4. LAB D – SINGLETON

Purpose

To demonstrate how to limit the amount of instances for a given class, we will follow the singleton design pattern and develop an eager singleton class. We will develop a singleton SingletonEasy class. The class will contain logic, to limit the amount of SingletonEasy objects in the JVM to a maximum of one.

Lab Scenario (WHAT to do)	Create a class SingletonEasy It has a static member of type SingletonEasy Eagerly initialized (initialized when the class is loaded) Define a static getInstance(…) method. Clients calling this method will always get the same instance.
Implementation Steps (HOW to do it)	**Code the SingletonEasy class** 1. Make the constructor private 2. Create a static SingletonEasy member variable and assign it a new SingletonEasy object 3. implement a static SingletonEasy getInstance() method which returns the value assigned to the member variable, thus always returning the same instance **Code the test class** Write a main method which asks for an instance twice, and compares the address of the objects received (== operator). If both objects have the same address the singleton is achieved.

Review of Singleton

We will create 1 class:

1. SingletonEasy will be a singleton class using eager loading

4.1.1. Create a new class – SingletonEasy

This class implements the singleton design pattern.

1. Create the class **SingletonEasy**

2. Implement a private static SingletonEasy INSTANCE; as member variable and assign a new SingletonEasy object

   ```
   private static SingletonEasy INSTANCE = new SingletonEasy();
   ```

   Note: the singleton is defined as Eager, so we create a new object when the class is loaded and return the same object for any client asking for an instance

3. Implement a getInstance() method. The method simply returns the object stored at INSTANCE:

   ```
   public static SingletonEasy getInstance() {
       return INSTANCE;
   }
   ```

4. Make the constructor private

   ```
   private SingletonEasy() {
       super();
       System.out.println("SingletonEasy -> Initialized");
   }
   ```

All client access should go through the getInstance method(). If there is a public constructor, clients can create as many objects as they want, bypassing the singleton. Making the constructor private, forces clients to call the getInstance() method for objects.

Note: The solution for this exercise is available in the ‘solutions’ directory