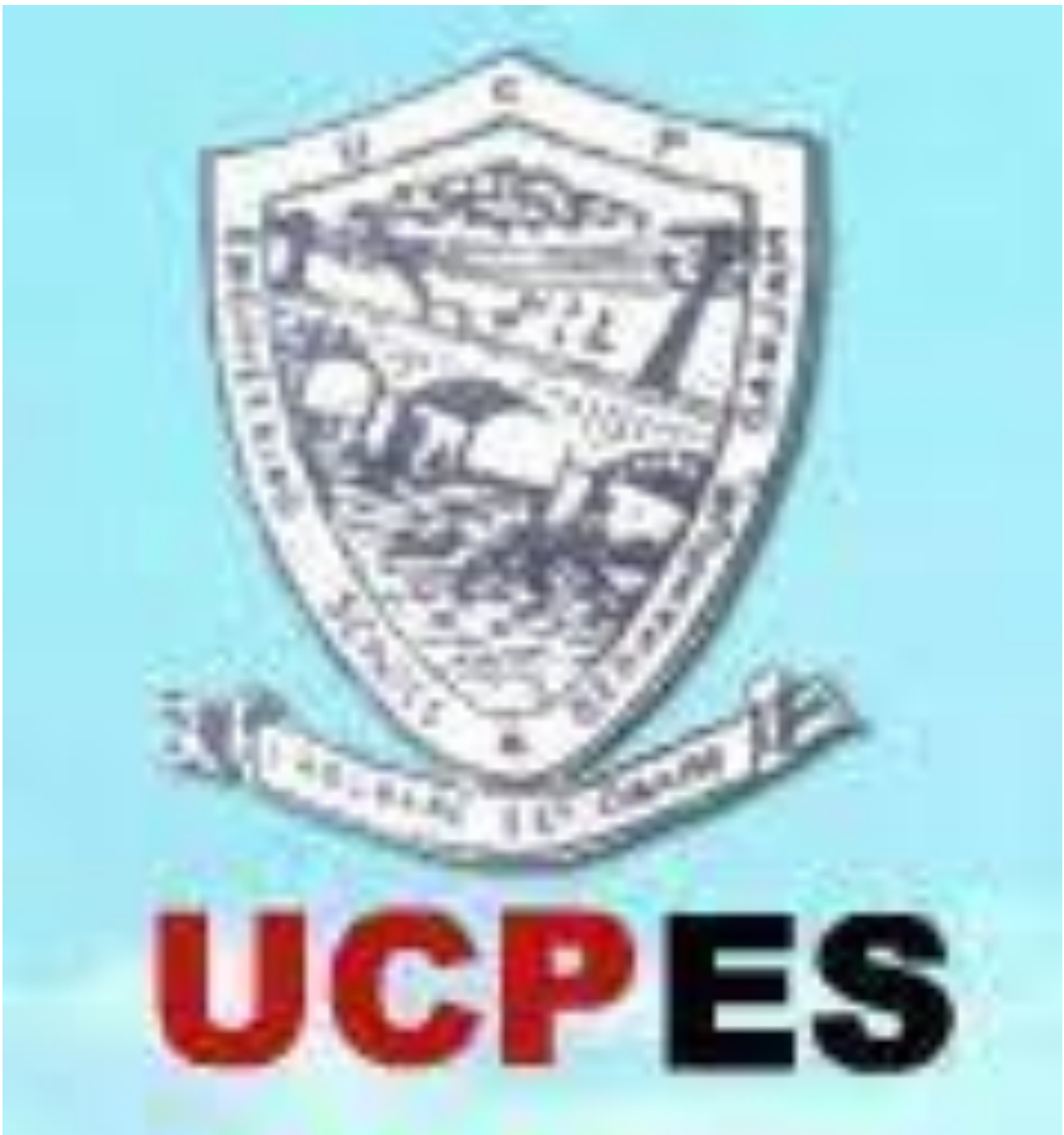
LECTURE NOTES ON OBJECT ORIENTED METHODOLOGY
(TH-4)(COMMON TO CSE/IT)



PREPARED BY

REETANJALI PANDA
LECTURER,UCPES,BERHAMPUR(GM).

CONTENTS

SYLLABUS

1 OBJECT ORIENTED PROGRAMMING (OOPS) CONCEPTS

1.1 Programming Languages
1.2 Object Oriented Programming
1.3 OOPS concepts and terminology
1.4 Benefit of OOPS
1.5 Application of OOPS

2 INTRODUCTION TO JAVA

2.1 What is Java ?
2.2 Execution Model of Java
2.3 The Java Virtual Machine
2.4 A First Java Program
2.5 Variables and Data types
2.6 Primitive Datatypes & Declarations
2.7 Numeric and Character Literals
2.8 String Literals
2.9 Arrays, Non-Primitive Datatypes
2.10 Casting and Type Casting
2.11 Widening and Narrowing Conversions
2.12 Operators and Expressions
2.13 Control Flow Statements

3 OBJECTS AND CLASSES

3.1 Concept and Syntax of class
3.2 Defining a Class
3.3 Concept and Syntax of Methods
3.4 Defining Methods
3.5 Creating an Object
3.6 Accessing Class Members
3.7 Instance Data and Class Data
3.8 Constructors
3.9 Access specifiers
3.10 Access Modifiers
3.11 Access Control

4 USING JAVA OBJECTS

4.1 String Builder and String Buffer
4.2 Methods and Messages
4.3 Parameter Passing
4.4 Comparing and Identifying Objects

5 INHERITANCE

5.1 Inheritance in Java
5.2 Use of Inheritance
5.3 Types of Inheritance
5.4 Single Inheritance
5.5 Multi-level Inheritance
5.6 Hierarchical Inheritance
5.7 Hybrid Inheritance

6 POLYMORPHISM

6.1 Types of Polymorphism
6.2 Method Overloading
6.3 Run time Polymorphism
6.4 Method Overriding

7 PACKAGES: PUTTING CLASSES TOGETHER

7.1 Introduction
7.2 Java API Packages
7.3 Using System Packages
7.4 Naming Convention
7.5 Creating Packages
7.6 Accessing a Package
7.7 Using a Package
7.8 Adding a Class to Package
7.9 Hiding Classes
7.10 Static Import

8 JAVA FILES AND I/O

8.1 What is a stream ?
8.2 Reading and writing to files(only txt files

UNIT-1 OBJECT ORIENTED PROGRAMMING (OOPS) CONCEPTS

1.1 Programming Languages:

A program is a set of instructions that tell a computer what to do. We execute a program to carry out the instruction listed by that program.

Instructions are written using programming languages.

Programming languages are classified as

1. Object Oriented Programming Languages Ex. C++, Java
2. High-Level Languages Ex. C, , FORTRAN, and COBOL.
3. MiddleLevel Language Ex Assembly Language
4. finally the lowest level as the Machine Language.

The lowest level Machine Language consists of binary codes to write the programs which is directly understandable by the machine.

MiddleLevel Language(Assembly Language) consists of mnemonic codes and symbolic names to write down the programs.

High-level programming languages are programming languages using English like language that are rather natural for people to write. It provides

Top Down approach to design a software.

These are of two types:

1. Structured or Procedure oriented : The program is divided in to modules. It is function oriented, provides functional abstraction and data moves freely from one function to another.

2. Unstructured : The most primitive of all programming languages having sequential flow of control. They do not contain any procedures or functions.

Using OOP concept, programs are written in terms of objects rather than functions as we perceive the real world in terms of objects. The basic concept of OOPs is to create objects, re-use them throughout the program, and manipulate these objects to get results. Data & actions are combined together in to a single unit called class. It provides Bottom Up approach to design a software.

1.2 Object Oriented Programming Concepts :

OBJECT ORIENTED PROGRAMMING (OOP) is a programming concept that allows users to write programs in terms of objects as we perceive the real world in terms of objects. It provides Bottom Up approach to design a software, to create the objects that they want and then, create methods to handle those objects, works on the principles of abstraction,encapsulation, inheritance, and polymorphism. The basic concept of OOPs is to create objects, re-use them throughout the program, and manipulate these objects to get results.

1.3 OOPS concepts and terminology

Features of OOPS concepts:

1) Class

The class is a group of similar entities. It is only an logical component and not the physical entity. For Example: Book can be considered as a class having different data to define it, such as name, title , publication, the accession number, cost, borrower, date of issue etc, called as Data members. The different methods it can contain required to process a book can be , open a book, close a book , issue abook ,return a book etc called Member methods.

2)Object

An object can be defined as an instance of a class, and there can be multiple instances of a class in a program. They are the real world entities about which want to record data and manipulate. An Object contains both the data and the function,which operates on the data. For example - chair, bike, student,teacher, pen, table,car, etc.

3) Inheritance(Is- A Relationship)

Inheritance is an OOPS concept in which one object acquires the properties and behaviours of the parent object. It creates a parent-child relationship between two classes. It offers robust and natural mechanism for organizing and structure any software. It provides the

benefit of reusability. The new class is called as the derived class and the existing class is called as the base class. For example: Books can be of two types Text Book or Reference Book. So these two classes are derived from the base class Book, where to the existing features of the base class Book some extra features are added in the derived classes. Similarly classes Automobiles and Pulled Vehicles can be derived from the class Vehicle from which car ,bus, cart andrickshaw classes can be derived subsequently.

Types of Inheritance: There are five types of inheritance.

1. Single Inheritance: A single derived class is derived from a single base class.
2. Multiple Inheritance: A single derived class is derived from more than one base class. This type of inheritance is not found in Java.
3. Multilevel Inheritance: There is multiple levels of inheritance. A single derived class is derived from a single base class from which another class is derived.
4. Hierarchical Inheritance: From a single base class, more than one classes are derived. It forms an inverted tree like structure.
5. Hybrid Inheritance: Combination of more than one type of inheritance.

4) Composition (Has- A Relationship)

The composition is the strong type of association. An association is said to be a composition if an Object owns another object and another object cannot exist without the owner object. Let's take an example of House and rooms. Any house can have several rooms. One room can't become part of two different houses. So, if you delete the house, room will also be deleted.

5) Polymorphism(Poly means many and Morph means form):

Polymorphism refers to the ability of a variable, object or function to take on multiple forms. For example, in English, the verb run has a different meaning if you use it with a laptop, a foot race, and business, they are having different meaning according to the context they are used. Similarly we can write an add method in a class and use it to attain polymorphism for adding two integers, two floating point numbers or concatenate two strings etc. Polymorphism is of two types: Compile

time polymorphism and Run time polymorphism based on the decision of invoking a function made at compile time or run time respectively.

6) Data Abstraction

An abstraction is an act of representing essential features without including background details. It is a technique of creating a new data type that is suited for a specific application. For example, while driving a car, you do not have to be concerned with its internal working. Here you just need to concern about the use of the parts like steering wheel, Gears, accelerator, etc. The main purpose of abstraction is hiding the unnecessary details from the users. Abstraction is selecting data from a larger pool to show only relevant details of the object to the user. It helps in reducing programming complexity and efforts. However, the same information once extracted can be used for a wide range of applications. For instance, you can use the same data of customers for a bank application , for hospital application, job portal application, a Government database, etc. with little or no modification. Hence, it becomes your Master Data. This is an advantage of Abstraction.

7) Encapsulation

Encapsulation is an OOP technique of wrapping the data members and member functions into a single unit called class. In this OOPS concept, the data members of a class are always hidden from other classes. It can only be accessed using the methods of their current class. It is the process of hiding information details and protecting data and behaviour of the object from the outside world . In

Function oriented programming , the global data was shared among the functions where as in Object Oriented Programming data are distributed among the objects and are localised inside a class. The objects can communicate using member functions in the class.

8) Data Hiding :

Data Hiding is hiding the variables of a class from other classes. It can only be accessed through the method of their current class. It hides the implementation details from the users. But more than data hiding, it is meant for better management or grouping bettermanagement of related data.

9) Message Passing:

Objects can communicate among themselves by invoking the member functions through a dot (.) operator as follows:
object name . Member method()

1.4 Benefit of OOPS

- OOP offers easy to understand and a clear modular structure for programs.
- Objects created for Object-Oriented Programs can be reused in other programs. Thus it saves significant development cost.
- Large programs are difficult to write, but if the development and designing team follow OOPS concept then they can better design with minimum flaws.
- It also enhances program modularity because every object exists independently.
- The data is more secured.
- Provides easy maintainability and High productivity.

1.5 Application of OOPS

- Real time Systems
- Simulation and Modeling
- Object Oriented Database
- Hypertext,Hypermedia
- AI and Expert Systems
- Neural Network and Parallel Programming
- Decision Support System
- Office Automation System
- CAD/CAM System

UNIT – 2 INTRODUCTION TO JAVA

2.1 What is Java ?

Java is one of the world's most important and widely used computer languages, and it hasheld this distinction for many years. Unlike some other computer languages whose influence has weared with passage of time, while Java's has grown.

As of 2020, Java is one of the most popular programming languages in use, particularlyfor client-server web applications, with a reported 9 million developers using and working on it.

2.2 Execution Model of Java

Unlike other programming languages, compiling Java source code does not result in a machine language program. Instead, when Java source code is compiled, we get what is called Java bytecode.

Java bytecode is a form of machine language instructions.However, it is not primitive to the CPU. Java bytecode runs on a program that mimics itself as a real machine. This program is called the Java Virtual Machine (JVM) or Java Run-time Environment (JRE). This architecture makes Java bytecode runs on any machines that have JVM, independent of the OSs and CPUs. This means the effort in writing Java source code for a certain program is spent once and the target program can run on any platforms. (E.g.Windows, MacOS, Unix, etc.)

2.3 The Java Virtual Machine

I. The source code written in java is saved as .java file.

II. Using the java compiler the code is converted into an intermediate code called the bytecode. The output is a .class file.

III. This code is not understood by any platform, but only a virtual platform called the Java Virtual Machine.

IV. This Virtual Machine resides in the RAM of your operating system. When the Virtual Machine is fed with this byte code, it identifies the platform it is working on and converts the byte code into the native machine code.

What is JVM?

JVM is a run time environment which acts as an interpreter and translates the byte code into object code (machine language). It is a platform-independent execution environment that converts Java byte code into object code and executes it. Java Virtual Machine (JVM) is a engine that provides runtime environment to drive the Java Code or applications.JVM is a part of Java Runtime Environment (JRE). In other programming languages, the compiler produces machine code for a particular system.However, Java compiler produces code for a Virtual Machine known as Java Virtual Machine.

How JVM works?

Java code is complied into bytecode. This bytecode getsinterpreted on different machines between host system and Java source, Bytecode is an intermediary language.JVM is responsible for allocating memory space.

Source Code

The core program or text written in any computer language (like C, C++, Java, etc.) is called source code.Usually it is a collection

of computer instructions writtenby using any human readable computer language. Source code files have the extension class.

Object Code

The program in the form of machine instructions or binary instructions (i.e., in computer readable form). It is generally produced by compiler/interpreter, but in case of Java, it is produced by JVM.

Byte Code

In Java, when a source code is compiled, it doesn‘t directly convert into object code, rather it converts into what, isknown as byte code. So, a byte code is machine instruction that the Java compiler (Javac) generates. Java code is written in .java files (also known as source file),which is compiled by javac, a Java compiler into .class files.Unlike C or C++ compiler, Java compiler doesn't generate native code. These class files contain byte code, which is different than machine or native code.

An interpreter is a computer program, which coverts each high-level program statement into the machine code. A compiler will convert the code into machine code before program run while Interpreters convert code into machine code when the program is run. In Java,To improve performance, JIT compilers interact with the Java Virtual Machine (JVM) at run time and compile suitable bytecode sequences into native machine code. The JIT compiler is able to perform certain simple optimizations while compiling a series of bytecode to native machine language.The Just-In-Time (JIT) compiler is a an essential part of the JRE.

JRE : The Java Runtime Environment (JRE) provides the libraries, the Java Virtual Machine, and other components to run applets and applications written in the Java programming language. JRE does not contain tools and utilities such as compilers or debuggers for developing applets and applications.

JDK : The JDK also called Java Development Kit is a superset of the JRE, and contains everything that is in the JRE, plus tools such as the compilers and debuggers necessary for developing applets and applications.

2.4 A First Java Program

In order to start with basic basic Java programming, let us look at the standard
Hello World program.

```
public class MyFirstJavaProgram
{
public static void main(String []args)
 {
System.out.println("Say Hello World To The
World!");
}
}
```

Open any text editor and write down the above code in that file. Save the file with a .java extension. For example, you can save the file as MyFirstJavaProgram.java. Name of the file and name of the class containing main()method(Driver method) must be the same. The next step is to to open the command prompt of the system and relocate its reference to the directory in which the file is saved. For instance, if you have saved the file in C:\, then you must take the prompt to the same directory. In order to compile the code, you must type the following:

```
javac MyFirstJavaProgram.java
```

If there are no errors, you will automatically be taken to the next line. Successful Compilation leads to the creation of the class file.You can now execute the code using the following command:

```
java MyFirstJavaProgram
```

This is the class file which is mentioned here without any extension name. Now You should be able to see the following output on the screen.

Say Hello World To The World!

Structure of a java program

Structure of a java program is the standard format released by Language developer to the Industry programmer. Sun Micro System has prescribed the following structure for the java programmers for developing java application

A package is a collection of classes, interfaces and subpackages. A sub package contains collection of classes,

interfaces and sub-sub packages etc. java.lang.*; package is imported by default and this package is known as default package.

class is the keyword used for developing user defined data type and every java program must start with a concept of class.

ClassName represents a java valid variable name treated as a name of the class each and every class name in java is treated as user-defined data type.

Data member represents either instance or static they will be selected based on the name of the class.

User-defined methods represents either instance or static they are meant for performing the operations either once or each and every time.

Each and every java program starts execution from the main() Method and hence main() method is known as program driver. Since

main() method of java is not returning any value and hence its return type must be void.Since main() method of java executes only once throughout the java program execution and hence its nature must be static. Since main() method must be accessed by

every java programmer and hence whose access specifier must be public.Each and every main() method of java must take array of objects of String.Block of statements represents set of executable statements which are in term calling user-defined methods are containing business-logic.

The file naming conversion in the java programming is that which-ever class is containing main() method, that class name must be given as a file name with an extension .java. main ( ) method is starting execution block of a java program or any java program start their execution from main method. If any class contain main( ) method known as main class.

```
public static void main(String args[])
{
……..;
……..;
}
```

public

public is a keyword in a java language whenever if it is preceded by main() method the scope is available anywhere in the java environment that means main() method can be executed from anywhere. main() method must be accessed by every java programmer and hence its access specifier must be public.

static

static is a keyword in java if it is preceded by any class properties for that memory is allocated only once in the program. static method are executed only once in the program. main() method of java executes only once throughout the java program execution and hence it declare must be static.

void

void is a special datatype also known as no return type, whenever it is preceded by main() method that will be never return any value to the operating system. main() method of java

is not returning any value and hence its return type must be void.

String args[]

String args[] is a string array used to hold command line arguments in the form of String values.

2.5 Variables and Data types

Data types specify size and the type of values
that can be stored in an identifier. In java, data types are classified into two catagories :

- Primitive Data Types :- These are also known as standard data type or built in data type. The java compiler contains detailed instructions on each legal operations supported by the data type. They include integer, character, boolean, and float etc.
- Non-primitive Data Types :- These are also known as derived data types or reference datatypes which are built on primitive datatypes. They include classes, arrays and interfaces.

Variables in Java: Variables are symbolic names of memory locations. They are used for storing values used in programs. Every variable is assigned data type which designates the type and quantity of value it can hold. In many programming languages including Java, before a variable can be used, it has to be declared so that its name is known and proper space in memory is allocated.In order to use a variable in a program you to need to perform two steps

- Variable Declaration
- Variable Initialization

Variable Declaration

To declare a variable, you must specify the data type & give the variable a unique name.For Example:

int x; double y;

Here a variable x is created for storing an int(integer) value and a variable y is created for storing a double (double- precision floating point) value.

Variable Initialization

To initialize a variable, you must assign it a valid value. Variables are normally used with the assignment operator (=), which assign the value on the right to the variable on the left. Example of other Valid Initializations are

pi =3.14f; d =20.22d; a='v'; You can combine variable declaration and initialization as follows:

Example :

int a=2,b=4,c=6; float pi=3.14f; double d=20.22d; char a='v';

Naming Rules and Styles:

There are certain rules for the naming of Java identifiers. Valid Java identifier must be consistent with the following rules.

An identifier cannot be a Java reserve word.

◦ An identifier must begin with an alphabetic letter, underscore (_), or a dollar sign ($).

◦ If there are any characters subsequent to the first one, those characters must be alphabetic letters, digits, underscores (_), or dollar signs ($).

◦ Whitespace cannot be used in a valid identifier.

◦ An identifier name must be unique.

◦ An identifier must not be longer than 65,535 characters.

◦ Java is case sensitive , so upper case and lower case letters are distinct.

Also, there are certain styles that programmers widely use in naming variables, classes and methods in Java. Here are some

of them.
◦ Use lowercase letter for the first character of variables' and methods' names.
◦ Use uppercase letter for the first character of class names.
◦ Use meaningful names.
◦ Compound words or short phrases are fine, but use uppercase letter for the first character of the words subsequent to the first. Do not use underscore to separate words.
◦ Use uppercase letter for all characters in a constant. Use underscore to separate words.
◦ Apart from the mentioned cases, always start with a lowercase letter.
◦ Use verbs for methods' names followed by nouns.
Here are some examples for good Java identifiers.

◦ Variables: height, speed, filename,tempInCelcius, incomingMsg, textToShow.
◦ Constant: SOUND_SPEED, KM_PER_MILE, BLOCK_SIZE.
◦ Class names: Account, DictionaryItem, FileUtility,Article.
◦ Method names: locate, sortItem,findMinValue, checkForError.
Invalid variables: 47123, #phone, basic pay, if

2.6 Primitive Datatypes & Declarations

Primitive Data Types

Primitive Data Types are predefined and available within the Java language. Primitive values do not share state with other primitive values.There are 8 primitive types: byte, short, int, long, char, float,double, and boolean.

Integer data types

This group includes byte, short, int, long datatypes

- byte : It is 8 bit integer data type. Value range from -128 to 127. Default value is zero. example: byteb=10;
- short : It is 16 bit integer data type. Value range from-32768 to 32767. Default value is zero.example: short s=11;
- int : It is 32 bit integer data type. Value range from-2147483648 to 2147483647. Default value is zero.example: int i=10;
- long : It is 64 bit integer data type. Value range from-9,223,372,036,854,775,808to9,223,372,036,854,775,807. Default value zero.example: long l=100012;

Floating-Point Number

This group includes float, double datatypes.

- float : It is 32 bit float data type. Default value 0.0f.example: float ff=10.3f;
- double : It is 64 bit float data type. Default value 0.0d.example: double db=11.123;

Characters

This group represent char, which represent symbols in a character set, like letters and numbers.

char : It is 16 bit unsigned unicode character. Range 0 to 65,535. example: char c='a';

Boolean

This group represent boolean, which is a special type for representing true/false values. They are defined constant of the language. example: boolean b=true;

2.7 Numeric and Character Literals

Literals are those data items whose value does not change during the programexecution. They are also known as constants. Java supports different types of literals which are

◦ Integer literal
◦ Floating-point literal
◦ Boolean literal
◦ Character literal
◦ String literal
◦ **Integer literal**

These are the primary literals used in Java. They are of three types-

1. decimal (base 10)
2. hexadecimal (base 16)
3. octal (base 8

(i)Decimal Interger Literals- Whose digits consists of the numbers 0 to 9.

(ii)Hexadecimal Interger Literals-Whose digits consists of the numbers 0 to 9 and letters A to F.

(iii) Octal Integer Literals- Whose digits consists of the numbers 0 to 7 only.

Some rules for integer literals are given below

◦ It must have at least one digit and can‘t use a decimal digit.

◦ It must have a positive or negative sign, if the number does appear without any sign, it is assumed to be a positive number.

◦ Hexadecimal literals appear with a leading Ox (zero, x).

Octal literals appear with a leading 0 (zero) in front of its digits. While decimal literals appears as ordinary numbers with no special notation.

For example, an decimal literal for the number 10 is represented as 10 in decimal, OxA in hexadecimal and 012 in octal.These literals represent a single Unicode character and appear within a pair of single quotation marks. Like : ‘a’, ‘x’ etc.

Floating-point literal

Floating-point numbers are like real numbers in mathematics. For example, 4.13179, -0.0001. Java has two kinds of floating-point number: float and double. The default type when you write a floating point literal is double. Float is of 32 bits, where as double is of 64 bits. A floating-point literal can be either of two data types float or double type. Floating point constants default to double precision. We have to add a suffix to the floating point literal as D, d, F or f (D or d for double and F or f for float). There are two ways of representing floating point constants.

1. Standard Decimal Notation: It consists of a whole number followed by a decimal point and fractional component.

Example: 0.375, 2.576

2. Scientific notation or Exponent form: Syntax: Mantissa E Exponent. It consists of two parts: Mantissa part which can either be decimal or fractional and exponent part which is always a whole number represented by E or e .

Example : 0.173 E +123 , 341 e -7

Boolean literal

True represents a true value and false represents a false value.

Example: boolean flag;flag= false;

Literals true or false should not be represented by the quotation marks around it. Java compiler will take it as a string of characters, if it is represented in quotation marks.

Character literal

There are some character literals which are not readily printable through a keyboard such as backspace, tabs, etc. These type of characters are represented by using escape sequences (\).

2.8 String Literals

It is a sequence of characters between a pairof double quotes. The characters may be alphabets,digits,special characters or blank space.
Example: “1937” , “ welcome”, “Berhampur”

2.9 Arrays, Non-Primitive Datatypes

An array is a collection of similar data types. Array is a container object that hold values of homogenous type in contiguous memory location. It is also known as static data structure because size of an array must be specified at the time of its declaration. An array is of reference type. It gets memory in heap area. Index of array starts from zero to size-1.

Array Declaration

Syntax :
datatype[ ] identifier;or datatype identifier[ ];
Both are valid syntax for array declaration.
Example :
int[ ] arr;char[ ] arr;short[ ] arr;long[ ] arr;int[ ][ ] arr; // two dimensional array.

Initialization of Array

new operator is used to allocate memory to an array dynamically.
Example :

```
int[ ] arr = new int[10]; //10 is the size of array or
int[ ] arr = {10,20,30,40,50};
```

Accessing array element

As mentioned ealier array index starts from 0. To access nth element of an array. Syntax

arrayname[n-1];
Example : To access 4th element of a given array
int[ ] arr = {10,20,30,40};
System.out.println("Element at 4th place" + arr[3]);
The above code will print the 4th element of array arr on console.

```
class Test
{
public static void main(String[] args)
{
int[] arr = {10, 20, 30, 40};
for(int i=0;i<4;i++)
{
System.out.println(arr[i]);
}
}
}
```

Output :

10
20
30
40

Multidimensional arrays

Multidimensional arrays are implemented as arrays of arrays. Multidimensional arrays are declared by appending the appropriate number of bracket pairs after the array name. For example,

 // integer array 512 x 128 elements
int[][] twoD = new int[512][128];

 // character array 8 x 16 x 24

```
char[][][] threeD = new char[8][16][24];
```
 // String array 4 rows x 2 columns

```
String[][] dogs =
{
{ "terry", "brown" },
{ "Kristin", "white" },
{ "toby", "gray"},
{ "fido", "black"}
};
```

To access an element in a multidimensional array is just the same as accessing the elements in a one dimensional array. For example, to access the first element in the first row of the array dogs, we write,

```
System.out.print( dogs[0][0] );
```

This will print the String "terry" on the screen.

```
public class ArrayDemo
{
public static void main(String[] args)
{
String[][] dogs =
{
{ "terry", "brown" },
{ "Kristin", "white" },
{ "toby", "gray"},
{ "fido", "black"}
};
```

```
for(int i=0;i<4;i++)
{
for(int j=0;j<2;j++)
{
System.out.print( dogs[i][j]+"\t" );
}
System.out.println();
}
}
}
```

The output is as follows:

```
terry       brown
kristin     white
toby        gray
fido        black
```

2.10 Casting and Type Casting

Type Casting

Assigning a value of one type to a variable of another type is known as Type Casting.

Example :

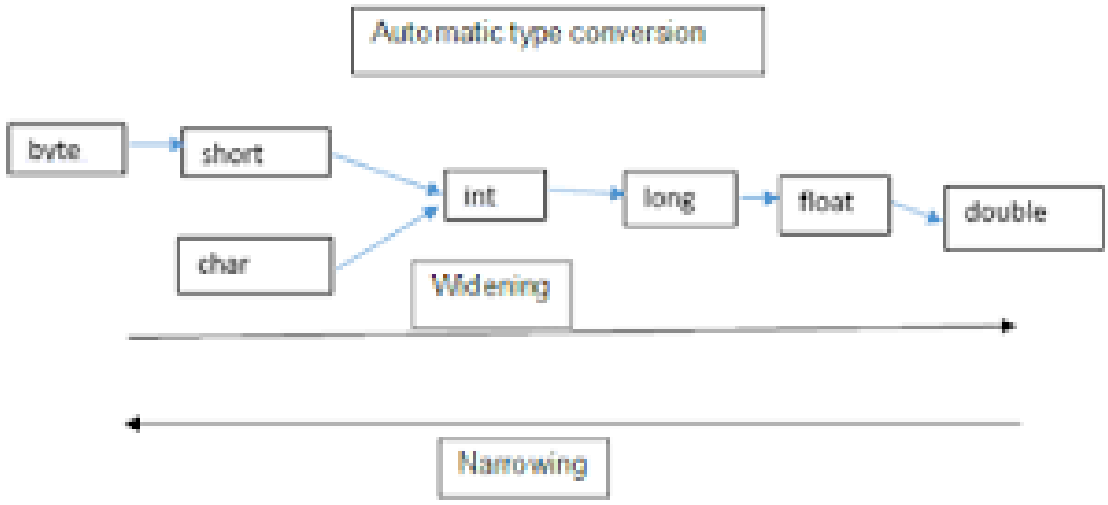
```
int x = 10;
byte y = (byte)x;
```

In Java, type casting is classified into two types,

Widening Casting(Implicit)

Narrowing Casting(Explicitly done)

2.11 Widening and Narrowing Conversions



Widening or Automatic type conversion: Automatic Type casting take place when the two types are compatible. The target type is larger than the source type.

Example :

```
public class Test
{
public static void main(String[] args)
{
int i = 100;
long l = i; //no explicit type casting required
float f = l; //no explicit type casting required
System.out.println("Int value "+i);
System.out.println("Long value "+l);
System.out.println("Float value "+f);
}
```

```
}
```
Output :
```
Int value 100
Long value 100
Float value 100.0
```

Narrowing or Explicit type conversion: When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.

Example :

```
public class Test
{
public static void main(String[] args)
{
double d = 100.04;
long l = (long)d; //explicit type casting required
int i = (int)l; //explicit type casting required
System.out.println("Double value "+d);
System.out.println("Long value "+l);
System.out.println("Int value "+i);
}}
```

Output :

```
Double value 100.04
Long value 100
Int value 100
```

Example :

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

```
byte x;
int a = 270;
double b = 128.128;
System.out.println("int converted to byte");
x = (byte) a;
System.out.println("a and x " + a + " " + x);
System.out.println("double converted to int");
a = (int) b;
System.out.println("b and a " + b + " " + a);
System.out.println("double converted to byte");
x = (byte)b;
System.out.println("b and x " + b + " " + x);
}
}
```

Output:

```
int converted to byte
a and x 270 14
double converted to int
b and a 128.128 128
double converted to byte
b and x 128.128 -128
```

2.12 Operators and Expressions

Java provides a rich operator environment. Most of its operators can be divided into the following categories:

Increment/Decrement Operator

- Arithmetic Operator
- Relational Operator
- Logical Operator
- Bitwise Operator

- Shift Operator
- Assignment Operator
- Conditional Operator
- Instance of Operator
- New Operator
- Member selection Operator

Each operator performs a specific task it is designed for.

Increment and Decrement operators

Aside from the basic arithmetic operators, Java also includes a unary increment operator(++) and unary decrement operator (--). Increment and decrement operators increase and decrease a value stored in a number variable by 1. For example, the expression,

count = count + 1; //increment the value of
count by 1
is equivalent to
count++;

When used before an operand, it causes the variable to be incremented or decremented by 1, and then the new value is used in the expression in which it appears. For example,

int i = 10, int j = 3; int k = 0;
k = ++j + i; //will result to k = 4+10 = 14

When the increment and decrement operators are placed after the operand, the old value of the variable will be used in the expression where it appears. For example,

int i = 10, int j = 3; int k = 0;
k = j++ + i; //will result to k = 3+10 = 13

```
class OperatorExample
{
public static void main(String args[])
{
int x=10;
```

```
System.out.println(x++);
System.out.println(++x);
System.out.println(x--);
System.out.println(--x);
}
}
```

Output:

```
10
12
12
10
class OperatorExample{
public static void main(String args[])
{
int a=10;
int b=10;
System.out.println(a++ + ++a);//10+12=22
System.out.println(b++ + b++);//10+11=21
}}
```

Output:

```
22
21
```

Arithmetic Operators:

Arithmetic Operator	Use	Description
op1 + op2		Adds op1 and op2
op1 * op2		Multiplies op1 by op2
op1 / op2		Divides op1 by op2
op1 % op2		Computes the remainder of dividing op1 by op2
op1 - op2		Subtracts op2 from op1

Relational operators:

Relational operators compare two values and determines the relationship between those values. The output of evaluation are the boolean values true or false.

op1 > op2	op1 is greater than op2
op1 >= op2	op1 is greater than or equal to op2
op1 < op2	op1 is less than op2
op1 <= op2	op1 is less than or equal to op2
op1 == op2	op1 and op2 are equal
op1 != op2	op1 and op2 are not equal

Logical operators

Logical operators have one or two boolean operands that yield a boolean result. There are three logical operators: && (logical AND), || (logical OR),, and ! (logical NOT) while the four bitwise operators are & (AND), | (inclusive OR), ^ (exclusive OR) and ~(NOT)can be used to integer types like long,int,short,char and byte as its operands. It can also be used with assignment form such as &= ,|= , ^= etc.

The basic expression for a logical operation is,

x1 op x2

where x1, x2 are the operands, and op is the operator.

Given an expression,

exp1 && exp2

&& will evaluate the expression exp1, and immediately return a false value if exp1 is false. If exp1 is false, the operator never evaluates exp2 because the result of the operator will be false regardless of the value of exp2. In contrast,

the & operator always evaluates both exp1 and

exp2 before returning an answer.

Given an expression,

exp1 || exp2

|| will evaluate the expression exp1, and immediately return a true value is exp1 is true. If exp1 is true, the operator never evaluates exp2 because the result of the operator will be true regardless of the value of exp2. In contrast, the | operator always evaluatesboth exp1 and exp2 before returning an answer.

The result of an exclusive OR operation is TRUE, if and only if one operand is true and the other is false. Note that both operands must always be evaluated in order to calculate the result of an exclusive OR.

The ! logical NOT/~ Bitwise NOT takes in one argument, wherein that argument can be an expression, variable or constant.

The bitwise shift operators shifts the bit value. The left operand specifies the value to be shifted and the right operand specifies the number of positions that the bits in the value are to be shifted. Both operands have the same precedence.

Different types of shift operators

1.Left shift(<<) op1<<op2 shifts bits at op1 left by distance of op2

2.Right shift(>>) op1>>op2 shifts bits at op1 right by distance of op2

3. 1.Right shift with zero fill or unsigned right shift (>>>)

op1>>>op2 shifts bits at op1 right by distance of op2 (unsigned or zero fill)

Example

a = 0001000

b= 2

a << b= 0100000 (8<<2 = 8*2^2=32)

a >> b= 0000010 (8>>2 = 8/2^2=2)

Assignment Operator

= assigns values from right side operands to left side operand

a=b

+= adds right operand to the left operand and assign the result to left

a+=b is same as a=a+b

-= subtracts right operand from the left operand and assign the result to left operand

a-=b is same as a=a-b

*= mutiply left operand with the right operand and assign the result to left operand

a*=b is same as a=a*b

/= divides left operand with the right operand and assign the result to left operand

a/=b is same as a=a/b

%= calculate modulus using two operands and assign the result to left operand

a%=b is same as a=a%b

The conditional operator ?:

The conditional operator ?: is a ternary operator. This means that it takes in three arguments that together form a conditional expression. The structure of an expression using a conditional operator is,

exp1?exp2:exp3

wherein exp1 is a boolean expression whose result must either be true or false. If exp1 is true, exp2 is the value returned. If it is false, then exp3 is returned.

instance of Operator:

Only object reference variables can be used with this operator. The objective of this operator is to check if an object is an instance of an exiting class or interface. The return type is boolean. If object is of the specified type ,then the instance of operator returns true otherwise it returns false. It is also known as runtime operator.

Syntax:
(<object >) instance of(<interface/class>)
Example: rose instance of flower is true if the object rose belongs to the class flower otherwise false.

new() operator:

we use the new() operator to allocate memory dynamically at run time.

Case-1: Declare the object and then allocate memory
Syntax:
Classname objectname;
Objectname= new Classname(); //memory allocation
Example:
Flower rose;
rose=new Flower();
Case-2: Declare the object and allocate memory in a single step
Syntax:
Classname objectname= new Classname(); //memory allocation
Example:
Flower rose =new Flower();
Java has automatic garbage collector , so, unlike C++., there is no delete operator in java to dynamically deallocate the memory .

member selection operator or dot operator:

The class consists of data members and member methods. It can be accessed through a member selection operator or dot operator.
Syntax
object.data member
object.member method
Example
Student s1;

s1.rollno;
s1.total();

Unary Operators

These are the operatorswhich work on single operands.

For example, !, -, ++, —, ( ), (cast) operator,Unary + and Unary – are some examples of unary operators.

Binary Operators :These are the mostly usedoperators. These operators work on two operands. Binary operators include arithmetic operators (+, *, /, % etc.).

Ternary Operators: Operator that works on three operands is known as ternary operator. Conditional operator (? :) is the example of ternary operator.

Java expressions:

An expression is a construct made up of variables, operators, and method invocations,which are constructed according to the syntax of the language, that evaluates to a single value.

int x = 3 + 2 * 6;

This expression is evaluated to 30 if the addition operator is executed first. However,it is 15 if the multiplication operator is executed first.

In fact, Java compiler has no problem with such ambiguity. Order of the operators can be determined using precedence and association rules. Each operator is assigned a precedence level. Operators with higher precedence levels are executed before ones with lower precedence levels. Associativity is also assigned to operators with the same precedence level. It indicates whether operators to the left or to the right are to be executed first, in the case of equal precedence levels. Expressions in parentheses () are executed first. In the case of nested parentheses, the expression in the innermost pair is executed

first. the order of operations of all operators according to the precedence and associativity rule.

Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (*type*)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Example-1 : 4*2+20/4

There are three operators in the above expression. They are *, + and /.The precedence values of * and / are both 12, while the precedence value of + is just 11. Therefore, * and / must be operated prior to +. Since * and / have the same precedence value, we need to look at their associativity which we can see that the one on the left have to be performed first.Therefore,the order of operation from the first to the last is *, / and then+. Consequently, the evaluation of the expression value can take place in the steps and the resulting value is 13.

4 * 2 + 20 / 4

=8 + 20 / 4

=8 + 5

=13

Example-2 : Evaluate the following expression.

2+2==6-2+0

Considering the precedence values of the four operators appearing in the expression, which are +(the leftmost one), ==, -, and + (the rightmost one),we can see that +, and - have the same precedence value of 11 (additive operators) which is higher than the one of ==. Among the three additive operators, we perform the operation from the left to the right according to their associativity. The resulting value of this expression can be evaluated to true.

2 + 2 == 6 - 2 + 0

4 == 6 - 2 + 0

4 == 4 + 0

4 == 4

True

Example-3 :

-9.0+5.0*3.0–1.0/0.5 >= 5.0%2.0&&6.0+3.0-
9.0==0

By considering the precedence values of all operators appearing in the expression above, we can place parentheses into the expression in order to explicitly determine the order of operation and then evaluate the values of each part.

((((-9.0)+(5.0*3.0))–(1.0/0.5)) >=
(5.0%2.0))&&(((6.0+3.0)-9.0)==0)
((((-9.0)+15.0)- 2.0) >= 1.0)&&((9.0 -9.0)==0)
((6.0- 2.0) >= 1.0)&&((9.0 -9.0)==0)
(4.0 >= 1.0)&&(0 ==0)
(true)&&(true)
true.

Example-4 : Given an expression, re-write the expression and place parentheses based on operator precedence

6%2*5+4/2+88-10

Answer:

((((6%2)*5)+(4/2))+88)-10;

2.13 Control Flow Statements

A control structure determines an order in which the statements used in a program are executed. A Java program contains classes and these classes have some methods which contains statements that are executed by the compiler. A control structure might cause a statement to be executed once, many times, or not at all. The following control statements are

1. Decision Structure
2. Looping Structure

DECISION STRUCTURE

Decision control structures are Java statements that allows us to select and execute specific blocks of code while skipping other sections.

Decision making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.Decision structures are also known as conditional statements or selection statements. It allows a user to choose any specified condition/statement from a slot of statements. Java programming language provides following types of decision making statements.

1.If structure
2. If……. else structure
3.If…..else if ….. structure
4. Switch structure

IF STRUCTURE

It is the most basic form of selection statement which executes a single block of statements if the specified condition evaluates to true, otherwise the given set of actions/ statements are ignored. The if-statement specifies that a statement (or block of code) will be executed if and only if a certain boolean statement specified in the expression is true.

Syntax:
The syntax of an if statement is:

```
If (expression)
{
//Statements will execute if the expression is true
}
```

If the expression evaluates to true then the block of code inside the if statement will be executed otherwise skip the statement.
For Example:

```
int grade = 68;
if( grade > 60 )
{
System.out.println("Congratulations!");
System.out.println("You passed!");
}
```

IF -ELSE STATEMENT

The if-else statement is used when we want to execute a certain statement if a condition is true, and a different statement if the condition is false.It is the type of selection statement in which a single statement or a group of statements enclosed within the if condition is executed when if condition is true. Otherwise the else block is executed.
Syntax

```
if(expression)
{
//Executes when the expression is true
}
else
{
```

```
//Executes when the expression is false
}
```
If the expression evaluates to true, then the if block of code will be executed, otherwise else block of code will be executed.

For Example:

```
int grade = 68;
if( grade > 60 )
{
System.out.println("Congratulations!");
System.out.println("You passed!");
}
else
{
System.out.println("Sorry you failed");
}
```

The statement in the else-clause of an if-else block can be another if-else structures. This cascading of structures allows us to make more complex selections.

Syntax:

The if-else if statement has the following form,

```
if( boolean_expression1 ) statement1;
else if( boolean_expression2 ) statement2;
else
statement3;
```

if boolean_expression1 is true, then the program executes statement1 and skips the other statements. If boolean_expression2 is true, then the program executes statement 2 and skips to the statements following statement3 otherwise statement3 is executed.

For example:

```
int grade = 68;
if( grade > 90 )
{
System.out.println("Very good!");
}
else if( grade > 60 )
{
System.out.println("Very good!");
}
else
{
System.out.println("Sorry you failed");
}
```

1. The condition inside the if-statement does not evaluate to a boolean value. For example,

```
//WRONG
int number = 0; if( number ){
//some statements here
}
```

The variable number does not hold a Boolean value.

2.Using = instead of == for comparison. For example,

```
//WRONG
int number = 0;
if( number = 0 )
{
//some statements here
}
```

This should be written as,

```
//CORRECT
int number = 0;
if( number == 0 )
{
//some statements here
}
```

3.Writing elseif instead of else if.

4. Proper indent of the curly braces should be maintained otherwise there is a chance of missing the brackets.The number of opening brackets must be same as the number of closing brackets.

```
public class Grade
{
public static void main( String[] args )
{
double grade = 92.0;
if( grade >= 90 )
{
System.out.println( "Excellent!" );
}
else if( (grade < 90) && (grade >= 80))
{
System.out.println("Good job!" );
}

else if( (grade < 80) && (grade >= 60))
{
System.out.println("Study harder!" );
}
else
{
```

```
System.out.println("Sorry, you failed.");
}
}
}
```

The switch construct allows branching on multiple outcomes. A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax:

```
switch(expression)
{ case value1 :
//Statements;
break; //optional
case value2 :
//Statements;
break; //optional
……..
……
……

case value N :
//Statements N; break; //optional
//You can have any number of case statements.
default : //Optional
//Statements;
}
```

where, switch_expression is an integer or character expression and, case value1, case value2 and so on, are unique integer or character constants.

When a switch is encountered, Java first evaluates the switch_expression, and jumps to the case whose selector matches the value of the expression. The program executes the statements in order from that point on until a break statement is encountered, skipping then to the first statement after the end of the switch structure.

If none of the cases are satisfied, the default block is executed. Take note however, that the default part is optional. A switch statement can have no default block.

Unlike with the if statement, the multiple statements are executed in the switch statement without needing the curly braces.

When a case in a switch statement has been matched, all the statements associated with that case are executed. Not only that, the statements associated with the succeeding cases are also executed.

To prevent the program from executing statements in the subsequent cases, we use a break statement as our last statement. Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.

A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

```
public class Grade
{
public static void main( String[] args )
{
int grade = 92;
switch(grade)
{
case 100:
System.out.println( "Excellent!" );
break;
case 90:
System.out.println("Good job!" );
break;

case 80:
System.out.println("Study harder!" );
break;
default:
System.out.println("Sorry, you failed.");
}
}
}
```

LOOPING STRUCTURE

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths. Repetition control structures(Loops) are Java statements that allows us to execute specific blocks of code a number of times.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages:
Java programming language provides the following types of loop to handle looping requirements.

- While loop
- Do…….While loop
- For loop

The while loop

The while loop is a statement or block of statements that is repeated as long as some condition is satisfied.
Syntax:
The while statement has the form,

```
while( boolean_expression )
{ statement1;
statement2;
. . .
}
```

The statements inside the while loop are executed as long as the boolean_expression evaluates to true. When the condition becomes false, program control passes to the line immediately following the loop.

For example, given the following code segment,

```
int i = 4;
while ( i > 0 )
{
System.out.print(i);
i--;
```

}

The sample code shown will print 4321 on the screen. Take note that if the line containing the statement i--; is removed, this will result to an infinite loop, or a loop that does not terminate. Therefore, when using while loops or any kind of repetition control structures(Loops), make sure that you add some statements that will allow your loop to terminate at some point.

```
public class natural
{
public static void main(String args[])
{
int x = 10;
while( x < 20 )
{
System.out.print(“value of x : “ + x );
x++;
System.out.print(“\n”);
}
}
}
```

This would produce the following result:

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
```

value of x : 19

do...while loop

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax:

```
do
{
//Statements
} while (Boolean_expression);
```

The expression appears at the end of the loop, so the statements in the loop execute once before the Boolean is tested.

If the expression is true, the control jumps back up to do statement, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

Here are a few examples that uses the do-while loop:

Example 1:

```
int x = 0;
do
{
System.out.print(x);
x++;
}while (x<10);
```

This example will output 0123456789 on the screen.

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

```
int x = 0;
do
{
System.out.print(“value of x : “ + x );
x++;
System.out.print(“\n”);
}while( x <10 );
}
}
```

The output of the above code is :

```
value of x : 0
value of x : 1
value of x :2
value of x : 3
value of x : 4
value of x : 5
value of x : 6
value of x : 7
value of x : 8
value of x : 9
```

for loop

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.A for loop is useful when you know how many times a task is to be repeated.

Syntax:

```
for(initialization; Boolean_expression; update)
{
 //Statements
```

}

The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables. and this step ends with a semi colon (;)
Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop will not be executed and control jumps to the next statement past the for loop.
After the body of the for loop gets executed, the control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank with a semicolon at the end.
The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the for loop terminates.

A simple example of the for loop is,

```
int i;
for( i = 0; i < 10; i++)
{
System.out.print(i);
}
```

In this example, the statement i=0, first initializes our variable. After that, the condition expression i<10 is evaluated. If this evaluates to true, then the statement inside the for loop is executed. Next, the expression i++ is executed, and then the

condition expression is again evaluated. This goes on and on, until the condition expression evaluates to false.
This example, is equivalent to the while loop shown below,

```
int i = 0;
while( i < 10 )
{
System.out.print(i);
i++;
}
```

```
public class example
{
public static void main(String args[])
{
for(int x = 100; x <110; x = x+1)
{
System.out.print(“value of x : “ + x ); System.out.print(“\n”);
}
}
}
```

This would produce the following result:

```
value of x : 100
value of x : 101
value of x : 102
value of x : 103
value of x : 104
value of x : 105
value of x : 106
value of x : 107
value of x : 108
value of x : 109
```

UNIT-3 OBJECTS AND CLASSES

3.1 Concept and Syntax of class

In Java everything is encapsulated under classes. Class is the core of Java language. Class can be defined as a template/ blueprint that describe the behaviours /states of a particular entity. A class defines new data type. Once defined this new type can be used to create object of that type. Object is an instance of class. You may also call it as physical existence of a logical template class.

A class is declared using class keyword. A class contain both data and code that operate on that data. The data or variables defined within a class are called instance variables or data members and the code that operates on this data is known as methods or member functions.

Rules for Java class:

- A class can have only public or default(no modifier) access specifier.
- It can be either abstract, final or concrete (normal class).
- It must have the class keyword, and class must be followed by a legal identifier.
- It may optionally extend one parent class. By default, it will extend java.lang.Object.
- The class's variables and methods are declared within a set of curly braces {}.
- Each .java source file may contain only one public class. A source file may contain any number of default visible classes.

- Finally, the source file name must match the public class name and it must have a .java suffix.

3.2 Defining a Class

Before writing your class, think first on where you will be using your class and how your class will be used. Think of an appropriate name for the class, and list all the information or properties that you want your class to contain. Also list down the methods that you will be using for your class.
To define a class, we write,
<modifier> class <name>
{
<attributeDeclaration>
<constructorDeclaration>
<methodDeclaration>
}
where
<modifier> is an access modifier, which may be combined with other types of modifier.
Coding Guidelines:
Remember that for a top-level class, the only valid access modifiers are public and default (i.e., if no access modifier prefixes the class keyword).

As an example, we will be creating a class that will contain a student record. Since we've already identified the purpose of our class, we can now name it. An appropriate name for our class would be StudentRecord.
Now, to define our class we write,
public class StudentRecord
{ where,

```
//we'll add more code here later
}
```

public - means that our class is accessible to other classes outside the package

class - this is the keyword used to create a class in Java

StudentRecord - a unique identifier that describes our class name

Coding Guidelines:

Think of an appropriate name for your class. Don't just call your class XYZ or any random names you can think of.

Class names should start with a CAPITAL letter.

The filename of your class should have the SAME NAME as your public class name.

Suppose, Student is a class and student's name, roll number, age will be its property. Lets see this in Java syntax

```
class Student
{
String name;
int rollno;
int age;
}
```

When a reference is made to a particular student with its property then it becomes an object, physical existence of Student class.

```
Student std=new Student();
```

To declare a certain attribute for our class, we write,

<modifier> <type> <name> [= <default_value>];

After the above statement std is instance/object of Student class. Here the new keyword creates an actual physical copy of the object and assign it to the std variable. It will have physical existence and get memory in heap area. The new operator dynamically allocates memory for an object

3.3 Concept and Syntax of Methods

Method describe behaviour of an object. A method is a collection of statements that are group together to perform an operation.

Syntax :

```
return-type methodName(parameter-list)
{
//body of method
}
```

Example of a Method

```
public String getName(String st)
{
String name=“welcome";
name=name+st;
return name;
}
```

Modifier : Modifier are access type of method. We will discuss it in detail later.

Return Type : A method may return value. Data type of value returned by a method is declared in the method heading.

Method name : Actual name of the method.

Parameter : Value passed to a method.

Method body : collection of statement that defines what method does.

In the examples we discussed before, we only have one method, and that is the main() method. In Java, we can define many methods which we can call from different methods.

A method is a separate piece of code that can be called by a main program or any other method to perform some specific function.

The following are characteristics of methods:

- It can return one or no values
- It may accept as many parameters it needs or no parameter at all. Parameters are also called function arguments.
- After the method has finished execution, it goes back to the method that called it.

Now, why do we need to create methods? Why don't we just place all the code inside one big method? The heart of effective problem solving is in problem decomposition. We can do this in Java by creating methods to solve a specific part of the problem. Taking a problem and breaking it into small, manageable pieces is critical to writing large programs.

Sometimes it is cumbersome to write, debug or try to understand a very long program. Many times, there are some parts of the program that redundantly perform similar tasks. Actually writing statements to perform those similar tasks many times is obviously not very efficient, when one can possibly write those statements once and reuse them later when similar functionalities are needed. Programmers usually write statements to be reused in sub-programs or subroutines or

methods. This does not only allow efficient programming but also makes programs shorter which, consequently, make the programs easier to be debug and understood. Dividing statements intended to perform different tasks into different subroutines is also preferred. Methods in Java can serve the purpose just mentioned.

While talking about method, it is important to know the difference between two terms parameter and argument.
Parameter or formal argument is variable defined by a method that receives value when the method is called. Parameter are always local to the method. They don't have scope outside the method. While argument or actual argument is a value that is passed to a method when it is called.

3.4 Defining Methods

Before we discuss what methods we want our class to have, let us first take a look at the general syntax for declaring methods.
To declare methods we write,
<modifier> <returnType> <name>(<parameter>*)
{
<statement>
}
where,
<modifier> can carry a number of different modifiers
<returnType> can be any data type (including void)
<name> can be any valid identifier
<parameter> ::= <parameter_type> <parameter_name>[,]

There are two ways to pass an argument to a method

call-by-value : In this approach copy of an argument value is pass to a method. Changes made to the argument value inside the method will have no effect on the arguments. This type of calling a function is read only as it can only access the values of the actual arguments but can not make changes to them.

call-by-reference : In this type of calling, reference of an argument is passed to a method. Any changes made inside the method will affect the argument value. This type of calling a function is read write as it can not only access the values of the actual arguments but also can make changes to them.

3.5 Creating an Object

To create an object or an instance of a class, we use the new operator. For example, if you want to create an instance of the class String, we write the following code,

String str2 = new String(“Hello world!”);

or also equivalent to,

String str2 = "Hello";

The new operator allocates a memory for that object and returns a reference of that memory location to you. When you create an object, you actually invoke the class‘s constructor. The constructor is a method where you place all the initializations and it is automatically called when the object of the class is created, it has the same name as the class.

3.6 Accessing Class Members

When a method is invoked, typical steps involving the values passed between the caller and the method as well as the method's local variables are:

- If there are input arguments to the method, the value of each argument is copied to its corresponding local variable declared inside the method header.
- Local variables are declared and assigned with values according to the statements implemented in the method.
- If the method returns a value back to the caller, the returned value is copied from inside the method to the caller. From the scope of the caller, the expression corresponding to the invoked method is evaluated to that returned value.
- When the program flow is returned to the caller, all local variables of the method are no longer accessible by the program.

Static methods are methods that can be invoked without instantiating a class (means without invoking the new keyword). Static methods belongs to the class as a whole and not to a certain instance (or object) of a class. Static methods are distinguished from instance methods in a class definition by the keyword static.

To call a static method, just type,

```
Classname.staticMethodName(params);
```

Examples of static methods, we've used so far in our examples are,

```
//prints data to screen
System.out.println("Hello world");
//converts the String 10, to an integer
```

```
int i = Integer.parseInt(“10”);
 //Returns a String representation of the integer argument as an
//unsigned integer base 16

String hexEquivalent = Integer.toHexString( 10 );
```

For the static variable studentCount, we can create a static method to access its value.

```
public class StudentRecord
{
private static int studentCount;
public static int getStudentCount()
{
return studentCount;
}
}
```

where,

public - means that the method can be called from objects outside the class

static - means that the method is static and should be called by typing,[ClassName].[methodName]. For example, in this case, we call the method StudentRecord.getStudentCount()

int - is the return type of the method. This means that the method should return a value of type int

getStudentCount - the name of the method

() - this means that our method does not have any parameters

For now, getStudentCount will always return the value zero since we haven't done anything yet in our program in order to set its value.

Coding Guidelines:
Method names should start with a SMALL letter.
Method names should be verbs

3.7 Instance Data and Class Data

Aside from instance variables, we can also declare class variables or variables that belong to the class as a whole. The value of these variables are the same for all the objects of the same class. Now suppose, we want to know the total number of student records we have for the whole class, we can declare one static variable that will hold this value. Let us call this as studentCount.
To declare a static variable,

```
public class StudentRecord
{
//static variables we have declared
private static int studentCount;
//we'll add more code here later
}
```

we use the keyword static to indicate that a variable is a static variable. So far, our whole code now looks like this.

```
public class StudentRecord
{
private String name;
private String address;
private int age;
private double mathGrade;
private double englishGrade;
private double scienceGrade;
```

```
private double average;
private static int studentCount;
//we'll add more code here later
}
```

Now that we have a list of all the attributes we want to add to our class, let us now add them to our code. Since we want these attributes to be unique for each object (or for each student), we should declare them as instance variables.
For example,

```
public class StudentRecord
{
private String name;
private String address;
private int age;
private double mathGrade;
private double englishGrade;
private double scienceGrade;
private double average;
}
```

where,
private here means that the variables are only accessible within the class. Other objects cannot access these variables directly.
Coding Guidelines:
Declare all your instance variables on the top of the class declaration.
Declare one variable for each line.
Instance variables, like any other variables should start with a SMALL letter.
Use an appropriate data type for each variable you declare.

Declare instance variables as private so that only class methods can access them directly.

```
public class C11A
{
public static int i;
public int j;
}
public class StaticDataMemberDemo
{
public static void main(String[] args)
{
C11A x = new C11A();
C11A y = new C11A();
C11A z = new C11A();

x.j = 5;
y.j = 10;
z.j = 15;
System.out.println("x.j = "+x.j);
System.out.println("y.j = "+y.j);
System.out.println("z.j = "+z.j);
x.i = 0;
y.i++;
z.i += 3;
System.out.println("x.i = "+x.i);
System.out.println("y.i = "+y.i);
System.out.println("z.i = "+z.i);
}
```

Three instances of the class C11A are created and referred to by x, y and z. Values are assigned to the instance variables j belonging to the objects referred to by x, y, and z, respectively. These objects do not share the value of j. However, the variable i is shared by the three objects. The statement x.i = 0 assign 0 to i. Note that at this point y.i and z.i are also 0 since they refer to the same memory location. i can be modified via any objects. Therefore, we can see that the resulting value of i, shared by x, y and z, is 4.

3.8 Constructors

A constructor is a special method that is used to initialize an object. It is automatically called when an object of the class is instantiated.Every class has a constructor,if we don't explicitly declare a constructor for any java class the compiler builds a default constructor for that class. A constructor does not have any return type.

A constructor has same name as the class in which it resides. Constructor in Java can not be abstract, static, final or synchronized. These modifiers are not allowed for constructor.

```
class Car
{
String name ;
String model;
Car( ) //Constructor
{
name ="";
model="";
}
}
```

There are three types of Constructor
Default Constructor or non parameterized constructor.
Parameterized constructor
Copy Constructor
Each time a new object is created at least one constructor will be invoked.

```
Car c = new Car(); //Default constructor invoked . No parameter
                   //  is passed.
Car c = new Car(name); //Parameterized constructor invoked.
                       //  Parameter is passed
Car c1=new Car(c); //object of the class is passed as an
                   //  argument
```

Constructors are special public methods invoked whenever an object of the class is created. Constructors are defined in the same fashion as defining methods. However, constructors must have the same name as the class name and there must not be any return types specified at the header of the constructors. Constructors are usually for initializing or setting instance variables in that class.
Given a class called MyClass, its constructors are in the following structure.

```
public Myclass(<input argument list>){
// Body of the constructor
}
```

Here is an example of a no-argument (no input) constructor for MyPoint, in which its instance variables are set with 1.0.

```
public MyPoint()
{ x = 1.0;
y = 1.0;
}
```

Adding this constructor to the class definition of MyPoint, we obtain:

```
public class MyPoint
{
// data members private double x; private double y;
// constuctors
public MyPoint(){
x = 1.0;
y = 1.0;
}
public String toString()
{
return "("+x+","+y+")";
}
}
```

3.9 Access specifiers

- Java language has four access modifier to control access levels for classes, variable methods and constructor.
- default : default has scope only inside the same package. The default means, we do not specify any specifier at all, then such a class, method or variable will be accessible within the class and other classes within same package. It is known as package level access specifier, because the

classes of the same package only are allowed to access the members of the class.

- public : public scope is visible everywhere. When a class member is declared as public it means this member can be accessed anywhere, i.e. in any class of any package. Java source code can only contain one public class, where name must also match with the filename. To declare the class and its members as public, use public keyword.
- protected : protected has scope within the package and all sub classes. Member of a class declared with protected access specifier can be accessed within the same class, and its subclasses within same package or in other package. To declare the members as protected, use protected keyword.
- private : private has scope only within the classes. Members of a class declared with private access specifier, can be accessed only within the class in which they are declared. These members are not available for subclass and other package. To declare the members as private, use private keyword. A class cannot be declared as private.

3.10 Access Modifiers

Access modifiers do not change the accessibility of variables and methods, but they do provide them special properties. Access modifiers are of 5 types,

- final
- static
- abstract
- transient
- synchronized

- volatile
- **final modifier**
- final modifier is used to declare a field as final i.e. it prevents its content from being modified. Final field must be initialized when it is declared.
- A class can also be declared as final. A class declared as final cannot be inherited. String class in java.lang package is a example of final class. Method declared as final can be inherited but you cannot override(redefine) it.

- Example :

```
class Cloth
{
final int MAX_PRICE = 999; //final variable
final int MIN_PRICE = 699;
final void display() //final method
{
System.out.println("Maxprice is" + MAX_PRICE );
System.out.println("Minprice is" + MIN_PRICE);
}
}
```

- **Static Modifier**
- Static Modifiers are used to create class variable and class methods which can be accessed without instance of a class. Let us study how it works with variables and member functions.
- Static Variables

- Static variables are defined as a class member that can be accessed without any object of that class. Static variable has only one single storage. All the object of the class having static variable will have the same instance of static variable. Static variables are initialized only once.
- Static variable are used to represent common property of a class. It saves memory. Suppose there are 100 employee in a company. All employee have its unique name and employee id but company name will be same for all the 100 employees. Here company name is the common property. So if you create a class to store employee detail, company_name field will be mark as static.
- **abstract modifier**

- An abstract class is a class that cannot be instantiated. It often appears at the top of an object-oriented programming class hierarchy, defining the broad types of actions possible with objects of all subclasses of the class. Abstract classes must be inherited so that the abstract methods can be implemented in the sub classes.
- Those methods in the abstract classes that do not have implementation and only contain method signature are called abstract methods. To create an abstract method, just write the method declaration without the body and use the abstract keyword. For example,
  public abstract void someMethod();
- **Transient modifier**
- When an instance variable is declared as transient, then its value doesn't persist when an object is serialized
- **Synchronized modifier**
- When a method is synchronized it can be accessed by only one thread at a time. We will discuss it in detail in Thread.

- **Volatile modifier**
- Volatile modifier tells the compiler that the volatile variable can be changed unexpectedly by other parts of your program. Volatile variables are used in case of multithreading program.

3.11 Access Control

- Accessibility of Different Types of Access Specifiers :
  The accessibility of different types of access specifiers can be rated through the following table

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

- public access specifier
- This specifies that class members are accessible to anyone, both inside and outside the class and from any package. Any object that interacts with the class can have access to the public members of the class. For example,

```
public class StudentRecord
{
//default access to instance variable
 public int name;
//default access to method
 public String getName()
```

```
{
return name;
}
}
```

- In this example, the instance variable name and the method getName() can be accessed from other classes and other packages also.
- Protected Access Specifier :
- Member of a class declared with protected access specifier can be accessed within the same class, and its subclasses within same package or in other package. To declare the members as protected, use protected keyword.
- e.g.

```
package my pack;
class XYZ
{
protected int a;
protected void add()
{
.....
}
}
```

- protected members can be accessed by the sub class inside or outside the package.We will have to extend this class using extends keyword. If ABC class (sub class of XYZ) is defined in another package and we want to access members of XYZ class within ABC class, write the following code:

  Example
  e.g.

```
package mypackl;
import mypack.XYZ;
class ABC extends XYZ
{
public void show()
{
 a = 15; // legal
add () ; // legal
}
}
```

- Default Access Specifier :
- The default means, we do not specify any specifier at all, then such a class, method or data member will be accessible within the class and other classes with in the same package. It is known as package level access specifier, because the classes of the same package only are allowed to access the members of the class.
- e.g.

```
package mypack;
class ABC
{
int a ;
void add()
{
.....
}
}
```

- Example

- If XYZ is a class within the same package and we want to access the members of class ABC within the XYZ class, write the following code:

```
package mypack;
class XYZ
{
protected void show ( )
{
ABC object= new ABC ( );
object.a = 5; / / legal
object.add (); / / legal
}
}
```

- Private Access Specifier :
- Members of a class declared with private access specifier, accessed only within the class in which they are declared. These members are not available for subclass and other package. To declare the members as private, use private keyword. A class cannot be declared as private.
- e.g.

```
class XYZ
{
private int a;
private void add()
{
a = 10;
}
}
```

UNIT-4 USING JAVA OBJECTS

4.1 String Builder and String Buffer

- String Class
- The String class is the class, whose instances or objects can hold unchanging strings. It means once the objects are initialised they cannot be modified or changed.
- It provides several methods for creating and manipulating String object. Strings in Java program are created by declaring object of string type class and initialising it with a string literal.

e.g.

- String name= “My name is Shreyash”;
- The String objects can also be created by using new keyword. e.g.
- String name = new String (“What is your name?”);
- charAt (int Index) - It returns the character at the specified index. e.g.
  String str = “Shreyash”;
  System.out.println (str .charAt(2));
  Output r
- concat (String Str) - It concatenates the specified string to the end of current string object.
  String sl=”Shreyash”;
  sl=sl.concat(“Pradhan “);
  System.out. println(sl) ;
- Output
- Shreyash Pradhan
- int length() - It returns the length of current or this string. e.g.

- String str = “Shreyash”;
- System.out.println(str.length( ));
- Output 8
- toLowerCase () - It converts all the characters of a given string to lowercase.
- e.g.
  String str = “SHREYASH” ;
  System.out.println (str.toLowerCase()) ;
  Output
  Shreyash

- toUpperCase ()- It converts all the characters of a given string to uppercase.
- e.g.
- String str = “Shreyash” ;
- System.out.println (str.toUpperCase()) ;
  Output SHREYASH
- trim ()- It removes the spaces from the beginning and end of the current or this string. e.g.
  String ob=” Hello “;
  String object= ob.trim ();
  Output Hello
- substring (int startlndex) - It returns a substring from the current string, which is starting from the start index character to the end of invoking string object.
- e.g.
- String str = “0123456789” ;
  System.out. println(str.substring(4));
  Output 456789

- String substring(int startIndex, int endIndex) : It returns a substring from the current string, which is starting from the startIndex character and ends at the endIndex character.
- e.g.
  String str = “0123456789” ;
  System.out. println(str.substring(4,7)) ;
  Output
  456

- StringBuffer class is used to create a mutable string object. As we know that String objects are immutable, so if we do a lot of changes with String objects, we will end up with a lot of memory leak.
- So StringBuffer class is used when we have to make lot of modifications to our string. It is also thread safe i.e multiple threads cannot access it simultaneously. StringBuffer defines 4 constructors. They are,
- StringBuffer ( )
- StringBuffer (int size)
- StringBuffer (String str)
- StringBuffer (charSequence [ ]ch)
- StringBuffer() creates an empty string buffer and reserves room for 16 characters.
- StringBuffer(int size) creates an empty string and takes an integer argument to set capacity of the buffer.
- StringBuilder is identical to StringBuffer except for one important difference it is not synchronized, which means it is not thread safe. Its because StringBuilder methods are not synchronised.

- StringBuilder Constructors
- StringBuilder( ), creates an empty StringBuilder and reserves room for 16 characters.
- StringBuilder(int size), create an empty string and takes an integer argument to set capacity of the buffer.
- StringBuilder (String str), create a StringBuilder object and initialize it with string str.

4.2 Methods and Messages

- The following methods are some most commonly used methods of StringBuffer class.
- append()
- This method will concatenate the string representation of any type of data to the end of the invoking StringBuffer object. append() method has several overloaded forms.
- StringBuffer append(String str)

  StringBuffer append(int n)

- StringBuffer append(Object obj)
- The string representation of each parameter is appended to StringBuffer object.
  StringBuffer str = new StringBuffer("test");
  str.append(123);
  System.out.println(str);
  Output : test123
- insert()
- This method inserts one string into another. Here are few forms of insert() method.
- StringBuffer insert(int index, String str)

- StringBuffer insert(int index, int num)
- StringBuffer insert(int index, Object obj)
- Here the first parameter gives the index at which position the string will be inserted and string representation of second parameter is inserted into StringBuffer object.
  StringBuffer str = new StringBuffer("test");
  str.insert(4, 123);
  System.out.println(str);
  Output : test123
- reverse()
- This method reverses the characters within a StringBuffer object.
  StringBuffer str = new StringBuffer("Hello");
  str.reverse();
  System.out.println(str);
  Output : olleH
- replace()
- This method replaces the string from specified start index to the end index.
  StringBuffer str = new StringBuffer("Hello World");
  str.replace(6, 11, "java");
  System.out.println(str);
  Output : Hello java

- Example
  class Test
  {
  public static void main(String args[])
  {
  StringBuffer strB = new StringBuffer("study");
  strB.append("tonight");
  System.out.println(strB);

```
}
}
```
Output: studytonight

```
class Test
 {
public static void main(String args[])
{
StringBuilder str = new StringBuilder("study");
str.append( "tonight" );
System.out.println(str);
str.replace( 5, 12, "today");
System.out.println(str);
str.reverse();
System.out.println(str);
}
}
```

Output :

studytonight
studytoday
yadotyduts

4.4 Comparing and Identifying Objects

To compare instances of a class and have meaningful results, we must implement special methods in our class and call those methods. A good example of this is the String class.

It is possible to have two different String objects that contain the same values. If the == operator is used to compare these objects, however, they would be considered unequal. Although their contents match, they are not the same object.

To see whether two String objects have matching values, a method of the class called equals() is used. The method tests each character in the string and returns true if the two strings have the same values.

The following code illustrates this,

```
class EqualsTest
{
public static void main(String[] arguments)
{
String str1, str2;
str1 = "Free the bound periodicals.";
str2 = str1;
System.out.println("String1: " + str1);
System.out.println("String2: " + str2);
System.out.println("Same object? " + (str1 == str2));
str2 = new String(str1);
System.out.println("String1: " + str1);
System.out.println("String2: " + str2);
System.out.println("Same object? " + (str1 == str2));
System.out.println("Same value? " + str1.equals(str2));
}
}
```

This program's output is as follows,
OUTPUT:
String1: Free the bound periodicals.
String2: Free the bound periodicals. Same object?
true
String1: Free the bound periodicals.
String2: Free the bound periodicals. Same object?
false
Same value? True

Now let's discuss the code.
String str1, str2;
str1 = "Free the bound periodicals.";

The first part of this program declares two variables (str1 and str2), assigns the literal "Free the bound periodicals." to str1, and then assigns that value to str2. Here str1 and str2 now point to the same object, and the equality test proves that.
str2 = new String(str1);
In the second part of this program, a new String object is created with the same value as str1 and assign str2 to that new String object. Now there are two different string objects in str1 and str2, both with the same value. Testing them to see whether they're the same object by using the == operator returns the expected answer: false—they are not the same object in memory. Testing them using the equals() method also returns the expected answer: true—they have the same values.

Similarly the stored value in a variable of any non-primitive data types, or class, is a reference to an object of that class. Therefore, when the relational equality (==) is used to compare two variables of non-primitive data types, it compares whether the references are identical. In other words,the relational equality will yield true if both variables contain exactly the same object, not just different objects that might have identical properties, or, more precisely, attributes. Equality testing for objects of non-primitive data types with identical attributes can be performed by using the equal() method.

The expression a.equals(b) where a is a variable referring to an object and b is a variable referring to another object is evaluated as true if both objects have identical properties without referring to the same object. Otherwise, it is evaluated to false. The case when two String objects with their contents containing the same character sequences is an example case of when two objects have identical properties.

Another method that involves String comparison is called compareTo().

Given that s1 and s2 are String objects. The expression s1.compareTo(s2) returns:

0 if s1 and s2 have the same character sequence.

s1.charAt(k)-s2.charAt(k) if there is a smallest position k, at which they differ.

s1.length()-s2.length() if there is no position k at which they differ.

Example : Comparing Strings with compareTo()

Observe the output of the following program.

```
public class StringCompareDemo1
{
public static void main(String[] args)
{
System.out.println("Wonderland".compareTo("Wonderla
nd"));
System.out.println("Wonderful".compareTo("Wonderboy
"));
System.out.println("Wonderful".compareTo("Wonderland
"));
}
}
```

The result from line 5 is 0 since both strings contain the same text. The result from line 6 is 4 since the characters at the smallest position that both strings differ are ‘f’ and ‘b’. Thus, the result equals ‘f’-‘b’. The result from line 7 is -6 due to the same reason since -6 is ‘f’-‘l’.

UNIT-5 INHERITANCE

5.1 Inheritance in Java

Inheritance is one of the key features of Object Oriented Programming. It provides a mechanism that allows a class to inherit property of another class. When a class extends another class it inherits all non-private members including data members and member methods. Inheritance in Java can be best understood in terms of Parent and Child relationship. Inheritance is the ability to derive a new class from an existing class. The existing class is known as Super class(Parent) or Base class and Sub class(child) or Derived class.

Inheritance defines is-a relationship between a Super class and its Sub class.

A subclass can be thought of as an extension of its superclass. It inherits all attributes and behaviours from its superclass. However, more attributes and behaviours can be added to existing ones of its superclass.

In Java, all classes, including the classes that make up the Java API, are subclassed from the Object superclass.

Any class above a specific class in the class hierarchy is known as a superclass. While any class below a specific class in the class hierarchy is known as a subclass of that class.

Inheritance is an important concept in object oriented programming because it allows re-using code without having to rewrite from scratch. Inheritance can be defined as the process, where one class acquires the properties of another. It supports the concepts of hierarchical classification.

Supper Class and Sub Class :

In Java language, inheritance is a mechanism that allows you to build new classes from existing classes, the old class is known as base class, super class or parent class and the new class is known as derived class or sub class or child class. The sub class is the specialised version of super class. It inherits all of the instance variables and methods defined by the super class and add its own unique members.

Super Class :

A super class is a class that has been extended to another class.

Syntax

<access_specifier> class <class_name >

{

}

e.g.

public class parent

{

}

Sub Class

A sub class is a class that derives from another class. A sub class is defined by using the keyword extends along with super_class_name.

Syntax

<access-spec i fi er> class <sub- class name> extends <super class-name >

{

}

e.g.

public class Child extends Parent

{

}

Note : The keyword extends signifies that, the properties of super class are extended to the sub class, that means, sub class contains its own members as well as share the members of the super class. The inheritance allows sub classes to inherit all properties (variables and methods) of their parent classes.

5.2 Use of Inheritance

Reusability : Inheritance helps the code to be reused in many situations. The base class is defined and once it is compiled, it needs not to be reworked. Using the concept of inheritance, the programmer can create as many derived classes from the base class as needed, while adding specific features to each derived class as needed.

Saves Time and Effort : Since, the main code written can be reused in various situations (as needed), it saves time and effort of the programmer.

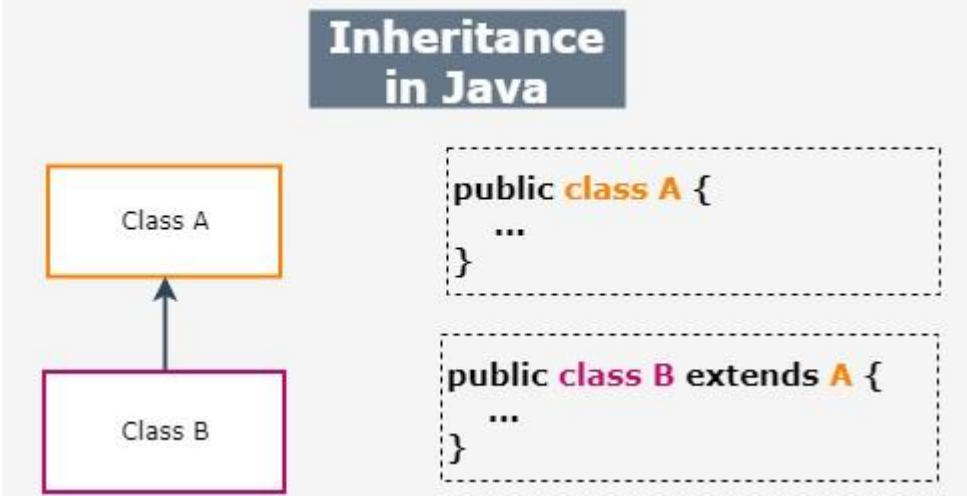
Reliability : Increases program structure which results in greater reliability.

5.3 Types of Inheritance

We know that, inheritance is an ability of Object Oriented Programming (OOP) to inherit the properties of an existing class to a sub class. There are five forms of inheritance:

5.4 Single Inheritance :

When a sub class inherits from only one super class or base class. It is known as single inheritance. It represents a linear relationship between a super class and its sub class.



To show single inheritance or class inheritance.

```
class A
{
int x;
int y;
int get(int p,int q)
{
x = p;
y = q;
return 0;
}

void show()
{
System.out.println(“value of x =”+x);
System.out.println(“value of y =”+y);
}
}
class B extends A
{
public static void main(String args[ ])
```

```
{
B b=new B();
b.get(5,6);
b.show();
}
}
```
Output: value of x = 5
value of y = 6

The super keyword

A subclass can also explicitly call a constructor of its immediate super class. This is done by using the super constructor call. A super class constructor call in the constructor of a subclass will result in the execution of relevant constructor from the super class, based on the arguments passed.

There are a few things to remember when using the super constructor call:

The super() call must occur as the first statement in a sub class constructor.

The super() call can only be used in a constructor definition.

The super() always refers to the super class above the calling class.

```
class A
{
protected int x,y;
A(int x1,int y1)
{ x=x1;
y=y1;
}
int getx()
{
```

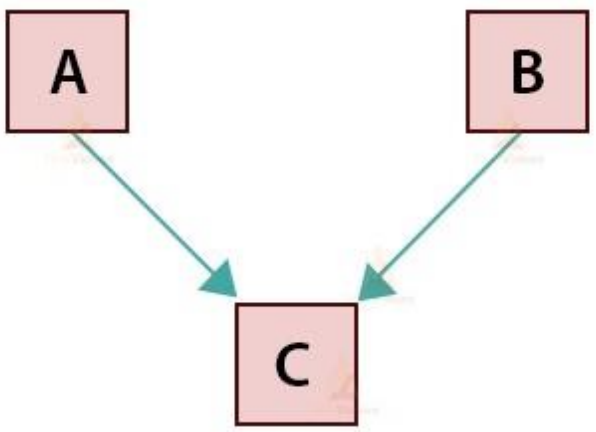
```
return x;
}
int gety()
{
return y;
}
}

class B extends A
{
protected int z;
B(int x1,int y1,int z1)
{super(x1,y1);
z=z1;
}
int getz()
{
return z;
}
}

Class test
{
public static void main(String args[ ])
{
B b=new B(5,6,7);
System.out.println(“ x= “+b.getx();
System.out.println(“ x= “+b.gety();
System.out.println(“ x= “+b.getz();
}
}
```

Output: x = 5

y = 6
z = 7

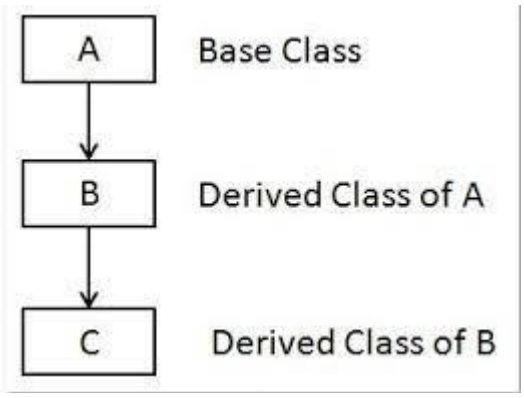
Multiple Inheritance :

When a sub class inherits from more than one super classes or base classes, it is known as multiple inheritances. Java does not support multiple inheritances, but the multiple inheritances can be achieved by using a keyword implements and concept of interface. Here, class C inherits the properties and methods of class A and class B. This type of inheritance is not available in java.



5.5 Multilevel Inheritance :

When a sub class inherits from a super class, which itself inherited from another super class, then such inheritance is known as multilevel inheritance. Here, class B inherits the properties and methods of class A and class C inherits the properties and methods of class B as well as inherits the properties and methods of class A.



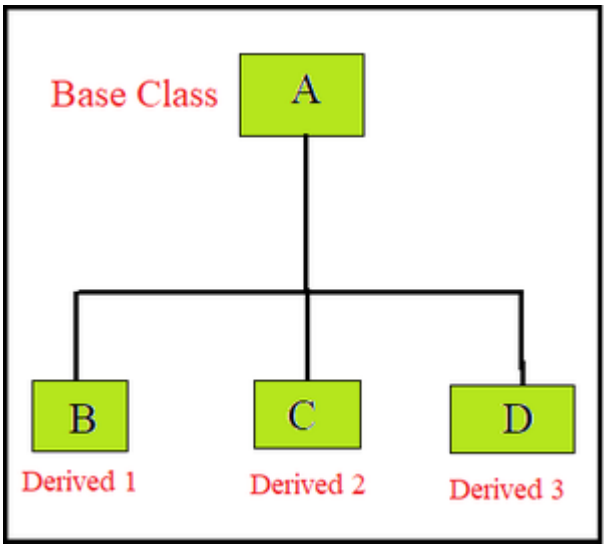

```
class A
{
A()
{
System.out.println(“ The Grand Parent class A”);
}
}
class B extends A
{
B()
{
System.out.println(“ The Parent class B”);
}
}

class C extends B
{
C()
{
System.out.println(“ The Child class C”);
}
}
Class test
{
public static void main(String args[ ])
```

```
{
A a=new A();
B b=new B();
C c=new C();
}
}
```

Output: The Grand Parent class A
The Parent class B
The Child class C

5.6 Hierarchical Inheritance :

When more than one sub classes are inherited from one super class or base class, it is known as hierarchical inheritance. Here, class B, C and class D inherits the properties and methods of class A.




```
/*
 * Example of hierarchical inheritance in java
 *
 */

//Base class
class Teacher {
```

```
	public void programming() {
		System.out.println("Java programming...");
	}

	void physics() {

		System.out.println("Physics...");
	}

	void chemistry() {

		System.out.println("Chemistry...");

	}
}

// Inherits feature of Teacher class
class ComputerDepartment extends Teacher {

	public void learn() {
		System.out.println("ComputerDepartment : Learn...");
	}

}

// Inherits feature of same Teacher class
class ScienceDepartment extends Teacher {

	public void learn() {
		System.out.println("\nScienceDepartment : Learn...");
	}
}
```

```
/*
 * Test hierarchical inheritance
 */
public class TestHierarchicalInheritance {

	public static void main(String[] args) {

		ComputerDepartment cd = new ComputerDepartment();
		cd.learn();
		cd.programming();

		// Science department
		ScienceDepartment sd = new ScienceDepartment();
		sd.learn();
		sd.physics();
		sd.chemistry();

	}

}
```

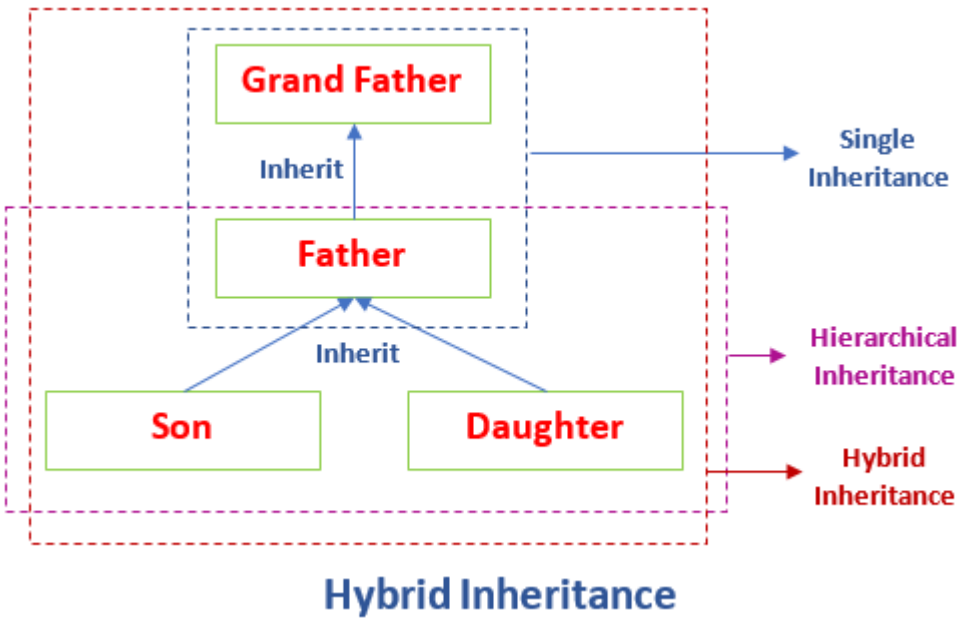
5.7 Hybrid Inheritance :

It is the combination of two or more than two types of inheritance available in Java.

When a sub class B inherits from a base classes A and from which two classes C and D are derived.This form of inheritance is known as hybrid inheritance because it is a combination of multilevel and hierarchical inheritance.

Here, class B inherits the properties and methods of class A. class C and class D inherit the properties and methods of class B as well as inherit the properties and methods of class A.

Note : In the combination, if one of the form is multiple inheritance then the inherited combination (hybrid inheritance) is not supported by Java.



Hybrid Inheritance

UNIT-6 POLYMORPHISM

6.1 Types of Polymorphism

The process of representing one form in multiple forms is known as Polymorphism. Polymorphism is derived from two greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

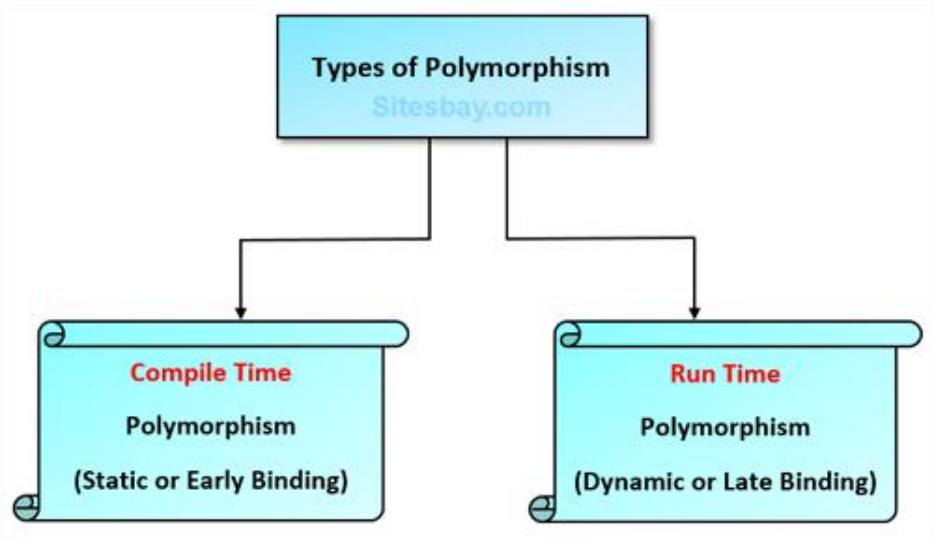
Real life example of polymorphism in Java

Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you are at your home at that time you behave like a son or daughter, Here one person present in different-different behaviours.

Polymorphism is divided into two sub categories, they are:

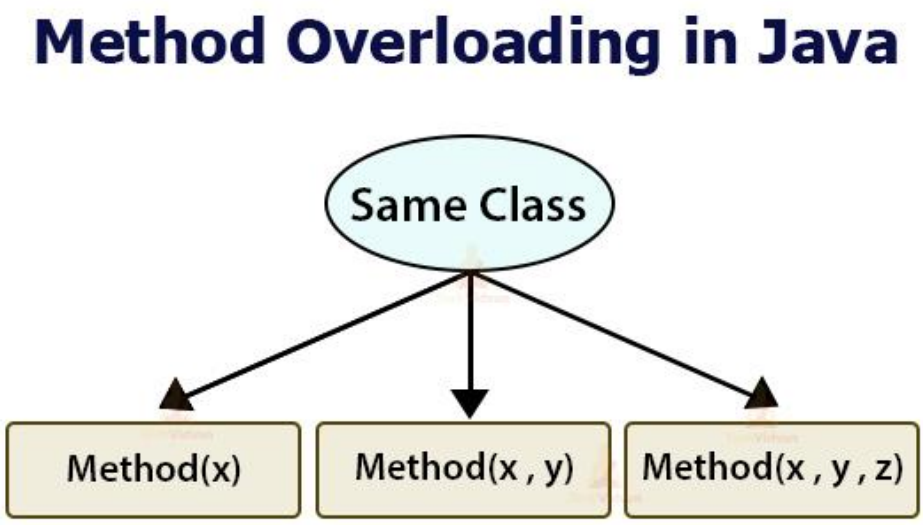
- Static or Compile time polymorphism
- Dynamic or Runtime polymorphism

Static polymorphism in Java is achieved by method overloading and Dynamic polymorphism in Java is achieved by method overriding.



6.2 Method Overloading

If two or more method in a class have same name but different parameters, it is known as method overloading.
Method overloading is one of the ways through which java supports polymorphism. Method overloading can be done by changing number of arguments or by changing the data type of arguments. If two or more method have same name and same parameter list but differs in return type are not said to be overloaded method.



Different ways of Method overloading
There are two different ways of method overloading
Method overloading by changing data type of Arguments
Example :

```
class Calculate
{
void sum (int a, int b)
{
System.out.println("sum is"+(a+b)) ;
}
void sum (float a, float b)
```

```
{
System.out.println("sum is"+(a+b));
}
Public static void main (String[] args)
{
Calculate cal = new Calculate();
cal.sum (8,5); //sum(int a, int b) is method is called.
cal.sum (4.6f, 3.8f); //sum(float a, float b) is called.
}
}
```

Output :
Sum is 13
Sum is 8.4
You can see that sum() method is overloaded two times. The first takes two integer arguments, the second takes two float arguments.

Method overloading by changing no. of arguments.
Example :

```
class Area
{
void find(int l, int b)
{
System.out.println("Area is"+(l*b)) ;
}
void find(int l, int b,int h)
{
System.out.println("Area is"+(l*b*h));
}
public static void main (String[] args)
```

```
{
Area ar = new Area();
ar.find(8,5); //find(int l, int b) is method is called.
ar.find(4,6,2); //find(int l, int b,int h) is called.
}
}
```

Output :

Area is 40

Area is 48

In this example the find() method is overloaded twice. The first takes two arguments to calculate area, and the second takes three arguments to calculate area.

When an overloaded method is called, java compiler look for a match between the arguments to call the method and the method's parameters. This match need not always be exact, sometimes when exact match is not found, Java automatic type conversion plays a vital role.

Different methods, even though they behave differently, can have the same name as long as their argument lists are different. This is called Method overloading. Method overloading is useful when we need methods that perform similar tasks but with different argument lists, i.e. argument lists with different numbers or types of parameters. Java compiler decides which method to be called by comparing the number and types of input parameters with the argument list of each method definition during compilation. So function overloading is a type of compile time polymorphism.

Once MyPoint is defined this way, whenever a MyPoint object is instantiated with new MyPoint(), a new object is created with

x and y initialized with 1.0 due to the two statements in the constructor.

Like methods, a constructor can also be overloaded. Overloaded constructors are differentiated on the basis of their type of parameters or number of parameters. Constructor overloading is not much different than method overloading. In case of method overloading you have multiple methods with same name but different signature, whereas in Constructor overloading you have multiple constructor with different signature but only difference is that Constructor doesn't have return type in Java.

Constructor overloading:

Constructors can be overloaded just like methods. A class can have multiple constructors with different input argument lists. Which constructor to be called when an instance of the class is created depends on the input argument list used with the new statement.

No-argument Constructor or non parameterised constructor

The no-argument constructor is the constructor that does not take any input arguments. Therefore, it usually contains a set of instructions that provide a default initialization to the object's data members.

Parametrised Constructor

The parametrised constructor usually refers to the constructor each input argument of which is corresponding to a data member of the class. Typical implementation of this constructor is to initialize every data member with its corresponding input argument.

Copy Constructor

The copy constructor usually refers to the constructor that takes another object of the same class as its input argument. It usually

initializes each data member of the new object with the value of the corresponding data member of the input object. This results in that the new object has all of its attributes copied from the original one.

There can be other constructors apart from the three listed above. The rules of overloading constructors are the same as the ones governing the overloading of any other methods.

Example of constructor overloading

```
class Cricketer
{
String name;
String team;
int age;
Cricketer () //default constructor.
{
name ="";
team ="";
age = 0;
}

Cricketer(String n, String t, int a) // overloaded // parameterised constructor
{
name = n;
team = t;
age = a;
}
Cricketer (Cricketer ckt) //overloaded copy constructor
{
name = ckt.name;
```

```
team = ckt.team;
age = ckt.age;
}

public String toString()
{
return "this is " + name + " of "+team;
}
}
Class test:
{
public static void main (String[] args)
{
Cricketer c1 = new Cricketer();
Cricketer c2 = new Cricketer("sachin", "India", 32);
Cricketer c3 = new Cricketer(c2 );
System.out.println(c2);
System.out.println(c3);
c1.name = "Virat";
c1.team= "India";
c1.age = 32;
System .out. print in (c1);
}
}
```

output:

this is sachin of india

this is sachin of india

this is virat of india

6.3 Run time Polymorphism

Polymorphism in Object Oriented Programming languages are of two types : One is compile time polymorphism and the other is runtime polymorphism.

In Compile time polymorphism, during method invocation , the compiler decides the target code(method) to be called according to the number and type of arguments passed during calling of the function and builds the corresponding machine level code. This is also called as early binding or static binding or compile time polymorphism.

In an object oriented programming environment, the decision of the target code during method invocation is based upon the class of the object used , at run time.This delayed decision making for calling the method dynamically at runtime is called as Late binding or Dynamic Binding or Runtime Polymorphism .

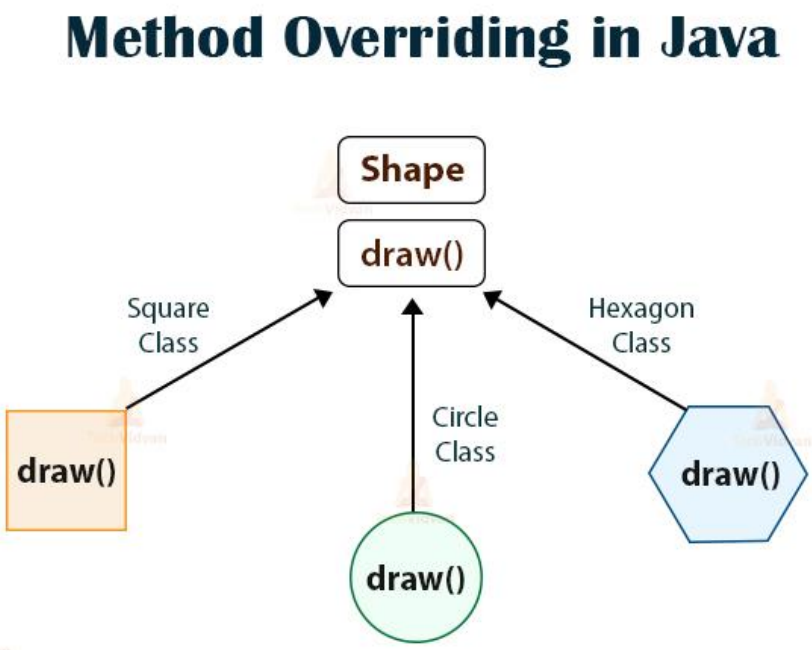
Advantages of Runtime Polymorphism

- Dynamic Polymorphism allows Java to support overriding of methods which is central for run-time polymorphism.
- It allows a class to specify methods that will be common to all of its derivatives while allowing subclasses to define the specific implementation of some or all of those methods.
- It also allows subclasses to add its specific methods subclasses to define the specific implementation of same.

6.4 Method Overriding

If for some reason a derived class needs to have a different implementation of a certain method from that of the super class, overriding methods could prove to be very useful. A

subclass can override a method defined in its super class by providing a new implementation for that method. When an overridden method is called from with in a subclass , it will always refer to the version of that method defined by the sub class. The super class method will be hidden by the sub class.



Example:

```
class base
{
void override()
{
System.out.println(“ The base class function”);
}
}
class derived extends base
{
void override()
{
System.out.println(“ The overriden derived class function”);
```

```
}
}

class test
{
public static void main(String args[ ])
{
base b=new base();
b.override();
derived d=new derived();
d.override(); // it calls override in derived
}
}
```

Output: The base class function
The overriden derived class function

If you want to access the base class version , you can achieve by including the statement super.override(); in the override() function of the derived class. or runtime polymorphism canbe used.

Example:

```
class x
{
void show()
{
System.out.println(“ Inside x”);
}
}
class y extends x
{
void show()
```

```
{
System.out.println(“ Inside y”);
}
}
class test
{
public static void main(String args[ ])
{
x x1=new x();
y y1=new y();
x ref;
ref=x1;
ref.show(); // call x method
ref=y1;
ref.show(); // call y method
}
}
```

Output: Inside x
Inside y

Example:

```
class Person
{
void walk()
{
System.out.println("Can Run....");
}
}
class Employee extends Person
{
void walk()
{
System.out.println("Running Fast...");
```

```
}
public static void main(String arg[])
{
Person p=new Employee(); //upcasting
p.walk();
}
}
```

Output: Running fast...

UNIT-7 PACKAGES: PUTTING CLASSES TOGETHER

7.1 Introduction

- Packages

Packages are Java's means of grouping related classes together in a single unit .This powerful feature provides a convenient mechanism for managing a large group of classes while avoiding potential naming conflicts. A Java package is like a directory in a file system. In fact, on the disk , a package is a directory. All Java source and class files of classes belonging to the same package are located in the same directory.

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages are used for:

- Preventing naming conflicts. For example there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee
- Making searching/locating and usage of classes, interfaces, enumerations and annotations easier.
- Providing controlled access: protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member (without any access specifier) is accessible by classes in the same package only.
- Packages can be considered as data encapsulation (or data-hiding).

All we need to do is to put related classes into packages. After that, we can simply write an import class from existing packages and use it in our program. A package is a container of a group of related classes where some of the classes are accessible are

exposed and others are kept for internal purpose.We can reuse existing classes from the packages as many time as we need it in our program.

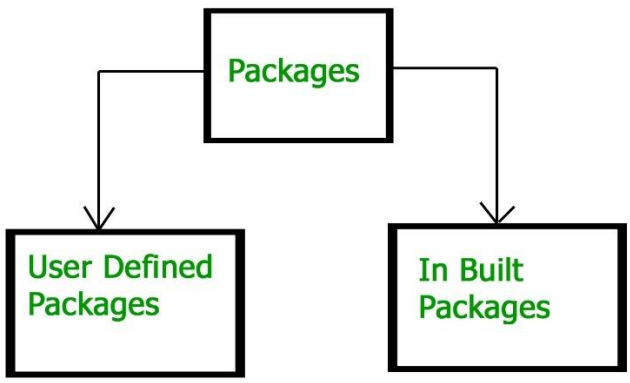
Importing Packages

To be able to use classes outside of the package you are currently working in, you need to import the package.By default, all the Java programs import the java.lang.* package, that is why we can use classes like String and Math inside the program even though we haven't imported any packages.

A package in Java is used to group related classes. Packages are divided into two categories:

Built-in Packages (packages from the Java API)

User-defined Packages (create your own packages)



How packages work?

Package names and directory structure are closely related. For example if a package name is college.staff.cse, then there are three directories, college, staff and cse such that cse is present in staff and staff is present college.

Java packages can contain subpackages. Java packages can thus make up what is called a package structure. A Java package structure is like a directory structure. Its a tree of packages,

subpackages and classes inside these classes. A Java package structure is indeed organized as directories on our hard drive, or as directories inside a zip file (JAR files).

7.2 Java API Packages

The Java API is a library of prewritten classes, that are free to use, included in the Java Development Environment.The library contains components for managing input, database programming, and much more.

The library is divided into packages and classes. Meaning we can either import a single class (along with its methods and attributes), or a whole package that contain all the classes that belong to the specified package.

7.3 Using System Packages

Some of the commonly used built-in packages are:

1) java.lang: Contains language support classes(e.g classed which defines primitive data types, math operations). This package is automatically imported.

2) java.io: Contains classed for supporting input / output operations.

3) java.util: Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.

4) java.applet: Contains classes for creating Applets.

5) java.awt: Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc).

6) java.net: Contain classes for supporting networking operations.

To use a class or a package from the library, we need to use the import keyword:

```
import package.name.Class;   // Import a single class
import package.name.*;   // Import the whole package
```

Import a class

If you find a class you want to use, for example, the Scanner class, which is used to get user input, write the following code:

Example

```
import java.util.Scanner;
```

In the example above, java.util is a package, while Scanner is a class of the java.util package.

To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read a complete line:

Example

Using the Scanner class to get user input:

```
import java.util.Scanner;

class MyClass {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);
    System.out.println("Enter username");

    String userName = myObj.nextLine();
    System.out.println("Username is: " + userName);
  }
}
```

Import a Package

There are many packages to choose from. In the previous example, we used the Scanner class from the java.util package.

This package also contains date and time facilities, random-number generator and other utility classes.

To import a whole package, end the sentence with an asterisk sign (*). The following example will import all the classes in the java.util package:

Example
import java.util.*;

7.4 Naming Convention

Package names are written in all lower case to avoid conflict with the names of classes or interfaces.

Packages in the Java language itself begin with java. or javax. Packages are named in reverse order of domain names, i.e., org.ucpes.javapractice. For example, in a college, the recommended convention is college.tech.cse, college.tech.ee, college.art.history, etc.

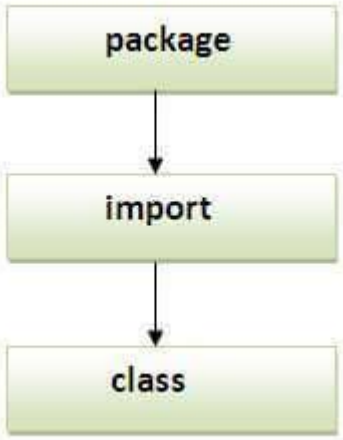
Structure of a Java Program

1.Package declaration
2.Import statement
3.Class Definition

1. Package declaration: The keyword package occurs first followed by the package name.

Example: package test;

2. Import statement : Using the import statement , we access the predefined as well as user defined packages. We can import a specific class of a package or an entire package.



7.5 Creating Packages

To create a Java package you must first create a source root directory on your hard disk. The root directory is not itself part of the package structure. The root directory contains all the Java sources that need to go into your package structure.

Once you have created a source root directory you can start adding subdirectories to it. Each subdirectory corresponds to a Java package. You can add subdirectories inside subdirectories to create a deeper package structure.

To create our own package, we write,
package <packageName>;
Java uses file system directories to store packages. The directory name must match the package name exactly.
Example: To create a package Test, First create a directory Test and under Test create the java file as

```
package Test;
public class First
{
public void view()
{
```

```
System.out.println(“ Belongs to Test package”);
}
}
```

If you want to import the Test package and access its contents in another directory, open a file say sample.java in another directory or its parent directory and type as follows:

```
import Test.*;
class sample
{
public static void main(String [] args)
{
First f = new First();
f.view();
}
}
```

Packages can also be nested. In this case, the Java interpreter expects the directory structure containing the executable classes to match the package hierarchy.

7.6 Accessing a Package

How to access package from another package?

There are three ways to access the package from outside the package.

```
import package.*;
import package.classname;
fully qualified name.
```

7.7 Using a Package

1) Using packagename.*
If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Output:Hello
2)Using packagename.classname
If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
//save by A.java
```

```
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.A;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Output:Hello

3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//using fully qualified name
   obj.msg();
  }
}
```
Output:Hello

7.8 Adding a Class to Package

Adding a class to a Package : We can add more classes to a created package by using package name at the top of the program and saving it in the package directory. We need a new java file to define a public class, otherwise we can add the new class to an existing .java file and recompile it.

Subpackages: Packages that are inside another package are the subpackages. These are not imported by default, they have to imported explicitly. Also, members of a subpackage have no access privileges, i.e., they are considered as different package for protected and default access specifiers.

Example :

```
import java.util.*;
```

util is a subpackage created inside java package.

7.9 Hiding Classes

When we import a package within a program, only the classes declared as public in that package will be made accessible within this program. In other words, the classes not declared as public in that package will not be accessible within this program.

We shall profitably make use of the above fact. Sometimes, we may wish that certain classes in a package should not be made accessible to the importing program. In such cases, we need not declare those classes as public. When we do so, those classes will be hidden from being accessed by the importing class. The class having default accessibility can be seen and used only by other classes in the same package. Note that a Java source file should contain only one public class and may include any number of non-public classes.

7.10 Static Import

The static import feature of Java 5 facilitate the java programmer to access any static member of a class directly. There is no need to qualify it by the class name.

Advantage of static import:
Less coding is required if you have access any static member of a class oftenly.
Disadvantage of static import:
If you overuse the static import feature, it makes the program unreadable and unmaintainable.
What is the difference between import and static import?
The import allows the java programmer to access classes of a package without package qualification whereas the static import feature allows to access the static members of a class without the class qualification. The import provides accessibility to classes and interface whereas static import provides accessibility to static members of the class.

UNIT-8 JAVA FILES AND I/O

8.1 What is a stream ?

Java abstracts these data from different sources and targets as "data streams". When a Java program needs to read data, it opens a stream to a data source, which can be a file, memory, or a network connection. When a Java program needs to write data, it will also open a stream to the destination. At this time, the data can be imagined as "water flowing on demand" in the pipeline. Streams provide a consistent interface for operating various physical devices. The stream is associated with the file by the open operation, and the stream is disassociated from the file by the close operation.

Stream in Java Programming Language is the sequence of the objects that are pipelined to get desired results. The Stream is not a data structure instead it just takes input from collections of I/O. There are two types of Stream in Java Programming Language:

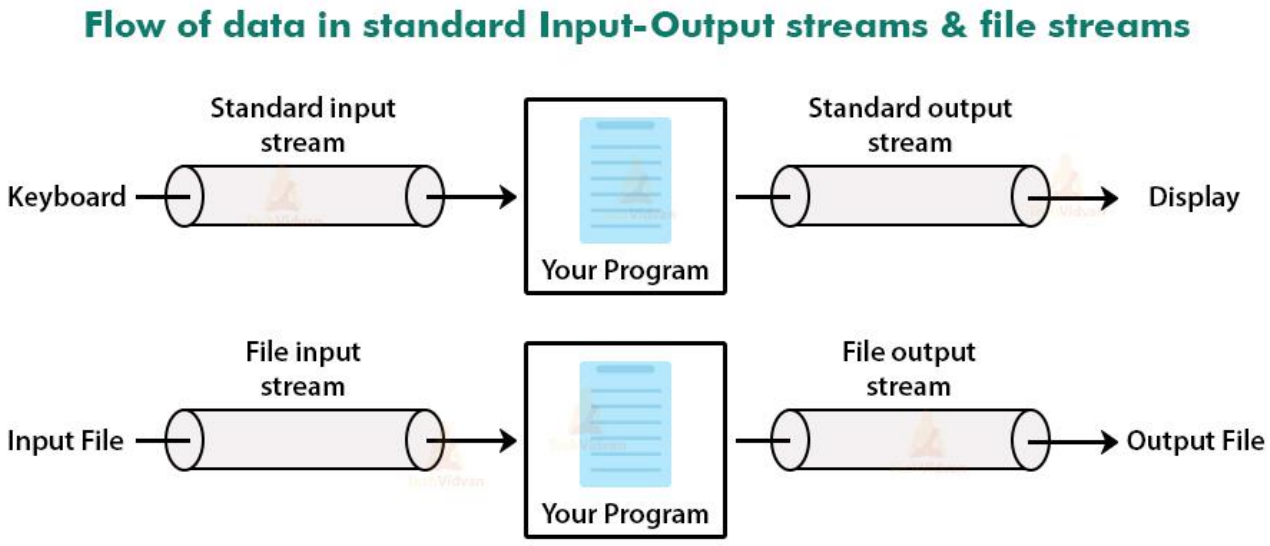
1. Input Stream:

The Input Stream provides the functionality of taking the Input from a Source. The Input Stream provides the set of Objects and pipelines together to provide us the functionality of Input the collection in I/O File.

2. Output Stream:

The Output Stream provides the functionality of writing the data to the Destination. The Output Stream provides the set of

Objects and pipelines together to provide us the functionality of Output to the collection in I/O File.



8.2 Reading and writing to files(only txt files)

Let us discuss how to read from and write to text (or character) files using classes available in the **java.io** package.

1. Reader, InputStreamReader, FileReader and BufferedReader

Reader is the abstract class for reading character streams. It implements the following fundamental methods:

read(): reads a single character.

read(char[]): reads an array of characters.

skip(long): skips some characters.

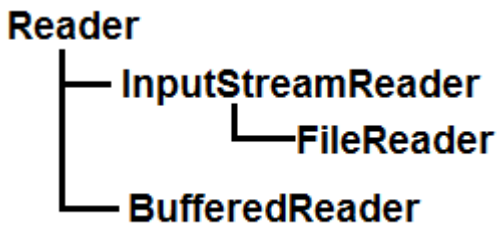
close(): closes the stream.

InputStreamReader is a bridge from byte streams to character streams. It converts bytes into characters using a specified charset. The charset can be default character encoding of the operating system, or can be specified explicitly when creating an InputStreamReader.

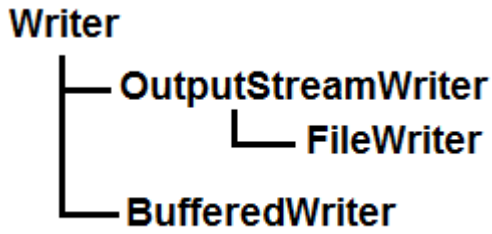
FileReader is a convenient class for reading text files using the default character encoding of the operating system.

BufferedReader reads text from a character stream with efficiency (characters are buffered to avoid frequently reading from the underlying stream) and provides a convenient method for reading a line of text readLine().
The following diagram show relationship of these reader classes in the java.io package:

Reader Hierarchy



2. Writer, OutputStreamWriter, FileWriter and BufferedWriter
Writer is the abstract class for writing character streams. It implements the following fundamental methods:
write(int): writes a single character.
write(char[]): writes an array of characters.
write(String): writes a string.
close(): closes the stream.
OutputStreamWriter is a bridge from byte streams to character streams. Characters are encoded into bytes using a specified charset. The charset can be default character encoding of the operating system, or can be specified explicitly when creating an OutputStreamWriter.
FileWriter is a convenient class for writing text files using the default character encoding of the operating system.
BufferedWriter writes text to a character stream with efficiency (characters, arrays and strings are buffered to avoid frequently writing to the underlying stream) and provides a convenient method for writing a line separator: newLine().
The following diagram show relationship of these writer classes in the java.io package:




```
FileReader reader = new FileReader("MyFile.txt");
FileWriter writer = new FileWriter("YourFile.txt");
```

8.3 Input and Output Stream

Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.In Java, 3 streams are created for us automatically. All these streams are attached with the console.

1) System.out: standard output stream
2) System.in: standard input stream
3) System.err: standard error stream

Let's see the code to print output and an error message to the console.

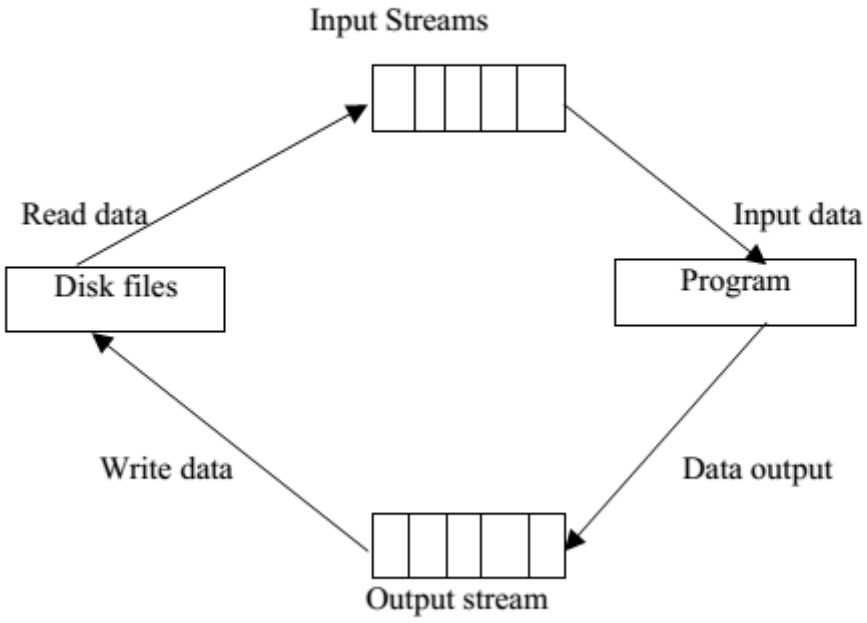
```
System.out.println("simple message");
System.err.println("error message");
```

OutputStream

Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.

InputStream

Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.



File Input and output stream

8.4 Manipulating Input data

Reading files

The process we use for reading a file is quite similar to writing a file. First, you will obtain a file channel of an object from a file stream and also use the same channel to read the data for one or more buffers. Initially, you will be using a channel object that you obtain from a FileInputStream object to read a file. Later you will be using a FileChannel object obtained from a RandomAccessFile object to read and write the same file.

A FileInputStream object encapsulates a file that is essentially intended to be read, so the file must already exist and contain some data.

Writing Files

The process for writing a file is basically quite simple. For writing a file, you will load a file that you have created as one or more

buffers and call a method for that particular object to write data to that file which is encapsulated by the file stream.

To start with, you will be using the simplest write() method for a file channel that writes the data contained in a single ByteBuffer object to a file. The number of bytes written to the file is determined by the buffers position and limit when the write() method executes.

To create a file in a specific directory, specify the path of the file and use double black slashes to escape the "\" character.
"C:\\Users\\MyName\\filename.txt"

8.5 Opening and Closing Streams

The close() method of FileOutputStream class is used to close the file output stream and releases all system resources associated with this stream.

Syntax

```
public void close();
```

Example:

```
// import java IO package
import java.io.*;
//import java utill package
import java.util.*;
public class FileOutputStreamcloseExample1
    {
   public static void main(String[] args)
           {
            FileInputStream fin= null;
      // creating new file output stream
      FileOutputStream fout= null;
      File file=null;
      Scanner sc;
```

```
        try {

            file = new File ("Fos.txt");
             file.createNewFile();
           sc=new Scanner (System.in);
           System.out.println("Enter a string  into txt file ");
           fout= new FileOutputStream(file);
           //takes an input and covert it into char array.
          char st[] = sc.nextLine().toCharArray();
         for (int i=0; i<st.length;i++)
         {//write  a string into the Fos.txt file
            fout.write(st[i]);
         }
         fin= new FileInputStream(file);
         int read;
         // read a string from the Fos.txt file.
       System.out.println("String that we  enter into the txt file”);
          while((read=fin.read())!=-1)
          {
         System.out.print((char)read);
              }
       // releases FileOutputStream resources from the streams
          fout.close();
         }
         catch(Exception e)
         {
           System.out.println(e);
         }}}
```

Output:

```
Enter a string  into txt file
UCPES.
 String that we  enter into the txt file
```

UCPES.

8.6 Predefined streams

Three Java Predefined streams or standard streams are available in the java.lang.System class. These are as follows:

System.in	This is the standard stream for input. This stream is used for reading data for the program from the keyboard by default.
System.out	This is the standard stream for output. This stream is used for writing data from the program to an output device such as a monitor / console by default or to some specified file.
System.err	This is a standard stream for error. This is used to show an error message on the screen i.e. console by default for the users.

System.in is an object of InputStream. On the other hand, System.out and System.err are both an object of type OutputStream.

All these Java Predefined Streams are automatically initialized by Java's JVM (Java Virtual Machine) on startup. All these streams are byte streams but these are used to read and write character streams as well.

Java Predefined Stream for Standard Input

System.in is the stream used for standard input in Java. This stream is an object of InputStream stream. In Java you need to

wrap the System.in in the BufferedReader object in order to obtain a character based stream.
This is done with the following code:
BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
Here, InputStreamReader is the input stream you are creating and br is the character based stream that will be linked through System.in. Now, br object can be used for reading inputs from users.
In order to understand how this stream is used, let us have a look at an **example**.

```
import java.io.*;
class InputEg
{
        public static void main( String args[] ) throws IOException
        {
                String city;
                BufferedReader br = new BufferedReader ( new InputStreamReader (System.in) );
                System.out.println ( “Where do you live?" );
                city = br.readLine();
                System.out.println ( “You have entered " + city + " city" );
        }
}
```

Java Predefined Stream for Standard Output
System.out is an object of the PrintStream stream. This stream is the stream used for standard output in Java. The output of this stream is directed to the console by default by the program. Have a look at the above example.

The statement System.out.println(“You have entered ” + city + ” city”); of the code will direct the output of the program on the console of the user.

Java Predefined Stream for Standard Error
System.err is the stream used for standard error in Java. This stream is an object of PrintStream stream.

System.err is similar to System.out but it is most commonly used inside the catch section of the try / catch block. A sample of the block is as follows:

```
try {

// Code for execution

}
catch ( Exception e)
 {
System.err.println ( “Error in code: ” + e );
}
```

8.7 File handling Classes and Methods
OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.
Example:

```
File f = new File("C:/java/hello");
OutputStream f = new FileOutputStream(f);
```

InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.

Example:
File f = new File("C:/java/hello");
InputStream f = new FileInputStream(f);

Methods

canread() – It is a Boolean type and it says whether the file which we want is readable or not.

canwrite() – It is a Boolean type and it says whether the file which we want is writable or not.

createnewfile() – It is a Boolean type that says that the method creates an empty file.

delete() – It is a Boolean type which deletes a file which we want.

exist() – It is a Boolean type and says whether the file exists or not.

getname() – It is a string type which returns the file name or name of the file.

getabsolutepath() – It is a string type which returns the correct path of name of the file.

length() – It is a long type which returns the wanted size of the file in bytes.

list() – It is a string type which returns the array of the file in the directory.

mkdir() – It is a Boolean type which creates a directory.

UNIT-9 EXCEPTION HANDLING

9.1 Exceptions Overview

An Exception is an event that occurs during the execution of a program that disrupts the normal flow of instruction. Exception Handling is the mechanism to handle runtime malfunctions. We need to handle such exceptions to prevent abrupt termination of a program. The term exception means exceptional condition, it is a problem that may arise during the execution of program. A bunch of events can lead to exceptions, including programmer error, hardware failures, files that need to be opened cannot be found, resource exhaustion etc.

Java Exception

A Java Exception is an object that describes the exception that occurs in a program. When an exceptional events occurs in java, an exception is said to be thrown. The code that's responsible for doing something about the exception is called an exception handler.

An exception is an event that interrupts the normal processing flow of a program. This event is usually some error of some sort. This causes our program to terminate abnormally.

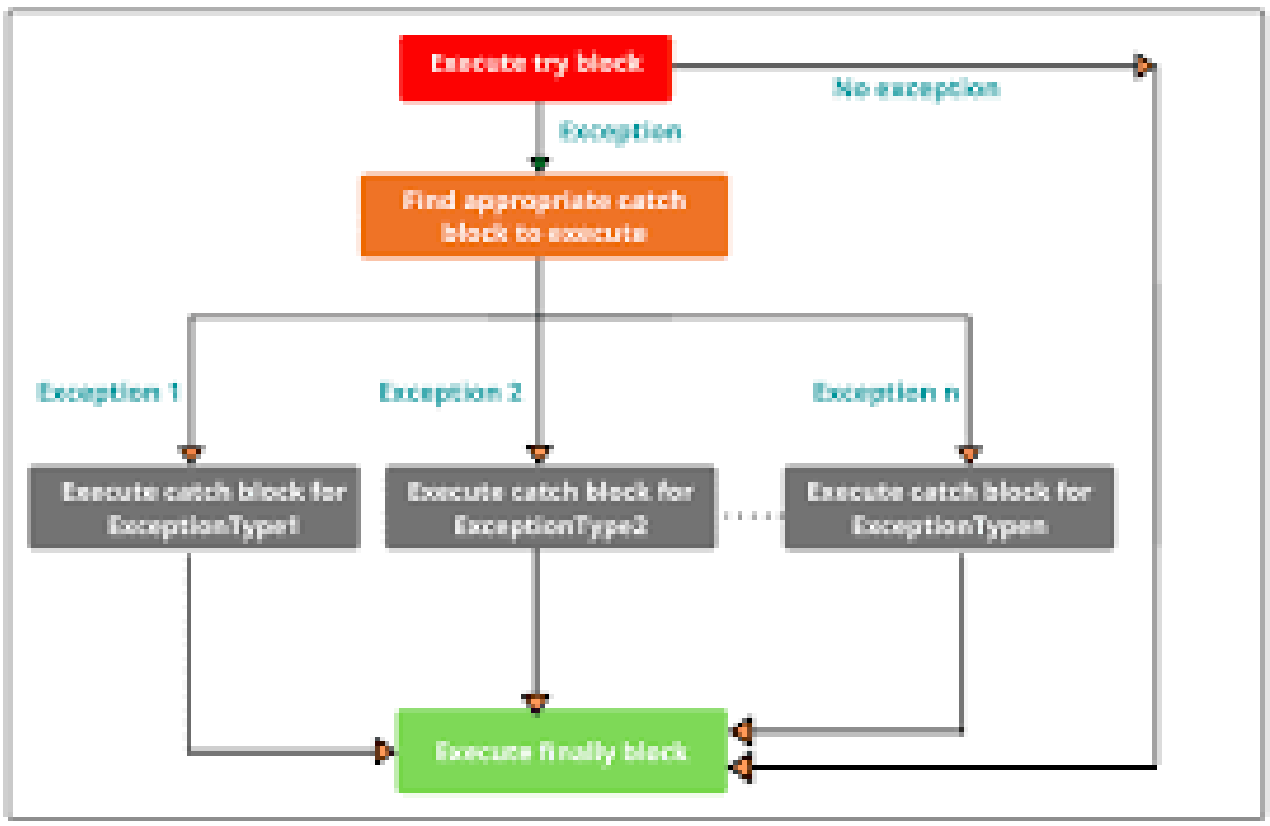
Some examples of exceptions are: ArrayIndexOutOfBounds exceptions, which occurs if we try to access a non-existent array element, or maybe a NumberFormatException, which occurs when we try to pass as a parameter a non-number in the Integer.parseInt method.

9.2 Exception Keywords

In java, exception handling is done using five keywords,

- try
- catch
- throw
- throws
- finally

A block of code that catches the exception thrown by JVM is called exception handler. Exception handling is done by transferring the execution of a program to an appropriate exception handler when exception occurs.



9.3 Catching Exceptions

Using try and catch

Try is used to guard a block of code in which exception may occur. This block of code is called guarded region or try block. A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in the try block , the catch block that follows the try is checked, if the type of exception that occured is listed in the catch block then the exception is handed over to the catch block which then handles it. A try block must have at least one catch block or finally that follows it immediately.

The three possible forms of try block are as follows:

try-catch: A try block is always followed by one or more catch blocks.

try-finally: A try block followed by a finally block.

try-catch-finally: A try block followed by one or more catch blocks followed by a finally block.

Syntax for try-catch-finally block:

```
try
{
statement1;
statement2;
}
```

Unlike catch, multiple finally blocks cannot be declared with a single try block. That is there can be only one finally clause with a single try block.

```
catch(Exceptiontype e1)
{
statement3;
}
statement4;
catch( <exceptionType1> <varName1> ){
//write the action your program will do if an exception
//of a certain type occurs

}
. . .
catch( <exceptionTypen> <varNamen> ){
//write the action your program will do if an
//exception of a certain type occurs

}
```

9.4 Using Finally Statement

A 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 exception has occurred. Using a finally block, lets you run any cleanup type statements that you want to execute, no matter what happens in the protected code. A finally block appears at the end of catch block.

```
finally
{
statement5;
}
```

Some important rules of using finally block or clause are:

1. A finally block is optional but at least one of the catch or finally block must exist with a try.

2. It must be defined at the end of last catch block. If finally block is defined before a catch block, the program will not compile successfully.

The code within the finally block is always get executed whether the exception is thrown by try block or not. If an exception is thrown with matching catch block, the first catch block is executed, and then finally block is executed.

On the other hand, if no matching catch block is found with an exception object thrown by try block, finally block is executed by JVM after the execution of try block and the program terminates.

9.5 Exception Methods

Java defines several types of exceptions that relate to its various class libraries. Java also allows users to define their own exceptions. Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

ArithmeticException

It is thrown when an exceptional condition has occurred in an arithmetic operation.

ArrayIndexOutOfBoundsException

It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

ClassNotFoundException

This Exception is raised when we try to access a class whose definition is not found

FileNotFoundException

This Exception is raised when a file is not accessible or does not open.

IOException

It is thrown when an input-output operation failed or interrupted

InterruptedException

It is thrown when a thread is waiting, sleeping, or doing some processing, and it is interrupted.

NoSuchFieldException

It is thrown when a class does not contain the field (or variable) specified

NoSuchMethodException

It is thrown when accessing a method which is not found.

NullPointerException

This exception is raised when referring to the members of a null object. Null represents nothing

NumberFormatException

This exception is raised when a method could not convert a string into a numeric format.

RuntimeException

This represents any exception which occurs during runtime.

StringIndexOutOfBoundsException

It is thrown by String class methods to indicate that an index is either negative or greater than the size of the string

Example on Arithmetic exception

```
// Java program to demonstrate ArithmeticException
class ArithmeticException_Demo
{
    public static void main(String args[])
    {
        try {
            int a = 30, b = 0;
            int c = a/b;  // cannot divide by zero
            System.out.println ("Result = " + c);
```

```
        }
        catch(ArithmeticException e) {
           System.out.println ("Can't divide a number by 0");
        }
    }
}
```

Output:

```
Can't divide a number by 0
```

Example on FileNotFound Exception

```
//Java program to demonstrate FileNotFoundException
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
 class File_notFound_Demo {

    public static void main(String args[])  {
        try {

            // Following file does not exist
            File file = new File("E://file.txt");

            FileReader fr = new FileReader(file);
        } catch (FileNotFoundException e) {
           System.out.println("File does not exist");
        }
    }
}
```

Output:

```
File does not exist
```

Example on ArrayIndexOutOfBounds Exception

```
class ArrayIndexOutOfBound_Demo
{
   public static void main(String args[])
   {
      try{
         int a[] = new int[5];
         a[6] = 9; // accessing 7th element in an array of
                  // size 5
      }
      catch(ArrayIndexOutOfBoundsException e){
         System.out.println ("Array Index is Out Of Bounds");
      }
   }
}
```

Output:

Array Index is Out Of Bounds

9.6 Declaring Exceptions

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

Following steps are followed for the creation of user-defined Exception.

The user should create an exception class as a subclass of Exception class. Since all the exceptions are subclasses of Exception class, the user should also make his class a subclass of it. This is done as:

class MyException extends Exception

We can write a default constructor in his own exception class.

MyException(){}
We can also create a parameterized constructor with a string as a parameter.
We can use this to store exception details. We can call super class(Exception) constructor from this and send the string there.

```
MyException(String str)
{
  super(str);
}
```

9.7 Defining and throwing exceptions

o raise exception of user-defined type, we need to create an object to his exception class and throw it using throw clause, as:

```
MyException me = new MyException("Exception details");
throw me;
```

The following program illustrates how to create own exception class MyException.
Details of account numbers, customer names, and balance amounts are taken in the form of three arrays.
In main() method, the details are displayed using a for-loop. At this time, check is done if in any account the balance amount is less than the minimum balance amount to be apt in the account. If it is so, then MyException is raised and a message is displayed "Balance amount is less".

```
// Java program to demonstrate user defined exception

// This program throws an exception whenever balance
// amount is below Rs 1000
class MyException extends Exception
{
```

```
//store account information
private static int accno[] = {1001, 1002, 1003, 1004};

private static String name[] =
          {"Nish", "Shubh", "Sush", "Abhi", "Akash"};

private static double bal[] =
    {10000.00, 12000.00, 5600.0, 999.00, 1100.55};

// default constructor
MyException() {    }

// parameterized constructor
MyException(String str) { super(str); }

// write main()
public static void main(String[] args)
{
   try  {
      // display the heading for the table
      System.out.println("ACCNO" + "\t" + "CUSTOMER" +
                              "\t" + "BALANCE");

      // display the actual account information
      for (int i = 0; i < 5 ; i++)
      {
         System.out.println(accno[i] + "\t" + name[i] +
                                 "\t" + bal[i]);

         // display own exception if balance < 1000
         if (bal[i] < 1000)
         {
```

```
                MyException me =
                  new MyException("Balance is less than 1000");
                throw me;
              }
            }
        } //end of try

        catch (MyException e) {
           e.printStackTrace();
        }
    }
}
```

RunTime Error

```
 MyException: Balance is less than 1000
   at MyException.main(fileProperty.java:36)
```

Output:

```
ACCNO    CUSTOMER    BALANCE
1001    Nish    10000.0
1002    Shubh    12000.0
1003    Sush    5600.0
1004    Abhi    999.0
```

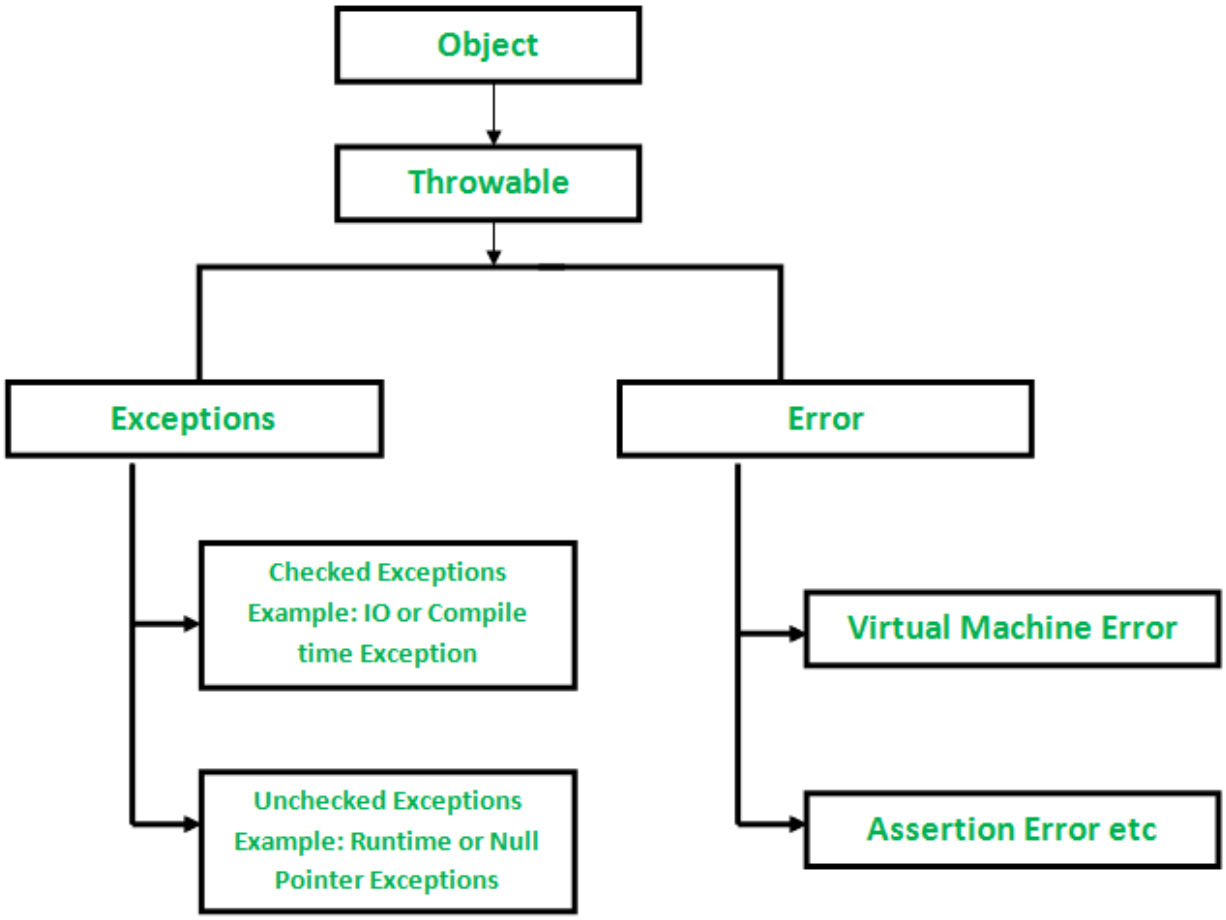
9.8 Errors and Runtime Exceptions

In java, both Errors and Exceptions are the subclasses of java.lang.Throwable class. Error refers to an illegal operation performed by the user which results in the abnormal working of the program. Programming errors often remain undetected until the program is compiled or executed. Some of the errors inhibit the program from getting compiled or executed. Thus

errors should be removed before compiling and executing. It is of three types:

- Compile-time
- Run-time
- Logical

Whereas exceptions in java refer to 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.



Errors	Exceptions
Recovering from Error is not possible.	We can recover from exceptions by either using try-

Errors	Exceptions
	catch block or throwing exceptions back to the caller.
All errors in java are unchecked type.	Exceptions include both checked as well as unchecked type.
Errors are mostly caused by the environment in which program is running.	Program itself is responsible for causing exceptions.
Errors can occur at compile time as well as run time. Compile Time: eg Syntax Error Run Time: Logical Error.	All exceptions occurs at runtime but checked exceptions are known to the compiler while unchecked are not.
They are defined in java.lang.Error package.	They are defined in java.lang.Exception package
Examples : java.lang.StackOverflowError, java.lang.OutOfMemoryError	Examples : Checked Exceptions : SQLException, IOException Unchecked Exceptions : ArrayIndexOutOfBoundException, NullPointerException, ArithmeticException.

Reference Books:

01 Programming With Java A Primer By E. Balagurusami
The McGraw-Hill Companies

02 JavaTM 2: The Complete Reference By Patric Naughton and Herbert Schildt
Tata McGraw-Hill Publishing Company Limited

03 Core Java For Beginners By Rashmi Kanta Das
Vikas Publishing

04 Java: A Beginner‟s Guide
By Herbert Schildt
McGraw-Hill Education

05 Core Java Volume I - Fundamentals
By Cay S. Horstmann
Prentice Hall

MODEL QUESTIONS

1.Questions carrying two marks each.

a. What is bytecode?
b. What is a Literal? What are the different types of literals?
c. What are token in Java?
d. What is a variable? What are the different types of variables?
e. What are the difference between static variable and instance variable?
f. Define Array? How to declare an array?
g. List out the operator in Java.
h. What is a class? Give an example?
i. Define method overloading.
j. What are the uses of the keyword ‘final’.
k. Define inheritance.
l. Define package.

2.Questions carrying five marks each.

a. Write a Java program to illustrate the use of constructors.
b. Write short notes on method overloading.
c. Explain the various access specifiers used in Java.
d. What is an exception? How exception is handled in java?
e. How does String class differ from the StringBuffer class?
f. Differentiate between static binding and dynamic binding.

3.Questions carrying ten marks each.

a. Discuss the control structures available in Java with an example.
b. Explain the concept of inheritance in Java with an example.
c. Explain the various features available in Java.
d. Describe inheritance in detail with example.
e. What is a package. Write a program in Java using package.
f. Discuss the various methods available in String class with examples.
g. Define File. Write a program in Java to create a file, write into it and read the file to output in the screen.