

**Corso di Laurea in Ingegneria Informatica e Automatica
A. A. 2018/2019**

Progettazione del Software

**Giuseppe De Giacomo, Paolo Liberatore,
Massimo Mecella**

Esercitazione 4



**APPLICAZIONE FIGURE
GEOMETRICHE**



Figure geometriche

Realizzare le classi necessarie per modellare figure geometriche. Nel nostro dominio ci interessano le seguenti figure geometriche: rettangoli, quadrati, cerchi, ellissi, e triangoli.

Definire un insieme di classi e una opportuna gerarchia partendo da una classe base [FiguraGeometrica](#).



Figure geometriche

Ogni figura geometrica è caratterizzata da una propria descrizione (una stringa) e da campi dati che consentono il calcolo dell'area.

Ogni figura geometrica deve avere dei costruttori per inizializzare i valori delle proprietà e deve implementare le seguenti operazioni

- [double area\(\)](#) : restituisce l'area della figura geometrica
- [double perimetro\(\)](#) : restituisce il perimetro della figura geometrica
- [String toString\(\)](#) : per la rappresentazione come stringa delle informazioni della figura geometrica (descrizione e dati)



Figure geometriche

Scrivere un metodo statico

```
double sommaAree(FiguraGeometrica[] f)
```

che, dato un array di figure geometriche **f** restituisce la somma delle aree di tutte le figure geometriche in **f**

Scrivere un programma Java che crea e inizializza un array di figure geometriche di diverso tipo (con valori random o costanti) e calcola la somma delle aree di tali figure geometriche.



API delle classi

Figura Geometrica

```
// proprietà  
descrizione  
// operazioni  
costruttori  
get/set  
// servizio  
toString()  
area(): [n.b. l'area di una figura generica è 0]
```



API delle classi

Triangolo

```
// proprietà  
l 3 lati (immutabile)  
descrizione  
// operazioni  
costruttori  
get/set  
// servizio  
area()  
toString()
```



API delle classi

Rettangolo

```
// proprietà  
base (immutabile)  
altezza (immutabile)  
descrizione  
// operazioni  
costruttori  
get/set  
// servizio  
area()  
toString()
```



API delle classi

Quadrato

```
// proprietà  
base (immutabile)  
altezza (immutabile)  
descrizione  
// operazioni  
costruttori [n.b. base e altezza sono uguali]  
get/set  
// servizio  
area()  
toString()
```



API delle classi

Ellisse

```
// proprietà  
semiasse maggiore (immutabile)  
semiasse minore (immutabile)  
descrizione  
// operazioni  
costruttori  
get/set  
// servizio  
area()  
toString()
```



API delle classi

Cerchio

```
// proprietà  
semiasse maggiore (immutabile)  
semiasse minore (immutabile)  
descrizione  
// operazioni  
costruttori [n.b. i due semiasse sono uguali]  
get/set  
// servizio  
area()  
toString()
```



SISTEMA BANCARIO



Sistema bancario

Realizzare le classi necessarie per rappresentare il seguente dominio applicativo. Una `Banca` è un insieme ordinato di oggetti `ContoCorrente`. Di una banca interessa conoscere il nome e l'indirizzo (entrambe stringhe). Un conto corrente (i cui dettagli sono riportati in seguito) deve essere necessariamente di uno di due tipi : di debito o di credito.

L'applicazione deve permettere di gestire la banca come insieme ordinato di conti, la contabilità dei vari conti, e di stampare degli opportuni riepiloghi della situazione finanziaria dei conti e della banca



ContoCorrente

Un conto corrente è caratterizzato da un codice (stringa, unico per il conto), dal saldo (intero), dal nome e cognome del proprietario (entrambi stringhe).

Inoltre un conto corrente offre due operazioni per fare il deposito ed il prelievo dal conto

- `public void deposito(int cifra)`
- `public void prelievo (int cifra)`

Un conto corrente viene creato inizializzando il suo codice e il saldo iniziale. Il nome e cognome del proprietario possono invece essere modificati successivamente ed in più occasioni



ContoCredito

Un `ContoCredito` è un particolare tipo di `ContoCorrente`, in cui viene pagata una commissione ad ogni operazione se il numero di operazioni effettuate supera una soglia prefissata

- la soglia (un intero) può essere rappresentata come una costante
- il costo della commissione (da sottrarre dal saldo ad ogni operazione oltre soglia) è una caratteristica del conto, che deve poter essere letta/modificata in ogni momento. Alla creazione di un conto si assume essere 0
- Deve essere disponibile un'operazione per azzerare il numero delle operazioni effettuate
 - `void reset()`



ContoDebito

Un `ContoDebito` è un particolare tipo di `ContoCorrente`, in cui vengono riconosciuti degli interessi (che si sommano al saldo attuale) quando ritenuto opportuno

- `void riconosciInteresse(double interesse)`
prende un valore di interesse – nel range `[0..1]` e lo applica al saldo attuale, aumentando il saldo della cifra così ottenuta (con arrotondamento)
Ad es., se il saldo attuale è 50, `riconosciInteresse(0.65)` fa sì che il nuovo saldo sia 82



Comparable e Main

Si vuole poter confrontare due conti correnti in base al loro saldo, cosa necessaria anche per il mantenimento dell'ordinamento dei conti correnti nella banca

Creare l'applicazione con il metodo `main` in modo che crei almeno 3 conti (di tipi differenti) e dopo averli operati, stampi la situazione finanziaria della banca. Verificare come la banca rimane ordinata, facendo varie prove che cambiano l'ordine di inserimento dei conti nella banca e l'ordine con cui l'operatività del conto si succede all'inserimento nella banca

