|  |  |  |  |
| --- | --- | --- | --- |
|  | **Course Name: Design Patterns/Thinking LAB** | **EXPERIMENT NO. 5** | |
| **Course Code: 20CP210P**  **Faculty: Dr. Ketan Sabale** | **Branch: CSE** | **Semester: IV** |
| **Submitted by: Jangle Parth**  **Roll no: 22BCP083** | | | |

Objective: To familiarize students with standard Creational design patterns.

Experiment: Explain the singleton design pattern and write a program using any object-oriented programming language to demonstrate the working of singleton design pattern.

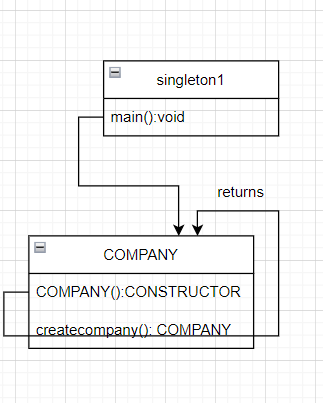
Theory : Imagine a Scenario when there is Object which needs to be created only once due to some reason . So how to implement it . There are majorly 5 ways in which this can be implemented.

In the way 1 we create a object of the class and we create it’s constructor declared as private . then we can create a method which is of Return type of the object and it will return the obj so even if the method is called more than once only one object is created.

**Problem Statement Explanation:**

Imagine we are Government Body which is responsible to give trademark to companies and there is a rule that only one company can get a trademark of a particular name . in that application singleton method can be used that if company is created then new object can’t be created.

**Flowchart Explanation:**

****

**Method 1: Eager Instantiation**

**Code:**

public class singleton1 {

    public static void main(String[] args) {

        COMPANY ATPL = COMPANY.createcompany();

        COMPANY ATPL = COMPANY.createcompany();

    }

}

class COMPANY {

    static COMPANY ATPL = new COMPANY();

    private COMPANY() {

        System.out.println("COMPANY CREATED SUCCESFULLY");

    }

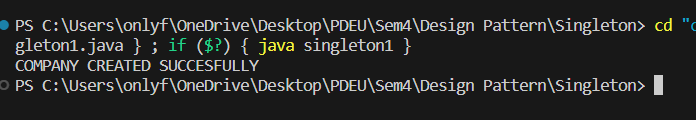
    public static COMPANY createcompany() {

        return ATPL;

    }

}

**Output:**

****

**Method 2: Lazy Instantiation**

**Limitation of Method1 :** we can’t get to know that if a object is already created or not

**Code:**

public class singleton2 {

    public static void main(String[] args) {

        COMPANY ATPL = COMPANY.createcompany();

        COMPANY AMUL = COMPANY.createcompany();

    }

}

class COMPANY {

    static COMPANY obj;

    private COMPANY() {

        System.out.println("COMPANY CREATED SUCESSFULLY");

    }

    public static COMPANY createcompany() {

        if (obj == null) {

            obj = new COMPANY();

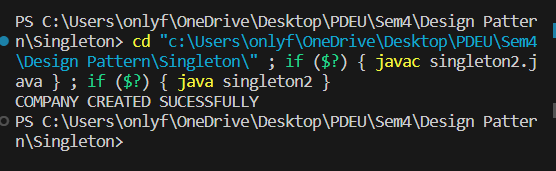
        }

        return obj;

    }

}

**Output:**

****

**Method 3: Synchronized**

**Limitation of Method2 :** If we create many thread’s which access the method parallelly then many objects of same can be created

**Code:**

public class singleton3 {

    public static void main(String[] args) {

        Thread t1 = new Thread(

                new Runnable() {

                    public void run() {

                        COMPANY ATPL = COMPANY.createcompany();

                    }

                });

        Thread t2 = new Thread(

                new Runnable() {

                    public void run() {

                        COMPANY AMUL = COMPANY.createcompany();

                    }

                });

        t1.start();

        t2.start();

    }}

class COMPANY {

    static COMPANY obj;

    private COMPANY() {

        System.out.println("COMPANY CREATED SUCESSFULLY");

    }

    public static synchronized COMPANY createcompany() {

        if (obj == null) {

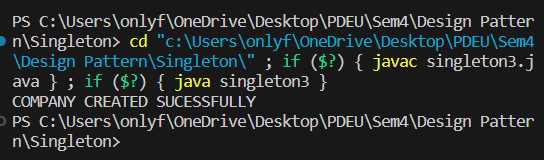
            obj = new COMPANY();

        }

        return obj;

    }}

**Output:**



**Method 4: Double Checked Locking**

**Limitation of Method3 :** If we mark whole method as synchronized the task other then creating object also have to wait for one thread to complete it’s process

**Code:**

public class singleton4 {

    public static void main(String[] args) {

        Thread t1 = new Thread(

                new Runnable() {

                    public void run() {

                        COMPANY ATPL = COMPANY.createcompany();

                    }

                });

        Thread t2 = new Thread(

                new Runnable() {

                    public void run() {

                        COMPANY AMUL = COMPANY.createcompany();

                    }

                });

        t1.start();

        try {

            Thread.sleep(100);

        } catch (Exception e) {

        }

        t2.start();

    }

}

class COMPANY {

    static COMPANY obj;

    private COMPANY() {

        System.out.println("COMPANY CREATED SUCESSFULLY");

    }

    public static synchronized COMPANY createcompany() {

        if (obj == null) {

            synchronized (COMPANY.class) {

                if (obj == null) {

                    obj = new COMPANY();

                }

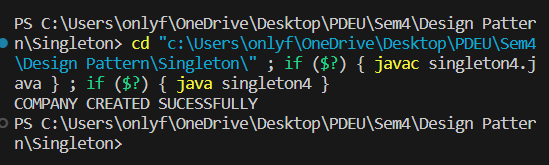
            }

        }

        return obj;

    }

}

**Output:**

**Method 5:** Just another way to implement Efficiently

**Code:**

public class singleton5 {

    public static void main(String[] args) {

        COMPANY AMUL = COMPANY.INSTANCE;

        COMPANY AMUL2 = COMPANY.INSTANCE;

    }

}

enum COMPANY {

    INSTANCE;

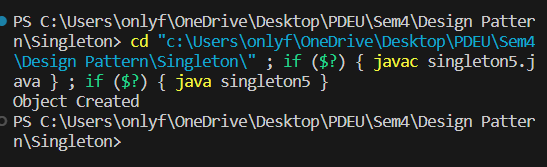
    COMPANY() {

        System.out.println("Object Created");

    }

}

**Output:**

****