

Acknowledgement

First and foremost, I would like to thank my god for giving me this knowledge to make this assignment in successful manner. And sincere thanks to Mr.B.Sabashan for the support and guidance towards me in making this assignment a great and an important aspect for all us.

And next I would like to thank my parents for making arrangement to study this HND program and specially for provide me all facilities in doing this assignment. The importance that I faced in my life after completing my Advanced Level examination and after coming to this field I feel happy because of my friends who are with me the way how they are with me and for their excellent support.

Once again thank you for your great support in the successful completion my thesis.

J.Thadsayini

Contents

Acknowledgement.....	1
LO1: Examine the key components related to the object-oriented programming paradigm,analyzing design pattern types	6
Part 1.1.....	6
Charecteristics of the object-orientated paradigm and about the different relationships among classes such as the generalization,associative,composition and aggregations.	6
What is Object-Oriented Programming.....	6
The important features of object-oriented programming	6
Characteristics of OOP	7
Part 1.2.....	9
Design patterns for creational, Structural and behavioral.....	9
Design pattern	9
Advantage of design pattern	9
When should we use the design patterns.....	9
Types of design patterns.....	10
Part 1.3.....	16
Analyse the relationship between the object-oriented paradigm and design patterns.	16
Object-Oriented Paradigm	16
benefits of Object-Oriented Programming	16
Design Patterns	17
Relationship between object-oriented paradigms and design patterns	18
LO2-Design a series of UML class diagrams.....	19
Part 2.1.....	19
Design and define sequence and class diagram for above scenario using UML tool.	19
What is a Sequence Diagram.....	19
Basic Sequence Diagram Notations.....	19

What is class diagram	21
Purpose of Class Diagrams	21
Sequence Diagram	22
Class Diagram	23
Part 2.2.....	24
Discuss in detail what Test-Driven Development(TDD).....	24
What is Test-Driven Development?.....	24
Why Test Driven Development.....	24
Need for Test Driven Development	25
Test Driven Development with Java	25
Test Driven Development Tools	26
1.Mockito for Rest API Testing	26
2.JMeter for Load/Performance Testing	26
3.JUnit for Unit Tests	26
Concluding Test Driven Development	27
LO3-Implement code applying design patterns	28
Part 3.1.....	28
Develop code that implements design patterns and utility technique to produce secure code using java and industry recommended IDE.	28
Library Management System Screen Shots	29
Part 3.2.....	34
test case	34
Unit testing.....	34
LO4-Investigate scenarios with respect to design patterns.....	37
Part 4.1.....	37
Create a detail report that discusses a range of design patterns and their examples	37
Singleton Pattern	37

Factory Pattern	38
Decorator Pattern.....	39
References	41

Introduction

I am Thadsayini Jeyaratnam. I have been studying computer system development, batch 12 at BCAS Campus. This is 2nd semester assignment. The assignment have four learning outcomes.mainly,I have to present the java IDEs and OOP. The purpose of this section to introduction theAdvanced programming to individuals and organizations. With the intention to raise the general awareness and understanding of advanced programming. Get more knowledge about advanced programming through this assignment.Overcome,This is introduction about assignment

LO1: Examine the key components related to the object-oriented programming paradigm,analyzing design pattern types

Part 1.1

Charecteristics of the object-orientated paradigm and about the different relationships among classes such as the generalization,associative,composition and aggregations.

What is Object-Oriented Programming

OOP is a programming paradigm based upon objects that aims to incorporate the advantages of modularity and reusability. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

The important features of object-oriented programming are –

- Bottom-up approach in program design
- Programs organized around objects, grouped in classes
- Focus on data with methods to operate upon object's data
- Interaction between objects through functions
- Reusability of design through creation of new classes by adding features to existing classes

Some examples of object-oriented programming languages are

- C++
- Java C#
- Python
- Ruby
- PHP.

Characteristics of OOP

Encapsulation – Encapsulation is capturing data and keeping it safely and securely from outside interfaces.

Inheritance- This is the process by which a class can be derived from a base class with all features of base class and some of its own. This increases code reusability.

Polymorphism- This is the ability to exist in various forms. For example an operator can be overloaded so as to add two integer numbers and two floats.

Abstraction- The ability to represent data at a very conceptual level without any details.

Generalization

In the generalization process, the common characteristics of classes are combined to form a class in a higher level of hierarchy, i.e., subclasses are combined to form a generalized super-class. It represents an “is – a – kind – of” relationship. For example, “car is a kind of land vehicle”, or “ship is a kind of water vehicle”.

Association

Association is a group of links having common structure and common behavior. Association depicts the relationship between objects of one or more classes. A link can be defined as an instance of an association.

Degree of an Association

Degree of an association denotes the number of classes involved in a connection. Degree may be unary, binary, or ternary.

- A unary relationship connects objects of the same class.
- A binary relationship connects objects of two classes.
- A ternary relationship connects objects of three or more classes.

Cardinality Ratios of Associations

Cardinality of a binary association denotes the number of instances participating in an association. There are three types of cardinality ratios, namely –

- **One-to-One** – A single object of class A is associated with a single object of class B.
- **One-to-Many** – A single object of class A is associated with many objects of class B.
- **Many-to-Many** – An object of class A may be associated with many objects of class B and conversely an object of class B may be associated with many objects of class A.

Aggregation or Composition

Aggregation or composition is a relationship among classes by which a class can be made up of any combination of objects of other classes. It allows objects to be placed directly within the body of other classes. Aggregation is referred as a “part-of” or “has-a” relationship, with the ability to navigate from the whole to its parts. An aggregate object is an object that is composed of one or more other objects.

Example

In the relationship, “a car has-a motor”, car is the whole object or the aggregate, and the motor is a “part-of” the car. Aggregation may denote –

- **Physical containment** – Example, a computer is composed of monitor, CPU, mouse, keyboard, and so on.
- **Conceptual containment** – Example, shareholder has-a share.

Part 1.2

Design patterns for creational, Structural and behavioral.

Design pattern

- Design Patterns are very popular among software developers.
- A design pattern is a well-described solution to a common software problem.

Advantage of design pattern

- They are reusable in multiple projects.
- They provide the solutions that help to define the system architecture.
- They capture the software engineering experiences.
- They provide transparency to the design of an application.
- They are well-proved and testified solutions since they have been built upon the knowledge and experience of expert software developers.
- Design patterns don't guarantee an absolute solution to a problem. They provide clarity to the system architecture and the possibility of building a better system.

When should we use the design patterns?

- We must use the design patterns during the analysis and requirement phase of SDLC(Software Development Life Cycle).
- Design patterns ease the analysis and requirement phase of SDLC by providing information based on prior hands-on experiences.

Types of design patterns

Creational patterns

- how objects are instantiated

Structural patterns

- how objects / classes can be combined

Behavioral patterns

- how objects communicate

Concurrency patterns

- how computations are parallelized / distributed

Creational patterns

Creational Design Patterns are concerned with the way in which objects are created. They reduce complexities and instability by creating objects in a controlled manner. The new operator is often considered harmful as it scatters objects all over the application. Over time it can become challenging to change an implementation because classes become tightly coupled. Creational Design Patterns address this issue by decoupling the client entirely from the actual initialization process.

Creational Patterns	
Abstract Factory	Creates an instance of several families of classes
Builder	Separates object construction from its representation
Factory Method	Creates an instance of several derived classes
Prototype	A fully initialized instance to be copied or cloned
Singleton	A class of which only a single instance can exist

Then , we'll discuss four types of Creational Design Pattern:

1. Singleton – Ensures that at most only one instance of an object exists throughout application
2. Factory Method – Creates objects of several related classes without specifying the exact object to be created
3. Abstract Factory – Creates families of related dependent objects
4. Builder – Constructs complex objects using step-by-step approach

Let's now discuss each of these patterns in detail.

1. Singleton Design Pattern

The Singleton Design Pattern aims to keep a check on initialization of objects of a particular class by ensuring that only one instance of the object exists throughout the Java Virtual Machine.

A Singleton class also provides one unique global access point to the object so that each subsequent call to the access point returns only that particular object.

So here, we're going to follow a more optimal approach that makes use of a static inner class:

```
1 public class Singleton {  
2     private Singleton() {}  
3  
4     private static class SingletonHolder {  
5         public static final Singleton instance = new Singleton();  
6     }  
7  
8     public static Singleton getInstance() {  
9         return SingletonHolder.instance;  
10    }  
11 }
```

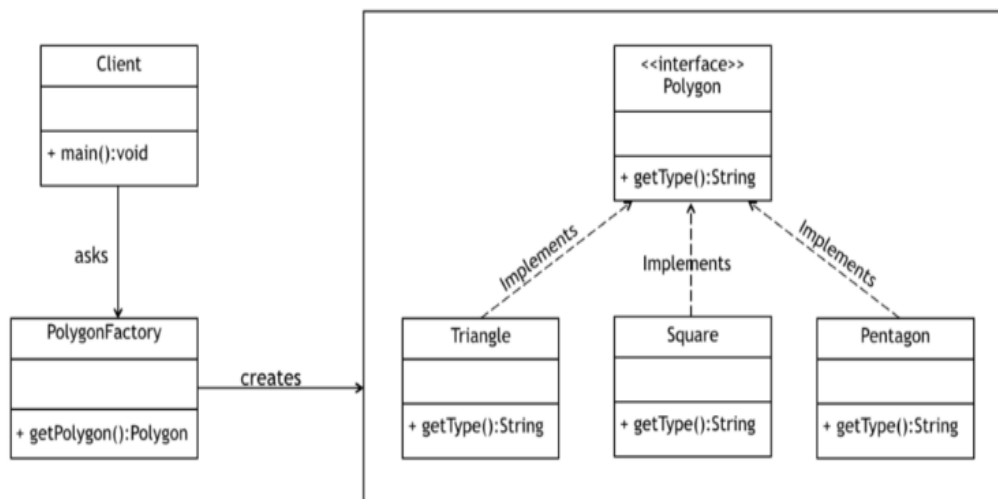
When to Use Singleton Design Pattern

- For resources that are expensive to create (like database connection objects)
- It's good practice to keep all loggers as Singletons which increases performance
- Classes which provide access to configuration settings for the application
- Classes that contain resources that are accessed in shared mode

2. Factory Method Design Pattern

- The Factory Design Pattern or Factory Method Design Pattern is one of the most used design patterns in Java.
- This pattern delegates the responsibility of initializing a class from the client to a particular factory class by creating a type of virtual constructor.
- To achieve this, we rely on a factory which provides us with the objects, hiding the actual implementation details. The created objects are accessed using a common interface.

In this example, we'll create a Polygon interface which will be implemented by several concrete classes. A Polygon Factory will be used to fetch objects from this family:



Let's first create the Polygon interface:

```

1 | public interface Polygon {
2 |     String getType();
3 | }

```

Next, we'll create a few implementations like Square, Triangle, etc. that implement this interface and return an object of Polygon type.

Now we can create a factory that takes the number of sides as an argument and returns the appropriate implementation of this interface:

```

1 | public class PolygonFactory {
2 |     public Polygon getPolygon(int numberOfSides) {
3 |         if(numberOfSides == 3) {
4 |             return new Triangle();
5 |         }
6 |         if(numberOfSides == 4) {
7 |             return new Square();
8 |         }
9 |         if(numberOfSides == 5) {
10 |             return new Pentagon();
11 |         }
12 |         if(numberOfSides == 7) {
13 |             return new Heptagon();
14 |         }
15 |         else if(numberOfSides == 8) {
16 |             return new Octagon();
17 |         }
18 |         return null;
19 |     }
20 | }

```

When to Use Factory Method Design Pattern

- When the implementation of an interface or an abstract class is expected to change frequently
- When the current implementation cannot comfortably accommodate new change
- When the initialization process is relatively simple, and the constructor only requires a handful of parameters

3. Abstract Factory Design Pattern

In the previous section, we saw how the Factory Method design pattern could be used to create objects related to a single family.

By contrast, the Abstract Factory Design Pattern is used to create families of related or dependent objects. It's also sometimes called a factory of factories.

4. Builder Design Pattern

The Builder Design Pattern is another creational pattern designed to deal with the construction of comparatively complex objects.

When the complexity of creating object increases, the Builder pattern can separate out the instantiation process by using another object to construct the object.

This builder can then be used to create many other similar representations using a simple step-by-step approach.

Structural design patterns

Structural design patterns are design patterns that ease the design by identifying a simple way to realize relationships between entities or defines a manner for creating relationships between objects.

Structural Patterns	
Adapter	Match interfaces of different classes
Bridge	Separates an object's interface from its implementation
Composite	A tree structure of simple and composite objects
Decorator	Add responsibilities to objects dynamically
Facade	A single class that represents an entire subsystem
Flyweight	A fine-grained instance used for efficient sharing
Proxy	An object representing another object

Behavioral pattern

Behavioral pattern explains how objects interact. It describes how different objects and classes send messages to each other to make things happen and how the steps of a task are divided among different objects. Where Creational patterns mostly describe a moment of time (the instant of creation), and Structural patterns describe a more or less static structure, Behavioral patterns describe a process or a flow

Behavioral Patterns	
Chain of Resp.	A way of passing a request between a chain of objects
Command	Encapsulate a command request as an object
Interpreter	A way to include language elements in a program
Iterator	Sequentially access the elements of a collection
Mediator	Defines simplified communication between classes
Memento	Capture and restore an object's internal state
Observer	A way of notifying change to a number of classes
State	Alter an object's behavior when its state changes
Strategy	Encapsulates an algorithm inside a class
Template Method	Defer the exact steps of an algorithm to a subclass
Visitor	Defines a new operation to a class without change

Part 1.3

Analyse the relationship between the object-oriented paradigm and design patterns.

Object-Oriented Paradigm

Object-Oriented Paradigm is where we focus real life objects while programming any solution. By focusing real life objects we mean that over solutions revolves around different objects, which represent respective objects in real life situation.

benefits of Object-Oriented Programming

Maintainable

Object-Oriented Paradigm methods make code more maintainable. Identifying the source of errors becomes easier because objects are self-contained. The principles of good methods design contribute to a system's maintainability.

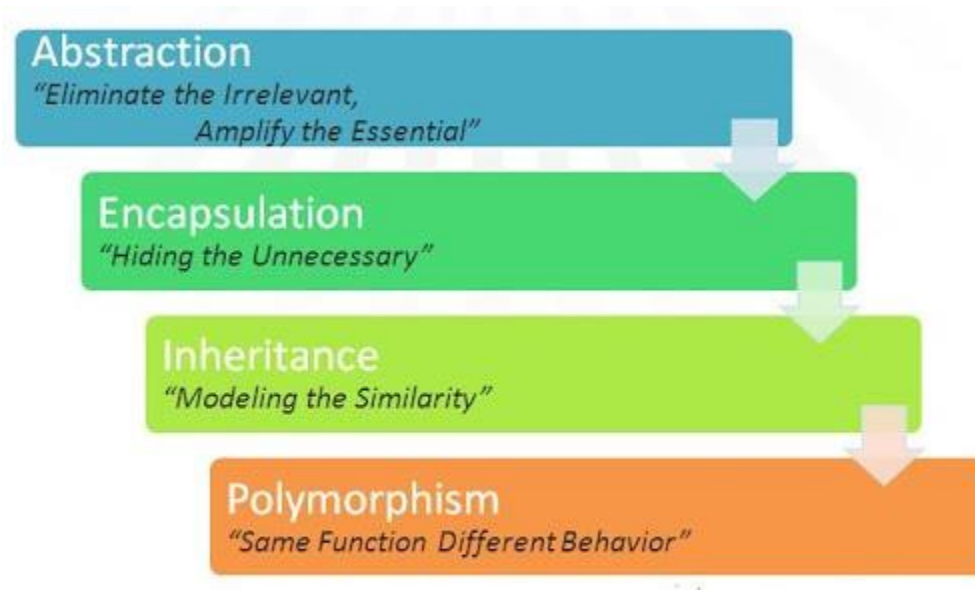
Reusable

Since objects contain both data and functions that act on data, objects can be thought of as self-contained "boxes". This makes it easy to reuse code in new systems.

Scalable

Object-Oriented applications are more scalable than their structured programming roots. As an object's interface provides a roadmap for reusing an object, it also provides you with all the information you need to replace the object without affecting others. This makes it easy to replace old and inefficient code with faster algorithms.

The 4 main characteristic of Object-Oriented Paradigm



Design Patterns

A design pattern provides a general reusable solution to a common design problem.

These are well-tested solutions to common problems and issues we run into in software development. They are best practices for solving common software design problems that occur again and again.

There are many design patterns, and they are categorized under three main categories: Structural, Behavioral, and Creational.

Relationship between object-oriented paradigms and design patterns

OOP is itself a paradigm for programming. Other programming paradigms are procedural programming, functional programming and such.

Design patterns are nothing but a solution to a recurring problem.. but this solution is robust, scalable, readable and flexible. They can exist for other programming paradigms too.

When you first start learning to program, your initial focus is on the language. You learn the syntax and the structure. Once you become comfortable and able to put statements together to solve programming problems, you are then free to think more about programming.

As experienced programmers have thought about the best ways to approach the building of an application or the best way to solve a particular problem, they have developed paradigms and design patterns. The smart developers draw on this expertise to improve their own programming skills by adopting paradigms and design patterns.

A paradigm is simply a style or an approach to programming. For example, Object Oriented programming is a paradigm. If you follow this paradigm you use objects to approach the overall program. The objects contain data and behaviors and you connect them in logical ways to successfully solve the task at hand.

Now a design pattern is a tried and tested solution to a common programming pattern. It could be considered a best practice. If you approach a program using an Object Oriented paradigm, there are a number of design patterns you can then draw on to solve specific problems.

So how do we apply this to JavaScript? JavaScript can support many different paradigms and there have been numerous design patterns developed to solve individual problems that arise with each paradigm or across paradigms. JavaScript is flexible. There is not one way to accomplish something and therefore numerous approaches have been developed. You can draw on those approaches and you can mix and match them as needed.

LO2-Design a series of UML class diagrams

Part 2.1

Design and define sequence and class diagram for above scenario using UML tool.

What is a Sequence Diagram?

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time. They're also called event diagrams. A sequence diagram is a good way to visualize and validate various runtime scenarios. These can help to predict how a system will behave and to discover responsibilities a class may need to have in the process of modeling a new system.

Basic Sequence Diagram Notations

1.Class Roles or Participants

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.



2.Activation or Execution Occurrence

Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process or waiting for a reply message, use a thin gray rectangle placed vertically on its lifeline.



Activation or Execution Occurrence

3.Destroying Objects

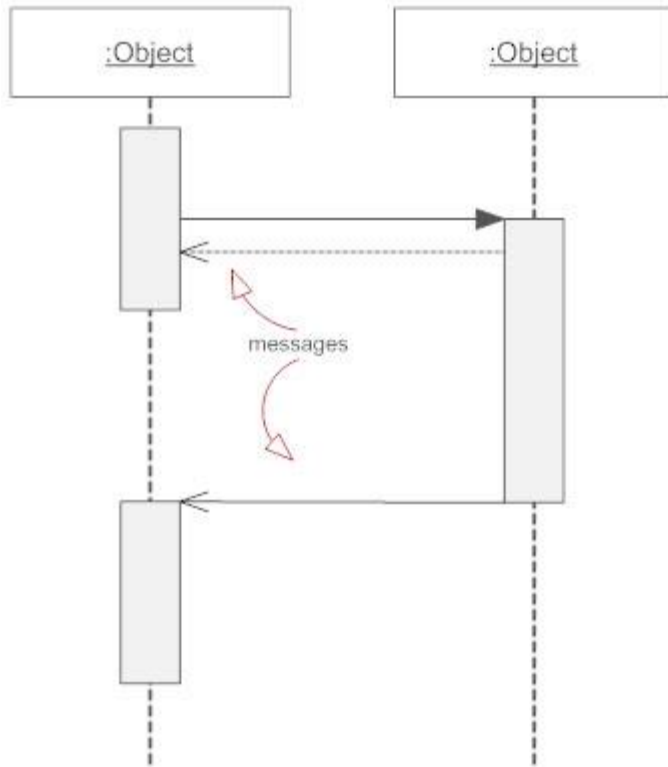
Objects can be terminated early using an arrow labeled "<< destroy >>" that points to an X. This object is removed from memory. When that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence.

4.Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [].

5.Messages

Messages are arrows that represent communication between objects. Use half-arrowed lines to represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks. For message types, see below.



What is class diagram?

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

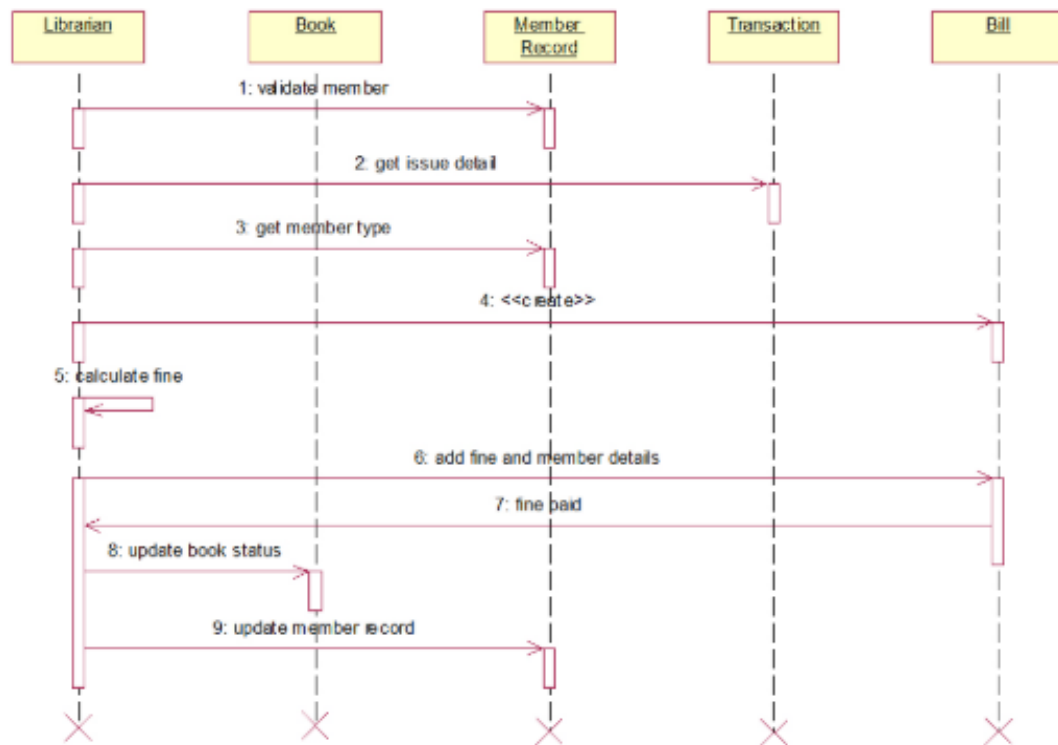
Purpose of Class Diagrams

- ❖ Shows static structure of classifiers in a system
- ❖ Diagram provides basic notation for other structure diagrams prescribed by UML
- ❖ Helpful for developers and other team members too
- ❖ Business Analysts can use class diagrams to model systems from business perspective

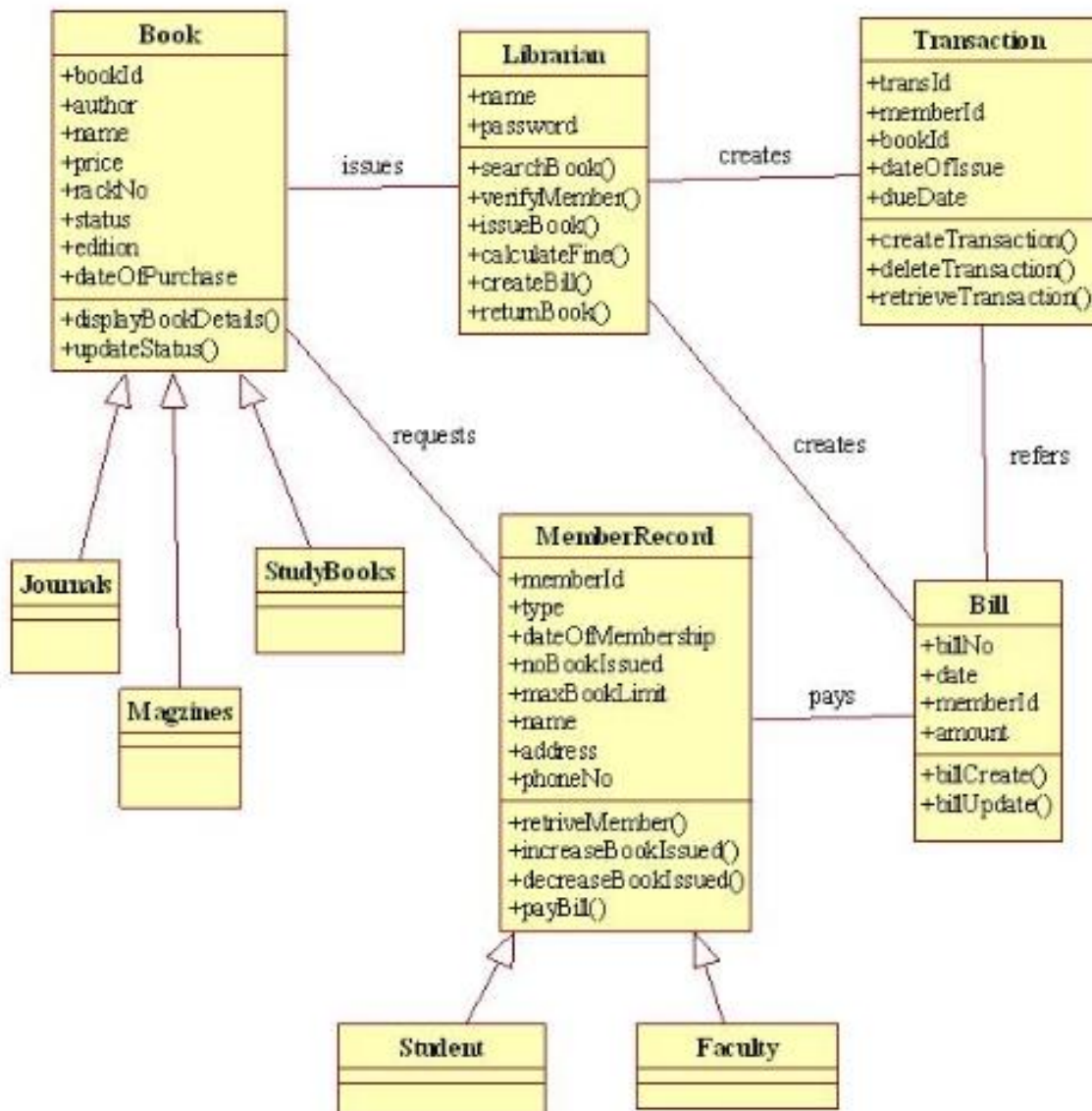
A UML class diagram is made up of:

- ❖ A set of classes and
- ❖ A set of relationships between classes

Sequence Diagram



Class Diagram



Part 2.2

Discuss in detail what Test-Driven Development(TDD)

What is Test-Driven Development?

Test-Driven Development (TDD) is a software development process which includes test-first development. It means that the developer first writes a fully automated test case before writing the production code to fulfil that test and refactoring. Steps for the same are given below –

- Firstly, add a test.
- Run all the tests and see if any new test fails.
- Update the code to make it pass the new tests.
- Run the test again and if they fail then refactor again and repeat.

Why Test Driven Development?

Advantages of Test Driven Development

- It gives a way to think through our requirements or design before we write our functional code.
- It is a programming technique that enables us to take small steps during building software.
- It is more productive in nature rather as compared attempting to code in large steps.

Let's take an example, assume that you write some code and then compile it and then test it and maybe there are chances of failure. In this case, it becomes easy to find and fix those defects if you've written two new lines of code than a thousand.

Basically, a Test Driven Development is –

The most efficient and attractive way to proceed in smaller and smaller steps.

Following Test Driven Development means –

- Less bugs
- Higher quality software
- Focus on single functionality at a given point in time

Need for Test Driven Development

- Requirements – Drive out requirement issues early.
- Rapid Feedback – Many small changes Vs One big change.
- Values Refactoring – Refactor often to lower impact and risk.
- Design to Test – Testing driving good design practice.
- Tests as information – Documenting decisions and assumptions.

Test Driven Development with Java

In the Java community, Test Driven Development plays an important role in designing and implementation of a software/program. Test Driven Development helps the programmer in several ways, such as –

- Improving the code
- Side by side, increasing the programmer's productivity.

Using Test Driven Development concept in our programming skills ,

- Will save our time which is getting wasted for rework.
- Able to identify the error/problem quicker and faster.
- The programmer will be able to write small classes which will be focused only on a single functionality instead of writing the big classes.

Test Driven Development Tools

1.Mockito for Rest API Testing



Mockito is designed as an open source testing framework for Java which is available under an MIT License. Mockito allows programmers to create and test double objects (mock objects) in automated unit tests for the purpose of Test-driven Development (TDD). In simple words, we can say that Mockito is a framework that we specifically use to efficiently write certain kind of tests.

2.JMeter for Load/Performance Testing



Apache JMeter may be used to test performance both on static and dynamic resources, Web dynamic applications (Mainly for Load/Performance testing). Basically, it is used to simulate a heavy load on a server, group of servers, network or object to test its strength or to analyze overall performance under different load types.

3.JUnit for Unit Tests



JUnit is a unit testing framework designed for Java programming language. Since unit tests are the smallest elements in the test automation process. With the help of unit tests, we can check the business logic of any class.

So JUnit plays an important role in the development of a test-driven development framework. It is one of the families of unit testing frameworks which is collectively known as the xUnit that originated with SUnit.

Concluding Test Driven Development

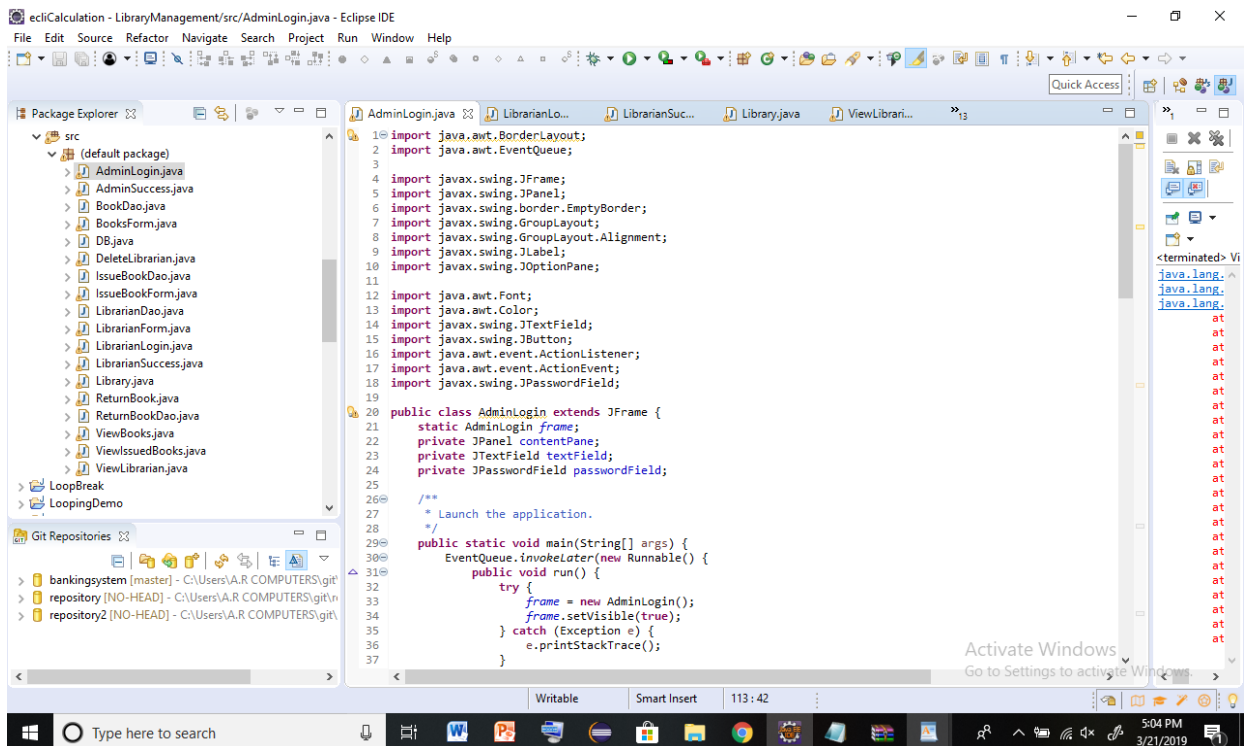
In the end, it is safe to say that Test Driven Development must be adopted by as many developers as possible, in order increase their productivity and improve not only the code quality but also to increase the productivity and overall development of software/program. TDD also leads to more modularized, flexible and extensible code.

LO3-Implement code applying design patterns

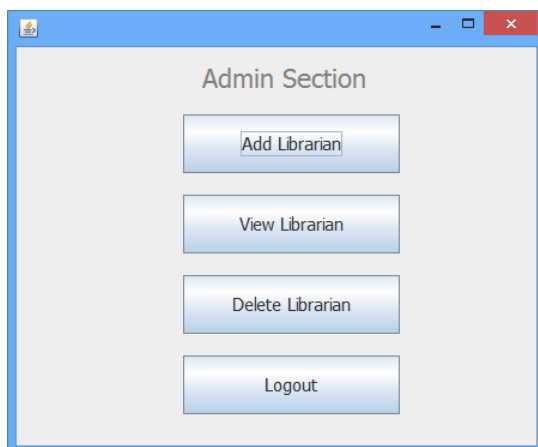
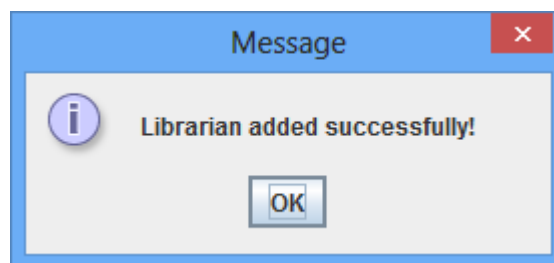
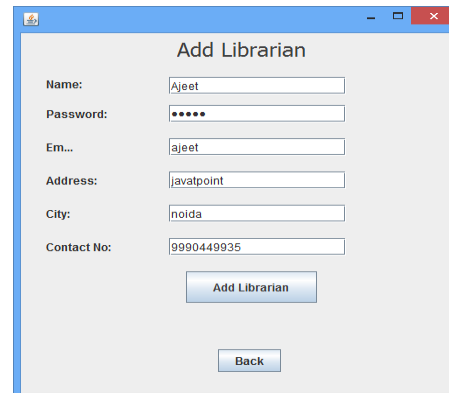
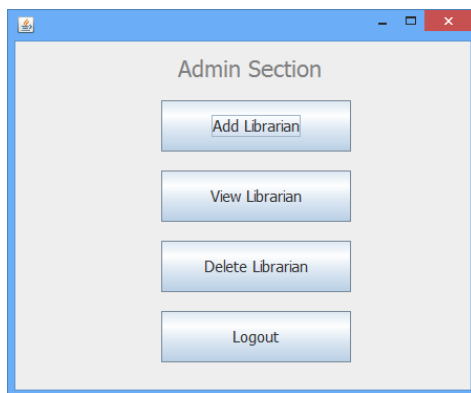
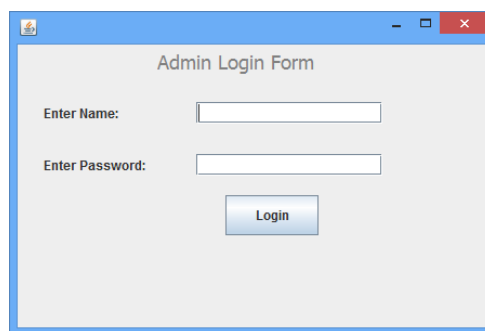
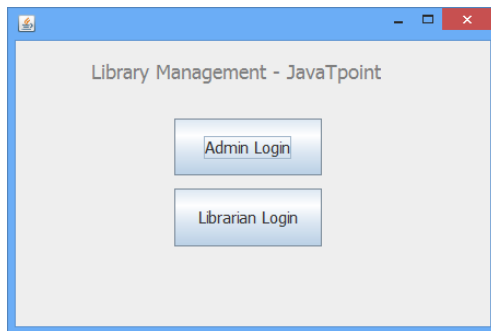
Part 3.1

Develop code that implements design patterns and utility technique to produce secure code using java and industry recommended IDE.

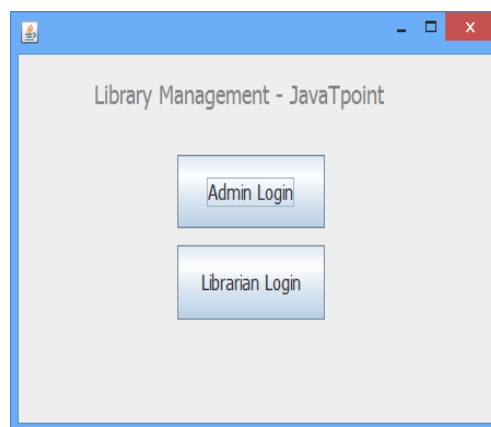
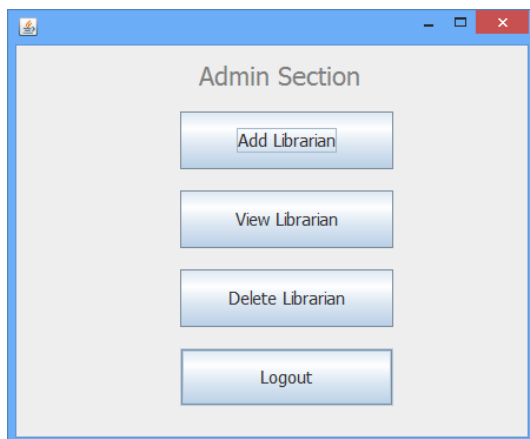
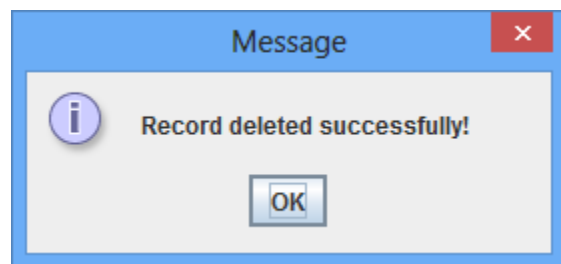
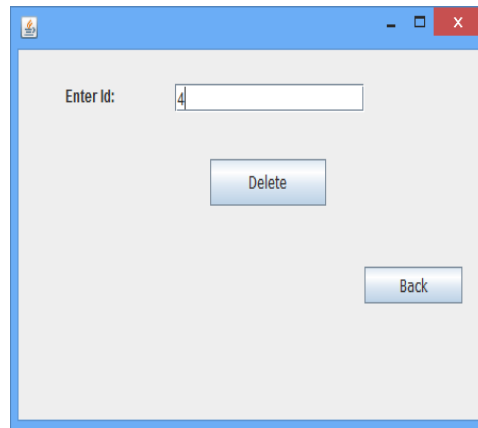
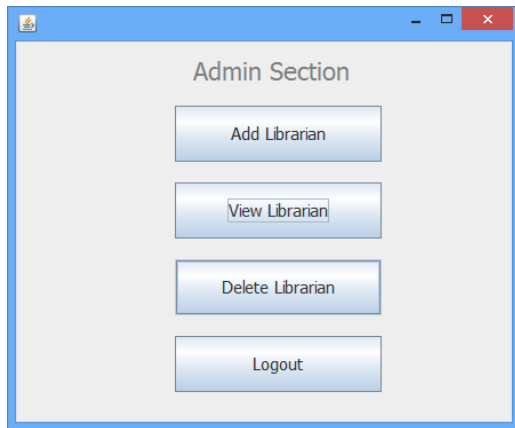
Coding send in bitbukket



Library Management System Screen Shots



id	name	password	email	address	city	contact
1	Prabhakar	ppp	prabhakar@gmail.com	javatpoint	noida	9998328238
4	sumedh	sumesh	sumesh@gmail.com	Kuch Bhi	noida	93823932823
6	abhi	abhi	abhi@gmail.com	javatpoint	noida	92393282323
7	Ajeet	ajeet	ajeet	javatpoint	noida	9990449935



Librarian Login Form

Enter Name:

Enter Password:

Librarian Section - JavaTpoint

Add Books

Call No:


Name:

Auth...:

Publisher:

Quantity:

Message

 **Books added successfully!**

Librarian Section - JavaTpoint

	id	callno	name	author	publisher	quantity	issued	added_date
1	A@4	C in Depth	Shrivastav	BPB	2	2	2016-07-20 ...	
2	B@1	DBMS	Korth	Pearson	3	0	2016-07-19 ...	
3	G@12	Let's see	Yashwant K.	BPB	10	0	2016-07-19 ...	
4	K@8	Networking	Forouzan	Tata Mc Gra...	5	0	2016-07-20 ...	

Librarian Section - JavaTpoint

Add Books
View Books
Issue Book
View Issued Books
Return Book
Logout

Issue Book
Book Callno: K@8
Student Id: 100
Student Name: Vimal
Student Contact: 9990449936

Issue Book
Back

Note: Please check Student ID Carefully before issuing book!

Message

i
Book issued successfully!

OK

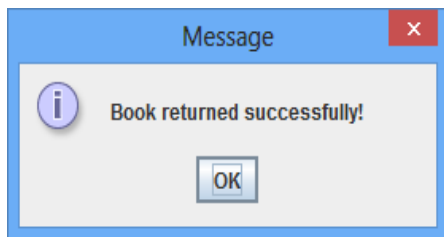
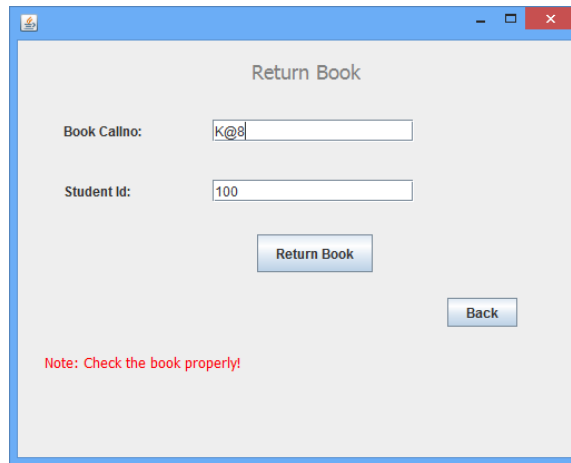
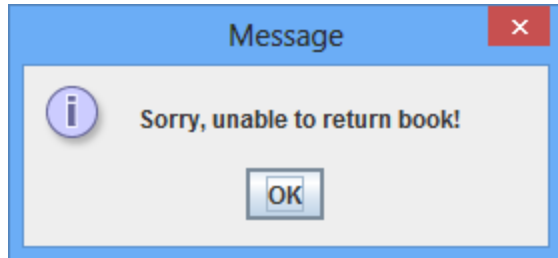
Librarian Section - JavaTpoint

Add Books
View Books
Issue Book
View Issued Books
Return Book
Logout

Return Book
Book Callno: K@4
Student Id: 100

Return Book
Back

Note: Check the book properly!



Part 3.2

test case

A test case is a specification of the inputs, execution conditions, testing procedure, and expected results that define a single test to be executed to achieve a particular software testing objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.

The process of developing test cases can also help find problems in the requirements or design of an application.

Unit testing

unit testing is a level of software testing where individual units/ components of a software are tested. ... A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. In procedural programming, a unit may be an individual program, function, procedure, etc.

Unit tests are used to test individual code components and ensure that code works the way it was intended to. Unit tests are written and executed by developers. Most of the time a testing framework like JUnit or TestNG is used. Test cases are typically written at a method level and executed via automation.

Unit tests detect changes that may break a design contract. They help with maintaining and changing the code. Unit testing reduces defects in the newly developed features or reduces bugs when changing the existing functionality. Unit testing verifies the accuracy of the each unit.

```

25
26 /**
27  * Launch the application.
28  */
29 public static void (String[] args) {
30     EventQueue.invokeLater(new Runnable() {
31         public void run() {
32             try {
33                 frame = new AdminLogin();
34                 frame.setVisible(true);
35             } catch (Exception e) {
36                 e.printStackTrace();
37             }
38         }
39     });
40 }

```

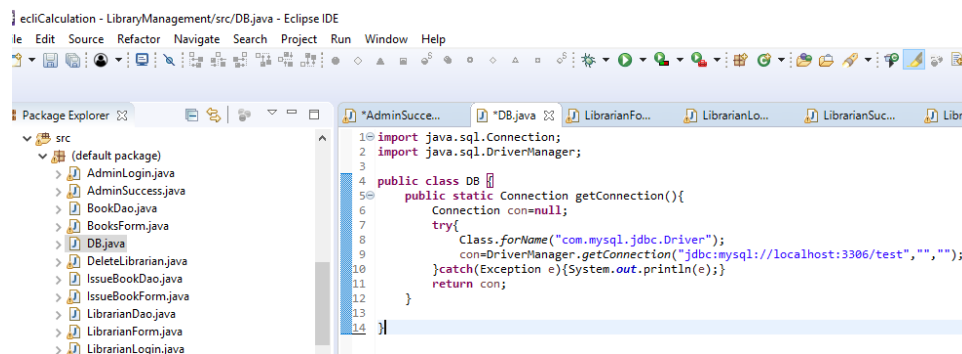
I didn't include "main" . main() : it is a method or a function name.

```

113 .addComponent(btnLogout, GroupLayout.PREFERRED_SIZE, 49, GroupLayout.PREFERRED_SIZE)
114 );
115 .addContainerGap(21, Short.MAX_VALUE))
116 contentPane.setLayout(g1_contentPane);
117 }
118

```

I didn't put "}" this is very important for coding. this is end use symbol.

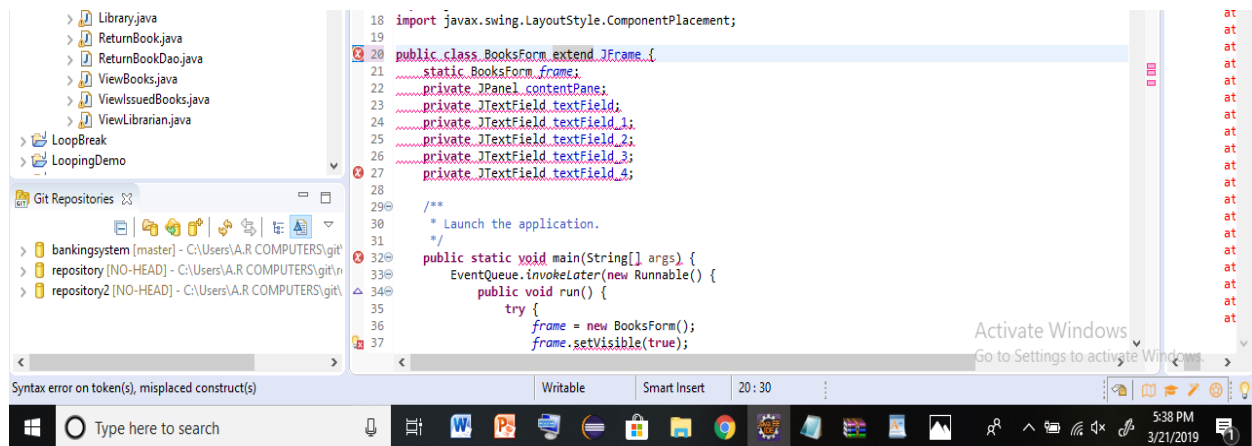


```

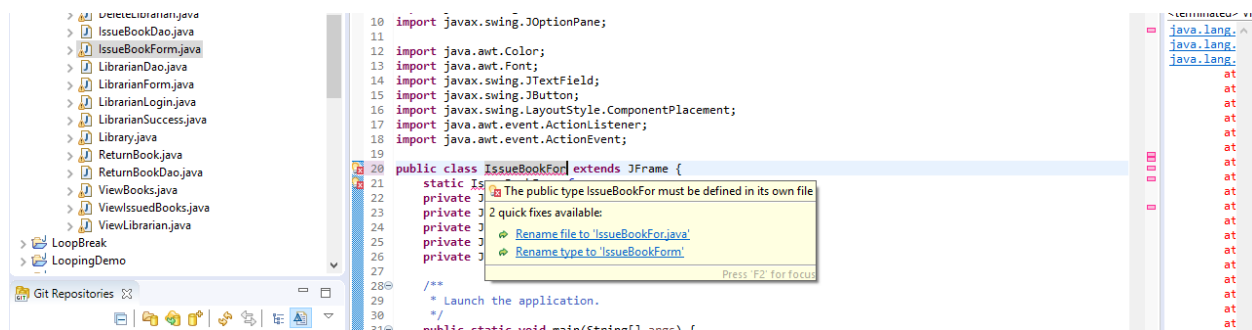
1  import java.sql.Connection;
2  import java.sql.DriverManager;
3
4  public class DB {
5      public static Connection getConnection(){
6          Connection con=null;
7          try{
8              Class.forName("com.mysql.jdbc.Driver");
9              con=DriverManager.getConnection("jdbc:mysql://localhost:3306/test","","");
10             }catch(Exception e){System.out.println(e);}
11             return con;
12         }
13     }
14 }

```

I didn't add return method. It does not need to contain a return statement, but it may do so. In such a case, a return statement can be used to branch out of a control flow block and exit the method and is simply used like this: return; If you try to return a value from a method that is declared void , you will get a compiler error.



This is spelling error. I put extend but correct spelling is Extends.



Class name error.so, rename

LO4-Investigate scenarios with respect to design patterns

Part 4.1

Create a detail report that discusses a range of design patterns and their examples

A design pattern is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

There are three basic kinds of design patterns:

- structural.
- Creational
- Behavioral

Singleton Pattern

The singleton pattern is one of the simplest design patterns in Java. This type of design pattern falls under the creational pattern as this pattern provides one of the best ways to create an object. This pattern involves a single class which is responsible for creating an instance while making sure that only a single object is created. This class provides a way to access its only object which can be accessed directly without the need to instantiate the object of the class.

```

public class SingletonClass {

    //create an object of SingleObject
    private static SingletonClass instance = new SingletonClass();

    //make the constructor private so that this class cannot be
    //SingletonClass
    private SingletonObject(){}

    //Get the only object available
    public static SingletonClass getInstance(){
        return instance;
    }

    public void showMessage(){
        System.out.println("Hello Java!");
    }
}

public class SingletonPatternDemo {
    public static void main(String[] args) {

        //illegal construct
        //Compile Time Error: The constructor SingleObject() is not vis:
        //SingletonClass object = new SingletonClass();

        //Get the only object available
        SingletonClass object = SingletonClass.getInstance();

        //show the message
        object.showMessage();
    }
}

```

Factory Pattern

The factory pattern provides an interface for creating families of related or dependent objects without specifying their concrete classes.

```

public class ShapeFactory {

    //use getShape method to get object of type shape
    public Shape getShape(String shapeType){
        if(shapeType == null){
            return null;
        }
        if(shapeType.equalsIgnoreCase("Quadrilateral")){
            return new Quadrilateral();
        }
        else if(shapeType.equalsIgnoreCase("Parallelogram")){
            return new Parallelogram();
        }

        return null;
    }
}

```

Decorator Pattern

The decorator pattern allows a user to add new functionality to an existing object without altering its structure. This type of design pattern falls under the structural pattern as this pattern acts as a wrapper to existing class. This pattern creates a decorator class which wraps the original class and provides additional functionality keeping class methods signature intact.

So, finally each patterns are used to coding, there are very important.

Conclusion

In this Advanced programming assignment I have learnt so many things through learning out comes one to four. I have improve my knowledge . So,I have learnt many things there are programming languages,I learn about how to create library management system in online. There are help to improve my knowledge. I got informations about advanced programming through this assignment by the help of internet, library books and networking handouts.so,finally I got many informations about networking and I have improve my knowledge. I assumed myself as I achieve the pass, merit and distinction criteria

References

- YouTube. (2019). *Library Management System in Java(Netbeans) Complete Project (Step by Step)* - YouTube. [online] Available at: https://www.youtube.com/playlist?list=PL_Ke9hJMFeR_TWfBOug40-2uSANxa8dtE [Accessed 13 Mar. 2019].
- www.javatpoint.com. (2019). *Library Management System in Java Swing Project* - javatpoint. [online] Available at: <https://www.javatpoint.com/library-management-system-in-java-swing> [Accessed 15 Mar. 2019].
- Baeldung. (2019). *Introduction to Creational Design Patterns* | Baeldung. [online] Available at: <https://www.baeldung.com/creational-design-patterns> [Accessed 16 Mar. 2019].
- Edureka. (2019). *Object Oriented Programming in Java | Java OOPs Concepts* | Edureka. [online] Available at: <https://www.edureka.co/blog/object-oriented-programming/> [Accessed 16 Mar. 2019].
- XenonStack. (2019). *Test Driven Development - TDD and Unit Testing in Java* - XenonStack. [online] Available at: <https://www.xenonstack.com/blog/test-driven-development-tdd-java/> [Accessed 17 Mar. 2019].