Object Oriented Programming Using Java

(For Computer Engg. 4th Sem)

Chapter 1
Introduction and Features

1. the features of Object-Oriented Programming

Features of OOPs :-

1) Data Abstraction :- Data Abstraction refers to the act of representing essential features without including the background details or explanations.

2) Polymorphism :- Polymorphism is made of two terms- Poly and Morph. Poly means 'Many' and Morph means 'Forms'. Polymorphism is the property by which the same message can be sent to the objects of several different classes.

3) Data Encapsulation or Data Hiding :- The wrapping up of data and functions into a single unit is called Data Encapsulation or Data Hiding.

4) Inheritance :- Inheritance is the property by which one class of things inherit or derive the properties of another class of things. A class from which another class is derived, is called **Base Class or Super Class**. A class which is derived from Base Class, is called **Sub Class or Derived Class.** Java doesn't support multiple inheritance.

5) Message Passing :- The calling member functions by objects of the class is called Message Passing.

2. Differentiate Procedure-oriented Programming and Object-oriented Programming

S.No.	Procedure Oriented Programming	Object Oriented Programming
1	Program is divided into small parts called functions	Program is divided into parts called objects.
2	Importance is not given to data but to functions as well as sequence of actions to be done.	Importance is given to the data rather than procedures or functions.
3	It does not model real world	It models real world
4	It follows Top-Down Approach.	It follows Bottom-Up Approach.
5	It does not have any proper way for hiding data so it is less secure.	It provides Data Hiding so provides more security.
6	There is no access specifiers.	It has three most important Access Specifiers – private, public and protected.
7	It doesn't support Abstraction, Inheritance, Encapsulation etc.	It supports Abstraction, Inheritance, Encapsulation etc.
8	Example of POP are : C, FORTRAN, Pascal	Example of OOP are : C++, JAVA, VB.NET, C#.NET

Chapter 2
Language Constructs

JVM (Java Virtual Machine) :-

A Java virtual machine (JVM), an implementation of the Java Virtual Machine Specification, interprets compiled Java binary code (called bytecode) for a computer's processor (or "hardware platform") so that it can perform a Java program's instructions.

JDK (Java Development Kit) :-

Writing Java applets and applications needs development tools like JDK. The JDK includes the Java Runtime Environment, the Java compiler and the Java APIs. It's easy for both new and experienced programmers to get started. Java Development Kit (JDK) is a bundle of software components that is used to develop Java based applications.

These tools are the foundation of the JDK. They are the tools you use to create and build applications.

Tool Name	Brief Description
appletviewer	Run and debug applets without a web browser.
Java	The interpreter for the Java programming language.
Javac	The compiler for the Java programming language.
Javadoc	Java Documenation
Javah	To create C header files
Javap	Class file disassemble
Jdb	The Java Debugger.

the types of Java programs :-

There are two types of Java Programs.

Application Programs:- these programs are stand-alone programs. These programs are used to perform particular tasks like payroll, students registration etc. These programs are executed on local computers.

Web based Programs:- These programs are executed on Internet. So, these programs are called web based programs. Examples of these programs are Applets, Servlets etc.

Variable :-

Variables are named locations in the memory that are used to hold or store a value.

Examples :-

```
int a, b, c;
int a = 10, b = 10;
```

There are three kinds of variables in Java:

- Local variables
- Instance variables
- Class/static variables

Local variables: Local variables are declared in methods, constructors, or blocks. Local variables are visible only within the declared method, constructor or block.

Instance variables: Instance variables are declared in a class, but outside a method, constructor or any block. Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed. The instance variables are visible for all methods, constructors and block in the class.

Class/static variables: Class variables also known as static variables are declared with the *static* keyword in a class, but outside a method, constructor or a block. There would only be one copy of each class variable per class, regardless of how many objects are created from it.

the Operators in Java :-

Operators are the symbols that are used to perform various operations on variables, constants and expressions. These variables or constants are called operands.

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators

the terms Operator, Operand and Expression.

Operator:- Operators are the symbols that are used to perform various operations on variables, constants and expressions.

Operands:- The variables or constants on which operations are performed by the operators, are called operands.

Expressions:- Expressions are the statements which are composed of operators and operands.

Control Flow Statements :-

Java Control statements control the order of execution in a java program, based on data values and conditional logic. There are three main categories of control flow statements. These are as follows:-

Selection statements: if, if-else and switch.

Loop statements: while, do-while and for.

Transfer statements: break, continue, return, try-catch-finally.

Managing Input – Output Stream

The java.io package contains every class you might need to perform input and output (I/O) in Java. All these streams represent an input source and an output destination.

A stream can be defined as a sequence of data. The InputStream is used to read data from a source and the OutputStream is used for writing data to a destination.

Byte Streams

Java byte streams are used to perform input and output of 8-bit. The most frequently used classes are **FileInputStream** and **FileOutputStream**.

Character Streams

Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are **FileReader** and **FileWriter.**

Array :-

An array is a collection of data elements of same type that are referred to by the same name.

Declaring Arrays:

```
dataType[] arrayRefVar;
        or
dataType arrayRefVar[];
```

Example:

```
double[] myList;
```

or

```
double myList[];
```

Creating Arrays:

```
arrayRefVar = new dataType[arraySize];
```

Differences between Java and C++

Java	C++
Java does not support pointers, unions, operator overloading, structures etc.	C++ supports structures, unions, operator overloading, pointers etc.
Java has built in support for threads. In Java, there is a Thread class that you inherit to create a new thread and override the run() method.	C++ has no built in support for threads.
There is no *goto* statement in Java.	C++ has *goto* statement.
Java doesn't support multiple inheritance.	C++ supports multiple inheritance.
Java has method overloading, but no operator overloading.	C++ supports both method overloading and operator overloading.
Java is interpreted for the most part and hence platform independent.	C++ generates object code and the same code may not run on different platforms.

Features of Java are as follows:-

- **Object Oriented:** In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform independent:** Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run.
- **Simple:**Java is designed to be easy to learn.
- **Secure:** With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architectural-neutral :**Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors, with the presence of Java runtime system.
- **Portable:**Being architectural-neutral and having no implementation dependent aspects of the specification makes Java portable.
- **Robust:**Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Multithreaded:** With Java's multithreaded feature it is possible to write programs that can do many tasks simultaneously.
- **Interpreted:**Java byte code is translated on the fly to native machine instructions and is not stored anywhere. **High Performance:** With the use of Just-In-Time compilers, Java enables high performance.
- **Distributed:**Java is designed for the distributed environment of the internet.
- **Dynamic:** Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment.

the Data Types in Java :-

Data Type means a kind of information we want to use or store in a variable.

There are two data types available in Java:

Primitive Data Types

Reference/Object Data Types

Primitive Data Types: There are eight primitive data types supported by Java. Primitive Data Types are predefined by the language.

byte, short, int, long, float, double, boolean, char

Reference Data Types:

Reference Data Types are created using defined constructors of the classes. They are used to access objects. For example, Employee, Student etc.

Class objects, and various type of array variables come under reference data type.

Default value of any reference variable is null.

Example: Animal animal = new Animal("horse");

the Operators in Java.

Operators are the symbols that are used to perform various operations on variables, constants and expressions. These variables or constants are called operands.

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators

The Arithmetic Operators:

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:

Operator	Description
+	Addition - Adds values on either side of the operator
-	Subtraction - Subtracts right hand operand from left hand operand
*	Multiplication - Multiplies values on either side of the operator
/	Division - Divides left hand operand by right hand operand
%	Modulus - Divides left hand operand by right hand operand and returns remainder

The Increment and Decrement Operators:

Increment Operator (++) increments the value of an expression by 1. Decrement Operator (--) decrements the value of an expression by 1.

The Relational Operators:

There are following relational operators supported by Java language

Operator	Description
==	Checks if the values of two operands are equal or not, if yes then condition

	becomes true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

The Bitwise Operators:

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs bit-by-bit operation.

Operator	Description
&	Binary AND Operator copies a bit to the result if it exists in both operands.
|	Binary OR Operator copies a bit if it exists in either operand.
^	Binary XOR Operator copies the bit if it is set in one operand but not both.
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.

The Logical Operators:

The Logical Operators are used to combine or negate the expressions containing relational operators

Operator	Description
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.
||	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.
!	Called Logical NOT Operator. Use to reverses the

	logical state of its operand. If a condition is true then Logical NOT operator will make false.

The Assignment Operators:

There are following assignment operators supported by Java language:

Operator	Description
=	Simple assignment operator, Assigns values from right side operands to left side operand
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand

Conditional Operator (? :)

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions.

```
variable x = (expression) ? value if true : value if false
```

Control Flow Statements :-

Java Control statements control the order of execution in a java program, based on data values and conditional logic. There are three main categories of control flow statements;

Selection statements: if, if-else and switch.

Loop statements: while, do-while and for.

Transfer statements: break, continue, return, try-catch-finally.

Selection Statements

The if Statement

The if statement executes a block of code only if the specified expression is true. If the value is false, then the if block is skipped and execution continues with the rest of the program.

The simple if statement has the following syntax:

```
if (<conditional expression>)
{
    <statement>
}
```

The if-else Statement

The if-else statement is an extension of the if statement. If the statements in the if statement fails, the statements in the else block are executed. You can either have a single statement or a block of code within if-else blocks.
The if-else statement has the following syntax:

```
if (<conditional expression>)
{
    <statements>
}
else
{
    <else statements>
}
```

switch case Statement

The switch statement successfully tests the value of an expression against a list of integer or character constants. If a match is found, then the statements following that particular case are executed. If no match is found, then the statement following default are executed.

```
switch (<expression)
{

    case value1:
        <statement 1>
         break;
    case value2:
        <statement2>
            break;
    default:
         <else statement>
         break;
}
```

Iteration Statements

while Statement

The while statement is a looping construct control statement that executes a block of code while a condition is true. The syntax of the while loop is

```
while (<loop condition>)
{
    <statements>
}
```

do-while Loop Statement

The do-while loop is similar to the while loop, except that the test is performed at the end of the loop instead of at the beginning. This ensures that the loop will be executed at least once. The syntax of the do-while loop is

```
do
{
        <loop body>
}while (<loop condition>);
```

for loop

The for loop is a looping construct which can execute a set of instructions a specified number of times.

The syntax of the loop is as follows:

```
for (<initialization>; <loop condition>; <increment expression>)
{
        <loop body>
}
```

Transfer Statements

continue Statement

A continue statement stops the iteration of a loop and causes execution to resume at the top of the nearest enclosing loop.

The syntax of the continue statement is

```
continue; // the unlabeled form
continue <label>; // the labeled form
```

break Statement

The break statement transfers control out of the enclosing loop (for, while, do or switch statement).

The Syntax for break statement is as shown below;

```
break; // the unlabeled form
break <label>; // the labeled form
```

Chapter 3
Classes and Objects

Class - A class can be defined as a template/blue print that describes the behaviors/states that object of its type support.

Syntax :-

```
public class  <class-name>
{
        public static void main(String[] args)
        {
                //Body of Function
        }
}
```

Example :-

```
public class  xyz
{
        public static void main(String[] args)
        {
                System.out.println("Good Morning India");
        }
}
```

Constructor :-

A constructor is a special member function that has the same name as that of class. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Example of a constructor is given below:

```
public class Student
{
        public Student()  //Default Constructor
        {
                // This constructor has no parameter.
        }

        public Student(String name)  //Parameterized Constructor
        {
                // This constructor has one parameter, name.
        }
}
```

Object Reference :-

Object Reference describes the address where object variables and methods are stored.
Example:-

```
Point pt1, pt2;              // pt1 and pt2 are two objects of Point class
pt1 = new Point(20,50);   // Initialization of pt1 object using constructor
pt2=pt1;                     // Object Refernce
```

Command-line Arguments :-

The java command-line argument is an argument i.e. passed at the time of running the java program. The arguments passed from the console can be received in the java program and it can be used as an input.

Simple example of command-line argument in java

```
class CommandLineExample
{
    public static void main(String args[])
    {
            System.out.println("Your first argument is: "+args[0]);
    }
}
```

compile the program by **javac CommandLineExample.java**
run the program by **java CommandLineExample sonoo**
Output: Your first argument is: sonoo

Object :-

Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors -wagging, barking, eating. An object is an instance of a class.

A class provides the blueprints for objects. So basically an object is created from a class. In Java, the new key word is used to create new objects.

There are three steps when creating an object from a class:

- **Declaration:** A variable declaration with a variable name with an object type.
- **Instantiation:** The 'new' key word is used to create the object.
- **Initialization:** The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Example of creating an object is given below:

```
public class Student
{
        public Student(String name)
        {
                // This constructor has one parameter, name.
                System.out.println("Passed Name is :" + name );
        }
        public static void main(String []args)
        {
                // Following statement would create an object myStudent
                Student myStudent = new Student( "Mukesh" );
        }
}
```

If we compile and run the above program, then it would produce the following result:

Passed Name is : Mukesh

Explain the Access Specifiers used in Java.

Java Access Specifiers (also known as Visibility Specifiers) regulate access to classes, fields and methods in Java. These Specifiers determine whether a field or method in a class, can be used or invoked by another method in another class or sub-class. Access Specifiers can be used to restrict access.

Types Of Access Specifiers

In java we have four Access Specifiers and they are listed below.

1. public
2. private
3. protected
4. default(no specifier)

public specifiers

Public Specifiers achieves the highest level of accessibility. Classes, methods, and fields declared as public can be accessed from any class in the Java program, whether these classes are in the same package or in another package.

private specifiers

Private Specifiers achieves the lowest level of accessibility. private methods and fields can only be accessed within the same class to which the methods and fields belong. private methods and fields are not visible within subclasses and are not inherited by subclasses.

protected specifiers

Methods and fields declared as protected can only be accessed by the subclasses in other package or any class within the package of the protected members' class. The protected access specifier cannot be applied to class and interfaces.

default(no specifier)

When you don't set access specifier for the element, it will follow the default accessibility level. There is no default specifier keyword. using default specifier we can access class, method, or field which belongs to same package, but not from outside this package.

Chapter 4
Inheritance

Inheritance :-

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

extends is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

Syntax

```
class BaseClass
{
        .....
        .....
}
class DerivedClass extends BaseClass
{
        .....
        .....
}
```

Inheritance :-

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

extends is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

Syntax

```
class BaseClass
{
        .....
        .....
}
class DerivedClass extends BaseClass
```

{

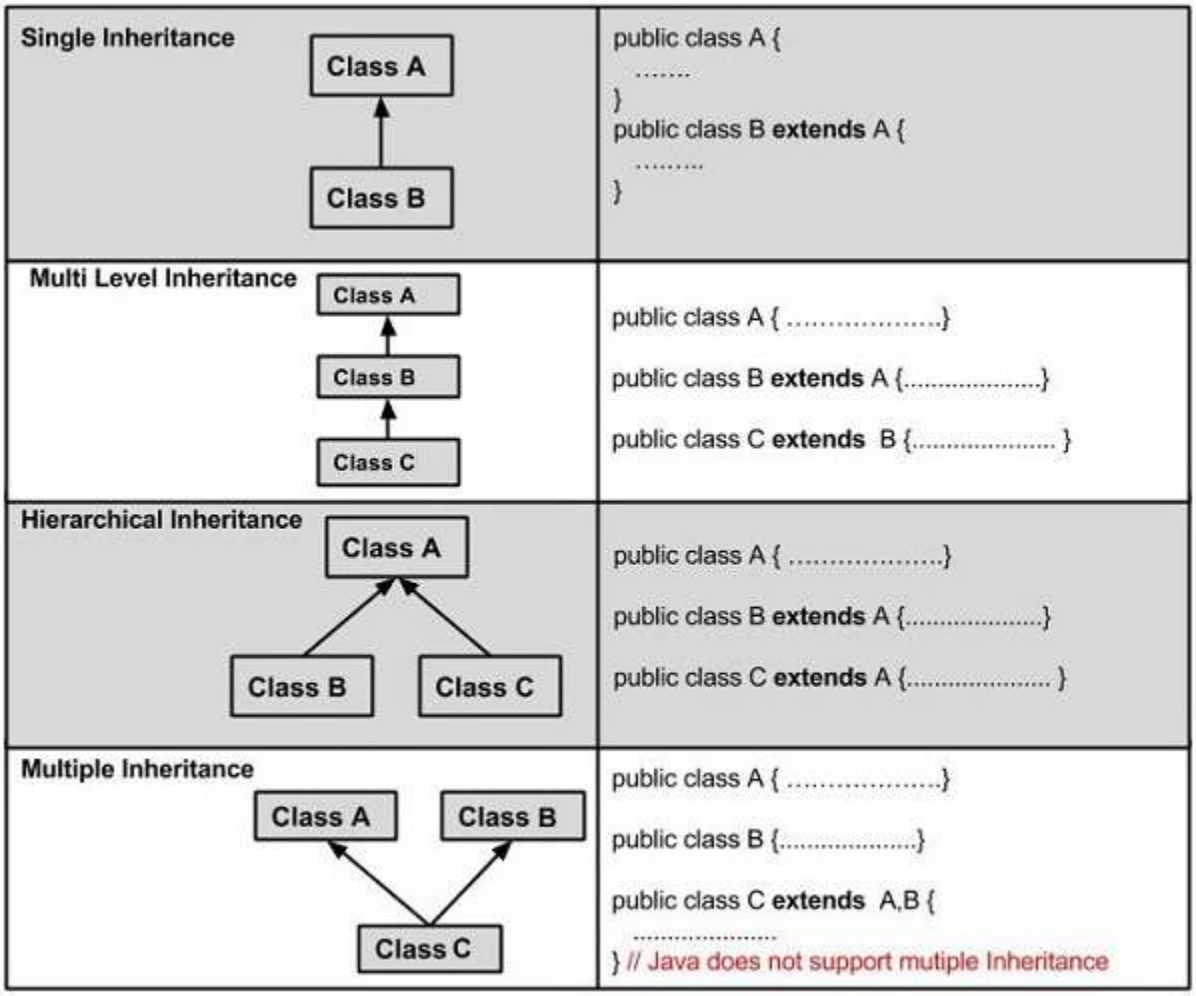
.....

.....

}

Types of Inheritance

There are various types of inheritance as demonstrated below.



Chapter 5

Polymorphism

Constructor Overloading :-

Constructors are special member functions that have the same name as that of class. When two or more constructors in the class have different number of arguments and different types of arguments, then this process is called Constructor Overloading.

Example of a Constructor Overloading is given below:

```
public class Student
{
        public Student()  //Default Constructor
        {
                // This constructor has no parameter.
        }

        public Student(String name)  //Parameterized Constructor
        {
                // This constructor has one parameter, name.
        }
}
```

In the above example, there are two constructors of class Student.

Method Overloading :-

When two or more functions in the class have different number of arguments and different types of arguments, then this process is called Function Overloading or Method Overloading.

Example of a Function Overloading is given below:

```
class Student
{
        public void get()
        {
                System.out.println("Hello");
        }
        public void get(int a)
        {
                System.out.println("Bye");
        }
}

 public class StInfo
 {
          public static void main(String args[])
          {
                  Student st = new Student();
                  st.get();          //It will display Hello
                  st.get(5);          //t will display Bye
          }
 }
```

Method Overriding :-

When a Base Class and its Derived Class have a function with same signature, then the Base Class function can be overridden by calling the Derived Class function using the object of the Derived Class. This concept is called Function Overriding.

Example of a Function Overriding is given below:

```
class Student
{
        public void get()  //Member Function of Base Class
        {
                System.out.println("Hello");
        }
}
class Stud extends Student
{
        public void get()  //Member Function of Derived Class
        {
                System.out.println("Bye");
        }

}

public class StInfo
{
          public static void main(String args[])
          {
                Stud st = new Stud();
                st.get();          //It will display Bye
          }
}
```

Chapter 6
Abstract Class and Interface

Interfaces in Java :-

An interface is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements.

the similarities between Interfaces and Classes :-

An interface is similar to a class in the following ways:

1. An interface can contain any number of methods.
2. An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
3. The bytecode of an interface appears in a **.class** file.

4. Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

the differences between Interfaces and Classes :-

An interface is different to a class in the following ways:

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Declaration of an Interface :-

The **interface** keyword is used to declare an interface. Here is a simple example to declare an interface:

Example:

```
/* File name : NameOfInterface.java */
import java.lang.*;
//Any number of import statements

public interface NameOfInterface
{
        //Any number of final, static fields
        //Any number of abstract method declarations
}

/* File name : Animal.java */
interface Animal
{
        public void eat();
        public void travel();
}
```

The properties of Interfaces :-

Interfaces have the following properties:

1. An interface is implicitly abstract. You do not need to use the **abstract** keyword when declaring an interface.
2. Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
3. Methods in an interface are implicitly public.

Implement Interfaces :-

When a class implements an interface, it agrees to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.

A class uses the **implements** keyword to implement an interface.

Example:-

```
/* File name : MammalInt.java */
public class MammalInt implements Animal
{
        public void eat()
        {
                System.out.println("Mammal eats");
        }
        public void travel()
        {
                System.out.println("Mammal travels");
        }
        public int noOfLegs()
        {
                return 0;
        }
        public static void main(String args[])
        {
                MammalInt m = new MammalInt();
                m.eat();
                m.travel();
        }
}
```

This would produce the following result:

```
Mammal eats
  Mammal travels
```

the interfaces in detail :-

An interface is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements.

Characteristics of an interface:-

- An interface is implicitly abstract. You do not need to use the **abstract** keyword when declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.

Similarities to a class:-

5. An interface can contain any number of methods.
6. An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
7. The bytecode of an interface appears in a **.class** file.

Difference between interface and class:-

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.

- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Chapter 7
Exception Handling

Exception and Exception Handling :-

Exception is an error event that can happen during the execution of a program and disrupts its normal flow. Java provides a robust and object oriented way to handle exception scenarios, known as **Java Exception Handling**.

try…..catch blocks are used to handle exceptions.

try…..catch block with example :-

A method catches an exception using a combination of the **try** and **catch** keywords. A try……catch block is placed around the code that might generate an exception. The syntax for using try…..catch looks like the following:

```
try
{
        //Protected code

}
catch(ExceptionName e1)
{
        //Catch block
}
```

A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block (or blocks) that follow the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block.

the throws/throw Keywords :-

If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.

You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword.

The following method declares that it throws a RemoteException:

```
import java.io.*;
public class className
{
        public void deposit(double amount) throws RemoteException
        {
                // Method implementation
                throw new RemoteException();
        }
       //Remainder of class definition
}
```

the finally Keyword :-

The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred.

A finally block appears at the end of the catch blocks and has the following syntax:

```
try
{
        //Protected code
}
catch(ExceptionType1 e1)
{
        //Catch block
}
catch(ExceptionType2 e2)
{
        //Catch block
}
finally
{
        //The finally block always executes.
}
```

Creating your own Exception Class :-

You can create your own exceptions in Java.

We can define our own Exception class as below:

```
class MyException extends Exception
{
}
```

You just need to extend the Exception class to create your own Exception class.

Exceptions :-

Exception is an error event that can happen during the execution of a program and disrupts its normal flow. Java provides a robust and object oriented way to handle exception scenarios, known as Java Exception Handling**.**

try…..catch blocks are used to handle exceptions.

A method catches an exception using a combination of the **try** and **catch** keywords. A try……catch block is placed around the code that might generate an exception. The syntax for using try…..catch looks like the following:

```
try
{
        //Protected code
}
catch(ExceptionName e1)
{
        //Catch block
}
```

A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block (or blocks) that follow the try is

checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block.

If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.

You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword.

The following method declares that it throws a RemoteException:

```
import java.io.*;
public class className
{
        public void deposit(double amount) throws RemoteException
        {
                // Method implementation
                throw new RemoteException();
        }
      //Remainder of class definition
}
```