Contents

Seminar 5 – Lamda - Stream

1. Interfete functionale (Functional Interface)

a) Declarati o interfata functionala Area.
b) Definiti functii lambda pentru calculul ariilor unor figuri geometrice.
c) Implementati metoda definita de urmatoarea signatura:

```
public static <E> void  printArie(List<E> l, Area<E> f) {}
```

Q1: Ce fel de metoda este metoda printArie?

Q2: Putem folosi wildcards?

d) Apelați funcția `printArie` pentru o lista de cercuri si apoi o lista de patrate folosind funcțiile lambda definite la punctul b) si referințe la metode definite in clasa AreaHelper.

2. Built-in functional interfaces

2.1 Consumer accept

```
Consumer<Person> greeter = (p) -> System.out.println("Hello, " + p.firstName);
greeter.accept(new Person(“Aprogramatoarei", “Dan"));
```

2.2. Predicates test

1. Definiti o metoda generica care filtreaza entitatile dintr-o colectie iterabila, care satisfac un anumit predicat. Lista si predicatul sunt specificate ca parametri.

   ```
   <T> Iterable <T> filter(Iterable <T> list, Predicate<T> cond)
   ```

   ```
   Observatie: Folositi list.forEach
   ```

2. Creati filtre concrete pentru filtrarea unei liste de mesaje dupa:
   a) subject
   b) expeditor
   c) dupa data
   d) dupa data si expeditor

3. Inlocuiti functia lamda ce defineste predicatul p printr-o referinta la metoda:

   ```
   String anamaria="anamaria";
   Predicate<String> p= x -> anamaria.startsWith(x);
   System.out.println(p.test("ana"));
   ```

2.3 Functions apply

1. Definiti o functie (Function) pentru conversia unei valori de tipul sir de caractere la o valoare intreaga. Folositi functii lambda si referinta la metode.

   ```
   Function<String,Integer> converterLambda=x->Integer.valueOf(x);
   Function<String,Integer> converterMethodReference=Integer::valueOf;

   Integer fromString=converterLambda.apply("12");
   Integer fromString2=converterMethodReference.apply("12");
   ```

2.4 Suppliers get

1. In clasa din `TaskContainerFactory` Sem 1 si 2, schimbati signatura metodei `createContainer` cu urmatoarea signatura: `public Supplier<Container> createContainer(Strategy strategy)`

   ```
   public class TaskContainerFactory implements Factory{

       public Supplier<Container> createContainer(Strategy strategy) {
           if (strategy==strategy.FIFO)
               return QueueContainer::new;
           else
               return StackContainer::new;
       }

   }
   ```

2.5 Comparators

FilterAndSorter – generic

```
public static <T> List<T> filterAnSorter(List<T> list, Predicate<T> cond, Comparator<T> comp)
{


}
```

3. Optional

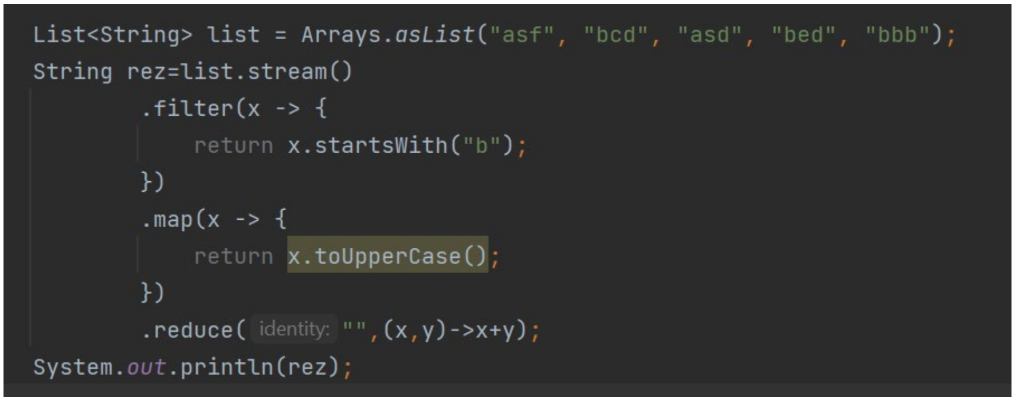
```
@Override
public Optional<E> delete(ID id) {
        return Optional.ofNullable(entities.remove(id));
}

@Override
public Optional<T> update(T entity) throws ValidatorException {
        validator.validate(entity);
        if (entities.containsKey(entity.getId())) {
                entities.put(entity.getId(), entity);
                return Optional.empty(); // in loc de null
        }
        return Optional.of(entity);
}
```

`Optional.isPresent() sau Optional.ifPresent(Consumer)  Optional.get`

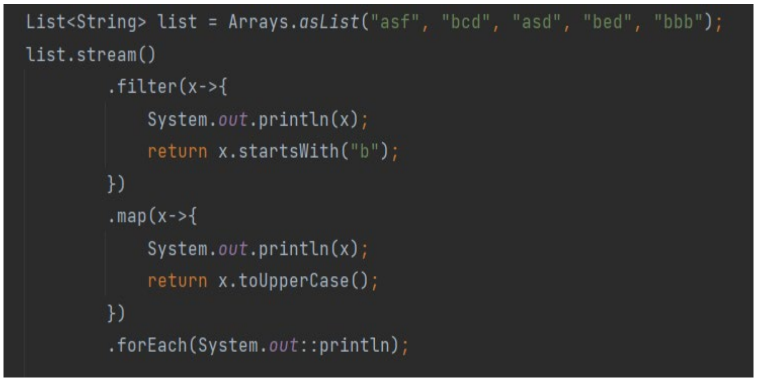
4. Stream

3.1 Filter – Map - Reduce

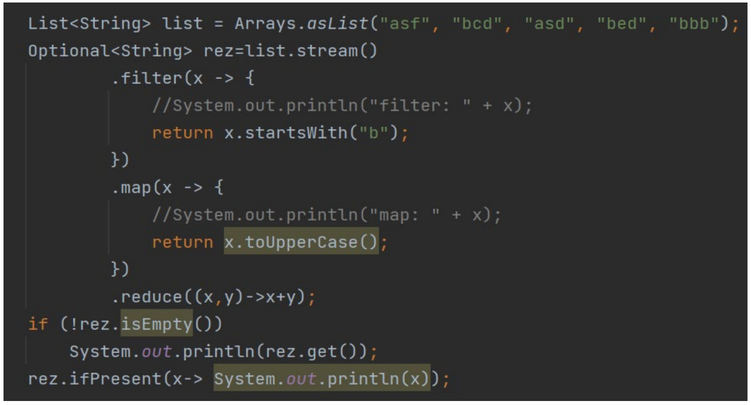
A. Ce afiseaza urmatoarele programe?



```
List<String> list = Arrays.asList("asf", "bcd", "asd", "bed", "bbb");
String rez=list.stream()
         .filter(x -> {
             return x.startsWith("b");
         })
         .map(x -> {
             return x.toUpperCase();
         })
         .reduce( identity: "",(x,y)->x+y);
System.out.println(rez);
```



```
List<String> list = Arrays.asList("asf", "bcd", "asd", "bed", "bbb");
list.stream()
        .filter(x->{
            System.out.println(x);
            return x.startsWith("b");
        })
        .map(x->{
            System.out.println(x);
            return x.toUpperCase();
        })
        .forEach(System.out::println);
```



```
List<String> list = Arrays.asList("asf", "bcd", "asd", "bed", "bbb");
Optional<String> rez=list.stream()
        .filter(x -> {
            //System.out.println("filter: " + x);
            return x.startsWith("b");
        })
        .map(x -> {
            //System.out.println("map: " + x);
            return x.toUpperCase();
        })
        .reduce((x,y)->x+y);
if (!rez.isEmpty())
    System.out.println(rez.get());
rez.ifPresent(x-> System.out.println(x));
```

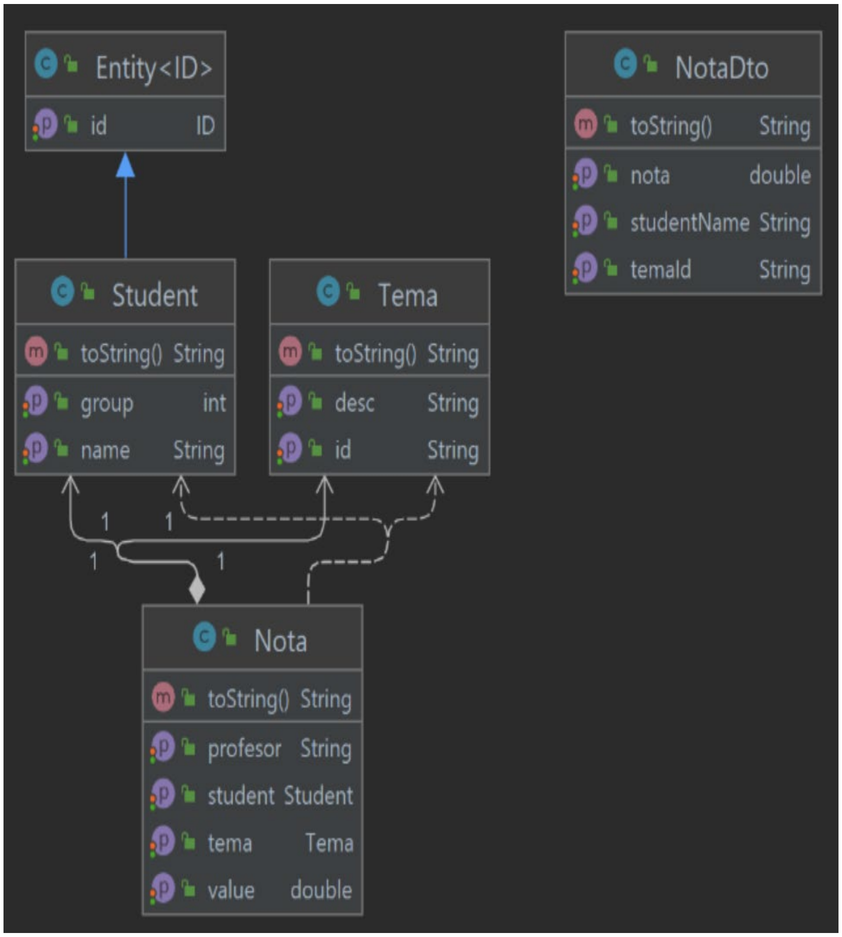
3.2 Collectors.groupingBy(BiConsumer <E,T>)

```
static void groupStudentsByGrade(){
    Map<Integer, List<Student>> studentsByGrade = students
            .stream()
            .collect(Collectors.groupingBy(s->Math.round(s.getMedia())));
    studentsByGrade
            .forEach((media, s) -> System.out.format("media %s: %s\n", media, s));
}
```

3.3 Files

```
Path path = Paths.get("./src/data/Studs.txt");
Stream<String> lines;
try {
   lines = Files.lines(path);   //Files – helper class
   lines.forEach(s -> System.out.println(s));
} catch (IOException e) { . . . }
```

3.4 Aplicatii: Consideram urmatoarea diagrama de clase:



Fiind date o lista de Studenti, o lista de Teme si o lista Note, sa se realizeze urmatoarele rapoarte:

1. toate notele acordate de un anumit profesor, la o anumita grupa
2. media notelor pt fiecare student (`Collectors.groupingBy`)
3. media notelor la o anumita tema
4. tema cu cea mai mare medie
5. tema cea mai grea (media notelor cea mai mica)