Esercitazione sul Polimorfismo
(Fondamenti di Informatica – Emilio Di Giacomo)

Esercizio 1. Obiettivo di questa esercitazione è realizzare un semplice programma che permette di manipolare delle immagini (cioè dei file jpg o png) applicando delle trasformazioni analoghe a quelle che si trovano di solito nei programmi di fotoritocco. Il programma utilizzerà le classi **Colore**, **Immagine**, **ManipolatoreDiImmagine** e l'interface **Trasformazione** che sono state già scritte e che vengono descritte di seguito.

La classe Colore.

Ogni oggetto della classe **Colore** è un colore rappresentato tramite le tre componenti RGB; la classe è dotata dei seguenti costruttori e metodi:

> **public Colore(int red, int green, int blue):** costruisce un oggetto che rappresenta un colore le cui componenti RGB sono tre numeri interi nell'intervallo [0,255] che rappresentano rispettivamente la component rossa, verde e blu del colore.
>
> **public int getRed():** restituisce la componente rossa del colore.
>
> **public int getGreen():** restituisce la componente verde del colore.
>
> **public int getBlue():** restituisce la componente blu del colore.
>
> **public boolean equals(Colore c):** restituisce **true** se l'oggetto ricevente e quello passato come parametro rappresentano lo stesso colore, **false** in caso contrario.

La classe Immagine.

Ogni oggetto della classe **Immagine** è un'immagine ed è rappresentato mediante una matrice di pixel ognuno dotato di un colore. La classe è dotata dei seguenti costruttori e metodi:

> **public Immagine(int larghezza, int altezza):** costruisce un immagine bianca le cui dimensioni sono quelle specificate.
>
> **public Colore getPixel(int i, int j):** restituisce il colore del pixel in posizione (**i**,**j**), con $0 \leq \mathbf{i} < \mathbf{a}$ e $0 \leq \mathbf{i} < \mathbf{l}$ essendo **a** l'altezza dell'immagine e **l** la sua larghezza.
>
> **public void setPixel(int i, int j, Colore c):** assegna al pixel (**i**,**j**) il colore **c**. Anche in questo caso $0 \leq \mathbf{i} < \mathbf{a}$ e $0 \leq \mathbf{i} < \mathbf{l}$ essendo **a** l'altezza dell'immagine e **l** la sua larghezza.
>
> **public int larghezza():** restituisce la larghezza dell'immagine.
>
> **public int altezza():** restituisce l'altezza dell'immagine.

L'interface Trasformazione.

L'interface **Trasformazione** è così definita:

```
public interface Trasformazione {

     public Immagine trasforma(Immagine img);

}
```

Il metodo **trasforma(Immagine img)**: riceve come parametro un oggetto **Immagine** e restituisce un'altra **Immagine** ottenuta da quella data applicando una certa trasformazione. Implementando l'interface **Trasformazione** e sovrascrivendo il metodo **trasforma** è possibile definire diversi tipi di trasformazioni

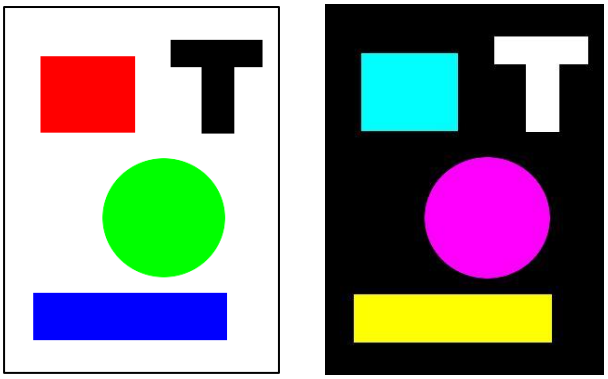
La classe ManipolatoreDiImmagine.

Ogni oggetto della classe **ManipolatoreDiImmagine** è un oggetto in grado di applicare una sequenza di trasformazioni ad un'immagine data. La classe è dotata dei seguenti costruttori e metodi:

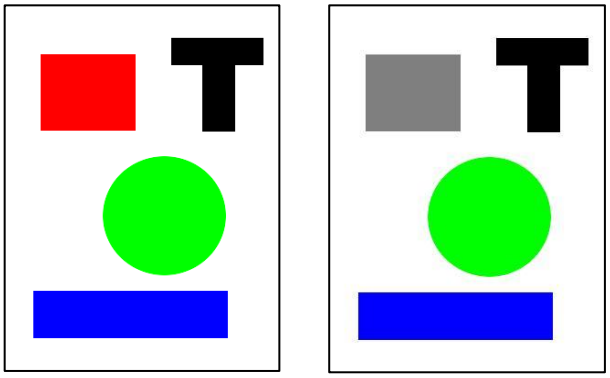
> **public ManipolatoreDiImmagine(String filename)**: crea un **ManipolatoreDiImmagine** associato all'immagine memorizzata nel file il cui percorso è passato come parametro al costruttore.
>
> **public void aggiungiTrasformazione(Trasformazione t)**: aggiunge una trasformazione alla sequenza di trasformazioni da applicare all'immagine. Le trasformazioni vengono applicate nell'ordine in cui vengono aggiunte.
>
> **public void trasforma()**: applica tutte le trasformazioni della sequenza all'immagine associata all'oggetto ricevente, che viene modificata di conseguenza.
>
> **public void scriviSuFile(String fileName)**: scrive l'immagine associate all'oggetto ricevente sul file il cui percorso è passato come parametro.

Si scrivano due classi a scelta tra le seguenti. Ogni classe realizza una trasformazione da applicare ad un'immagine.

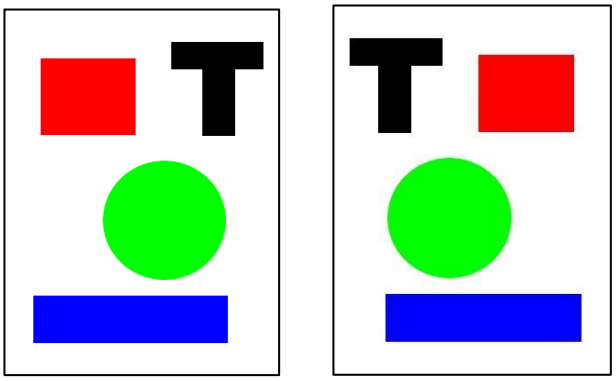
1. La classe **Negativo** realizza una trasformazione che converte un'immagine nel suo negativo. La classe non ha attributi e il suo metodo **trasforma** cambia il colore (**r**,**g**,**b**) di ciascun pixel nel colore (255-**r**,255-**g**,255-**b**). Esempio di applicazione della trasformazione **Negativo**:



2. La classe **CambioColore** realizza una trasformazione che cambia il colore a tutti pixel di un certo colore dato. La classe ha come attributi il colore **c1** da sostituire e il colore **c2** con il quale sostituirlo. Il costruttore della classe riceve come parametro i due colori. Il metodo **trasforma** cambia il colore di ciascun pixel di colore **c1** nel colore **c2**. Esempio di applicazione della trasformazione **CambioColore** in cui il colore rosso è sostituito dal grigio:

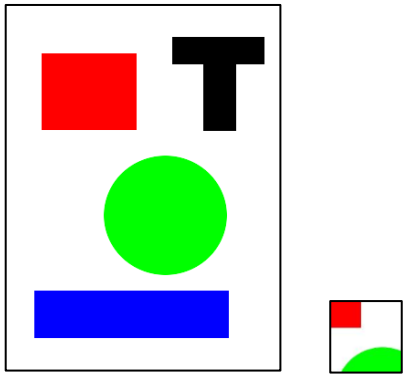


3. La classe **Riflessione** realizza una trasformazione che riflette orizzontalmente o verticalmente l'immagine. La classe ha un attributo **orizzontale** che è true se la riflessione è orizzontale, false in caso contrario. Il costruttore della classe riceve come parametro il valore da assegnare all'attributo **orizzontale**. Il metodo **trasforma** riflette l'immagine. Esempio di applicazione della trasformazione **Riflessione** in cui l'immagine è riflessa orizzontalmente:

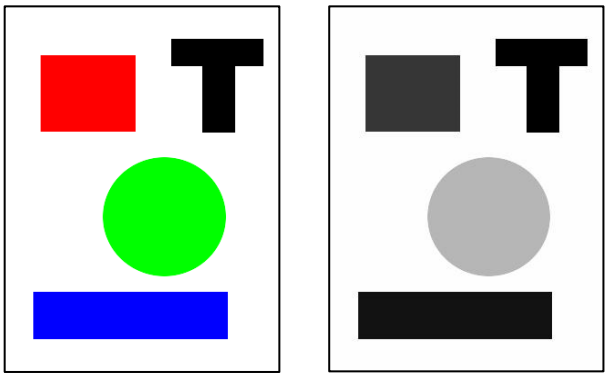


4. La classe **Taglio** realizza una trasformazione che ritaglia una porzione dell'immagine. La classe ha come attributi due coppie di indici (**i1**,**j1**) e (**i2**,**j2**) (con **i1** ≤ **i2** e **j1** ≤ **j2**) che delimitano l'area dell'immagine da ritagliare. Il costruttore della classe riceve come parametro i quattro indici. Il metodo **trasforma** restituisce un'immagine che è la porzione dell'immagine

di partenza che contiene tutti i pixel (**i**,**j**) con **i1** ≤ **i** ≤ **i2** e **i1** ≤ **i** ≤ **i2.** Esempio di applicazione della trasformazione **Taglio**:



5. La classe **BiancoENero** realizza una trasformazione che converte l'immagine in scala di grigi. Il metodo **trasforma** cambia il colore (**r**,**g**,**b**) di ciascun pixel nel colore (**l**,**l**,**l**) dove **l** è il valore di luminosità del pixel calcolato come 0.2126***r**+0.7152***g**+0.0722***b** (valore che deve essere convertito a intero per ottenere **l**).



Si scriva poi una classe **ProvaManipolatoreDiImmagine** che contiene il solo metodo **main** che svolge le seguenti azioni:

- Crea un oggetto **ManipolatoreDiImmagine m** associato ad un'immagine presente nel file system.
- Aggiunge ad **m** le trasformazioni realizzate.
- Applica tutte le trasformazioni.
- Scrive l'immagine risultato su un file.

Esercizio 2. Si scrivano le classi che non si scelto di realizzare nell'Esercizio 1 e si modifichi la classe **ProvaManipolatoreDiImmagine** in modo da testare le nuove classi create.

Esercizio 3. Si vuole creare una classe **Sequenza** per rappresentare sequenze di oggetti qualsiasi. Una sequenza è un insieme ordinato di elementi. Ogni elemento ha cioè una posizione e quindi un predecessore e un successore (eccetto per il primo e l'ultimo elemento). La classe ha i seguenti costruttori e metodi:

public Sequenza(int dimMax): costruisce una sequenza in grado di contenere al più **dimMax** elementi.

public int lunghezza(): restituisce la lunghezza della sequenza, cioè il numero di elementi in essa memeorizzati.

public boolean inserisci(Object o): inserisce l'oggetto **o** nella sequenza come ultimo elemento. Se la sequenza contiene già il numero massimo di elementi, l'oggetto non viene inserito. Restituisce **true** se l'oggetto è stato inserito, **false** in caso contrario.

public int contiene(Object o): restituisce la posizione dell'oggetto **o** nella sequenza (la posizione è un indice tra **0** e **lunghezza()-1**). Se l'oggetto non è presente nella sequenza restituisce -1.

public boolean rimuovi(Object o): rimuove l'oggetto **o** dalla sequenza. Se l'oggetto non è presente nella sequenza la sequenza rimane inalterata. Restituisce **true** se l'oggetto è stato rimosso, **false** in caso contrario.

Implementare la classe **Sequenza** e scrivere poi una classe **ProvaSequenza** per testarla.

Nota: si consiglia di utilizzare un array di Object per memorizzare gli elementi della sequenza.