**Exercise 1: Implementing the Singleton Pattern**

class Singleton {

public static void main(String[] args) {

Abc obj1 = Abc.getInstance();

Abc obj2 = Abc.getInstance();

System.out.println("obj1 = "+obj1);

System.out.println("obj2 = "+obj2);

System.out.print("We are getting same object as we are using singleton design pattern here.");

}

}

class Abc {

static Abc obj = new Abc();

private Abc(){}

public static Abc getInstance(){

return obj;

}

}

**OUTPUT:**

obj1 = Abc@429bd883

obj2 = Abc@429bd883

We are getting same object as we are using singleton design pattern here.

* **The static object obj in Abc class is created and it will be in memory when the class is loaded so even if we are not using the object it will be there in memory which is wastage of memory. To overcome this we will perform Lazy Instantiation.**

class Singleton {

public static void main(String[] args) {

Abc obj1 = Abc.getInstance();

Abc obj2 = Abc.getInstance();

}

}

class Abc {

static Abc obj;

private Abc(){

System.out.println("Instance created only once.");

}

public static Abc getInstance(){

if(obj == null){

obj = new Abc();

}

return obj;

}

}

**OUTPUT:**

Instance created only once.

* **If we use a thread concept and create 2 threads (one working with obj1 and other working with obj2), the problem is both threads might call getInstance() at the sametime and for both threads the obj will be null and we will get 2 different instances. So we have to use Double Checked Locking.**

class Singleton {

public static void main(String[] args) {

Thread t1 = new Thread(new Runnable() {

public void run(){

Abc obj1 = Abc.getInstance();

}

});

Thread t2 = new Thread(new Runnable() {

public void run(){

Abc obj2 = Abc.getInstance();

}

});

t1.start();

t2.start();

}

}

class Abc {

static Abc obj;

private Abc(){

System.out.println("Instance created only once.");

}

public static Abc getInstance(){

if(obj == null){

obj = new Abc();

}

return obj;

}

}

**OUTPUT:**

Instance created only once.

Instance created only once.

* **Double Checked Locking**

class Singleton {

public static void main(String[] args) {

Thread t1 = new Thread(new Runnable() {

public void run(){

Abc obj1 = Abc.getInstance();

}

});

Thread t2 = new Thread(new Runnable() {

public void run(){

Abc obj2 = Abc.getInstance();

}

});

t1.start();

t2.start();

}

}

class Abc {

static Abc obj;

private Abc(){

System.out.println("Instance created only once.");

}

public static Abc getInstance(){

if(obj == null){

synchronized(Abc.class){

if(obj == null){

obj = new Abc();

}

}

}

return obj;

}

}

**OUTPUT:**

Instance created only once.