Assignment No 1
(Classes and Objects)

Aim: 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.

Complex numbers are numbers that consist of two parts — a real number and an imaginary number. Complex numbers are the building blocks of more intricate math, such as algebra. The standard format for complex numbers is a + bi, with the real number first and the imaginary number last.

General form for any complex number is:

a+ib

Where "a" is real number and "b" is Imaginary number.

Construction of Complex number

For creating a complex numbers, we will pass imaginary numbers and real numbers as parameters for constructors.

Program:

```
public class Complex{
   float real,img;
   Complex(){          }
   Complex(float r,float i){
      real = r;
      img = i;
   }
   Complex add(Complex a){
      Complex temp = new Complex();
      temp.real = this.real + a.real;
      temp.img = this.img + a.img;
      return temp;
   }
   Complex sub(Complex a){
      Complex temp = new Complex();
      temp.real = this.real - a.real;
      temp.img = this.img - a.img;
      return temp;
   }
      public static void main(String args[]){
      Complex c1 = new Complex(5.6f,6.12f);
      Complex c2 = new Complex(12.6f,4.03f);
      Complex result;
      result          =          c1.add(c2);
      System.out.println("Addition... "+result.real+"+"+result.img+"i");
      result          =          c1.sub(c2);
      System.out.println("Subtraction... "+result.real+"+"+result.img+"i");
   }
}
```

Output:

java -cp /tmp/RGwcsL7qcy Complex

Addition.....+10.15i

Subtraction..... .....+2.0899997i

Assignment No 2

(Polymorphism)

Aim: 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 order Copies(). 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.

Class is a detailed description, the definition, and the template of what an object will be. But it is not the object itself. Also, what we call, a class is the building block that leads to Object-Oriented Programming. It is a user-defined data type, that holds its own data members and member functions, which can be accessed and used by creating an instance of that class. It is the blueprint of any object. Once we have written a class and defined it, we can use it to create as many objects based on that class as we want. In Java, the class contains fields, constructors, and methods. For example, consider the Class of Accounts. There may be many accounts with different names and types, but all of them will share some common properties, as all of them will have some common attributes like balance, account holder name, etc. So here, the Account is the class.

Program:

```
import java.util.Scanner;
class Publication{
   String title;
   int copies,price;
   void saleCopy(){

   }
}

class Book{
   String title,author;
   int copies,price;
   void saleCopy(){
      price = 250;
      System.out.println("Total sale: "+copies*price);
   }
   void orderCopies(){
      Scanner in = new Scanner(System.in);
      System.out.println("Enter no. of copies");
      copies = in.nextInt();
   }
}

class Magazine{
   String title;
   int copies,price;
```

```
    void saleCopy(){

    }
    void orderQty(){

    }
    void currentIssue(){

    }
    void receiveIssue(){

    }
}

public class P{
    public static void main(String args[]){
        Book b = new Book();
        b.orderCopies();
        b.saleCopy();
    }
}
/*
Output:
(base) os@os-Vostro-3268:~$ javac P.java
(base) os@os-Vostro-3268:~$ java P
Enter no. of copies
5
Total sale: 1250
*/
```

Assignment No 3

(Inheritance)

Aim: 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, Assistant Professor, Associate Professor and Professor 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.

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

Program:

```
class Employee{
 String name, add, mail;
 float id, mobile,basic;
 void salary()
 {
   float da,hra,pf,cf,gross;
   da = basic * 97/100;
   hra = basic * 10/100;
   pf = basic * 12/100;
   cf = basic * 0.1f/100;
   gross = basic + da + hra - pf - cf;
   System.out.println("Name: " + name);
   System.out.println("Basic Salary: " + basic);
   System.out.println("Gross Salary: " + gross);
   }
}

class Programmer extends Employee{
  Programmer(String name,int sal){
          this.name = name;
          basic = sal;
  }
```

```
}
class Assistant_Professor extends Employee{
        Assistant_Professor(String name,int sal){
        this.name = name;
        basic = sal;
        }
}
class Associate_Professor extends Employee{
        Associate_Professor(String name,int sal){
        this.name = name;
        basic = sal;
        }
}
class Professor extends Employee{
        Professor(String name,int sal){
        this.name = name;
        basic = sal;
        }
}
public class Inheritance{
        public static void main(String args[]){
        Assistant_Professor ast = new Assistant_Professor("Jai",40000);
        ast.salary();
        Associate_Professor aso = new Associate_Professor("Satyajeet",60000);
        aso.salary();
        Professor pro = new Professor("Ram",75000);
        pro.salary();
        Programmer pm = new Programmer("Akash",100000);
        pm.salary();
        }
}
```

/*

Output:

(base) os@os-Vostro-3268:~$ javac Inheritance.java
(base) os@os-Vostro-3268:~$ java Inheritance
Name: Jai
Basic Salary: 40000.0
Gross Salary: 77960.0
Name: Satyajeet
Basic Salary: 60000.0
Gross Salary: 116940.0
Name: Ram
Basic Salary: 75000.0
Gross Salary: 146175.0
Name: Akash
Basic Salary: 100000.0
Gross Salary: 194900.0

****/***

Assignment No 4

(Dynamic Binding)

Aim: ***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.***

Dynamic binding in Java is an essential concept that every programmer must be familiar with as it solves various real-time problems.
To understand the working of dynamic binding, we must be aware of another type of binding, known as Static Binding.

Static Binding

Also known as Early Binding, it is a binding that is resolved by the compiler at compile time.

It must be noted that the binding of private, final and static methods is done during compile time. This is due to the fact that static binding provides a better performance. The compiler is aware that these methods cannot be overridden and are always accessed by the object of local class. Thus, the object of the class(local class) is determined easily by the compiler.

```
public class Main
{
public static class superclass
{
static void print()
{
System.out.println("This is the superclass");
}
}
public static class subclass extends superclass
{
static void print()
{
System.out.println("This is the subclass");
}
}
public static void main(String[] args)
{
superclass sup = new superclass();
superclass sub = new subclass();
sup.print();
sub.print();
}
}
```

In the example given above, the print method of superclass is static, and the compiler is aware that it cannot be overridden in the subclass.
Thus, the output of the code is as follows:

Output:

```
This is the superclass
This is the superclass
```

Moving on with this article on Dynamic Binding in Java

Dynamic Binding In Java

In this type of binding, the method to be called is not decided by the compiler. An appropriate example for dynamic binding is Overriding. Here, the parent class and the child class have the same method.

```
public class Main
{
public static class superclass
{
void print()
{
System.out.println("This is superclass");
}
}
public static class subclass extends superclass
{
@Override
void print()
{
System.out.println("This is subclass");
}
}
public static void main(String[] args)
{
superclass sup = new superclass();
superclass sub = new subclass();
sup.print();
sub.print();
}
}
```

Output:

In the example given above, the compiler is unaware about the print that is to be called. The compiler works by referencing variables and not by the type of objects. Due to this, binding is deferred to runtime.
This is superclass
This is subclass

Difference between Static and Dynamic Binding

- The static binding is used by private, static and final members. For the virtual methods in Java, the binding is done at the run time in accordance with the run time object.
- While static binding uses Type information, dynamic binding makes use of Objects for binding.
- Static supports Overloading, while Dynamic Binding Supports Overriding.

Program:

```
import java.util.Scanner;
abstract class Shape{
	double val1,val2;
	void input() {
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter first value");
		val1 = sc.nextDouble();
		System.out.println("Enter second value");
		val2 = sc.nextDouble();
	}
	abstract void compute_area();
	}
	class Triangle extends Shape {
		void compute_area() {
			double area;
			area = 1.0f/2.0f * val1 * val2;
			System.out.println("Triangle Area: " + area);
	}
}
class Rectangle extends Shape {
	void compute_area() {
		double area;
		area = val1 * val2;
		System.out.println("Rectangle Area: " + area);
	}
}
```

```
public class Dynamic {

	public static void main(String args[]) {

		Shape s;

		Triangle t = new Triangle();

		Rectangle r = new Rectangle();

		s = t;

		s.input();

		s.compute_area();

		s = r;

		s.input();

		s.compute_area();

	}

}
```

Output:

```
java -cp /tmp/WvFGZKJONF Dynamic
Enter first value
10
Enter second value
20
Triangle Area: 100.0
Enter first value
20
Enter second value
20
Rectangle Area: 400.0
```

Assignment No 5

(Interface)

***Aim:* 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.**

An **Interface in Java** programming language is defined as an abstract type used to specify the behavior of a class. An interface in Java is a blueprint of a class. A Java interface contains 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 the method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also **represents the IS-A relationship**. Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

- Interfaces specify what a class must do and not how. It is the blueprint of the class.
- An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.
- If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.
- A Java library example is Comparator Interface. If a class implements this interface, then it can be used to sort a collection.

S. No.	Class	Interface
1.	In class, you can instantiate variables and create an object.	In an interface, you can't instantiate variables and create an object.
2.	Class can contain concrete(with implementation) methods	The interface cannot contain concrete(with implementation) methods
3.	The access specifiers used with classes are private, protected, and public.	In Interface only one specifier is used- Public.

Program

```
import java.io.*;
interface Vehicle {
void changeGear(int a);
void speedUp(int a);
void applyBrakes(int a);
}

class Bicycle implements Vehicle{
int speed;
int gear;
public void changeGear(int newGear){
gear = newGear;
}

public void speedUp(int increment){
speed = speed + increment;
}

public void applyBrakes(int decrement){
speed = speed - decrement;
}

public void printStates() {
System.out.println("speed: " + speed
+ " gear: " + gear);
}
}

class Bike implements Vehicle {
int speed;
int gear;
public void changeGear(int newGear){
gear = newGear;
}

public void speedUp(int increment){
speed = speed + increment;
}

public void applyBrakes(int decrement){
speed = speed - decrement;
}
```

```
 public void printStates() {
 System.out.println("speed: " + speed
 + " gear: " + gear);
 }
 }
 class StartVehicle {
 public static void main (String[] args) {
 Bicycle bicycle = new Bicycle();
 bicycle.changeGear(2);
 bicycle.speedUp(3);
 bicycle.applyBrakes(1);

 System.out.println("Bicycle present state :");
 bicycle.printStates();

 Bike bike = new Bike();
 bike.changeGear(1);
 bike.speedUp(4);
 bike.applyBrakes(3);
 System.out.println("Bike present state :");
 bike.printStates();
 }
}
```

Output:

```
Bicycle present state:

speed: 2 gear: 2

Bike present state:

speed: 1 gear: 1
```

Assignment No 6

(Exception Handling)

Aim: ***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***

Exception Handling in Java is one of the effective means to handle the runtime errors so that the regular flow of the application can be preserved. Java Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions. Exceptions can be caught and handled by the program. When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception, such as the name and description of the exception and the state of the program when the exception occurred.

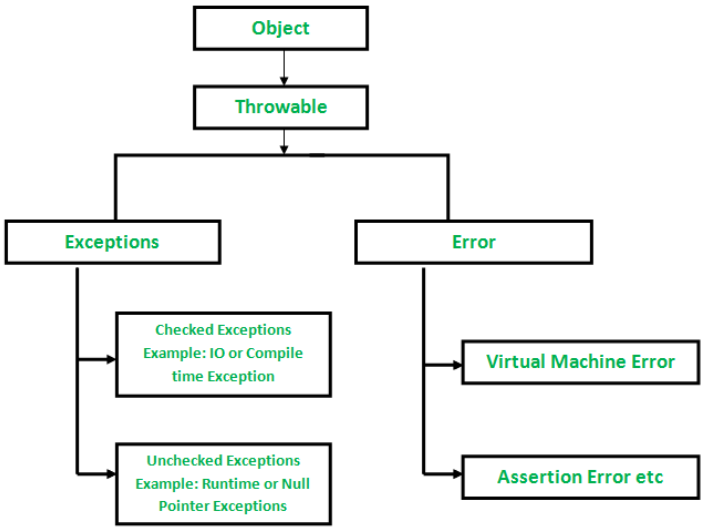
Major reasons why an exception Occurs
Invalid user input
Device failure
Loss of network connection
Physical limitations (out of disk memory)
Code errors
Opening an unavailable file
Errors represent irrecoverable conditions such as Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc. Errors are usually beyond the control of the programmer, and we should not try to handle errors.

Let us discuss the most important part which is the differences between Error and Exception that is as follows:

- **Error:** An Error indicates a serious problem that a reasonable application should not try to catch.
- **Exception:** Exception indicates conditions that a reasonable application might try to catch.

Exception Hierarchy

All exception and error types are subclasses of class **Throwable**, which is the base class of the hierarchy. One branch is headed by **Exception**. This class is used for exceptional conditions that user programs should catch. NullPointerException is an example of such an exception. Another branch, **Error** is used by the Java run-time system(JVM) to indicate errors having to do with the run-time environment itself(JRE). StackOverflowError is an example of such an error.



Exceptions can be categorized in two ways:

1. **Built-in Exceptions**
   - Checked Exception
   - Unchecked Exception
2. **User-Defined Exceptions**

Let us discuss the above-defined listed exception that is as follows:

A. Built-in Exceptions:

Built-in exceptions are the exceptions that are available in Java libraries. These exceptions are suitable to explain certain error situations.

- **Checked Exceptions:** Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.

- **Unchecked Exceptions:** The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time. In simple words, if a program throws an unchecked exception, and even if we didn't handle or declare it, the program would not give a compilation error.

B. User-Defined Exceptions:

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, users can also create exceptions, which are called 'user-defined Exceptions'.

Assignment No 7

(Template)

Aim: ***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.***

Generics means **parameterized types**. The idea is to allow type (Integer, String, … etc., and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types. An entity such as class, interface, or method that operates on a parameterized type is a generic entity.

Why Generics?

The **Object** is the superclass of all other classes, and Object reference can refer to any object. These features lack type safety. Generics add that type of safety feature. We will discuss that type of safety feature in later examples.
Generics in Java are similar to templates in C++. For example, classes like HashSet, ArrayList, HashMap, etc., use generics very well. There are some fundamental differences between the two approaches to generic types.

Types of Java Generics

Generic Method: Generic Java method takes a parameter and returns some value after performing a task. It is exactly like a normal function, however, a generic method has type parameters that are cited by actual type. This allows the generic method to be used in a more general way. The compiler takes care of the type of safety which enables programmers to code easily since they do not have to perform long, individual type castings.

Generic Classes: A generic class is implemented exactly like a non-generic class. The only difference is that it contains a type parameter section. There can be more than one type of parameter, separated by a comma. The classes, which accept one or more parameters, are known as parameterized classes or parameterized types.

```
// Java program to show multiple
// type parameters in Java Generics

// We use < > to specify Parameter type
class Test<T, U>
{
    T obj1;  // An object of type T
    U obj2;  // An object of type U

    // constructor
    Test(T obj1, U obj2)
    {
        this.obj1 = obj1;
        this.obj2 = obj2;
    }

    // To print objects of T and U
    public void print()
    {
```

```
        System.out.println(obj1);
        System.out.println(obj2);
    }
}

// Driver class to test above
class Main
{
    public static void main (String[] args)
    {
        Test <String, Integer> obj =
            new Test<String, Integer>("Hello", 15);

        obj.print();
    }
}
```

Output

```
Hello

15
```

Program:

Assignment No 8

(File Handling)

Aim: ***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

In Java, with the help of File Class, we can work with files. This File Class is inside the java.io package. The File class can be used by creating an object of the class and then specifying the name of the file.

Why File Handling is Required?

- File Handling is an integral part of any programming language as file handling enables us to store the output of any particular program in a file and allows us to perform certain operations on it.
- In simple words, file handling means reading and writing data to a file

In Java, the concept Stream is used in order to perform I/O operations on a file. So at first, let us get acquainted with a concept known as Stream in Java.

Streams in Java

- In Java, a sequence of data is known as a stream.
- This concept is used to perform I/O operations on a file.
- There are two types of streams :

1. Input Stream:

The Java InputStream class is the superclass of all input streams. The input stream is used to read data from numerous input devices like the keyboard, network, etc. InputStream is an abstract class, and because of this, it is not useful by itself. However, its subclasses are used to read data.

There are several subclasses of the InputStream class, which are as follows:

1. AudioInputStream
2. ByteArrayInputStream
3. FileInputStream
4. FilterInputStream
5. StringBufferInputStream
6. ObjectInputStream

Creating an InputStream

```
// Creating an InputStream

InputStream obj = new FileInputStream();
```

Here, an input stream is created using FileInputStream.

S No.	Method	Description
1	read()	Reads one byte of data from the input stream.
2	read(byte[] array)()	Reads byte from the stream and stores that byte in the specified array.
3	mark()	It marks the position in the input stream until the data has been read.
4	available()	Returns the number of bytes available in the input stream.
5	markSupported()	It checks if the mark() method and the reset() method is supported in the stream.
6	reset()	Returns the control to the point where the mark was set inside the stream.
7	skips()	Skips and removes a particular number of bytes from the input stream.
8	close()	Closes the input stream.

2. Output Stream:

The output stream is used to write data to numerous output devices like the monitor, file, etc. OutputStream is an abstract superclass that represents an output stream. OutputStream is an abstract class and because of this, it is not useful by itself. However, its subclasses are used to write data.

There are several subclasses of the OutputStream class which are as follows:

1. ByteArrayOutputStream
2. FileOutputStream
3. StringBufferOutputStream
4. ObjectOutputStream
5. DataOutputStream
6. PrintStream

Creating an OutputStream

```
// Creating an OutputStream

OutputStream obj = new FileOutputStream();
```

Here, an output stream is created using FileOutputStream.

S. No.	Method	Description
1.	write()	Writes the specified byte to the output stream.
2.	write(byte[] array)	Writes the bytes which are inside a specific array to the output stream.
3.	close()	Closes the output stream.
4.	flush()	Forces to write all the data present in an output stream to the destination.

Based on the data type, there are two types of streams :

1. Byte Stream:

This stream is used to read or write byte data. The byte stream is again subdivided into two types which are as follows:

- **Byte Input Stream:** Used to read byte data from different devices.
- **Byte Output Stream:** Used to write byte data to different devices.

2. Character Stream:

This stream is used to read or write character data. Character stream is again subdivided into 2 types which are as follows:

- **Character Input Stream:** Used to read character data from different devices.
- **Character Output Stream:** Used to write character data to different devices.

Owing to the fact that you know what a stream is, let's polish up File Handling in Java by further understanding the various methods that are useful for performing operations on the files like creating, reading, and writing files.

Java File Class Methods

The following table depicts several File Class methods:

Method Name	Description	Return Type
canRead()	It tests whether the file is readable or not.	Boolean
canWrite()	It tests whether the file is writable or not.	Boolean
createNewFile()	It creates an empty file.	Boolean
delete()	It deletes a file.	Boolean
exists()	It tests whether the file exists or not.	Boolean
length()	Returns the size of the file in bytes.	Long

Method Name	Description	Return Type
getName()	Returns the name of the file.	String
list()	Returns an array of the files in the directory.	String[]
mkdir()	Creates a new directory.	Boolean
getAbsolutePath()	Returns the absolute pathname of the file.	String

Program:

Assignment No 9

(Case Study)

Aim: ***Using concepts of Object Oriented programming develop solution for any one application***

1) Banking solution contains following operations 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 following operations such as

1. List of all products
2. Display individual product information
3. Purchase
4. Shipping
5. Balance stock
6. Loss and Profit calculation.

1) Banking Solution

Assignment No 10

(Factory Design pattern)

Aim: ***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.***

Assignment No 11

(Strategy Design Patten)

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.***