Sofia University
Department of Mathematics and Informatics

Course : : OO Programming Java
Date: December 14, 2018
Student Name:

Lab No. 9a

Задача 1а

Дадени са следните **class** Salesperson и **class** LambdaDemo.

```
public class Salesperson
{
   private String name;
   private double salary;
   private int numsales;

  public   Salesperson(String name, double salary, int numsales)
   {
      this.name = name;
      this.salary = salary;
      this.numsales = numsales;
   }

  public   void addBonus(double amount)
   {
      salary += amount;
   }

  public   int getNumSales()
   {
      return numsales;
   }
  public   double getSalary()
   {
      return salary;
   }
  public  void printNumSales (Salesperson obj){
        System.out.println(obj.getNumSales());
    }
   @Override
   public String toString()
   {
```

```
        return String.format("name: %s, salary: %.2f numsales: %d
",
                                name, salary, numsales);
    }

}
 class LambdaDemo
{
    public static void main(String[] args)
    {
        Predicate<Salesperson> predicate1; // да се инициализира
        Predicate<Salesperson> predicate2; // да се инициализира
        Predicate<Salesperson> predicate ; // да се инициализира

        Consumer<Salesperson> consumer1; // да се инициализира
        Consumer<Salesperson> consumer2; // да се инициализира

        Comparator<Salesperson> comparator1; // да се инициализира
        Comparator<Salesperson> comparator2; // да се инициализира

        Salesperson[] salespersons =
        {
            new Salesperson("John Doe", 2000, 949),
            new Salesperson("Jane Doe", 3900, 1500),
        // да се добавят  още 10 обекти от тип Salesperson
        // или да се инициализират с Random стойности

        };
        List<Salesperson> listOfSalespersons = new ArrayList<>();
        // обектите на salespersons да се запишат в
listOfSalespersons
        for (Salesperson salesperson: salespersons)
        {
            applyBonus(salesperson, predicate1,   consumer1);
            System.out.println(salesperson);
            salesperson.printNumSales(salesperson);

        }
        for (Salesperson salesperson: salespersons)
        {
            applyBonus(salesperson, predicate2,  consumer2);
            System.out.println(salesperson);
        }
        sort(listOfSalespersons, comparator1);
        System.out.println(listOfSalespersons);

        sort(listOfSalespersons, comparator2);
```

```
        System.out.println(listOfSalespersons);

    }

    public static void applyBonus(Salesperson salesperson,
                                  Predicate<Salesperson>
predicate,
                                  Consumer<Salesperson> consumer)
    {
        // Напишете if  команда, където използвайте predicate
        //    за да определите дали да изпълните consumer
        // Изпълнете consumer, когато условието на if  командата е
изпълнено
    }

    public static void applyBonus(List<Salesperson> salespersons,
                                  Predicate<Salesperson>
predicate,
                                  Consumer<Salesperson> consumer)
    {
        // Напишете if  команда, където използвайте predicate
        //    за да определите дали да изпълните consumer
        // Изпълнете consumer, когато условието на if  командата е
изпълнено
    }

    public static void sort(List<Salesperson> salespersons,
                                  Comparator<Salesperson>
comparator)
    {
        // Сортирайте  salespersons като използвате  comparator
    }
    public static void group(List<Salesperson> salespersons)
    {
        // Групирайте имената на salespersons по първата буква в
името изведете отделните групи на стандартен изход
    }

}
```

Инициализирайте `predicate1, predicate2 и predicate  с Ламбда` изрази, където

a) predicate1 връща true, когато getNumSales() на даден Salesperson е по-голямо от 1200
b) predicate2 връща true, когато predicate1 на даден Salesperson е по-голямо от 900
c) predicate връща true, когато predicate1 или predicate1 връщат true

Инициализирайте consumer1 и consumer2 с Ламбда изрази, където

a) consumer1 увеличава с 5% заплатата на даден Salesperson и извежда на стандартен изход всичките му данни
b) consumer2 увеличава с 2% заплатата на даден Salesperson, ако predicate1 връща true. В противен случай намалява заплатата на този Salesperson с 2% и извежда на стандартен изход всичките му данни
c) comparator1 задава наредба в низходящ ред на обекти Salesperson по отношение на свойството **salary**
d) comparator2 задава наредба низходящ ред на обекти Salesperson по отношение на свойството **salary**, а при дублиране на сортира във възходящ ред на свойство **numsales**

Допълнете <u>методите</u> applyBonus(), sort()и main() с липсващите команди

<u>Компилирайте</u> **class** Salesperson и **class** LambdaDemo и **<u>изпълнете</u>** **class** LambdaDemo.

<u>Задача 1b</u>

Имплементирайте в class Salesperson

```
@FunctionalInterface
interface IAdder<T extends Salesperson>{
    T add (T op1, T op2);
```

```
    default String printNumSales (T obj){
        return obj.getNumSales() ;
    }
    static void printSales(Salesperson s)
    {
        System.out.println(s.getNumSales());
    }
}
```

1. **Предефинирайте метода** `add (Salesperson op1, Salesperson op2)` **да създава обект** `Salesperson` **със същото име и заплата като** `op1`, **но** `numSales` **да е сума от** `numSales` **на** `op1` **и** `op2`
2. **Предефинирайте** `printNumSales (T obj)` **в клас Salesperson, която да връща форматиран стринг на резултата от изпълнението на printNumSales в interface Adder .**
3. **Изпълнете методите** `add (Salesperson op1, Salesperson op2)` **и** `printNumSales (T obj)` **в клас** `LambdaDemo`

<u>Задача 1c.</u>

Да се предефинират в `Salesperson` **методите наследени от клас Object**
`equals()`
`hashCode()`
където два обекта са равни, когато имат еднакви имена

Да се добави метод

```
    public static Set<Salesperson> distinct (List< Salesperson >
list) {
        // връща Set на Salesperson с различни имена
    }
```

Методът да се тества в `LambdaDemo`

<u>Задача 2.</u>
а) Да се напишат Ламбда изрази от следните типове и да се дадат примери за тяхното изпълнение

a) `Function<Integer, String>`

b) `BiPredicate<Double, Double>`

c) `BiConsumer<String, String>`

d) `Supplier<int[]>`

e) `IntFunction<double[]>`

b) Редактирайте следните Ламбда изрази като реферирате използваните методи

a) `Function<Double, Double> func = (x) -> Math.cos(x);`

b) `Consumer<String>  task = ( x) ->System.out.println(x);`

c) `Runnable task = (  ) ->System.out.println( );`

d) `Predicate<String> isEqual = (s)->s.equals("");`

f) `Runnable`

c) Нека е даден метод

```
public void method(Function<Double, Double> function) {}
```

Дайте пример за изпълнение на този метод по един от следните начини:

1. **Като подавате за аргумент подходящо дефиниран Ламбда израз**
2. **Като подавате за аргумент подходящо дефиниран статичен метод**
3. **Като подавате за аргумент подходящо дефиниран нестатичен метод**

d) Да се определи стандартния тип на следните Ламбда изрази

```
(x) -> Math.cos(x)
(x) -> System.out.println(x);
()   -> System.out.println("Това е Ламбда");
(int x, int y) -> x > y;
(int x) -> x %2 == 0;
(int x, int y) -> x +  y;
```

<u>Задача 3</u>

Напишете Ламбда за изпълнението на следните задания:

a) Напишете Ламбда израз, който да се използва вместо следния анонимен клас:

```
new IntConsumer()
{
    public void accept(int value)
    {
       System.out.printf("%d ", value);
    }
}
```

b) Напишете функционален интерфейс съответен на следния Ламбда израз:

```
(String s) -> {return s.toUpperCase();}
```

С помощта на този интерфейс напишете анонимен клас, който е еквивалентен на дадения Ламбда израз

c) Напишете Ламбда израз без параметри , който връща String `"Welcome to lambdas!„`

d) Напишете Ламбда израз с два целочислени параметъра, който връща по-големия от тях