Java Copy Notes

ICT 5th Semester

by

Mukesh Singh
ICT 5th batch 2073

Sukuna Multiple Campus
Sundarharaincha, Morang

Mukesh Singh SMC

Unit-1

Introduction to Java

1.1. Java as a Programming Platform

Java is a general-purpose programming language that is class-based, object oriented and designed to have as few implementation dependencies as possible. ~~Java is~~ a

Java is a programming language and a computing platform for application development. It was first released by Sun Microsystem in 1995 and later acquired by Oracle Corporation. It is one of the most used programming languages.

Java ~~as~~ Platform

Java platform is a collection of programs that help to develop and run programs written in the Java programming language. Java platform includes an execution engine, a compiler, and a set of libraries. Java is platform-independent language. It is not specific to any processor or operating system.

1.2. The Java "White Paper" Buzzwords

The authors of Java have written an influencial White Paper that explains their design goals and accomplishments. The features of Java are also known as Java buzzwords.

Following are the enlisted Java buzzwords / features:

1) Simple:

Java is easy to learn, and its syntax is simple, clean and easy to ~~est~~ understand.

Mukesh Singh

Java is simple programming language because:
→ Java syntax is based on C++
→ Java has removed many complicated and rarely-used features.
→ There is no need to remove unferenced objects because there is an Automatic Garbage Collection in Java.

2) Object Oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-Oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.

OOPs is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

a) Object
b) Class
c) Inheritance
d) Polymorphism
e) Abstraction
f) Encapsulation

3. Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:
- No explicit pointer

- Java ~~machine~~ programs run inside a virtual machine sandbox.
- Security manager

4. Platform Independent

Java is platform independent because it is different from other languages like C, C++, etc. which are compiled ~~to~~ into platform specific machines ~~with~~ while Java is write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms: software-based and hardware-based. Java provides a software-based platform

5. Robust

Robust simply means strong. Java is robust because:

- It uses strong memory management.
- There is lack of pointer that avoids security problems.
- There are exception handling and the type checking mechanism in Java.

6. Portable

Java is portable because it facilitates us to carry the Java bytecode to any platform. It doesn't require any implementation.

7. Architecture-neutral:

Java is architecture neutral because there are

Mukesh Singh SMC

no implementation dependent features, for example, the size of primitive type is fixed.

8. Dynamic

Java is dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e. C and C++.

Java supports dynamic compilation and automatic memory management.

9. Interpreted

Java is an interpreted language. The bytecode is generated by the Java compiler can certainly run on any machine has an interpreter. Early Java development tools were quite slower. Today, the Just-in-time (JIT) compiler helps to translate the bytecode into machine code.

10. High performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than compiled languages like C, C++. It is interpret language so it is slower than compiled languages.

11. Multithreaded

A thread is like a separe program, executing concurrently. ~~The~~ We can write Java programs ~~by~~ that deal with many tasks at once by defending multiple threads. The main advantage of multithreading is that it doesn't occupy memory for each thread. It shares a common memory area.

12. Distributed

Java is distributed because it ~~shares~~ facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

1.4. A Short History of Java

Java was created by three scientists James Gosling, Mike Sheridan and Patrick Naughton (members of green team).

Java was developed by Sun Micro-System later acquired by Oracle.

Java began from 1991 and the first publically arrival version of Java (Java 1.0) was released in 1995.

1.5 Writing Simple Java Programs

1) Hello World Program

```
Public class Hello
{
public static void main (String[] args)
{
system.out.printin ("Hello World!");
}
}
```

2) /* First Program */

```
class A
{
Public static void main (String args[])
{
system.out.println ("5th Sem");
}
}
```

Mukesh Singh SMC

Java Virtual Machine (JVM)

The Java Virtual Machine (JVM) is a Software implementation of a computer that ~~exceeds~~ executes programs like a real machines.

Mukesh Singh SMC

Unit-2

Fundamental Programming Structures

A simple Java Program

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

2.1. Writing Comments:

Java programming language supports three types of comments:

i) Single line (or end-of-line) Comment:

It starts with a double slash symbol (//) and terminates at the end of the current line. The compiler ignores everything from // to the end of the line.

Example

```
// Calculate sum of two numbers
```

ii) Multiline Comment

Java programmer can use C/C++ comment style that begins with delimiter /* and ends with */.

Mukesh Singh SMC

Example
/* Calculate sum of two
numbers
and it is a multiline Comment */

iii) Documentation Comments:
This comment style is new in Java. Such comments begin with delimiter /** and end with */. The compiler also ignores this type of comment just like it ignores comments that use /* and */.
Example:
/** The text enclosed here will be part of program documentation */

2.2 Basic Data Types

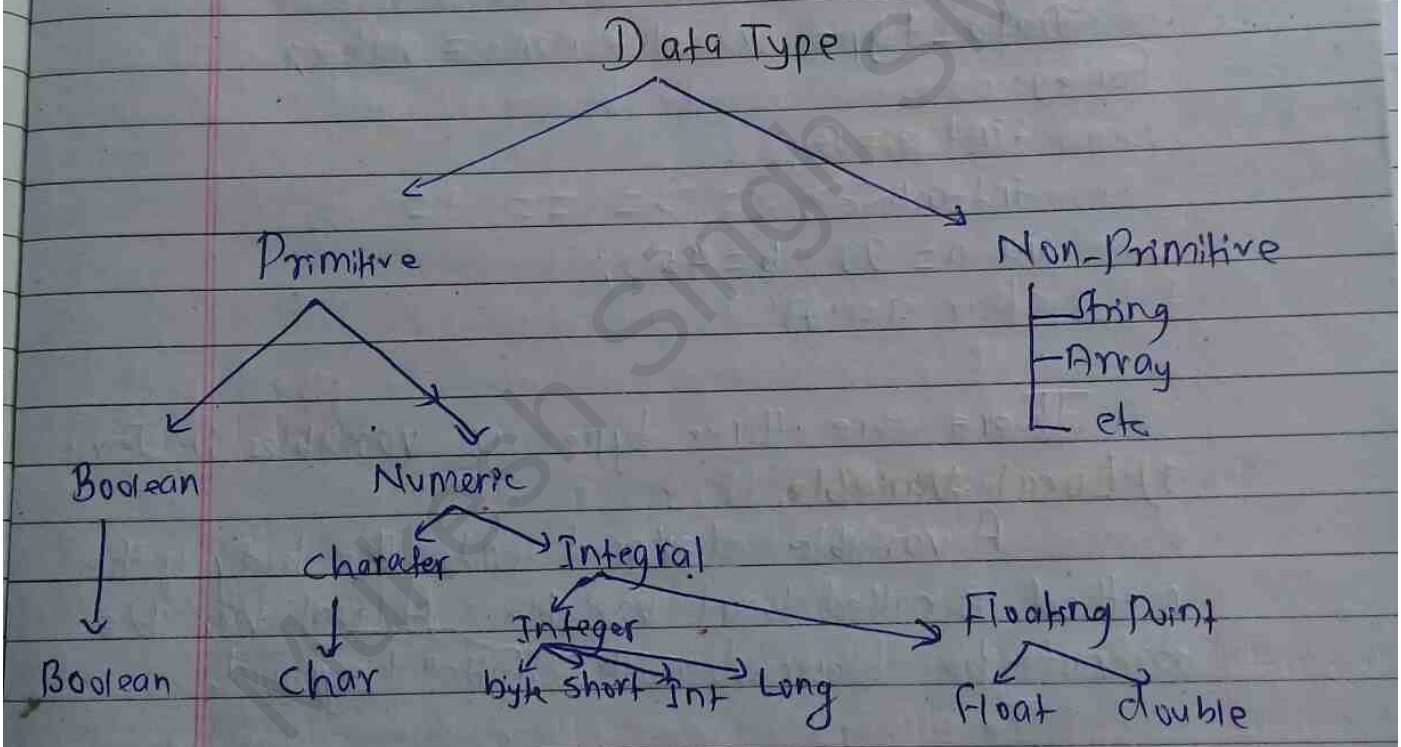
Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. Primitive (fundamental) data types:
In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.
The primitive data types are: boolean, char, byte, short, int, long, float and double

2. Non-primitive (derived) data types

The non-primitive data types are those whose ~~values~~ variables allow us to store multiple values of same type. But they never allows to store multiple values of different types.

The non-primitive data types includes classes, interfaces and arrays.



Mukesh Singh SMC

2.3. Variables and Constants

Variables

A variable is a container which holds the value while the java program is executed. A variable is assigned with a datatype. Variable is a name of memory location.

Syntax:

Data_type variable name = value;

For eg:

~~int a=B~~;

int a,b, c;

int a= 99, b=450;

char a='A'.

There are three types of variables in Java:

1) Local Variable

A variable declared inside the body of the method is called local variable. A local variable cannot be defined with "static" keyword.

2) Instance Variable (Global variables)

A variable declared inside the class but outside the body of the method, is called instance variable. It is declared as static. Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.

Mukesh Singh SMC

25 m

3) Class / Static Variables

The variables those are declared inside a class are called as class variables.

Class variables are known as static variables are declared with the 'static' keyword in a class, but outside the method, constructor or block.

Static variables are created when the program starts and destroyed when the program stops.

Constants

Constants in Java are fixed values those are not changed during the execution of program.

Java supports several types of constants.

Some constant are: Integer Constants, real Constants, String Constants etc.

Syntax

Static final data type constant name = value;

for Eg:

Static final float pi = 3.14;

2.4 Operators

Operators are symbols that can be used for mathematical and logical operations on variables and values.

There are various types of operators explained below:

Mukesh Singh SMC

1. Arithmetic Operators (Binary Operators)

It performs simple arithmetic calculations such as addition, subtraction, multiplication etc. It uses two operands so that it is called binary operator.

Examples: +, -, *, /, % etc.

2. Relational Operators (Comparision Operators)

It is used to compare the relationship between operands and bring out a decision accordingly. It is also called comparision operator because it is used to compare any two expressions.

Examples: <, >, <=, >=, ==, != etc.

3. Logical Operators (Boolean Operations)

Logical operators are used to give logical value either true or false.

Examples: && , || , !

AND OR NOT

4. Assignment Operators

The assignment operator is used to assign the value of expression into variable. It is written in the form:

variable = expression

i.e $x = a+b$.

Examples: +=, -=, *=, /=, %= etc.

5. Unary Operators (Increment / Decrement Operators)

The increment and decrement operators are very useful operators in C language. Two unusual and unique operators '++' and '--' are called increment and decrement operators respectively.

The increment operator increases the value of operand by 1; and the decrement operator decreases the value of operand by 1.

Syntax	Example
• ++ variable name (pre-increment)	++a = a = a+1;
• variable name ++ (post-increment)	a++ = a = a+1;
• -- variable name (pre-decrement)	--a = a = a-1;
• variable name -- (post decrement)	a-- = a = a-1;

6. Conditional Operators (Ternary Operators) or. (?: operator)

Ternary operator is also known as conditional operators as it checks condition to make decision. It uses two symbols : the question mark (?) and the colon (:) but not together.

Syntax

Condition ? Statement 1 : Statement 2

Example

a = 10; b = 15;

x = (a > b) ? a : b;

7. Bitwise Operators

These operators are used to perform manipulating individual bits of a number. They can be used with any of the integer types.

Examples: Bitwise AND (&), Bitwise OR (|), Bitwise XOR (^), ~~etc~~ Bitwise Complement (~).

8. Shift Operators

These operators are used to shift the bits of a number left or right thereby multiplying or dividing the number by two respectively. They can be used when we have to multiply or divide a number by two (2).

Examples/Types: → Left Shift (<<), Right Shift (>>)

9.

Mukesh Singh SMC

2.5. Type Casting

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

The process of converting one type of data into another type is called type conversion.

Example of Type Casting:

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

In Java, type casting is classified into two types:

1. Widening or Automatic or Implicit Casting

Widening casting is done automatically when passing a smaller size type to a larger size type:

byte → short → int → long → float → double

Widening

Example

```
public class ~~Type Casting~~ Low2High {
  public static void main (String [] args)
  {
  int i = 100;
  long l = i;
  float f = l;
  System.out.println ("Int Value" + i);
  System.out.println ("Long value" + l);
  System.out.println ("float Value" + f);
  }
}
```

2. Narrowing or Explicit Casting

Narrowing casting is done manually when passing a ~~smaller~~ larger type to a smaller size type.

When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.

double → float → long → Int → Short → Byte

Narrowing

Example

```
public class High2Low
{
public static void main (String[] args)
{
  double d = 100.05;
  Long l = (long)d;
  int i = (int)l;
System.out.println ("Double value" +d);
System.out.println ("Long value" +l);
System.out.println ("Int value" +i);
}
}
```

Mukesh Singh SMC

2.6. Control flow

Control structures/flow are those programming constructs which controls the flow of program statements execution in a program.

Control flow statements, change or break the flow of execution by implementing decision making, looping, and branching your program to execute particular blocks of code based on the conditions.

There are three types of control flow statements supported by the Java programming language.

1. Decision/Selection Statements
2. Looping/Iteration Statements
3. Branching/Jumping statements

1. Decision / Selection Statements

Selection statement is also called as decision making statements because it provides the decision making capabilities to the statements.

There are two types of selection statements:

i) If (Conditional Statement)

It is the most common decision making control structure which controls the flow of program statement execution based on the condition checked. It can be used in different forms as

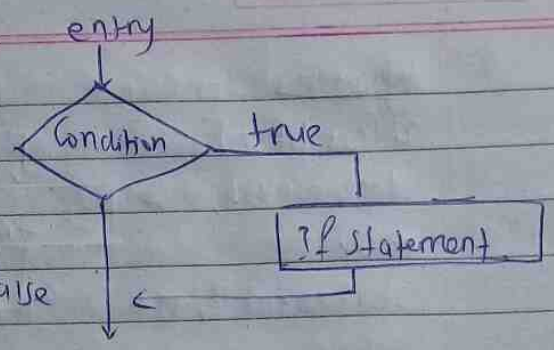
a) The If statement

It is the simplest form of conditional statement in which statements are executed if the test expression (condition) is true otherwise the control exit from the

condition.

Syntax

```
if (condition)
{
statements;
}
```



Example

```
public class IfStatements
{
public static void main (String [] args)
{
int user = 20;
if (user < 21)
{
System.out.println("User is less than 21");
}
}
}
```

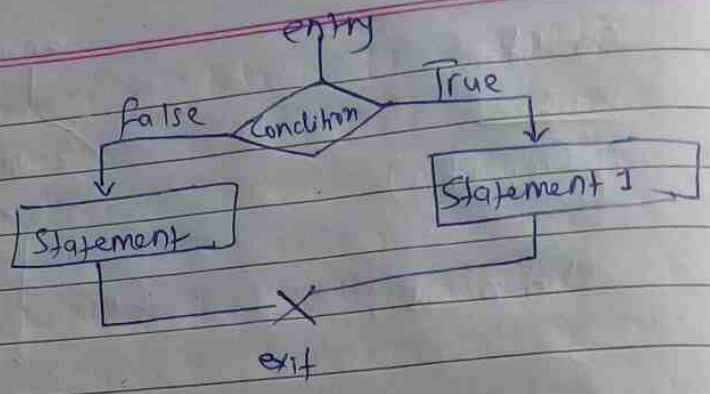
b) The If... else statement

In this control structure, if the condition is true, then statement 1 is executed. Otherwise statement 2 is executed. The else portion of the If... else statement is optional.

Mukesh Singh SMC

Syntax,

```
if(condition)
statement 1;
else
statement 2;
}
```



Example

```
class Ifelse statements
{
public static void main(String[] args)
{
int user=20;
if(user<=21)
{
System.out.println("User is 21 or younger");
}
else
{
System.out.println("User is older than 21");
}
}
```

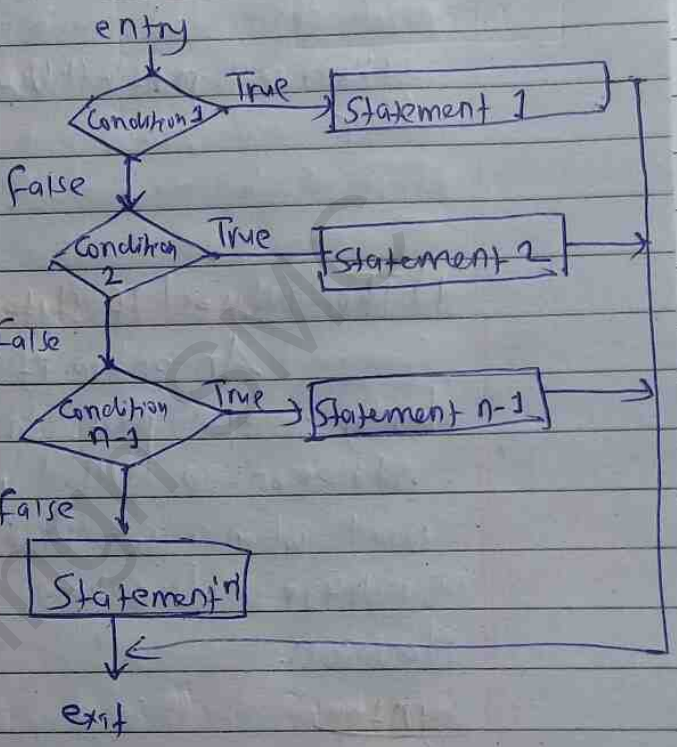
c) If-- else if Statement

~~In this~~ It is also known as ladder type if else statement. In this statement, if the condition 1 is true, statement 1 will be executed otherwise condition 2 is checked. If condition 2 is true ~~is~~, statement 2 will be

Mukesh Singh SMC

executed and similarly goes for next conditions. If all conditions are false then last statement n will be executed.

Flowchart



Syntax

```
if (Condition 1)
 Statement 1;
else if (condition 2)
 Statement 2;
  :
else if (Condition n-1)
 Statement n-1;
else
 Statement n;
  }
```

Example

```
    class IfElseIf_statements
    {
public static void main (String[] args)
  {
  int user=21;
  if (user<=21)
    {
System.out.println("User is 21 or Younger");
  }
else if (user>21 && user<50)
  {
```

Mukesh Singh SIMC

```
System.out.println("User is between 19 & 49");
}
elseif(user >50)
{
System.out.println("User is older than 50");
}
}
```

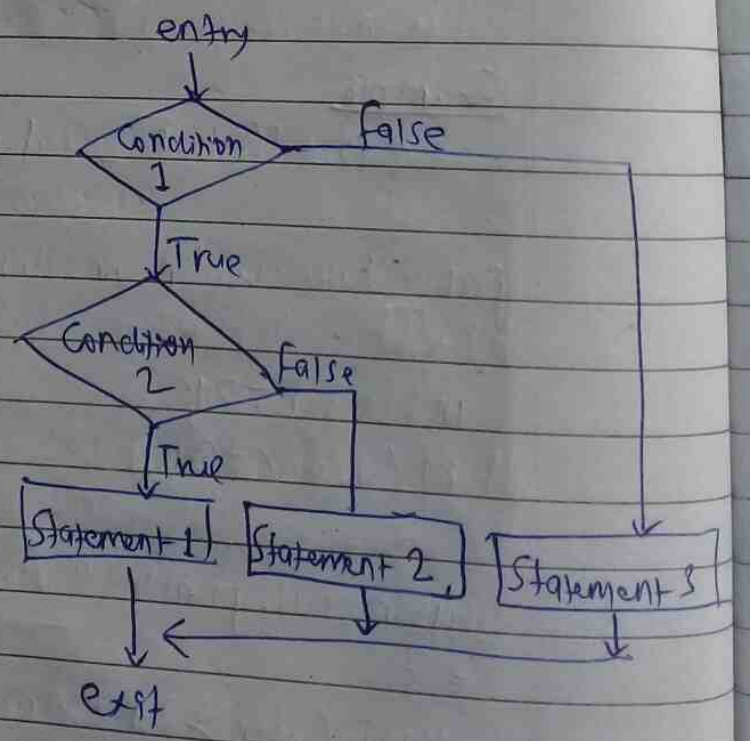
d) The Nested If Statement

An entire if else statement written within the body of if part or else part of another if else statement is called nested if else statement. It is used when a condition is to be checked inside another condition at a time in the same program to make decision.

Syntax

```
if(condition 1)
{
if (Condition 2)
 Statement 1;
else
 Statement 2;
}
else
statement 3;
}
```

flowchart



Mukesh Sharma

Example

```
Class NestedIfelseStatement
{
public static void main (String [] args)
{
int user = 21;
if (user <21)
{
System.out.println ("21 or younger");
}
if (user >18 && user <21)
{
System.out.println(" You are 19, 20 or 21");
}
}
~~if (user~~
else
{
System.out.println("18 or younger");
}
}
```

Mukesh Singh SMC

ii) Switch Statement

Switch is multiple branching statement. If a given problem has one ~~or more~~ condition and respective more than two action then we can use switch case statement. It checks the value of the variable to the case value, and then statements associated with it will be executed. If expression doesn't match any of the case value then the default statement will be executed.

Syntax

```
switch (expression)
  {
case expression 1: <statement 1>;
   break;
case expression 2: <statement 2>;
   break;
   :
case expression n-1 <statement n-1>;
   break;
default:
   statement n;
  }
```

Mukesh Singh SMC

Example

```
class SwitchStatement
{
public static void main (String [] args)
{
 int day=5;
 Switch (day)
 {
 Case 1: System.out.println("Monday");
  break;
 Case 2: System.out.println("Tuesday");
  break;
 Case 3: System.out.println("Wednesday");
  break;
 Case 4: System.out.println("Thursday");
  break;
 Case 5: System.out.println("Friday");
  break;
 Case 6: System.out.println("Saturday");
  break;
 Case 7: System.out.println("Sunday");
 break;
 default: System.out.println("Invalid entry");
 break;
 }
```

Mukesh Singh SMC

2. Looping / Iterative / Repetitive Statements

Loop is a control structure that executes the same program statement or block of program statements repeatedly for specified number of times or till the given condition is satisfied. Loop structure is used to carry out looping.

There are 3 types of loop:

i) The for loop

The for loop is most commonly used loop in any programming language. It executes the program statements or the block of statements repeatedly for specified number of times.

Syntax

```
for (initialization; condition; increment / decrement)
{
statements;
}
```

Example

```
public class forloop
{
public static void main (String [] args)
{
for (int i = 1; i < 10; i++)
{
System.out.println ("Welcome to my College");
}
}
}
```

Mukesh Singh SMC

ii) The ~~do~~ while loop

It executes the program statements repeatedly until the given condition is true. It checks the condition at first; if it is found true then it executes the statements written in its body part (inside parenthesis) otherwise it just gets out from the loop structure. It is also known as entry control or pre-test loop.

Syntax

```
initialization
while(condition)
{
statements;
increment/decrement;
}
```

where initialization means starting point, condition means stopping point and increment/decrement means counter.

Example

```
public class whileloop
{
public static void main (String [] args)
{
int i = 1;
while (i<=10)
{
System.out.println(i);
i++;
}
```

Mukesh Singh SMC

iii) The do..while loop

It also executes the ~~given~~ program statements repeatedly until the given condition is true. It executes the program statements once at first then only condition is checked. If condition is found true, then it executes the program statements again, otherwise it gets out from the loop structure. As it checks condition at last, it is also known as post-test loop or exit control loop.

Syntax

```
initialization
do
{
Statements;
= = =
increment/decrement;
}
while(condition)
```

Example

```
public class doWhileLoop
{
Public static void main(String [] args)
{
int i=1;
do
{
~~Print~~ System.out.println(i);
```

Mukesh Singh SMC

```
i=i+1;
}
while (i<=10){
}
}
```

3. Branching / Jumping Statements

Java provides the three branching statements; break, continue and return. Jumping statements are particularly used to jump execution of program statements from one place to another place inside a program.

i) break statement

Break statement is used inside loop or switch statement. When compiler finds the break statement inside the loop, compiler will abort the loop and continue to execute statements followed by loop.

Example

```
class BreakStatement
  {
public static void main (String args [])
  {
  int i;
for (i=1; i<=5; i++)
  {
if (i==4)
```

Mukesh Singh CMC

```
{
break;
}
System.out.println(i);
}
}
}
```

Output
1
2
3
4

ii) Continue Statement

It is used to continue the normal flow of program statement execution in loop. When continue is encountered in the program (loop body) then that particular iteration is skipped and the loop will be continued with the next iteration.

Example

```
Class ContinueStatement
{
public static void main (String args[])
{
int i;
for (i=1; i<=10; i++)
{
if (i==4)
{
continue;
}
System.out.println(i)
}
```

Mukesh Singh SMC

}
}

Output

1 2 3 4 5

iii) The Return Statement

The last control statement is return. The return statement is used to explicitly return from a method. It is an optional statement. That is, even if a method doesn't include a return statement, control returns back to the caller method after execution of the method. Return statement may or may not return parameters to the caller method.

Mukesh Singh 2016

2.7. Arrays

An area is a collection of similar type of data items treated as single unit. It is a container that holds data (values) of ~~the~~ one single type.

It can hold both primitive and object type data.

Each item in ~~the~~ an array is called an element and each element is accessed by its numeric index.

Array data types may be: int, float, char etc.

Characteristics of array:

- All array elements share a common name.
- Retrieve and sort the data easily.
- Arrays are created during runtime.
- Arrays are fixed in size.

Types of Arrays:

1. One/single dimensional arrays

An array which has only one subscript is named as one dimensional array. A subscript is ~~the~~ a number of large bracket in which we put the size of array.

Declaration of Single Dimensional Array:

Giving the name and type to the arrays is called declaration of the array.

~~For example,~~ Syntax

data_type array_name [array_size];

Example

int age [5];

Mukesh Singh SMC

Creating, initializing and accessing an Array:

int marks[] = {19, 10, 8, 17, 9};

mark[0]	mark[1]	mark[2]	mark[3]	mark[4]
19	10	8	17	9

Example

```
class Single array
{
public static void main (String args[])
{
int a[] = {33, 3, 4, 5}
for (int i = 0; i < a.length; i++)
System.out.println(a[i]);
}
}
```

Output

33
3
4
5

Mukesh Singh SMC

2. Multidimensional arrays

The multidimensional array is an Array of arrays (having more than one dimension).
In multidimensional arrays, data is stored in the form of rows and columns.

Declaration of multidimensional arrays:

Syntax

data_type array_name [row_size][column_size];

Eg: int a[2][2]

Creating, initializing and accessing multidimensional arrays

int matrix [2][2] = {2, 3, 1, 4};

where

A [0][0] = 2
A [0][1] = 3
A [1][0] = 1
A [1][1] = 4

Example program of multidimensional array:

```
Class multidimensionalarray
{
Public Static void main (String args[])
{
int arr[][] ={{1,2,3}{2,4,5}{4,4,5}};
for (int i=0; i<3; i++)
{
```

Mukesh Sir SMC

```
for (int j = 0; j<3; j++)
  {
   System.out.println (arr[i][j] + " ");
  }
 System.out.println ();
    }
  }
}
```

Output

1	2	3
2	4	5
4	4	5

Mukesh Singh SMC

Some programs in Java:

①

```
class IfElseTest {
public static void main (String args[]) {
int x=9;
if(x>10)
System.out.println(" Greater than 10");
else
System.out.println(" Smaller than 10");
}
}
```

②

```
import.java.util.scanner;
class IfElseTest {
public static void main(String args[]){
int num;
Scanner in = new scanner(System.in);
System.out.println("Enter the number:");
num = in.nextInt();
If(num%2 ==0)
System.out.println("The number is even");
else
System.out.println(" The number is odd");
}
}
```

Mukesh Singh SMC

③

```
Class WhileLoop Test
  {
  public static void main (String args[]){
   int i=1;
  While (i<10)
    {
System.out.println (i);
    i=i+2;
    }
  }
```

④ WAP to print area of rectangle:

```
public class rectangle {
 public static void main (String args[])
  {
  int ~~width~~ length = 5;
  int  breadth = 10;
  int area = ~~width~~ length * breadth;
System.out.println ("Area of rectangle =" + area);
    }
  }
```

OR

```
    import java.util.Scanner;
  Class Area of Rectangle {
public static void main (String args[])
```

Mukesh Singh SMC

```
{ int length, breadth, area;
Scanner in = new Scanner (System.in);
System.out.println(" Enter the length : ");
length = in.nextInt();
System.out.println(" Enter the breadth : ");
breadth = in.nextInt();
area = length * breadth;
System.out.println(" Area of rectangle is : " + area);
    }
}
```

⑤ Matrix Multiplication

```
public class Matrixmultiplication
{
public static void main(String args[])
{
int a[][] = {{1,1,1},{2,2,2}{3,3,3}};
int b[][] = {{1,1,1},{2,2,2}{3,3,3}};
int c[][] = new int[3][3];
for(int i=0; i<3; i++)
{
for (int j=0; j<3; j++)
{
c[i][j] = 0;
for (int k=0; k<3; k++)
{
c[i][j] += a[i][k] * b[k][j];
```

Mukesh Singh SMC

```
    }
    System.out.println(c[i][j] + " ");
  }
  System.out.println();
  }
}}
```

Mukesh Singh SMC

Unit-3

Object & Class

3.1. An Introduction to Object-Oriented Programming:

The programming in which data is logically represented in the form of a class and physically represented in the form of an object is called object oriented programming.

Characteristics of OOP:

- Emphasis on data rather than procedure.
- Programs are divided into objects.
- Function and data are tied together in a single unit.
- Data is hidden and cannot be accessed by external functions.
- Follows bottom-up approach of program design.

Advantages

- We can eliminate redundant codes by using inheritance feature of OOP.
- System can easily be upgraded from small to large.
- Easy for managing complex and large size problems.
- Efficient for testing and implementation of the system.
- Code reusability is easier.
- Follows bottom up approach.

Disadvantages

- Compiler and runtime overhead.
- Requires the mastery in software engineering.
- Difficult to debug and trace errors.

Mukesh Singh SMC

Features of OOP

Major features of OOP are as follows:

1. Object

An object is any entity that has state (attribute) and behaviour. For example: a dog is an object ~~because~~ having attributes such as colour, weight, age, type etc. and behaviours such as barking, wagging tail, eating etc. Object can be physical or logical. Some examples of objects are: chair, pen, table, keyboard etc.

In OOP, an object can be defined as an instance of a class.

In OOP, entities are represented as objects in the program. Attributes are represented by data (variables) and behaviours are represented by functions.

An object can communicate with others by using message passing mechanism.

2. Class

A class is the collection of similar objects. It is a logical entity. ~~It~~

It is user defined data type used to declare the object.

In OOP, it is most import to create class for representing data. It doesn't consume any space.

for example: mango, apple, orange are objects/members of class fruit. If fruit is declared as a class, then the statement fruit mango will create an object mango belonging to the class fruit.

3. Inheritance:

Creating a new class from an existing class is called as inheritance. Objects of new class acquires the attributes and behaviours of the existing parent class.

The newly created class is called derived class or child class or sub-class and the existing class is called base class or parent class or super class.

Advantage of inheritance is reusability.

4. Data Encapsulation

Wrapping up of data and function ~~into~~ together into a single unit is called data encapsulation. For example capsule is wrapped with different medicines.

A java class is the example of encapsulation.

5. Data Abstraction

The process of hiding internal details of a class from rest of the program (after declaring private) and showing functionality is data abstraction.

It is an act of representing essential features without including the background details.

In Java, we use abstract class and interface to achieve abstraction.

6. Polymorphism:

Polymorphism means "having many forms". If one task is performed in different ways, it is polymorphism.

So, ~~Polymorphism~~ the different ways of using

the same function or operator depending upon what they are operating is called polymorphism.
Example: operator overloading and function overloading.

Procedure Oriented Programming	Object Oriented Programming
1. Follows top-down approach of program design.	1. Follows bottom-up approach of program design.
2. Emphasis on procedure (function).	2. Emphasis is given on data.
3. Generally, data cannot be hidden.	3. Data can be hidden so that non-member function cannot access them.
4. Programs are divided into functions.	4. Programs are divided into objects.
5. Maintaining & enhancing code is still difficult.	5. Maintaining & enhancing code is easier.
6. Data move from function to function.	6. Data & function are tied together.
7. Code reusability is still difficult.	7. Code reusability is easy in compared to POP.
8. Examples: FORTRAN, COBOL, C, Pascal, C-BASIC.	8. Examples: C++, JAVA, C#, Python, SmallTalk etc.

Mukesh Singh SMC

3.2 Using Predefined Classes

In Java, using predefined class name as class or variable name is allowed. However, according to Java Specification language the basic rule for naming in Java is that you cannot use a keyword as name of a class, name of a variable, nor the name of a folder used for package.

Using any predefined class in Java won't cause such compiler error as Java ~~specified~~ predefined classes are not keywords.

Following are some invalid variable declaration in Java:

→ boolean break = false; // not allowed as break is keyword.

→ in boolean = 8; // not allowed as boolean is keyword

→ boolean goto = false; // not allowed as goto is keyword.

Using predefined class name as user defined class name in our program:

```
// Number is predefined class name in java.lang. package
// Note: java.lang.package is included in every java program
  by default.
public class Number
{
  public static void main (String args[])
  {
  System.out.println ("It works");
    }
}
```

3.3. Defining Your Own 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.

Example

```
public class StudentRecord {
  //we will add more code here later
}
```

- where:

→ 'public' means our class is accessible to other classes outside the package.

→ 'class' is the keyword used to create a class in Java.

→ 'StudentRecord' is a unique identifier that describes our class.

Create a class called "MyClass" with a variable x:

```
public class MyClass
{
int x=5;
}
```

Mukesh Singh SMC

3.4. Static Fields and Methods

Static field

A Java field can be static or non-static. A static field belongs to the class, not instances of the class. If a field is declared static then there is one field for the entire class instead of one per object.

A static field of a class is often referred to as a class variable because static fields associated with a class and not with individual instances of the class. A static field gets memory only once for the whole class no matter how many objects of a class are created.

To make a static field, you simply place the 'static' keyword in the field declaration.

Eg: ①

```
public class Human{
    public static int numHumans = 7;
    ---
}
```

Eg ②

```
public class Customer{
    static String staticfields;
}
```

Once the static field exists, you can access it from outside or inside of the class without an instance (object) of the class.

Mukesh Singh SMC

Properties of Static field:

A static field of class has following characteristics:

i) It is a variable which belongs to the class and not to object (instance).

ii) Static field (variables) are initialized only once, at the start of the execution.

iii) A static field can be accessed before any object of a class is created without reference to any object.

or

A static field (variable) can be accessed directly by the class name and doesn't need any object.

iv) There is only one copy of static field in a class which can be shared by/among all objects of the class.

v) The static field is similar to static variable. The main difference is that a static variable is used to retain information between calls ~~and~~ whereas static field is used to share information among multiple objects of a class.

Mukesh Shrivastava

Static Methods:

When a method is declared with 'static' keyword, it is known as static method. It is a method in Java which belongs to the class and not to the object. A static method can access only static data.

Properties / Characteristics of static methods:

i) A static method belongs to the class rather than the instance (object) of a class.

ii) A static method can access only static data. It cannot access non-static data (instance variables)

iii) A static method can call only other static methods and cannot call a non-static method from it.

iv) A static method ~~cannot~~ be accessed directly by the class name and ~~don't~~ doesn't need any object.

v) A static method cannot refer to "this" or "super" keywords in anyway.

Mukesh Singh SMC

Example of static method in Java to get cube of given number:

```
class Calculate
{
static int cube (int x)
{
return x*x*x;
}
public static void main (String args[])
{
int result = Calculate.cube(5);
System.out.println(result);
}
}
```

Output : 125

Restrictions for the Static method:

There are two main restrictions for static method:

1. The static method cannot use non static data member or call non-static method directly.

2. This and super cannot be used in static context.

Example

```
class A {
int a = 40; //non-static
public static void main (String args[]){
System.out.println (a);
}}
```

Mukesh Singh SMC

Output

Compile Time Error

Q1 Why is Java main method static?

→ It is because ~~mai~~ the object is not required to call a static method.

Q2 Can we execute a program without main( ) method?

→ Not possible.

Mukesh Singh SMC

3.5. Method Parameters:

- What is passed "to" a method is referred to as an "argument". The type of data that a method can receive is referred to as "parameter".
- A parameter is a value passed to the method. Methods can take one or more parameters.
- If methods work with data, we must pass the data to the methods. This is done by specifying them inside the parenthesis.
- In the method definition, we must ~~pass~~ provide a name and type for each parameter.

A method's declaration includes a list of variables which tells us type and order ~~and~~ of variables that the method can accept. This is called the "method's parameters".

Example In the following example:

~~// Invoke (call) the method~~

~~int number1~~ = 25;

int

Mukesh Singh SMC

Classication by example

Formal parameters

```
// Method definition
public int add(int x, int y)
{
return (x+y);
}
```

Note: Actual parameters (arguments) are variable names used in method call.

Formal paramaters should be of same data type and used in method definition.

```
// Invoke (call) the method
// where add method is used
int number 1 = 25;
int number 2 = 47;
int sum = add(number 1, number 2);
```

actual parameters or arguments

Passing parameters / arguments by Value:

In Java, parameters sent to methods are passed by-value.

In Java, arguments/parameters are always passed by value to methods.

Pass-by-value means that when you call a method, a copy of the value of each actual parameter (argument) is passed to the method. This copy can be changed inside the method, but such a change will have no effect on the argument.

Note: Java does not support passing arguments by reference, like C# or C++.

Example of pass-by-value:

①
```
public class ChangeParameters
{
public static void main (String args[])
{
int number = 1;
tryToChangeNumber (number);
System.out.println (number);
}
public static void tryToChangeNumber (int i)
{
i = 2;
}
}
```

② Program to swap two numbers using pass-by-value

```
public class Swapping
{
public static void main (String args[])
{
int x, y, temp;
System.out.println ("Enter value of x and y");
Scanner in = new Scanner (System.in);
x = in.nextInt ();
y = in.nextInt ();
```

Mukesh singh GMC

```
System.out.println("Before swapping\n x="+x"\ny="+y);
  temp=x;
  x=y;
  y= temp;
System.out.println("After swapping \n x="+x+"\ny="+y);
  }
}
```

OR

```
public class swapping
{
public static void main (String args[])
{
int a=10;
int b=20;
System.out.println("Before Swapping");
System.out.println("Value of a is :"+a);
System.out.println("Value of b is :"+b);
swap(a, b); // Swap the value
}
public static void swap(int a, int b)
{
int temp;
temp = a;
a= b;
b= temp;
System.out.println("After Swapping");
}}
```

Mukesh Singh SMC

Output :- Swapping

Before Swapping
Value of a is : 10
Value of b is : 20
After Swapping
Value of a is : 20
Value of b is : 10

Mukesh Singh SMC

3.6. Object Construction

A member function with same name as its class is called a constructor.

It is a special type of method which is used to initialize the objects of its class. It is called when an instance of object is created and memory is allocated for the object.

It is called constructor because it constructs the values at the time of object creation.

Rules for creating a Java Constructor:

i) Constructor name must be the same as its class name.
ii) A Constructor must not have explicit return type.
iii) It should not return a value not even void.
iv) A Java constructor cannot be abstract, static, final and synchronized.

Types of Java Constructors:

There are two types of constructors in Java.

1. Default Constructor (No-arg Constructor)

A constructor is called default constructor when it doesn't have any parameter.

Syntax

```
<class_name> () {}
Sukuma () {}
```

Mukesh Shrivastava

Example of default constructor:

```
Class Bike
{
Bike() // creating default constructor
{
System.out.println("Bike is created");
}
public static void main(String args[])
{
Bike b=new Bike();  //calling default constructor
}
}
```

Output

Bike is created.

2. ~~Parameterized~~

Q. What is the purpose of default constructor?

→ The default constructor is used to provide the default values to the objects like 0, null etc depending on the type.

Mukesh Singh SMC

2 ~~Default~~ Parameterized Constructor

A constructor which has a specific number of parameters (arguments) is called a parameterized constructor.

Example of parameterized constructor:

```
class Student
{
int roll;
Student (int x)
{
roll = x;
}
void display ()
{
System.out.println("roll");
}
public static void main(String args[])
{
Student s1 = new Student(7);
Student s2 = new Student (10);
s1.display();
s2.display();
}
}
```

Mukesh Singh SMC

Constructor Overloading in Java:

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task.

Example program:

```
class Student
  int roll;
  String name;
  int age;
  Student(int x, String n) //creating two arg constructor
  {
  roll=x;
  name=n;
  }
  Student(int x, String n, int a) //creating three arg constructor
  {
  roll=x;
  name=n;
  age=a;
  }
  void display()
  {
System.out.println(roll+" "+name+" "+age);
  }
```

Mukesh Singh SMC

```
public static void main (String args[])
{
Student s1 = new Student (7, "Ram");
Student s2 = new Student (30, "Shyam", 25);
s1. display();
s2 display();
}
}
```

Output.
7 Ram 0
25 Shyam 25

	Java Constructor	Java Method
1.	It is used to initialize the state of an object.	1. It is used to expose the behaviour of an object.
2	It must not have a return type.	2 It must have a return type.
3.	It is invoked implicitly.	3. It is invoked explicitly.
4.	The Java compiler provides a default constructor if you don't have any constructor in a class.	4. The method is not provided by the compilor in any case.
5.	The constructor name must be same as the class name.	5. The method name may or may not be same as class name.

Mukesh Singh SMC

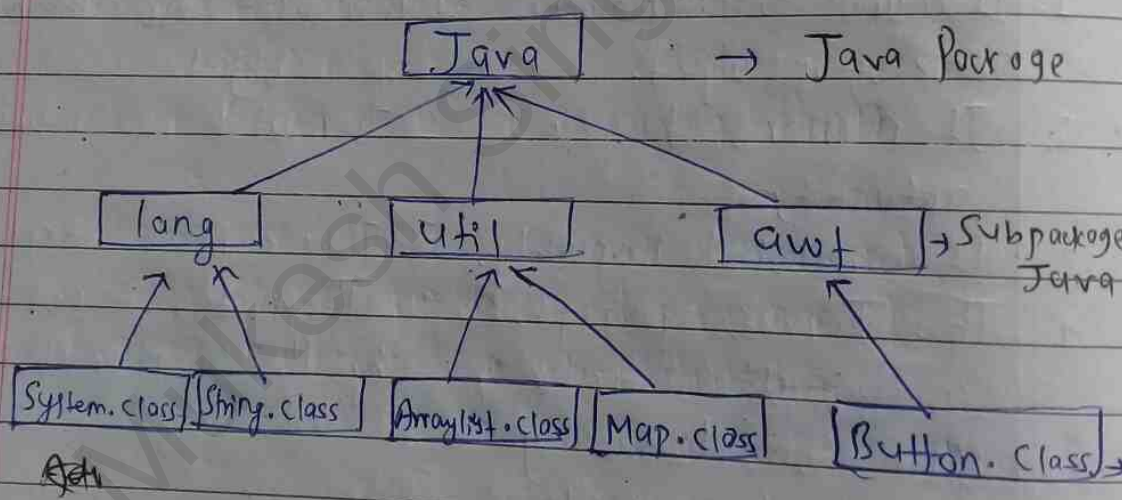
3.7. Packages:

A package is a group of similar types of classes, interfaces, enumerations or sub-packages.

Packages are used in Java in order to prevent naming conflicts, to control access ~~of class~~, to make searching/locating and usage of classes, interfaces, enumerations ~~and~~ easier, etc.

Package in java can be categorized in two form: built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.



Advantages of Java Package:

- Categorizes the classes & interfaces so that they can be easily maintained.
- Java package provides access protection.
- Java package removes naming collision.

Mukesh Singh SYC

Simple example of Java Package:

The 'package' keyword is used to create a package in Java.

```
//save as simple.java
package mypack;
public class Simple
{
public static void main (String args[])
{
System.out.println("Welcome to package");
}
}
```

[It is example of user defined package.]

How to compile & run Java package:

If you are not using any IDE:

To compile: java -d . Simple.java

To run : java mypack.Simple

Output

Welcome to package

Subpackage in Java:

Package inside the package is called the subpackage. It should be created to categorize the package further.

Example

```
package com.javatpoint.core;
class Simple
{
```

Mukesh Singh SMC

```
public static void main (String args[])
  {
System.out.println ("Hello Subpackage");
  }
}
```

To compile: java -d . simple.java
To run: java com.javatpoint.com.simple

Output: Hello Subpackage

Example of Java built-in package:

```
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 : " + userName);
    }
}
```

Mukesh Singh SMC

Access Modifiers

Access modifiers are keywords that are used to set/define the accessibility (visibility) of classes, variables, methods, data members and constructors.

There are four types of Java access modifiers:

1. Default (visible to the package, no modifiers needed)
2. Private (visible to the class only)
3. Public (visible to the world)
4. Protected (visible to package and all subclasses).

~~1. Default access modifier~~

For better understanding:

Access Modifiers	Same Class	Same Package	Subclass	Other Packages
Public	Y	Y	Y	Y
Protected	Y	Y	Y	N
Default	Y	Y	N	N
Private	Y	N	N	N

1. Default access modifier

If we don't use any modifier, it is called default access modifier.

~~The scope of~~ This modifier is ~~limited to~~ accessible within the package only. This means if we have a class with the default access modifier in a package, only those classes that are in this package can access this class. No other class outside the package can access this class.

Mukesh Singh SMC

Example of default access modifier:

```
package default package;
  class A
   {
void Message()
  {
System.out.println("Hello");
   }
  }

// save by B.java
package mypack;
import pack.*;
class B
  {
public static void main (String args[])
  {
A obj = new A(); // Compile time error
~~obj.msg(); //Compile time error~~
obj.message(); // Compile time error
  }
}
```

In the above example the scope of class A and its method message() is default so it cannot be accessed from outside the package.

Mukesh singh SMC

2. Private access modifier

The private access modifier is accessible only within class. It means the methods, variables and constructors that are declared as private can only be accessed within the declared class itself. It is very restrictive access modifier.

Note: We cannot make class and interfaces private.

Example program:

```
Class A
{
private int data= 40;
private void msg()
{
System.out.println("Hello java");
}
}
public class Simple
{
Public static void main(String args[])
{
A object=new A();
System.out.println(obj.data); // Compile time error
obj.msg(); //Compile time error
}
}
```

Here we have created two classes A and Simple. A class contains private data member and

Mukesh Singh SMC

private method. We are accessing there private members from outside the class, so there is compile time error.

3. Public Access Modifier:

The public access modifier is accessible everywhere. It means a class, method, constructor, interface etc declared as public can be accessed from any other class. It has the widest scope among all other modifiers.

Example program

```
//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();
```

Mukesh Singh SMC

```
}
}
```

Output
Hello

4. Protected Access modifier

The protected access modifier is accessible within package and outside the package but through inheritance only. It means variables, methods and constructors which are declared as protected in a base class can be accessed only by the child class in other package.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Example program

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

Mukesh Chandra SMC

```
//save by B.java
package mypack;
import pack*;
class B extends A
{
public static void main (String args[])
{
B obj=new B();
obj.msg();
}
}
```

Output: Hello

In the above example, we have created two packages pack and my_pack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

Mukesh Singh SMC

Function / Method Overloading:

If two or more methods in a class have same name but different parameters, it is known as method overloading.

Overloading always occur in the same class (unlike method overriding).

Advantage

Method overloading increases the readability of the program.

~~Differ~~ Ways of method Overloading:

~~1. By changing~~

1. Method Overloading by changing number of arguments:

Syntax

add(int, int)

add(int, int, int)

Example program:

```
Class ~~adder~~ Method overloading
{
void add(int a, int b)
  {
  System.out.println("Sum is :"+(a+b));
  }
void add (int a, int b, int c)
  {
System.out.println("Sum is :" +(a+b+c));
```

Mukesh Singh SMC

```
}
public static void main (String args[])
{
~~function~~
Method overloading obj = new ~~func~~ Method overloading();
obj.add (10,20);
obj.add (10,20,30);
}
}
```

Output:
Sum is 30
Sum is 60

In this example, we have created two methods, first add() method performs addition of two numbers and the second add method performs addition of three numbers.

2. Method overloading by changing data type of arguments:

Syntax

```
add(int, int)
add(int, float)
```

Mukesh Singh SMC

Example program:

```
class Methodoverloading
{
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[])
{
Method overloading obj = new method overloading ();
obj.sum(8,5);
obj.sum(4.0f, 3.8f);
}
}
```

Output

Sum is 13
Sum is 8.4

In this example, two methods differ in data types. The first add method receives two integer arguments and the second add method takes two float arguments.

Mukesh Sir of SMC

Method Overriding:

If a subclass (child class) has the same name method as declared in the parent class, it is known as method overriding in Java.

Uses of method overriding:

- It is used to provide the specific implementation of a method which is already provided by its superclass.
- It is used in runtime polymorphism.

~~Rules for~~

Rules for Java method Overriding:

- The method must have same name as in the parent class
- The method must have same parameters as in the parent class
- There must be an Is-A relationship (inheritance)

Example program:

```
class Vehicle // creating parent class
  {
void run() //defining method
  {
System.out.println("Vehicle is running");
  }
}
class Bike extends Vehicle //creating child class
  {
void run()   // same method as in parent class
  {
System.out.println("Bike is running safely");
  }
```

Mukesh Singh SMC

```
public static void main (String args[])
{
Bike obj = new Bike();   //creating object
obj.run();   //calling method
}
}
```

Output

Bike is running safely.

In the above example, we have defined run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

Q. Why method overloading is not possible by changing the return type of method?

→ Because there may occur ambiguity.

Mukesh Singh GMC

Method Overloading	Method Overriding
1. If a class has more more than one method with same name but with different parameters, then we call it as method overloading.	1. ~~When~~ If a subclass (child) has same method as declared in the parent class, we call it as method overriding.
2. It is called compile time polymorphism.	2. It is called runtime polymorphism.
3. It shows static polymorphism.	3. It shows dynamic polymorphism.
4. It is possible in single class only i.e only one class is required.	4. It requires two classes - super (parent) class and sub class (child).
5. It doesn't need inheritance.	5. It needs inheritance.
6. Private & final methods can be overloaded.	6. Private & final methods can't be overridden.
7. Uses static binding.	7. Uses dynamic binding.
8. Access modifier can be any.	8. Access modifier must be public.

Mukesh Singh SMC

Keyword "This"

In Java, 'this' keyword is a reference variable that refers to the current object inside the methods or constructors.

There can be lots of usage of java 'this' keyword. Various usage of 'this' keyword are mentioned below:

- this can be used to refer current class instance variable.
- this can be used to invoke or initiate current class method (implicitly).
- this() can be used to invoke or initiate current class constructor.
- this can be passed as an argument in the method call.
- this can be passed as an argument in the constructor call.
- this can be used to return the current class object (instance) from the method.

1. Using 'this' keyword to refer current class instance variables

```
Class Test
{
int a;
int b;
Test (int a, int b)
{
this.a = a;
this.b = b;
}
void display ()
```

Mukesh Singh SMC

```
{
 System.out.println("a =" + a + "b =" + b);
 }
 public static void main (String argn[])
 {
 Test object = new Test (10, 20);
 Object.display ();
 }
}
```

Output
a = 10, b = 20

2 Using 'this' keyword to invoke current method

```
Class Test
 {
 void display()
 {
 this.show();
 System.out.println(" Inside display function");
 }
 void show()
 {
 System.out.println(" Inside show function");
 }
 Public static void main (String args[])
 {
 Test t1 = new Test ();
 t1.display();
 }
}
```

Mukesh Singh SMC

Output.
Inside show function
Inside display function.

3. Using 'this' keywork to invoke current class constructor

```
class Test
{
int a;
int b;
Test()   // default constructor
{
this(10,20);
System.out.println(" Inside default Constructor \n");
}
Test(int a, int b)  // parameterized constructor
{
this.a = a;
this.b = b;
System.out.println(" Inside parameterized constructor");
}
public static void main (String args[])
{
Test object = new Test();
}
}
```

Output
Inside default Constructor
Inside parameterized Constructor

Mukesh Singh SMC

4. Using 'this' keyword as method parameter

```
class Test
{
int a;
int b;
Test()    // default Constructor
{
a=10;
b=20;
}
void display(Test obj) // method that receives 'this' keyword as parameter
{
System.out.println("a="+a+" b="+b);
}
void get()  // method that returns current class instance
{
display(this);
}
public static void main(String args[])
{
Test object=new Test();
Object.get();
}
}
```

Output

a=10 b=20

Mukesh Singh SMC

5. Using 'this' keyword as an argument in constructor call

```
Class A
 {
 B obj;
A(B obj)  // Parameterized constructor with object of B
 {          // as parameter
this.obj = obj;
 Obj.display();  // calling display method of class B
 }
}
class B
 {
 int x=5;
 B()
 {
 A obj = new A(this);
 }
void display()
{
System.out.println("Value of x in class B:"+x);
 }
public static void main (String argv[])
  {
  B obj = new B();
 }
}
```

Output

Value of x in class B: 5

Mukesh Singh SMC

6. Using 'this' keyword to return the current class instance

```
class Test
{
int a;
int b;
Test() //default constructor
{
a=10;
b=20;
}
Test get() //method that returns current class instance
{
return this;
}
void display()
{
System.out.println("a="+a +" b="+b);
}
public static void main(String args[])
{
Test Object=new Test();
Object.get().display();
}
}
```

Mukesh Singh SMC

Proving 'this' keyword:

```
class A {
void m() {
System.out.println(this); // prints same reference ID
}
public static void main(String args[])
{
A obj = new A();
System.out.println(obj); // prints the reference ID
obj.m();
}
}
```

Output

```
A@22b3ea59
A@22b3ea59
```

In the above program, we are printing the reference variable and this, output of both variables are same.

Mukesh Singh SMC

Unit - 4

Inheritance & Interfaces

Creating a new class from an existing class is called inheritance. Objects of new class acquires the attributes and behaviours of the existing parent class.

Advantage of inheritance is reusability.

4.1 Classes, Super Class and Sub Classes

Classes:

A class is a group of similar objects which have common properties. It is a template or blue-print from which objects are created.

Super Classes:

Super class is a class which inherits the other class. It is also called a derived class or extended class or child class.

Super Class:

Super class is the class from where a subclass inherits the features. It is also called a base class or a parent class.

Syntax of Java inheritance:

```
class SubClass-name extends Superclass-name
{
 // methods & fields
}
```

The extend keyword indicates that you are making a new class that derives from an existing class.

Mukesh Singh SMC

Program showing Superclass and Subclass:

```
Class Teacher
{
void teach()
{
System.out.println("teaching");
}
}
Class Student extends Teacher
{
void learn()
{
System.out.println("learning");
}
}
Class TestInheritance
{
public static void main(String args[])
{
Student s = new Student();
s.learn();
s.teach();
}
}
```

Output

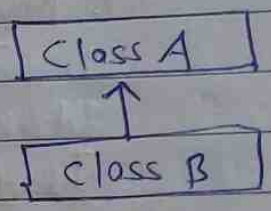
learning
teaching

Mukesh Singh SMC

Types of inheritance: in Java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchial.
In java programming, multiple & hybrid inheritance are supported through interface only.
Note: Multiple inheritance is not supported in Java through ~~1. Single Inh~~ class.

1. Single Inheritance:
When a class is derived from only one base class, it is called single inheritance.



Example program:

```
class Teacher
{
void teach()
{
System.out.println("teaching");
}
}
class student extends Teacher
{
void learn()
{
System.out.println("learning");
}
}
Class SingleInheritance
```

Mukesh Singh SMC

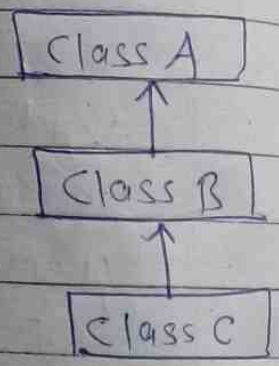
```
{
public static void main (String args [])
{
Student s = new Student ();
s.learn ();
s.teach();
}}
```

Output
learning
teaching

2 Multilevel Inheritance

The mechanism of deriving a class from another derived class is known as multilevel inheritance.

It is represented below:



Example program

```
Class Principal
{
void supervise ()
{
System.out.println ("Supervising");
}
}
Class Teacher extends Principal
{
void teach()
{
System.out.println ("Teaching");
}
}
```

Mukesh Singh SMC

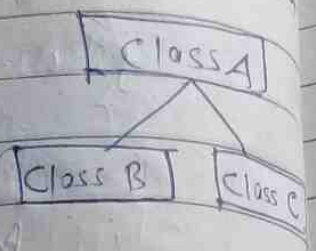
```
class Student extends Teacher
  {
  void learn()
  {
System.out.println ("Learning");
  }
}

Class Multilevel
  {
public static void main (String args[])
  {
Student s = new Student();
s.learn();
s.teach();
s.supervise();
}}
```

Output
learning
Teaching
Supervising

Mukesh Singh SMC

3. Hierarchial Inheritance

When two or more classes are derived from one base/parent class, it is called hierarchial inheritance.



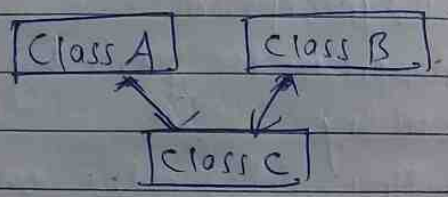
Example Program:

```
class College
 {
 void run()
  {
  System.out.println("running");
  }}
class Teacher extends College
 {
 void teach()
  {
  System.out.println("teaching");
  }}
class Student extends College
 {
 void learn()
  {
  System.out.println("learning");
  }}
class Hierarchial
 {
 public static void main(String args[])
  {
  Student S = new Student();
```

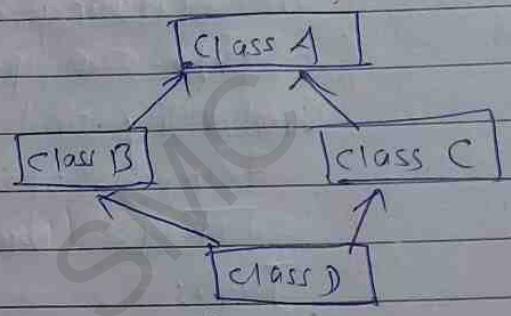
Mukesh Singh SMC

```
s.learn();
s.run();
}}
```

Output
learning
running



Multiple



Hybrid

Q. Why multiple inheritance is not supported in Java?
→ To reduce the complexity and simplify the language, multiple inheritance is not supported in Java.

Eg:
```
Class A {
  void msg() {
  System.out.println("Hello");}
  }
Class B{
  void msg() {
   System.out.println("Welcome");}}
Class C extends A,B{ //Suppose if it were
public static void main (String args[]){
  C obj = new C();
obj.msg(); // Compile time error
}
}
```

Mukesh Singh SMC

4.2 Polymorphism:

Polymorphism means "having many forms". If a single task is performed in different ways, it is polymorphism. So, the different ways of using the same function or operator depending on what they are operating is called polymorphism.

We can perform polymorphism in java by method overloading and method overriding.

Examples - operator overloading and function overloading

There are two types of polymorphism in Java:

1. Compile-time polymorphism
2. Run-time polymorphism

1. Compile-time polymorphism (static polymorphism)

Polymorphism that is resolved ~~at~~ during compile time is known as static polymorphism.

Method overloading is an example of compile time polymorphism.

Method overloading allows two or more methods having same name but different parameters.

Example program of Compile time polymorphism:

```
class StaticPolymor
{
void add(int a, int b)
{
System.out.println("Sum is:"+(a+b));
```

```
}
void add(int a, int b, int c)
{
System.out.println("Sum is :" + (a+b+c));
}
public static void main(String args[])
{
Static obj = new Static();
Obj.add(10,20);
obj.add(10,20,30);
}
}
```

Output
30
60

Here, we have two definitions of the same method add() which add method would be called is determined by the parameter list at the compile time. That is the reason, this is also called compile time polymorphism.

2 Runtime Polymorphism (Dynamic Method Dispatch)

Runtime polymorphism is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

Method overriding allows a subclass (child class) to have the same method as declared in the parent class.

Mukesh Singh SMC

Example of runtime polymorphism:

```
class Vehicle    //creating parent class
{
void run()
{
System.out.println("Vehicle is running");
}
}
class Bike extends Vehicle    // creating child class
{
void run()    //same method as in parent class
{
System.out.println("Bike is running safely");
}
public static void main(String args[])
{
Vehicle ~~Bike~~ b = new Bike();    //creating object
b.run();
}
}
```

Mukesh Singh SMC

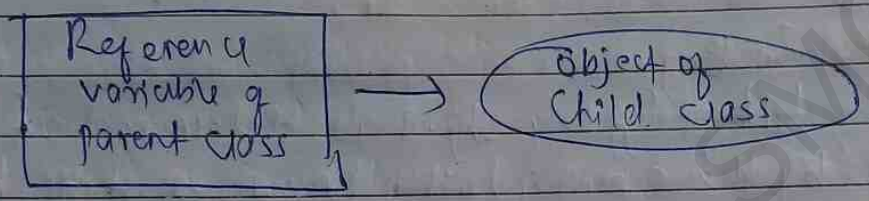
Page No.
Date

Upcasting

Converting a subclass type to a super class type is known as upcasting

If the reference variable of the parent class refers to the object of child class, it is known as upcasting.

for example:




```
Class Vehicle{
  ---
  }
Class Bike extends Vehicle{
  ---
  }
Vehicle V = new Bike();   // upcasting
```

Mukesh Singh SMC

Binding

Connecting a method call to the method body is known as binding.

There are two types of binding:

1. Static Binding (also known as Early Binding)
2. Dynamic Binding (also known as Late Binding)

1. Static Binding

When type of object is determined at compile time (by the compiler), it is known as static binding.

If there is any private, final or static method in a class, there is static binding.

Example of static binding:

```
Class Dog() {
private void eat()
  {
  System.out.println("Dog is eating");
  }
public static void main(String args[])
  {
  Dog d = new Dog();
  d.eat();
  }
}
```

(4.3) 2. Dynamic Binding

When type of object is determined at ~~compile~~ run-time, it is known as dynamic binding.

Example of dynamic binding:

```
class Animal
{
void eat()
{
System.out.println("animal is eating");
}}
class Dog extends Animal
{
void eat()
{
System.out.println("dog is eating");
}}
public static void main(String args[])
{
Animal a = new Dog();
a.eat();
}
}
```

Output
Dog is eating

In the above example, object type cannot be determined by the compiler, because the instance (object) of Dog is also an instance of Animal. So compiler doesn't know its type only its base type.

Mukesh Singh SMC

4.4. Final Classes and Methods

Final Classes

When a class is declared with final ~~clas~~ keyword, it is called final class. A final class cannot be extended (inherited).

There are two uses of final class:

i) first is definitely to prevent inheritance, as final classes cannot be extended.

ii) The other use is to create an immutable class like the predefined String class.

~~Example program of final class~~:

Note: Though a final class cannot be extended, it can extend other classes. In simpler words, a final class can be a sub class but not a super class.

Syntax

```
final class classname
  }
```

Example program:

```
final class Bike
  {
  }
class Hero extends Bike
  {
  void run()
    {
    System.out.println("running safely with 60 kmph");
    }
```

Mukesh Singh SMC

```
public static void main(String args[])
{
Hero h = new Hero();
h.run();
}
}
```

Output
Compile Time Error

Final Methods

When a method is declared with final keyword, it is called a final method. A final method cannot be overridden.

Example of final method:

```
Class Bike
{
final void run()
{
System.out.println("running "); }
}
Class Hero extends Bike
{
void run()
{
System.out.println("running safely with 60kmph");
}
public static void main(String args[])
{
```

Mukesh Singh SMC

```
Hero h = new Hero();
  h.run();
  }
}
```

Output
Compile Time Error

Q. Is final method inherited?
→ Yes, final methods inherited but you cannot ~~override~~ override it. for example:

```
Class Bike {
final void run(){System.out.println("running");}
  }
class Honda extends Bike{
public static void main(String args[]){
new Honda().run();
  }
}
```

Output
running

Mukesh Singh SMC

9.5 Abstract Classes:

Before learning abstract classes ~~but~~ for understand abstraction in Java:

Abstraction in Java:

The process of hiding internal details of a class and showing only functionality to the user is abstraction.
Abstraction can be achieved through (i) Abstract class and (ii) Interface

Abstract class:

A class which is declared with 'abstract' keyword is known as abstract class in Java. It can have abstract and non-abstract methods (method with the body). An abstract class can't be instantiated, which means you are not allowed to create object of it.

Rules of abstract class:

- An abstract class must be declared with an abstract keyword.
- It can have abstract & non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

```
Eg - abstract class A{}
```

Abstract method

A method which is declared as abstract and does not have implementation is known as abstract method.

```
Eg: abstract void printStatus();
```

Example of abstract class using abstract method:

```
abstract class Bike
{
abstract void run();
}
class Hero extends Bike
{
void run()
{
System.out.println("running safely");
}
public static void main(String args[])
{
Bike obj = new Hero();
obj.run();
}
}
```

Output:

running safely

Mukesh Sirjan SMC

4.7. Interfaces

An interface in Java is a blueprint of a class. It looks like a class but, it is not a class. It has static constants and abstract methods.

The interface in java is a mechanism to achieve abstraction. Interfaces can have abstract methods and variables. It cannot have a method body. Also the variables declared in an interface are public static & final by default.

Java uses interface to implement multiple inheritance.

Why use java interface?

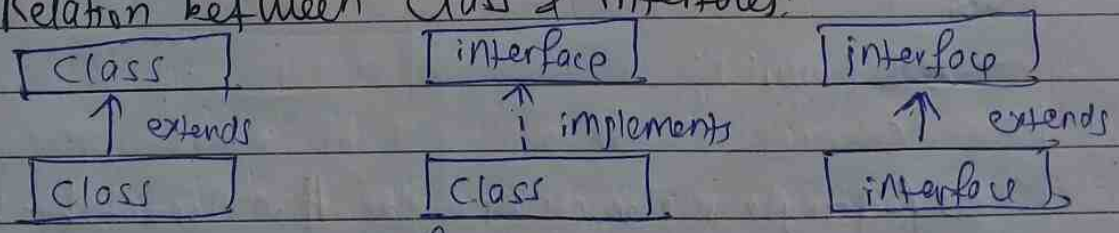
Mainly 3 reasons:

- To achieve abstraction.
- To implement multiple inheritance.
- To achieve loose coupling.

Syntax

```
interface <interface_name> {
  {
  // declare constant fields
  //declare methods that are abstract by default
  }
```

Relation between class & interfaces:



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

Mukesh Singh SMC

① Example of Java Interfaces:

```
interface printable {
 void print();
 }
class printer implements printable
 {
public void print()
 {
 System.out.println("Hello");
 }
public static void main(String args[])
 {
 printable p = new printer();
 p.print();
 }
}
```

Output
Hello

② Java Interface example : Drawable

```
interface Drawable // interface declaration by 1st user
 {
 void draw();
 }
Class Rectangle implements Drawable // implementation by 2nd user
 {
 public void draw()
 {
```

Mukesh Singh SMC

```
System.out.println(" drawing rectangle"); }
    }
class Circle implements Drawable
    {
public void draw()
    {
System.out.println ("drawing circle"); }
    }
Class TestInterface  // Using interface by 3rd user
    {
public static void main (String args[])
    {
Drawable d = new Circle(); // in real scenario, object is
d.draw();                  // provided by method().
}}
```

Output

Drawing Circle

Mukesh Singh SMC

Abstract class	Interface
1. Abstract class can have abstract & non-abstract methods.	1. Interfaces can have only abstract methods.
2. It doesn't support multiple inheritance.	2. It supports multiple inheritance
3. Can have final, non-final, static and non-static variables.	3. Has only static & final variables.
4. Can provide implementation of interface.	4. Can't provide implementation of abstract class.
5. Abstract keyword is used to declare abstract class.	5. Interface keyword is used to declare interface.
6. Can be extended using keyword "extends".	6. Can be implemented using keyword "implements".
7. Can have class members like private, protected etc.	7. Members of interface are public by default

8. Example
public abstract class Shape
{
public abstract void draw();
}
	8. Example
	public interface Drawable
	{
	void draw();
	}

Mukesh Singh SMC

Unit-5

Exception Handling and Multithreading

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.

What is Exception in Java?

→ → Exception is abnormal condition.

→ An exception is an unwanted or unexpected event (error), which occurs during the execution of a program i.e. at runtime, that disrupts the normal flow of the program. It is an object which is thrown at runtime.

What is Exception handling?

→ Exception handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException etc.

Advantage

The core advantage of Exception handling is to maintain the normal flow of application program.

Mukesh Singh

5.1. Dealing with Errors

There are mainly two types of exceptions: Checked and unchecked. Here, error is considered as unchecked exception.

According to Oracle, there are 3 types of exceptions:

1. Checked 2. Unchecked 3. Error

1. Checked Exception

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

OR, The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions. Eg: IOException, SQLException etc.

2. Unchecked Exception

The classes which inherit runtime exception are known as unchecked exception.

It is an exception that occurs at the time of execution. These are also called Runtime exceptions. Runtime exceptions (unchecked) are ignored at the time of compilation.

Examples → ArithmeticException, NullPointerException, etc.

Mukesh Singh SMC

3. Error
- Error is irrecoverable.
- There are not exception at all, but problems that are beyond the control of the user or programmer.
- An error indicates serious problems that a reasonable application should not try to catch.
- Errors are conditions which cannot be got recovered by any handling techniques.
- Error belongs to unchecked type and mostly occur at runtime.
- They are also ignored at the time of compilation.
Examples - Out of Memory Error, Virtual Machine Error, Assertion Error, System Crash error etc

Errors	Exceptions
1. It indicates serious problem that a reasonable application should not try to catch.	1. It indicates conditions that a reasonable application might try to catch.
2. Recovering from error is not possible.	2. We can recover from exceptions by either using try-catch block or throwing exceptions back to caller.
3. All errors in Java are unchecked type.	3. Exceptions include both checked as well as unchecked type.
4. Occurs at runtime and not known to the compiler.	4. Occurs at runtime but checked exceptions are known to compiler while unchecked are not.
5. They are defined in java.lang.Error package.	5. They are defined in java.lang.Exception package

Mukesh Singh SMC

6. Examples - Stack overflow Error, Out of Memory Error, Virtual Memory Error etc.	6. Examples - SQL Exception, IO Exception, Arithmetic Exception, Null Pointer Exception.

Mukesh Singh SMC

Hierarchy of Java Exception Classes

Mukesh Singh

5.2. Catching Exceptions:
5.3 try, catch, throw, throws and finally

Try-Catch block

We can use try-catch block for exception handling in our code. A method catches an exception using a combination of the 'try' and 'catch' keywords. The try and catch keyword comes in pairs. The try is the start of the block and catch is at the end of try block to handle exceptions.

Try block

- The try block contains set of statements where an exception can occur.
- The try statement allows you to define a block of code to be tested for errors while it is being executed.
- The try block must be followed by either catch or finally or both.
- It means, we can't use try block alone.

Syntax:

```
try {
  // block of code to try
}
```

Catch block

- The catch block is where you handle the exceptions.
- The catch ~~keyword~~ statement allows you to define a block of code to be executed, if an error occurs in the try block.
- The catch block must be preceded by try block (catch must follow try block) which means we can't use catch block alone.
- It can be followed by final block later.

Syntax of try-catch block:

```
try
{
 // Statements that may cause exception
}
Catch (Exception e)
{
 // error handling code
}
```

Program

①
```
class trycatch
 {
public static void main( String args[])
 {
try
 {
int [] myNumbers ={1,2,3};
```

Mukesh Singh SMC

```
System.out.println(myNumbers[10]);
    }
    Catch (Exception e)
    {
    System.out.println("Something went wrong");
    }
  }
}
```

Output

Something went wrong

② Lets see an example to ~~print~~ print custom ~~the exception~~ ~~in a catch block~~ message on exception.

```
Class tryCatch2
  {
  public static void main(String Args[])
  {
  try
    {
    int data= 50/0;  //may throw exception
    }
  Catch(Exception e) // handle the exception
  {
  System.out.println("Can't divide by zero"); //display
                                                the custom msg
  }
}
}
```

Mukesh Singh SMC

Output

Can't divide by zero

③ Let's see an example to resolve the exception in a catch block.

```
public class TryCatch3
{
public static void main (String args[])
{
int i=50;
int j= 0;
int data;
try
{
data= i/j;   //may throw exception
}
catch (Exception e) //handling the exception
{
System.out.println (i/(j+2));
}
}
}
```

Output

25

Mukesh Singh SMC

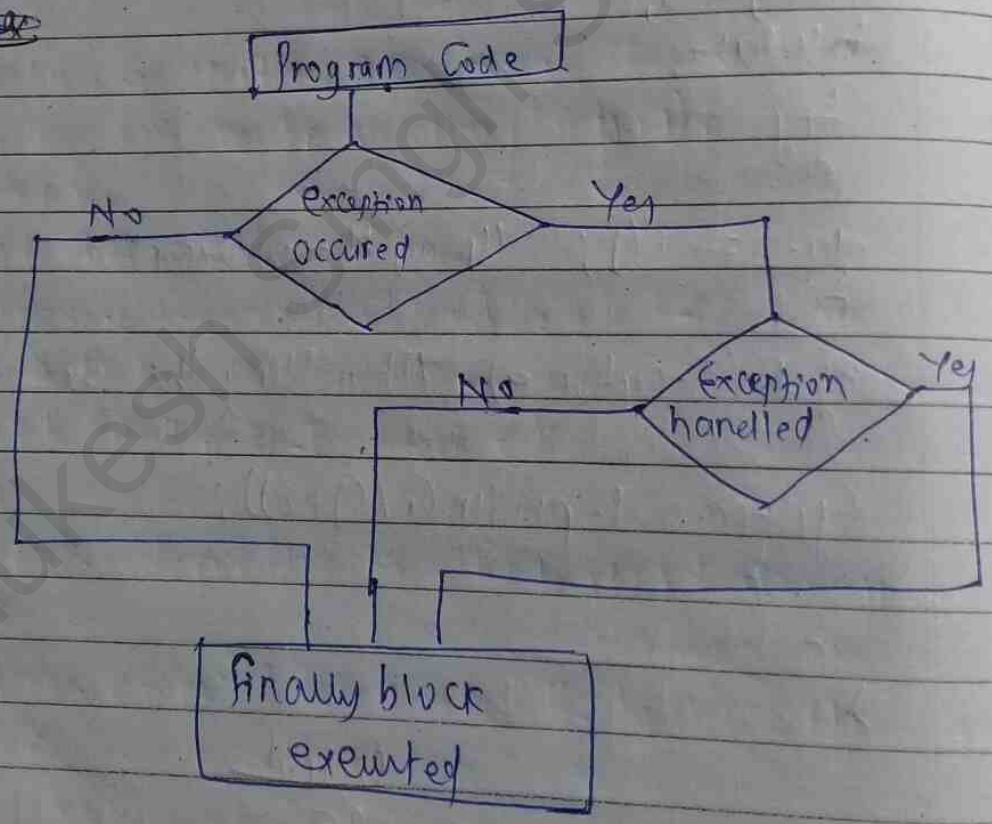
Finally block:

The finally block is used to execute the important code of the program. It is executed whether an exception is handled or not.

The finally block follows a try-catch block. The "finally" statement lets you execute code after try-catch, regardless of occurence of an exception.

Used for closing connection, stream etc.

~~Syntax~~



Mukesh Singh SMC

Syntax

```
try
{
 // statements that may cause an exception
}
Catch
 {
 // handling exception
 }
finally
{
// statements to be executed
 }
```

Program

```
Class Finallyblock
 {
public static void main(String args[])
 {
try
 {
int[] myNumbers = {1,2,3};
System.out.println(myNumbers[10]);
 }
Catch(Exception e)
 {
System.out.println("Something went wrong.");
 }
```

Mukesh Singh SMC

```
finally
 {
System.out.println ("The try-catch is finished.");
 }
 }
}
```

Output

Something went wrong.
The try catch is finished.

(2)

```
class finallyblock2 {
 public static void main (String args[]){
  try {
  int data = 25/0;
  System.out.println(data);
  }
  catch (ArithmeticException e)
   {
  System.out.println("finally block is always executed");
   }
  System.out.println("rest of the code...");
  }
}
```

Output

Exception in thread main java.
finally block is always executed.
rest of the code...

Mukesh Singh SMC

Few important point regarding finally block:
- A finally block must be associated with a try block, you cannot use finally without a try block.
- finally block is optional.
- Normally, when there is no exception in try block then the finally block is executed after try block.
- An exception in finally block behaves exactly like any other exceptions.

Throw Statement:

The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either checked or unchecked exception.

The throw keyword is mainly used to throw custom exceptions. Only objects of Throwable class or its subclass can be thrown.

Mukesh Singh SMC

Program of throw:

Q. Throw an exception if age is below 18 (print "Access denied"). If age is 18 or older, print "Access granted".

```
class throwexception
  {
  static void checkAge (int age)
   {
   if (age <18)
    {
    throw new ArithmeticException("Access denied - You must be at least 18 years old.");
    }
    else {
    System.out.println("Access granted - You are old enough!");
    }
  }
  public static void main (String args[])
   {
   CheckAge(15); //set age to 15 (which is below 18...)
   }
}
```

Mukesh Singh SMC

Example 2

```
class TestThrow
{
static void validate (int age)
{
if (age<18)
throw new ArithmeticException ("Not valid");
else
System.out.println ("Welcome to vote");
}
public static void main (String arg[]){
{
validate (13);
System.out.println ("rest of the code...");
}
}
```

Output

Mukesh Singh SMC

Throws Statement:

- The "throws" keyword is used to declare exceptions. It doesn't throw an exception.
- It specifies that there may occur an exception in the method. It is always used with method signature.
- The caller must catch the exceptions.

Syntax

```
type method_name (parameter list)
throws exception_class_name
  {
  // method code
  }
```

Example

```
Class B_Ed
  {
Static void Sutuna() throws Arithmetic Exception
  {
System.out.println("This is Sutuna");
throw new Arithmetic Exception("JCT");
  }
Public Static void main (String args[]);
  {
try
  {
```

Mukesh Singh SMC

```
fukuna());
  }
catch(ArithmeticException e)
  {
System.out.println("Caught"+e);
    }
  }
}
```

Output

This is fukuna.

Caught java.lang.ArithmeticException ICT

Mukesh Singh SMC

throw	throws
1. 'throw' keyword is used to explicitly throw an exception.	1. 'throws' keyword is used to declare an exception.
2. Only single exception is thrown by using throw.	2. Multiple exceptions can be ~~thrown~~ declared by using throws.
3. throw keyword is used within the method.	3. throws keyword is used with the method signature.
4. throw is followed by an instance (object).	4. throws is followed by class.
5. Checked exception cannot be propogated using throw only.	5. Checked exception can be propagated with throws.
5. Example: a) throw new IOException("can't open connection"); b) void m() { throw new ArithmeticException(e); }	5. Example: a) throws IOException, ArrayOfIndexBoundException; b) void m() throws ArithmeticException { // method code }

Mukesh Singh SWC

Unit-6
Input/Output

6.1 Input/Output Basics

Java input and output is an essential concept while working on Java programming. It consists of elements such as input, output and stream.

The input is data that we give to the program.

The output is data what we receive from the program in the form of result.

~~The~~ Stream represents the flow of data or the sequence of data.

Java uses the concept of stream to make I/O operation fast. To give input we use input stream and to give output we use output stream.

Java.io package contains all the classes required for input and output operations.

Stream

Java performs I/O through streams. 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 general, streams means continuous flow of data.

In java, 3 streams are created for us automatically. All these streams are attached with the console.

i) System.out : standard output stream

ii) System.in : standard input stream

iii) System.err : standard error stream

Mohesh Singh SMC

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

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

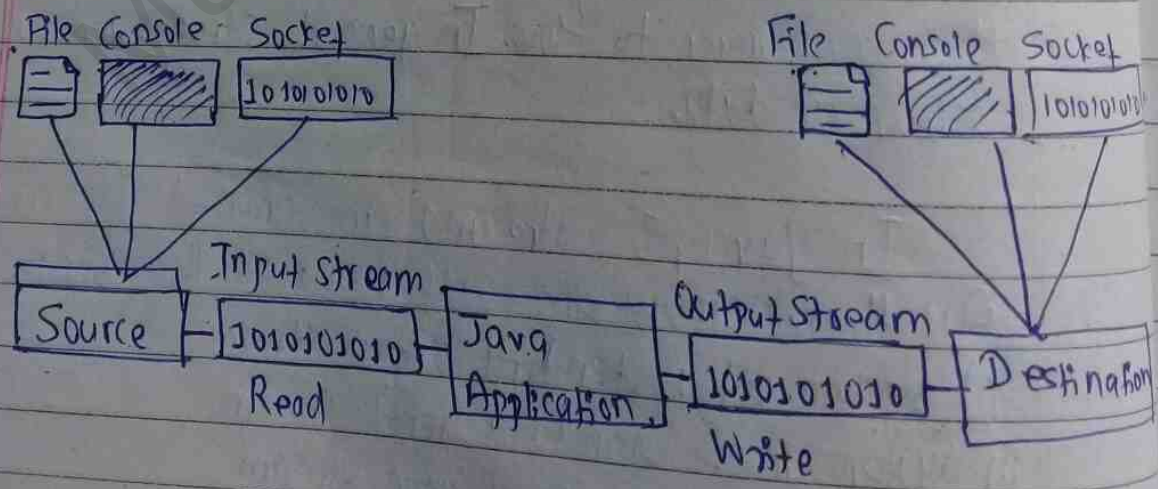
Let's see the code to get input from console.

```
int i=System.in.read();
System.out.println((char)i);
```

Output Stream vs Input Stream

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

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



Output_Stream Class

OutputStream class is an abstract class. It is the super class of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Useful methods of OutputStream:

	Method	Description
1.	public void write(int) throws IOException	is used to write a byte to the current output stream.
2	public void write(byte[]) throws IOException	is used to write an array of byte to the current output stream.
3.	public void flush() throws IOException	flushes the current output stream.
4	public void close() throws IOException	is used to close the current output stream.

Mukesh Singh SMC

InputStream Class

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

Useful methods of InputStream:

Method	Description
1. public abstract int read() throws IOException	reads the next byte of data from the input stream. It returns -1 at the end of the file.
2. public int available() throws IOException	returns an estimate of the number of bytes that can be read from the current input stream.
3. public void close() throws IOException	is used to close the current input stream.

Mukesh Singh SMS

6.3 Reading and writing Files

① Java FileOutputStream

⇒ It is an output stream used for writing data to a file.

Java FileOutputStream Example 1: Write byte

```
import java.io.FileOutputStream;
public class FileOutputStreamExample
{
public static void main (String args[])
{
try
{
FileOutputStream fout = new FileOutputStream ("D:\\testout.txt");
fout.write(65);
fout.close();
System.out.println ("Success...");
}
Catch (Exception e)
{
System.out.println(e);
}
}
}
```

Output

Success....

The content of a test file testout.txt as set with the data A

testout.txt

A

Mukesh Singh SMC

Java FileOutputStream example 2: write string

```
import java.io.FileOutputStream;
public class FileOutputStreamExample
{
public static void main (String Args[])
{
try
{
FileOutputStream fout=new FileOutputStream
        ("D:\\testout.txt");
String s="Welcome to JavaTpoint.";
byte b[]=s.getBytes();
fout.write(b);
fout.close();
System.out.println("Success...");
}
Catch (Exception e)
{
System.out.println(e);
}
}}
```

Output

[Success...]

The content of a text file testout.txt is set with the data Welcome to JavaTpoint.

testout.txt

Welcome to JavaTpoint.

Mukesh Singh SWC

② Java FileInputStream

Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image, data, audio, video, etc. You can also read character-stream data. But for reading stream of characters, it is recommended to use FileReader class.

Java FileInputStream example 1: read single character

```
import java.io.FileInputStream;
public class DataStream Example {
public static void main (String args[]) {
 try {
 FileInputStream fin = new FileInputStream ("D:\\testout.txt");
 int i = fin.read();
System.out.println((char) i);
 fin.close();
 }
 catch (Exception e)
  {
 System.out.println (e); }
 }
}
```

Note: Before running the code, a textfile named as "testout.txt" is required to be created.

Output

Welcome to J; javaTpoint

Mukesh Singh SMC

After executing the above program, you will get a single character from the file which is 87 (in byte form). To see the text, you need to convert it into character:

Output
W

Java FileInputStream Example 2 : read all characters

```
package com.javatpoint;
import java.io.FileInputStream;
public class DataStreamExample
{
public static void main (String args[])
{
try
{
FileInputStream fin=new FileInputStream("D:\\testout.txt");
int i=0;
While ((i=fin.read())!=-1)
{
System.out.println((char)i);
}
fin.close();
}
Catch(Exception e)
{
System.out.println(e);}
```

Mukesh Singh SMC

}
}

Output

Welcome to JavaTpoint

Mukesh Singh SMC

Unit-7

Event Handling and User Interface Components with Swing

7.1 Basics of Event Handling

Events

Change in the state of an object is known as event i.e event describes the change in state of source. Events are generated as a result of user interaction with the graphical user interface components. For example: Clicking on a button, moving the mouse, entering a character through keyboard, scrolling the page etc are the activities that causes event to happen.

Event handling

- Event handling is in Java refers to executing some code when specific things occur such as a window being minimized or a button being clicked.
- Event handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has the code which is known as event handler that is executed when an event occurs.

In Java, event handling may comprised the following four classes:

i) Event Sources: Event source is an object that generates an event.

Source for generating an event may be the components. In Java java.awt.Component specifies the components that may or may not generate events. These components classes are the subclasses of the above class. These event

Source may be the button, combobox, textbox, etc.

ii) Event Classes:
Event classes in Java are the classes defined for almost all the components that may generate events.
These ~~component~~ events classes are named by giving the specific name such as for the component source button event class is Action Event.
Following are the list of Event Classes:

- Action Event : Button, Text field, List, Menu
- Window Event: Frame
- Item Event : Checkbox, List
- Adjustment Event : Scrollbar
- Mouse Event : Mouse
- Key Event : Keyboard

iii) Event Listeners: A listener is an object that listens to the event. A listener gets notified when an event occurs.
In Java almost all components has its own listener that handles the event generated by the component. For example, there is a listener named ActionListener handles the events generated from button, textfield list, menus.

Vikesh Singh SMC

iv) Event adapters:

Event adapters classes are abstract class that provide some methods used for avoiding the heavy coding. Adapter class is defined for the listener that has more than one abstract methods.

Mukesh Singh SMC