**Name**: Ravi Jamanbhai Makwana

**Roll No**: 21BCP418

Div-6, G-12

**Assignment-5 (Singleton Pattern)**

**Aim:**

To implement Singleton Design Pattern using Accommodation management system.

**1) Eager Singleton Design Pattern:**

**Code:**

room1.java

/\*In Eager Singleton design pattern,

although only one object is created, it is created the moment the constructor is called,

so it occupies a lot of memory in the system

\*/

public class room1{

    public static final room1 r = new room1();  //creating static object

    private room1(){     //creating constructor as private so that only one instance can be created

    }

    public static room1 getRoom(){

        return r;

    }

    public void printAlloted(){

        System.out.println("Room created");

    }

}

Test1.java

public class Test1 {

    public static void main(String args[]){

        room1 r1 = room1.getRoom();

        r1.printAlloted();

        room1 r2 = room1.getRoom();

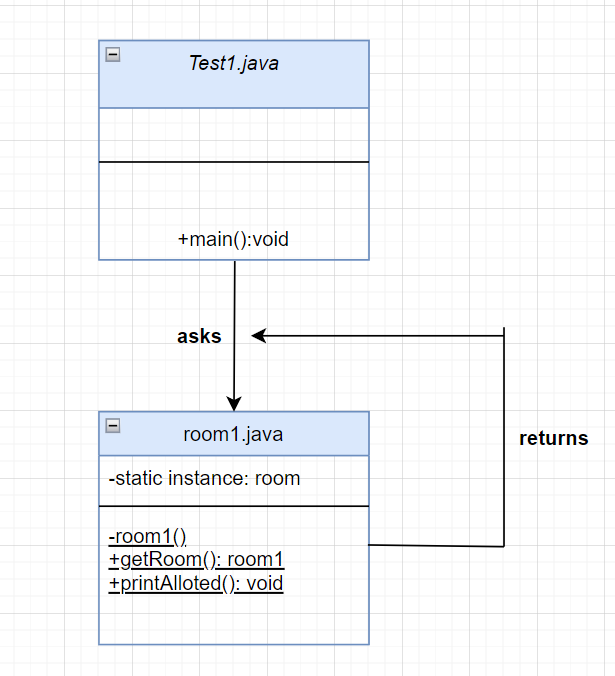
        r2.printAlloted();

        // room r3 = new room();        this is not possible as constructor of the class "room" is private

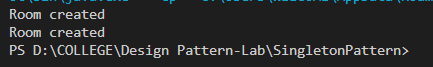
    }

}

**Class Diagram:**



**Output:**



**2) Lazy Singleton Design Pattern:**

**Code:**

room2.java

/\*In Lazy Singleton design pattern,

the object is only instantiated, not created

object is created later during the getRoom function call

this is helpful as it does not occupy more space in the memory

\*/

public class room2{

    public static room2 r;      //object is only instantiated here, it is later created inside getRoom() function

    private room2(){     //creating constructor as private so that only one instance can be created

        System.out.println("Room only instatantiated");

    }

    public static room2 getRoom(){

        if (r == null){

            r = new room2();        //object is created here. It is lazy object

            System.out.println("Room created");

        }

        return r;

    }

}

Test2.java

public class Test2 {

    public static void main(String args[]){

        room2 r1 = room2.getRoom();

        room2 r2 = room2.getRoom();     //although we try to create so many rooms, only one room is created

        room2 r3 = room2.getRoom();     //and we can call methods only for that object

        room2 r4 = room2.getRoom();

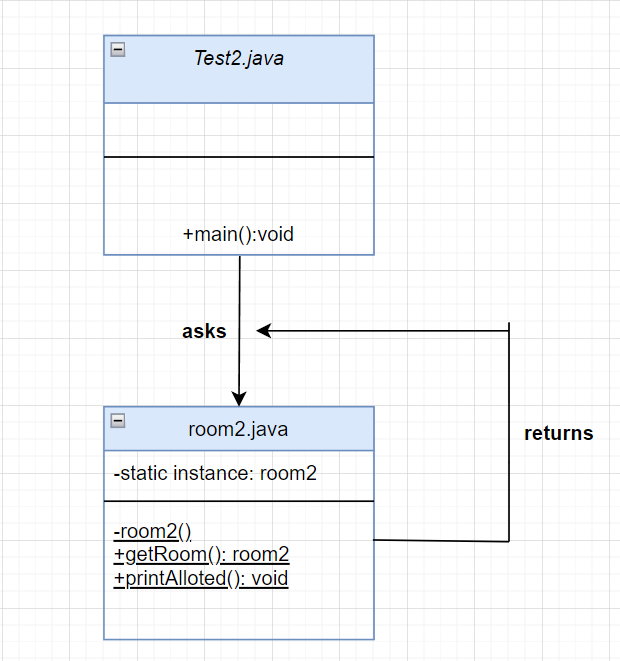
        room2 r5 = room2.getRoom();

        room2 r6 = room2.getRoom();

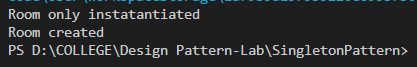
    }

}

**Class Diagram:**



**Output:**



**3) Singleton Design Pattern with Synchronized instance**

**Code:**

room3.java

/\*

 \* In synchronized method of singleton design pattern,

 \* the main objective is that only one object is created,

 \* even if we use multithreading

 \*/

public class room3{

    static int counter=0;

    public static room3 r;

    private room3(){     //creating constructor as private so that only one instance can be created

        System.out.print("Room instantiated");

        counter++;

        System.out.println("     Room object instantiated: "+ counter + " times.");

    }

    public static room3 getRoom(){

        if (r == null){

            synchronized (room3.class){     //put synchronized here

                if (r == null)      //double checking for null

                r = new room3();        //object created is lazy

                System.out.println("Room created");

            }

        }

        return r;

    }

}

Test3.java

public class Test3 {

    public static void main(String args[]){

        Thread t1 = new Thread (new Runnable() {

            public void run(){

                room3 r = room3.getRoom();

            }

        });

        Thread t2 = new Thread (new Runnable() {

            public void run(){

                room3 r = room3.getRoom();

                //although it is another thread trying to create a new object, no new object will be created.

                //the object created by thread-1 will only be used to call the method again

            }

        });

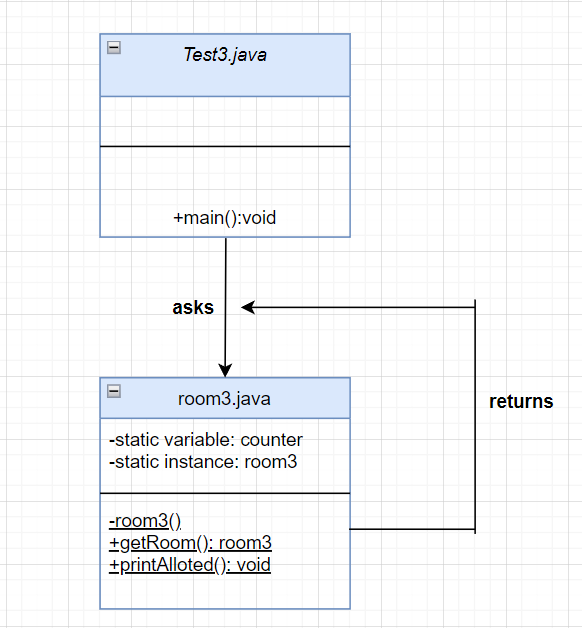
        t1.start();

        t2.start();

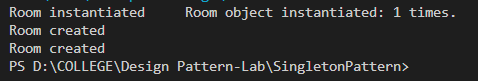
    }

}

**Class Diagram:**



**Output:**



**4) Enum Singleton Design Pattern:**

**Code:**

room.java

package Enum;

public class Test {

    public static void main(String args[]){

        room r1 = room.INSTANCE;

        System.out.print("Value of i for object r1 is: ");

        System.out.println(r1.getI());

        r1.setI(2);

        System.out.print("Value of i for object r1 is: ");

        System.out.println(r1.getI());

        room r2 = room.INSTANCE;

        r2.setI(5);     //we set value of attribute 'i' of object r2

        System.out.print("Value of i for object r1 is: ");

        System.out.println(r1.getI());  //although we call getI() function for 1st object,

                    //whose attribute's value we set above as 2, we get the value of i as 5

    }

}

enum room{

    INSTANCE;

    int i;

    public int getI(){

        return i;

    }

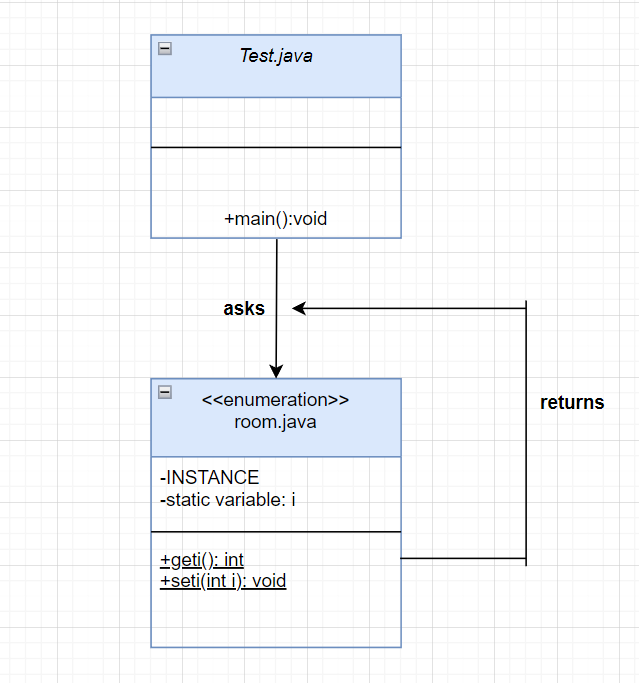
    public void setI(int i){

        this.i = i;

    }

}

**Class Diagram:**



**Output:**

