Esercizi per il Corso di Metodologie di Programmazione

Capitolo 8

Università degli Studi di Roma "La Sapienza"

Esercizio 1

Realizzare la classe *Coin* così descritta:

- **Attributi:**
  - *value*: di tipo *double*, contiene il valore della moneta
  - *scoresSize*: di tipo *String*, contiene il nome della moneta
- **Costruttore:** accetta due parametri che indicano rispettivamente il valore ed il nome della moneta.
- **Metodi:**
  - *getValue*: ritorna il valore della moneta
  - *getName*: ritorna il nome della moneta

Realizzare successivamente la classe *CashRegister* così strutturata:

- **Attributi:**
  - *purchase*: di tipo *double*, contiene l'importo da pagare
  - *payment*: di tipo *double*, contiene il totale pagato
- **Costruttore:** imposta i due attributi a 0.
- **Metodi:**
  - *recordPurchase*: dato un parametro che rappresenta il prezzo di un oggetto acquistato, il totale da pagare viene aumentato di tale importo
  - *enterPayment*: dati due parametri che rappresentano il numero di monete ed il loro tipo, il totale pagato viene aumentato di tale importo
  - *giveChange*: calcola il resto come la differenza tra *payment* e *purchase* e reimposta il registratore di cassa per il prossimo cliente. Il valore del resto dovrà essere ritornato dalla funzione al chiamante

Esempio: contenuto nel file *example_1.java*.

Esercizio 2

Modificare il metodo *giveChange* della classe *CashRegister* in modo tale che assuma la seguente firma: *int giveChange(Coin coinType).* Tale metodo dovrà fornire il resto secondo il taglio delle monete specificato nel parametro.
Esempio: contenuto nei file *example_2.java.*

Esercizio 3

I veri registratori di cassa possono gestire sia le monete che le banconote. Progettare un'unica classe che rappresenti ciò che accomuna questi due concetti. Successivamente, riprogettare la classe *CashRegister* dell'esercizio 1 e dotarla di un metodo che consenta la registrazione di pagamenti descritti da tale classe. Uno degli obiettivi principali è l'identificazione di un nome significativo per la classe.
Esempio: contenuto nel file *example_3.java.*

Esercizio 4

Creare una classe chiamata *Reader* che permetta la lettura di una frase, inserita in input dall'utente, e la stampa di ciascuna parola contenuta al suo interno.
Esempio: contenuto nel file *example_4.java.* Guardando con attenzione questo esempio, si può notare come la lettura della parola non avvenga esplicitamente nel *main.* Ciò è dovuto al costruttore della classe *Reader* che si occuperà di leggere lo *stream* di input. Una sua possibile implementazione prevede l'istanziazione di un oggetto di tipo *Scanner* il cui parametro è l'attributo statico *in* della classe *System.* Uno schema riassuntivo dell'implementazione secondo questi criteri è il seguente:

```
public class Reader
{
    private Scanner in;
    private String current;

    public Reader()
    {
        in = new Scanner(System.in);
        current = "";
        if (in.hasNext())
        {
            current = in.next();
        }
    }

    // Implementazione dei metodi rimanenti
}
```

Esercizio 5

Riprogettare la classe *BankAccount* (Capitolo 3, Esercizio 6) in modo che sia immutabile: i metodi devono restituire nuovi oggetti *BankAccount* aventi il saldo opportuno.
Esempio: contenuto nel file *example_5.java*. Oltre alla gestione dell'immutabilità, la nuova versione di *BankAccount* conterrà un nuovo attributo che rappresenta il numero di conto corrente.

Esercizio 6

Dato il file *example_6.java*, ricostruire la classe *Day*.
Consiglio: questa classe è stata vista in una sua versione più semplice nelle esercitazioni precedenti.

Esercizio 7

Progettare i seguenti metodi statici per calcolare il volume e l'area della superficie di un cubo con altezza h, di una sfera con raggio r, di un cilindro con altezza h e base circolare di raggio r e di un cono con altezza h e base circolare di raggio r. Inserirli nella classe *Geometry* e scrivere un programma che chieda all'utente di inserire i valori per r e per h, che invochi gli otto metodi e che visualizzi i risultati:

- *public static double cubeVolume(double h)*
- *public static double cubeSurface(double h)*
- *public static double sphereVolume(double r)*
- *public static double sphereSurface(double r)*
- *public static double cylinderVolume(double r, double h)*
- *public static double cylinderSurface(double r, double*
- *h)*
- *public static double coneVolume(double r, double h)*
- *public static double coneSurface(double r, double h)*

Esempio: contenuto nel file *example_7.txt*

Esercizio 8

Risolvete nuovamente l'esercizio precedente realizzando le classi *Cube*, *Sphere*, *Cylinder* e *Cone*. Quale approccio tra i due esercizi è maggiormente orientato agli oggetti?

Esercizio 9

Creare un programma che rappresenti un registro di classe. Esso dovrà:

- Memorizzare i nomi degli studenti
- Memorizzare i voti di ciascun studente, rappresentati con numeri che vanno da 1 a 10.

Per testare il corretto funzionamento sarà necessario chiedere all'utente i nomi di tutti gli studenti. Successivamente, dovranno essere letti i voti di tutte le singole prove d'esame, una dopo l'altra, chiedendo il punteggio di ciascuno studente (visualizzandone il nome); infine, visualizzare i nomi di tutti gli studenti, con il relativo voto finale.
NB: usare un'unica classe che contenga soltanto metodi statici.
Esempio: contenuto nel file *example_9.txt*

Esercizio 10

Risolvete nuovamente l'esercizio precedente usando più classi: progettate la classe *GradeBook* (registro delle valutazioni) in modo che memorizzi un insieme di oggetti di tipo *Student*.
Esempio: il funzionamento dovrà essere lo stesso dell'esercizio precedente, pertanto l'esempio presente in *example_9.txt* potrà essere riapplicato.

Esercizio 11

Partendo dalla classe *Picture* vista negli Esercizi 14 e 15 del Capitolo 6, creare la classe *PictureUtilities* che conterrà il metodo *public static Picture superimpose(Picture pici, Picture pic2)*. Esso sovrapporrà due immagini, generando un'immagine le cui dimensioni (larghezza e altezza) siano uguali al valore massimo tra le dimensioni corrispondenti delle due immagini *pic1* e *pic2*. Nell'area in cui le due immagini si sovrappongono, calcolare la media tra i colori, pixel per pixel.
Esempio: contenuto nella cartella *es_11*. Al suo interno troverete le classi *JFrame* e *Picture*, usate nel capitolo precedente, insieme alla classe *PictureViewer*. Le immagini *pic1.jpg* e *pic2.jpg*, se usate come input per il programma, produrranno il risultato contenuto in *frame.png*.

Esercizio 12

Partendo nuovamente dalla classe *Picture* vista negli Esercizi 14 e 15 del Capitolo 6, creare la classe *PictureUtilities* che conterrà il metodo *public static Picture greenScreen(Picture pici, Picture pic2)* che sovrappone due immagini, generando un'immagine le cui dimensioni (larghezza e altezza) siano uguali al valore massimo tra le dimensioni corrispondenti delle due immagini *pic1* e *pic2*. Nell'area in cui le due immagini si sovrappongono, usare i pixel di *pic1*, tranne quando sono verdi. In quel caso, usare i pixel di *pic2*.
Esempio: presente nella cartella *es_12*. Per decretare se un pixel sia trattabile come verde, si può calcolare la media dei tre parametri RGB per tale pixel. Successivamente, si può verificare se il parametro *Green* sia maggiore di tale media, trattandolo come se fosse verde in caso affermativo. Un esempio di codice per questo passaggio può essere il seguente:

```
Color c = pic1.getColorAt(x, y);
int avg = (c.getRed() + c.getGreen() + c.getBlue()) / 3;
if (c.getGreen() > avg * 1.25)
{
    result.setColorAt(x, y, pic2.getColorAt(x, y));
```

```
}
else
{
    result.setColorAt(x, y, pic1.getColorAt(x, y));
}
```

Esercizio 13

Creare la classe *Input* che contiene il metodo *public static int readlnt(Scanner in, String prompt, String error, int min, int max).* Tale metodo dovrà visualizzare un messaggio (prompt), leggere un numero intero e verificare se si trovi tra il valore minimo e il valore massimo specificati. In caso negativo, dovrà essere visualizzato un messaggio d'errore (error), ripetendo l'acquisizione del dato.
Esempio: contenuto nel file *example_13.java.*

Esercizio 14

Esaminare il seguente algoritmo per calcolare x^n, con n intero.

- Se $n < 0$, x^n è uguale a $1/x^{-n}$
- Se n è positivo e pari, allora $x^n = (x^{n/2})^2$
- Se n è positivo e dispari, allora $x^n = x^{n-1} \cdot x$

Successivamente, creare il metodo statico *double intPower(double x, in t n)* che applichi questo algoritmo, inserendolo nella classe *Numeric.*
Esempio: per testare il corretto funzionamento della classe, utilizzare *example_14.java.*

Esercizio 15

Aggiungere alla classe *BankAccount* il metodo *ArrayList<Double> getStatement()* che restituisca un elenco di tutti i versamenti e prelievi sotto forma di valori, rispettivamente, positivi e negativi. Aggiungere anche il metodo *void clearStatement()* che svuoti l'elenco.
Esempio: contenuto nel file *example_15.java.*