Esercizi programmazione generica

Esercizio 1

- Scrivere un metodo generico per effettuare la ricerca di elementi in un array

Esercizio 2

- Realizzare e testare una classe generica che implementa il concetto di lista.
- Una lista è formata da nodi che contengono un dato (di tipo arbitrario) e che sono collegati con un link
  - ogni nodo conosce l'indirizzo del suo successore nella lista
- La lista si scorre a partire dal primo elemento

Esercizio 3

- Realizzare e testare uno stack che può contenere oggetti arbitrari
- Realizzare e testare una coda che può contenere oggetti arbitrari

Esercizio 4

- Implementare una classe parametrica **Sorter**, con un solo metodo **check**. Il metodo check confronta l'oggetto che riceve come argomento con quello che ha ricevuto alla chiamata precedente, o con quello passato al costruttore se si tratta della prima chiamata a check. Il metodo restituisce -1 se il nuovo oggetto è più piccolo del precedente, 1 se il nuovo oggetto è più grande del precedente e 0 se i due oggetti sono uguali. Per effettuare i confronti, Sorter si basa sul fatto che il tipo usato come parametro implementi l'interfaccia Comparable.
- Sorter s = new Sorter(7);
- System.out.println(s.check(4)); -1
- System.out.println(s.check(1)); -1
- System.out.println(s.check(6)); 1
- System.out.println(s.check(6)); 0

Esercizi Java Collections Framework

Esercizio 1

- Realizzare un metodo chiamato **merge** che rispetti il seguente contratto:
- Pre-condizione: Accetta due LinkedList dello stesso tipo e di pari lunghezza.
- Post-condizione: Restituisce una nuova LinkedList ottenuta alternando gli elementi della prima lista e quelli della seconda.
- Ad esempio, se la prima lista contiene (1, 2, 3) e la seconda lista (4, 5, 6), la nuova lista deve contenere (1, 4, 2, 5, 3, 6).

Esercizio 2

- Implementare il metodo statico **isSetSmaller**, che accetta due insiemi e un comparatore, e restituisce vero se e solo se tutti gli elementi del primo insieme sono più piccoli, in base al comparatore, di tutti gli elementi del secondo insieme.
- Implementare il metodo **isSorted** che accetta una collezione e un comparatore, e restituisce true se la collezione risulta già ordinata in senso non-decrescente rispetto a quel comparatore, e false altrimenti.

Esercizio 3

- Implementare il metodo statico **mergeIfSorted**, che accetta due liste a e b, e un comparatore c, e restituisce un'altra lista. Inizialmente, il metodo verifica che le liste a e b siano ordinate in senso non decrescente. Poi, se le liste sono effettivamente ordinate, il metodo le fonde (senza modificarle) in un'unica lista ordinata, che viene restituita al chiamante. Se, invece, almeno una delle due liste non è ordinata, il metodo termina restituendo null.

Esercizio 4

- Realizzare le classi *Book* e *Library*, che rappresentano rispettivamente un libro e una collezione di libri. Il metodo *addBook* di *Library* aggiunge un libro alla collezione, con un dato titolo e un dato autore. A ciascun libro è possibile attribuire uno o più argomenti tramite il suo metodo *addTag*. Il metodo *getBooksByTag* di *Library* restituisce in tempo costante l'insieme dei libri di un argomento dato.

```
Library casa = new Library(), ufficio = new Library();
Book b1 = casa.addBook("Esercizi␣di␣stile", "Queneau");
b1.addTag("letteratura");
b1.addTag("umorismo");
Book b2 = casa.addBook("Me␣parlare␣bene␣un␣giorno", "Sedaris");
b2.addTag("umorismo");
Book b3 = ufficio.addBook("Literate␣programming", "Knuth");
b3.addTag("programmazione");
Set<Book> humorCasa = casa.getBooksByTag("umorismo");
System.out.println(humorCasa);
Set<Book> humorUfficio = ufficio.getBooksByTag("umorismo");
System.out.println(humorUfficio);
```