

Metode Avansate de Programare SEMINAR 3

Collections and Generics/InFileRepository

- I. 1) Creați o clasă `Student` având următorii membri: `nume(String)`, `media(float)`, un constructor cu parametrii care initializează un `Student` și metoda `toString()`;

Instantiați următorii studenți:

```
Student s1= new Student("Dan", 4.5f);
Student s2= new Student("Ana", 8.5f);
Student s3= new Student("Dan", 4.5f);
```

- 2) Instantiați un obiect de tip `HashSet<Student>` și adăugați studenții de la punctul I.1. Ce observați?
3) Respectați *Contractul equals - hashCode*: dacă `obj1.equals(obj2)` atunci `obj1.hashCode() == obj2.hashCode()`.

Obs. Atunci când doriți să salvați obiecte în colecții ce sunt reprezentare în memorie pe tabele de dispersie (hash table), dacă implementați `equals()` trebuie să implementați și `hashCode()`.

- 4) Instantiați un obiect de tipul `TreeSet<Student>` și adăugați studenții de la punctul I.1). Definiți un comparator care compară doi studenți după nume.
5) Repetați exercitiile 2-4 folosind **HashMap și TreeMap**

- II. Scrieți o clasă `MyMap`, ce va reprezenta un `Map` pentru reținerea studenților după medie. Caracteristicile clasei definite sunt:

- Cheile pot avea valori de la 0 la 10 (corespunzătoare mediilor posibile).
- Valoarea asociată fiecărei chei va fi o listă (`List`) care va reține toți studenții cu media rotunjită egală cu cheia. Considerăm că un student are media rotunjită 8 dacă media sa este în intervalul [7.50, 8.49]. (`Math.round`)
- `Map`-ul vostru va menține cheile (mediile) ordonate descrescător. Folosiți o implementare potrivită a interfeței `Map`, care să permită acest lucru, și definiți un `Comparator` pentru stabilirea ordinii cheilor. (clasa internă statică)
- Definiți în clasa `MyMap` metoda `add(Student)`, ce va adăuga un student în lista corespunzătoare mediei lui. Dacă, în prealabil, nu mai există nici un student cu media respectivă (rotunjită), atunci lista va fi creată la cerere.

Observație: Ce se întâmplă când apelăm metoda put moștenită de la dicționar? „Favor Composition instead of Inheritance” – DISCUTIE CU STUDENTII....

- Adăugați câțiva studenți.
- Definiți o metoda `getEntries()` care returnează mulțimea intrărilor – `Entry<Key, Value>`
- Creați un dicționar de tipul `MyMap` și adăugați următorii studenți.

```
public static List<Student> getList(){
    List<Student> l=new ArrayList<Student>();
    l.add(new Student("1",9.7f));
    l.add(new Student("2",7.3f));
    l.add(new Student("3",6f));
    l.add(new Student("4",6.9f));
    l.add(new Student("5",9.5f));
    l.add(new Student("6",9.9f));
    return l;
}
```

- Iterați mulțimea intrărilor – `Entry<Key, Value>` și sortați alfabetic fiecare listă de studenți.

- III. 1) Considerăm interfața `Repository` care specifică operațiile CRUD pentru un `Repository` cu elemente generice care au un id de un tip generic ID.

(curs3.zip)

```
/**
 * CRUD operations repository interface
 * @param <ID> - type E must have an attribute of type ID
 * @param <E> - type of entities saved in repository
```

```

*/

public interface Repository<ID, E extends Entity<ID>> {

    /**
     * @param id -the id of the entity to be returned
     *           id must not be null
     * @return the entity with the specified id
     * or null - if there is no entity with the given id
     * @throws IllegalArgumentException if id is null.
     */
    E findOne(ID id);

    /**
     * @return all entities
     */
    Iterable<E> findAll();

    /**
     * @param entity entity must be not null
     * @return null- if the given entity is saved
     * otherwise returns the entity (id already exists)
     * @throws ValidationException if the entity is not valid
     * @throws IllegalArgumentException if the given entity is null.
     */
    E save(E entity);
}

```

2. Instantiati un repo InMemoryRepository pt entitatea Utilizator si adaugati 10 utilizatori.