

Semester-3

# Java Programming

(According to Purvanchal University Syllabus)

# Unit – 1

## Introduction to Java

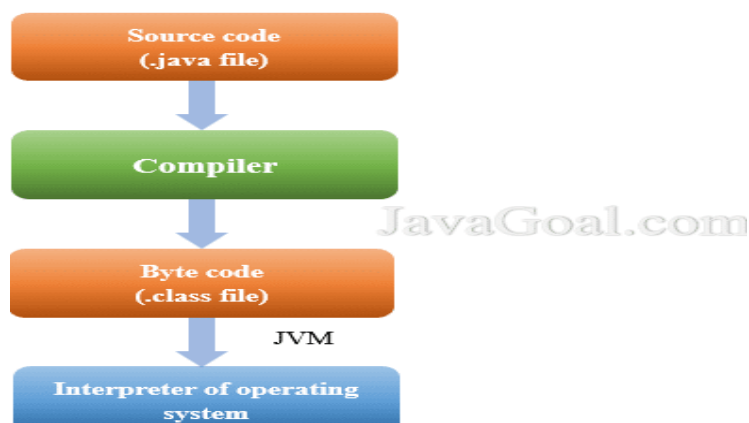
- JAVA was developed by Sun Microsystems Inc in 1991, later acquired by Oracle Corporation. It was developed by James Gosling and Patrick Naughton. It is a simple programming language. Writing, compiling and debugging a program is easy in java. It helps to create modular programs and reusable code.

### Java terminology

- Before we start learning Java, let's get familiar with common java terms.

### Java Virtual Machine (JVM)

- This is generally referred to as JVM. Before, we discuss about JVM let's see the phases of program execution. Phases are as follows: we write the program, then we compile the program and at last we run the program.
  - 1) Writing of the program is of course done by java programmer like you and me.
  - 2) Compilation of program is done by java compiler, javac is the primary java compiler included in java development kit (JDK). It takes java program as input and generates java byte code as output.
  - 3) In third phase, JVM executes the byte code generated by compiler. This is called program run phase.



## bytecode

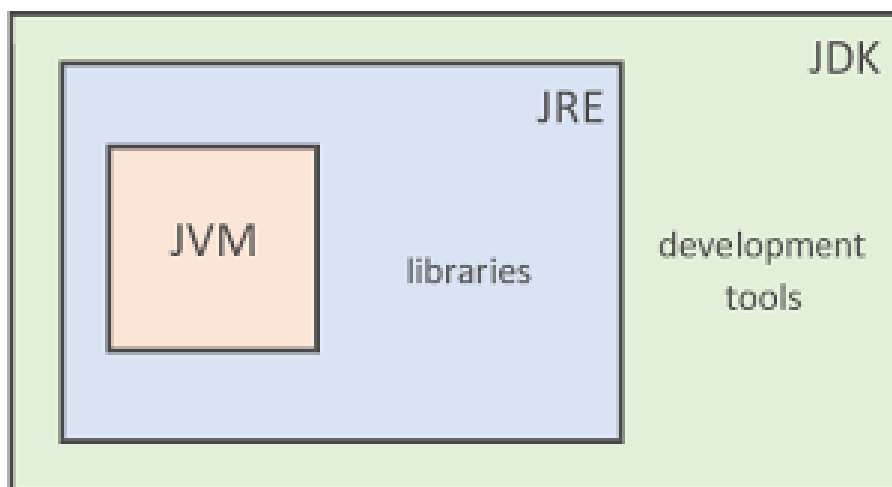
As discussed above, javac compiler of JDK compiles the java source code into bytecode so that it can be executed by JVM. The bytecode is saved in a .class file by compiler.

## Java Development Kit(JDK)

While explaining JVM and bytecode, I have used the term JDK. Let's discuss about it. As the name suggests this is complete java development kit that includes JRE

(Java Runtime Environment), compilers and various tools like JavaDoc, Java debugger etc.

In order to create, compile and run Java program you would need JDK installed on your computer.



## Java Runtime Environment (JRE)

JRE is a part of JDK which means that JDK includes JRE. When you have JRE installed on your system, you can run a java program however you won't be able to compile it. JRE includes JVM, browser plugging and applets support. When you only need to run a java program on your computer, you would only need JRE.

## Importance and Features of Java–

### Main Features of JAVA

1. Java is a platform independent language

- Compiler (javac) converts source code (.java file) to the byte code (.class file). As mentioned above, JVM executes the byte code produced by compiler. This byte code can run on any platform such as Windows, Linux, and Mac OS etc. Which means a program that is compiled on windows can run on Linux and vice-versa?
- Each operating system has different JVM; however the output they produce after execution of byte code is same across all operating systems. That is why we call java as platform independent language.

## 2. Java is an Object Oriented language

- Object oriented programming is a way of organizing programs as collection of objects, each of which represents an instance of a
- Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

### Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

**Object-** An entity that has state and behavior is known as an object e.g. chairs, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

**Class-** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

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

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

**Polymorphism-** Polymorphism in Java is a concept by which we can perform a single action in different ways. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

**Abstraction-** Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

**Encapsulation-** Encapsulation in Java is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.

### 3. Simple

- Java is considered as one of simple language because it does not have complex features like Operator overloading, multiple inheritance, pointers and Explicit memory allocation.

### 4. Robust Language

- Robust means reliable. Java programming language is developed in a way that puts a lot of emphasis on early checking for possible errors, that's why java compiler is able to detect errors that are not easy to detect in other programming languages. The main features of java that makes it robust are garbage collection, Exception Handling and memory allocation.

### 5. Secure

- We don't have pointers and we cannot access out of bound arrays (you get `ArrayIndexOutOfBoundsException` if you try to do so) in

java. That's why several security flaws like stack corruption or buffer overflow is impossible to exploit in Java.

## 6. Java is distributed

- Using java programming language we can create distributed applications. RMI (Remote Method Invocation) and EJB (Enterprise Java Beans) are used

for creating distributed applications in java. In simple words: The java programs can be distributed on more than one systems that are connected to each other using internet connection. Objects on one JVM (java virtual machine) can execute procedures on a remote JVM.

## 7. Multithreading

- Java supports multithreading. Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU.

## 8. Portable

- As discussed above, java code that is written on one machine can run on another machine. The platform independent byte code can be carried to any platform for execution that makes java code portable.

## Keywords–

- Keywords are words that have already been defined for Java compiler. They have special meaning for the compiler. Java Keywords must be in your information because you cannot use them as a variable, class or a method name.

### Some Examples of keywords:

- **abstract:** It is a non-access modifier applicable for classes and methods. It is used to achieve abstraction. For more, refer abstract keyword in java
- **enum :** It is used to define enum in Java
- **instanceof:** It is used to know whether the object is an instance of the specified type (class or subclass or interface).
- **private:** It is an access modifier. Anything declared private cannot be seen outside of its class.
- **protected :** If you want to allow an element to be seen outside your

current package, but only to classes that subclass your class directly, then declare that element protected.

- **public:** Anything declared public can be accessed from anywhere.

## Constants–

- Constants in Java. A constant is a variable which cannot have its value changed after declaration. It uses the 'final' keyword.

### Syntax

```
modifier final dataType variableName = value; //global constant
```

```
modifier static final dataType variableName = value; //constant within a class
```

## Variables and Data type–

A variable is a name which is associated with a value that can be changed. For example when I write `int i=10;` here variable name is `i` which is associated with value 10, `int` is a data type that represents that this variable can hold integer values. We will cover the data types in the next tutorial. In this tutorial, we will discuss about variables.

### How to Declare a variable in Java

To declare a variable follow this syntax:

```
data_type variable_name = value;
```

here value is optional because in java, you can declare the variable first and then later assign the value to it.

**For example:** Here `num` is a variable and `int` is a data type. We will discuss the data type in next tutorial so do not worry too much about it, just understand that `int` data type allows this `num` variable to hold integer values. You can read data types here but I would recommend you to finish reading this guide before proceeding to the next one.

```
int num;
```

Similarly we can assign the values to the variables while declaring them, like this:

```
char ch = 'A';
```

```
int number = 100;
```

like this:

```
char ch;
```

```
int number;
...
ch = 'A';
number = 100;
```

## Types of Variables in Java

There are three types of variables in Java.

1) Local variable 2) Static (or class) variable 3) Instance variable

### Static (or class) Variable

Static variables are also known as class variable because they are associated with the class and common for all the instances of class. For example, If I create three objects of a class and access this static variable, it would be common for all, the changes made to the variable using one of the object would reflect when you access it through other objects.

### Instance variable

Each instance (objects) of class has its own copy of instance variable. Unlike static variable, instance variables have their own separate copy of instance variable. We have changed the instance variable value using object obj2 in the following program and when we displayed the variable using all three objects, only the obj2 value got changed, others remain unchanged. This shows that they have their own copy of instance variable.

### Local Variable

These variables are declared inside method of the class. Their scope is limited to the method which means that You can't change their values and access them outside of the method.

## Data types

Data type defines the values that a variable can take, for example if a variable has int data type, it can only take integer values. In java we have two categories of data type: 1) Primitive data types 2) Non-primitive data types – Arrays and Strings are non-primitive data types.

Java is a statically typed language. A language is statically typed, if the data type of a variable is known at compile time. This means that you must specify the type of the variable (Declare the variable) before you can use it.



## 1) Primitive data types

In Java, we have eight primitive data types: boolean, char, byte, short, int, long, float and double. Java developers included these data types to maintain the portability of java as the size of these primitive data types do not change from one operating system to another.

byte, short, int and long data types are used for storing whole

numbers. float and double are used for fractional numbers.

char is used for storing characters(letters).

boolean data type is used for variables that holds either true or false.

## 2) No primitive data types

Non-primitive, or reference data types, are the more sophisticated members of the data type family. They don't store the value, but store a reference to that value. Instead of partNumber 4030023, Java keeps the reference, also called address, to that value, not the value itself.

## Operators and expressions–

An operator is a character that represents an action, for example + is an arithmetic operator that represents addition.

## Types of Operator in Java

- 1) Basic Arithmetic Operators
- 2) Assignment Operators
- 3) Auto-increment and Auto-decrement Operators
- 4) Logical Operators
- 5) Comparison (relational) operators
- 6) Bitwise Operators
- 7) Ternary Operator

### 1) Basic Arithmetic Operators

Basic arithmetic operators are: +, -, \*, /, %

+ is for addition.

- is for subtraction.

\* is for multiplication.

/ is for division.

% is for modulo.

Note: Modulo operator returns remainder, for example 10 % 5 would

return 0 Example of Arithmetic Operators

```
public class ArithmeticOperatorDemo {
    public static void main(String args[]) {
        int num1 = 100;
        int num2 = 20;

        System.out.println("num1 + num2: " + (num1 + num2) );
        System.out.println("num1 - num2: " + (num1 - num2) );
        System.out.println("num1 * num2: " + (num1 * num2) );
        System.out.println("num1 / num2: " + (num1 / num2) );
        System.out.println("num1 % num2: " + (num1 % num2) );
    }
}
```

Output:

```
num1 + num2: 120
num1 - num2: 80
num1 * num2: 2000
num1 / num2: 5
num1 % num2: 0
```

## 2) Assignment Operators

Assignments operators in java are: =, +=, -=, \*=, /=, %=

num2 = num1 would assign value of variable num1 to the

variable. num2+=num1 is equal to num2 = num2+num1

num2-=num1 is equal to num2 = num2-num1

$\text{num2} *= \text{num1}$  is equal to  $\text{num2} = \text{num2} * \text{num1}$

$\text{num2} /= \text{num1}$  is equal to  $\text{num2} = \text{num2} / \text{num1}$

$\text{num2} \% = \text{num1}$  is equal to  $\text{num2} = \text{num2} \% \text{num1}$

## Example of Assignment Operators

```
public class AssignmentOperatorDemo {
    public static void main(String args[]) {
        int num1 = 10;
        int num2 = 20;
```

```
        num2 = num1;
        System.out.println("= Output: "+num2);
```

```
        num2 += num1;
        System.out.println("+= Output: "+num2);
```

```
        num2 -= num1;
        System.out.println("-= Output: "+num2);
```

```
        num2 *= num1;
        System.out.println("*= Output: "+num2);
```

```
        num2 /= num1;
        System.out.println("/= Output: "+num2);
```

```
        num2 %= num1;
        System.out.println("%= Output: "+num2);
    }
}
```

Output:

```
= Output: 10
+= Output: 20
-= Output: 10
*= Output: 100
/= Output: 10
%= Output: 0
```

### 3) Auto-increment and Auto-decrement Operators

++ and --

num++ is equivalent to `num=num+1;`

num-- is equivalent to `num=num-1;`

Example of Auto-increment and Auto-decrement Operators

```
public class AutoOperatorDemo {

    public static void main(String args[]){
        int num1=100;
        int num2=200;
        num1++;
        num2--;
        System.out.println("num1++ is: "+num1);
        System.out.println("num2-- is: "+num2);
    }
}
```

Output:

```
num1++ is: 101
num2-- is: 199
```

### 4) Logical Operators

Logical Operators are used with binary variables. They are mainly used in conditional statements and loops for evaluating a condition.

Logical operators in java are: &&, ||, !

Let's say we have two boolean variables b1 and b2.

b1&&b2 will return true if both b1 and b2 are true else it would return

false. b1||b2 will return false if both b1 and b2 are false else it would return true.

!b1 would return the opposite of b1, that means it would be true if b1 is

false and it would return false if b1 is true.

## Example of Logical Operators

```
public class LogicalOperatorDemo {
    public static void main(String args[]) {
        boolean b1 = true;
        boolean b2 = false;

        System.out.println("b1 && b2: " + (b1&&b2));
        System.out.println("b1 || b2: " + (b1||b2));
        System.out.println("!(b1 && b2): " + !(b1&&b2));
    }
}
```

Output:

```
b1 && b2: false
b1 || b2: true
!(b1 && b2): true
```

## 5) Comparison(Relational) operators

We have six relational operators in Java: ==, !=, >, <, >=, <=

== returns true if both the left side and right side are equal

!= returns true if left side is not equal to the right side of

operator. > returns true if left side is greater than right.

< returns true if left side is less than right side.

>= returns true if left side is greater than or equal to right

side. <= returns true if left side is less than or equal to

right side.

## Example of Relational operators

```
public class RelationalOperatorDemo {
    public static void main(String args[]) {
        int num1 = 10;
```

```
int num2 = 50;
if (num1==num2) {
    System.out.println("num1 and num2 are equal");
}
else{
```

```
    System.out.println("num1 and num2 are not equal"); }
}
```

```
if( num1 != num2 ){
    System.out.println("num1 and num2 are not equal"); }
else{
    System.out.println("num1 and num2 are equal");
}
```

```
if( num1 > num2 ){
    System.out.println("num1 is greater than num2");
}
else{
    System.out.println("num1 is not greater than num2"); }
```

```
if( num1 >= num2 ){
    System.out.println("num1 is greater than or equal to num2"); }
else{
    System.out.println("num1 is less than num2");
}
```

```
if( num1 < num2 ){
    System.out.println("num1 is less than num2");
}
else{
    System.out.println("num1 is not less than num2");
}
```

```
if( num1 <= num2){
    System.out.println("num1 is less than or equal to num2"); }
else{
    System.out.println("num1 is greater than num2");
}
```

```
}
}
```

Output:

num1 and num2 are not equal

```

num1 and num2 are not equal
num1 is not greater than num2
num1 is less than num2
num1 is less than num2
num1 is less than or equal to num2

```

## 6) Bitwise Operators

There are six bitwise Operators: &, |, ^, ~, <<, >>

```

num1 = 11; /* equal to 00001011*/
num2 = 22; /* equal to 00010110 */

```

Bitwise operator performs bit by bit processing.

num1 & num2 compares corresponding bits of num1 and num2 and generates 1 if both bits are equal, else it returns 0. In our case it would return: 2 which is 00000010 because in the binary form of num1 and num2 only second last bits are matching.

num1 | num2 compares corresponding bits of num1 and num2 and generates 1 if either bit is 1, else it returns 0. In our case it would return 31 which is 00011111

num1 ^ num2 compares corresponding bits of num1 and num2 and generates 1 if they are not equal, else it returns 0. In our example it would return 29 which is equivalent to 00011101

~num1 is a complement operator that just changes the bit from 0 to 1 and 1 to 0. In our example it would return -12 which is signed 8 bit equivalent to 11110100

num1 << 2 is left shift operator that moves the bits to the left, discards the far left bit, and assigns the rightmost bit a value of 0. In our case output is 44 which is equivalent to 00101100

num1 >> 2 is right shift operator that moves the bits to the right, discards the far right bit, and assigns the leftmost bit a value of 0. In our case output is 2 which is equivalent to 00000010

### Example of Bitwise Operators

```

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

```

```
int num1 = 11; /* 11 = 00001011 */
int num2 = 22; /* 22 = 00010110 */
int result = 0;
```

```
result = num1 & num2;
System.out.println("num1 & num2: "+result);
```

```
result = num1 | num2;
System.out.println("num1 | num2: "+result);
```

```
result = num1 ^ num2;
System.out.println("num1 ^ num2: "+result);
```

```
result = ~num1;
System.out.println("~num1: "+result);
```

```
result = num1 << 2;
System.out.println("num1 << 2: "+result); result = num1 >>
2; System.out.println("num1 >> 2: "+result);
}
}
```

Output:

```
num1 & num2: 2
num1 | num2: 31
num1 ^ num2: 29
~num1: -12
num1 << 2: 44 num1 >> 2: 2
```

## 7) Ternary Operator

This operator evaluates a boolean expression and assign the value based on the result.

Syntax:

```
variable num1 = (expression) ? value if true : value if false
```

If the expression results true then the first value before the colon (:) is assigned to the variable num1 else the second value is assigned to the num1.

Example of Ternary Operator



```

public class TernaryOperatorDemo {

    public static void main(String args[]) {
        int num1, num2;
        num1 = 25;
        /* num1 is not equal to 10 that's why
        * the second value after colon is assigned
        * to the variable num2
        */
        num2 = (num1 == 10) ? 100: 200;
        System.out.println( "num2: "+num2);

        /* num1 is equal to 25 that's why
        * the first value is assigned
        * to the variable num2
        */
        num2 = (num1 == 25) ? 100: 200;
        System.out.println( "num2: "+num2);
    }
}

```

Output:

```

num2: 200
num2: 100

```

## Operator Precedence in Java

This determines which operator needs to be evaluated first if an expression has more than one operator. Operator with higher precedence at the top and lower precedence at the bottom.

Unary Operators

++ -- ! ~

Multiplicative

\* / %

Additive

+ -

Shift

<< >> >>>

Relational

> >= < <=

Equality

== !=

Bitwise AND

&

Bitwise XOR

^

Bitwise OR

|

Logical AND

&&

Logical OR

||

Ternary

?:

Assignment

= += -= \*= /= %= > >= < <= &= ^= |=

Expressions

- Expressions are essential building blocks of any Java program, usually created to produce a new value, although sometimes an expression simply assigns a value to a variable. Expressions are built using values, variables, operators and method calls.
- An expression is a statement that can convey a value. Some of the most common expressions are mathematical, such as in the following source code example:

```
int x = 3;
```

```
int y = x;
```

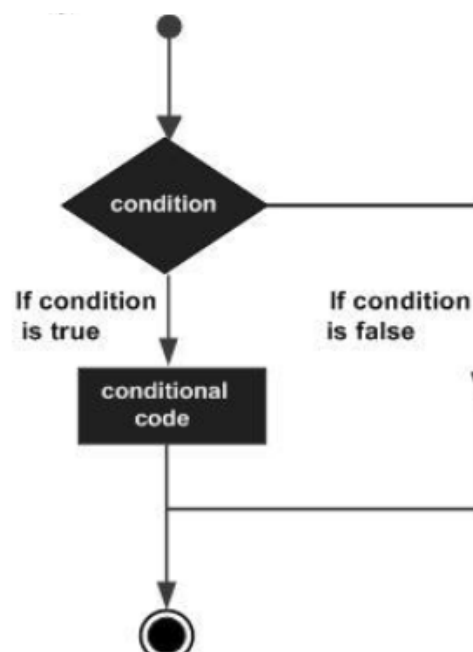
```
int z = x * y;
```

All three of these statements can be considered expressions—they convey values that can be assigned to variables. The first assigns the literal 3 to the variable x. The second assigns the value of the variable x to the variable y. The multiplication operator \* is used to multiply the x and y integers, and the expression produces the result of the multiplication. This result is stored in the z integer.

## **Decision making–**

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.

Following is the general form of a typical decision making structure found in most of the programming languages –



Java programming language provides following types of decision making statements. Click the following links to check their detail.

## 1 if statement

An if statement consists of a boolean expression followed by one or more statements.

// Java program to illustrate If statement without curly block

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

```
class IfDemo {
```

```
public static void main(String args[])
```

```
{
```

```
    int i = 10;
```

```
    if (i < 15)
```

```
        System.out.println("Inside If block"); // part of if block(immediate
one statement after if condition)
```

```
        System.out.println("10 is less than 15"); //always executes as it
is outside of if block
```

```
        // This statement will be executed
```

```
        // as if considers one statement by default again below statement is
outside of if block
```

```
        System.out.println("I am Not in if");
```

```
    }
```

```
}
```

## **2 if...else statement**

An if statement can be followed by an optional else statement, which executes when the boolean expression is false.

// Java program to illustrate if-else statement

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

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

        if (i < 15)
            System.out.println("i is smaller than 15");
        else
            System.out.println("i is greater than 15");
    }
}
```

## **3) nested if statement**

You can use one if or else if statement inside another if or else if statement(s).

// Java program to illustrate nested-if statement

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

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

```

        int i = 10;

        if (i == 10 || i<15) {
            // First if statement
            if (i < 15)
                System.out.println("i is smaller than 15");

            // Nested - if statement
            // Will only be executed if statement above
            // it is true
            if (i < 12)
                System.out.println(
                    "i is smaller than 12 too");
        } else{
            System.out.println("i is greater than 15");
        }
    }
}

```

#### **4) switch statement**

*/\*package whatever //do not write package name here \*/*

```

import java.io.*;

class GFG {
    public static void main (String[] args) {
        int num=20;
        switch(num){
            case 5 : System.out.println("It is 5");
                    break;
            case 10 : System.out.println("It is 10");
                    break;
            case 15 : System.out.println("It is 15");
                    break;
            case 20 : System.out.println("It is 20");
                    break;
            default: System.out.println("Not present");

        }
    }
}

```

## ?: Operator

We have **covered conditional** operator `?:` in the previous chapter which can be used to replace `if...else` statements. It has the following general form –

**Exp1 ? Exp2 : Exp3;**

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.

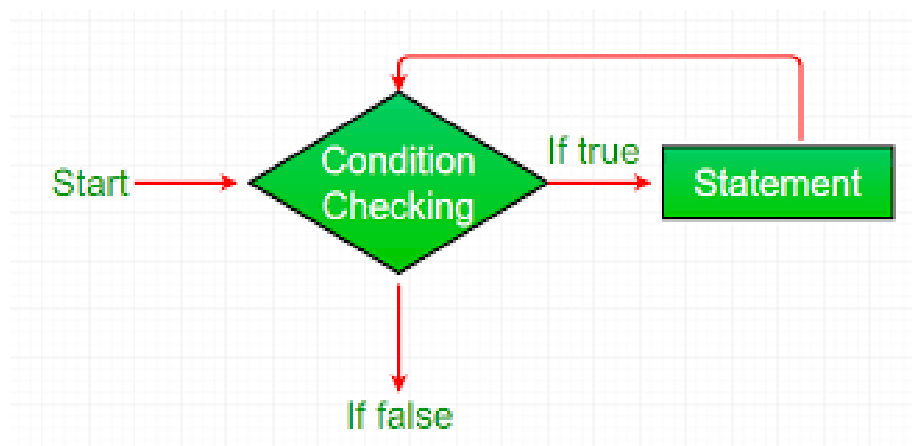
To determine the value of the whole expression, initially exp1 is evaluated.

- If the value of exp1 is true, then the value of Exp2 will be the value of the whole expression.
- If the value of exp1 is false, then Exp3 is evaluated and its value becomes the value of the entire expression.

## Loops

Programming languages provide various control structures that allow for more complicated execution paths.

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. Click the following links to check their detail.

## 1 while loop

Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

// Java program to illustrate while loop.

```
class whileLoopDemo {  
  
    public static void main(String args[])  
  
    {  
  
        // initialization expression  
  
        int i = 1;  
  
  
        // test expression  
  
        while (i < 6) {  
  
            System.out.println("Hello World");  
  
  
            // update expression  
  
            i++;  
  
        }  
  
    }  
  
}
```



## **2) for loop**

```
/*package whatever //do not write package name here */

// Java program to write a code in for loop from 1 to 10

class GFG {

    public static void main(String[] args)

    {

        for (int i = 1; i <= 10; i++) {

            System.out.println(i);

        }

    }

}
```

## **3) do...while loop**

```
// Java Program to Illustrate One Time Iteration

// Inside do-while Loop

// When Condition IS Not Satisfied


// Class

class GFG {
```

```
// Main driver method

public static void main(String[] args)

{

    // initial counter variable

    int i = 0;


    do {

        // Body of loop that will execute minimum

        // 1 time for sure no matter what

        System.out.println("Print statement");

        i++;

    }


    // Checking condition

    // Note: It is being checked after

    // minimum 1 iteration

    while (i < 0);

}

}
```

## Loop Control Statements-

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Java supports the following control statements. Click the following links to check their detail.

Sr.No. Control Statement & Description

### 1 break statement

- Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.

```
import java.io.*;
```

```
// Use of break statement in switch
```

```
class GFG {
```

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

```
//assigning n as integer value
```

```
int n = 1;
```

```
//passing n to switch
```

```
// it will check n and display output accordingly
```

```
switch(n){
```

case 1:

```
System.out.println("GFG");
```

```
break;
```

case 2:

```
System.out.println("Second Case");
```

```
break;
```

default:

```
System.out.println("default case");
```

```
}
```

```
}
```

```
}
```

## **2)continue statement**

// Java Program to illustrate the use of continue statement

```
// Importing Classes/Files
```

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

```
public class GFG {
```

```
// Main driver method

public static void main(String args[])

{

    // For loop for iteration

    for (int i = 0; i <= 15; i++) {

        // Check condition for continue

        if (i == 10 || i == 12) {

            // Using continue statement to skip the

            // execution of loop when i==10 or i==12

            continue;

        }

        // Printing elements to show continue statement

        System.out.print(i + " ");

    }

}

}
```

# Introducing Classes

## Object and methods:

### Defining a class

Java is an Object-Oriented Language. As a language that has the Object-Oriented feature, Java supports the following fundamental concepts –

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects
- Instance
- Method
- Message Parsing

**Class** – A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

### Adding variables and methods

A Java method is a collection of statements that are grouped together to perform an operation. When you call the `System.out.println()` method, for example, the system actually executes several statements in order to display a message on the console.

#### **Creating Method**

Considering the following example to explain the syntax of a method – Syntax

```
public static int methodName(int a, int b) {
```

```
// body
}
```

Here,

- `public static` – modifier
- `int` – return type
- `methodName` – name of the method

- a, b – formal parameters
- int a, int b – list of parameters

Method definition consists of a method header and a method body. The same is shown in the following syntax –

Syntax

```
modifier returnType nameOfMethod (Parameter List) {
// method body
}
```

**The syntax shown above includes –**

- modifier – It defines the access type of the method and it is optional to use.
- returnType – Method may return a value.
- nameOfMethod – This is the method name. The method signature consists of the method name and the parameter list.
- Parameter List – The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
- method body – The method body defines what the method does with the statements.

## Creating objects

In this example, we have created a Student class which has two data members id and name. We are creating the object of the Student class by new keyword and printing the object's value.

Here, we are creating a main() method inside the class.

File: Student.java

// Java program to Illustrate Creation of Object

// Using new keyword

// Main class

class GFG {

// Declaring and initializing string

// Custom input string

String name = "GeeksForGeeks";

// Main driver method

public static void main(String[] args)

{

    // As usual and most generic used we will

```

        // be creating object of class inside main()
        // using new keyword
        GFG obj = new GFG();

        // Print and display the object
        System.out.println(obj.name);
    }
}

```

## **Constructors-**

Constructor is a block of code that initializes the newly created object. A constructor resembles an instance method in java but it's not a method as it doesn't have a return type. In short constructor and method are different (More on this at the end of this guide). People often refer constructor as special type of method in Java.

// Java Program to demonstrate

// Constructor

```
import java.io.*;
```

```
class Geeks {
```

```
// Constructor
```

```
Geeks()
```

```
{
```

```
    super();
```

```
    System.out.println("Constructor Called");
```



```
}

// main function

public static void main(String[] args)

{

    Geeks geek = new Geeks();

}

}
```

## **Class Inheritance-**

A class that is derived from another class is called subclass and inherits all fields and methods of its superclass. In Java, only single inheritance is allowed and thus, every class can have at most one direct superclass. A class can be derived from another class that is derived from another class and so on.

## Unit – 2

### Arrays and Strings

- Java provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- Strings, which are widely used in Java programming, are a sequence of characters. In Java programming language, strings are treated as objects. The Java platform provides the String class to create and manipulate strings.

### Creating an array-

You can create an array by using the new operator with the following

**syntax – Syntax**

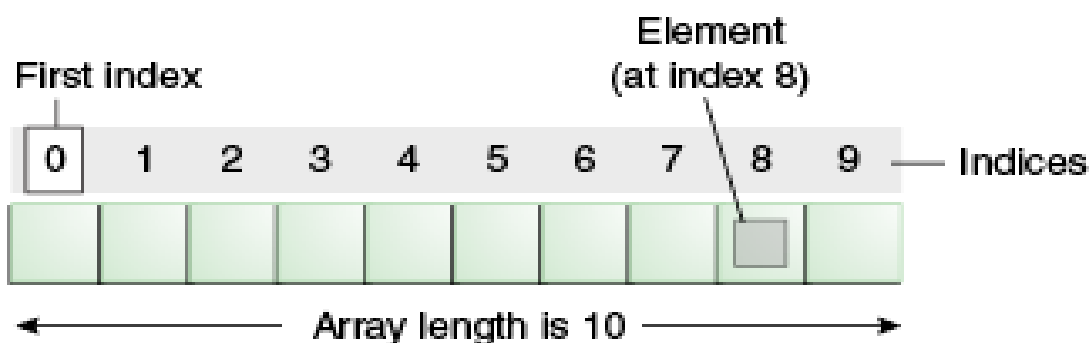
```
arrayRefVar = new dataType[arraySize];
```

**The above statement does two things –**

- It creates an array using new dataType[arraySize].
- It assigns the reference of the newly created array to the variable arrayRefVar.

### One and two dimensional array

A list of items group in a single variable name with only one index is called **1-D array**.



### **2-D Array-**

- The Two Dimensional Array in Java programming language is nothing but an Array of Arrays. If the data is linear we can use the One Dimensional Array but to work with multi-level data we have to use Multi-Dimensional Array.

- Two Dimensional Array in Java is the simplest form of Multi-Dimensional Array. In Two Dimensional Array, data is stored in row and columns and we can access the record using both the row index and column index (like an Excel File).

|       | Column 0             | Column 1             | Column 2             |
|-------|----------------------|----------------------|----------------------|
| Row 0 | <code>x[0][0]</code> | <code>x[0][1]</code> | <code>x[0][2]</code> |
| Row 1 | <code>x[1][0]</code> | <code>x[1][1]</code> | <code>x[1][2]</code> |
| Row 2 | <code>x[2][0]</code> | <code>x[2][1]</code> | <code>x[2][2]</code> |

## String array and methods-

- A Java String Array is an object that holds a fixed number of String values. Arrays in general is a very useful and important data structure that can help solve many types of problems.

### String Array Declaration

Square brackets is used to declare a String array. There are two ways of using it. The first one is to put square brackets after the String reserved word. For example:

```
String[] thisIsAStringArray;
```

Another way of declaring a String Array is to put the square brackets after the name of the variable. For example:

```
String thisIsAStringArray[];
```

Both statements will declare the variable "thisIsAStringArray" to be a String Array. Note that this is just a declaration, the variable "thisIsAStringArray" will have the value null. And since there is only one square brackets, this means that the variable is only a one-dimensional String Array. Examples will be shown later on how to declare multi-dimensional String Arrays.

## String and string buffer classes-

- String and StringBuffer both are the classes which operate on strings.
- StringBuffer class is the peer class of the class String.
- The object of String class is of fixed length.
- The object of the StringBuffer class is growable.
- The basic difference between String and StringBuffer is that the object of the “String” class is immutable. The object of the class “StringBuffer” mutable.

| <b>String</b>   | <b>StringBuffer</b>  |
|---|--|
| String is immutable.  | StringBuffer is mutable.   |
| When we concatenate too many strings, it takes longer and uses more memory because it creates a new instance each time.   | When concatenating two strings, StringBuffer is faster and uses less memory.           |
| When executing a concatenation operation, the String class is slower.   | When performing concatenation operations, the StringBuffer class is faster.            |
| The equals() method of the Object class is overridden by the String class. As a result, the equals() method can be used to compare the contents of two strings. | The equals() function of the Object class is not overridden by the StringBuffer class. |
| It overrides the object class's equal() and hashCode() methods.   | It cannot overrides the object class's equal() and hashCode() methods.                 |
| String constant pool is used by the String class.   | Heap memory is used by StringBuffer  |

## Inheritance-

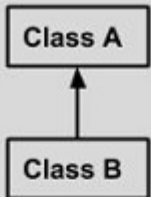
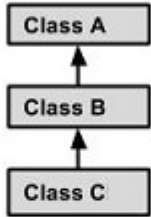
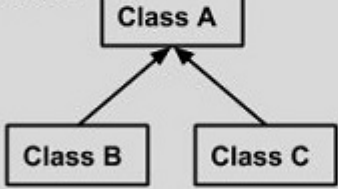
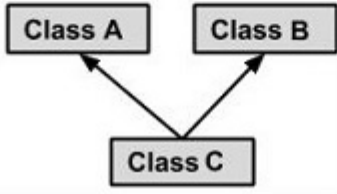
- Inheritance can be defined as the process where one class acquires the Inheritance can be defined as the process where one class acquires the

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

- The class which inherits the properties of other is known as subclass

## Basic types-

There are various types of inheritance as demonstrated below. There are various types of inheritance as demonstrated below. There are various types of inheritance as demonstrated below.

|  |  |
|--|--|
| <b>Single Inheritance</b>  <pre> graph BT     B[Class B] --&gt; A[Class A] </pre>                                 | <pre> public class A {     ..... } public class B extends A {     ..... } </pre>   |
| <b>Multi Level Inheritance</b>  <pre> graph BT     C[Class C] --&gt; B[Class B]     B --&gt; A[Class A] </pre>   | <pre> public class A { .....} public class B extends A {.....} public class C extends B {.....} </pre>   |
| <b>Hierarchical Inheritance</b>  <pre> graph BT     B[Class B] --&gt; A[Class A]     C[Class C] --&gt; A </pre> | <pre> public class A { .....} public class B extends A {.....} public class C extends A {.....} </pre>   |
| <b>Multiple Inheritance</b>  <pre> graph BT     C[Class C] --&gt; A[Class A]     C --&gt; B[Class B] </pre>     | <pre> public class A { .....} public class B {.....} public class C extends A,B {     ..... } // Java does not support mutple Inheritance </pre> |

## Using super-

### The super keyword

The super keyword is similar to this keyword. Following are the scenarios where the super keyword is used.

- It is used to differentiate the members of superclass from the members of subclass, if they have same names.
- It is used to invoke the superclass constructor from subclass. Differentiating the Members

If a class is inheriting the properties of another class. And if the members of the superclass have the names same as the sub class, to differentiate

these variables we use super keyword as shown below.

```
super.variable  
super.method();
```

## **Multilevel hierarchy abstract and final classes-**

- The classes in the lower hierarchy inherit all the variables (static attributes) and methods (dynamic behaviors) from the higher hierarchies. A class in the lower hierarchy is called a subclass (or derived, child, extended class). A class in the upper hierarchy is called a superclass (or base, parent class).
- In Multilevel hierarchy type of Inheritance, each subclass inherits all of the traits found in all of its super classes. It is perfectly acceptable to use a subclass as superclass of another.

## **Object class-**

- Every class in Java is directly or indirectly derived from the Object class. If a Class does not extend any other class then it is direct child class of Object and if extends other class then it is an indirectly derived. Therefore the Object class methods are available to all Java classes.

## **Packages and interfaces-**

- The content in packages and interfaces can be used by the classes by importing and implementing it correspondingly. The basic difference between packages and interfaces is that a package contains a group of classes and interfaces whereas; an interface contains methods and fields.

## **Extending interfaces-**

An interface can extend another interface in the same way that a class can extend another class. The extends keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

The following Sports interface is extended by Hockey and Football interfaces. Example

// Filename: Sports.java

```
public interface Sports {
    public void setHomeTeam(String name);
    public void setVisitingTeam(String name);
}
```

// Filename: Football.java

```
public interface Football extends Sports {
    public void homeTeamScored(int points);
    public void visitingTeamScored(int points);
    public void endOfQuarter(int quarter);
}
```

// Filename: Hockey.java

```
public interface Hockey extends Sports {
```

For More Information go to [www.dpmishra.com](http://www.dpmishra.com)

41

Keep Learning with us VBSPU BCA 3<sup>rd</sup> sem. JAVA Programming

```
    public void homeGoalScored();
    public void visitingGoalScored();
    public void endOfPeriod(int period);
    public void overtimePeriod(int ot);
}
```

The Hockey interface has four methods, but it inherits two from Sports; thus, a class that implements Hockey needs to implement all six methods. Similarly, a class that implements Football needs to define the three methods from Football and the two methods from Sports.

## Packages

A package as the name suggests is a pack(group) of classes, interfaces and other packages. In java we use packages to organize our classes and interfaces. We have two types of packages in Java: built-in packages and the packages we can create (also known as user defined package). In this guide we will learn what are packages, what are user-defined packages in java and how to use them.

In java we have several built-in packages, for example when we need user input, we import a package like this:

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

Here:

- java is a top level package
- util is a sub package
- and Scanner is a class which is present in the sub package util.

## Types of packages in Java

As mentioned in the beginning of this guide that we have two types of packages in java.

**1) User defined package:** The package we create is called user-defined package.

**2) Built-in package:** The already defined package like java.io.\*, java.lang.\* etc are known as built-in packages.



## Unit – 3

### Exception Handling

- The **Exception Handling in Java** is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

### Fundamentals exception types-

we have three categories of Exceptions. You need to understand them to know how exception handling works in Java.

- **Checked exceptions** – A checked exception is an exception that is checked (notified) by the compiler at compilation-time, these are also called as compile time exceptions. These exceptions cannot simply be ignored, the programmer should take care of (handle) these exceptions.

For example, if you use `FileReader` class in your program to read data from a file, if the file specified in its constructor doesn't exist, then a `FileNotFoundException` occurs, and the compiler prompts the programmer to handle the exception.

Example

```
import java.io.File;

import java.io.FileReader;

public class FileNotFound_Demo {

    public static void main(String args[]) {

        File file = new File("E://file.txt");

        FileReader fr = new FileReader(file);

    }

}
```

- **Unchecked exceptions** – An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.

For example, if you have declared an array of size 5 in your program and trying to call the 6<sup>th</sup> element of the array then an `ArrayIndexOutOfBoundsException` occurs.

Example

```
public class Unchecked_Demo {

    public static void main(String args[]) {
        int num[] = {1, 2, 3, 4};
        System.out.println(num[5]);
    }
}
```

**Errors** – These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

## Uncaught exceptions-

If you do not handle a runtime exception in your program, the default behavior will be to terminate the program and dump the call stack trace on the console. Java provides an elegant mechanism of handling Runtime Exceptions that you might have not handled in your program. `UncaughtExceptionHandler` can be defined at three levels. From highest to lowest they are:

1. `Thread.setDefaultUncaughtExceptionHandler`
2. `ThreadGroup.uncaughtException`
3. `Thread.setUncaughtExceptionHandler`

Uncaught exception handling is controlled first by the thread, then by the thread's Thread Group object and finally by the default uncaught exception handler.

## Throw-

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

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

Example

```
import java.io.*;

public class className {

    public void deposit(double amount) throws
RemoteException { // Method implementation

    throw new RemoteException();

}

// Remainder of class definition

}
```

## Final-

The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception.

Using a finally block allows you to 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 the catch blocks and has the following syntax –

Syntax

```
try {
    // Protected code
} catch (ExceptionType1 e1) {
    // Catch block
} catch (ExceptionType2 e2) {
    // Catch block
```

```

} catch (ExceptionType3 e3) {
// Catch block
}finally {
// The finally block always executes.
}

```

## Built in exception-

Java defines several exception classes inside the standard package `java.lang`.

Java defines several other types of exceptions that relate to its various class libraries. Following is the list of Java Unchecked `RuntimeException`.

| Exception                                    | Meaning   |
|--|---|
| <code>ArithmeticException</code>             | Arithmetic error, such as divide-by-zero.                         |
| <code>ArrayIndexOutOfBoundsException</code>  | Array index is out-of-bounds.                                     |
| <code>ArrayStoreException</code>             | Assignment to an array element of an incompatible type.           |
| <code>ClassCastException</code>              | Invalid cast.   |
| <code>EnumConstantNotPresentException</code> | An attempt is made to use an undefined enumeration value.         |
| <code>IllegalArgumentException</code>        | Illegal argument used to invoke a method.                         |
| <code>IllegalMonitorStateException</code>    | Illegal monitor operation, such as waiting on an unlocked thread. |
| <code>IllegalStateException</code>           | Environment or application is in incorrect state.                 |
| <code>IllegalThreadStateException</code>     | Requested operation not compatible with current thread state.     |
| <code>IndexOutOfBoundsException</code>       | Some type of index is out-of-bounds.                              |
| <code>NegativeArraySizeException</code>      | Array created with a negative size.                               |
| <code>NullPointerException</code>            | Invalid use of a null reference.                                  |
| <code>NumberFormatException</code>           | Invalid conversion of a string to a numeric format.               |
| <code>SecurityException</code>               | Attempt to violate security.                                      |
| <code>StringIndexOutOfBoundsException</code> | Attempt to index outside the bounds of a string.                  |
| <code>TypeNotPresentException</code>         | Type not found.   |
| <code>UnsupportedOperationException</code>   | An unsupported operation was encountered.                         |

**Table 10-1** Java's Unchecked `RuntimeException` Subclasses Defined in `java.lang`

## Creating your own exceptions-

You can create your own exceptions in Java. Keep the following points in mind when writing your own exception classes –

- All exceptions must be a child of `Throwable`.
  - If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the

Exception class.

- If you want to write a runtime exception, you need to extend the RuntimeException class.

We can define our own Exception class as below –

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

You just need to extend the predefined Exception class to create your own Exception. These are considered to be checked exceptions. The following InsufficientFundsException class is a user-defined exception that extends the Exception class, making it a checked exception. An exception class is like any other class, containing useful fields and methods.

Example

// File Name InsufficientFundsException.java

```
import java.io.*;
```

```
public class InsufficientFundsException extends Exception {
```

```
    private double amount;
```

```
    public InsufficientFundsException(double amount) {
```

```
        this.amount = amount;
```

```
    }
```

```
    public double getAmount() {
```

```
        return amount;
```

```
    }
```

```
}
```

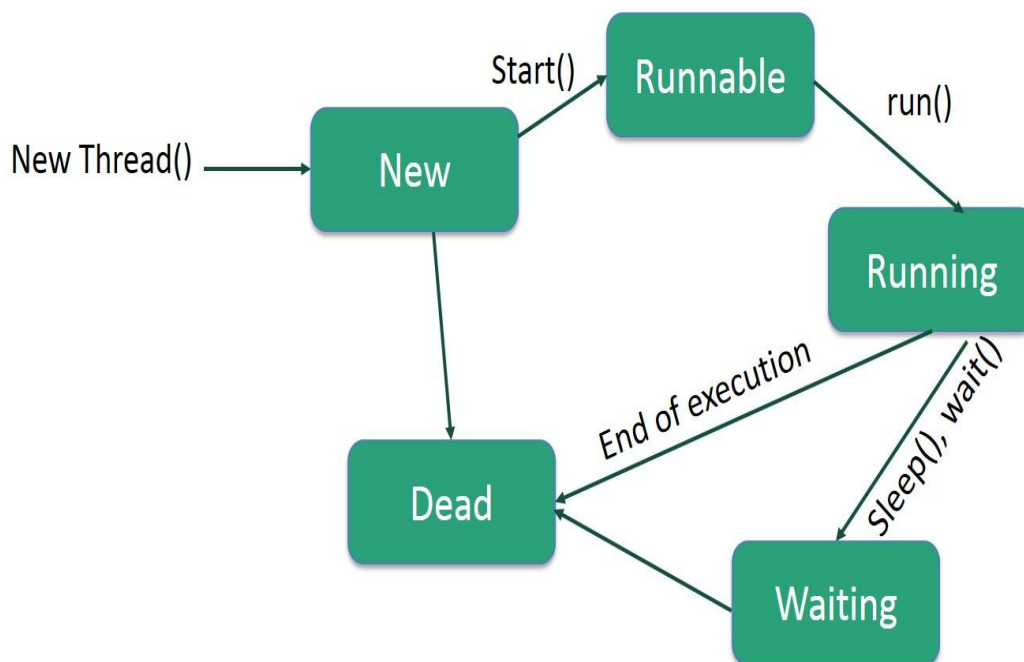
## Multithreaded programming Fundamentals-

- Java is a multi-threaded programming language which means we can develop multi-threaded program using Java. A multi-threaded program

contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

## Life Cycle of a Thread

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread.



Following are the stages of the life cycle –

- **New** – A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
- **Runnable** – After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting** – Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed Waiting** – A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated (Dead)** – A runnable thread enters the terminated state when it completes its task or otherwise terminates.

## **Java thread model:**

Java thread model can be defined in the following three sections:

### **Thread Priorities**

Each thread has its own priority in Java. Thread priority is an absolute integer value. Thread priority decides only when a thread switches from one running thread to next, called context switching. Priority does increase the running time of the thread or gives faster execution.

**Synchronization** Java supports an asynchronous multithreading, any number of thread can run simultaneously without disturbing other to access individual resources at different instant of time or shareable resources. But some time it may be possible that shareable resources are used by at least two threads or more than two threads, one has to write at the same time, or one has to write and other thread is in the middle of reading it. For such type of situations and circumstances Java implements synchronization model called monitor. The monitor was first defined by C.A.R. Hoare. You can consider the monitor as a box, in which only one thread can reside. As a thread enter in monitor, all other threads have to wait until that thread exits from the monitor. In such a way, a monitor protects the shareable resources used by it being manipulated by other waiting threads at the same instant of time. Java provides a simple methodology to implement synchronization.

### **Messaging**

A program is a collection of more than one thread. Threads can communicate with each other. Java supports messaging between the threads with lost-cost. It

provides methods to all objects for inter-thread communication. As a thread exits from synchronization state, it notifies all the waiting threads.

## **Thread class-**

Thread class is the main class on which Java's Multithreading system is based. Thread class, along with its companion interface Runnable will be used to create and run threads for utilizing Multithreading feature of Java.

## Runnable interface-

If your class is intended to be executed as a thread then you can achieve this by implementing a Runnable interface. You will need to follow three basic steps –

### Step 1

As a first step, you need to implement a run() method provided by a Runnable interface. This method provides an entry point for the thread and you will put your complete business logic inside this method. Following is a simple syntax of the run() method –

```
public void run( )
```

### Step 2

As a second step, you will instantiate a Thread object using the following constructor –

```
Thread(Runnable threadObj, String threadName);
```

Where, threadObj is an instance of a class that implements the Runnable interface and threadName is the name given to the new thread.

### Step 3

Once a Thread object is created, you can start it by calling start() method, which executes a call to run( ) method. Following is a simple syntax of start() method –

```
void start();
```

### Example

Here is an example that creates a new thread and starts running it –

```
class RunnableDemo implements Runnable {
    private Thread t;
    private String threadName;

    RunnableDemo( String name) {
        threadName = name;
        System.out.println("Creating " + threadName );
    }
```



```

public void run() {
    System.out.println("Running " + threadName );
    try {
        for(int i = 4; i > 0; i--) {
            System.out.println("Thread: " + threadName + ", " + i);
            // Let the thread sleep for a while.
            Thread.sleep(50);
        }
    } catch (InterruptedException e) {
        System.out.println("Thread " + threadName + " interrupted.");
    }

    System.out.println("Thread " + threadName + "
    exiting."); }

public void start () {

    System.out.println("Starting " + threadName );
    if (t == null) {
        t = new Thread (this, threadName);
        t.start ();
    }
}
}

```

```

public class TestThread {

```

```

    public static void main(String args[]) {

        RunnableDemo R1 = new RunnableDemo( "Thread-1");

```

```
R1.start();
```

```
RunnableDemo R2 = new RunnableDemo( "Thread-2");
```

```
R2.start();
```

```
}
```

```
}
```

## Interthread communication-

Interthread communication is important when you develop an application where two or more threads exchange some information.

There are three simple methods and a little trick which makes thread communication possible. All the three methods are listed below –

These methods have been implemented as final methods in Object, so they are available in all the classes. All three methods can be called only from within a synchronized context.

## Suspended Resuming and stopping thread

Core Java provides complete control over multithreaded program. You can develop a multithreaded program which can be suspended, resumed, or stopped completely based on your requirements. There are various static methods which you can use on thread objects to control their behavior. Following table lists down those methods –

. Method & Description

### **1 public void suspend()**

This method puts a thread in the suspended state and can be resumed using resume() method.

### **2 public void stop()**

This method stops a thread completely.

### **3 public void resume()**

This method resumes a thread, which was suspended using suspend() method.

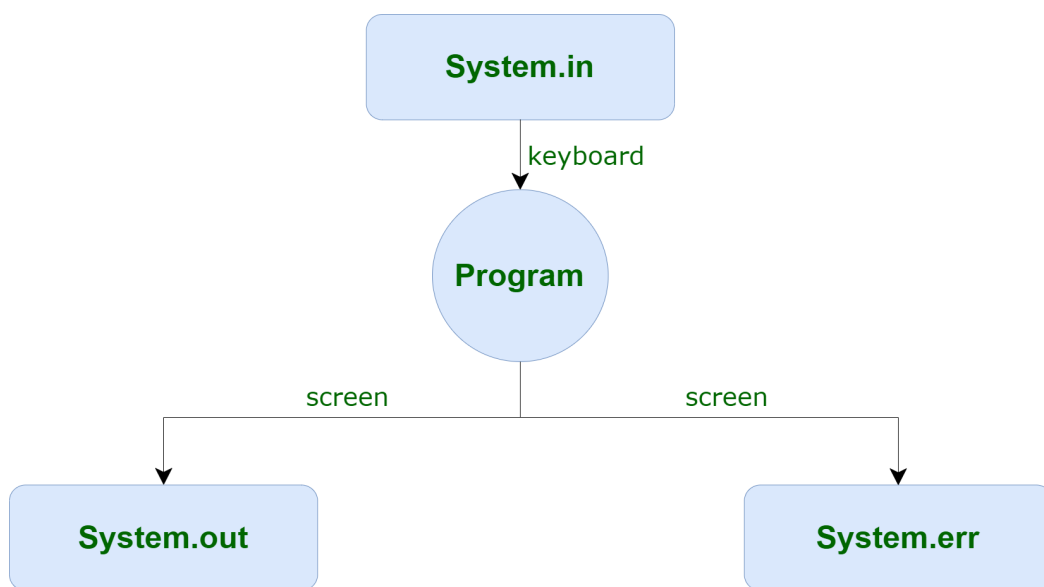
### **4 public void wait()**

Causes the current thread to wait until another thread invokes the notify().

Be aware that the latest versions of Java has deprecated the usage of suspend( ), resume( ), and stop( ) methods and so you need to use available alternatives.

## Unit – 4

### Input/output Basics



**Standard I/O Streams in Java**



# Stream

## Byte and character stream

Java byte streams are used to perform input and output of 8  
 there are many classes related to byte streams but the most frequently  
 used there are many classes related to byte streams but the most  
 frequently used there are many classes related to byte streams but the  
 most frequently used

```
import java.io.*;
```

```
public class CopyFile {
```

```
    public static void main(String args[]) throws IOException {
```

```
        FileInputStream in = null;
```

```
        FileOutputStream out = null;
```

```
        try {
```

```
            in = new FileInputStream("input.txt");
```

```
            out = new FileOutputStream("output.txt");
```

```
            int c;
```

```
            while ((c = in.read()) != -1) {
```

```
                out.write(c);
```

```
            }
```

```
        }finally {
```

```
            if (in != null) {
```

```
                in.close();
```

```
            }
```

```
            if (out != null) {
```

```

out.close();

}

}

}

}

```

## Character Streams

Java Byte streams are used to perform input and output of 8-bit bytes, whereas Java Character streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are, `FileReader` and `FileWriter`. Though internally `FileReader` uses `FileInputStream` and `FileWriter` uses `FileOutputStream` but here the major difference is that `FileReader` reads two bytes at a time and `FileWriter` writes two bytes at a time.

We can re-write the above example, which makes the use of these two classes to copy an input file (having unicode characters) into an output file –

### Example

```

import java.io.*;

public class CopyFile {

    public static void main(String args[]) throws IOException {

        FileReader in = null;

        FileWriter out = null;

        try {

            in = new FileReader("input.txt");

            out = new FileWriter("output.txt");

            int c;

```

```

while ((c = in.read()) != -1) {
    out.write(c);
}
}finally {
    if (in != null) {
        in.close();
    }
    if (out != null) {
        out.close();
    }
}
}
}
}
}

```

## **Predefined streams-**

- As you know, all Java programs automatically import the java.lang package. This package defines a class called System, which encapsulates several aspects of the run-time environment. Among other things, it contains three predefined stream variables, called in, out, and err. These fields are declared as public, final, and static within System. This means that they can be used by any other part of your program and without reference to a specific System object.

## **Reading and writing from console and file-**

As described earlier, a stream can be defined as a sequence of data. As described earlier, a stream can be defined as a sequence of data.

As described earlier, a stream can be defined as a sequence of data. The InputStream is used to read data from a source and the

is used to read data from a source and the OutputStream is used for writing data to a destination.

used for writing data to a destination.

Here is a hierarchy of classes to deal with Input and Output streams.

```
// Java program to demonstrate BufferedReader
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Test {
    public static void main(String[] args)
        throws IOException
    {
        // Enter data using BufferReader
        BufferedReader reader = new BufferedReader(
            new InputStreamReader(System.in));

        // Reading data using readLine
        String name = reader.readLine();

        // Printing the read line
        System.out.println(name);
    }
}
```

Once you have InputStream object in hand, then there is a list of helper methods which can be used to read to stream or to do other operations on the stream.

### **Sr.No. Method & Description**

#### **1 public void close() throws IOException{}**

This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException.

#### **2 protected void finalize()throws IOException {}**

This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException.

#### **3 public int read(int r)throws IOException{}**



This method reads the specified byte of data from the InputStream. Returns an int. Returns the next byte of data and -1 will be returned if it's the end of the file.

#### **4 public int read(byte[] r) throws IOException{}**

This method reads r.length bytes from the input stream into an array. Returns the total number of bytes read. If it is the end of the file, -1 will be returned.

## **Networking**

### **Basics-**

- The term network programming refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.
- Networking is the concept of connecting multiple remote or local devices together. Java program communicates over the network at application layer.

### **Networking classes and interfaces-**

- Java supports TCP/IP both by extending the already established stream I/O interface introduced in Chapter 19 and by adding the features required to build I/O objects across the network. Java supports both the TCP and UDP protocol families. TCP is used for reliable stream based I/O across the network. UDP supports a simpler, hence faster, point-to point datagram-oriented model. The classes contained in the java.net package are shown here:
- Authenticator
- Inet6Address
- ServerSocket
- CacheRequest
- InetAddress
- Socket
- CacheResponse
- InetSocketAddress
- SocketAddress
- ContentHandler

- Interface Address (Added by SocketImpl)
- Java SE 6.)
- CookieHandler
- JarURLConnection
- SocketPermission
- CookieManager (Added by MulticastSocket)
- URI
- Java SE 6.)
- DatagramPacket
- NetPermission
- URL

## Using java.net package

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.

The java.net package provides support for the two common network protocols –

- **TCP** – TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
- **UDP** – UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

- 
- 
- 
- 
- 
-

## **Unit – 5**

### **Event Handling**

- Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events. Let's have a brief introduction to this model.

### **Different mechanism**

#### **Steps involved in event handling**

- The User clicks the button and the event is generated.
- Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.
- Event object is forwarded to the method of registered listener class.
- the method is now get executed and returns.

#### **Points to remember about listener**

- In order to design a listener class we have to develop some listener interfaces. These Listener interfaces forecast some public abstract callback methods which must be implemented by the listener class.
- If you do not implement the any if the predefined interfaces then your class can not act as a listener class for a source object.

### **The delegation event model-**

- The Delegation Event model is one of the many techniques used to handle events in GUI (Graphical User Interface) programming languages. GUI represents a system where an user visually/graphically interacts with the system. Other interactive systems are text based or called CUI (Character User Interface). CUI interacts with the system by typing out commands in the

console.

- Back in the old days, Java used a Chain of Responsibility pattern to process events. For example, when a button is clicked, an event is generated, which

then is passed through a chain of components. The chain of components is defined by the hierarchy of classes and interfaces. An event is caught and handled by the handler class. This mechanism was used by Java version 1.0, which is very different from the event handling scheme of Java version 1.1 onwards. Old methods are still supported, but deprecated and hence not recommended for new programs. A modern approach is based on the delegation event model.

## **Event classes-**

- The Event classes represent the event. Java provides us various Event classes but we will discuss those which are more frequently used.

Following is the list of commonly used event classes.

### **1 AWTEvent**

It is the root event class for all AWT events. This class and its subclasses supercede the original `java.awt.Event` class.

### **2 ActionEvent**

The `ActionEvent` is generated when a button is clicked or the item of a list is double clicked.

### **3 InputEvent**

The `InputEvent` class is the root event class for all component-level input events.

### **4 MouseEvent**

This event indicates a mouse action occurred in a component.

### **5 TextEvent**

The object of this class represents the text events.

## **6 WindowEvent**

The object of this class represents the change in state of a window.

## **7 AdjustmentEvent**

The object of this class represents the adjustment event emitted by Adjustable objects.

## **8 ComponentEvent**

The object of this class represents the change in state of a window.

## **9 ContainerEvent**

The object of this class represents the change in state of a window.

## **10 MouseMotionEvent**

The object of this class represents the change in state of a window.

# **Event Listener Interface-**

- It is a marker interface which every listener interface has to extend. This class is defined in java.util package.
- Following is the list of commonly used event listeners.

## **Sr. No. Control & Description**

### **1 ActionListener**

This interface is used for receiving the action events.

### **2 ComponentListener**

This interface is used for receiving the component events.

### **3 ItemListener**

This interface is used for receiving the item events.

### **4 KeyListener**

This interface is used for receiving the key events.

**5 MouseListener**

This interface is used for receiving the mouse events.

**6 TextListener**

This interface is used for receiving the text events.

**7 WindowListener**

This interface is used for receiving the window events.

**8 ContainerListener**

This interface is used for receiving the container events.

**9 MouseMotionListener**

This interface is used for receiving the mouse motion events.

**Adapter and inner classes****Adapters:**

Following is the list of commonly used adapters while listening GUI events in AWT.

**Sr. No. Adapter & Description****1 FocusAdapter**

An abstract adapter class for receiving focus events.

**2 KeyAdapter**

An abstract adapter class for receiving key events.

**3 MouseAdapter**

An abstract adapter class for receiving mouse events.

**4 MouseMotionAdapter**

An abstract adapter class for receiving mouse motion events.

## Inner Classes-

Java inner class or nested class is a class which is declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Additionally, it can access all the members of outer class including private data members and methods.

### Syntax of Inner class

```
1. class Java_Outer_class{
2. //code
3. class Java_Inner_class{
4. //code
5. }
6. }
```

## Working with windows-

- The Java WindowListener is notified whenever you change the state of window. It is notified against WindowEvent. The WindowListener interface is found in java.awt.event package. It has three methods.

### Methods of WindowListener interface

The signature of 7 methods found in WindowListener interface are given below:

```
1. public abstract void windowActivated(WindowEvent e);
2. public abstract void windowClosed(WindowEvent e);
3. public abstract void windowClosing(WindowEvent e);
4. public abstract void windowDeactivated(WindowEvent e);
5. public abstract void windowDeiconified(WindowEvent e);
6. public abstract void windowIconified(WindowEvent e);
7. public abstract void windowOpened(WindowEvent e);
```