LABORATORIO DI PROGRAMMAZIONE
CORSO DI LAUREA IN SICUREZZA DEI SISTEMI E DELLE RETI INFORMATICHE
UNIVERSITÀ DEGLI STUDI DI MILANO
2025–2026

Indice

Ultima revisione: 6 dicembre 2025.

Parte 1. Programmazione orientata agli oggetti

Create una cartella diversa per ogni esercizio. Se usate VS Code, aprite VS Code volta per volta sulla cartella dell'esercizio su cui state lavorando. Prestate attenzione al fatto che alcuni esercizio richiedono di lavorare sullo stesso *progetto*, quindi nella stessa cartella. In generale, usate il concetto di *progetto* per organizzare i vostri file.

Esercizio 1

Implementare e testare una classe fornita.

Tempo: 5 min.

Implementate la classe `Contatore`, definita in Figura 1. Istanziatela e chiamatela da una apposta classe, definita in un file a parte, contenente il `main`.

Potete definire il metodo `toString` e usarlo come strategia di debug, alternativa ad avere delle stampe dei singoli valori degli attributi.

Contatore.java

```
public class Contatore {
        int a;

    public Contatore(){
        a=0;
    }

    public Contatore(int aa){
        a=aa;
    }

    void inc() {
        a=a+1;
    }

    void reset() {
        a=0;
    }
}
```

Figura 1. Classe `Contatore`.

Esercizio 2

Implementare e testare una classe.

Tempo: 10 min.

Scrivete l'ADT di un numero complesso `Complex`, partendo da una versione scritta in C. Scrivete prima le specifiche su carta, poi implementarle.
Modellate lo stato e il comportamento in Java, definendo tutte le funzioni necessarie ad operare sul tipo di dato.
Istanziate e testate la classe `Complex` eseguendo delle operazioni semplici sul dato. Usate un'apposita classe separata, contenente il `main`.

Esercizio 3

Progettare, implementare, e testare una classe.

Tempo: 20 min.

In `Java`, quando dovete definire una o più classi, è bene ragionare *prima* di scrivere il codice su quanti e quali classi avete bisogno e su come devono interagire tra loro. La fase di design iniziale della soluzione è più importante rispetto a quanto avete visto in `C`. Se la fate bene, siete a metà dell'opera. Se la fate male, dovrete continuare a modificare il codice perdendo tempo e introducendo errori.

Scrivete una classe che descriva un vettore `Vett`, che definisce un punto nello spazio 2D. Pensate allo stato e all'interfaccia di tale classe:

Prima di scrivere il codice che implementa una classe, pensate al funzionamento che deve avere e alla sua semantica (il suo significato). Una classe può avere tanti costruttori, pensate a quali possono essere utili (oltre quello di default).

- **stato**: caratteristiche intrinseche della classe.
- **interfaccia**: come interagisce con il resto del mondo
- **costruttori**: senza parametri, a partire da coordinate intere, a partire da un altro punto.

Durante la fase di sviluppo, considerare la visibilità di attributi e metodi della classe. Quali devono essere interni, e quali devono essere richiamabili dall'esterno? Ci sono attributi che devono essere statici e attributi che devono essere protetti da eventuali modifiche non desiderate?

Definite quindi gli attributi e i metodi che la classe `Vett` deve avere. Alcuni esempi sono:

- Possibilità di muovere il punto nello spazio dopo la creazione
- Possibilità di calcolare la distanza dal punto di origine degli assi
- Altre operazioni possono essere necessarie? Definitele e implementatele.

Create un nuovo progetto contenente il main e aggiungete la classe `Vett` a questo progetto.

Quando dovete definire i metodi che caratterizzano un oggetto, dovete chiedervi "Cosa deve fare la classe?". I metodi da implementare sono definiti dalle specifiche del programma, ma anche dalla semantica dell'oggetto stesso. Definire quali sono i metodi che caratterizzano un oggetto è un compito da svolgere prima di scrivere codice, nella fase di design del programma, e poi rifinito in maniera iterativa durante lo sviluppo.

Esercizio 4

Progettare e implementare una classe sulla base di una definita in precedenza.

Tempo: 20 min.

Gli esercizi 3–8 sono pensati per essere svolti insieme. Con questi esercizi, costruirete una piccola libreria per fare semplici operazioni geometriche. Per fare questo, utilizzerete le relazioni tra i singoli oggetti, riutilizzando il codice e i metodi degli esercizi precedenti ed estendendoli.

Scrivete un programma che permetta di calcolare il perimetro di un cerchio dato il raggio. Per farlo, creare una classe `Cerchio`.

Per definire il centro della classe `Cerchio`, utilizzate un `Vett`, definito nell'esercizio 3. Aggiungete metodi a `Vett` se ne avete bisogno.

Aggiungete ulteriori classi che possono servire a definire un oggetto di tipo `Cerchio`. Utilizzare le classi definite negli esercizi precedenti. Progettate con criterio i costruttori di `Cerchio`, i metodi, e gli attributi, considerando la loro visibilità.

Esercizio 5

Modificare una classe.

Tempo: 20 min.

Implementate nuovi metodi per la classe `Cerchio`:

- Calcolo dell'area del cerchio
- Metodo `isConcentric` per valutare se un cerchio è concentrico rispetto ad un altro
- Metodo `contains` che specifica se un cerchio è contenuto in un altro.

Esercizio 6

Progettare e implementare classi che usano classi definite in precedenza.

Tempo: 30 min.

Creare una classe `Segmento` basandosi su `Vett`. Per farlo, aggiungete i metodi necessari a `Vett`. Create *i)* un metodo `equals` per le classi `Vett` e per `Segmento`, che stabilisca se due oggetti – dello stesso tipo – sono equivalenti, e *ii)* un metodo `copy`, che crea una copia di un oggetto identico all'attuale. Fate lo stesso per la classe `Cerchio`.
Testate i metodi sviluppati in un apposito `main`.

Cosa succede quando copio un oggetto che è a sua volta composto da altri oggetti? Ad esempio, se copio il cerchio C, ottenendo il cerchio C', i rispettivi centri P e P' saranno il *medesimo* oggetto `Vett` o due oggetti differenti? Provate a verificarlo, spostando il centro di C e controllando se il centro di C' viene spostato di conseguenza. Come posso fare se voglio rendere un oggetto creato copiandone uno già esistente totalmente indipendente dall'originale? E invece come posso fare se voglio legare le modifiche dell'oggetto copiato a quelle dell'originale?

Esercizio 7

Riuso di classi definite in precedenza.

Tempo: 10 min.

Create una classe `Figura` fatta da 2 cerchi e tre segmenti. Verificare se `Figura` contiene cerchi concentrici tramite apposito `main`. Verificate anche se i tre segmenti costituiscono un triangolo con una funzione apposita.

Esercizio 8

Enumeratori.

Tempo: 10 min.

Create una `enum` che definisca i tipi di triangolo (equilatero, scaleno, etc). Definite un metodo che, dati tre segmenti, verifica se formino un triangolo e, se sì, indichi di che tipo si tratta.

Per implementare questa funzionalità dovete calcolare la lunghezza di ogni lato del triangolo. Di che tipo è? Dove dovete definire quindi il metodo che calcola la lunghezza di un lato?

Esercizio 9

Progettare e implementare una classe.

Tempo: 15 min.

Definite una classe `Frazione`. Come per gli esercizi precedenti, prima di iniziare l'implementazione, pensate al suo stato, alla sua interfaccia. Riflettete su quali sono le operazioni possibile su tale oggetto, e che requisiti anno. Definite ed implementate i suoi costruttori. In particolare, controllate quali sono le condizioni da rifiutare durante la creazione di un nuovo oggetto `Frazione`. Usate il `main` per testare l'interfaccia e verificarne i vincoli.

Usate opportunamente le *eccezioni*.

Esercizio 10

Modificare e testare una classe.

Tempo: 15 min.

Estendete la classe `Frazione`. Fornite i seguenti metodi (e testateli in un apposito `main`):

- Fornite metodo `toString` che la trasforma in stringa per stamparla
- Fornite metodi predicativi che testano maggiore e minore
- Fornite un metodo `equals` per dire se una frazione passata in ingresso è equivalente
- Calcolare la frazione inversa
- Fornite la funzione `semplifica`. Ad esempio, 4/8 va semplificato in 1/2. Per farlo implementate una funzione esterna che calcola il MCD.

I metodi che eseguono una operazione sulla `Frazione` devono restituire un nuovo oggetto; ad esempio, la somma di due frazioni sarà un nuovo oggetto `Frazione`. L'inversa di una frazione sarà un nuovo oggetto `Frazione`.

Il termine funzione *esterna* ci fa pensare al fatto che la funzione non deve essere parte della classe `Frazione`.