}} = 12\text{V} \leq V_m \leq V_{\text{max}} = 18\text{V}$, scrivete un programma che acquisisca un valore di V_m e verifichi che sia compreso tra 12 e 18. Il programma deve, poi, calcolare e visualizzare la temperatura del gas in gradi Celsius quando il valore di V_m è valido e un messaggio d'errore quando non è valido.

★★★ **P5.21 (scienze).** I danni al raccolto dovuti al gelo costituiscono uno dei tanti rischi con cui si confrontano gli agricoltori. La figura mostra un semplice circuito di allarme progettato per segnalare la presenza di ghiaccio: usa un dispositivo, detto termistore, che fa suonare un cicalino quando la temperatura scende al di sotto della soglia di congelamento dell'acqua. I termistori sono dispositivi a semiconduttore caratterizzati da una resistenza dipendente dalla temperatura, descritta dall'equazione seguente:

$$R = R_0 e^{\beta\left(\frac{1}{T} - \frac{1}{T_0}\right)}$$

dove R è la resistenza (misurata in Ω) alla temperatura T (misurata in °K) e R_0 è la resistenza (in Ω) alla temperatura T_0 (in °K); β è una costante che dipende dal materiale usato per costruire il termistore.



Il circuito è progettato in modo che l'allarme suoni quando:

$$\frac{R_2}{R + R_2} < \frac{R_4}{R_3 + R_4}$$

Il termistore utilizzato nel circuito di allarme ha R_0 = 33192 Ω a T_0 = 40 °C, con β = 3310 °K (si noti che β è misurato in °K e che un valore di temperatura in °K si ottiene aggiungendo 273° alla temperatura misurata in °C). I resistori $R2$, $R3$ e $R4$ hanno una resistenza di 156.3 kΩ = 156300 Ω.

Scrivete un programma Java che chieda all'utente un valore di temperatura in °F (gradi Fahrenheit) e visualizzi un messaggio che dica se, a quella temperatura, l'allarme suona oppure no.

★ **P5.22 (scienze).** Un oggetto avente massa m = 2 kg è attaccato a un estremo di una fune di lunghezza r = 3 m e viene fatto roteare ad alta velocità. La fune può resistere a una tensione massima T = 60 N. Scrivete un programma che acquisisca la velocità di rotazione v e determini se tale velocità provoca la rottura della fune (*suggerimento*: $T = m\ v^2/r$).

★ **P5.23 (scienze).** Un oggetto avente massa m è attaccato a un estremo di una fune di lunghezza r = 3 m e può essere fatto roteare a una velocità pari a 1, 10, 20 o 40 m/s. La fune può resistere a una tensione massima T = 60 N. Scrivete un programma che acquisisca il valore della massa m (in kg) e determini la massima velocità a cui può essere fatto roteare l'oggetto (tra le velocità elencate) senza rompere la fune (*suggerimento*: $T = m\ v^2/r$).

★★ **P5.24 (scienze).** Una persona di peso medio può saltare verso l'alto sul suolo terrestre a una velocità di 7 mph (*miles per hour*, miglia/ora) senza rischiare di abbandonare il pianeta. Ma se un astronauta salta con la stessa velocità mentre si trova sulla cometa di Halley, tornerà su di essa? Scrivete un programma che consenta all'utente di fornire un valore per la velocità di salto (in mph) a partire dalla superficie della cometa di Halley, per poi determinare se il saltatore tornerà sulla

sua superficie. In caso negativo, il programma deve calcolare quanto maggiore dovrebbe essere la massa della cometa per fare in modo che il saltatore torni sulla sua superficie.

Suggerimento. La velocità di fuga da un corpo celeste è $v_{escape} = (2GM/R)^{1/2}$, dove $G = 6.67 \times 10^{-11}$ N m^2/kg^2 è la costante di gravitazione, M è la massa del corpo celeste e R è il suo raggio. La cometa di Halley ha una massa di 1.3×10^{22} kg e un diametro di 1.153×10^6 m.

Capitolo 6

** **P6.1.** Migliorate il programma presentato nella sezione Esempi completi 6.1 in modo da verificare che il numero della carta di credito sia valido, cosa che è vera quando la procedura descritta nel seguito genera un numero divisibile per 10.

Sommate tutte le cifre e aggiungete a tale somma la seconda cifra a partire da destra e, poi, tutte le cifre alternativamente, una sì e una no, procedendo verso sinistra. Infine, tra le cifre appena descritte contate quelle il cui valore è maggiore di quattro e aggiungete alla somma tale conteggio. Il risultato deve essere divisibile per 10.

Considerate, come esempio, il numero **4**012 **8**888 **8**888 **1**881. La somma di tutte le sue cifre è 89. La somma delle cifre coinvolte nella seconda fase, che sono quelle in grassetto, è 46; tra queste, cinque sono maggiori del valore quattro, quindi il risultato è 89 + 46 + 5 = 140, che è divisibile per 10, quindi il numero della carta di credito è valido.

** **P6.2.** I *numeri di Fibonacci* sono definiti da questa sequenza:

$$f_1 = 1$$
$$f_2 = 1$$
$$f_n = f_{n-1} + f_{n-2}$$

che si può riformulare in questo modo:

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

Fatto questo, eliminate `fold2`, che non serve più, e assegnate `fold1` a `fold2`, e `fnew` a `fold1`; quindi, ripetete quante volte volete.

Realizzate un programma che chieda all'utente un numero intero n e visualizzi l'n-esimo numero di Fibonacci, usando l'algoritmo appena visto.

*** **P6.3.** *Fattorizzazione di numeri interi*. Scrivete un programma che chieda all'utente un numero intero e ne visualizzi tutti i fattori. Se, ad esempio, l'utente fornisce il numero 150, il programma deve visualizzare:

```
2
3
5
5
```

Progettate la classe `FactorGenerator` dotata del costruttore `FactorGenerator(int numberToFactor)`, che riceve come parametro il numero da scomporre in fattori, e dei metodi `nextFactor` (che restituisce il successivo fattore) e `hasMoreFactors` (che restituisce `true` se e solo se esistono altri fattori da restituire). Progettate la classe `FactorPrinter`, il cui metodo `main` chieda un numero all'utente, costruisca un oggetto di tipo `FactorGenerator` e visualizzi i fattori del numero in questione.

★★★ **P6.4.** *Numeri primi*. Scrivete un programma che chieda all'utente un numero intero e, poi, visualizzi in ordine crescente tutti i numeri primi fino a tale numero. Se, ad esempio, l'utente fornisce il numero 20, il programma deve visualizzare:

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

Ricordate che un numero è primo se non è divisibile per nessun altro numero, eccetto 1 e se stesso.

Progettate la classe `PrimeGenerator` dotata dei metodi `nextPrime` (che restituisce il successivo numero primo) e `hasMorePrimes` (che restituisce `true` se e solo se esistono altri numeri primi da restituire). Progettate la classe `PrimePrinter`, il cui metodo `main` chieda un numero all'utente, costruisca un oggetto di tipo `PrimeGenerator` e visualizzi i numeri primi fino al numero in questione.

★★★ **P6.5.** *Il gioco di Nim*. Si tratta di un gioco molto noto, con un certo numero di varianti: quella qui descritta ha una strategia vincente davvero interessante. Due giocatori prelevano alternativamente biglie da un mucchietto. Ad ogni mossa il giocatore di turno sceglie quante biglie prendere: almeno una e al massimo metà delle biglie disponibili. Poi è il turno dell'altro giocatore. Il giocatore che prende l'ultima biglia perde la partita.

Scrivete un programma che consenta all'utente di giocare contro il computer. Generate un numero intero compreso tra 10 e 100 e usatelo come dimensione iniziale del mucchietto di biglie. Generate un numero intero, 0 o 1, e utilizzatelo per decidere se sarà l'utente o il computer a giocare per primo. Generate un altro numero intero, 0 o 1, e usatelo per decidere se il computer giocherà in modo *intelligente* o *stupido*: giocando in modo stupido, ad ogni sua mossa il computer semplicemente preleva dal mucchietto un numero di biglie casuale (ma valido, cioè compreso tra 1 e $n/2$, se nel mucchietto sono rimaste n biglie); in modalità intelligente, invece, preleva un numero di biglie tale che il numero di quelle che rimangono nel mucchio sia una potenza di due diminuita di un'unità, cioè 3, 7, 15, 31 o 63. Quest'ultima è sempre una mossa valida, tranne quando la dimensione del mucchio è proprio uguale a una potenza di due diminuita di un'unità: in tal caso il computer fa una mossa scelta a caso (ovviamente tra quelle valide).

Come potrete verificare sperimentalmente, il computer non può essere battuto quando gioca in modalità intelligente e fa la prima mossa, a meno che la dimensione iniziale del mucchio non sia 15, 31 o 63. Analogamente, un giocatore umano che faccia la prima mossa e conosca la strategia qui descritta è in grado di battere il calcolatore.

★★ **P6.6.** *La passeggiata dell'ubriaco*. Trovandosi in una griglia ortogonale di strade, un ubriaco sceglie a caso una delle quattro direzioni e procede barcollando fino all'incrocio successivo, dove sceglie di nuovo una direzione a caso, e così via. Potreste pensare che, in media, l'ubriaco non vada molto lontano, dal momento che le scelte si annullano a vicenda, ma non è così.

Rappresentate le posizioni come coppie di numeri interi, (x, y), e scrivete un programma che faccia barcollare l'ubriaco per 100 incroci, a partire dalla posizione $(0, 0)$, visualizzando la posizione finale raggiunta.

★ **P6.7.** Usando la formula seguente:

$$r_{\text{new}} = (a \cdot r_{\text{old}} + b) \ \% \ m$$

e, poi, assegnando r_{new} a r_{old}, si ottiene un semplice generatore di numeri casuali. Scegliendo $m = 2^{32}$, si può semplificare la formula in questo modo:

$$r_{new} = a \cdot r_{old} + b$$

perché, usando il tipo di dato `int`, il troncamento di un risultato che provochi un *overflow* è equivalente al calcolo del resto di una divisione.

Scrivete un programma che chieda all'utente di fornire il valore iniziale per r_{old} (valore che viene spesso chiamato "seme", *seed*), poi visualizzi i primi 100 numeri interi generati dalla formula, usando $a = 32310901$ e $b = 1729$.

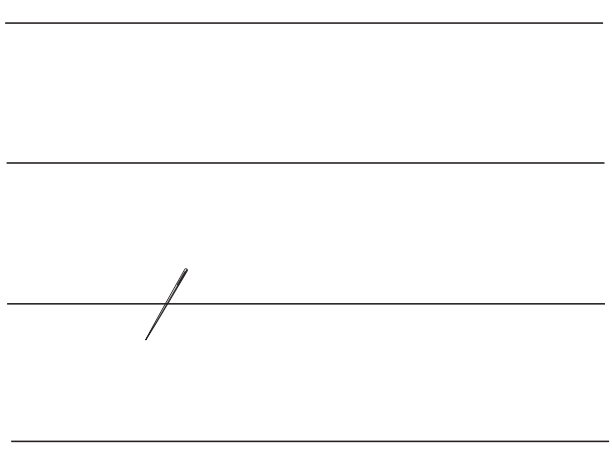
★★ **P6.8.** *L'esperimento dell'ago di Buffon*. L'esperimento seguente fu ideato da Comte Georges-Louis Leclerc de Buffon (1707-1788), un naturalista francese. Un ago lungo un pollice viene lasciato cadere su un foglio, sul quale sono state tracciate linee parallele equidistanti, una ogni 2 pollici. Se l'ago cade su una linea, contiamo un "bersaglio colpito" (*hit*, si veda la figura). Buffon scoprì che il rapporto tra "tentativi" (*tries*) e bersagli colpiti, cioè *tries/hits*, è circa uguale a π.

Per l'esperimento dell'ago di Buffon dovete generare due numeri casuali: uno descrive la posizione iniziale lungo l'asse y e l'altro descrive l'angolo tra l'ago e l'asse x. Poi, bisogna verificare se l'ago tocca una linea.

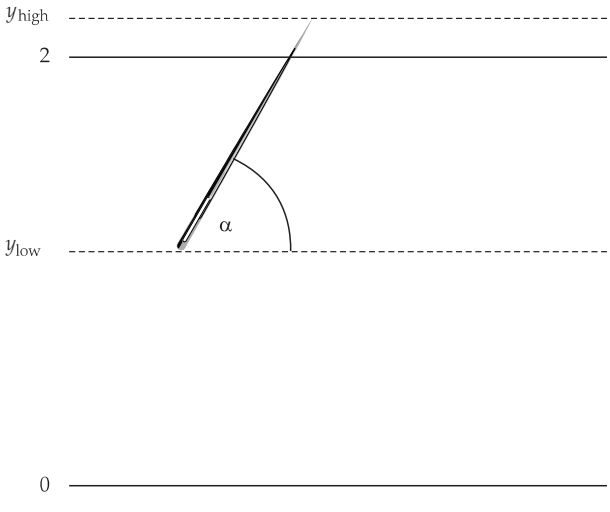
Generate la posizione del punto *più basso* dell'ago. La sua coordinata x è ininfluente, mentre la sua coordinata y, indicata come y_{low}, può essere un numero casuale compreso tra 0 e 2. L'angolo α tra l'ago e l'asse x può essere qualsiasi valore compreso tra 0 gradi e 180 gradi (cioè π radianti). La coordinata y del punto *più alto* dell'ago è:

$$y_{high} = y_{low} + \sin \alpha$$

Un lancio è un "bersaglio colpito" se y_{high} vale almeno 2, come si può vedere nella figura. Fermatevi dopo 10000 tentativi e visualizzate il rapporto *tries/hits* (questo programma, però, non è adatto a calcolare il valore di π, perché vi serve proprio il valore di π per il calcolo dell'angolo).



L'esperimento dell'ago di Buffon



Un "bersaglio colpito" nell'esperimento dell'ago di Buffon

** **P6.9.** Nel diciassettesimo secolo nacque la teoria delle probabilità, quando un giocatore d'azzardo chiese a un amico matematico di spiegargli alcune osservazioni sperimentali relative al gioco dei dadi. Perché si trovava mediamente a vincere quando scommetteva che, lanciando un dado quattro volte, sarebbe comparso almeno un sei? Perché, invece, si trovava mediamente a perdere quando, in un caso apparentemente simile, scommetteva che, lanciando una coppia di dadi ventiquattro volte, sarebbe comparso almeno un doppio sei?

Ai giorni nostri sarebbe assurdo mettersi a lanciare 24 volte di fila una coppia di dadi, per poi ripetere l'esperimento più volte: è meglio fare tale esperimento con il calcolatore. Simulate ciascuna scommessa un milione di volte, visualizzando le vincite e le perdite, nell'ipotesi che ciascuna scommessa sia di un dollaro.

** **P6.10.** *Media e deviazione standard*. Scrivete un programma che legga un insieme di valori in virgola mobile. Adottate il meccanismo più adeguato per consentire all'utente di segnalare la fine dei dati. Quando tutti i valori sono stati acquisiti, visualizzate il numero di valori, il loro valore medio e la loro deviazione standard. Il valore medio dell'insieme di dati $\{x_1, \ldots, x_n\}$ è

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

dove $\Sigma x_i = x_1 + \ldots + x_n$ è la somma dei valori acquisiti. La deviazione standard è così definita:

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

Questa formula, però, non è adatta alla nostra elaborazione, perché, dopo che il programma ha calcolato il valore medio, i singoli valori non sono più disponibili. Fino a quando non avrete imparato a memorizzare questi valori acquisiti, usate la formula seguente, anche se numericamente è meno stabile:

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

Con quest'ultima formula la deviazione standard viene calcolata aggiornando e memorizzando, via via che i valori vengono acquisiti, il numero di valori, la loro somma e la somma dei loro quadrati.

Il programma deve usare un'apposita classe `DataSet` per rappresentare l'insieme dei dati acquisiti, dotata del metodo

```
public void add(double value)
```

per aggiungere un valore all'insieme, oltre ai metodi `getAverage` e `getStandardDeviation`, che risolvono, nell'ordine, il secondo e il terzo dei problemi posti.

*** **P6.11.** Un lucchetto per bicicletta a combinazione numerica ha quattro anelli (*ring*), ciascuno avente i numeri da 0 a 9. Scrivete un programma che, conoscendo i numeri su cui sono attualmente posizionati gli anelli e la combinazione di sblocco, visualizzi le istruzioni necessarie a sbloccare il lucchetto facendo il numero minimo di rotazioni (*twist*). Una "rotazione verso l'alto" (*twist up*) aumenta di un'unità il valore presente nell'anello su cui agisce, mentre una "rotazione verso il basso" (*twist down*) lo diminuisce. Ad esempio, se gli anelli fossero impostati al valore 1729 e la combinazione corretta fosse 5714, le istruzioni di sblocco dovrebbe essere queste:

```
Ring 1: Twist up 4 times
Ring 3: Twist down once
Ring 4: Twist up or down 5 times
```

Ricordate che *once* significa "una volta". Nell'ultimo caso l'istruzione afferma "alto o basso", perché, dovendo fare 5 rotazioni, il senso di rotazione è indifferente (potete, però, anche decidere che in tal caso la rotazione sia sempre "verso l'alto", oppure sempre "verso il basso").

** **P6.12 (economia).** Un'azienda possiede azioni che vorrebbe vendere quando il prezzo supera un determinato valore di soglia. Scrivete un programma che legga il valore di soglia e, poi, legga il prezzo attuale finché non diventa almeno uguale al valore di soglia (il programma deve, quindi, acquisire una sequenza di valori di tipo `double` forniti in ingresso, usando un oggetto di tipo `Scanner`): a quel punto, il programma deve visualizzare un messaggio che segnali il superamento del valore di soglia che innesca la vendita.

** **P6.13 (economia).** Scrivete un'applicazione che gestisca la prevendita di un numero limitato si biglietti del cinema. Ogni acquirente può comprare al massimo 4 biglietti e non ne possono essere venduti più di 100. Realizzate il programma `TicketSeller` che chieda all'utente quanti biglietti intende acquistare, per poi visualizzare il numero di biglietti rimasti. L'operazione va ripetuta fino all'esaurimento dei biglietti, visualizzando al termine il numero di acquirenti.

** **P6.14 (economia).** Dovete tenere sotto controllo il numero di persone che possono essere presenti contemporaneamente all'interno di un *oyster bar*. Chiunque può uscire dal bar in qualsiasi momento, ma un gruppo di persone non vi può entrare se, insieme alle persone già presenti nel bar, porta il numero di presenze a superare il limite di 100. Scrivete un programma che legga, in sequenza, le dimensioni dei gruppi di persone che entrano ed escono dal bar, usando numeri negativi per indicare un'uscita. Dopo aver letto ciascun valore, il programma deve visualizzare il numero di persone presenti nel bar in quel momento. Se il bar arriva ad avere al proprio interno il massimo numero di clienti, il programma termina visualizzando un messaggio che segnali la cosa.

** **P6.15 (scienze).** In una simulazione predatore-preda, si calcolano le popolazioni di predatori (*predator*) e prede (*prey*) usando le equazioni seguenti:

$$prey_{n+1} = prey_n \times (1 + A - B \times pred_n)$$
$$pred_{n+1} = pred_n \times (1 - C + D \times prey_n)$$

In queste equazioni, A è il ritmo con cui le nascite di prede sovrastano le loro morti naturali, B è il ritmo di predazione, C è il ritmo con cui le morti di predatori sovrastano le nascite in caso di assenza di cibo e D è il ritmo con cui aumentano i predatori in presenza di cibo.

Scrivete un programma che chieda questi valori all'utente, oltre alle dimensioni iniziali delle popolazioni e al numero di periodi di tempo coinvolti dalla simulazione. Successivamente, il programma deve visualizzare la dimensione delle due popolazioni per il numero di periodi di tempo assegnato. Eseguite, come esempio, una simulazione con $A = 0.1$, $B = C = 0.01$ e $D = 0.00002$, con popolazione iniziale di 1000 prede e 20 predatori.

** **P6.16 (scienze).** *Traiettoria di un proiettile*. Supponete che un palla di cannone venga lanciata in aria in direzione verticale con una velocità iniziale v_0. Qualunque libro di fisica vi dirà che la posizione della palla dopo t secondi è $s(t) = -(1/2)gt^2 + v_0t$, dove $g = 9.81$ m/s^2 è l'accelerazione gravitazionale terrestre, ma nessuno di quei libri vi spiegherà il motivo per cui qualcuno dovrebbe voler fare un esperimento così pericoloso, per cui lo faremo, in tutta tranquillità, con una simulazione al calcolatore.

In pratica, confermeremo mediante una simulazione la correttezza della formula che troviamo sui libri. Nella nostra simulazione considereremo come si sposta la palla durante intervalli di tempo molto brevi, indicati con Δt: in un intervallo di tempo molto breve, la velocità v si mantiene quasi

costante e possiamo calcolare la distanza percorsa dalla palla con la formula $\Delta s = v\Delta t$. Nel nostro programma useremo semplicemente il valore:

```
final double DELTA_T = 0.01;
```

e aggiorneremo la posizione in questo modo:

```
s = s + v * DELTA_T;
```

La velocità si modifica di continuo: viene ridotta dalla forza gravitazionale terrestre. In un breve intervallo di tempo, $\Delta v = -g\Delta t$, quindi possiamo aggiornare la velocità in questo modo:

```
v = v - g * DELTA_T;
```

Nell'iterazione successiva, relativa all'intervallo di tempo seguente, la posizione verrà aggiornata usando questa nuova velocità.

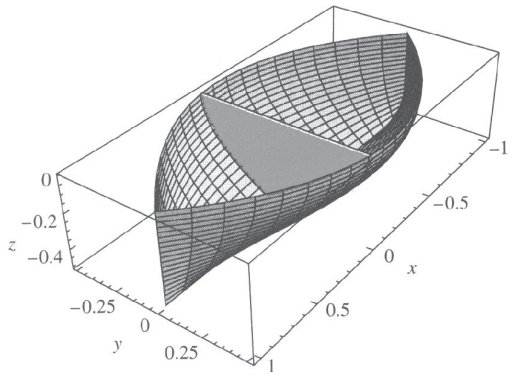
A questo punto non ci resta che eseguire la simulazione fino a quando la palla non ricade sulla superficie terrestre. Acquisiamo la velocità iniziale come dato di ingresso (un buon valore potrebbe essere 100 m/s) e aggiorniamo la posizione e la velocità 100 volte al secondo (`DELTA_T = 0.01`), ma visualizziamo la posizione soltanto una volta al secondo. Per confronto, visualizziamo anche i valori calcolati usando la formula esatta, $s(t) = -(1/2)gt^2 + v_0 t$.

Nota. Probabilmente vi starete chiedendo che vantaggio ci sia nel fare questa simulazione, dato che disponiamo di una formula esatta. Bene, la formula che troviamo nei libri di fisica *non* è esatta: in realtà, la forza gravitazionale diminuisce all'aumentare della distanza dalla superficie terrestre. Questo fenomeno complica sufficientemente il problema da rendere impossibile l'individuazione di una formula esatta che descriva il moto effettivo della palla, mentre la simulazione al calcolatore può facilmente essere estesa al caso in cui l'accelerazione gravitazionale venga considerata variabile. Per le palle di cannone la formula che troviamo nei libri di fisica è sufficientemente accurata, ma per calcolare con precisione le traiettorie di oggetti volanti a un'altitudine più elevata, come i missili balistici, servono simulazioni al calcolatore.

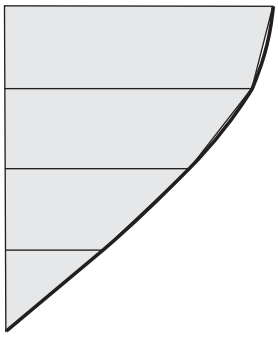
★★★ **P6.17 (scienze).** Come modello semplificato dello scafo di una nave si può usare questa formula:

$$|y| = \frac{B}{2}\left[1 - \left(\frac{2x}{L}\right)^2\right]\left[1 - \left(\frac{z}{T}\right)^2\right]$$

dove B è la larghezza (*beam*), L è la lunghezza (*length*) e T è l'altezza (*draft*). Si noti che esistono due valori di y per ogni coppia di valori di x e z, perché lo scafo è simmetrico rispetto all'asse y.



In termini nautici, l'area di una sezione trasversale dello scafo in un punto x è detta semplicemente "sezione". Per calcolarla, facciamo variare z da 0 a $-T$ in n passi incrementali, ciascuno di dimensione T/n: per ogni valore di z, calcoliamo il valore di y, poi sommiamo le aree delle strisce trapezoidali che compongono la sezione (nella figura sono rappresentate le strisce corrispondenti al caso $n = 4$).

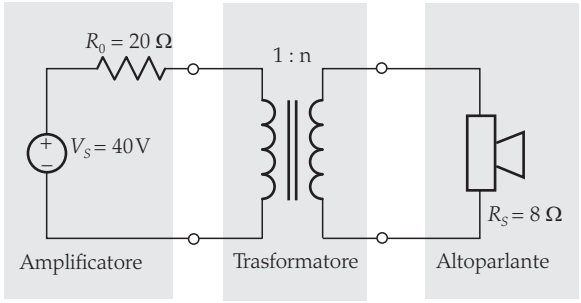


Scrivete un programma che acquisisca in ingresso i valori di B, L, T, x e n, per poi visualizzare l'area della sezione trasversale nel punto x.

★ **P6.18 (scienze).** Il decadimento dei materiali radioattivi può essere modellato con l'equazione $A = A_0 e^{-t(\log 2/h)}$, dove A è la quantità di materiale al tempo t, A_0 è la quantità di materiale al tempo 0 e h è il cosiddetto "tempo di dimezzamento" o "emivita".

Il Tecnezio-99 è un radioisotopo che viene usato nella tomografia cerebrale e ha un tempo di dimezzamento di 6 ore. Il vostro programma deve visualizzare la percentuale di radioisotopo, A/A_0, rimasto nel corpo di un paziente, ora dopo ora per 24 ore, dopo aver ricevuto la dose A_0.

★★★ **P6.19 (scienze).** I trasformatori sono dispositivi elettrici che vengono spesso costruiti avvolgendo bobine di filo attorno a un nucleo di ferrite. La figura illustra una situazione che si verifica in vari dispositivi acustici, come i telefoni cellulari e i riproduttori musicali: in questo circuito viene usato un trasformatore per collegare un altoparlante (*speaker*) all'uscita di un amplificatore acustico.



Il simbolo usato per rappresentare il trasformatore nel circuito suggerisce la presenza di due bobine di filo o "avvolgimenti". Il parametro n che caratterizza il trasformatore viene detto "rapporto di spira" (*turn ratio*) ed è, letteralmente, il rapporto tra il numero di spire che costituiscono i due avvolgimenti.

Quando si progetta un circuito di questo tipo, la cosa che interessa di più è il valore della potenza che viene fornita all'altoparlante, potenza che, poi, farà in modo che l'altoparlante produca i suoni. Se collegassimo l'altoparlante direttamente all'amplificatore, senza usare il trasformatore,

soltanto una frazione della potenza disponibile nell'amplificatore raggiungerebbe l'altoparlante: il resto di tale potenza verrebbe dissipato nell'amplificatore stesso. Il trasformatore viene aggiunto per aumentare la frazione della potenza che l'amplificatore trasferisce all'altoparlante.

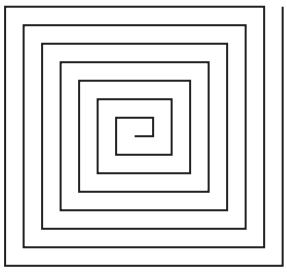
La potenza, P_S, trasferita all'altoparlante viene calcolata usando questa formula:

$$P_S = R_S \left(\frac{nV_S}{n^2 R_0 + R_S} \right)^2$$

Scrivete un programma che costituisca un modello per il circuito in esame e consenta di variarne il rapporto di spira da 0.01 a 2, con incrementi di 0.01, al fine di individuare il valore del rapporto che rende massima la potenza trasferita all'altoparlante.

* **P6.20 (grafica).** Scrivete un'applicazione grafica che visualizzi una scacchiera con 64 caselle, alternativamente bianche e nere.

*** **P6.21 (grafica).** Scrivete un'applicazione grafica che disegni una spirale quadrata, come questa:



** **P6.22 (grafica).** Usando la libreria grafica di Java, disegnare grafici di funzioni è facile e divertente: semplicemente, disegnate 100 segmenti che colleghino i punti $(x, f(x))$ e $(x + d, f(x + d))$, con x che varia da x_{min} a x_{max} e $d = (x_{max} - x_{min})/100$.

Usate questo metodo per tracciare il grafico della funzione $f(x) = 0.00005x^3 - 0.03x^2 + 4x + 200$, con x che varia da 0 a 400.

*** **P6.23 (grafica).** Disegnate la figura della "rosa con quattro petali", la cui equazione in coordinate polari è $r = \cos(2\theta)$. Fate variare θ da 0 a 2π con 100 incrementi. Ogni volta, calcolate r e ottenete le coordinate cartesiane (x, y) a partire dalle coordinate polari usando queste formule:

$$x = r \cdot \cos(\theta), y = r \cdot \sin(\theta)$$

Capitolo 7

** **P7.1.** Uno smartphone memorizza la propria posizione, sotto forma di coordinate x e y, una volta ogni minuto. Usate tali dati per risolvere un fastidioso problema: dove avete parcheggiato l'automobile? Cercate, nei dati, le posizioni corrispondenti a un veicolo fermo, seguite da quelle di una persona che cammina fuori dall'auto.

** **P7.2.** Una *serie* ("run") è definita come una sequenza di valori adiacenti ripetuti. Scrivete un programma che generi una sequenza di 20 lanci casuali di un dado, memorizzando i risultati in un array, per poi visualizzare i valori ottenuti, identificando le ripetizioni e 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:

```
Assegna false alla variabile booleana inRun.
Per ogni indice i valido nell'array values
   Se inRun è true
      Se values[i] è diverso dal valore precedente
         Visualizza ).
         inRun = false.
   Se inRun è false
      Se values[i] è uguale al valore seguente
         Visualizza (.
         inRun = true.
   Visualizza values[i].
Se inRun è true, visualizza ).
```

** **P7.3.** Scrivete un programma che generi una sequenza di 20 lanci casuali di un dado, memorizzando i risultati in un array, per poi visualizzare 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.

** **P7.4.** È scientificamente accertato che, in un bagno pubblico, gli uomini generalmente preferiscono rendere massima la distanza tra sé e le postazioni già occupate, occupando quella che si trova al centro della più lunga sequenza di postazioni libere. Ad esempio, esaminiamo questa situazione, in cui sono presenti dieci postazioni, tutte libere:

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

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 postazioni disponibili e visualizzi diagrammi nel formato appena descritto, uno per volta, man mano che le postazioni vengono occupate. *Suggerimento*: usate un array di valori `boolean` per indicare se una postazione è occupata oppure no.

*** **P7.5.** 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 mucchietti di dimensione casuale: potreste, ad esempio, iniziare con mucchietti di dimensioni pari a 20, 5, 1, 9 e 10. Ad ogni turno prendete una carta da ciascun mucchietto, formando con queste un nuovo mucchietto: la configurazione iniziale che abbiamo usato prima come esempio verrebbe trasformata in un insieme di mucchietti 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 del solitario, uno dopo l'altro, visualizzando la situazione dopo ogni turno. Fermatevi quando è stata raggiunta la configurazione finale del solitario.

★★★ **P7.6.** *Quadrati magici*. Una matrice $n \times n$ riempita con i numeri $1, 2, 3, \ldots, n^2$ 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 16 valori dalla tastiera e verifichi se, disposti in una matrice 4 × 4, formano un quadrato magico.

Dovete verificare due caratteristiche:

- Nei dati inseriti dall'utente sono presenti tutti i numeri 1, 2, …, 16?
- 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?

★★★ **P7.7.** Implementate il seguente algoritmo per costruire quadrati magici $n \times n$, che funziona solo se n è dispari.

```
riga = n – 1.
colonna = n / 2.
Ripeti per k = 1, 2, . . ., n * n
   Scrivi k nella posizione [riga][colonna].
   Incrementa riga e colonna.
   Se riga == n, allora riga = 0.
   Se colonna == n, allora colonna = 0.
   Se la posizione [riga][colonna] contiene già un valore
      Ripristina riga e colonna ai loro valori precedenti.
      Decrementa riga.
```

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

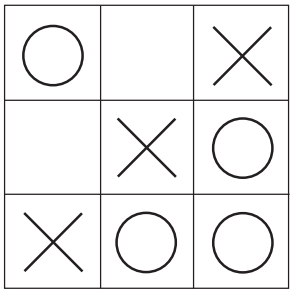
Scrivete un programma il cui dato in ingresso sia il numero n e il cui prodotto in uscita sia il quadrato magico di ordine n, se n è dispari.

★★ **P7.8.** Lo schema dei posti a teatro è realizzato mediante una tabella che indica i prezzi dei biglietti per ciascun posto, come questa:

```
10 10 10 10 10 10 10 10 10 10
10 10 20 20 20 20 20 20 10 10
10 10 20 20 20 20 20 20 10 10
10 10 20 20 20 20 20 20 10 10
20 20 30 30 40 40 30 30 20 20
20 30 30 40 50 50 40 30 30 20
30 40 50 50 50 50 50 50 40 30
```

Scrivete un programma che chieda all'utente di scegliere un posto o un prezzo. Contrassegnate con un prezzo uguale a 0 i posti già venduti. Quando l'utente specifica un posto, accertatevi che sia libero. Quando, invece, specifica un prezzo, assegnategli un posto qualsiasi tra quelli disponibili a quel prezzo.

★★★ **P7.9.** Scrivete un programma che giochi a tic-tac-toe (in italiano, *tris* o *schiera*). Il tic-tac-toe si gioca su una scacchiera 3 × 3 con due giocatori che, a turno, posizionano in una casella libera il proprio simbolo, scelto tra una croce e un cerchio. Vince la partita il giocatore che compone una schiera di tre propri simboli su una riga, una colonna o una diagonale. Il programma deve disegnare la scacchiera, chiedere all'utente le coordinate del suo prossimo simbolo, cambiare il giocatore di turno dopo ogni mossa e decretare il vincitore.



★★★ **P710.** 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, sulla base delle cinque carte che il giocatore ha in mano, il programma comunica il punteggio ottenuto, che deve essere il maggiore tra i seguenti, elencati in ordine crescente:

- No pair ("Niente"). La configurazione peggiore, che contiene cinque carte spaiate che non compongono alcuna delle configurazioni elencate nel seguito.
- One pair ("Coppia"). Due carte dello stesso valore, ad esempio due regine.
- Two pairs ("Doppia coppia"). Due coppie, ad esempio due regine e due cinque.
- Three of a kind ("Tris"). Tre carte dello stesso valore, ad esempio tre regine.
- Straight ("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.
- Flush ("Colore"). Cinque carte dello stesso seme, con valori non consecutivi.
- Full House ("Full"). Un tris e una coppia, ad esempio tre regine e due 5.
- Four of a kind ("Poker"). Quattro carte con lo stesso valore, ad esempio quattro regine.
- Straight Flush ("Scala colore"). Una scala e, contemporaneamente, un colore: cinque carte con valori consecutivi e dello stesso seme.
- Royal Flush ("Scala reale"). La mano migliore possibile: 10, fante, regina, re e asso, tutti del medesimo seme.

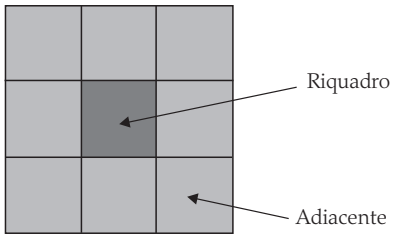
★★★ **P7.11.** *Il Gioco della Vita* (*The Game of Life*) è un gioco matematico molto conosciuto che genera un comportamento incredibilmente complesso, sebbene sia definibile con poche e semplici regole (in realtà, non è un vero e proprio gioco in senso tradizionale, perché non ha giocatori che gareggiano per vincere). Il gioco si svolge su una scacchiera rettangolare e ogni suo riquadro può essere vuoto o occupato. All'inizio deve essere possibile specificare in qualche modo quali siano i riquadri vuoti e quali siano, invece, occupati, dopodiché il gioco procede in modo autonomo. A ogni passo viene calcolata la *generazione* successiva della popolazione sulla scacchiera: si assiste a una "nascita" in un riquadro vuoto se questo ha tre riquadri adiacenti occupati; si verifica una "morte per sovraffollamento" in un riquadro occupato se questo ha quattro o più riquadri adiacenti occupati, mentre c'è una "morte per solitudine" se il riquadro occupato ha al più un solo riquadro adiacente occupato. Un riquadro adiacente può trovarsi a sinistra, a destra, sopra, sotto o in direzione diagonale. La figura mostra un riquadro e i suoi adiacenti.

Molte configurazioni, quando vengono assoggettate a queste regole, presentano un comportamento interessante. Nella figura più in basso 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.

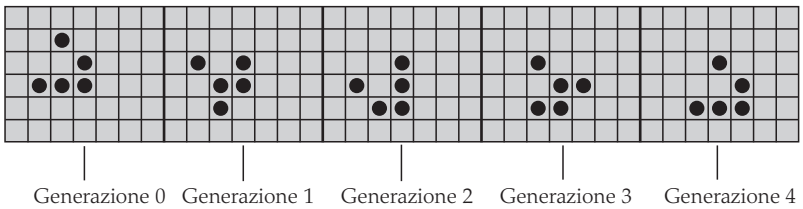
Una delle configurazioni più sorprendenti è lo spara-alianti (*glider gun*), un complesso insieme di riquadri che dopo 30 mosse ritorna alla sua forma iniziale.

Scrivete un programma che elimini l'ingrato compito del calcolo manuale delle generazioni che si susseguono, mostrandole sullo schermo. Usate un array bidimensionale per memorizzare la configurazione rettangolare e chiedete all'utente di indicare la configurazione iniziale, fornendo in ingresso una sequenza di spazi (per i riquadri vuoti) e di caratteri `o` (per i riquadri occupati).

Riquadri "adiacenti" a un riquadro



Configurazione "aliante" (*glider*)



★★ **P7.12 (economia).** Un negozio di animali domestici (*pet*) vuole fare uno sconto ai clienti che acquistano un animale (o più) e almeno altri cinque articoli. Lo sconto è il 20% del costo degli altri articoli, mentre gli animali sono esclusi.

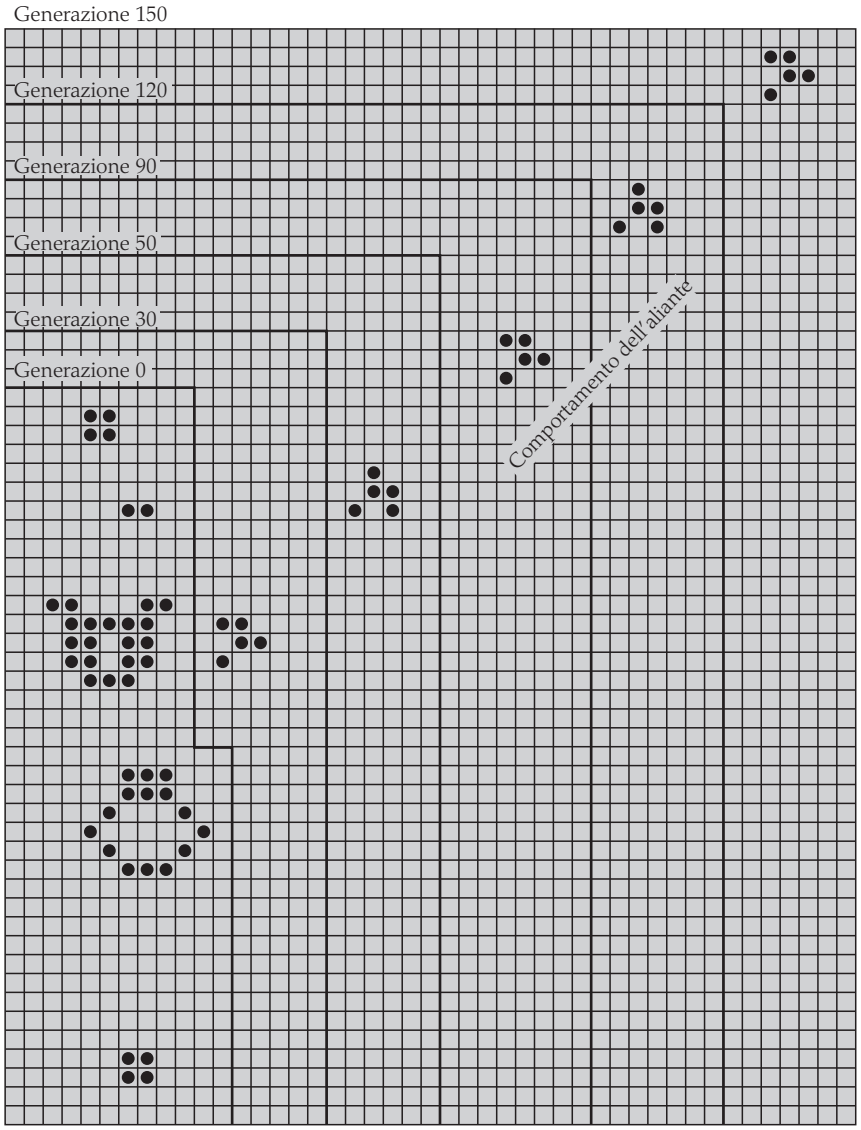
Progettate la classe `Item` che descriva un articolo in vendita, dotata di tutti i metodi necessari e del costruttore:

```
public Item(double price, boolean isPet, int quantity)
```

Una fattura (*invoice*) contiene un insieme di oggetti di tipo `Item`, memorizzati in un array o in un vettore. Nella classe `Invoice` inserite i metodi:

```
public void add(Item anItem)
public double getDiscount()
```

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



Scrivete un programma che chieda al cassiere di digitare tutti i prezzi, ciascuno seguito dalla relativa quantità acquistata e da una lettera: `Y` se si tratta di un animale, `N` per altri tipi di articoli. Il prezzo –1 svolge il ruolo di sentinella conclusiva. In un ciclo, invocate ripetutamente il metodo `add` per aggiungere alla fattura gli articoli acquistati, poi invocate il metodo `getDiscount` e visualizzate lo sconto.

** **P7.13 (economia).** Un supermercato vuole ricompensare il proprio miglior cliente (*customer*) del giorno, mostrandone il nome su uno schermo all'interno del negozio. A questo scopo gestisce un vettore di tipo `ArrayList<Customer>`. Nella classe `Store`, che rappresenta il negozio, realizzate i metodi:

```
public void addSale(String customerName, double amount)
public String nameOfBestCustomer()
```

usati per registrare ciascuno scontrino, associandone l'importo al cliente, e per conoscere il nome del miglior cliente.

Scrivete un programma che chieda al cassiere di digitare tutti gli importi spesi e i nomi dei relativi clienti, aggiungendoli a un oggetto di tipo `Store`, per poi visualizzare il nome del miglior cliente. Il prezzo 0 funge da sentinella.

*** **P7.14 (economia).** Migliorate il programma dell'esercizio precedente in modo che visualizzi un elenco dei clienti migliori, cioè i primi `topN` clienti per importo speso, dove `topN` è un valore fornito dall'utente del programma. Implementate il metodo:

```
public ArrayList<Customer> nameOfBestCustomers(int topN)
```

Se ci sono meno di `topN` clienti, faranno tutti parte dell'elenco dei migliori.

** **P7.15 (scienze).** I suoni possono essere rappresentati mediante un array di "valori campionati" che descrivono l'intensità del suono in un dato istante. Nel pacchetto dei file scaricabili per questo libro, la cartella `sound` del Capitolo 7 contiene il codice sorgente completo di un programma che legge un file audio (in formato WAV), elabora i valori campionati e mostra il risultato. Il vostro compito è quello di elaborare il suono in modo che introduca un eco: ad ogni valore campionato aggiungete il valore campionato nell'istante che lo precede di 0.2 secondi. I risultati vanno scalati proporzionalmente in modo che nessun valore sia maggiore di 32767.

*** **P7.16 (scienze).** Disponete di un array bidimensionale (`heights`) che riporta l'altezza del terreno in diversi punti di una zona quadrata. Scrivete un costruttore

```
public Terrain(double[][] heights)
```

e un metodo

```
public void printFloodMap(double waterLevel)
```

che visualizzi una mappa di inondazione, per mostrare i punti del terreno che verrebbero sommersi da un'inondazione il cui livello viene fornito come argomento.

In una tale mappa, un asterisco indica un punto sommerso e uno spazio indica un punto emerso. Ecco un esempio:

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

Poi, scrivete un programma che legga cento valori di altezza del terreno e mostri come quella zona venga sommersa quando il livello dell'acqua aumenta in dieci incrementi a partire dal punto più basso del terreno fino ad arrivare al punto più elevato.

** **P7.17 (scienze).** Spesso i valori raccolti durante un esperimento vanno "ripuliti", per togliere parte del rumore di misura. Un approccio semplice a questo problema prevede di sostituire, in un array, a ciascun valore la media tra il valore stesso e i due valori adiacenti (o un unico adiacente se il valore in esame si trova a una delle due estremità dell'array). Nella classe `Data`, avente le variabili di esemplare

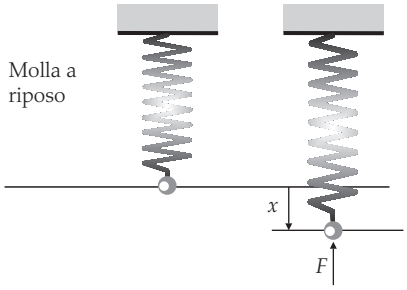
```
private double[] values;
private int valuesSize;
```

realizzate il metodo

```
public void smooth()
```

che svolga tale operazione senza creare un altro array.

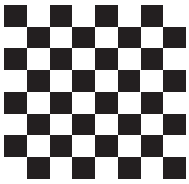
*** **P7.18 (scienze).** Scrivete un programma che simuli il movimento di un oggetto di massa m appeso a una molla oscillante. Quando una molla viene spostata di una quantità x dalla sua posizione di equilibrio, la legge di Hooke afferma che la forza di reazione è $F = -kx$, dove k è una costante che dipende dalla molla (in questa simulazione, usate il valore 10 N/m).



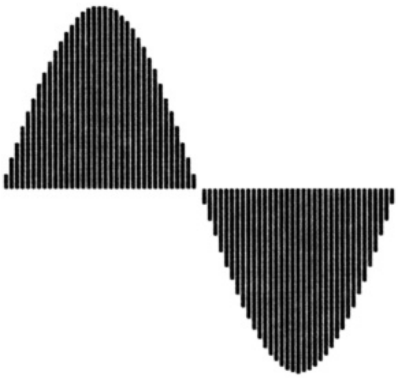
Iniziate con uno spostamento x di 0.5 metri, dopo aver azzerato la velocità iniziale, v. Calcolate l'accelerazione a usando la legge di Newton ($F = ma$) e la legge di Hooke, con una massa di 1 kg. Usate intervalli di tempo di durata $\Delta t = 0.01$ secondi e aggiornate ripetutamente la velocità, che, nell'intervallo Δt, cambia di una quantità pari a $a\Delta t$, e, conseguentemente, aggiornate lo spostamento, la cui variazione è $v\Delta t$.

Ogni dieci iterazioni disegnate lo spostamento della molla mediante una piccola barra, con 1 pixel che rappresenta 1 cm, come in questa immagine.

** **P7.19 (grafica).** Generate l'immagine di una scacchiera.



* **P7.20 (grafica).** Generate l'immagine di un'onda sinusoidale, con una riga di pixel ogni cinque gradi.



* **P7.21 (grafica).** Realizzate una classe `Cloud` ("nuvola") che contenga un vettore di oggetti di tipo `Point2D.Double`. Dotatela dei metodi:

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

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

** **P7.22 (grafica).** Realizzate una classe `Polygon` che contenga un vettore di oggetti di tipo `Point2D.Double` e dotatela dei metodi:

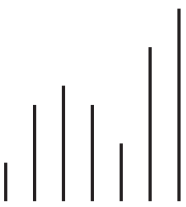
```
public void add(Point2D.Double aPoint)
public void draw(Graphics2D g2)
```

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

* **P7.23 (grafica).** Scrivete una classe `Chart` dotata dei metodi

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

che visualizzi un diagramma a segmenti con i valori che sono stati aggiunti mediante il metodo `add`, in modo simile a quanto si può qui vedere.



Potete ipotizzare che i valori rappresentino posizioni misurate in pixel.

** **P7.24 (grafica).** Scrivete una classe `BarChart` dotata dei metodi

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

che visualizzi un diagramma a barre con i valori che sono stati aggiunti mediante il metodo `add`. 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.

*** **P7.25.** Migliorate la classe `BarChart` dell'esercizio precedente in modo che funzioni correttamente anche con valori negativi.

** **P7.26.** Scrivete una classe `PieChart` dotata dei metodi

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

che visualizzi un diagramma a torte con i valori che sono stati aggiunti mediante il metodo `add`. Potete ipotizzare che tutti i valori siano positivi.

Capitolo 8

*** **P8.1.** Realizzate una classe, `ComboLock`, che simuli il funzionamento di un lucchetto a combinazione numerica con manopola rotante, come quelli solitamente usati per gli armadietti delle palestre. La combinazione è costituita da tre numeri compresi tra 0 e 39. Il metodo `reset` azzera la rotazione della manopola, in modo che punti sullo zero, mentre i metodi `turnLeft` e `turnRight` ruotano la manopola, rispettivamente verso sinistra (cioè in senso antiorario) e verso destra (cioè in senso orario), facendole compiere il numero di scatti indicato come argomento. Il metodo `open` tenta di aprire il lucchetto, che si apre soltanto se, dalla posizione iniziale, la manopola è stata fatta girare verso destra per numero complessivo di scatti uguale al primo numero della combinazione, poi a sinistra per un numero complessivo di scatti uguale al secondo numero, infine di nuovo verso destra per il terzo numero.

```
public class ComboLock
{
   . . .
   public ComboLock(int secret1, int secret2, int secret3) { . . . }
   public void reset() { . . . }
   public void turnLeft(int ticks) { . . . }
   public void turnRight(int ticks) { . . . }
   public boolean open() { . . . }
}
```

*** **P8.2.** Migliorate il programma di visualizzazione di una galleria di immagini visto nel Paragrafo 8.5 in modo che usi lo spazio in modo più efficiente. Invece di allineare tutte le immagini sul loro bordo superiore, il programma deve trovare il primo spazio disponibile in cui inserire una nuova immagine, pur rispettando le spaziature minime.

Suggerimento: iniziate risolvendo un problema più semplice, disponendo le immagini senza spazi di separazione.

Anche questo, comunque, non è facile. Bisogna verificare se c'è posto per una nuova immagine. Memorizzate in un vettore (`ArrayList`) i rettangoli che delimitano ciascuna singola immagine già posizionata e progettate il metodo

```
public static boolean intersects(Rectangle r, ArrayList<Rectangle> rectangles)
```

che verifichi se `r` si sovrappone, anche solo parzialmente, a uno dei rettangoli del vettore. Usate il metodo `intersects` della classe `Rectangle`.

Successivamente dovete scoprire dove poter cercare di posizionare una nuova immagine. Prima fate un tentativo semplice, controllando i vertici di tutti i rettangoli dell'elenco, a partire dalle coordinate y di valore minore.

Per un posizionamento migliore, controllate tutti i punti le cui coordinate x e y sono le coordinate x e y di vertici, anche se non necessariamente di uno stesso vertice.

Fatto questo, aggiungete gli spazi tra le immagini.

*** **P8.3.** In un gioco di carte in cui si cercano le coppie, le carte vengono disposte in una griglia con righe e colonne. Il giocatore scopre due carte per volta e, se sono uguali, guadagna un punto (e un punto ulteriore se le due carte sono adiacenti). Progettate un programma che consenta di giocare, usando le classi `Tile` (carta), `Location` (posizione, che incapsula l'indicazione di una riga e di una colonna), `Grid` (griglia) e `MatchingGame` (il gioco). Potete visualizzare le carte usando scritte o, come nel Paragrafo 8.5, figure. Nel progettare le classi, fate attenzione a coesione e accoppiamento.

** **P8.4.** Simulate un sistema di *car sharing* usato da lavoratori pendolari per caricare e scaricare passeggeri in apposite stazioni. Immaginate che ci siano 30 stazioni, a un miglio di distanza una dall'altra lungo un determinato percorso: per ogni stazione generate un numero casuale di automobili, ciascuna con una destinazione determinata casualmente, e un numero casuale di passeggeri, ciascuno con una destinazione ancora determinata casualmente.

Ogni guidatore carica a bordo tre passeggeri a caso, scelti tra quelli la cui destinazione precede quella del guidatore, scaricandoli nella stazione giusta e caricando, se possibile, altri passeggeri. La tariffa che viene pagata al guidatore è un importo costante per ciascun miglio di distanza percorso da ciascun passeggero. Eseguite la simulazione mille volte e visualizzate l'incasso medio per miglio percorso.

Progettate le classi `Car` (*automobile*), `Passenger` (*passeggero*), `Station` (*stazione*) e `Simulation` (*simulazione*).

** **P8.5.** Nell'esercizio precedente i guidatori caricano passeggeri a caso: cercate di migliorare questa strategia. Sarebbe meglio che i guidatori caricassero i passeggeri con cui condividono il percorso più lungo? Per rendere ottimo il piano di viaggio, sarebbe utile poter analizzare la situazione delle stazioni lungo l'intero percorso? Trovate una soluzione che aumenti l'incasso medio per miglio percorso.

** **P8.6.** Spesso i dati di una tabella vengono memorizzati nel formato CSV (*comma-separated values*, cioè valori separati da virgole). Ciascuna riga della tabella viene memorizzata in una riga del file, separando tra loro i valori delle singole colonne con una virgola. Se, però, un valore contiene una virgola, l'intero valore viene racchiuso tra virgolette, mentre se contiene un carattere virgolette, questo viene raddoppiato, come in questi esempi:

```
John Jacob Astor,1763,1848
"William Backhouse Astor, Jr.",1829,1892
"John Jacob ""Jakey"" Astor VI",1912,1992
```

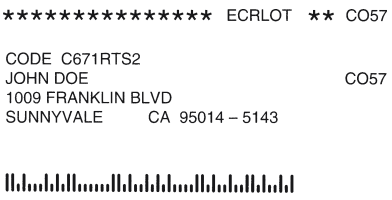
Progettate la classe `Table` con i metodi:

```
public void addLine(String line)
public String getEntry(int row, int column)
public int rows()
public int columns()
```

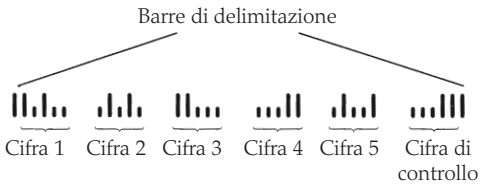
Risolvete il problema generando versioni intermedie della classe che siano sempre più complesse, progettando anche una classe di collaudo, seguendo l'approccio visto nel Paragrafo 8.5.

*** **P8.7.** 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 (analogo al codice di avviamento postale usato in Italia, come si può vedere nella figura).

Codice postale a barre



Codifica di cinque cifre mediante codice a barre



Lo schema di codifica per un codice ZIP di cinque cifre è illustrato nella figura. 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.

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:

	Peso				
Cifra	**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, genererebbe il valore 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.

★★★ **P8.8 (economia).** 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.

** **P8.9 (economia).** Realizzate una classe, `Customer`, che gestisca le informazioni relative a un cliente di una campagna di promozione commerciale sotto forma di premi di fedeltà. Dopo aver effettuato complessivamente acquisti per almeno $100, il cliente riceve uno sconto di $10 sull'acquisto successivo. Progettate questi metodi:

```
public void makePurchase(double amount) // registra un acquisto
public boolean discountReached() // true se e solo se ha raggiunto lo sconto
```

Scrivete un programma di collaudo che verifichi il funzionamento della classe in questa situazione: un cliente ha ottenuto uno sconto, poi lo usa spendendo più di $90 ma meno di $100 e questo non deve fargli ottenere un secondo sconto; infine, fa un altro acquisto, che lo porta ad ottenere un secondo sconto.

*** **P8.10 (economia).** L'associazione dei commercianti del centro cittadino vuole promuovere le vendite con una promozione a premi di fedeltà simile a quella vista nell'esercizio precedente. I negozi sono identificati da un numero intero compreso tra 1 e 20, quindi aggiungete una nuova variabile parametro al metodo `makePurchase` per indicare il negozio in cui è avvenuto l'acquisto. Lo sconto si ottiene facendo acquisti in almeno tre negozi diversi, per un importo totale di almeno $100.

*** **P8.11 (scienze).** Realizzate una classe, `Cannonball`, che rappresenti una palla di cannone che viene lanciata in aria. Una palla è caratterizzata da una posizione (coordinate x e y) e una velocità (con componenti nelle direzioni x e y). Progettate:

- un costruttore che riceve la coordinata x della posizione iniziale (la coordinata y iniziale vale zero);
- il metodo `move(double deltaSec)` che sposta la palla nella posizione successiva; prima calcola la distanza percorsa in `deltaSec` secondi, usando le velocità attuali, poi aggiorna le posizioni x e y, infine aggiorna la velocità lungo la direzione y, tenendo conto dell'accelerazione gravitazionale terrestre, pari a -9.81 m/s^2; la velocità lungo la direzione x non cambia;
- il metodo `Point getLocation()` che restituisce la posizione attuale della palla, con coordinate arrotondate al numero intero più vicino;
- il metodo `ArrayList<Point> shoot(double alpha, double v, double deltaSec)` che lancia la palla, i cui argomenti sono l'angolo α tra la traiettoria iniziale e il terreno, e la velocità iniziale v (calcola la velocità in direzione x come v cos α e la velocità lungo y come v sen α, poi invoca ripetutamente `move` con un intervallo di tempo pari a `deltaSec` secondi finché la coordinata y della posizione non diventa uguale a zero, cioè finché la palla di cannone non è ricaduta al suolo); il metodo restituisce un vettore contenente, nell'ordine, le posizioni in cui si trova la palla dopo ciascuna invocazione di `move`.

Usate questa classe in un programma che chieda all'utente l'angolo iniziale e la velocità di lancio, poi invochi `shoot` e visualizzi le posizioni.

* **P8.12 (grafica).** Aggiungete all'esercizio precedente il disegno della traiettoria della palla di cannone.

** **P8.13 (scienze).** I resistori come componenti elettronici discreti sono caratterizzati da una *resistenza nominale* (ad esempio, 6.2 kΩ) e da una *tolleranza*, espressa in percentuale della resistenza nominale (ad esempio, $\pm$ 5%), che è considerata una variazione accettabile della resistenza per

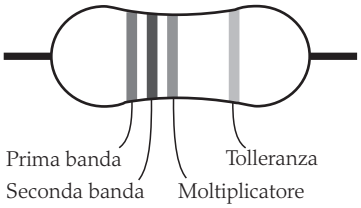
quel componente. Quindi, un resistore la cui resistenza sia 6.2 kΩ ± 5% può avere una resistenza effettiva che va da un valore minimo, 5.89 kΩ, a un valore massimo, 6.51 kΩ.

Progettate una classe, `Resistor`, che rappresenti un resistore come componente elettronico discreto e che sia dotata di un costruttore che accetta un valore per la resistenza nominale e un valore per la tolleranza, per poi determinare in modo casuale il valore della resistenza effettiva all'interno dell'intervallo ammissibile. La classe mette a disposizione metodi pubblici per ispezionare tre valori: la resistenza nominale, la tolleranza e la resistenza effettiva.

Scrivete un programma il cui metodo `main` utilizzi la classe `Resistor` e dimostri che questa funziona correttamente, visualizzando le resistenze effettive di dieci resistori caratterizzati dai valori 330 Ω ± 10%.

** **P8.14 (scienze).** Nella classe `Resistor` dell'esercizio precedente aggiungete un metodo che restituisca una descrizione delle "bande colorate" presenti sul resistore e che ne qualificano la resistenza nominale e la tolleranza. Un resistore (che è un componente quasi cilindrico) ha quattro bande colorate sulla superficie laterale, disposte parallelamente alle basi circolari, che seguono questa codifica, con i valori indicati nella tabella:

- la prima banda è la cifra più significativa del valore della resistenza nominale;
- la seconda banda è la seconda cifra più significativa del valore della resistenza nominale;
- la terza banda è l'esponente della potenza di dieci che moltiplica il valore della resistenza nominale;
- la quarta banda, un po' distante dalle altre tre, indica la tolleranza.



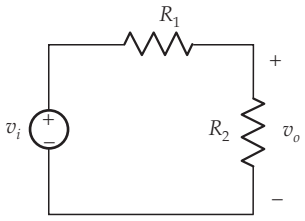
Colore	Cifra	Moltiplicatore	Tolleranza
Nero	0	$\times 10^0$	—
Marrone	1	$\times 10^1$	± 1%
Rosso	2	$\times 10^2$	± 2%
Arancione	3	$\times 10^3$	—
Giallo	4	$\times 10^4$	—
Verde	5	$\times 10^5$	± 0.5%
Blu	6	$\times 10^6$	± 0.25%
Violetto	7	$\times 10^7$	± 0.1%
Grigio	8	$\times 10^8$	± 0.05%
Bianco	9	$\times 10^9$	—
Oro	—	$\times 10^{-1}$	± 5%
Argento	—	$\times 10^{-2}$	± 10%
Nessuno	—	—	± 20%

Ad esempio, usando i valori presenti nella tabella, un resistore le cui bande siano, nell'ordine, di colore rosso, violetto, verde e oro, avrà la cifra più significativa del valore della resistenza nominale uguale a 2, la seconda cifra uguale a 7 e un moltiplicatore pari a 10^5, con una tolleranza $\pm$ 5%. Tutto questo indica una resistenza di 2700 kΩ $\pm$ 5%.

** **P8.15 (scienze).** La figura mostra un circuito elettrico di uso comune: un "divisore di tensione". L'ingresso del circuito è la tensione v_i, mentre l'uscita è la tensione v_o. L'uscita di un divisore di tensione è proporzionale al valore di ingresso e la costante di proporzionalità si chiama "guadagno" del circuito. Il divisore di tensione è rappresentato dall'equazione:

$$G = \frac{v_o}{v_i} = \frac{R_2}{R_1 + R_2}$$

dove G è il guadagno e R_1 e R_2 sono le resistenze dei due resistori che costituiscono il divisore di tensione.



Il processo di fabbricazione dei componenti introduce inevitabilmente nelle resistenze deviazioni dal valore nominale, come descritto nell'Esercizio P8.13. A loro volta, l'alterazione dei valori delle resistenze provoca variazioni nel valore del guadagno del divisore di tensione: parliamo, quindi, di valore *nominale* del guadagno, calcolato usando i valori nominali delle resistenze, e di valore *effettivo* del guadagno, calcolato con i valori effettivi delle resistenze.

Scrivete un programma costituito da due classi, `VoltageDivider` e `Resistor`, la seconda delle quali è già stata progettata nell'Esercizio P8.13. La classe `VoltageDivider` deve avere due variabili di esemplare che siano oggetti di tipo `Resistor` e un costruttore che riceva, appunto, due oggetti di tipo `Resistor`, ciascuno con un proprio valore di resistenza nominale e di tolleranza. Inoltre, la classe deve avere metodi pubblici per ispezionare il valore nominale e il valore effettivo del guadagno.

Il metodo `main` del programma deve verificare che la classe `VoltageDivider` funzioni correttamente, visualizzando il valore nominale e il valore effettivo del guadagno di dieci divisori, ciascuno dei quali costruito con resistori aventi tolleranza 5% e valori di resistenza nominale R_1 = 250 Ω e R_2 = 750 Ω.

Capitolo 9

** **P9.1.** Realizzate la classe `ChessPiece` ("pezzo nel gioco degli scacchi") con il metodo `setPosition(String coordinates)`. La stringa `coordinates` identifica riga e colonna di una posizione su una scacchiera usando la notazione scacchistica, dove, ad esempio, `"d8"` rappresenta la posizione iniziale della regina nera. Dotate la classe anche del metodo `ArrayList<String> canMoveTo()` che restituisca

la lista delle mosse valide per quel pezzo a partire dalla sua posizione attuale. Definite, poi, le sottoclassi `Pawn` (*pedone*), `Knight` (*cavallo*), `Bishop` (*alfiere*), `Rook` (*torre*), `Queen` (*regina*) e `King` (*re*).

** **P9.2.** Realizzate la classe `Clock` i cui metodi `getHours` e `getMinutes` restituiscano l'indicazione dell'istante attuale (sotto forma, rispettivamente, di ore e minuti) nel luogo in cui vi trovate (estraete questa informazione dalla stringa restituita dall'invocazione `java.time.LocalTime.now().toString()`. Aggiungete anche il metodo `getTime` che restituisca una stringa con ore e minuti, invocando i metodi `getHours` e `getMinutes`. Progettate, poi, la sottoclasse `WorldClock` il cui costruttore accetti il valore di una differenza oraria. Ad esempio, se vivete in California, un oggetto costruito con `new WorldClock(3)` vi deve mostrare l'ora di New York, che dista tre fusi orari. Quali metodi dovrete sovrascrivere, senza sovrascrivere `getTime`?

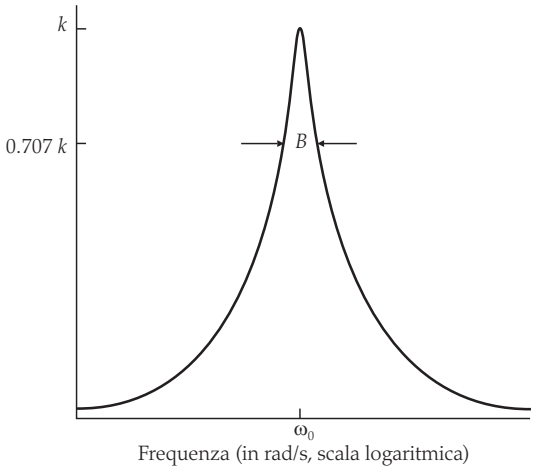
** **P9.3.** Aggiungete la funzione di "sveglia" (*alarm*) alla classe `Clock` progettata nell'esercizio precedente. Quando viene invocato il metodo `setAlarm(hours, minutes)`, l'orologio memorizza l'ora della sveglia. Quando viene invocato il metodo `getTime` e l'ora della sveglia è stata raggiunta o superata, viene restituito l'istante di tempo seguito dalla stringa `"Alarm"` (oppure, se preferite, la stringa `"\u23F0"`) e l'impostazione della sveglia viene annullata. Cosa dovete fare perché il metodo `setAlarm` funzioni anche con oggetti di tipo `WorldClock`?

** **P9.4 (economia).** Implementate la classe `Appointment` (*appuntamento*) e le sue sottoclassi `OneTime` (*occasionale*, ricorre una sola volta), `Daily` (*giornaliero*) e `Monthly` (*mensile*). Un appuntamento ha una descrizione (ad esempio, "visita dal dentista") e ricorre in una o più date. Scrivete il metodo `occursOn(int year, int month, int day)` che verifichi se l'appuntamento è previsto nella data descritta mediante anno (`year`), mese (`month`) e giorno (`day`). Ad esempio, nel caso di un appuntamento con ricorrenza mensile, il metodo deve verificare se il giorno del mese corrisponde. Poi, popolate una lista di appuntamenti di vario tipo (realizzata con un array) e chiedete all'utente di indicare una data, visualizzando tutti gli appuntamenti che sono previsti in tale data.

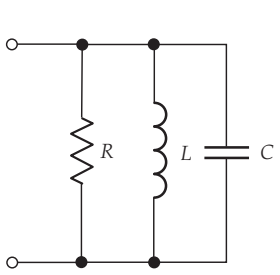
** **P9.5 (economia).** Migliorate l'agenda di appuntamenti dell'esercizio precedente consentendo all'utente di inserire un nuovo appuntamento, specificandone il tipo, la descrizione e la data.

*** **P9.6 (economia).** Migliorate ancora l'agenda di appuntamenti dei due esercizi precedenti consentendo all'utente di archiviare l'agenda in un file e di leggerla da un file scritto in precedenza. L'azione di archiviazione è semplice: implementate il metodo `save`, che, per ogni appuntamento, memorizzi nel file il tipo, la descrizione e la data. L'azione di lettura dal file è più complessa: per prima cosa bisogna determinare il tipo dell'appuntamento che si sta per leggere, poi si crea un oggetto di quel tipo e si invoca il suo metodo `load`, che prosegue la lettura dei dati dell'appuntamento.

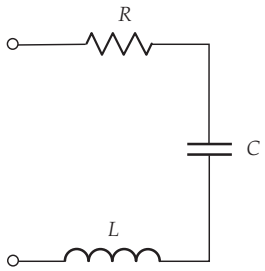
** **P9.7 (scienze).** I circuiti risonanti vengono usati per selezionare un segnale (ad esempio, una stazione radio o un canale televisivo) tra altri segnali in competizione e sono caratterizzati dalla risposta in frequenza riportata in figura. La risposta in frequenza di un circuito risonante è completamente descritta da tre parametri: la frequenza di risonanza ω_o, l'ampiezza di banda B e il guadagno k alla frequenza di risonanza.



La figura riporta due semplici circuiti risonanti. Il circuito (a) è un *circuito risonante in parallelo*, mentre il circuito (b) è un *circuito risonante in serie*. Entrambi sono costituiti da un resistore di resistenza *R*, un condensatore di capacità *C* e un induttore di induttanza *L*.



a) Circuito risonante in parallelo



b) Circuito risonante in serie

Questi circuiti vengono progettati determinando i valori di *R*, *C* e *L* che fanno in modo che la risposta in frequenza del circuito risonante sia quella descritta da specifici valori di ω_o, *B* e *k*. Le equazioni per il progetto del circuito risonante in parallelo sono:

$$R = k \qquad C = \frac{1}{BR} \qquad L = \frac{1}{\omega_0^2 C}$$

Analogamente, le equazioni per il progetto del circuito risonante in serie sono:

$$R = \frac{1}{k} \qquad L = \frac{R}{B} \qquad C = \frac{1}{\omega_0^2 L}$$

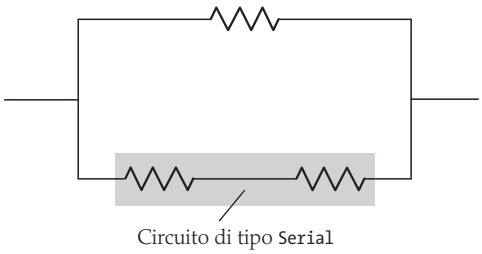
Scrivete un programma che usi la superclasse `ResonantCircuit` per rappresentare qualunque circuito risonante, con le sottoclassi `SeriesResonantCircuit` e `ParallelResonantCircuit`. Definite nella superclasse tre variabili di esemplare private per memorizzare i parametri ω_o, *B* e *k* della risposta in frequenza del circuito. La superclasse deve definire metodi pubblici per ispezionare tutte le variabili di esemplare e per assegnare loro un valore (i cosiddetti metodi *get* e *set*), oltre al metodo `display` che visualizzi una descrizione della risposta in frequenza.

Ogni sottoclasse deve definire un metodo che progetti il corrispondente circuito risonante e deve sovrascrivere il metodo `display` della superclasse in modo che visualizzi sia la risposta in

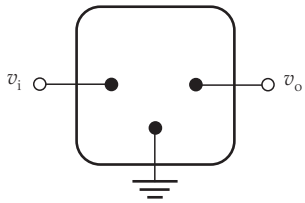
frequenza (cioè i suoi parametri, ω_o, B e k) sia il circuito (cioè i suoi parametri, R, C e L). Tutte le classi devono avere i costruttori opportuni.

Realizzate un programma di collaudo che evidenzi come tutte le sottoclassi funzionino correttamente.

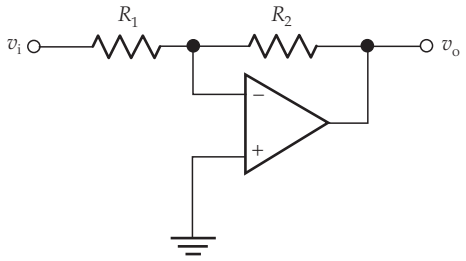
★★★ **P9.9 (scienze).** Questo problema ha l'obiettivo di rappresentare un circuito elettrico costituito da una configurazione arbitraria di resistori, ciascuno con la propria resistenza, usando la legge di Ohm per calcolare la resistenza complessiva. Definite la superclasse `Circuit`, con il metodo di esemplare `getResistance`, poi definite la sua sottoclasse `Resistor`, i cui oggetti siano singoli resistori. Progettate le sottoclassi `Serial` e `Parallel`, ciascuna delle quali contiene un oggetto di tipo `ArrayList<Circuit>`. Un circuito di tipo `Serial` rappresenta un insieme di circuiti posti in serie, ciascuno dei quali può essere un singolo resistore o un altro circuito. Analogamente, un circuito di tipo `Parallel` rappresenta un insieme di circuiti posti in parallelo. Ad esempio, il circuito qui raffigurato è un circuito di tipo `Parallel` contenente un singolo resistore (in alto nella figura) e un circuito di tipo `Serial`.



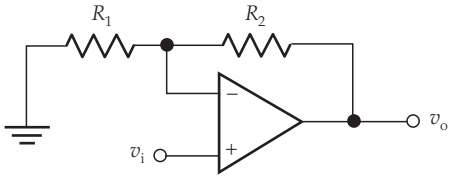
★★ **P9.10 (scienze).** La figura (a) mostra una rappresentazione simbolica di un circuito elettrico che prende il nome di *amplificatore*: riceve in ingresso la tensione v_i e fornisce in uscita la tensione v_o, proporzionale a v_i secondo una costante di proporzionalità che è il guadagno (*gain*) dell'amplificatore.



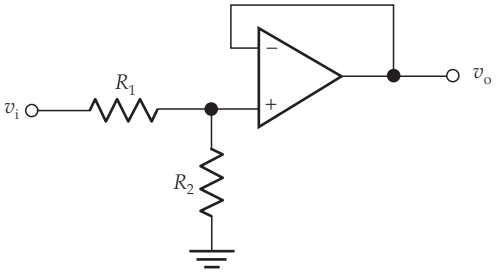
a) Amplificatore



b) Amplificatore invertente



c) Amplificatore non invertente



d) Amplificatore a divisione di tensione

Le figure (b), (c) e (d) mostrano tre esempi di amplificatori (rispettivamente, l'amplificatore invertente, l'amplificatore non invertente e l'amplificatore a divisione di tensione), ciascuno dei quali è costituito da due resistori e un amplificatore operazionale (sinteticamente, *op amp*). Il valore del guadagno di ciascun amplificatore dipende dai valori delle sue resistenze. In particolare, il guadagno dell'amplificatore invertente è $g = -R_2/R_1$. Analogamente, il guadagno dell'amplificatore invertente e dell'amplificatore a divisione di tensione sono, rispettivamente, $g = 1 + R_2/R_1$ e $g = R_2/(R_1 + R_2)$.

Scrivete un programma che rappresenti l'amplificatore mediante una superclasse avente i tre esempi di amplificatore qui citati come sue sottoclassi. La superclasse deve avere due metodi, `getGain` che restituisce il guadagno e `getDescription` che restituisce una stringa che descrive l'amplificatore. Ogni sottoclasse deve avere un costruttore con due parametri (le sue resistenze) e deve sovrascrivere i metodi `getGain` e `getDescription` della superclasse.

Realizzate un programma di collaudo che evidenzi come tutte le sottoclassi funzionino correttamente con alcuni diversi valori di resistenze.

Capitolo 10

** **P10.1.** Nell'ipotesi che in un programma sia presente una stringa che contiene il testo di un intero libro e che si vogliano analizzare sue porzioni, costruire sottostringhe è piuttosto inefficiente: è decisamente preferibile memorizzare il riferimento al testo originale e usare una coppia di numeri interi per identificare le posizioni iniziale e finale della porzione interessata. Progettate una classe `Segment` che svolga il compito di rappresentare, in tal modo, una porzione di una stringa, implementando l'interfaccia `CharSequence` della libreria standard.

Scrivete, poi, un programma di collaudo che verifichi che oggetti di tipo `Segment` possono essere forniti come parametri a metodi che accettano riferimenti di tipo `CharSequence`, come `String.join` e `PrintStream.append`.

** **P10.2.** Modificate il metodo `display` della classe `LastDigitDistribution` vista nella sezione Esempi completi 10.1 in modo che produca un istogramma, come questo:

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

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

** **P10.3.** Progettate una classe `PrimeSequence` che implementi l'interfaccia `Sequence` vista nella sezione Esempi completi 10.1 generando la sequenza dei numeri primi.

* **P10.4.** Aggiungete all'interfaccia `Sequence` della sezione Esempi completi 10.1 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 a ogni invocazione di `next`, fino all'esaurimento dei dati stessi. La classe `SequenceDemo` deve visualizzare la distribuzione dell'ultima cifra in tutti i valori della sequenza.

* **P10.5.** Progettate una classe `FirstDigitDistribution` che funzioni come la classe `LastDigitDistribution` vista nella sezione Esempi completi 10.1, 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.

** **P10.6.** Definite un'interfaccia `Filter` (*filtro*) in questo modo:

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

Modificate la classe `Data` vista nel Paragrafo 10.4 in modo che usi sia un oggetto `Measurer` sia un oggetto `Filter`, misurando soltanto oggetti accettati dal filtro. Mostrate il corretto funzionamento della vostra modifica elaborando un insieme di conti bancari, nel quale vengano ignorati tutti i conti con saldo minore di $ 1000.

** **P10.7.** Risolvete l'esercizio precedente usando come filtro un'espressione lambda.

** **P10.8.** Aggiungete all'interfaccia `Filter` definita nell'Esercizio P10.6 un metodo che conti quanti oggetti sono accettati dal filtro:

```
static int count(Object[] values, Filter condition)
```

** **P10.9.** Aggiungete all'interfaccia `Filter` definita nell'Esercizio P10.6 un metodo che lasci nel contenitore gli oggetti che sono accettati dal filtro, eliminando gli altri:

```
static void retainAll(Object[] values, Filter condition)
```

** **P10.10.** Aggiungete all'interfaccia `Filter` definita nell'Esercizio P10.6 un metodo predefinito `default boolean reject(Object x)` che restituisca `true` per tutti gli oggetti che non sono accettati dal filtro.

** **P10.11.** Aggiungete all'interfaccia `Filter` definita nell'Esercizio P10.6 un metodo predefinito `default Filter invert()` che restituisca un filtro che accetti tutti e soli gli oggetti che vengono rifiutati dal filtro in esame.

** **P10.12.** Considerate la seguente interfaccia, che descrive il comportamento di un oggetto che trasforma in stringa un numero intero:

```
public interface NumberFormatter
{
   String format(int n);
}
```

Progettate quattro classi che implementino questa interfaccia. La prima, `DefaultFormatter`, trasforma in stringa un numero intero nel modo consueto. La seconda, `DecimalSeparatorFormatter`, aggiunge virgole come separatori delle migliaia: ad esempio, il numero un milione viene trasformato nella stringa `"1,000,000"`. La terza, `AccountingFormatter`, racchiude i numeri negativi tra parentesi tonde: ad esempio, il numero –1 viene trasformato nella stringa `"(1)"`. La quarta, `BaseFormatter`, considera il numero in base *n*, dove *n* è un numero fornito al costruttore e compreso tra 2 e 36.

** **P10.13.** Scrivete un metodo che riceva come parametri un array di numeri interi e un oggetto `NumberFormatter` (come definito nell'esercizio precedente) e visualizzi ciascun numero su una riga separata, dopo averlo trasformato in stringa usando l'oggetto `NumberFormatter` ricevuto. I numeri visualizzati devono essere incolonnati a destra.

** **P10.14.** 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 oggetti di qualunque classe che implementi 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 implementi l'interfaccia `Formattable`. Ignorate i parametri `flags` e `precision`, visualizzando semplicemente il saldo del conto mediante una stringa di lunghezza `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'ulteriore interfaccia, implementata 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 lunghezza desiderata. Infine, passate tale stringa al metodo `append`.

*** **P10.15.** Migliorate il metodo `formatTo` dell'esercizio precedente in modo che prenda in considerazione anche il parametro `precision`.

* **P10.16.** Si vuole progettare un programma generico che sia in grado di gestire giochi da tavolo con due giocatori, sufficientemente flessibile da poter essere utilizzato per giochi come tris (*tic-tac-toe*, in inglese), scacchi (*chess*) o il Gioco di Nim visto nell'Esercizio P6.5.

Progettate l'interfaccia `Game` che descriva un gioco da tavolo, pensando a ciò che può servire al vostro programma. Innanzitutto, il programma chiederà al primo giocatore di comunicare la sua prima mossa (descritta mediante una stringa in un formato caratteristico di quel particolare gioco, come la stringa `Be3` nel gioco degli scacchi). Il vostro programma non ha informazioni relative ai singoli giochi, per cui l'interfaccia `Game` deve avere un metodo come questo, che restituisca `true` se e solo se la mossa `move` è valida nel gioco:

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

Dopo che una mossa è stata ritenuta valida, deve essere messa in atto (o "eseguita"), per cui l'interfaccia necessita di un ulteriore metodo, `executeMove`. Successivamente, il programma ha bisogno di verificare se il gioco è terminato: in caso contrario, viene acquisita ed elaborata la mossa dell'altro giocatore. Infine, occorre prevedere qualche meccanismo (cioè un metodo) per visualizzare lo stato del gioco.

Dopo aver progettato l'interfaccia `Game`, fornitene due implementazioni a vostra scelta, ad esempio `Nim` e `Chess` (oppure `TicTacToe` se siete meno ambiziosi). La vostra classe `GamePlayer` ("giocatore") deve gestire un riferimento di tipo `Game` senza avere alcuna informazione sul gioco, elaborando le mosse di entrambi i giocatori. Scrivete due programmi che differiscano solamente per il valore iniziale del riferimento `Game`.

*** **P10.17 (grafica).** Scrivete un programma che visualizzi un messaggio che può scorrere in un pannello, un meccanismo solitamente chiamato *scrolling*. Usate un temporizzatore per ottenere l'effetto grafico di scorrimento. Nel ricevitore di eventi del temporizzatore spostate la posizione in cui inizia il messaggio e ridisegnatelo. Quando il messaggio ha abbandonato la finestra riportate la sua posizione all'altro angolo. Progettate un'interfaccia di interazione con l'utente che consenta di personalizzare: il testo del messaggio, il *font* utilizzato per la visualizzazione, i colori dello sfondo e del testo, la velocità di scorrimento e la sua direzione.

*** **P10.18 (grafica).** Scrivete un programma che consenta all'utente di specificare un triangolo mediante tre "click" del mouse. Dopo il primo click disegnate un cerchietto. Dopo il secondo click disegnate un segmento che colleghi i primi due punti. Dopo il terzo click disegnate l'intero triangolo. Il quarto click cancella il triangolo visualizzato e ne inizia un altro.

*** **P10.19 (grafica).** Scrivete un programma che consenta a due giocatori di giocare a TicTacToe ("tris"). Disegnate la scacchiera del gioco e segnalate quale giocatore sia di turno (X oppure O). Quando viene premuto un pulsante del mouse, controllate che il puntatore del mouse si trovi all'interno di una casella vuota della scacchiera, dopodiché inserite nella casella il contrassegno del giocatore di turno e assegnate il turno successivo all'altro giocatore. Quando un giocatore vince, segnalate chi sia il vincitore. Aggiungete anche un pulsante che consenta di iniziare una nuova partita.

*** **P10.20 (grafica).** Scrivete un programma che consenta all'utente di disegnare un istogramma usando il mouse. Quando l'utente preme il pulsante del mouse (cioè "fa click") all'interno di una barra dell'istogramma, il click successivo porta la lunghezza della barra a un valore pari alla coordinata *x* del punto di click (se tale lunghezza è vicina allo zero, la barra viene eliminata). Quando l'utente fa click al di sotto dell'ultima barra, viene aggiunta una nuova barra la cui lunghezza è pari alla coordinata *x* del punto di click.

* **P10.21 (collaudo).** Considerate 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 e la 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 (*mock*) che sceglie la prima casella libera.

** **P10.22 (collaudo).** Considerate il compito di 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, racchiudendolo tra i corretti marcatori HTML e convertendo i caratteri speciali presenti.

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

Recuperare il testo da Internet e scomporlo in segmenti è un compito arduo, per cui dovete definire un'interfaccia e progettare una classe semplificata con i cui esemplari si possa collaudare la classe che si occupa di portare a termine la conversione in HTML.

** **P10.23 (grafica).** Scrivete un programma che visualizzi la crescita di una popolazione di scarafaggi: iniziate con due scarafaggi e raddoppiate il numero di scarafaggi presenti a ogni click del mouse.

** **P10.24 (grafica).** Scrivete un programma che faccia muovere un'automobile all'interno di una finestra.

*** **P10.25 (grafica).** Scrivete un programma che faccia muovere all'interno di una finestra due automobili, in direzioni opposte ma ad altezze diverse, in modo che non si scontrino.

** **P10.26 (grafica).** Scrivete un programma che chieda all'utente di fornire, mediante finestre di dialogo di tipo `JOptionPane`, le coordinate *x* e *y* del centro di un cerchio e la lunghezza del suo raggio. Quando l'utente preme il pulsante “Draw”, il programma deve chiedere i dati appena descritti e disegnare all'interno di un componente grafico un cerchio avente tale centro e raggio.

** **P10.27 (grafica).** Scrivete un programma che chieda all'utente di fornire, mediante una finestra di dialogo di tipo `JOptionPane`, la lunghezza del raggio di un cerchio, indicandone poi il centro con un click del mouse all'interno del componente grafico. In questo caso non serve un pulsante “Draw”.

*** **P10.28 (grafica).** Scrivete un programma che consenta all'utente di specificare 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 acquisita la posizione del centro del cerchio in un evento precedente.

* **P10.29 (grafica).** Progettate un'interfaccia, `MoveableShape`, che possa essere utilizzata come meccanismo generale per l'animazione di una forma geometrica. Una 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`). Progettate poi le sagome mobili di un rettangolo e di un'automobile.

Capitolo 11

** **P11.1.** Scrivete un programma che legga un file nel formato di `babynames.txt` (visto nella sezione Esempi completi 11.1) e generi due file, `boynames.txt` e `girlnames.txt`, separando i dati relativi a maschi e femmine.

*** **P11.2.** Scrivete un programma che legga un file nel formato di `babynames.txt` (visto nella sezione Esempi completi 11.1) e visualizzi tutti i nomi che sono sia maschili sia femminili (come Alexis o Morgan).

** **P11.3.** Usando la procedura descritta nella sezione Argomenti avanzati 11.1, scrivete un programma che legga tutti i dati presenti in una pagina web e li scriva in un file di testo. Chiedete all'utente l'indirizzo (URL) della pagina web e il nome del file.

*** **P11.4.** Il formato CSV (l'acronimo di *comma-separated values*, cioè "valori separati da virgole") è molto usato per memorizzare dati in forma tabulare. Ogni riga della tabella è una riga del file, con colonne separate da virgole. I singoli valori possono essere racchiusi tra virgolette, cosa che deve certamente accadere se contengono virgole o virgolette. Le virgolette presenti all'interno di valori racchiusi tra virgolette vanno raddoppiate. Ecco un esempio, una riga con quattro valori:

```
1729, San Francisco, "Hello, World", "He asked: ""Quo vadis?"""
```

Realizzate la classe `CSVReader` che legga un file CSV e metta a disposizione i metodi seguenti:

```
int numberOfRows() // numero di righe
int numberOfFields(int row) // numero di valori della riga row
String field(int row, int column) // valore presente nella riga row e colonna comumn
```

** **P11.5.** Trovate dei dati interessanti in formato CSV (o nel formato di uno *spreadsheet* che sapete utilizzare, per poi convertirli in formato CSV). Usando la classe `CSVReader` progettata nello svolgimento dell'Esercizio P11.4, leggete i dati e calcolatene una forma riassuntiva, come il valore massimo, minimo o medio di una delle colonne.

** **P11.6.** Scaricate il file `airports.dat` dall'indirizzo web `https://openflights.org/data.html`, poi scrivete un programma che chieda all'utente il nome di una città, legga i dati presenti nel file e visualizzi i nomi di tutti gli aeroporti presenti in tale città. Questo file utilizza la codifica UTF-8, descritta nella sezione Argomenti avanzati 11.3.

** **P11.7.** Usando la procedura descritta nella sezione Argomenti avanzati 11.1, scrivete un programma che legga tutti i dati presenti in una pagina web e ne visualizzi tutti i collegamenti ipertestuali (*hyperlink*), aventi il formato:

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

Otterrete una valutazione migliore se il 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).

** **P11.8.** 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` vista nel Paragrafo 8.2 il metodo

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

che lanci un'eccezione se una riga 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.

*** **P11.9.** 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 nella classe `Bank` un metodo `readFile` 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 collaudo che legga un file di conti correnti, per poi visualizzare il numero del conto con il saldo maggiore. Se il file non contiene dati corretti, date all'utente la possibilità di selezionare un diverso file.

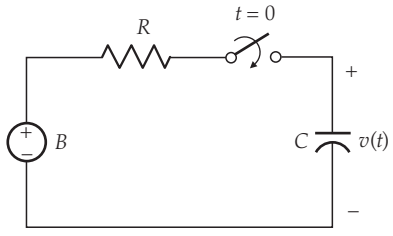
** **P11.10 (economia).** Il responsabile amministrativo di un albergo registra le transazioni in un file di testo. Ogni riga contiene le seguenti informazioni, separate da "punti e virgola": il nome del cliente, il servizio venduto (ad esempio, cena, conferenza, alloggio e così via), l'importo pagato e la data dell'evento. Scrivete un programma che legga un tale file di testo e visualizzi l'importo totale relativo a ciascun servizio, segnalando un errore se il file non esiste oppure se il suo formato non è corretto.

** **P11.11 (economia).** Scrivete un programma che legga un file di testo nel formato descritto nell'esercizio precedente e scriva un file distinto per ciascun servizio venduto, contenente le righe a esso relative. I nomi dei file che vengono scritti devono ricordare il servizio a cui si riferiscono.

** **P11.12 (economia).** Il proprietario di un negozio registra le transazioni giornaliere in un file di testo. Ogni riga contiene tre voci: il numero dello scontrino, l'importo della vendita e la lettera P nel caso in cui si tratti di un importo pagato, R nel caso di un importo ricevuto. Le singole voci sono separate da spazi. Scrivete un programma che chieda al proprietario del negozio di inserire, come dati, la quantità di denaro presente in cassa all'inizio e alla fine della giornata, e il nome del file, per poi verificare se la quantità di denaro a fine giornata è quella prevista.

*** **P11.13 (scienze).** Nel circuito elettrico qui raffigurato, quando l'interruttore è chiuso la tensione (in volt) ai capi del condensatore è data dall'equazione seguente:

$$v(t) = B\left(1 - e^{-t/(RC)}\right)$$



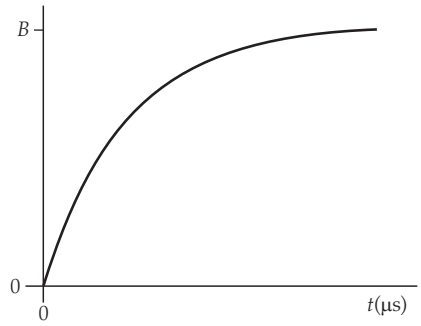
Quindi se, ad esempio, i parametri del circuito sono $B = 12\,\text{V}$, $R = 500\ \Omega$ e $C = 0.25\ \mu\text{F}$, si ha:

$$v(t) = 12\left(1 - e^{-0.008t}\right)$$

dove l'istante di tempo, t, è espresso in μs. Scrivete un programma che legga il file `params.txt` contenente i valori di B, R e C, nonché gli istanti di tempo iniziale e finale; poi, il programma deve scrivere un file, `rc.txt`, contenente i valori del tempo t e della corrispondente tensione $v(t)$ ai capi del condensatore, con t che varia dal proprio valore iniziale a quello finale con 100 incrementi della stessa durata. Nel nostro esempio, se t andasse da 0 a 1000 μs, la dodicesima riga del file prodotto sarebbe:

```
110 7.02261
```

★★★ **P11.14 (scienze).** La figura mostra il grafico della tensione ai capi del condensatore nel circuito elettrico dell'esercizio precedente. La tensione aumenta da 0 a B volt e il "tempo di salita" (*rise time*) è definito come il tempo necessario perché la tensione passi da $v_1 = 0.05 \times B$ a $v_2 = 0.95 \times B$.



Il file `rc.txt` contiene un elenco di istanti di tempo, t, e della corrispondente tensione $v(t)$. Ogni riga contiene un istante di tempo (espresso in μs) e la tensione corrispondente (in volt), come in questo esempio:

```
110 7.02261
```

dal quale si deduce che all'istante di tempo 110 μs la tensione ai capi del condensatore è pari a 7.02261 V. Allo scorrere delle righe nel file, il tempo aumenta.

Scrivete un programma che legga il file `rc.txt` e usi i dati per calcolare il tempo di salita. Come approssimazione del valore di B usate la tensione presente nell'ultima riga del file e individuate gli istanti di tempo nei quali la tensione è più vicina ai valori $0.05 \times B$ e $0.95 \times B$.

★★ **P11.15 (scienze).** Supponiamo che un file contenga le energie e le lunghezze di alcuni legami covalenti, nel seguente formato:

Legame singolo, doppio o triplo	Energia di legame (kJ/mol)	Lunghezza di legame (nm)
C|C	370	0.154
C||C	680	0.13
C|||C	890	0.12
C|H	435	0.11
C|N	305	0.15
C|O	360	0.14
C|F	450	0.14
C|Cl	340	0.18
O|H	500	0.10
O|O	220	0.15
O|Si	375	0.16
N|H	430	0.10
N|O	250	0.12
F|F	160	0.14
H|H	435	0.074

Scrivete un programma che chieda dall'utente un dato presente in una colonna e visualizzi i dati presenti sulla stessa riga nelle altre due colonne, cercandoli nel file. Se il dato fornito dall'utente trova corrispondenza in più righe, il programma deve restituire i dati di tutte. Ad esempio, ricevendo in ingresso il valore 0.12 (che è una lunghezza di legame), il programma deve rispondere con il triplo legame C|||C, avente energia di legame pari a 890 kJ/mol, e con il singolo legame N|O, avente energia 250 kJ/mol.

Capitolo 12

** **P12.1.** Usando il metodo `isPalindrome` visto nel Paragrafo 12.2 e il metodo ricorsivo `reverse` progettato nell'Esercizio E12.6, scrivete un programma che verifichi la seguente ipotesi matematica: si ritiene (ma non è dimostrato) che, sommando un numero decimale qualsiasi e il numero che si ottiene invertendone le cifre, si giunga prima o poi, continuando la procedura con il risultato ottenuto, a un numero palindromo. Ad esempio:

89 + 98 = **187**
187 + 781 = **968**
968 + 869 = 1837
...
1801200002107 + 7012000021081 = ***8813200023188***

Scrivete un programma che esegua tale procedura, a partire da un numero intero casuale generato in questo modo:

```
new BigInteger(16, new java.util.Random())
```

** **P12.2.** I numeri di telefono e i codici PIN si ricordano più facilmente se si trovano parole che coincidano con la sequenza numerica quando vengono digitate su una normale tastiera telefonica, dove ogni tasto corrisponde a un numero e ad alcune lettere. Ad esempio, invece di ricordare la combinazione numerica 5282, si può pensare alla parola JAVA.

Scrivete un metodo ricorsivo che, dato un numero, restituisca tutte le possibili sequenze di lettere corrispondenti (indipendentemente dal fatto che siano parole reali oppure no).

** **P12.3.** Proseguite l'esercizio precedente, verificando che le sequenze di lettere generate siano parole presenti in un dizionario standard, come quello contenuto nel file `/usr/share/dict/words` (eventualmente reperibile in Internet). Dato un numero, restituite soltanto le parole reali.

*** **P12.4.** Sempre con riferimento all'Esercizio P12.2, se il numero è più lungo possono servire più parole per ricordarlo. Ad esempio, il numero 263-346-5282 può essere CODE IN JAVA. Usando quanto prodotto come soluzione dell'esercizio precedente, scrivete un programma che, dato un numero, elenca tutte le sequenze di parole reali che vi corrispondono.

*** **P12.5.** Modificate il metodo `permutations` visto nel Paragrafo 12.4 (che calcolava tutte le permutazioni di una stringa), trasformandolo 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, individuare una strategia 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 agisca all'interno dell'insieme delle permutazioni della sottostringa "at": nel metodo nextPermutation chiediamo semplicemente a tailIterator quale sia la *sua* successiva permutazione e vi 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 di permutazioni per tale nuova stringa.

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

*** **P12.6.** La classe seguente genera tutte le permutazioni dei numeri $0, 1, 2, \ldots, n - 1$, 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 di una stringa, `word`. 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

*** **P12.7.** Realizzate un iteratore che generi le mosse necessarie per risolvere il rompicapo delle Torri di Hanoi (descritto nella sezione Esempi completi 12.2), mettendo a disposizione i metodi `hasMoreMoves` e `nextMove`. Il metodo `nextMove` restituisce una stringa che descrive la mossa successiva. Ad esempio, il frammento di codice seguente visualizza tutte le mosse necessarie per spostare cinque dischi dal piolo 1 al piolo 3:

```
DiskMover mover = new DiskMover(5, 1, 3);
while (mover.hasMoreMoves())
{
   System.out.println(mover.nextMove());
}
```

Suggerimento: un `DiskMover` ("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 partenza to peg arrivo
```

Un `DiskMover` che deve spostare più di un disco deve faticare di più e ha bisogno di un altro oggetto `DiskMover` che lo aiuti a spostare i primi $d - 1$ dischi. Il metodo `nextMove` chiede a tale trasportatore di dischi "ausiliario" quale sia la sua successiva mossa finché non ha terminato, dopodiché visualizza il comando per spostare il disco d-esimo. Infine, costruisce un altro `DiskMover` che genera le mosse rimanenti.

È utile tenere traccia dello stato di un `DiskMover`:

- `BEFORE_LARGEST`: un `DiskMover` ausiliario sposta la pila di dimensione $d - 1$ sul piolo ausiliario.
- `LARGEST`: sposta il disco più grande dal piolo di partenza al piolo di arrivo.

- `AFTER_LARGEST`: un `DiskMover` ausiliario sposta la pila di dimensione $d - 1$ dal piolo ausiliario al piolo di arrivo.
- `DONE`: tutte le mosse sono state eseguite.

★★★ **P12.8.** *Uscire da un labirinto.* Vi trovate all'interno di un labirinto, le cui pareti 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.

★★ **P12.9.** Usando la tecnica di backtracking, scrivete un programma che verifichi se una parola data possa essere scritta usando soltanto simboli di elementi chimici: ad esempio, "brother" può essere scritta come Br O Th Er, usando i simboli di bromo, ossigeno, torio e erbio.

★★ **P12.10.** Usando l'interfaccia `PartialSolution` e il metodo `solve` progettati nell'Esercizio E12.23, definite la classe `MazePartialSolution` che risolva il problema dell'uscita da un labirinto, così come definito nell'Esercizio P12.8

★★★ **P12.11.** Il valutatore di espressioni visto nel Paragrafo 12.5 restituisce il valore di un'espressione. Modificatelo in modo che restituisca un esemplare dell'interfaccia `Expression`, dotata dell'unico metodo `int value()`. Le classi che implementano tale interfaccia sono `Constant` (un valore numerico), `Sum` (il risultato di un'addizione), `Difference` (il risultato di una sottrazione), `Product` (il risultato di una moltiplicazione) e `Quotient` (il risultato di una divisione). La classe `Constant` memorizza un numero, che viene restituito dal suo metodo `value`. Le altre quattro classi memorizzano due argomenti di tipo `Expression` e il loro metodo `value` restituisce, rispettivamente, la somma, la differenza, il prodotto e il quoziente dei valori dei due argomenti. Scrivete un programma di collaudo che legga una stringa contenente un'espressione, la traduca in un oggetto di tipo `Expression` e ne visualizzi il risultato invocandone il metodo `value`.

★★★ **P12.12.** Migliorate il valutatore di espressioni dell'esercizio precedente in modo che le espressioni possano contenere la variabile `x`. Ad esempio, `3*x*x+4*x+5` è un'espressione valida. Modificate l'interfaccia `Expression` in modo che il metodo `value` riceva come parametro il valore di `x` e aggiungete la classe `Variable` che rappresenti una variabile `x`. Scrivete un programma che legga il valore di `x` e una stringa contenente un'espressione, la traduca in un oggetto di tipo `Expression` e ne visualizzi il risultato invocandone il metodo `value(x)`.

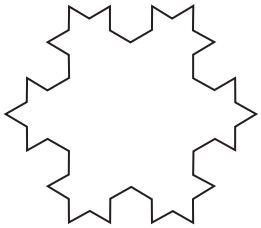
★★ **P12.13.** Aggiungete all'interfaccia `Expression` (descritta nei due esercizi precedenti) il metodo `toString` che restituisca una stringa che rappresenta l'espressione. È lecito utilizzare un numero di parentesi maggiore di quello effettivamente richiesto dalla consueta notazione matematica: ad esempio, l'espressione `3*x*x+5` può essere rappresentata dalla stringa `(((3*x)*x)+5)`.

*** **P12.14.** Scrivete un programma che legga un'espressione contenente numeri interi e la variabile `x`, costruendo il corrispondente oggetto di tipo `Expression`, per poi calcolarne la derivata. Aggiungete all'interfaccia `Expression` il metodo `Expression derivative()`. Usate le regole consuete per calcolare la derivata di una somma, di una differenza, di un prodotto, di un quoziente, di una costante e di una variabile, senza semplificare il risultato. Visualizzate l'espressione risultante: ad esempio, dopo aver letto `x*x`, il programma deve visualizzare `((1*x)+(x*1))`.

*** **P12.15 (grafica).** *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. Aggiungete un pulsante che, quando premuto, genera l'iterazione successiva.

Capitolo 13

** **P13.1.** Spesso si usano nomi di cartelle come `dir1`, `dir2`, e così via. Quando ci sono dieci cartelle o più, il sistema operativo le visualizza secondo l'ordine tipico di un dizionario, cioè `dir1`, `dir10`, `dir11`, `dir12`, `dir2`, `dir3`, e così via. Una cosa fastidiosa a cui si può porre facilmente rimedio. Progettate un comparatore che confronti stringhe che terminano con sequenze di cifre in modo da seguire il senso comune: prima confrontate come stringa la parte che precede le cifre, poi confrontate il valore numerico delle cifre.

*** **P13.2.** A volte i nomi delle cartelle o dei file contengono numeri in posizione intermedia (né all'inizio né alla fine) e ci possono essere più numeri, ad esempio `sec3_14.txt` o `sec10_1.txt`. Progettate un comparatore che confronti tali stringhe in modo da seguire il senso comune: scomponete ciascuna stringa in stringhe che non contengono cifre e stringhe che sono gruppi di cifre, poi confrontate due stringhe confrontando come stringa la prima stringa priva di cifre, poi il primo gruppo di cifre come se fosse un numero, e così via.

** **P13.3.** Implementate il seguente algoritmo di ordinamento: si suddivide l'array dato, `a`, in segmenti non decrescenti e decrescenti (cioè segmenti tali che `a[i]` $\leq$ `a[i+1]` $\leq \ldots \leq$ `a[i+k]` oppure `a[j]` $>$ `a[j+1]` $> \ldots >$ `a[j+m]`), dopodiché si invertono tutti i segmenti decrescenti, in modo che l'array

risulti suddiviso in segmenti non decrescenti; a questo punto, si fondono i primi due segmenti, poi i due segmenti successivi, e così via, in modo che ciascuna coppia di segmenti consecutivi dia luogo a un segmento non decrescente, ripetendo la procedura finché non siano stati fusi tutti i segmenti.

** **P13.4.** La mediana m di una sequenza di n elementi è quell'elemento che si troverebbe al centro della sequenza se questa fosse ordinata, per cui m è non minore di metà degli elementi e non maggiore degli altri. La mediana si può ovviamente identificare ordinando la sequenza, ma si possono ottenere prestazioni un po' migliori usando il seguente algoritmo, che trova il k-esimo elemento di una sequenza che va dalla posizione a (inclusa) alla posizione b (esclusa). Per trovare la mediana si usa $k = n/2$, $a = 0$ e $b = n$.

```
select(k, a, b):
   Scegli un pivot p nella sotto-sequenza individuata da a e b.
   Suddividi gli elementi della sotto-sequenza in tre sotto-sequenze:
         gli elementi < p, gli elementi = p, gli elementi > p.
   Siano n1, n2 e n3 le dimensioni di queste tre sotto-sequenze.
   if k < n1
      return select(k, 0, n1).
   else if k > n1 + n2
      return select(k, n1 + n2, n).
   else
      return p.
```

Implementate questo algoritmo e misurate la sua efficienza temporale nel calcolo della mediana di una sequenza casuale di grandi dimensioni, confrontandola con quella dell'algoritmo che ordina la sequenza e, poi, ne prende l'elemento centrale.

** **P13.5.** Implementate la seguente modifica dell'algoritmo quicksort, dovuta a Bentley e McIlroy: invece di usare il primo elemento come pivot, usa un'approssimazione della mediana.

Se $n \leq 7$, usa l'elemento centrale. Altrimenti, se $n \leq 40$, usa la mediana della sequenza costituita dal primo elemento, dall'ultimo elemento e dall'elemento centrale. Altrimenti calcola la "pseudo-mediana" dei nove elementi di tipo `a[i * (n - 1) / 8]`, con `i` che varia tra 0 e 8, definita come $\text{med}(\text{med}(v_0, v_1, v_2), \text{med}(v_3, v_4, v_5), \text{med}(v_6, v_7, v_8))$.

Confrontate il tempo d'esecuzione di questa variante con quello dell'algoritmo originale quando si devono ordinare sequenze quasi ordinate o quasi ordinate al contrario, e sequenze aventi molti elementi identici. Cosa si può osservare?

*** **P13.6.** Bentley e McIlroy suggeriscono la seguente modifica all'algoritmo quicksort, da usare quando si devono elaborare sequenze che contengono molti elementi ripetuti.

Invece di suddividere nelle due sotto-sequenze contenenti elementi non maggiori e elementi non minori del pivot, è meglio individuare tre sotto-sequenze, contenenti gli elementi minori del pivot, uguali al pivot e maggiori del pivot. Un risultato che, però, non è facile da ottenere in modo diretto, per cui raccomandano di eseguire la suddivisione in quattro sotto-sequenze: la prima contenente alcuni degli elementi uguali al pivot, la seconda e la terza contenenti, rispettivamente, gli elementi minori e maggiori del pivot, e la quarta con gli altri elementi uguali al pivot. Poi, si scambiano tra loro le prime due e le ultime due sotto-sequenze, in modo che tutti gli elementi uguali al pivot vadano a finire nella sotto-sequenza centrale. Implementate questa variante e verificate se effettivamente le prestazioni migliorano nel caso di sequenze con molti elementi ripetuti.

* **P13.7.** Implementate l'algoritmo di ordinamento per radice (*radix sort*), descritto nell'Esercizio R13.26, limitatamente al caso dell'ordinamento di array di numeri interi compresi tra 0 e 999.

* **P13.8.** Implementate l'algoritmo di ordinamento per radice (*radix sort*), descritto nell'Esercizio R13.26, limitatamente al caso dell'ordinamento di array di numeri interi compresi tra 0 e 999, usando, però, un solo array ausiliario invece di usarne dieci.

** **P13.9.** Implementate l'algoritmo di ordinamento per radice (*radix sort*), descritto nell'Esercizio R13.26, per ordinare array di valori di tipo `int` qualsiasi (positivi o negativi).

*** **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.

Capitolo 14

** **P14.1.** Leggete tutte le parole contenute in un elenco e aggiungetele a una mappa le cui chiavi sono le sequenze di tasti della tastiera telefonica che corrispondono alla parola e i cui valori sono gli insiemi di parole che condividono lo stesso codice. Ad esempio, 26337 corrisponde all'insieme `{ "Andes", "coder", "codes", . . . }`. Poi, chiedete ripetutamente all'utente di fornire un numero e visualizzate le parole corrispondenti presenti nell'elenco. Usate anche una mappa che a ogni lettera associ la cifra corrispondente sulla tastiera.

*** **P14.2.** Risolvete nuovamente l'Esercizio E14.4 in modo che le chiavi della mappa siano oggetti di tipo `Student`. Uno studente è caratterizzato da un nome, un cognome e un numero intero che funge da identificatore univoco (ID). Per modificare o eliminare un voto, le ricerche avvengono tramite ID. Nelle visualizzazioni di dati, gli studenti devono essere ordinati per cognome, usando il nome per dirimere le situazioni di parità. Se due studenti hanno lo stesso nome, oltre che lo stesso cognome, li si ordina usando l'identificatore ID. *Suggerimento*: usate due mappe.

*** **P14.3.** Progettate una classe `Polynomial` che memorizzi in una lista concatenata di termini un polinomio del tipo:

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

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 a un costruttore che crei un polinomio a partire da un singolo termine. Il polinomio p, ad esempio, potrebbe essere costruito in questo modo:

```
Polynomial p = new Polynomial(new Term(-10, 0));
p.add(new Polynomial(new Term(-1, 1)));
p.add(new Polynomial(new Term(9, 7)));
p.add(new Polynomial(new Term(5, 10)));
```

Successivamente calcolate $p(x) \times p(x)$.

```
Polynomial q = p.multiply(p);
q.print();
```

*** **P14.4.** Risolvete nuovamente l'esercizio precedente, usando un oggetto di tipo `Map<Integer, Double>` per memorizzare i coefficienti.

** **P14.5.** Cercate di trovare, all'interno di un file di testo di grandi dimensioni, due parole che abbiano lo stesso codice di hash, usando un oggetto di tipo `Map<Integer, HashSet<String>>`. Dopo aver letto una parola dal file, calcolatene il codice di hash, *h*, e inserite la parola nell'insieme associato alla chiave *h*. Al termine, esaminate tutte le chiavi e visualizzate gli insiemi la cui dimensione è maggiore di uno.

** **P14.6.** Realizzate, nella classe `Student` descritta nell'Esercizio P14.2, metodi `hashCode` e `equals` che siano fra loro compatibili. Verificate la correttezza dell'implementazione del metodo `hashCode` aggiungendo oggetti `Student` a un insieme realizzato con tabella hash.

*** **P14.7.** Modificate il valutatore di espressioni visto nel Paragrafo 14.6.3 in modo che sia in grado di convertire in notazione polacca inversa un'espressione algebrica. *Suggerimento*: invece di valutare la cima e impilare il risultato, aggiungete le istruzioni corrispondenti in fondo a una stringa.

* **P14.8.** Risolvete nuovamente l'Esercizio E14.22, usando però una coda.

** **P14.9.** Usate una pila per elencare tutte le permutazioni di una stringa. Ecco un esempio: il calcolo delle permutazioni della stringa `meat`.

```
Impila la stringa +meat.
Finché la pila non è vuota
   Estrai la cima della pila.
   Se tale stringa termina con + (come tame+)
      Elimina il + e aggiungi la stringa all'elenco.
   Altrimenti
      Elimina a turno una delle lettere a destra di +.
      Inseriscila subito prima di +.
      Impila la stringa risultante.
```

Ad esempio, dopo aver estratto `e+mta`, si impila `em+ta`, `et+ma` e `ea+mt`.

** **P14.10.** Risolvete nuovamente l'esercizio precedente, usando però una coda.

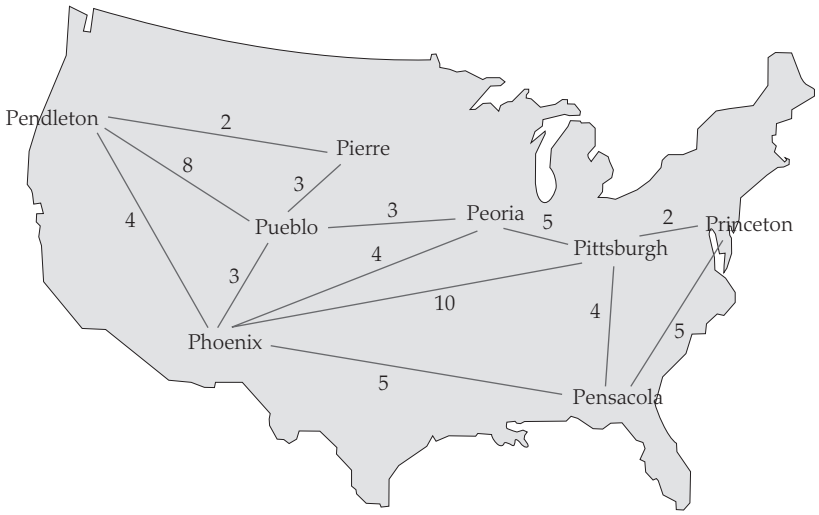
** **P14.11 (economia).** Un aeroporto ha un'unica pista. Quando la pista è occupata, gli aerei che vogliono atterrare o decollare devono attendere. Realizzate un programma di simulazione usando due code, una per gli aerei in attesa di decollare e una per quelli in attesa di atterrare. Gli aerei in attesa di atterrare hanno la precedenza. L'utente del simulatore può digitare i seguenti comandi: `takeoff` *codiceDelVolo*, `land` *codiceDelVolo*, `next` e `quit`. I primi due comandi inseriscono il volo nella coda corrispondente (decollo per `takeoff` e atterraggio per `land`). Il comando `next` pone termine all'operazione in corso (decollo o atterraggio) e fa partire la successiva, visualizzando l'azione da intraprendere (decollo o atterraggio) e il codice del volo. Il comando `quit`, ovviamente, termina la simulazione.

** **P14.12 (economia).** Immaginate di aver acquistato 100 azioni al costo unitario di $12, poi altre 100 a $10 ciascuna, per poi vendere 150 azioni al prezzo unitario di $15. Dovete pagare le tasse sul profitto ottenuto, ma qual è esattamente? Negli Stati Uniti vige una regola FIFO: le prime 100 azioni che vendete sono quelle del primo acquisto, ricavando un profitto di $300, dopodiché vendete 50 azioni provenienti dal secondo acquisto, con un profitto di $250, dando luogo a un profitto complessivo di $550. Scrivete un programma che sia in grado di

eseguire questi calcoli per acquisti e vendite qualsiasi, relativamente ad azioni di un'unica azienda. L'utente può digitare i comandi: `buy` *quantità prezzoUnitario* (per indicare un acquisto), `sell` *quantità prezzoUnitario* (per indicare una vendita, provocando anche la visualizzazione del profitto ottenuto) e `quit` (per terminare). *Suggerimento*: usate una coda di oggetti di tipo `Block`, una classe da progettare i cui oggetti contengono la quantità e il prezzo unitario di azioni acquistate o vendute in blocco.

*** **P14.13 (economia).** Estendete il programma progettato come soluzione dell'esercizio precedente in modo che possa gestire azioni di più aziende. I comandi possibili per l'utente sono ora `buy` *codiceAzienda quantità prezzoUnitario* e `sell` *codiceAzienda quantità prezzoUnitario*. *Suggerimento*: usate una `Map<String, Queue<Block>>` che gestisca una coda distinta per ogni azienda.

*** **P14.14 (economia).** Considerate il problema di trovare il percorso più economico che colleghi tutte le città di una rete stradale, a partire da un punto assegnato. Ad esempio, nella rete qui rappresentata in figura, il percorso più economico tra Pendleton e Peoria ha costo 8 (passando per Pierre e Pueblo).



La classe ausiliaria che segue esprime la distanza tra il punto di partenza e un'altra città.

```
public class DistanceTo implements Comparable<DistanceTo>
{
   private String target;
   private int distance;

   public DistanceTo(String city, int dist) { target = city; distance = dist; }
   public String getTarget() { return target; }
   public int getDistance() { return distance; }
   public int compareTo(DistanceTo other) { return distance - other.distance; }
}
```

Tutte le connessioni dirette tra due città sono memorizzate in una mappa di tipo `Map<String, HashSet<DistanceTo>>` e l'algoritmo funziona in questo modo:

Sia from il punto di partenza.
Aggiungi DistanceTo(from, 0) a q, una coda prioritaria vuota.

```
Costruisci la mappa shortestKnownDistance che mette in corrispondenza nomi di città (le chiavi) e distanze.
Finché q non è vuota
   Estrai da q l'elemento minimo.
   Se il suo target non è una chiave in shortestKnownDistance
      Sia d la distanza dalla città target.
      Inserisci la coppia (target, d) in shortestKnownDistance.
      Per ogni città c che ha una connessione diretta con target
         Aggiungi a q DistanceTo(c, d + distanza tra target e c).
```

Quando l'algoritmo termina, la mappa `shortestKnownDistance` contiene la distanza minima tra il punto di partenza e tutte le città raggiungibili.

Il vostro compito è quello di scrivere un programma che implementi questo algoritmo, leggendo righe del tipo *citta*1 *citta*2 *distanza*. Il punto di partenza è la prima città della prima riga. Visualizzate le distanze minime verso tutte le altre città.

Capitolo 15

** **P15.1.** Scrivete un metodo statico generico, `PairUtil.minmax`, che trovi gli elementi minimo e massimo presenti in un array con elementi di tipo `T`, restituendo una coppia contenente tali valori. Fate in modo che gli elementi dell'array implementino necessariamente l'interfaccia `Measurable` vista nel Paragrafo 10.1.2.

** **P15.2.** Risolvete nuovamente l'esercizio precedente, rendendo però necessario che gli elementi dell'array implementino l'interfaccia `Comparable`.

*** **P15.3.** Risolvete nuovamente l'Esercizio P15.1, facendo in modo che il vincolo espresso sul tipo parametrico richieda che questo estenda il tipo generico `Comparable`.

** **P15.4.** Rendete generica l'interfaccia `Measurable` vista nel Paragrafo 10.1.2. Dotatela di un metodo statico che restituisca l'elemento di misura maggiore presente in un oggetto di tipo `ArrayList<T>`, con la condizione che contenga soltanto elementi che siano esemplare di `Measurable<T>`. Assicuratevi che il metodo restituisca un valore di tipo `T`.

*** **P15.5.** Migliorate la soluzione dell'esercizio precedente in modo che gli elementi del vettore di tipo `ArrayList<T>` possano implementare l'interfaccia `Measurable<U>`, con tipi `U` adeguati.

** **P15.6.** Usando l'interfaccia `java.util.function.Predicate`, scrivete un metodo statico generico con questa firma:

```
List<T> filter(List<T> values, Predicate<? super T> p)
```

che restituisca una lista contenente i valori per i quali la funzione predicativa restituisce `true`. Illustrate il funzionamento di tale metodo costruendo una lista con tutte le stringhe aventi lunghezza maggiore di dieci a partire da una lista di stringhe data. Usate un'espressione lambda (descritta nella sezione Argomenti avanzati 10.4).

** **P15.7.** Usando l'interfaccia `java.util.function.Function`, scrivete un metodo statico generico con questa firma:

```
List<R> map(List<T> values, Function<T, R> f)
```

che restituisca una lista contenente i valori restituiti dalla funzione `f` quando viene invocata con argomenti presenti nella lista `values`.

** **P15.8.** Usando l'interfaccia `java.util.function.Function`, scrivete un metodo statico generico con questa firma:

```
List<Pair<T, R>> map(List<T> values, Function<T, R> f)
```

che restituisca una lista contenente coppie di tipo `(v, f.apply(v))`, dove v è uno dei valori presenti nella lista `values`.