SCTR's

PUNE INSTITUTE OF COMPUTER TECHNOLOGY

SECOND YEAR
INFORMATION TECHNOLOGY
(2019 COURSE)

LABORATORY MANUAL
FOR

OBJECT ORIENTED ROGRAMMING LABORATORY

SEMESTER - III

[Subject code: 214448]

[Prepared By]

Mr. Sachin D. Shelke

Dr. Jayashree B. Jagdale

DEPARTMENT OF INFORMATION TECHNOLOGY

INSTITUTE VISION AND MISSION

VISION

Pune Institute of Computer Technology aspires to be the leader inhigher technical education and research of international repute.

MISSION

To be leading and most sought after Institute of education and research in emerging engineering and technology disciplines that attracts, retains and sustains gifted individuals of significant potential.

DEPARTMENT VISION AND MISSION

VISION

The department endeavors to be recognized globally as a center of academic excellence & research in Information Technology.

MISSION

To inculcate research culture among students by imparting information technology related fundamental knowledge, recent technological trends and ethics to get recognized as globally acceptable and socially responsible professionals.

Savitribai Phule Pune University, Pune
Second Year Information Technology (2019 Course)
214448: Object Oriented Programming Lab

Home

Teaching Scheme:	Credit Scheme:	Examination Scheme:
Practical (PR) : 04 hrs/week	02	PR: 25 Marks TW: 25 Marks

Prerequisites: Student should have knowledge of programming language.

Course Objectives:

1. Apply concepts of object-oriented paradigm.
2. Design and implement models for real life problems by using object-oriented programming.
3. Develop object-oriented programming skills.

Course Outcomes:

On completion of the course, students will be able to–

CO1: Differentiate various programming paradigms.
CO2: Identify classes, objects, methods, and handle object creation, initialization, and destruction to model real-world problems.
CO3: Identify relationship among objects using inheritance and polymorphism.
CO4: Handle different types of exceptions and perform generic programming.
CO5: Use file handling for real world application.
CO6: Apply appropriate design patterns to provide object-oriented solutions.

Guidelines for Instructor's Manual

The instructor's manual is to be developed as a hands-on resource and reference. The instructor's manual need to include prologue (about University/program/ institute/ department/foreword/ preface etc.), University syllabus, conduction & Assessment guidelines, topics under consideration concept, objectives, outcomes, set of typical applications/assignments/ guidelines, and references.

Guidelines for Student's Lab Journal

1. The laboratory assignments are to be submitted by student in the form of journal.
2. Journal consists of prologue, Certificate, table of contents, and handwritten write-up of each assignment (Title, Objectives, Problem Statement, Outcomes, software & Hardware requirements, Date of Completion, Assessment grade/marks and assessor's sign, Theory- OOP feature/Concept in brief, algorithm, flowchart, test cases, conclusion/analysis.
3. Program codes with sample output of all performed assignments are to be submitted as hardcopy.
4. As a conscious effort and little contribution towards Green IT and environment awareness, attaching printed papers as part of write-ups and program listing to journal may be avoided.
5. Use of DVD containing students programs maintained by lab In-charge is highly encouraged.
6. For reference one or two journals may be maintained with program prints at Laboratory.

Guidelines for Lab /TW Assessment

1. Continuous assessment of laboratory work is done based on overall performance and lab

assignments performance of student.
2. Each lab assignment assessment will assign grade/marks based on parameters with appropriate weightage.
3. Suggested parameters for overall assessment as well as each lab assignment assessment include- timely completion, performance, innovation, efficient codes, punctuality and neatness.

Guidelines for Practical Examination

Both internal and external examiners should jointly set problem statements. During practical assessment, the expert evaluator should give the maximum weightage to the satisfactory implementation of the problem statement. The supplementary and relevant questions may be asked at the time of evaluation to test the student's for advanced learning, understanding of the fundamentals, effective and efficient implementation. So encouraging efforts, transparent evaluation and fair approach of the evaluator will not create any uncertainty or doubt in the minds of the students.

Guidelines for Laboratory Conduction

The instructor is expected to frame the assignments by understanding the prerequisites, technological aspects, utility and recent trends related to the topic. The assignment framing policy need to address the average students and inclusive of an element to attract and promote the intelligent students. The instructor may set multiple sets of assignments without changing its complexity level and distribute among batches of students. Encourage students for the use of industry coding standards such as appropriate use of Hungarian notation, Indentation and comments. Use of open source software is encouraged. Set of suggested assignment list is provided, instructors may take different case studies with similar complexity level. Operating System recommended :- 64-bit Open source Linux or its derivative
Programming tools recommended: - JAVA IDE

List of Assignments

1.Classes and object -- CO1 and CO2

Design a class 'Complex 'with data members for real and imaginary part. Provide default and Parameterized constructors. Write a program to perform arithmetic operations of two complex numbers.

2. Polymorphism -- CO3

Identify commonalities and differences between Publication, Book and Magazine classes. Title, Price, Copies are common instance variables and saleCopy is common method. The differences are, Bookclass has author and orderCopies(). Magazine Class has methods orderQty, Current issue, receiveissue().Write a program to find how many copies of the given books are ordered and display total sale of publication.

3.Inheritance -- CO3

Design and develop inheritance for a given case study, identify objects and relationships and implement inheritance wherever applicable. Employee class hasEmp_name, Emp_id, Address,

Mail_id, and Mobile_noas members. Inherit the classes: Programmer, Team Lead, Assistant Project Manager and Project Manager from employee class. Add Basic Pay (BP) as the member of all the inherited classes with 97% of BP as DA, 10 % of BP as HRA, 12% of BP as PF, 0.1% of BP for staff club fund. Generate pay slips for the employees with their gross and net salary.

4.Dynamic Binding -- CO3

Design a base class shape with two double type values and member functions to input the data and compute_area() for calculating area of shape. Derive two classes: triangle and rectangle. Make compute_area() as abstract function and redefine this function in the derived class to suit their requirements. Write a program that accepts dimensions of triangle/rectangle and display calculated area. Implement dynamic binding for given case study.

5.Interface -- CO1, CO3

Design and develop a context for given case study and implement an interface for Vehicles Consider the example of vehicles like bicycle, car and bike. All Vehicles have common functionalities such as Gear Change, Speed up and apply breaks. Make an interface and put all these common functionalities. Bicycle, Bike, Car classes should be implemented for all these functionalities in their own class in their own way.

6.Exception handling -- CO4

Implement a program to handle Arithmetic exception, Array Index Out of Bounds. The user enters two numbers Num1 and Num2. The division of Num1 and Num2 is displayed. If Num1 and Num2 are not integers, the program would throw a Number Format Exception. If Num2 were zero, the program would throw an Arithmetic Exception. Display the exception.

7.Template -- CO4

Implement a generic program using any collection class to count the number of elements in a collection that have a specific property such as even numbers, odd number, prime number and palindromes.

8.File Handling -- CO5

Implement a program for maintaining a database of student records using Files.
Student has Student_id,name, Roll_no, Class, marks and address. Display the data for few students.

1. Create Database
2. Display Database
3. Delete Records
4. Update Record
5. Search Record

9.Case Study -- CO2, CO5

Using concepts of Object-Oriented programming develop solution for any one application
1) Banking system having following operations :

1. Create an account 2. Deposit money 3. Withdraw money 4. Honor daily withdrawal limit
5. Check the balance 6. Display Account information.

2) Inventory management system having following operations :
1. List of all products 2. Display individual product information 3. Purchase 4. Shipping
5. Balance stock6. Loss and Profit calculation.

10. Factory Design Pattern -- CO6

Implement Factory design pattern for the given context. Consider Car building process, which requires many steps from allocating accessories to final makeup. These steps should be written as methods and should be called while creating an instance of a specific car type. Hatchback, Sedan, SUV could be the subclasses of Car class. Car class and its subclasses, CarFactory and Test Factory Pattern should be implemented.

11. Strategy Design Pattern -- CO6

Implement and apply Strategy Design pattern for simple Shopping Cart where three payment strategies are used such as Credit Card, PayPal, Bit Coin. Create an interface for strategy pattern and give concrete implementation for payment.

Text Books:

1. E. Balagurusamy, "Programming with Java – A Primer", Tata – McGraw-Hill Publication, 4th Edition, 2019
2. Kathy Sierra, "OCA /OCP Java SE 7 Programmer I & II Study Guide"(Exams 1Z0-803 & IZ-804) Oracle Press (2017)
3. Steven Holzner et al. "Java 2 Programming", Black Book, Dreamtech Press, 2009

Reference Books:

1. H.M. Deitel, P.J. Deitel, "Java - How to Program", PHI Publication, 6th Edition, 2005
2. Bruce Eckel, "Thinking in Java", PHI Publication
3. Poo, Danny, Kiong, Derek, Ashok, Swarnalatha," Object-Oriented Programming and Java", ISBN 978-1-84628-963-7
4. Erich Gamma,Richard Helm ,Ralph Johnson,JohnVlissides, "Design Patterns ,Elements of Reusable Object- Oriented Software" ISBN-13: 978-0201633610
5. RohitJoshi, "Java Design patterns, Reusable solutions to common problems" Java Code Geeks

ASSIGNMENT: 1

Title : Classes and Objects

Problem Statement: Design a class 'Complex 'with data members for real and imaginary part. Provide default and Parameterized constructors. Write a program to perform arithmetic operations of two complex numbers.

Objectives: To learn the concept of class, object and constructor

Theory:

1. What is a class in Java?

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.
A class in Java can contain:

- ✓ Fields
- ✓ Methods
- ✓ Constructors
- ✓ Blocks
- ✓ Nested class and interface

Syntax to declare a class:

```
class <class_name>
{
  field;
 method;
}
```

2. Instance variable in Java

- ➢ A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.
- ➢ Method in Java
- ➢ In Java, a method is like a function which is used to expose the behaviour of an object.
- ➢ new keyword in Java
- ➢ The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

```
classname object =new classname();
```

3. What is an object in Java ?

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible).

An object has three characteristics:

State: represents the data (value) of an object.

Behavior: represents the behaviour (functionality) of an object such as deposit, withdraw, etc.

Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behaviour.

An object is an instance of a class. A class is a template or blueprint from which objects are created.

Object Definitions:

An object is a real-world entity.
An object is a runtime entity.
The object is an entity which has state and behavior.
The object is an instance of a class.

Object and Class Example: main within the class

In this example, create a Student class which has two data members id and name. Create the object of the Student class by new keyword and print the object's value.

Here, create a main() method inside the class.

File: Student.java

```
//Java Program to illustrate how to define a class and fields
//Defining a Student class.
        class Student
        {
 //defining fields
         int id;//field or data member or instance variable
         String name;
 //creating main method inside the Student class
         public static void main (String args[])
                 {
  //Creating an object or instance
         Student s1=new Student();//creating an object of Student
  //Printing values of the object
         System.out.println(s1.id);//accessing member through reference variable
        System.out.println(s1.name);
                 }
        }
```

Object and Class Example: main outside the class

In real time development, create classes and use it from another class. It is a better approach than previous one. Let's see a simple example, where main () method in defined in another class.

File: TestStudent1.java

```
//Java Program to demonstrate having the main method inanother class
//Creating Student class.
        class Student
        {
                int id;
         String name;
        }
//Creating another class TestStudent1 which contains the main method
        class TestStudent1
        {
                public static void main(String args[])
                {
                        Student s1=new Student();
                        System.out.println(s1.id);
                        System.out.println(s1.name);
                }
        }
```

4. Constructors in Java

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

•Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

•Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor
A constructor is called "Default Constructor" when it doesn't have any parameter.
Syntax of default constructor:
<class_name> ()
{
}
Example of default constructor
In this example, creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
//Java Program to create and call a default constructor
        class Bike1
        {
//creating a default constructor
                Bike1()
                {
                System.out.println("Bike is created");
                }
//main method
                public static void main(String args[])
                {
//calling a default constructor
                Bike1 b=new Bike1();
                }
        }
```

Java Parameterized Constructor
A constructor which has a specific number of parameters is called a parameterized constructor. The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.
Syntax of parameterized constructor:
class <class_name>
{
int x;
float y;
<class_name> (int m, float n)
{
x=m;
y=n;
}
Example of parameterized constructor
In this example, create the constructor of Student class that have two parameters. Initialize this parameter value using parameterised constructor.

```
//Java Program to demonstrate the use of the parameterized constructor.
```

```
class Student4
{
   int id;
   String name;
      Student4(int i,String n)//creating a parameterized constructor
          {
                    id = i;
                    name = n;
           }
   void display()
          {
                    System.out.println(id+" "+name);
          }

   public static void main(String args[])
          {
           Student4 s1 = new Student4(111,"Karan"); //creating objects and passing values
          Student4 s2 = new Student4(222,"Aryan");
      s1.display(); //calling method to display the values of object
   s2. display();
          }
}
```

Example:

Create a class called Complex for performing arithmetic on complex numbers.
Complex numbers have the form a+bi where a is real part and b is imaginary part and i=√-1.
Use floating point variables to represent the private data of the class. Provide constructor that enable an object to be initialized when it is declared. Provide no argument constructor with default values in case no initializers are provided. Provide public methods for addition, subtraction, multiplication and division of complex numbers. Pass objects of Complex as parameters of the method.

Algorithm:

1. Begin
2. Define a class operation with instance variables real and imag
3. Input the two complex numbers c1=(a+ib) and c2=(c+id)
4. Define the method add (c1, c2) as (a+ib) + (c+id) and stores result in c3
5. Define the method sub (c1, c2) as (a+ib) - (c+id) and stores result in c3
6. Define the method mul (c1, c2) as (a+ib) * (c+id) and store the result in c3 as (ac-bd) + i(bc+ad)
7. Define the method div (c1, c2) as (a+ib)/(c+id) and stores the quotient c3 as {(ac+bd)/(c2+d2)} +i{(bc-ad)/(c2+d2)}
8. Define the method display () which outputs each result
9. End

Input and Output Requirements:

Program reads real and imaginary parts of two complex numbers through keyboard and displays their sum, difference, product and quotient as result.

Conclusion:

Hence, we have studied concept of class, object and constructor.

Questions:

1. Describe benefits of OOP?
2. Define constructor?
3. Sketch pictorial representation of object from your program.
4. Compare class variables and static variables
5. Explain types of constructor with example.
6. Discuss use of constructor
7. Differentiate constructor and destructor.
8. Define class & object.
9. What is return type of constructor.

Assignment: 2

Title: Polymorphism

Problem Statement: Identify commonalities and differences between Publication, Book and Magazine classes. Title, Price, Copies are common instance variables and saleCopy is common method. The differences are, Book class has author and orderCopies(). Magazine Class has orderQty, Currentissue, reciveissue(). Write a program to find how many copies of the given books are ordered and display total sale of publication.

Objective: To learn the concept of polymorphism

Theory:

This section explains the Object Oriented Concept, Polymorphism. Polymorphism is the ability of an entity to behave in different forms. Take a real-world example; the Army Ants. There are different size of ants in the same ant colony with different responsibilities; workers are in different sizes and the queen is the largest one. This is a polymorphic behavior where the entities have a unique feature while they share all other common attributes and behaviors.

Polymorphism is considered as one of the important features of Object Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word "poly" means many and "morphs" means forms, So it means many forms.

There are two types of polymorphism in java:

1) Static Polymorphism also known as compile time polymorphism
2) Dynamic Polymorphism also known as runtime polymorphism

Compile time Polymorphism or Static polymorphism:

Polymorphism that is resolved during compiler time is known as static polymorphism. Method overloading is an example of compile time polymorphism.

Method Overloading: This allows us to have more than one method having the same name, if the parameters of methods are different in number, sequence and data types of parameters.

Example :Method overloading is one of the way java supports static polymorphism. Here we have two definitions of the same method add() which add method would be called is determined by the parameter list at the compile time. That is the reason this is also known as compile time polymorphism.

```
classSimpleCalculator
{
int add(int a,int b)
```

```
{
returna+b;
}
int add(int a,int b,int c)
{
returna+b+c;
}
}
publicclassDemo
{
publicstaticvoid main(Stringargs[])
{
SimpleCalculatorobj=newSimpleCalculator();
System.out.println(obj.add(10,20));
System.out.println(obj.add(10,20,30));
}
}
```

Runtime Polymorphism or Dynamic polymorphism

It is also known as Dynamic Method Dispatch. Method overriding is an example of runtime polymorphism. Dynamic polymorphism is a process in which a call to an overridden method is resolved at runtime, thats why it is called runtime polymorphism.

Method Overriding: Declaring a method in sub class which is already present in parent class is known as method overriding. Overriding is done so that a child class can give its own implementation to a method which is already provided by the parent class. In this case the method in parent class is called overridden method and the method in child class is called overriding method.

Method Overriding Example: We have two classes: A child class Boy and a parent class Human. The Boy class extends Human class. Both the classes have a common method void eat (). Boy class is giving its own implementation to the eat () method or in other words it is overriding the eat () method.

The purpose of Method Overriding is clear here. Child class wants to give its own implementation so that when it calls this method, it prints Boy is eating instead of Human is eating.

```
classHuman{
//Overridden method
publicvoid eat()
{
System.out.println("Human is eating");
}
}
classBoyextendsHuman{
//Overriding method
publicvoid eat(){
```

```
System.out.println("Boy is eating");
}
publicstaticvoid main(Stringargs[]){
Boyobj=newBoy();
//This will call the child class version of eat()
obj.eat();
}
}
```

Output:

Boy is eating

Advantage of method overriding: The main advantage of method overriding is that the class can give its own specific implementation to a inherited method without even modifying the parent class code. This is helpful when a class has several child classes, so if a child class needs to use the parent class method, it can use it and the other classes that want to have different implementation can use overriding feature to make changes without touching the parent class code.

Method Overriding is an example of runtime polymorphism. When a parent class reference points to the child class object then the call to the overridden method is determined at runtime, because during method call which method (parent class or child class) is to be executed is determined by the type of object. This process in which call to the overridden method is

```
class ABC{
//Overridden method
publicvoiddisp()
{
        System.out.println("disp() method of parent class");
}
}
classDemoextends ABC{
//Overriding method
publicvoiddisp(){
        System.out.println("disp() method of Child class");
}
publicvoidnewMethod(){
        System.out.println("new method of child class");
}
publicstaticvoid main(Stringargs[]){
        /* When Parent class reference refers to the parent class object
         * then in this case overridden method (the method of parent class)
        *  is called.
         */
        ABC obj=newABC();
```

```
        obj.disp();

        /* When parent class reference refers to the child class object
         * then the overriding method (method of child class) is called.
         * This is called dynamic method dispatch and runtime polymorphism
         */
        ABC obj2 =newDemo();
        obj2.disp();
}
}
```

Output:

disp() method of parent class

disp() method of Childclass

In the above example the call to the disp() method using second object (obj2) is runtime polymorphism In dynamic method dispatch the object can call the overriding methods of child class and all the non-overridden methods of base class but it cannot call the methods which are newly declared in the child class. In the above example the object `obj2` is calling the `disp()`. However if you try to call the `newMethod()` method (which has been newly declared in Demo class) using obj2 then you would give compilation error.

Rules of method overriding in Java

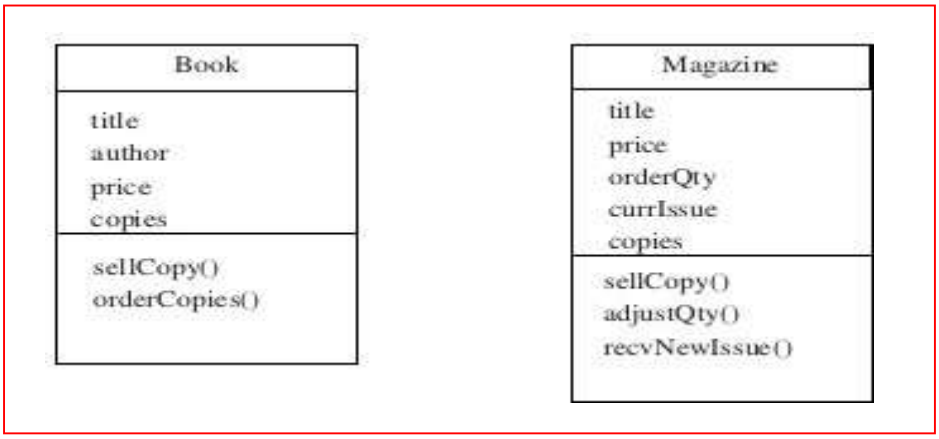
Rule #1:
Overriding method name and the overridden method name must be exactly same.
Rule #2:
Overriding method must have the same set of parameters as the overridden method.
Rule #3:
The return type of overriding method name must be same as the super class's method.
Rule #4:
Access modifier of the overriding method must be same or less restrictive than the overridden method's access modifier.
Rule #5:
The overriding method can throw new unchecked exceptions but cannot throw new checked exceptions.

Method Overriding with example (Run time Polymorphism):

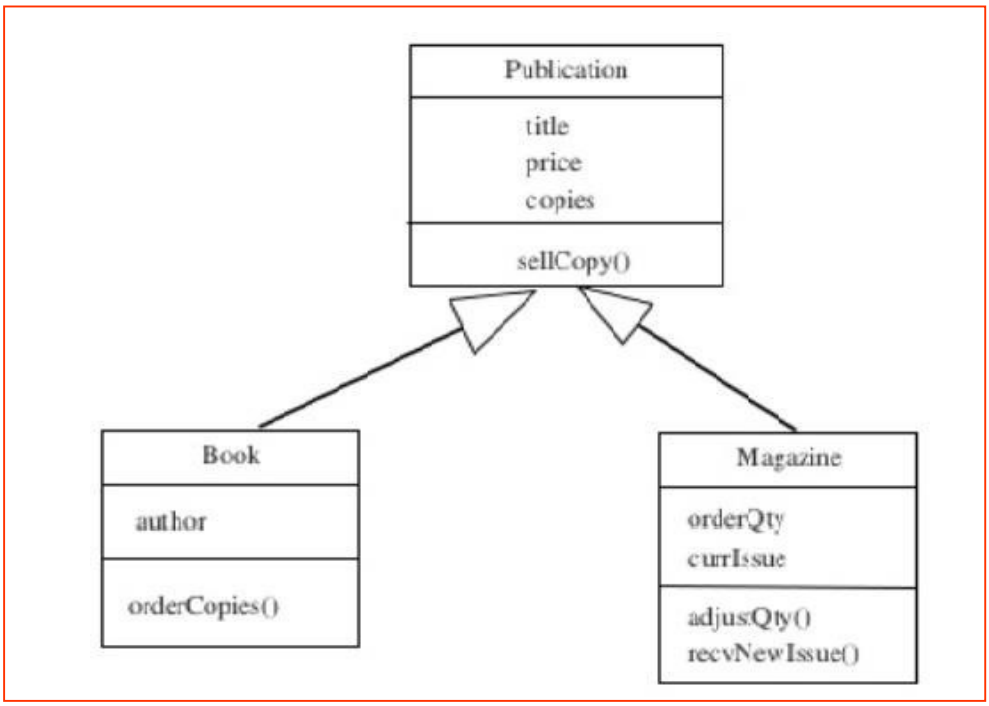
Advantages of method overriding:

Sample Code:

Consider Book & Magazines both specific type of publication



Attribute title, author & price are obvious parameter. For Book, orderCopies() takes parameter specifying how many copies are added to stock.For Magazine, orderQty is number of copies received of each new issue and currIssue is date/period of current issue.We can separate out these common member of classes into superclass called Publication.The differences will need to be specified as additional member for the 'subclasses' Book and Magazine.



public class Publication {

```
    // Title of the publication.
    private String title;
    // Price of the publication.
    private double price;
    // copies of the publication.
privateint copies;
    public String getTitle() {
        returnthis.title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public void setPrice(double price) {
        if (price > 0) {
            this.price = price;
        } else {
            System.out.println("Invalid price");
        }
    }
    public double getPrice() {
        returnthis.price;
    }
publicintgetcopies() {
        returnthis.copies;
    }
    public void setcopies(int copies) {
        this.copies = copies;
    }

    public void sellcopy(intqty) {
        System.out.println("Total Publication sell: $" + (qty * price));
    }
}
public class Book extends Publication {
    // Author of the book.
    private String author;
    public String getAuthor() {
        returnthis.author;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
    Public void ordercopies(intpcopies){
Setcopies(getcopies() + pcopies);
}
    public void sellcopy(intqty) {
        System.out.println("Total Book sell: $" + (qty * price));
    }
}
public class Magazine extends Publication {
    privateintorderQty;
private String currIssue;
    public String getcurrIssue() {
```

```
      returnthis.currIssue;
   }
   public void setcurrIssue(String issue) {
      this.currIssue = issue;
   }
publicintgetorderQty() {
      returnthis.orderQty;
   }
   public void setorderQty(int copies) {
      this.orderQty = copies;
   }
public void sellcopy(intqty) {
      System.out.println("Total Magazine sell: $" + (qty * price));
   }
  Public void recvNewIssue(string pNewIssue){
Setcopies(orderQty);
currIssue=pNewIssue;
}
}
Class mainClass{
Public static void main(String [] args)
{
//accept all details of book to be order such as title, author, price & copies;
Book obj1 = new Book();
Obj1.ordercopies(copies);
Publication obj2 = new Book();
Obj2.sellcopy(copies); //Overriden method is invoke
Publication obj3 = new Publication();
Obj3.sellcopy(copies);
}
```

Questions:

1. Define is polymorphism.
2. List out polymorphism types.
3. Differentiate static & runtime polymorphism.
4. Define method overloading.
5. Define method overriding.
6. Differentiate method overloading and method overriding.
7. Explain superclass & subclass.

Assignment:3

Title: Inheritance

Problem Statement: Design and develop inheritance for a given case study, identify objects and relationships and implement inheritance wherever applicable. Employee class with Emp_name, Emp_id, Address, Mail_id, and Mobile_no as members. Inherit the classes, Programmer, Team Lead, Assistant Project Manager and Project Manager from employee class. Add Basic Pay (BP) as the member of all the inherited classes with 97% of BP as DA, 10 % of BP as HRA, 12% of BP as PF, 0.1% of BP for staff club fund. Generate pay slips for the employees with their gross and net salary**.**

Objectives:
1) To Study Inheritance and its types
2) To implement inheritance using OOP language

Theory:-

Inheritance:
Different kinds of objects often have a certain amount in common with each other. Mountain bikes, road bikes, and tandem bikes, for example, all share the characteristics of bicycles (current speed, current pedal cadence, current gear). Yet each also defines additional features that make them different: tandem bicycles have two seats and two sets of handlebars; road bikes have drop handlebars; some mountain bikes have an additional chain ring, giving them a lower gear ratio. Object-oriented programming allows classes to inherit commonly used state and behavior from other classes. In this example, Bicycle now becomes the superclass of MountainBike, RoadBike, and TandemBike. In the Java programming language, each class is allowed to have one direct superclass, and each superclass has the potential for an unlimited number of subclasses:

The syntax for creating a subclass is simple. At the beginning of your class declaration, use

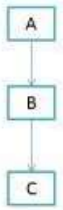
the `extends` keyword, followed by the name of the class to inherit from:

```
class MountainBike extends Bicycle {
```

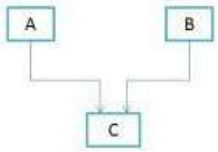
```
    // new fields and methods defining
    // a mountain bike would go here

}
```

This gives MountainBike all the same fields and methods as Bicycle, yet allows its code to focus exclusively on the features that make it unique. This makes code for your subclasses easy to read. However, you must take care to properly document the state and behavior that each superclass defines, since that code will not appear in the source file of each subclass.

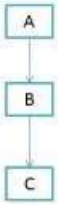
- **Single Inheritance:** When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a parent class of B and B would be a child class of A.



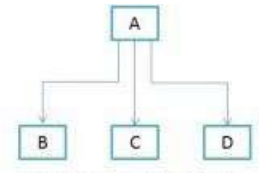
- Multiple Inheritance: It refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with "multiple inheritance" is that the derived class will have to manage the dependency on two base classes.



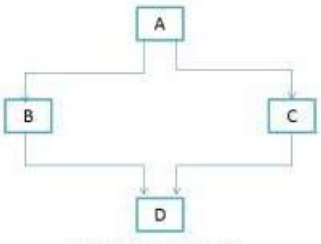
- Multilevel Inheritance: Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A.



- Hierarchical Inheritance: In such kind of inheritance one class is inherited by many sub classes. In below example class B,C and D inherits the same class A. A is parent class (or base class) of B,C & D.



- Hybrid Inheritance :In simple terms you can say that Hybrid inheritance is a combination of Single and Multiple inheritance. A typical flow diagram would look like below. A hybrid inheritance can be achieved in the java in a same way as multiple inheritance can be!! Using interfaces. yes you heard it right. By using interfaces you can have multiple as well as hybrid inheritance in Java.



Steps :

1. Start
2. Create the class Employee with name, Empid, address, mailid, mobileno as data members.
3. Inherit the classes Programmer, Team Lead, Assistant Project Manager and Project Manager from employee class.
4. Add Basic Pay (BP) as the member of all the inherited classes.
5. Calculate DA as 97% of BP, HRA as 10% of BP, PF as 12% of BP, Staff club fund as 0.1% of BP.
6. Calculate gross salary and net salary.
7. Generate payslip for all categories of employees.
8. Create the objects for the inherited classes and invoke the necessary methods to display the Payslip
9. Stop

Input:
Empid, address, mailid, mobileno, Basic Pay (BP)

Output:
gross and net salary slip

Implementation: -

```
class Employee {
int empid;
long mobile;
 String name, address, mailid;
void getdata() { ----}
void display() { -----}
}
class Programmer extends Employee {
double salary,bp,da,hra,pf,club,net,gross;
void getasst() {--- }
void calculateasst() { -- }
}
class TeamLead extends Employee {
---
}
class AssistantProjectManager extends Employee {
---
}
class Project Manager extends Employee {
---
}
class Salary {
public static void main(String args[]) { --- }
}
```

Frequently asked Question

1. Justify inheritance/ class relationship with classes Surgeon and Doctor.
2. Explain why multiple inheritance is not supported by Java.
3. Compare Composition and Inheritance in OOP.
4. What are different types of Inheritance supported by Java?
5. Why multiple inheritance is not supported by Java?
6. How to use Inheritance in Java?
7. What is the difference between Composition and Inheritance in OOP?
8. What is Super Keyword in Java?

Assignment: 4

Title: Dynamic Binding

Problem Statement:- Design a base class shape with two double type values and member functions to input the data and compute_area() for calculating area of figure. Derive two classes' triangle and rectangle. Make compute_area() as abstract function and redefine this function in the derived class to suit their requirements. Write a program that accepts dimensions of triangle/rectangle and display calculated area. Implement dynamic binding for given case study.

Aim :- To Study Polymorphism using Java

Theory:-

Polymorphism in Java

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

Real life example of polymorphism: A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So the same person posses different behaviour in different situations. This is called polymorphism.
Polymorphism is considered as one of the important features of Object Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word "poly" means many and "morphs" means forms, So it means many forms.

In Java polymorphism is mainly divided into two types:

- Compile time Polymorphism
- Runtime Polymorphism

1. Compile time polymorphism: It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading.

Method Overloading: When there are multiple functions with same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by change in number of arguments or/and change in type of arguments.

Example: By using different types of arguments

```
// Java program for Method overloading

class MultiplyFun {

   // Method with 2 parameter
    static int Multiply(int a, int b)
   {
       return a * b;
   }

   // Method with the same name but 2 double parameter
    static double Multiply(double a, double b)
    {
        return a * b;
    }
}

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

           System.out.println(MultiplyFun.Multiply(2, 4));

           System.out.println(MultiplyFun.Multiply(5.5, 6.3));
       }
}
```

Output:
8
34.65

Operator Overloading: Java also provide option to overload operators. For example, we can make the operator ('+') for string class to concatenate two strings. We know that this is the addition operator whose task is to add two operands. So a single operator '+' when placed between integer operands, adds them and when placed between string operands, concatenates them.
In java, Only "+" operator can be overloaded:

To add integers
To concatenate strings

Example:

```
/ Java program for Operator overloading

class OperatorOVERDDN {

    void operator(String str1, String str2)
    {
        String s = str1 + str2;
        System.out.println("Concatinated String - "
                           + s);
```

```
    }

    void operator(int a, int b)
    {
        int c = a + b;
        System.out.println("Sum = " +c);
    }
}

class Main {
    public static void main(String[] args)
    {
        OperatorOVERDDN obj = new OperatorOVERDDN();
        obj.operator(2, 3);
        obj.operator("joe", "now");
    }
}
```

Output:
Sum =5
Concatinated String -joenow

Runtime polymorphism: It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

Method overriding, on the other hand, occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.

Objectives :- 1) To understand the concept of Polymorphism using Java
2) To implement run time polymorphism

Steps :
1. Start
2. Create an abstract class named shape that contains two double type numbers and an empty method named compute_area().
3. Provide two classes named rectangle and triangle such that each one of the classes extends the class Shape.
4. Each of the inherited class from shape class should provide the implementation for the method compute_area().
5. Get the input and calculate the area of rectangle and triangle.
6. In the fourth separate class, create the objects for the two inherited classes and invoke the methods and display the area values of the different shapes.
7. Stop.

Input:
length and breadth of rectangle
base and height of triangle

Output:
area of rectangle
area of circle

Implementation :-

```
abstract class shape {
 abstract public void compute_area();
}
class rectangle extends shape {
public void compute_area() {
---
}
}
class triangle extends shape {
public void compute_area() {
---
}
}

public class Shapeclass {
public static void main(String[] args) {
---
}
```

Test case or Validation:
Different values for length and breadth of rectangle and base and height of triangle.

Result:- Area of Circle and area of Rectangle

Conclusion :- Thus studied the concept of Polymorphism using java.

Frequently asked Question
1. What is polymorphism and what are the types of it?
2. What is method overriding?
3. What is method overloading?
4. Difference between method overloading and overriding?
5. What is static and dynamic binding?

Assignment: 5

Title: Interface

Problem Statement: Design and develop a context for given case study and implement an interface for Vehicles Consider the example of vehicles like bicycle, car, and bike. All Vehicles have common functionalities such as Gear Change, Speed up and apply breaks . Make an interface and put all these common functionalities. Bicycle, Bike, Car classes should be implemented for all these functionalities in their own class in their own way.

Aim :
To understand Interface in Java

Theory
An interface in Java is a blueprint of a class. It has static constants and abstract methods.The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.A programmer uses an abstract class when there are some common features shared by all the objects .A programmer writes an interface when all the features have different implementations for different objects Interfaces are written when the programmer wants to leave the implementation to third party vendors An interface is a specification of method prototypes.

All the methods in an interface are abstract methods

1. An interface is a specification of method prototypes
2. An interface contains zero or more abstract methods
3. All the methods of interface are public, abstract by default
4. An interface may contain variables which are by default public static final
5. Once an interface is written any third party vendor can implement it
6. All the methods of the interface should be implemented in its implementation classes
7. If any one of the method is not implemented, then that implementation class should be declared as abstract
8. We cannot create an object to an interface
9. We can create a reference variable to an interface
10. An interface cannot implement another interface
11. An interface can extend another interface
12. A class can implement multiple interfaces

Syntax:

```
interface <interface_name>{

  // declare constant fields
```

```
   // declare methods that abstract
   // by default.
}
```

1. Java Interface Example

In this example, the Printable interface has only one method, and its implementation is provided in the A6 class.

```
interface printable{
void print();
}

class A6 implements printable{
public void print()
{
System.out.println("Hello");
}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

2. Write an example program for interface

```
Interface  Shape
{
      void area ();
      void volume ();
      double pi = 314;
 }

 class Circle implements Shape
{
         double r;
         Circle (double radius)
          {r = radius; }
         public void area ()
         {
System.out.println ("Area of a circle is : " + pi*r*r );
          }
         public void volume ()
```

```
        {
System.out.println ("Volume of a circle is : " + 2*pi*r);
        }
 }

class Rectangle implements Shape
{
       double l,b;
       Rectangle (double length, double breadth)
        {
            l = length; b = breadth;.
         }
       public void area ()
        {
System.out.println ("Area of a Rectangle is : " + l*b );
         }

        public void volume ()
        {
System.out.println ("Volume of a Rectangle is : " + 2*(l+b));
        }
}

class InterfaceDemo
{
        public static void main (String args[])
        {
             Circle ob1 = new Circle (102);
             ob1.area ();
             ob1.volume ();
             Rectangle ob2 = new Rectangle (126, 2355);
             ob2.area ();
             ob2.volume ();
          }
 }
```

Interface vs Abstract Class

An interface is like having a 100% Abstract Class. Interfaces cannot have non-abstract Methods while abstract Classes can. A Class can implement more than one Interface while it can extend only one Class. As abstract Classes come in the hierarchy of Classes, they can extend other Classes while Interface can only extend Interfaces.

The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a class implements an interface.

Multiple inheritance in Java by interface
If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.

Result:
Bicycle state:
speed: 2 gear: 2
Bike state:
speed: 1 gear: 1

Conclusion:

Thus, we have studies interface concept using java

Frequently asked Questions

1. Write a java interface which provides the implementation of Bank interface to calculate Rate of Interest.
2. Write a Drawable interface has only one methoddraw(). Its implementation is provided by Rectangle and Circle classes
3. A class implements an interface, but one interface extends another interface.
4. Create a vehicles as interface mention all common functionalities and create classes like bicycle, car, bike implement all these functionalities in their own class in their own way.
5. Create a animal class using interface and provide some common functionalities and implement into some other classes.
6. Write a java interface which provides the implementation of Bank interface to calculate Rate of Interest.

Assignment: 6

Title: Exception handling

Problem Statement: Implement a program to handle Arithmetic exception, Array Index Out Of Bounds. The user enters two numbers Num1 and Num2. The division of Num1 and Num2 is displayed. If Num1 and Num2 were not integers, the program would throw a Number Format Exception. If Num2 were zero, the program would throw an Arithmetic Exception. Display the exception.

Aim :
Exception handling

Theory

Exception handling in java with examples
Exception handling is one of the most important feature of java programming that allows us to handle the runtime errors caused by exceptions. In this guide, we will learn what is an exception, types of it, exception classes and how to handle exceptions in java with examples.

What is an exception?
An Exception is an unwanted event that interrupts the normal flow of the program. When an exception occurs program execution gets terminated. In such cases we get a system generated error message. The good thing about exceptions is that they can be handled in Java. By handling the exceptions we can provide a meaningful message to the user about the issue rather than a system generated message, which may not be understandable to a user.

Why an exception occurs?
There can be several reasons that can cause a program to throw exception. For example: Opening a non-existing file in your program, Network connection problem, bad input data provided by user etc.

Exception Handling
If an exception occurs, which has not been handled by programmer then program execution gets terminated and a system generated error message is shown to the user. For example look at the system generated exception below:

An exception generated by the system is given below

```
Exceptionin thread "main"java.lang.ArithmeticException:
/by zero at ExceptionDemo.main(ExceptionDemo.java:5)
ExceptionDemo:Theclass name
main :The method name
ExceptionDemo.java :The filename
java:5:Line number
```

This message is not user friendly so a user will not be able to understand what went wrong. In order to let them know the reason in simple language, we handle exceptions. We handle such

conditions and then prints a user-friendly warning message to user, which lets them correct the error as most of the time exception occurs due to bad data provided by user.
Advantage of exception handling

Exception handling ensures that the flow of the program doesn't break when an exception occurs. For example, if a program has bunch of statements and an exception occurs mid way after executing certain statements then the statements after the exception will not execute and the program will terminate abruptly.

By handling we make sure that all the statements execute and the flow of program doesn't break. Difference between error and exception.Errors indicate that something severe enough has gone wrong, the application should crash rather than try to handle the error.

Exceptions are events that occurs in the code. A programmer can handle such conditions and take necessary corrective actions.
Few examples:
NullPointerException – When you try to use a reference that points to null.
ArithmeticException – When bad data is provided by user, for example, when you try to divide a number by zero this exception occurs because dividing a number by zero is undefined.
ArrayIndexOutOfBoundsException – When you try to access the elements of an array out of its bounds, for example array size is 5 (which means it has five elements) and you are trying to access the 10th element.

Types of exceptions

There are two types of exceptions in Java:
1)Checked exceptions
2)Unchecked exceptions

Checked exceptions
All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation to see whether the programmer has handled them or not. If these exceptions are not handled/declared in the program, you will get compilation error. For example, SQLException, IOException, ClassNotFoundException etc.

Unchecked Exceptions
Runtime Exceptions are also known as Unchecked Exceptions. These exceptions are not checked at compile-time so compiler does not check whether the programmer has handled them or not but it's the responsibility of the programmer to handle these exceptions and provide a safe exit.
For example,ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

Compiler will never force you to catch such exception or force you to declare it in the method using throws keyword.
Try block

The try block contains set of statements where an exception can occur. A try block is always followed by a catch block, which handles the exception that occurs in associated try block. A try block must be followed by catch blocks or finally block or both.

Syntax of try block
try{
//statements that may cause an exception
}

While writing a program, if you think that certain statements in a program can throw a exception, enclosed them in try block and handle that exception

Catch block
A catch block is where you handle the exceptions, this block must follow the try block. A single try block can have several catch blocks associated with it. You can catch different exceptions in different catch blocks. When an exception occurs in try block, the corresponding catch block that handles that particular exception executes. For example if an arithmetic exception occurs in try block then the statements enclosed in catch block for arithmetic exception executes.

Syntax of try catch in java

```
try
{
//statements that may cause an exception
}
catch(exception(type) e(object))
{
//error handling code
}
```

Example: try catch block

If an exception occurs in try block then the control of execution is passed to the corresponding catch block. A single try block can have multiple catch blocks associated with it, you should place the catch blocks in such a way that the generic exception handler catch block is at the last (see in the example below).
The generic exception handler can handle all the exceptions but you should place is at the end, if you place it at the before all the catch blocks then it will display the generic message.You always want to give the user a meaningful message for each type of exception rather then a generic message.

```
classExamp

classExample1{
publicstaticvoid main(Stringargs[]){
int num1, num2;
try{
```

```
      /* We suspect that this block of statement can throw
       * exception so we handled it by placing these statements
       * inside try and handled the exception in catch block
       */
      num1 =0;
      num2 =62/ num1;
System.out.println(num2);
System.out.println("Hey I'm at the end of try block");
}
    catch(ArithmeticException e){
/* This block will only execute if any Arithmetic exception
       * occurs in try block
       */
System.out.println("You should not divide a number by zero");
}
catch(Exception e){
/* This is a generic Exception handler which means it can handle
       * all the exceptions. This will execute if the exception is not
       * handled by previous catch blocks.
       */
System.out.println("Exception occurred");
}
System.out.println("I'm out of try-catch block in Java.");
}
}
```

Output:

You should not divide a number by zero
I'm out of try-catch block in Java.

Multiple catch blocks in Java

The example we seen above is having multiple catch blocks, lets see few rules about multiple catch blocks with the help of examples. To read this in detail, see catching multiple exceptions in java.

1. As mentioned above, a single try block can have any number of catch blocks.
2. A generic catch block can handle all the exceptions. Whether it is ArrayIndexOutOfBoundsException or ArithmeticException or NullPointerException or any other type of exception, this handles all of them. To see the examples of NullPointerException and ArrayIndexOutOfBoundsException, refer this article: Exception Handling example programs.

```
catch(Exception e)
{
  //This catch block catches all the exceptions
}
```

If you are wondering why we need other catch handlers when we have a generic that can handle all. This is because in generic exception handler you can display a message but you are not sure for which type of exception it may trigger so it will display the same message for all the exceptions

and user may not be able to understand which exception occurred. Thats the reason you should place is at the end of all the specific exception catch blocks.
3. If no exception occurs in try block then the catch blocks are completely ignored.
4. Corresponding catch blocks execute for that specific type of exception:
catch(ArithmeticException e) is a catch block that can hanldeArithmeticException
catch(NullPointerException e) is a catch block that can handle NullPointerException
5. You can also throw exception, which is an advanced topic and I have covered it in separate tutorials: user defined exception, throws keyword, throw vs throws.

Example of Multiple catch blocks

```
classExample2{
publicstaticvoid main(Stringargs[]){
try{
int a[]=newint[7];
      a[4]=30/0;
System.out.println("First print statement in try block");
}
catch(ArithmeticException e){
System.out.println("Warning: ArithmeticException");
}
catch(ArrayIndexOutOfBoundsException e){
System.out.println("Warning: ArrayIndexOutOfBoundsException");
}
catch(Exception e){
System.out.println("Warning: Some Other exception");
}
System.out.println("Out of try-catch block...");
}
}
```

Output:
Warning: ArithmeticException
Out of try-catch block...

In the above example there are multiple catch blocks and these catch blocks executes sequentially when an exception occurs in try block. Which means if you put the last catch block (catch(Exception e)) at the first place, just after try block then in case of any exception this block will execute as it can handle all exceptions. This catch block should be placed at the last to avoid such situations.

Finally block
You should place those statements in finally blocks, that must execute whether exception occurs or not.

Input: Values of Text field T1, T2
Output: Displays the result in Text field T3

Step1: Start.
Step2: Import java.awt package
Step3: Import java.lang.string,awt. event,applet.Applet packages.

Step4: Create Class
Step5: Create Buttons and Text Fields.
Step6: Create the Data.
Step7: Perform the division.
Step8: Print the Data.
Step9: Stop.

Conclusion
Thus we have studies exception handling concept using java

Assignments
1. Write a java interface which provides the implementation of Bank interface to calculate Rate of Interest.
2.The Drawable interface has only one method draw(). Its implementation is provided by Rectangle and Circle classes.

Frequently asked Questions
1. Design one login page and ask user to enter user id and password. If one of the field is empty generate null pointer exception.
2. Write a method to process only text file , so we can provide caller with appropriate error code when some other type of file is sent as input.
3. Write a bank class if user will provide account number which is greater than specified size then method will produce array out of bound exception.

Assignment :7

Title: Template

Problem Statement: Implement a generic program using any collection class to count the number of elements in a collection that have a specific property such as even numbers, odd number, prime number and palindromes.

Objectives: To learn the concept templates and generic programming

Theory:

1. Java Generic methods
   - Syntax to declare class
   - Instance variable in Java
   - Method in Java
   - 'new' keyword in Java
2. Generic classes
   - multiple Type parameters

3. Advantages of Generics:

Generic Types

- ✓ Generic type represents classes, interfaces and methods in a type safe manner
- ✓ Generic types can act on any type of data
- ✓ All Generic types are subclasses of Object class, it acts on Objects only
- ✓ Generic types act on advanced data type only
- ✓ It is not possible to create an object to Generic type itself
- ✓ Using generic types, we can avoid casting in many cases Generic Class:

When we create a class with an instance variable to store an Integer object, it can be used to store Integer type data only
We cannot use that instance variable to store a Float class object or a String type Object To store different types of data into a class, we have to write the same class again and again by changing the data type of the variables This can be avoided using a generic class A generic class represents a class that is type-safe This means a generic class can act upon any data type Generic classes and generic interfaces are also called „parameterized types" because they use a parameter that determines which data type they should work upon

Generic Method: We can make a method alone as generic method by writing the generic parameter before the method return type as:

```
returntypemethodname ()
{
Method code;
}
```

eg: void display_data () { Method body; }

Generic Interface:
It is possible to develop an interface using generic type concept. The general form of generic interface looks like:

interface interface_name

{ //method that accepts any object return_typemethod_name (T object_name); }
Here, T represents any data type which is used in the interface.
We can write an implementation class for the above interface as: class class_name implements interface_name
{ public return_typemethod_name (T object_name)
{ //provide body of the method }
}

Frequently asked Question

1. Implement a generic program using any collection class to count the number of elements in a collection that have a specific property such as even numbers, odd number, prime number and palindromes.
2. What Is a Generic Type Parameter
3. What Are Some Advantages of Using Generic Types
4. If a Generic Type Is Omitted When Instantiating an Object, Will the Code Still Compile
5. How Does a Generic Method Differ from a Generic Type?
6. What Is Type Inference?
7. What Is a Bounded Type Parameter
8. Is It Possible to Declared a Multiple Bounded Type Parameter?
9. What Is a Wildcard Type

Assignment : 8

Title : File Handling

Problem Statement
Implement a program for maintaining a student records database using File Handling. Student has Student_id, name, Roll_no, Class, marks and address. Display the data for five students.
i) Create Database
ii)Display Database
iii) Clear Records
iv)Modify record
v)Search Record

Objectives: To understand the concept of Java FileWriter and FileReader classes.

Theory:
Importance of file handling?

A Stream represents flow of data from one place to another place Input Streams reads or accepts data Output Streams sends or writes data to some other place All streams are represented as classes in javaio package The main advantage of using stream concept is to achieve hardware independence This is because we need not change the stream in our program even though we change the hardware Streams are of two types in Java:

Byte Streams: Handle data in the form of bits and bytes Byte streams are used to handle any characters (text), images, audio and video files For example, to store an image file (gif or jpg), we should go for a byte stream To handle data in the form of 'bytes' the abstract classes: InputStream and OutputStream are used The important classes of byte streams are:

FileWriter is a class which is in java.io package that is use to create a file by directly writing characters. Java FileWriter and FileReader classes are used to write and read data from text files (they are Character Stream classes). Reading and writing take place character by character, which increases the number of I/O operations and effects performance of the system. BufferedWriter can be used along with FileWriter to improve speed of execution.

FileWriter
FileWriter is useful to create a file writing characters into it.
- This class inherits from the OutputStream class.

- The constructors of this class assume that the default character encoding and the default byte-buffer size are acceptable. To specify these values yourself, construct an OutputStreamWriter on a FileOutputStream.
- FileWriter is meant for writing streams of characters. For writing streams of raw bytes, consider using a FileOutputStream.
- FileWriter creates the output file, if it is not present already.

Constructors of FileWriter

1. FileWriter(String filepath)
2. FileWriter(String filepath, boolean append)
3. FileWriter(File fileobj)

Methods of FileWriter

Method Name	Description
public void write(String text)	Use to write String into file.
public void write(char c)	se to write char into file.
public void close()	Use to to close the file object.
public void flush()	Use to flush the FileWriter contents.

If you will not close the file object then your file data may be lost so don't forget to close file object using close() method.

Buffer in java

File Manipulation and operations

Tokenizing the Input Using the Scanner Class

Example

```
import java.io.*;
class FileWriterTest{
        public static void main(String[] args)throws IOException {
        FileWriterfw=new FileWriter("myfile.txt");
        BufferedReaderbr=new BufferedReader(new InputStreamReader(System.in));
      char ch;
        System.out.println("Enter Char to Exit '@'");
        while( (ch=(char) br.read()) !='@' ){
                fw.write(ch);
       }
        fw.close();
        }
}
```

In the above step a new file will be created every time and previous data will be lost.
FileWriterfw = new FileWriter("myfile.txt",true);

In this case file will not be create every time, If file already exist in given location then it will append contents to existing file because mode true is added at the time creating FileWriter object.

FileReader
FileReader class is use to read text from the file.
Constructors of FileReader
1. FileReader(String filepath)
2. FileReader(File fileobj)

Methods of FileReader
1. int read() : Use to return integer value of next character.
2. int read(char buff[]) : Use to up to buffer length.
3. abstract void close() : Use to close the input source

Example

```
import java.io.*;
class FileWriterTest{
        public static void main(String[] args)throws IOException {
        FileReaderfr = new FileReader("myfile.txt");
        intch;
System.out.println("File contents are:");
          while((ch=fr.read())!=-1){
        System.out.print((char)ch);
      }
        fr.close();
        }
}
```

Buffer in java
A buffer is a memory block that is used to store data. Buffer improved the speed of execution while reading and writing data. We can improve the speed by execution using the following Buffered class.
Buffer Classes: There are four types of buffer classes which work with Stream classes.

BufferedReader
BufferedWriter
BufferedInputStream
BufferedOutputStream

Implementation
Algorithm for Adding Records:-

1. Start
2. Open the database file.
3. Read data from the user.

4. Print the data to file.
5. Close the file.
6. End

Algorithm for Displaying Records:-

1. Start
2. Open the database file.
3. Read data from the filr.
4. Print the data on screen.
5. Close the file.
6. End

Algorithm for Clearing All Records:-

1. Start
2. Overwrite the database file with a blank file.
3. Close the file.
4. End

Frequently asked Question

1. Give the basic methods in File class?
2. Why to use FileWriter class and give its advantages .
3. Write a File-Copy program which copies the content of one file to another. Take both the file names from the user
4. How to read data from a file ,using FileReader class?
5. Give usage of BufferedWriter and BufferedReader classes in Java with example
6. Write a code to generate database for Criket player ising file handling operations

Assignment: 9

Title: Case **Study**

Aim: Using conceptsof Object Oriented programming develop solutionfor any one application

1) Banking solution contains followingoperations such as 1. Create an account 2.Deposit money 3.Withdraw money 4. Honor daily withdrawal limit 5. Check the balance 6. Display Account information.

2) Inventory management contains followingoperations such as List of all products 2. Display individual
product information 3. Purchase 4. Shipping 5.Balance stock 6. Loss and Profit calculation.

Objectives: To implement real time context.

Sample Code:
Create following classes and Methods
class Account : Set balance(),Getbalance(),setAccount type(),getaccounttype
Class Bank: creatAccount(),withdrawAmount(),depositAmount(),displayinfo()
Class customer: getCustomerName(),setCustomerName(),getcustomerAge(),setCustomerAge()
Class saving account: setMinimumBalance (),withdraw()

Input:
Enter your name: Sai
Enter your age: 15
Minimum age should be 18 to create an account.
Please enter valid age: 21
Enter your account Id: 1
Enter your account type: savings
Enter balance: 10000
Enter minimum balance: 1000

Output :
1. Create Account
2.Display Account
3.Check Balance
4.Deposit Amount
5.Withdraw Amount
Enter your choice: 1
Enter your name: Sai
Enter your age: 15
Minimum age should be 18 to create an account.
Please enter valid age: 21
Enter your account Id: 1
Enter your account type: savings
Enter balance: 10000
Enter minimum balance: 1000

```
Do you want to perform more actions? (yes/no): yes
1.Create Account
2. Display Account
3.Check Balance
4.Deposit Amount
5.Withdraw Amount
Enter your choice: 2
Welcome Sai Pande! Following are your account details:
Age :21
Account Id: 1
Account Type: savings
Balance: 10000.0
Minimum balance: 1000.0
Do you want to perform more actions? (yes/no): yes
1.Create Account
2.Display Account
3.Check Balance
4.Deposit Amount
5.Withdraw Amount
Enter your choice: 3
Balance is: 10000.0
Do you want to perform more actions? (yes/no): yes
1.Create Account
2.Display Account
3.Check Balance
4.Deposit Amount
5.Withdraw Amount
Enter your choice: 4
Enter the amount you want to deposit: 20000
Amount deposited successfully. Balance is: 30000.0
Do you want to perform more actions? (yes/no): yes
1.Create Account
2.Display Account
3.Check Balance
4.Deposit Amount
5.Withdraw Amount
Enter your choice: 5
Enter the amount you want to withdraw: 30000
Withdrawal failed. Maximum limit of withdrawal in one transaction is Rs.20000.
Do you want to perform more actions? (yes/no): yes
1. Create Account
2.Display Account
3.Check Balance
4.Deposit Amount
5.Withdraw Amount
Enter your choice: 5
Enter the amount you want to withdraw: 15000
Withdrawal successful. Balance is: 15000.0
```

Frequently Ask Question:

1. List the features which are used for application development
2. How the polymorphism applied
3. Can we apply interface or abstract class in given case study ? How.
4. Did the application takes care of garbage collection

5. Have you applied user defined exceptions in given case study ? Give examples .
6. How many objects are created and how they are stored in memory

Assignment : 10

Title: Factory Design pattern

Aim: Design and implement Factory design pattern for the given context. Consider Carbuilding process, which requires many steps from allocating accessories to final makeup. These steps should be written as methods and should be called while creating an instance of a specific car type. Hatchback, Sedan, SUV could be the subclasses of Car class. Car class and its subclasses, CarFactory and TestFactoryPattern should be implemented.

Objectives: To learn the concept of Design pattern

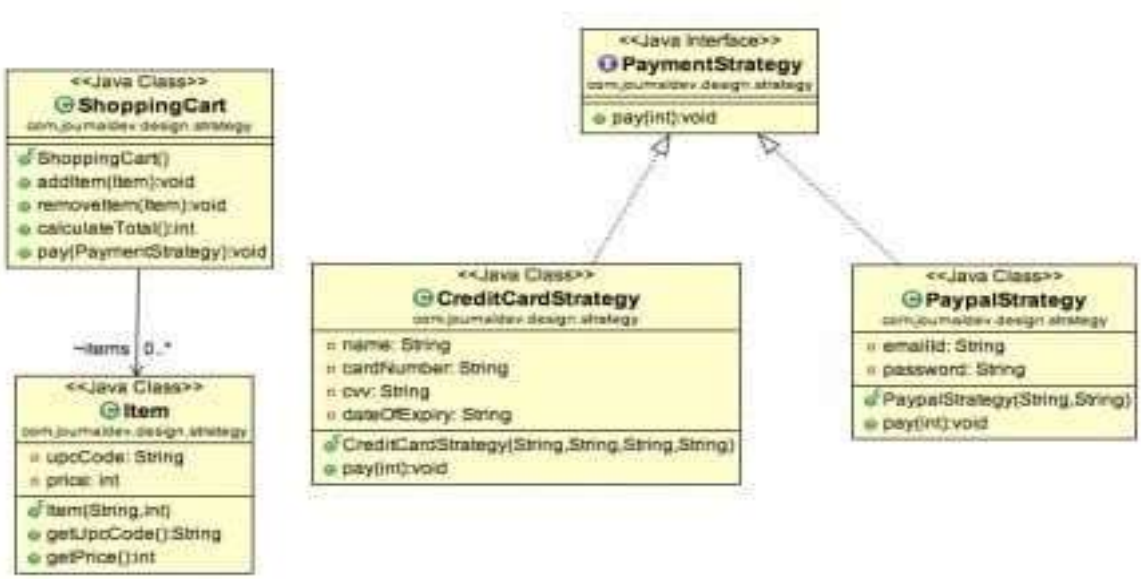
Theory:

1. Design pattern
2. Factory design pattern diagram with example
3. Advantages of factory design pattern
4. Usage and the application where factory design patterns can be applied .

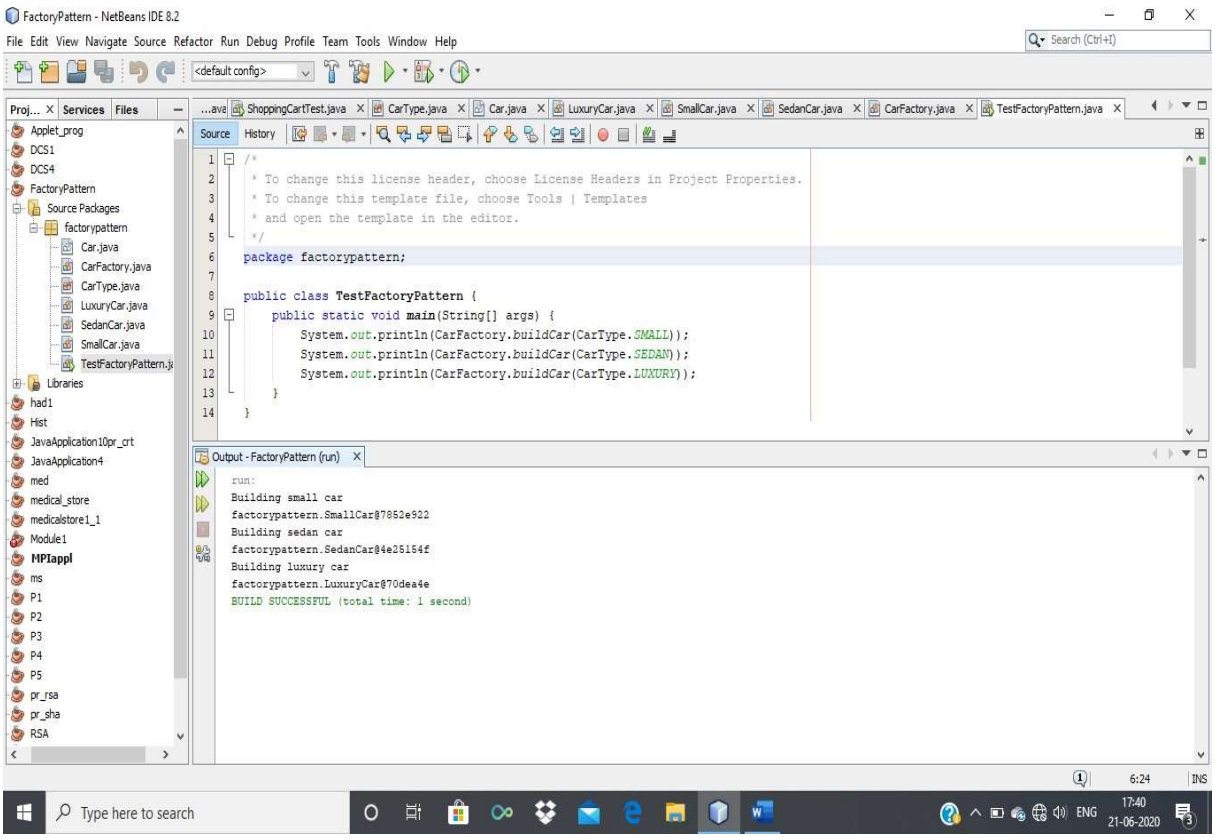
Sample Code:

- Draw the class diagram for given context
- Crate classes such as **Car.java ,CarFactory.java ,**CarType .java
- LuxuryCar.java ,SedanCar.java ,SmallCar.java ,TestFactorypatern.java

Input: Design and implement Factory design pattern for the given context. Consider Car building process, which requires many steps from allocating accessories to final makeup. These steps should be written as methods and should be called while creating an instance of a specific car type. Hatchback, Sedan, SUV could be the subclasses of Car class. Car class and its subclasses, CarFactory and TestFactoryPattern should be implemented.



Output :.
Factory pattern –
Building small car
factorypattern.SmallCar@7852e922
Building sedan car
factorypattern.SedanCar@4e25154f
Building luxury car
factorypattern.LuxuryCar@70dea4e
BUILD SUCCESSFUL (total time: 1 second)



Frequently Ask Question:

1. Give the applications where design patterns can be applied ?
2. Why factory pattern?
3. Explain factory pattern?
4. Draw the design Pattern with a context.
5. Give examples of creational design pattern .
6. Design application by applying the factory pattern .
7. Represent and implement a *Shape* interface which implements Circle ,Square, rectangle using Fatory pattern
8. Represent and implement for bill generation using GetPlanFactory to get a Plan object. Pass information (Domestic / commercial/ institutional) to get the type of object it needs.

Assignment : 11

Title: Strategy Design Pattern

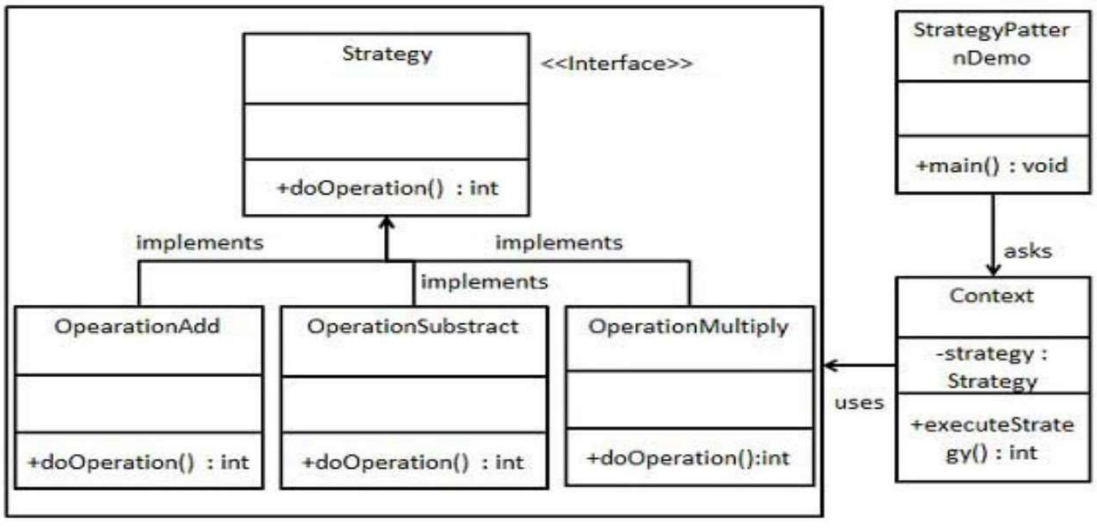
Aim: Implement and apply Strategy Design pattern for simple Shopping Cart where three payment strategies are used such as Credit Card, PayPal, BitCoin. Create the interface for strategy pattern and give concrete implementation for payment.

Objectives: To learn the concept of strategy design pattern

Theory:

1. What is strategy design pattern
2. Design pattern representation
3. Intent
4. Solution of given context with diagram

In Strategy pattern, a class behavior or its algorithm can be changed at run time. This type of design pattern comes under behavior pattern. In Strategy pattern, we create objects which represent various strategies and a context object whose behavior varies as per its strategy object.The strategy object changes the executing algorithm of the context object.We are going to create a Strategy interface defining an action and concrete strategy classes implementing the Strategy interface. Context is a class which uses a Strategy.



Step 1
Create an interface.
Strategy.java

```
public interface Strategy
 { publicintdoOperation(int num1, int num2);
}
```

Step 2
Create concrete classes implementing the same interface.
OperationAdd.java
```
public class OperationAdd implements Strategy{
@Override
publicintdoOperation(int num1, int num2)
{
return num1 + num2;
}
}
```
OperationSubstract.java
```
public class OperationSubstract implements Strategy
{ @Override
publicintdoOperation(int num1, int num2)
 {
return num1 - num2;
 }
}
```

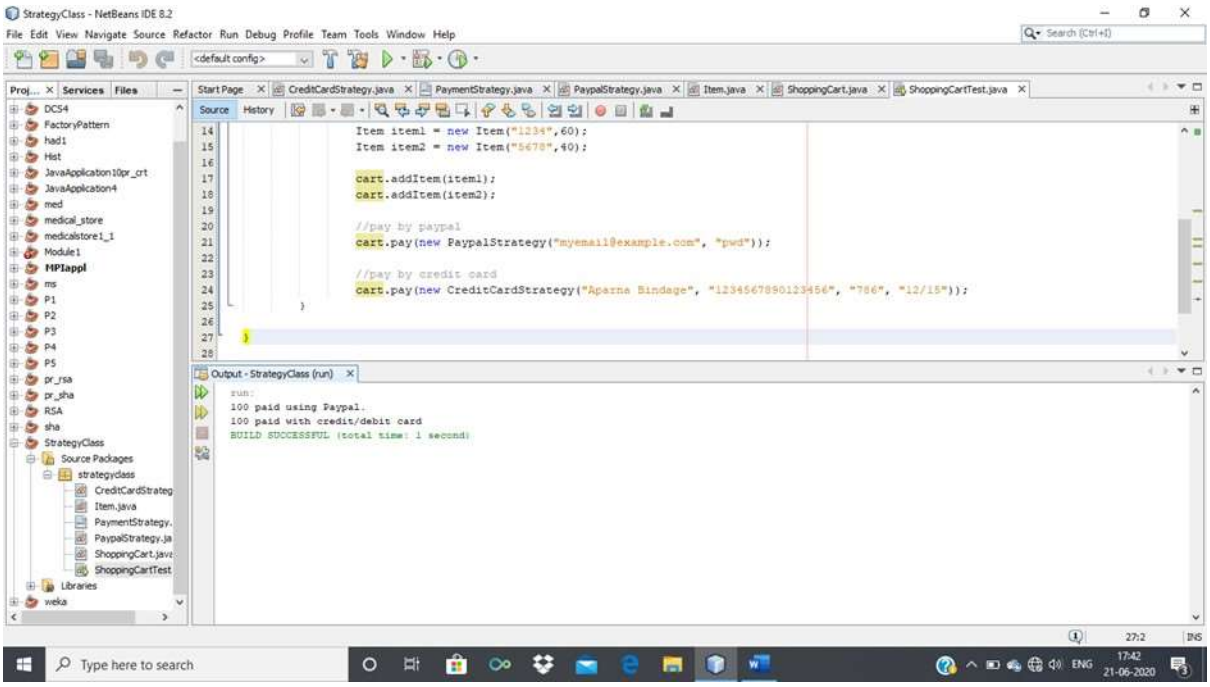
OperationMultiply.java
```
 public class OperationMultiply implements Strategy
{ @Override
publicintdoOperation(int num1, int num2)
{ return num1 * num2;
}
}
```
Step 3
Create Context Class.

Context.java
```
public class Context
{
private Strategy strategy;
public Context(Strategy strategy){
this.strategy = strategy;
}
publicintexecuteStrategy(int num1, int num2){ return strategy.doOperation(num1, num2);
}
}
```
Step 4
 Use the Context to see change in behaviour when it changes its Strategy.
StrategyPatternDemo.java
```
public class StrategyPatternDemo {
public static void main(String[] args) {
Context context = new Context(new OperationAdd());
System.out.println("10 + 5 = " + context.executeStrategy(10, 5));
context = new Context(new OperationSubstract());
System.out.println("10 - 5 = " + context.executeStrategy(10, 5));

context = new Context(new OperationMultiply());
System.out.println("10 * 5 = " + context.executeStrategy(10, 5));
}
}
```
Output:
```
 10 + 5 = 15
10 - 5 = 5
```

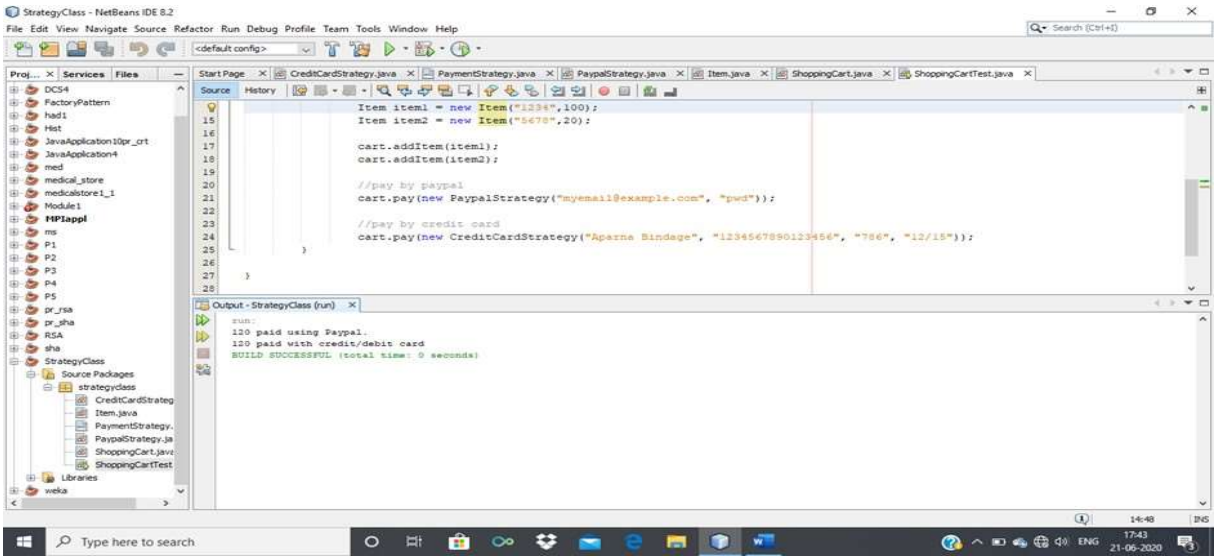
10 * 5 = 50

Output:

Strategy Pattern –
100 paid using Paypal.
100 paid with credit/debit card
BUILD SUCCESSFUL (total time: 1 second)



120 paid using Paypal.
120 paid with credit/debit card
BUILD SUCCESSFUL (total time: 0 seconds)



Test case or Validation:

1. Identify an algorithm (i.e. a behavior)
2. Specify the signature for that algorithm in an interface.
3. Bury the alternative implementation details in derived classes.

Frequently Ask Question:

1. Give intents of Strategy Design pattern
2. How problems can be designed and solved using design pattern
3. Represent Solution of real world problem using Strategy design pattern
4. Give Real-World Analogy of strategy design pattern .
5. Design application by applying the Strategy design pattern .
6. Represent and implement Strategy design pattern to perform mathematical operations such as add,sub,mul,div
7. Represent and implement Strategy design pattern for sorting operation such as Quick sort ,Merge sort etc
8. Represent and implement Strategy design pattern for searching techniques such as Sequential Search, Binary Search etc.