Lab 8

Problem 1:

A. Differences between Imperative and Functional Programming.

- With Functional Programming, all is oriented to use functions as the main way to transform data. Delegating every single task to specific functions. Making the code more readable and concise. In imperative programming, you can do process everywhere and need to specify many details to achieve a goal.

- With funcional programming, Functions are oriented to don't cause changes in the environment. This means that a Function can use their params and other free variables of the environment but should not modify them, just return value if is needed. In other words, Functions do not make changes of state, they are stateless.

B. Explain the meaning of declarative programming. Give an example.

- Declarative programming is oriented to define and use methods that make only specific tasks without specify many details, looking always for WHAT to do instead of HOW to do.

In Example, the Markup language HTML is declarative, because it only describes what should appear on a webpage, it's not in charge to define the control flow for rendering a page.

C. Explain the difference between functional interface, functor, and closure, and give examples of each using Java 7 syntax

- Functional Interface:

Is an interface that only has ONE abstract method but that method should not be one declared from Object. Also, this interface should have the annotation: *@FunctionalInterface*

In example:

```
interface Comparator<T> {
        int compare(T o1, T o2);
}
```

- Functor:

The implementation of a functional interface is called a functor.

In example:

```
public class EmployeeNameComparator implements Comparator<Employee> {
        @Override
        public int compare(Employee e1, Employee e2) {
                return e1.name.compareTo(e2.name);
        }
}
```

- Closure:

A closure is a functor embedded inside another class, and it can use the state of its enclosing object

In example:

```
public class Main {

        private String attribute1 = "Class Attribute";

        private void methodA(String param1) {
                class methodAInnerClass {
                        public void innerMethod(String myParam) {
                                System.out.println("Printing " + myParam);// From param
                                System.out.println("Printing " + param1);// From first closure
                                System.out.println("Printing " + attribute1);// From second closure
                        }
                }
                new methodAInnerClass().innerMethod("InnerParam");
        }

        public static void main(String[] args) {
                (new Main()).methodA("Params method");
        }
}
```

D. Name three benefits of including functional style programming in Java
- In difference to Imperative programming, the Functions can be manipulated as an object with Functional Programming.

- Self-documenting

- Functional programming has a clear correspondence to mathematical logic.

E. Express the functions defined below using Church's lambda notation:

	Math function		***Church's lambda***
i.	$f(x) = x + 2x^2$	i.	$\lambda x.x + 2x\text{^}2$
ii.	$g(x,y) = y - x + x^y$	ii.	$\lambda xy.y - x + x\text{^}y$
iii.	$h(x,y,z) = z - (x + y)$	iii.	$\lambda xyz.z - (x + y)$

F. For each lambda expression below, name the parameters and the free variables.

i. Parameters: *"nothing"*
Free variables: *s, t*

ii. Parameters: *u, v*
Free variables: *d, x, a, b*

iii. Parameters: *s, t*
Free variables: *ignoreCase*

H. Explanation (The code for this point it's on *problem1.partH* package)
- Method reference: *System.out::println*

- Lambda expression: *p -> System.out.println(p);*

- Explanation:
System is final class which contains static attributes, among them there is *out* variable. The VM(virtual machine) will call the *initializeSystemClass* method to initialize this class and therefore initialize the static variable *out*.
To finalize, the type of method reference in this case is *object::instanceMethod* which in turn is an implementation of *Consumer<String>*.