**a Singleton class in Java that will be used by multiple threads concurrently.**

1. **Singleton design pattern with Eager Instantiation :**

|  |
| --- |
| package in.bench.resources.singleton.design.pattern;    public class EagerIstantiation {        // Step 1: private static variable of INSTANCE variable      private static EagerIstantiation      INSTANCE = new EagerIstantiation ();        // Step 2: private constructor      private EagerIstantiation () {        }        // Step 3: Provide public static getInstance() method      // returning same INSTANCE same time      public static EagerIstantiation () {          return INSTANCE;      }  } |

1. **Singleton design pattern with Lazy Instantiation :**

package in.bench.resources.singleton.design.pattern;

public class LazyInstantiation {

    // Step 1: private static variable of INSTANCE variable

    private static LazyInstantiation

            INSTANCE;

    // Step 2: private constructor

    private LazyInstantiation () {

    }

    // Step 3: Provide public static getInstance() method

    // returning INSTANCE after checking

    public static LazyInstantiation () {

        if(null == INSTANCE){

            INSTANCE = new

                 LazyInstantiation ();

        }

        return INSTANCE;

    }

}

1. **Singleton design pattern in a multi-threaded environment**

package in.bench.resources.singleton.design.pattern;

public class MultiThreadedEnvironment {

    // Step 1: private static variable of INSTANCE variable

    private static volatile

        MultiThreadedEnvironment INSTANCE;

    // Step 2: private constructor

    private MultiThreadedEnvironment () {

    }

    // Step 3: Provide public static getInstance() method

    // returning INSTANCE after checking

    public static MultiThreadedEnvironment () {

        // synchronized block

        synchronized

          (MultiThreadedEnvironment.class){

          if(null == INSTANCE){

              INSTANCE =

                new

         MultiThreadedEnvironment ();

          }

          return INSTANCE;

        }

    }

}

1. **Double-checked locking – DCL**

package in.bench.resources.singleton.design.pattern;

public class DoubleCheckedLockingDCL {

    // Step 1: private static variable of INSTANCE variable

    private static volatile DoubleCheckedLockingDCL

            INSTANCE;

    // Step 2: private constructor

    private DoubleCheckedLockingDCL () {

    }

    // Step 3: Provide public static getInstance() method

    // returning INSTANCE after checking

    public static DoubleCheckedLockingDCL getInstance() {

        // double-checking lock

        if(null == INSTANCE){

            // synchronized block

            synchronized (DoubleCheckedLockingDCL.class) {

                if(null == INSTANCE){

                    INSTANCE = new DoubleCheckedLockingDCL ();

                }

            }

        }

        return INSTANCE;

    }

}