LABORATORIO DI PROGRAMMAZIONE
CORSO DI LAUREA IN SICUREZZA INFORMATICA
LEZIONE 15 - ASTRAZIONE ED EREDITARIETÀ
UNIVERSITÀ DEGLI STUDI DI MILANO
2025–2026

Esercizio 1

A partire dalla classe `Vett`, che rappresenta un vettore/punto bidimensionale, estenderla per creare una classe che rappresenti vettori/punti in tre dimensioni. Usare la visibilità `protected` e overriding dei metodi, se possibile.

Esercizio 2

Definire la classe `Rettangolo`. Per questa classe definire i metodi `getter`, `setter`, `equals`, `toString`, ecc. Definire poi la classe `Quadrato`. Che relazione dovete utilizzare per queste due classi?

> Come è definito un rettangolo? Il costruttore deve verificare che tale proprietà sia rispettata. Lo stesso vale per quadrato.

Testarle con un main adatto e facendo delle operazioni semplice, come calcolare area e perimetro. Appoggiarsi sui concetti di `Vett` e `Linea`, nel caso ve ne fosse bisogno.

> Area e perimetro devono essere attributi di rettangolo, e restituiti tramite un getter, o calcolati al momento tramite una apposita funzione? Quali sono gli svantaggi e i vantaggi di entrambe le implementazioni?

Esercizio 3

Definire una classe astratta `Figura`, che racchiuda il concetto ripreso dalle classi definite fino ad ora, come `Cerchio`, `Rettangolo` e `Quadrato`.

> La classe `Rettangolo` può derivare direttamente da `Figura` o, in alternativa, può derivare dalla classe `Quadrilatero`; nel secondo caso, un `Rettangolo` è una versione specifica di `Quadrilatero`. Il vostro programma potrebbe voler gestire diversi tipi di quadrilateri come `Quadrato`, `Parallelogramma`, `Trapezio`. In fase di design del vostro programma, scegliete quali classi rappresentare, quale è la loro gerarchia.

Ultima revisione: 15 dicembre 2025.

Tale classe deve avere dei metodi astratti per il calcolo di perimetro ed area. Testare questa classe mediante un main che chieda all'utente di gestire un array di 10 figure, chiamandone i metodi, e istanziandole secondo quanto inserito dall'utente. Viene chiesto all'utente cosa creare, e quindi verrà istanziato mediante il costruttore relativo l'oggetto richiesto.

Per semplicità, iniziate a definire un `main` in cui la dichiarazione del vettore di figure avvenga senza input da utente (quindi scritto direttamente nel codice).

Esercizio 4

All'esercizio precedente, aggiungete la possibilità di stampare le figure in ordine di dimensione, secondo area o secondo perimetro. A tale riguardo, considerate che gli oggetti possono implementare l'interfaccia `comparable`, che può essere utilizzata per confronare due figure.
Per effettuare l'ordinamento, utilizzare le modalità predefinite su come ordinare array di oggetti confrontabili (es: `java.utils.Array.sort`).
Per stampare l'elenco, fate riferimento a `java.utils.Array.toString`. Testare con un main apposito.

> Per testare il vostro programma, è buona norma generare nuove istanze in maniera procedurale. Per farlo, una opzione è di generare punti ed oggetti in maniera casuale; ad esempio, usate il metodo `SecureRandom` indicato nella scorsa sessione di laboratorio.

Esercizio 5

Progettare e scrivere un programma di gestione del personale di una azienda. Tale programma deve gestire i `Dipendenti` e i `Manager` (che saranno a loro volta dipendenti).
Organizzare tutti i dipendenti come una collezione (array), direttamente nel `main`.

> Progettare i dati necessari a descrivere l'anagrafica di un utente. Quale deve essere la loro visibilità?

> Un attributo come la data di nascita è immutabile; in questo caso è bene definire tali campi come `final`, per garantire il principio del *minimo privilegio*: ogni modulo computazionale dovrebbe avere visibilità delle sole risorse/informazioni immediatamente necessarie al suo funzionamento. Questo è utile per evitare modifiche accidentali a tali campi.

Decidete se utilizzare (o meno) una classe astratta, o una interfaccia `Persona`. Motivate la scelta. Create un main che permette di aggiungere dei dipendenti all'azienda, il loro salario.

Esercizio 6

Estendere l'esercizio precedente aggiungendo una entità `Società`. In che relazione è con i suoi dipendenti?

Fare in modo che i dipendenti possano essere ordinati e stampati per salario percepito. Per stampare il record relativo ad un dipendente utilizzate il metodo `toString`, da voi implementato.

Esercizio 7

A partire dal metodo scritto in C, implementate una funzione che effettui un `BubbleSort` di un array di tipo `Object` in Java. Potete implementarlo come metodo statico o definendo una classe wrapper che faccia da contenitore per un oggetto generico.

Testare con un main.

Esercizio 8

Create la classe `IntegerSet`. Ogni sua istanza rappresenta un insieme di interi nell'intervallo $0-100$, rappresentato internamente per mezzo di un array di boolean. L'elemento `a[i]` dell'array è vero se l'intero `i` fa parte dell'insieme. L'elemento `a[j]` è falso se l'intero `j` non fa parte dell'insieme. Il costruttore senza argomenti inizializza l'array all'insieme vuoto (l'array conterrà quindi tutti valori `false`).

Fornite i seguenti metodi:

- Il metodo statico `union` crea un terzo insieme che è l'unione insiemistica di due insiemi esistenti (ogni elemento dell'array del terzo insieme è impostato a true se quell'elemento vale true in uno o in entrambi gli insiemi esistenti).
- Il metodo statico `intersection` crea un terzo insieme che è l'intersezione insiemistica di due insiemi preesistenti (ogni elemento dell'array del terzo insieme è impostato a `false` se l'elemento è `false` in uno o entrambi gli insiemi esistenti, in caso contrario vale `true`).
- Il metodo `insertElement` inserisce un nuovo intero `k` nell'insieme (impostando `a[k]` a `true`).
- Il metodo `deleteElement` rimuove un intero `m` dall'insieme (impostando `a[m]` a `false`).
- Il metodo `toString` restituisce una stringa che rappresenta l'insieme come lista di numeri separati da spazi. Naturalmente dovranno essere inclusi solo gli elementi presenti nell'insieme: usate `--` per l'insieme vuoto.
- Il metodo `isEqualTo` determina se due insiemi sono uguali.

Scrivete un programma che verifichi il funzionamento della classe `IntegerSet`. Istanziate vari oggetti `IntegerSet` e verificate che i metodi funzionino correttamente. Per generare numeri casuali da inserire all'interno di un `IntegerSet` utilizzare la funzione vista nei laboratori scorsi.

Esercizio 9

Estendete la classe definita all'esercizio precedente creando la classe `RealSet`. Tale classe deve accettare con input dei numeri reali compresi tra $0 - 100$ e aggiornare la casella corrispettiva dell'esercizio precedente. Ad esempio, se il valore inserito è 3.5, dovrà essere aggiornata il valore di `a[3]`.