# Metode Avansate de Programare SEMINAR 3

Collections and Generics/InFileRepository

I. 1) Creaţi o clasă Student avand urmatorii membri: nume(String), media(float), un constructor cu parametrii care initializeaza un Student si metoda toString();
Instantiati urmatorii studenti:

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

2) Instantiati un obiect de tip HashSet<Student> si adaugati studentii de la punctul I.1. Ce observati?
3) Respectati *Contractul equals - hashcode:* dacă obj1.equals(obj2) atunci obj1. hashCode() == obj2.hashCode().
Obs. Atunci cand doriti sa salvati obiecte în colecții ce sunt reprezentare in memorie pe tabele de dispersie (hash table), dacă implementați equals()trebuie sa implementati și hashcode().
4) Instantiati un obiect de tipul TreeSet<Student> si adaugati studentii de la punctul I.1). Definiti un comparator care compara doi studenti dupa nume.
5) Repetati exercitiile 2-4 folosind HashMap si TreeMap

II. Scrieţi o clasă MyMap, ce va reprezenta un Map pentru reţinerea studenţilor după medie. Caracteristicile clasei definite sunt:

1) Cheile pot avea valori de la 0 la 10 (corespunzătoare mediilor posibile).
2) Valoarea asociată fiecărei chei va fi o listă (List) care va reţine toţi studenţii 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)
3) Map-ul vostru va menţine cheile (mediile) ordonate descrescător. Folositi o implementare potrivită a interfeţei Map, care să permită acest lucru, şi definiti un Comparator pentru stabilirea ordinii cheilor. (clasa internă statică)
4) Definiţi 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.

Observatie: Ce se intampla cand apelam metoda put mostenita de la dictionar? „Favor Composition instead of Inheritance" – DISCUTIE CU STUDENTII....

5) Adaugati câţiva studenţi.
6) Definiti o metoda getEntries() care returneaza multimea intrarilor – Entry<Key, Value>
7) Creati un dictionar de tipul MyMap si adaugati urmatorii studenti.

```
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;
}
```

8) Iterati multimea intrarilor – Entry<Key, Value> şi sortaţi alfabetic fiecare listă de studenţi.

III. 1) Consideram interfata `Repository` care specifica operatiile 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 InMemoryReppository pt entitatea Utilizator si adaugati 10 utilizatori.