Experiment#		Student ID	
Date		Student Name	@KLWKS_BOT THANOS

2. Creational Design Pattern I

Aim/Objective: To understand the concept and implementation of Singleton for the real time scenarios.

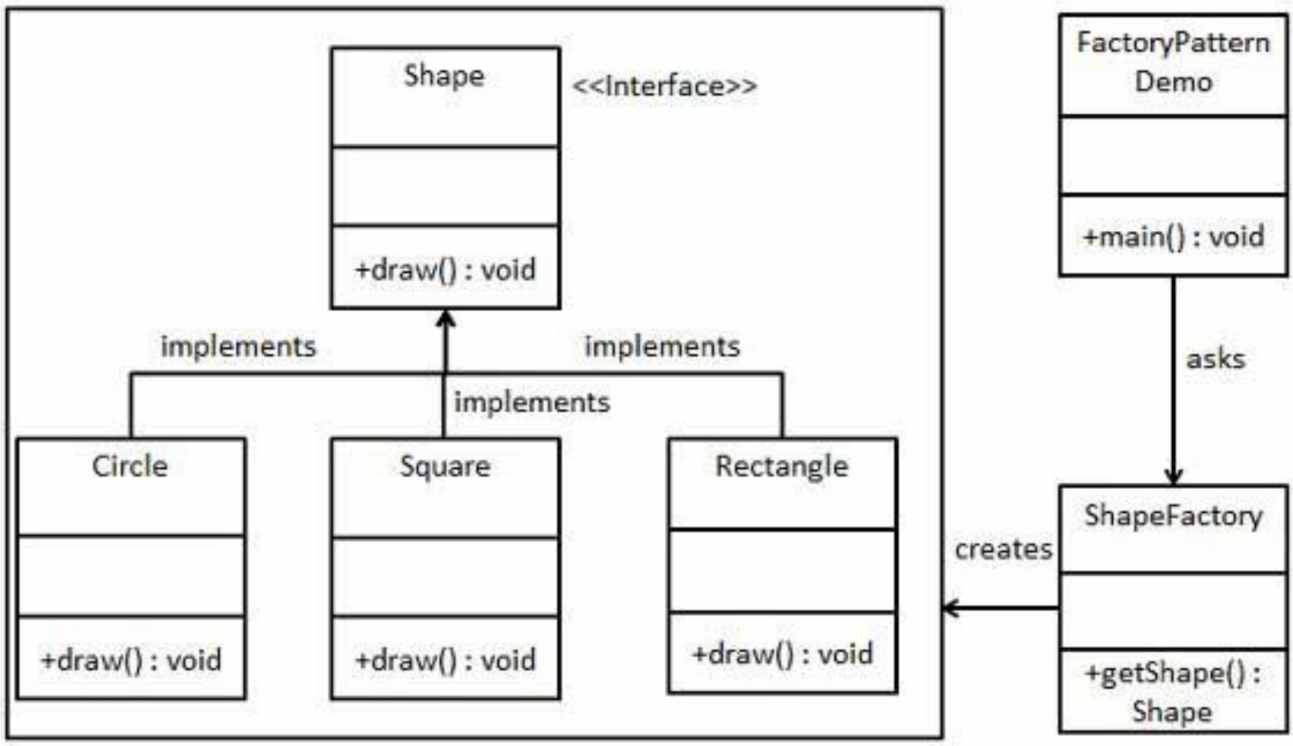
Description: The student will understand the concept of Creational Design Patterns (Singleton)

Pre-Requisites: Classes and Objects in Java

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1) Draw the UML Relationship Diagram for Factory Design Pattern for any customized scenario.



In-Lab:

1) Write a Java Program on Singleton Pattern for Real-life Logging Syst

One common use case for the Singleton pattern is in implem systems, where it's important to ensure that only one instance of the throughout the entire application.

Note: The Logger class uses the Singleton pattern to ensure instance of the logger is created and used throughout the entire applicatio property provides a way to access the single instance of the logger, and can be used to log messages.

Procedure/Program:

```
package inlab;
public class Logger1
{
  private static Logger1 instance;
   private Logger1 ()
    {
    }
  Public static logger1 getInstance ()
  {
     if (instance == null)
       {
          instance= new Logger1();
       }
     return instance;
}
```

Experiment#			
Date		Student ID	
		Student Name	

```
public void log (string message)
  {
    System.out.println ("Logging message:" +message);
  }
}
```

2) In a server application, managing configurations like API keys, server settings, and connections is crucial. To ensure that there is only one configuration manager hand these settings, we can use the Singleton pattern. This ensures consistency and prev multiple instances from causing conflicts.

Implementation

A. **ConfigurationManager Class**: This class will use the Singleton patter manage server configurations.

B. **Properties and Methods**: It will have methods to get and set configura values.

```
public class ConfigurationManager {
private static ConfigurationManager instance;
private String apikey;
 private String serverSettings;
 private int maxConnections;
private ConfigurationManager() {
  this.apikey = " ";
  this.serverSettings = " ";
  this.maxConnections = 0;
 }
public static synchronized ConfigurationManager
                          getInstance() {
 if (instance == null) {
 instance = new ConfigurationManager();
}
```

```
return instance; }
public String getApikey() {
    return apikey;
} public void setApikey(String apikey) {
    this.apikey = apikey;
} public String getserverSettings() {
    return serverSettings;
} public void setServerSettings(String serverSettings) {
 this.serverSettings = serverSettings;
} public int getMaxConnections() {
    return maxConnections;
} public void setMaxConnections(int maxConnections) {
 this.maxConnections = maxConnections;
    }
 }
```

✓ **Data and Results:**

API Key: your_api_key_here

Server settings: server_settings_value

Max connections: 100

✓ **Analysis and Inferences:**

Singleton 'ConfigurationManager' class provides a solid foundation for managing configuration settings in a consistent and controlled manner.

VIVA-VOCE Questions (In-Lab):

1) Which classes are candidates of Singleton? Which kind of class do you make Singleton in Java?

Any class which you want to be available in the whole application and only one instance should be created in the whole can be made as a singleton candidate. Not only that, the runtime classes can be made into singleton candidates which would be the right decision.

2) State about Lazy Initialization.

Lazy initialization is a design pattern that defers the creation of an object until it is actually needed. This can improve performance and reduce resource comsumption, especially when the initialization of an object is costly or unnecessary at the start.

3) State about Eager Initialization.

Eager initialization is a design pattern where the object is created as soon as the class is loaded. This approach is opposite to Lazy insitialization, where the object is created only when needed.

4) Mention the cons of Singleton Design pattern.

Inflexibility: Singleton limits flexibility in scenarios where multiple instances are needed, as it strictly enforces a single instance constraint.

5) List the comparison between "Singleton vs Static Class".

Aspect	Singleton Class	Static Class
Instantiation	Can only be instantiated once.	Cannot be instantiated at all.
State Management	Can maintain state b/w method calls.	Doesnot maintain state b/w method calls.
Inheritance	Can be inherited.	Cannot be inherited.
Interface implementation	Can implement interface	Cannot implement interfaces.

Experiment#		Student ID	
Date		Student Name	

Post-Lab:

1) In a Banking process, create a Singleton pattern to manage the Login state of the user in all the operations like View Balance, Deposit & Withdraw.

Note: Create a Singleton Class to maintain the User Login-State.

Procedure/Program:

```
public class BankingProcess {
private UserLoginManager loginManager;
 public BankingProcess() {
   loginManager = UserLoginManager.getInstance();
} public void viewBalance() {
  if (loginManager.isLoggedIn()){
    System.out.println("Balance: $1000");
}else {
    System.out.println("Please login to view your balance");
    }
}  public void deposit (double amount){
if (loginManager.isLoggedIn()) {
System.out.println("$" + amount + " deposit successfully");
}else {
System.out.println("Please login to make a deposit");
```

```
    }
} public void withdraw (double amount) {
if (loginManager.isLoggedIn()) {
System.out.println ("$" + amount + "withdraw
                                successfully");
} else {
System.out.println ( "Please login to make withdraw");
    }
  }
}
```

Experiment#		Student ID	
Date		Student Name	

✓ **Data and Results:**

Showing Balance.

Balance = 20000

Depositing 10000 amount.

Balance = 30000

Showing Balance.

Balance = 30000

✓ **Analysis and Inferences:**

Using a singleton pattern to manage the user login state in banking system is effective for maintaining a consistent state across various operations.

Evaluator Remark (if Any):	**Marks Secured: ____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.