Esercitazione di FdP-B

Lunedì, ore 15:30 - 17:30

Merenda Saverio Mattia, 05/05/2025

Esercizio 1

Gestione Magazzino in Java

Interfaccia Catalogabile: definisce la struttura degli oggetti che possono essere catalogati nel magazzino

- Un metodo **String getCodiceUnivoco()** che restituisce un codice identificativo univoco per l'articolo.
- Un metodo **String getDescrizione()** che restituisce una descrizione testuale dell'articolo.
- Un metodo **double getValore()** che restituisce il valore (e.g., prezzo) dell'articolo.

Gestione Magazzino in Java

Classe ProdottoFisico: implementa l'interfaccia Catalogabile per rappresentare prodotti fisici.

- Deve implementare l'interfaccia **Catalogabile**.
- Deve includere attributi per **peso** (double) e **dimensioni** (String).
- Un costruttore che accetta codice, descrizione, valore, peso e dimensioni.
- Implementazione dei metodi **getCodiceUnivoco()**, **getDescrizione()**, **getValore()** (restituisce il prezzo).
- Ridefinizione del metodo **boolean equals(Object obj)**: due **ProdottoFisico** sono uguali se hanno lo stesso **codiceUnivoco**.

Gestione Magazzino in Java

Classe Generica Magazzino<T extends Catalogabile>: implementare un contenitore generico per articoli di tipo T, dove T implementa Catalogabile, utilizzando un array interno a capacità fissa di 200

- Un costruttore senza argomenti che crea un magazzino vuoto.
- Un metodo **void aggiungiArticolo(T articolo)** che aggiunge un articolo. Deve lanciare **IllegalArgumentException** se un articolo con lo stesso codice è già presente, e **RuntimeException** se il magazzino è pieno.
- Un metodo **boolean rimuoviArticolo(String codiceUnivoco)** che rimuove l'articolo specificato, restituendo true se rimosso, false altrimenti.
- Un metodo **T trovaArticolo(String codiceUnivoco)** che restituisce l'articolo con il codice dato, o null se non trovato.
- Un metodo **double calcolaValoreTotale()** che restituisce la somma dei valori di tutti gli articoli presenti.

Esercizio 2

Gestione di Task con Priorità in Java

Interfaccia Task: definisce la struttura base per un task

- Un metodo **int getPriorita()** che restituisce la priorità (intero, valore più alto indica priorità maggiore).
- Un metodo **String getDescrizione()** che restituisce la descrizione del task.
- Un metodo **String getDataScadenza()** che restituisce la data di scadenza (formato "GG/MM/AAAA").

Gestione di Task con Priorità in Java

Classe TaskSemplice: implementa l'interfaccia Task e l'interfaccia Comparable<TaskSemplice>

- Deve implementare le interfacce **Task** e **Comparable<TaskSemplice>**.
- Attributi per **priorita** (int), **descrizione** (String), **dataScadenza** (String).
- Un costruttore che accetta priorità, descrizione e data di scadenza.
- Implementazione dei metodi dell'interfaccia **Task**.
- Implementazione del metodo **int compareTo(TaskSemplice other)**: ordina per priorità decrescente, poi per data di scadenza crescente (gestire il confronto delle date stringa).
- Ridefinizione del metodo **boolean equals(Object obj)**: due task sono uguali se hanno stessa priorità, descrizione e data.

Gestione di Task con Priorità in Java

Classe TaskRipetitivo: estende TaskSemplice per rappresentare task che si ripetono

- Deve estendere **TaskSemplice**.
- Attributo aggiuntivo **intervalloGiorni** (int).
- Un costruttore che accetta priorità, descrizione, data scadenza iniziale e intervallo.
- Ridefinizione del metodo **boolean equals(Object obj)**: uguaglianza basata su **TaskSemplice** e **intervalloGiorni**.

Gestione di Task con Priorità in Java

Classe Generica AgendaTask<T extends Task>: gestisce una collezione ordinata di task di tipo T, usando un array interno ordinato a capacità fissa di 100.

- Un costruttore senza argomenti che crea un'agenda vuota.
- Un metodo **void aggiungiTask(T task)** che inserisce un task mantenendo l'ordine nell'array. Lancia **RuntimeException** se l'agenda è piena.
- Un metodo T **getProssimoTask()** che restituisce il task più prioritario (primo elemento) senza rimuoverlo. Lancia **NoSuchElementException** se l'agenda è vuota.
- Un metodo T **completaProssimoTask()** che rimuove e restituisce il task più prioritario (primo elemento), gestendo lo shift dell'array. Lancia **NoSuchElementException** se l'agenda è vuota.
- Un metodo **int getNumeroTask()** che ritorna il numero di task presenti.

Esercizio 3

Implementazione di una Mappa Generica

Interfaccia Mappa<K, V>: definisce la struttura standard per una mappa

- Un metodo **void put(K chiave, V valore)** per inserire/aggiornare un'associazione.
- Un metodo **V get(K chiave)** per ottenere il valore associato a una chiave (**null** se non presente).
- Un metodo **V remove(K chiave)** per rimuovere un'associazione e restituire il vecchio valore (**null** se non presente).
- Un metodo **boolean containsKey(K chiave)** per verificare la presenza di una chiave.
- Un metodo **int size()** per ottenere il numero di associazioni.
- Un metodo **boolean isEmpty()** per verificare se la mappa è vuota.

Implementazione di una Mappa Generica

Classe Coppia<K, V>: modella una singola associazione chiave-valore.

- Attributi **chiave** (di tipo K) e **valore** (di tipo V).
- Un costruttore che accetta chiave e valore.
- Metodi **K getChiave()**, **V getValore()**, **void setValore(V nuovoValore)**.
- Ridefinizione del metodo **boolean equals(Object obj)**: due oggetti **Coppia** sono uguali se le loro **chiave** è uguale.

Implementazione di una Mappa Generica

Classe MiaMappa<K, V>: implementa l'interfaccia Mappa<K, V> utilizzando un array di Coppia<K, V> a capacità fissa di 100 elementi

- Deve implementare l'interfaccia **Mappa<K, V>**.
- Un costruttore senza argomenti che crea una mappa vuota.
- Implementazione del metodo **void put(K chiave, V valore)**: cerca la chiave; se esiste aggiorna il valore, altrimenti aggiunge una nuova **Coppia** (se c'è spazio, altrimenti lancia **RuntimeException**).
- Implementazione del metodo **V get(K chiave)**: cerca la **Coppia** con la chiave data e restituisce il valore, o **null** se non presente.
- Implementazione del metodo **V remove(K chiave)**: cerca la **Coppia**, la rimuove e restituisce il vecchio valore, o **null** se non presente.
- Implementazione del metodo **boolean containsKey(K chiave)**: cerca la **Coppia** con la chiave data.
- Implementazione dei metodi **int size()** e **boolean isEmpty()**.