COURSE CONTENT

SSSIT COMPUTER EDUCATION

SECTION 1: INTRODUCTION

- Introduction about Programming Language Paradigms
- Why Java?
- Flavors of Java.
- Versions of Java
- Java's Magic Bytecode.
- Features of Java Language.

What is Java?

- Java is an object-oriented programming language developed by Sun Microsystems, and it was released in 1995.
- James Gosling initially developed Java in Sun Microsystems
- Now, it is owned by Oracle
- It is used for:
    - Mobile applications (specially Android apps)
    - Desktop applications
    - Web applications
    - Web servers and application servers
    - Games
    - Database connection
    - And many more……

Why Java?

- Java has automatic memory management system.
- No dangling pointers, and no memory leaks.
- Java simplifies pointer handling, no reference and no deference operators.
- Java follows dynamic loading whereas C and C++ follows static loading.

What is the difference between static and dynamic loading

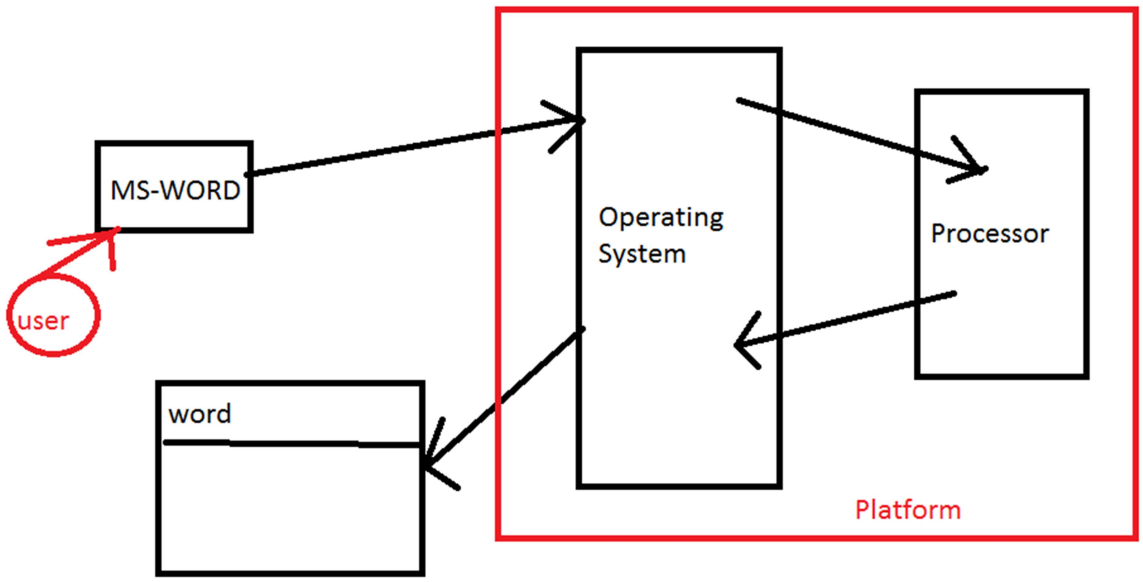
- static:
    - This is the process of allocating memory for variable or program during compilation
    - In this static memory there is the feasibility for wastage of memory, which increases the length of the program and reduce the efficiency of the program
- dynamic:

- Dynamic memory allocation is a process of allocating the memory for the program during execution.
- There is no feasibility for wastage of memory and efficiency of the program will be increased.

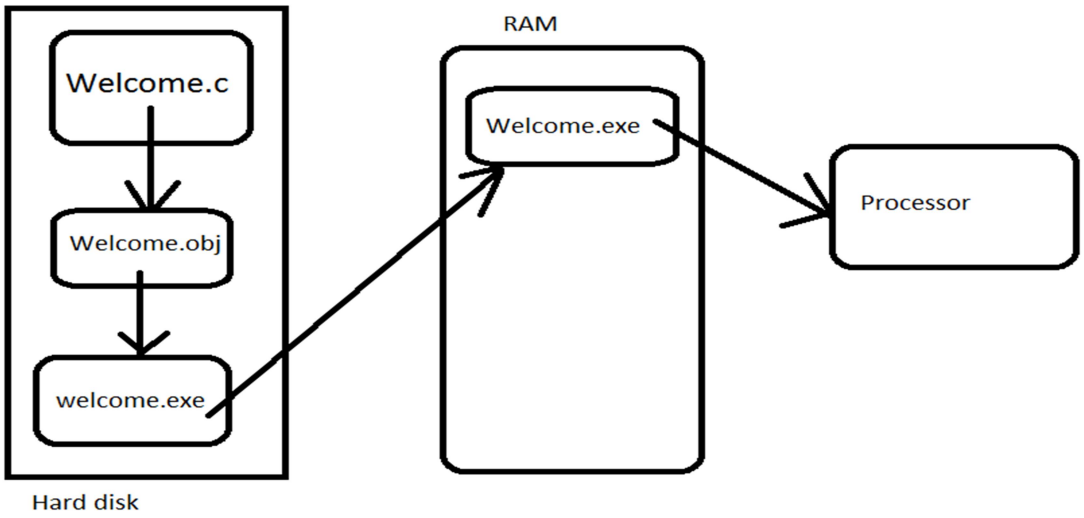
What is platform Independent?

What is Platform?

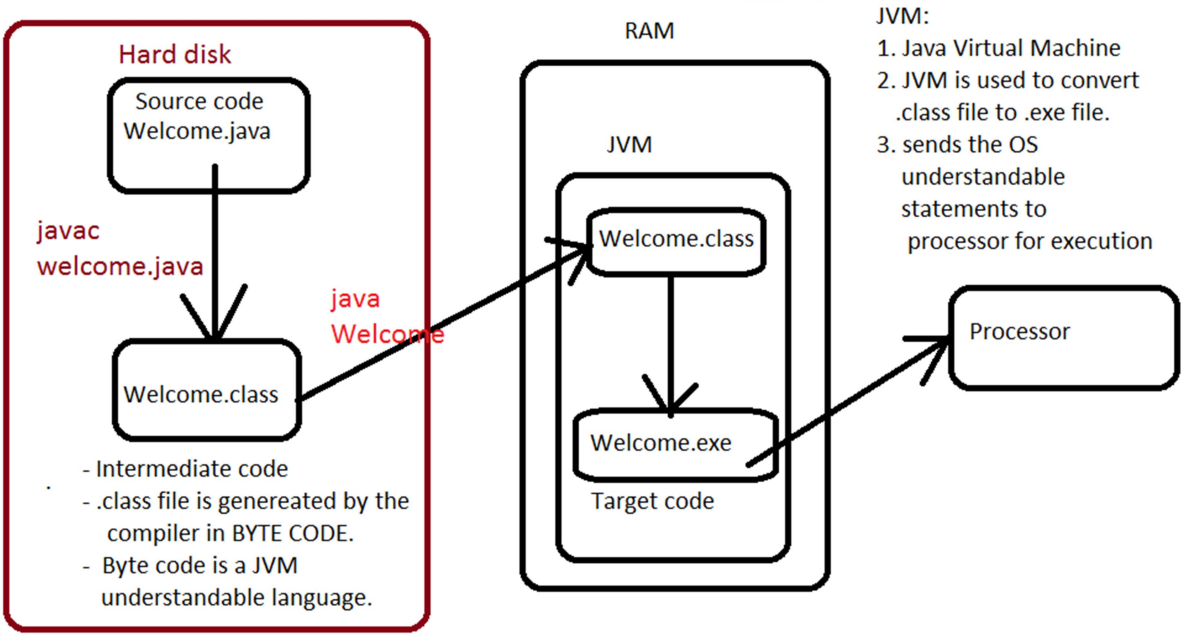
- A platform is the hardware or software environment in which the program runs.
- This platform in real-life is the laptop and the underlying operating system runs on that laptop.

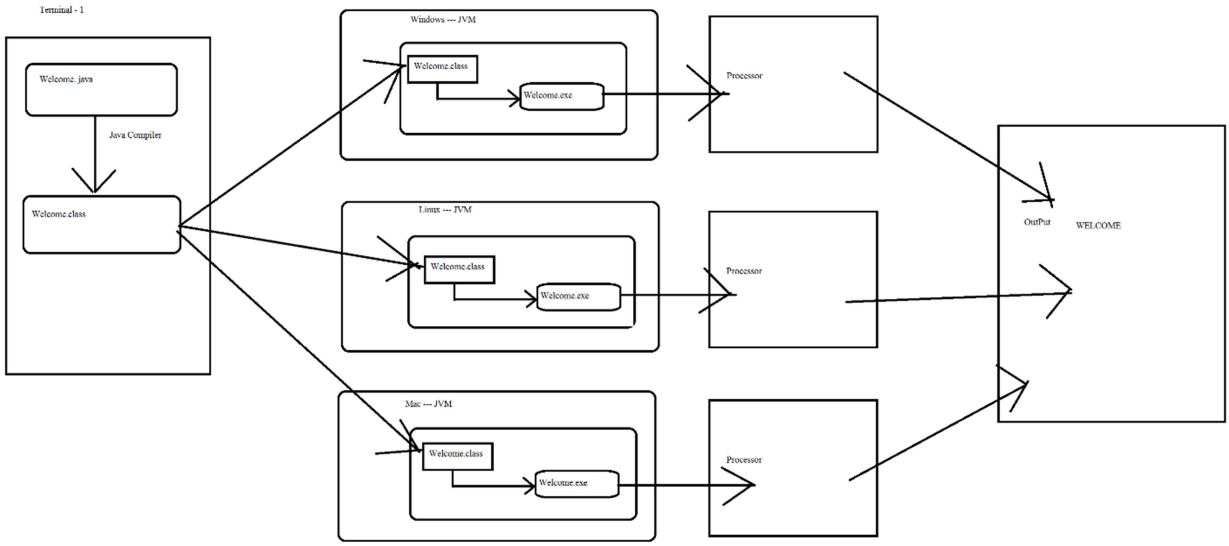


Why C/C++ Programs are platform dependent?



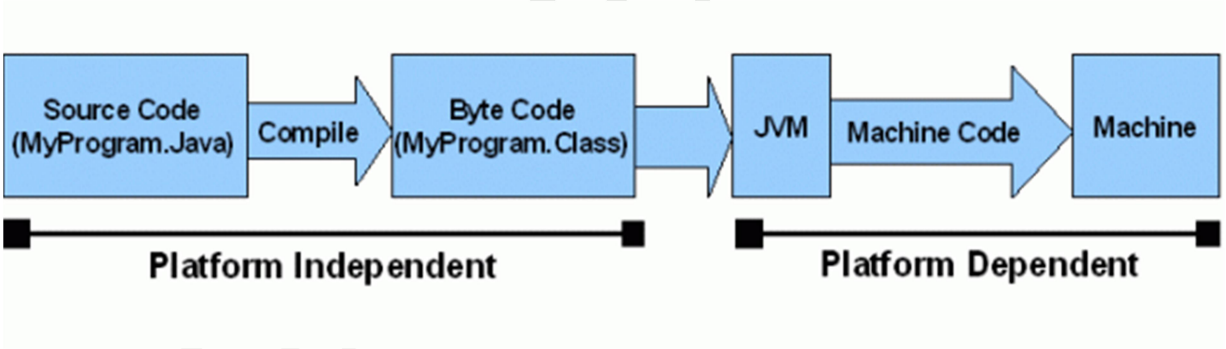
How a Java Program Executes?





Java achieved Platform independent with the help of

- JVM
- Bytecode



What makes Java so popular?

Java has a number of advantages which make it a wide spreading technology:

- Platform independent:
  A Java program can run on any platform including Windows, Linux, Mac, Solaris, ... with a Java Virtual Machine installed on the target operating system. Developers write code once and the program can run anywhere, thus the very well know slogan "Write once, run anywhere".

- Powerful language:
  The Java programming language is strong typed, has clear syntax and addresses some limitations of C/C++... that makes it a easy to learn

language for programmers. There are thousands of libraries written in Java for almost any field.

- Secure:
  Java has a built-in strong security manager from ground-up that prevents malicious code and viruses.

- High performance:
  With the JVM has been improved over the years, Java applications are fast.

- Networking:
  With built-in support for networking, developers can write internet-based and networked applications easily.

Flavors of Java

According to the Sun Microsystem java is available in 3 flavors

- Standard edition:
  - Which is used for developing Standalone Applications.
  - It is also known as J2SE, JSE.
- Enterprise edition:
  - Which is used for developing web applications.
  - It is also known as J2EE, JEE.
- Micro edition:
  - Which is used for developing device programs and mobile applications
  - It is also known as J2ME, JME.

SSIT COMPUTER EDUCATION

Java SE Version	Version Number	Release Date
JDK 1.0 (Oak)	1.0	January 1996
JDK 1.1	1.1	February 1997
J2SE 1.2 (Playground)	1.2	December 1998
J2SE 1.3 (Kestrel)	1.3	May 2000
J2SE 1.4 (Merlin)	1.4	February 2002
J2SE 5.0 (Tiger)	1.5	September 2004
Java SE 6 (Mustang)	1.6	December 2006
Java SE 7 (Dolphin)	1.7	July 2011
Java SE 8	1.8	March 2014
Java SE 9	9	September, 21st 2017
Java SE 10	10	March, 20th 2018
Java SE 11	11	September, 25th 2018
Java SE 12	12	March, 19th 2019
Java SE 13	13	September, 17th 2019
Java SE 14	14	March, 17th 2020
Java SE 15	15	September, 15th 2020

Java SE 16	16	March, 16th 2021
Java SE 17	17	September, 14th 2021
Java SE 18	18	March , 22nd 2022

Features of Java or Java Buzz words

1. Simple
2. Object oriented
3. Distributed
4. Interpreted
5. Robust
6. Secure
7. Architecture neutral
8. Portable
9. High performance
10. Multithreaded
11. Dynamic

Simple

- Java was designed to be easy for professional programmer to learn and use effectively.
- It's simple and easy to learn if you already know the basic concepts of Object Oriented Programming.
- C++ programmer can move to JAVA with very little effort to learn.

- Most of the complex or confusing features in C++ are removed in Java like pointers etc.

Object Oriented

- Java is true object oriented language.
- It supports all the features of object oriented model like:
    - Encapsulation
    - Inheritance
    - Polymorphism
    - Abstraction
- Almost "Everything is an Object" paradigm. All program code and data reside within objects and classes.
- The object model in Java is simple and easy to extend.
- Java comes with an extensive set of classes, arranged in packages that can be used in our programs through inheritance.

Distributed

- Java is designed for distributed environment of the Internet. Its used for creating applications on networks.
- Java applications can access remote objects on Internet as easily as they can do in local system.
- Java enables multiple programmers at multiple remote locations to collaborate and work together on a single project.
-

Compiled and Interpreted

- Usually a computer language is either compiled or Interpreted. Java combines both this approach and makes it a two-stage system.
- Compiled : Java enables creation of a cross platform programs by compiling into an intermediate representation called Java Bytecode.
- Interpreted : Bytecode is then interpreted, which generates machine code that can be directly executed by the machine that provides a Java Virtual machine.

Robust

- It provides many features that make the program execute reliably in variety of environments.
- Java is a strictly typed language. It checks code both at compile time and runtime.
- Java takes care of all memory management problems with garbage-collection.
- Java, with the help of exception handling captures all types of serious errors and eliminates any risk of crashing the system.

Secure

- Java provides data security through encapsulation.
- Java provides a “firewall” between a networked application and your computer.
- When a Java Compatible Web browser is used, downloading can be done safely without fear of viral infection or malicious intent.
- Java achieves this protection by confining a Java program to the java execution environment and not allowing it to access other parts of the computer.

Architecture Neutral

- Bytecode helps Java to achieve portability.
- •Bytecode can be executed on computers having any kind of operating system or any kind of CPU.
- Since Java applications can run on any kind of CPU ie Changes and upgrades in operating systems, processors and system resources will not force any changes in Java Programs., Java is architecture – neutral
- Java language and Java Virtual Machine helped in achieving the goal of "write once; run anywhere, any time, forever."

Portable

- Applications written using Java are portable in the sense that they can be executed on any kind of computer containing any CPU or any operating system.
- When an application written in Java is compiled, it generates an intermediate code file called as "bytecode".
- Bytecode helps Java to achieve portability.
- This bytecode can be taken to any computer and executed directly.
- Java Provides a way to download programs dynamically to all the various types of platforms connected to the Internet.
- It helps in generating Portable executable code.

High Performance

- Java performance is high because of the use of bytecode.
- The bytecode was used, so that it was easily translated into native machine code.

Multithreaded

- Multithreaded Programs handled multiple tasks simultaneously, which was helpful in creating interactive, networked programs.
- Java run-time system comes with tools that support multiprocess synchronization used to construct smoothly interactive systems.
- Real world example for multithreading is computer. While we are listening to music, at the same time we can write in a word document or play a game.

Dynamic

- Java is capable of linking in new class libraries, methods, and objects.
- It can also link native methods (the functions written in other languages such as C and C++).
- The Java Virtual Machine(JVM) maintains a lot of runtime information about the program and the objects in the program.
- Libraries are dynamically linked during runtime.
- So, even if you make dynamic changes to pieces of code, the program is not effected.

SSSIT

SECTION 2: INSTALLATION JAVA

- Installing Java
- Difference between JDK,JRE and JVM
- JVM –The heart of Java .
- Java Architecture

How to install Java?

1. Download the java software from oracle web site and Install

2. Set class path in order to execute the java programs from command prompt.

    1. right click on MyPC or Mycomputer
    2. Select system properties or properties (Last option in the menu)
    3. select Advanced System properties (available in the left side pane of the window)
    4. Click on Environment Variables button (Extreme bottom)
    5. Click on New button available below the user variables area.
        - Variable Name : classpath
        - Variable Value : C:\Program Files\Java\jdk1.7.0_79\lib;.;
    6. Click on OK.
    7. Click on New button available below the user variables area.
        - Variable Name : path
        - Variable Value : C:\Program Files\Java\jdk1.7.0_79\bin;
    8. Click on OK.
    9. Click on OK

Note: Dot(.) in the class path refers current working directory

How to test whether Java is successfully installed or not?

Open Command Prompt and type the following commands

C:\> Java

We can see some help

C:\> Javac

We can see some help

- If we got some help means java is installed successfully.

- If we got an error like
  - Unrecognized command
  - means Java is not installed successfully.
- Then uninstall and install once again.
- What is JVM?
- Java Virtual Machine interprets compiled Java binary code for a processor so it can perform a java program's instructions. Specification specifies an instruction set of register, stack, and garbage heap and method area. JVM can interpret the byte code one instruction at a time or the byte code can be compiled further for a real processor using a Just-In-Time Compiler.

- Just-In-Time: Java programming language environment, Just-In-Time compiler is a program that converts java byte code into instructions that can be sent directly to the processor.

What is the difference between JDK, JRE and JVM?

JVM	JRE	JDK
• It is an abstract machine. It gives runtime environment in which byte code can execute. • JVM is platform dependent because configuration and file format of each operating system is different. • JVM performs Load, Verify, and Execute code and provides runtime environment.	• It used to provide Runtime Environment. • It is the implementation of JVM. • It contains set of libraries and core classes.It physically exists.	• It contains JRE and development tools. • Java Development Kit is a bundle of Java Compiler, Java Interpreter, Java Disassembler, Java Header File Generator, Java Documentation, Java Debugger, Java Applet viewer, JRE.

SSIT COMPUTER EDUCATION

SECTION 3: JAVA SAMPLE PROGRAM

- Writing a simple Java program
- Compilation &Execution
- Rectifying common errors
- Lexical Tokens
    - Identifiers
    - Keywords
    - Literals and Comments
- Command Line Arguments
- Real-time Practical's
    - Print Hai! In the first line and print How are You !! in the second line using single print statement.
    - Get the input 10 25 a 1.2 Hi via command line arguments and print the values one by one.

Write a Java program to display a Hello world?

Step 1:

Open any text editor like Notepad/Notepad++/Editpuls/any text editor to type the java program

Step 2:

Define a class followed by class name

Syntax:

```
class <<class-name>>
{
}
```

Note: According to Java naming standards, the class name should be valid identifier and should begin with capital letter.

Example

```
class HelloWorld
{
}
```

SSIT COMPUTER EDUCATION

Step 3:

Define the main() method inside the class.

main() is the entry point of any Java Program

Syntax:

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

Step 4:

Define the output statement inside the main method

Syntax:

System.out.println(<<Output-message>>);

Syntax:

System.out.println("Hello World");

Step 5:

Save the program as <<file-name>>.java

Note:

According to Java naming conventions, it is recommended to match the file name with class name

Example: HelloWorld.java

Step 6:

Compile the program by invoking Java compiler

Step 6.1: open command prompt (Windows → Run→ command)

Step 6.2: move to the directory where .java file is saved.

SSSIT COMPUTER EDUCATION

Step 6.3: Use javac command to compile the file

Example: javac HelloWorld.java

The above steps performs the following operations:

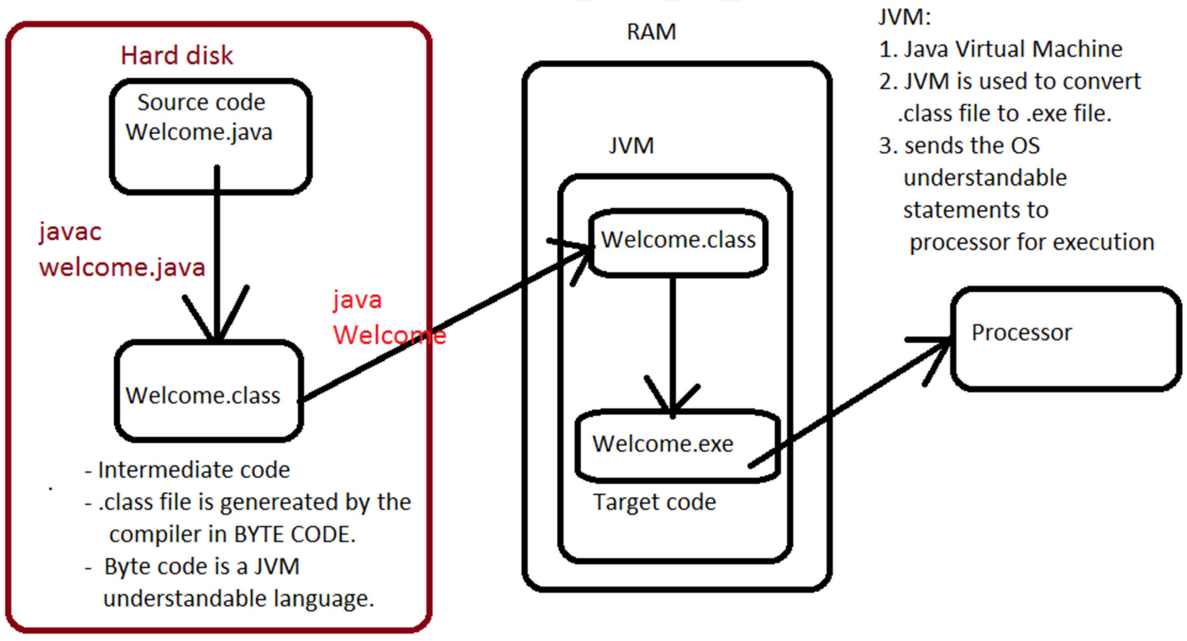
1. Checks the syntactical errors in the program. If any errors are identified, then it notifies to the user.
2. If no syntactical errors are available, then it creates a .class file (ie. HelloWorld.class)

Step 7: Run the Java program using Java command followed by class name

Example: java HelloWorld

It performs the following the steps:

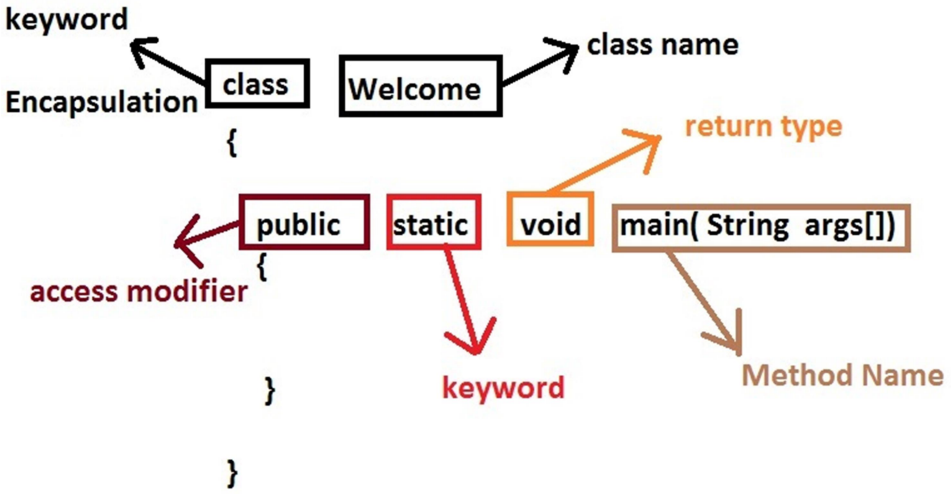
1. Loads the .class file into JVM running inside RAM.
2. JVM translates the .class into Platform or Machine understandable form and gives to processor for execution.
3. Processor executes the program and displays the output on console (Monitor).



Lexical Tokens

- Tokens are the various Java program elements which are identified by the compiler.
- A token is the smallest element of a program that is meaningful to the compiler.
- Tokens supported in Java include

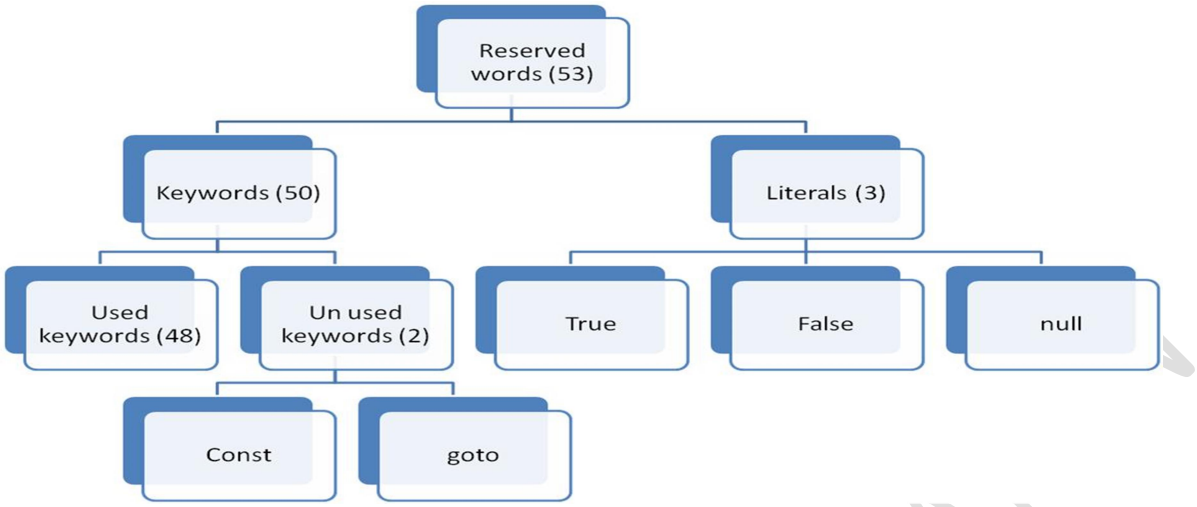
(1) Literals (2) keywords (3) Identifiers (4) constants



Literals in Java can be classified into six types, as below:

1. Integral Literals
2. Floating-point Literals
3. Char Literals
4. String Literals
5. Boolean Literals
6. Null Literals

Keywords



Note:

If we use goto or const keywords in java then it raises COMPILE TIME ERROR

Used Words in Java

Data types	Flow Controls	Modifiers	Exception Handling	Class Related	Object Related		Return type
Byte	If	public	Try	Class	new	enum(1.5)	void
Short	Else	private	Catch	interface	this		
Int	Switch	protected	Finally	package	super		
Long	Case	static	Throw	import	instanceof		
Float	Default	final	throws	extends			
Double	For	Abstract	Assert (1.6)	implements			
Char	While	Native					
Boolean	Do	synchronized					
	Break	Volatile					
	Continue	Transient					
	Return	strictfp					

Identifiers

- A name in java is called as Identifier.
- An identifier can be a
  - Class name
  - Variable name
  - Array name
  - Method name

SSIT COMPUTER EDUCATION

Rules to define an Identifier:

- Identifier must begin with
  - a letter (A-Z, a-z, or any other language letters supported by UFT 16)
  - An underscore (_)
  - A dollar ($)

  after which it can be sequence of letters/digits.
- Digits: 0-9 or any Unicode that represents digit.
- Length of the variable name is unlimited
- Java reserved words should not be used as variable names.
- Must begin with lower case
- Must always begin your variable names with a letter, not "$" or "_".
- Avoid abbreviations, use meaningful names.
- If a variable consists of two or more words, then the second and subsequent words should start with upper case.

SSIT COMPUTER EDUCATION

SECTION 4: PROGRAMMING FUNDAMENTALS

- Data types
- Variable Declaration & Initialization
- Type Casting
- Operators and its types
- Real-time Practicals
    - Swap the values with using temporary variables and without using temporary variable.
    - Find the maximum of two numbers and three numbers using ternary operator.

Group	Data type	Size	Range
Integer Group	Byte	1 byte	-128 to + 127
	Short	2 bytes	-32,768 to + 32,767
	Int	4 bytes	-2,147483648 to +2147,483,647
	Long	8 bytes	
Float group	Float	4 bytes	
	Double	8 bytes	
Character group	Char	2 bytes	
Boolean	Boolean		true or false

Note:

- The default data type in Integer group is int.
- Every long value may have 'l' or 'L' as a suffix.
- The default data type in Float group is double
- Every float value may have 'f' or 'F' as a suffix.
- Every double value may have 'd' or 'D' as a suffix.
- In java, all the local variables must be explicitly initialized by programmer.
- Float takes max of 6 decimal places whereas double will take 16 decimal places.
- We can't assign a float value to an integer variable directly.

What is variable?

- Variable is a Location or an area where data is stored.

How to declare a variable in java?

SSIT COMPUTER EDUCATION

```
<<data-type>>  <<identifier>> = <<default-value>>;
```

In How many ways we can provide inputs to Java Program?

1. Using command line arguments
2. Using scanner class
3. Using BufferedReader in java.io Package

Using command line arguments

```
class Addition
{
	public static void main(String[] args)
	{
		int num1 = Integer.parseInt(args[0]);
		int num2 = Integer.parseInt(args[1]);
		System.out.println("Number of inputs are:" + args.length);
		int result = num1 + num2;
		System.out.println("Result is:" + result);
		System.out.println("Result1 is:" + (args[0]+args[1]));
	}
}
```

How to compile the above program?

C:\>javac Addition.java

How to run the above program?

C:\>java Addition 100 200

- In the above program 100 and 200 are input's supplied to the java program using command line arguments.
- These are stored in the form of a string in an array called args.

SSSIT COMPUTER EDUCATION

- To convert the String type of inputs into respective primitive data types we need to take support of WRAPPER CLASSES.

Primitive Data type	Wrapper class
byte	Byte.parseByte(String)
short	Short.parseShort(String)
int	Integer.parseInt(String)
long	Long.parseLong(String)
float	Float.parseFloat(String)
double	Double.parseDouble(String)
boolean	Boolean.parseBoolean(String)

Scanner Class in Java

- Scanner is a class in java.util package
- used for obtaining the input of primitive types like int, double etc. and strings.
- It is the easiest way to read input in a Java program
- To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream.

How to use Scanner class in the Java Program?

Step 1: import the scanner class into program

```
Import java.util.Scanner;
```

Step 2: create the object of scanner class.

```
Scanner <<ref>> = new Scanner(<<input-ref>>);
```

Ex: Scanner scan = new Scanner(System.in);

Methods defined in Scanner class:

Group	Data type	Method
Integer Group	Byte	nextByte( )
	Short	nextShort( )
	Int	nextInt( )
	Long	nextLong( )
Float group	Float	nextFloat( )
	Double	nextDouble( )
Character group	Char	next( ).charAt(0)
Boolean	Boolean	nextBoolean( )

SSIT COMPUTER EDUCATION

String		next()

Write a Java program that accepts and display Student details.

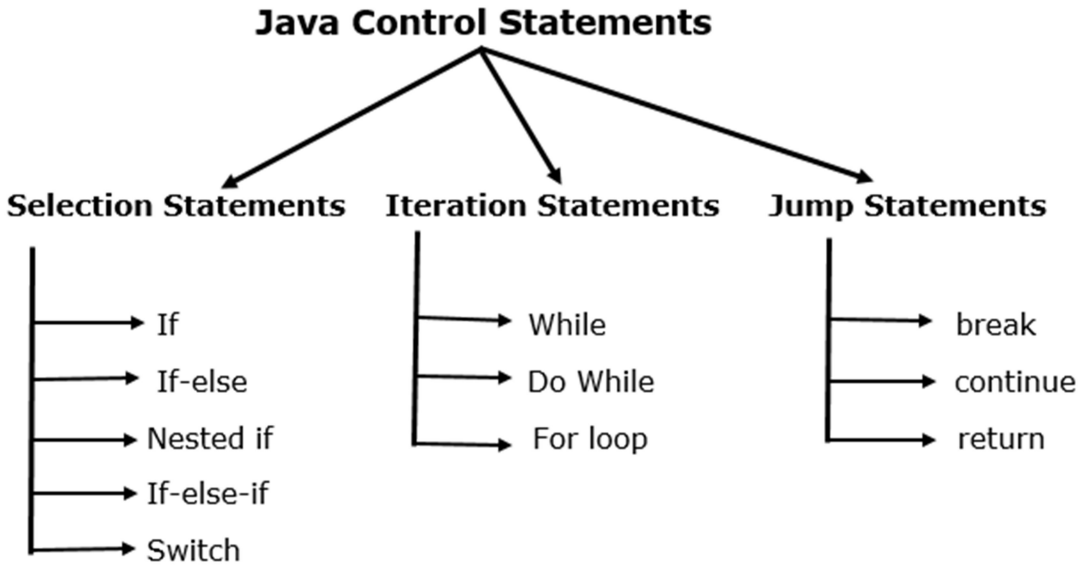
```
import java.util.Scanner;
class StudentDemo
{
	public static void main(String[] args)
	{
		Scanner scan = new Scanner(System.in);

		System.out.println("Enter Htno:");
		int htno = scan.nextInt();

		System.out.println("Enter Name:");
		String name = scan.next();

		System.out.println("Enter CGPA:");
		double cgpa = scan.nextDouble();

		System.out.println("Htno = " + htno);
		System.out.println("Name = " + name);
		System.out.println("cgpa = " + cgpa);

	}
}
```

SSIT COMPUTER EDUCATION

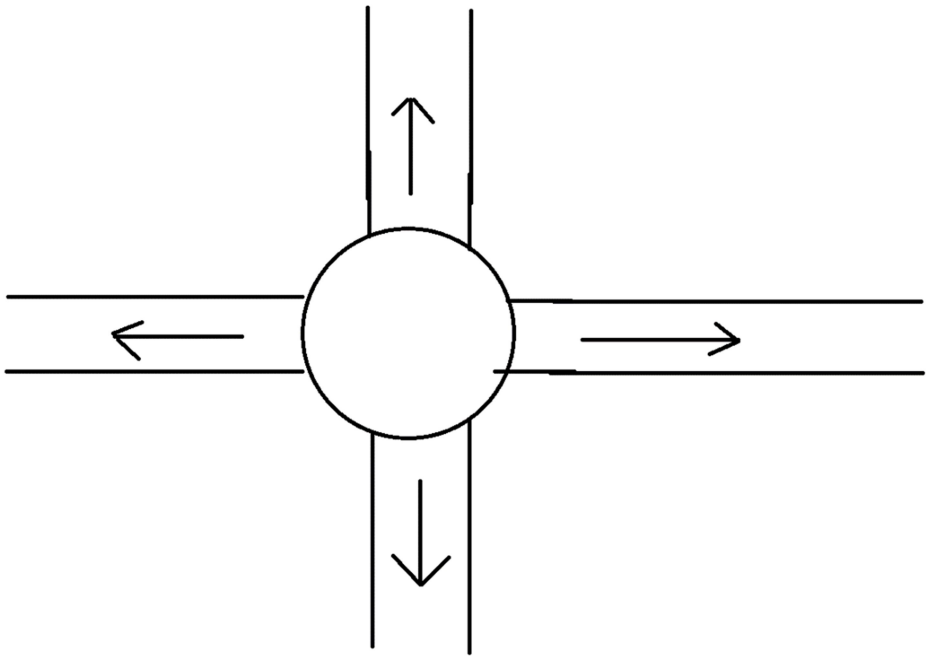
SECTION 5: CONTROL STRUCTURE AND LOOPING STATEMENTS

- IF conditions
- IF-ELSE conditions
- Nested IF conditions
- ELSE-IF Ladder conditions
- SWITCH-CASE statements
- The “break” and “continue” keywords
- “FOR” Loop
- Different forms of FOR Loop
- “WHILE” Loop
- “DO-WHILE” Loop
- ENHANCED “FOR” Loop
- Nested Loops
- Real-time Practicals
  - Find the maximum of 2 numbers and 3 numbers without using ternary operator.
  - Print the factorial of 5!

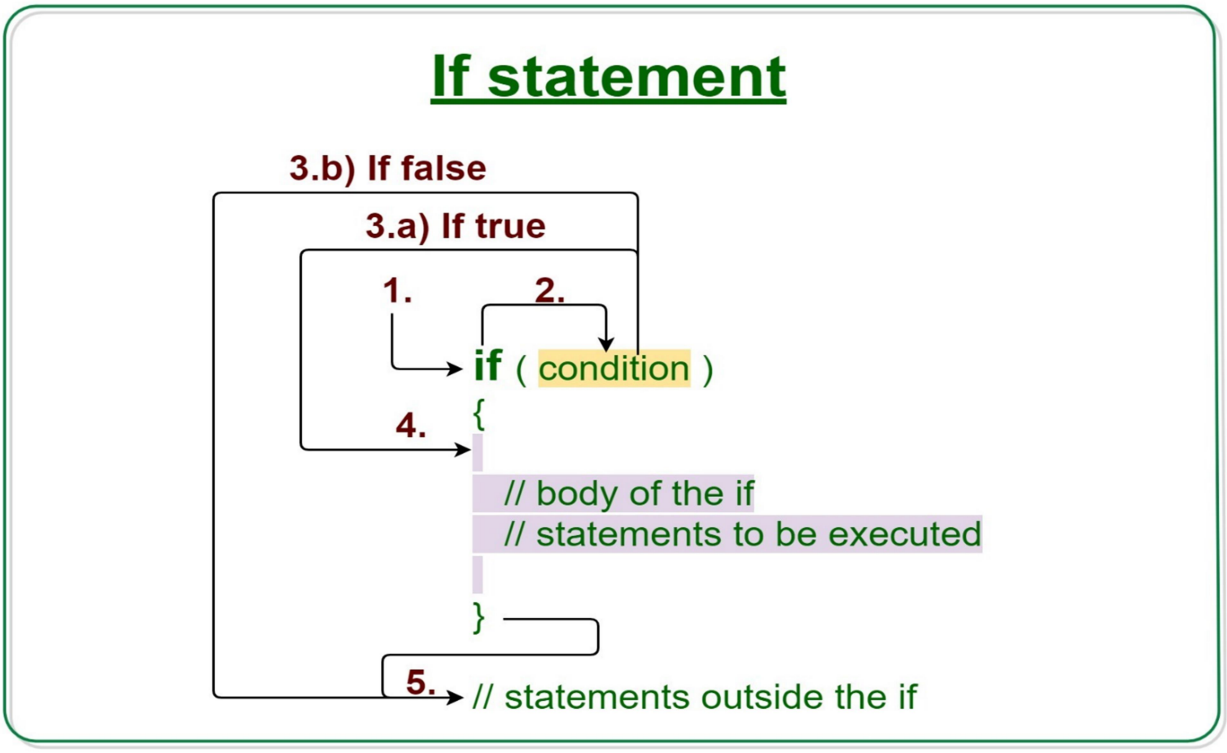
Control Structures



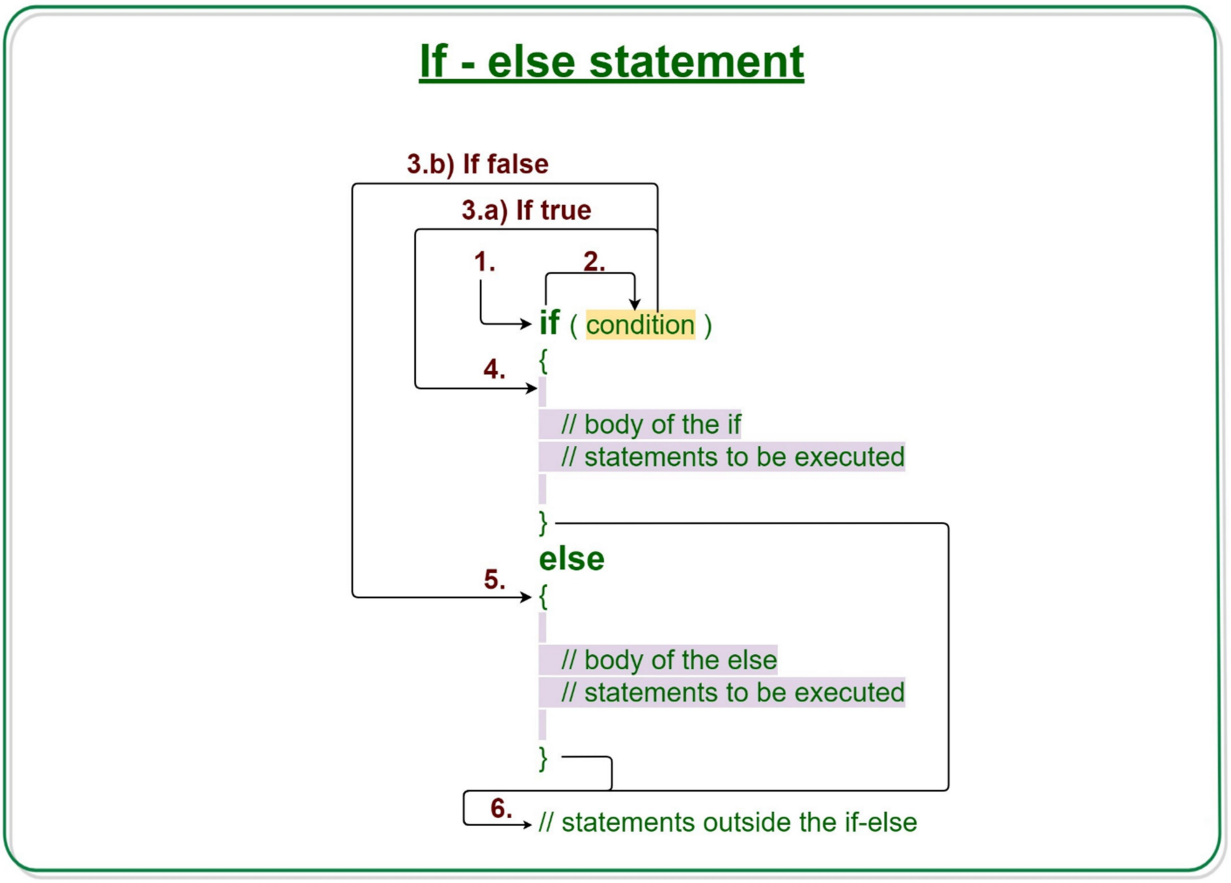
Logic of If statement:



Simple If statement flow

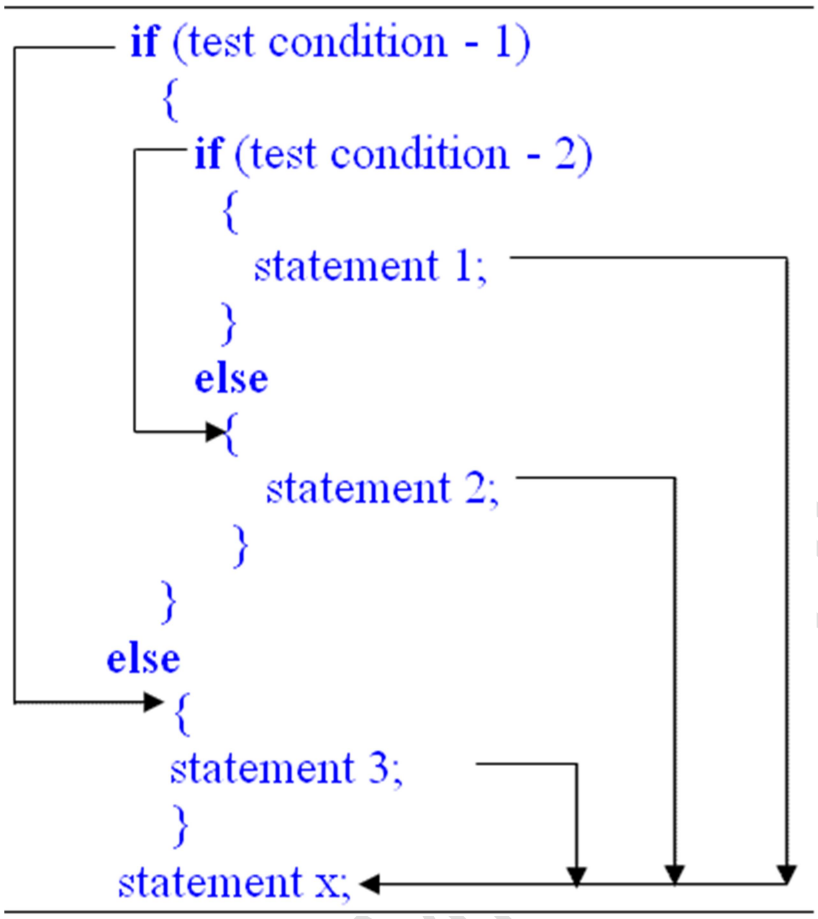


If – Else statement



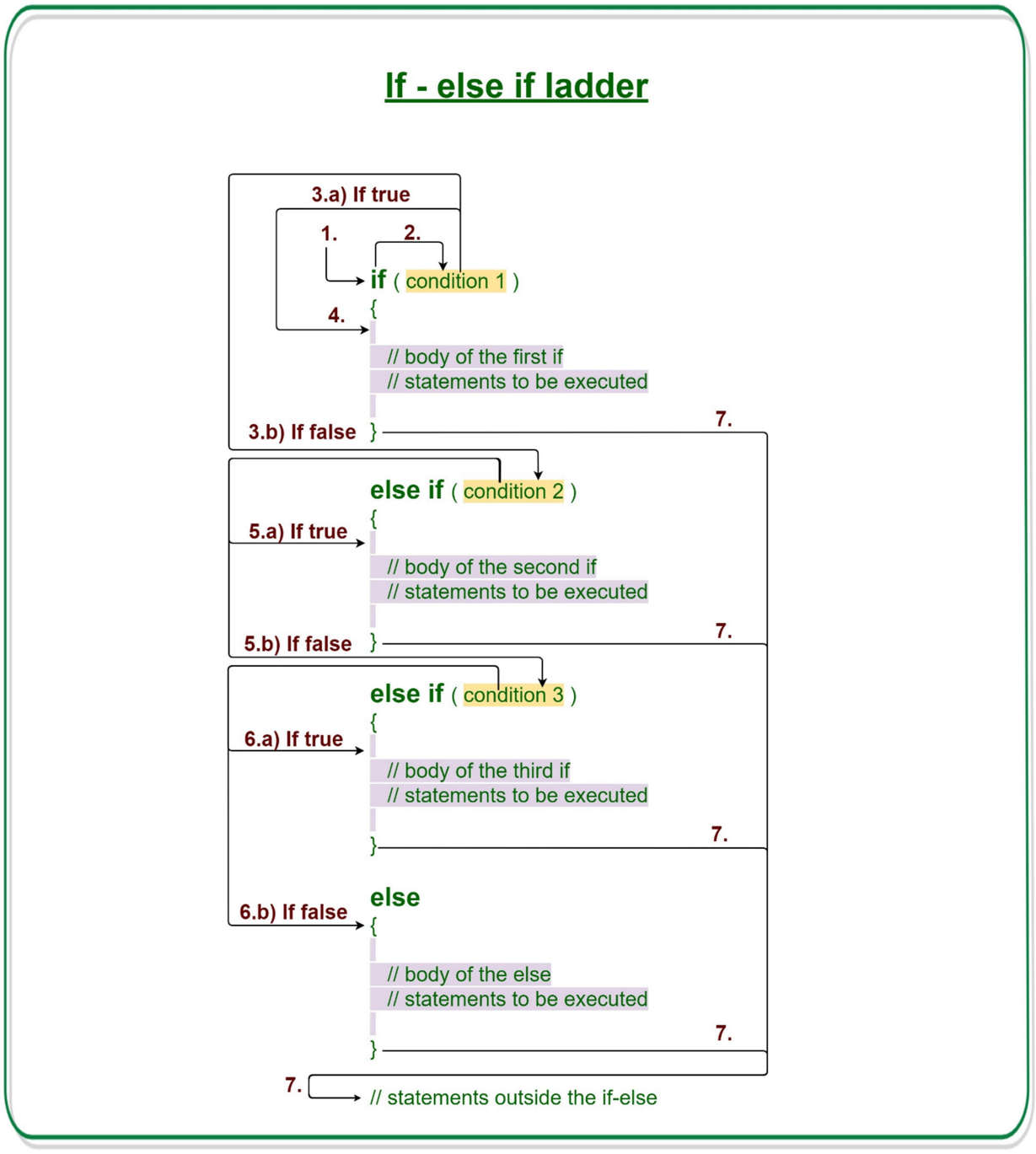
SSIT COM

Nested – If Statement

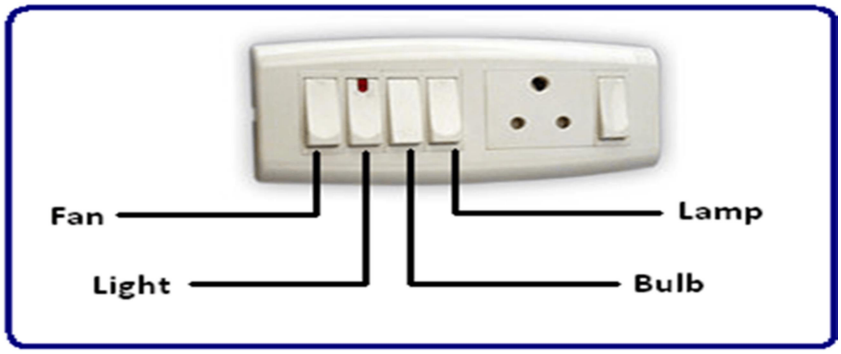


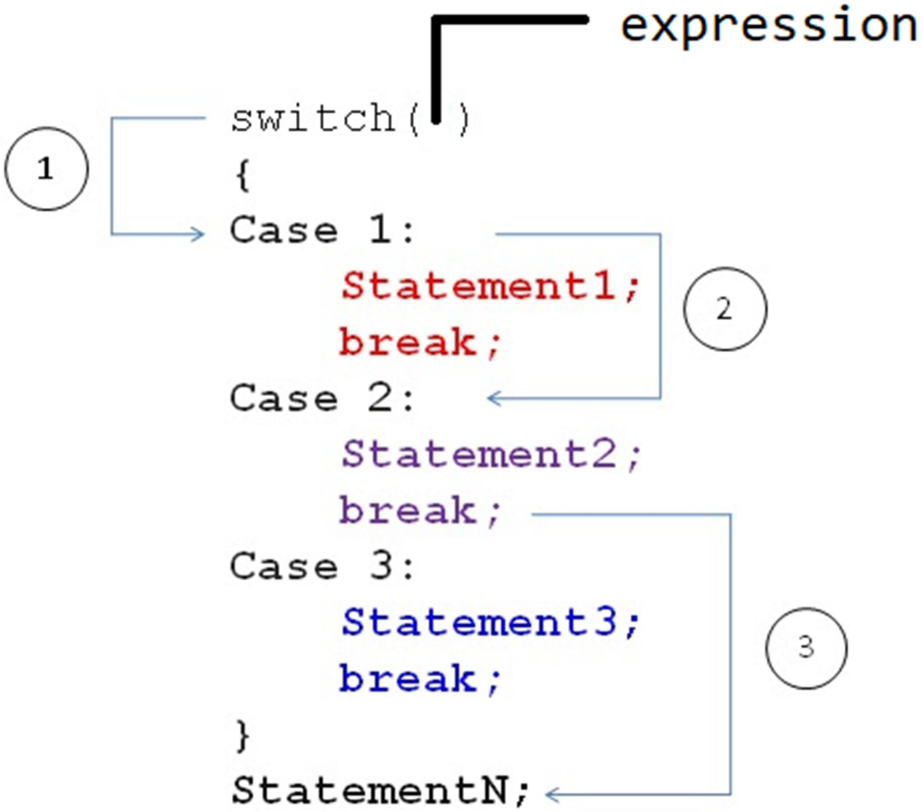
SSSIT COMPUTER EDUCATION

Ladder – If or If – else – if

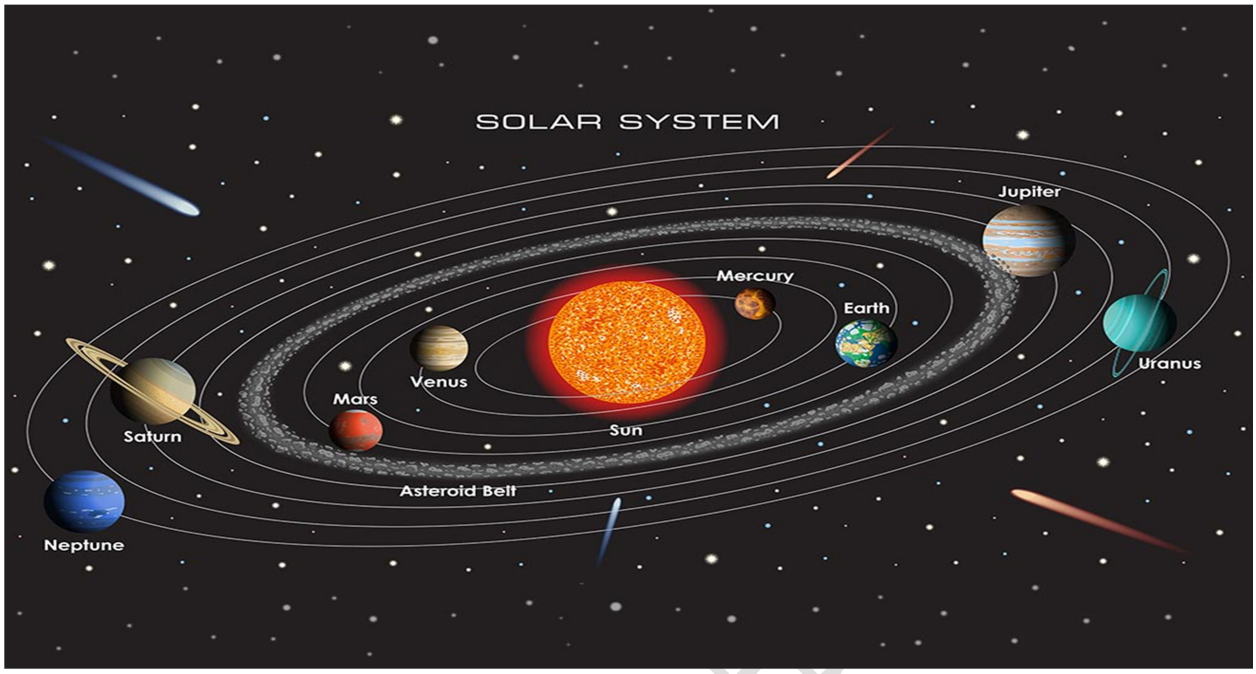


SWITCH….CASE

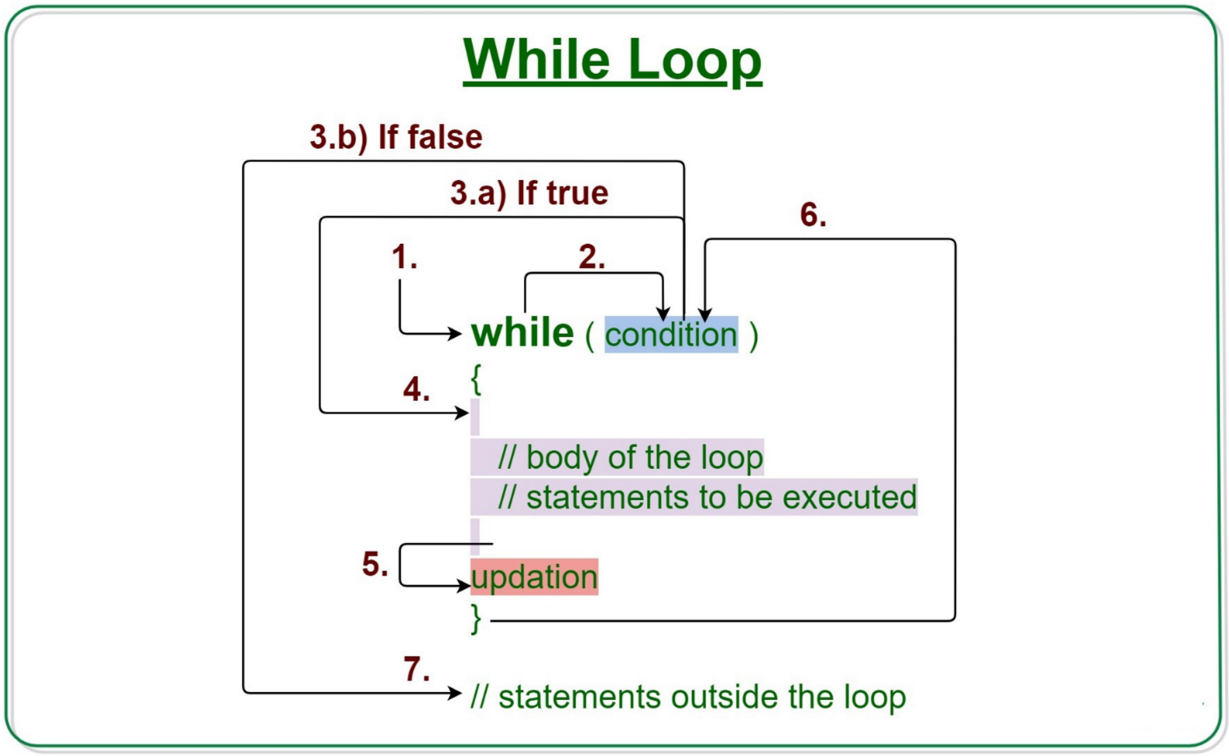

Fan
Light
Lamp
Bulb


expression
switch( )
{
Case 1:
Statement1;
break;
Case 2:
Statement2;
break;
Case 3:
Statement3;
break;
}
StatementN;
1
2
3

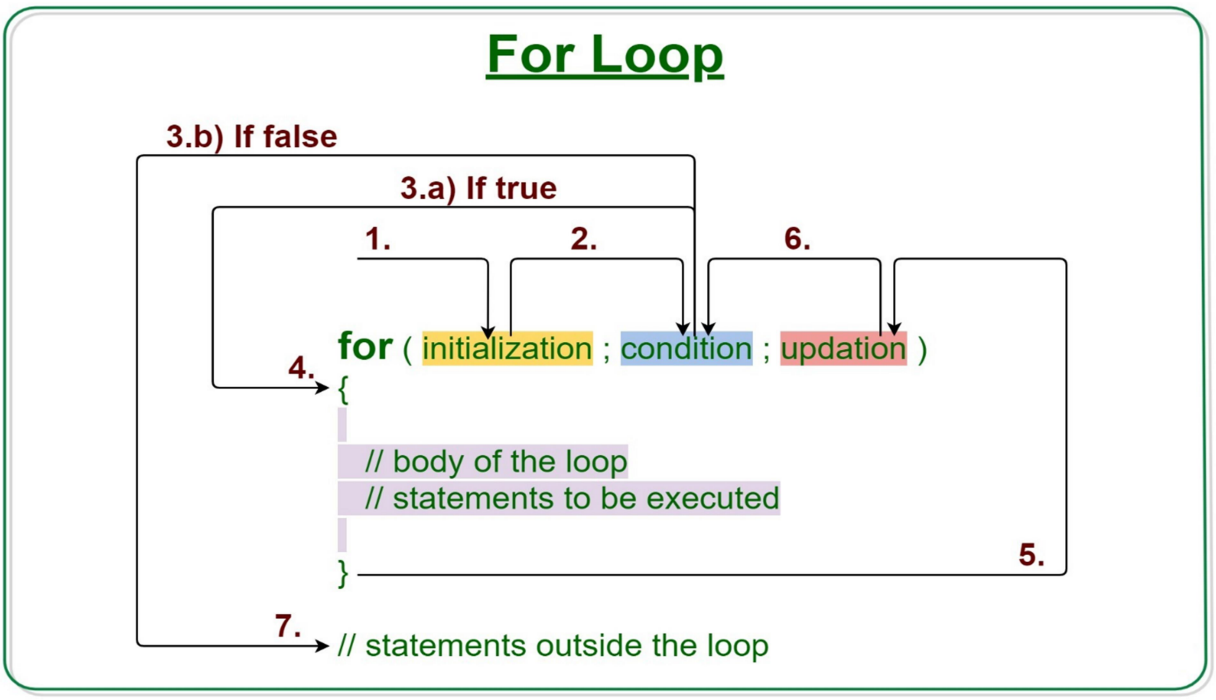
Loops or Iterative Control Structures



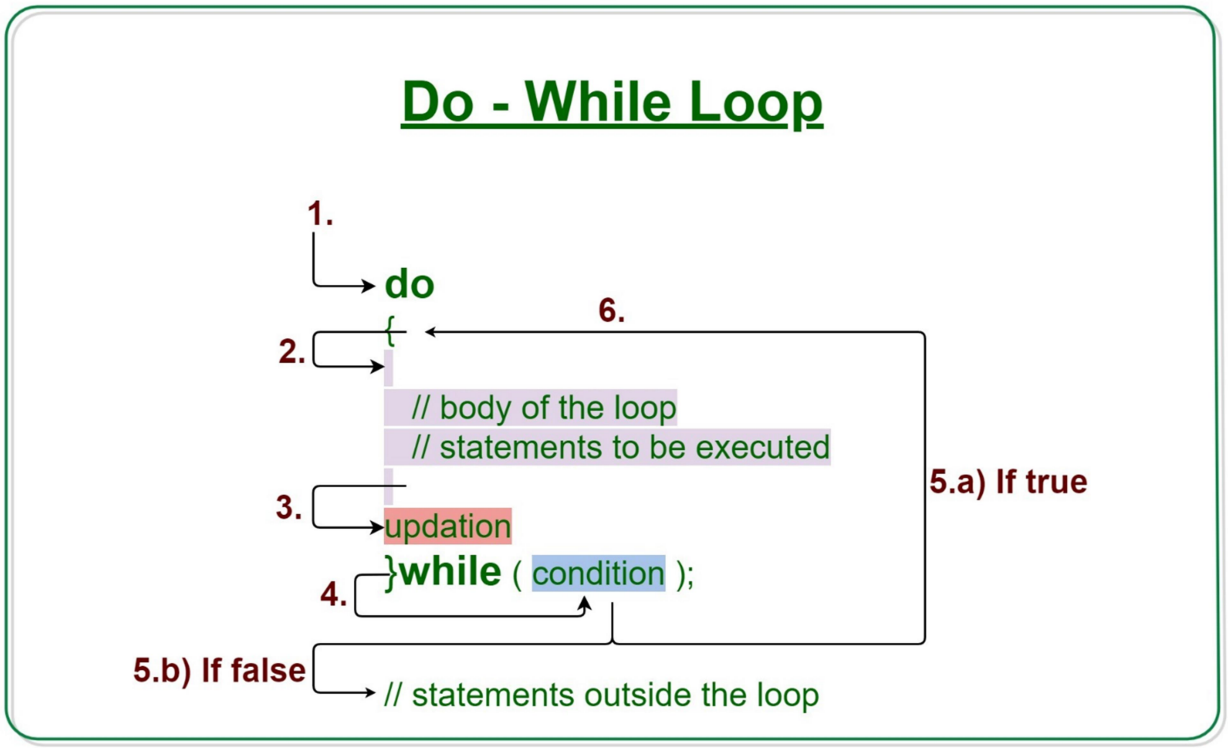
While Loop



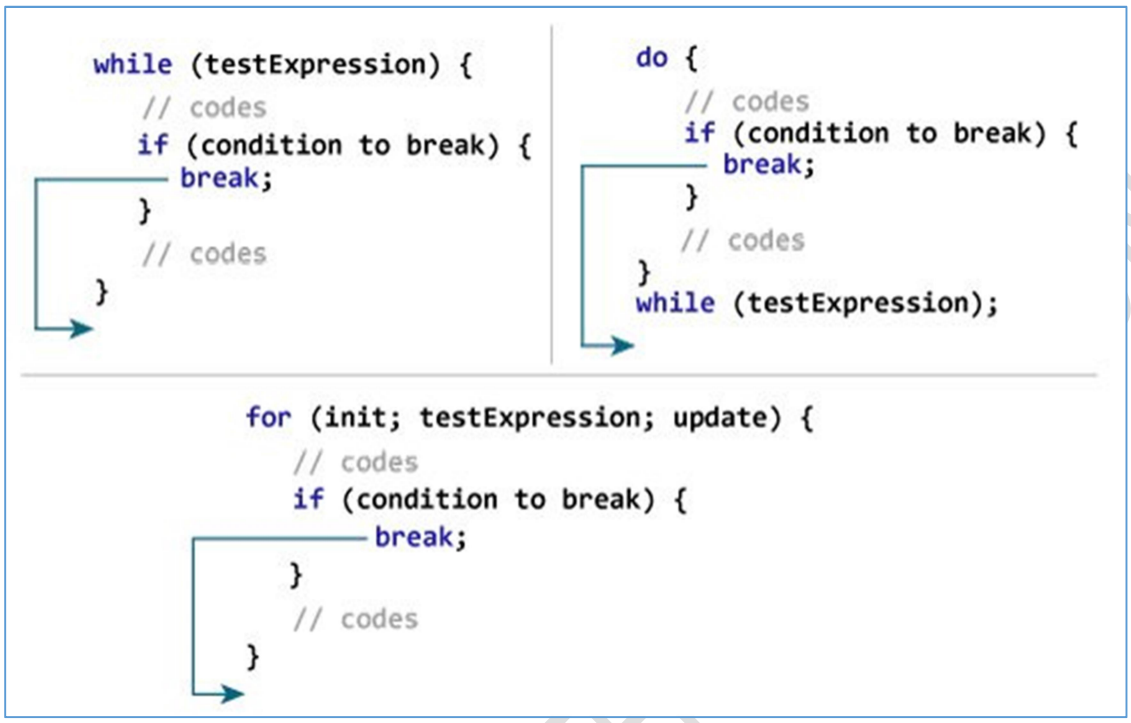
For Loop



Do .. While Loop



Break Statement



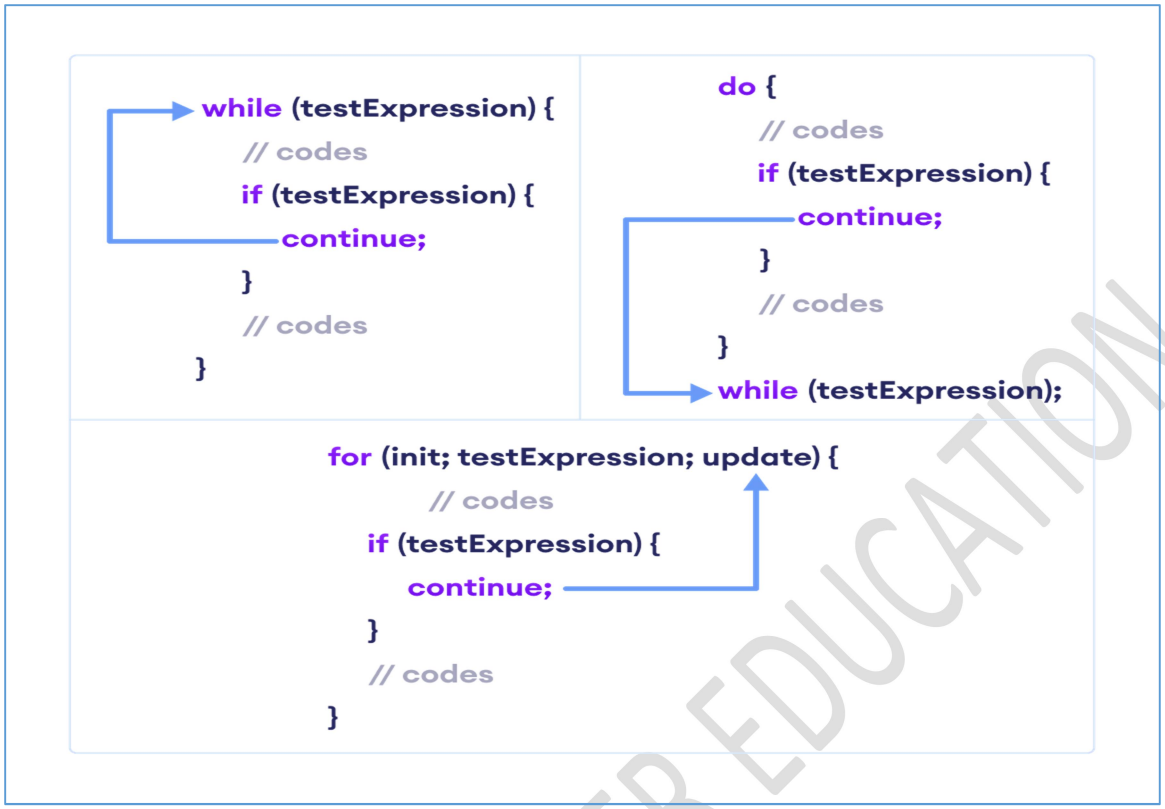
```
while (testExpression) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

```
do {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
while (testExpression);
```

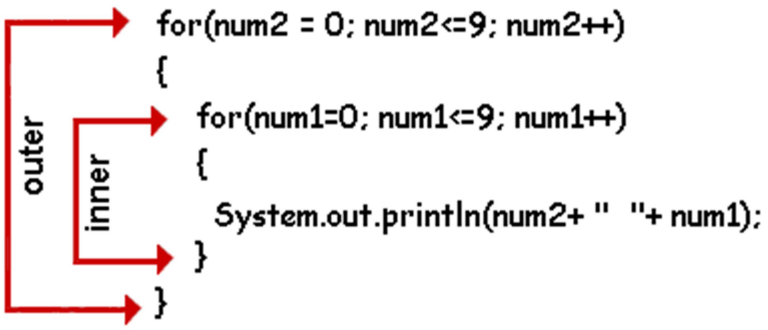
```
for (init; testExpression; update) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

Continue

SSIT COMPUTER



Nested Loops

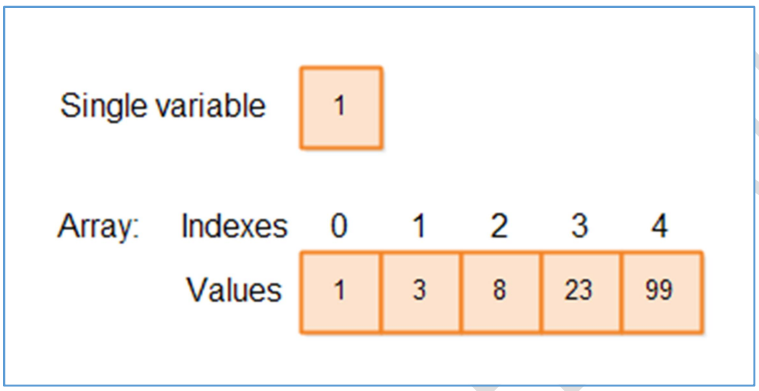


SSSIT COMPUTER EDUCATION

SECTION 6: ARRAYS

- What is Array and its advantages
- Types of Array
- Jagged Array
- Real-time Practicals
    - Matrix Addition, Multiplication, Transpose

Difference between Variable and an Array



What is Array?

- Collection of elements.
- All the elements must be same data type (homogenous types)
- Every individual element is referred using a unique number called Index or position
- Array index always begins with zero.

How to declare an array in Java?

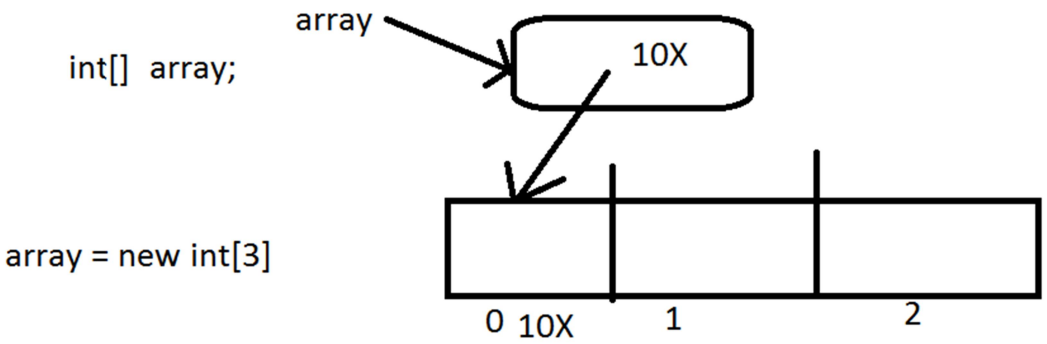
SSIT COMPUTER EDUCATION

How to declare a normal variable?

int num;

---

int[] array;



Two Steps:

Step 1: Allocate the memory to store the array reference.

<<data-type>> variable[];

or

<<data-type>>[] variable;

Example:

int array[];

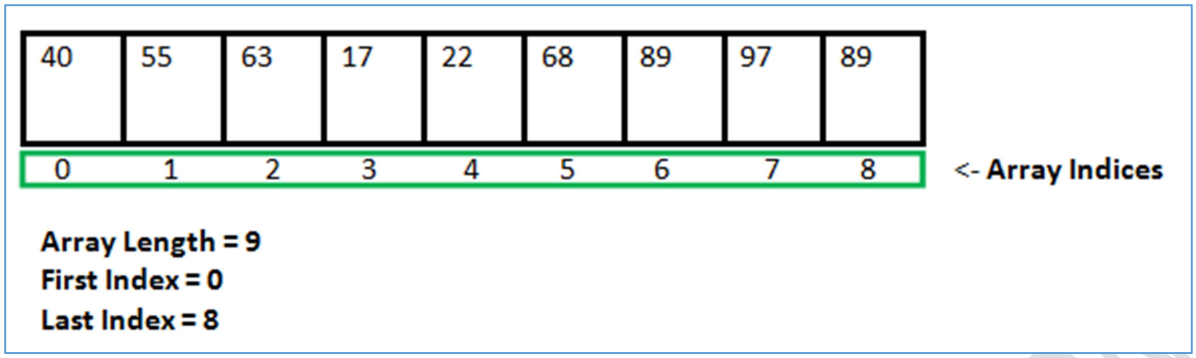
Step 2:

Allocate the required amount memory using "new" operator.

<<variable>> = new <<data-type>>[size];

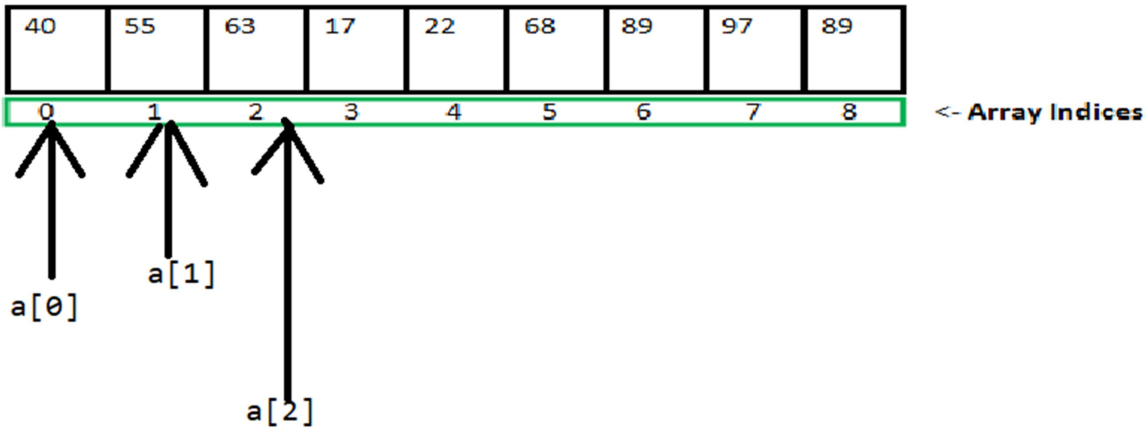
Ex: array = new int[9];

SSIT COMPUTER EL



How to access the array elements?

<<array-variable>>[index];



Example:

a[1] --- refers element present in first index position.

a[1] ==== a + 1 === 100X + 1X4 === 104X

Write a Java program that accepts and display array elements.

```
import java.util.Scanner;
class ArrayDemo
{
    public static void main(String[] args)
```


```
    {
        Scanner scan = new Scanner(System.in);

        int ar[]; /* ar is ref. variable */
        ar = new int[10]; /* new operators allocates 5 int. cells */

        System.out.println("Enter Array elements:");
        for(int i=0;i<ar.length;i++)
            ar[i] = scan.nextInt();

        /* ar.length gives the size of an array ar */

        System.out.println("Array elements are as follows:");
        for(int i = 0; i<ar.length;i++)
            System.out.println(ar[i]);
    }
}
```

How to Initialize the array?

```
TYPE variable[] = {value1, value2, ..., valueN};
eg.
int marks[] = {45, 56, 76, 55};
```

<<data-type>> <<array-name>>[] = {.,.,.,...};

Write a java program to display the sum of array elements.

class ArraySum

COMPUTER EDUCATION

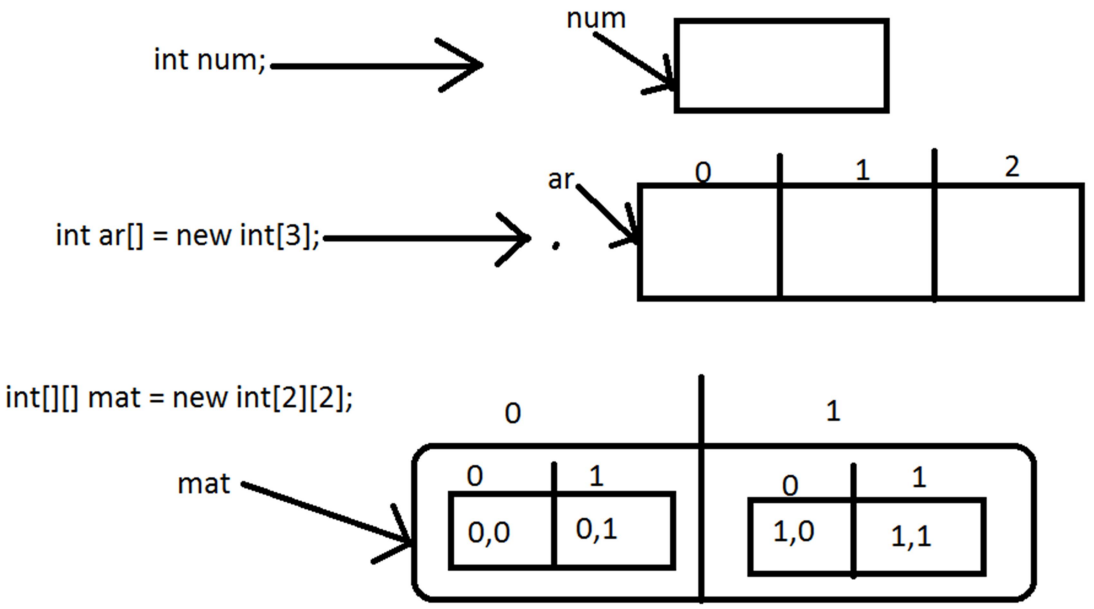

```
{
        public static void main(String[] args)
        {
                int ar[] = {10,5,3,15,1};
                int sum = ar[0];

                for(int i=1;i<ar.length;i++)
                        sum += ar[i];
                System.out.println("Sum is: " + sum);
        }
}
```

Types of Arrays:

1. Single Dimensional arrays
2. Multi dimensional arrays
   2.1 Two Dimensional array
   2.3 Three Dimensional array
   .....

How to declare a two dimensional array?

SSIT COMPUTER EDUCATION



Step 1: we need to a reference variable of type array

int a[][] or int[][] a or int[] a[]

Step 2: allocate the required amount of memory using new operator

a = new int[3][4];

first sub script operator --- rows

second ---- columns

SSIT COMPUTER ED

	Column 1	Column 2	Column 3	Column 4
Row 1	a[0][0] 1	a[0][1] 2	a[0][2] 3	a[0][3] 0
Row 2	a[1][0] 4	a[1][1] 5	a[1][2] 6	a[1][3] 9
Row 3	a[2][0] 7	a[2][1] 0	a[2][2] 0	a[2][3] 0

Write a java program that accepts and display a two dimensional array.

```
import java.util.Scanner;
class TwoDemo1
{
	public static void main(String[] args)
	{
		Scanner scan = new Scanner(System.in);
		int[][] arr = new int[2][2];

		System.out.println("Enter two dimensional array");

		for(int i=0;i<arr.length;i++)
		{
			for(int j=0;j<arr[i].length;j++)
				arr[i][j] = scan.nextInt();
		}
```

SSIT COMPUTER EDU

```
            System.out.println("Array elements are:");
            for(int i=0;i<arr.length;i++)
            {
                  for(int j=0;j<arr[i].length;j++)
                        System.out.print(arr[i][j]+ "\t");
                  System.out.println();
            }

      }
}
```

Initialization of Two dimensional array

<<data-type>> <<array-name>>[][] = {{first_row},{second_row},.,.,.,...};

Write a Java program to find the sum of two matrix.

```
class SumMatrix
{
      public static void main(String[] args)
      {
            int[] a[] = {{1,2},{3,4}};
            int[][] b = {{5,6},{7,8}};

            int c[][] = new int[2][2];


            for(int i=0;i<a.length;i++)
            {
                  for(int j=0;j<a[i].length;j++)
```

SSSIT COMPUTER EDUCATION


```
                    c[i][j] = a[i][j] + b[i][j];
            }

            System.out.println("Array elements are:");
            for(int i=0;i<c.length;i++)
            {
                for(int j=0;j<c[i].length;j++)
                    System.out.print(c[i][j]+ "\t");
                System.out.println();
            }

        }
}
```

SSIT COMPUTER EDUCATION

Three dimensional array

How to declare a two dimensional array?

Step 1:

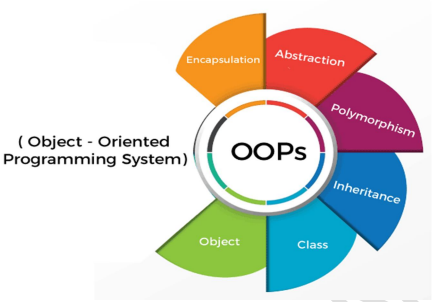
```
int a[][][];
int[][][] a;
int[] a[][];
int[][] a[];
```

Step 2:

```
a = new int[2][2][2];
```

SSIT COMPUTER EDUCATION

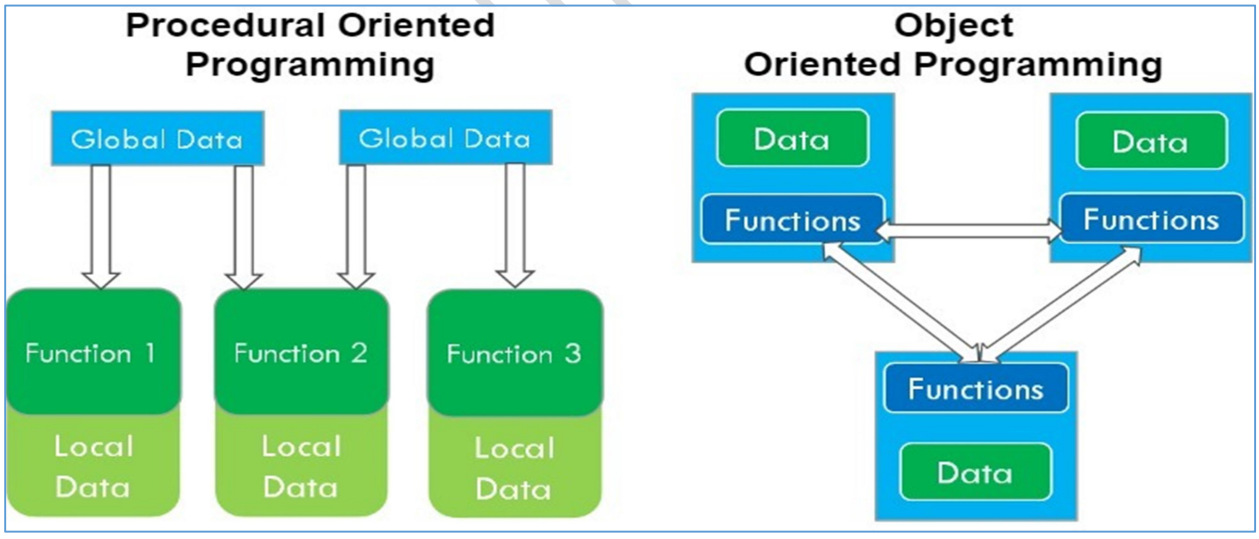
SECTION 7: OOPS INTRODUCTION AND BASICS



- What is OOPS?
- Necessity and Advantage of OOPS
- OOPS Designs with real-time examples.
- What is mean by class and object?
- Relation between a Class and an Object
- How to create class and object
- Real-time Practical's
    - Give the real time example for Object and describes its properties and functionalities.
    - Analyze the OOPs concept and give real-time examples of all OOPS concepts in your own thinking

What is OOPs?

- It stands for Object-Oriented Programming.
- It is based on objects
- It follows Bottom-up programming approach.
- It is based on real world.
- It provides data hiding so it is very secure.
- It provides reusability feature(Write Once Read Anywhere - WORA).

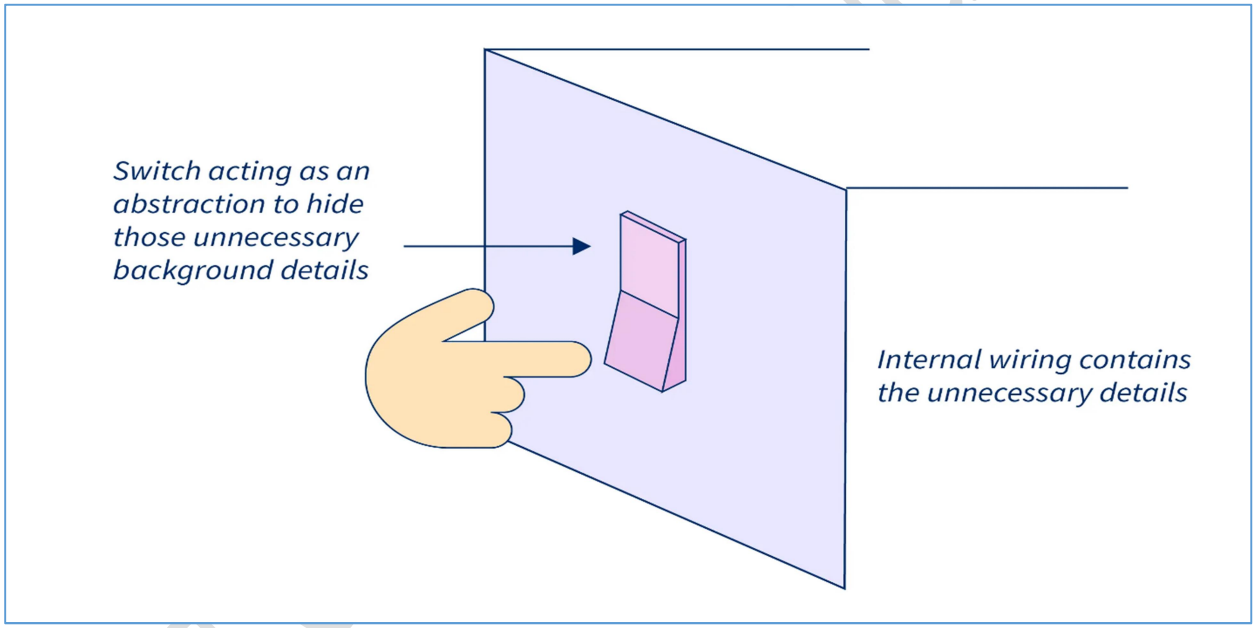


What are the different Object Oriented features are available?

- ABSTRACTION
- ENCAPSULATION
- INHERITANCE
- POLYMORPHISM

What is Abstraction?

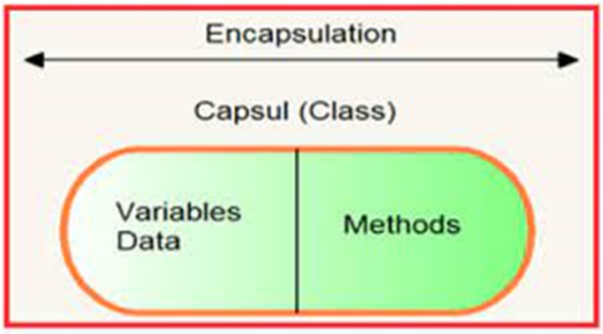
- ABSTRACTION IS THE FEATURE OF EXTRACTING ONLY THE REQUIRED INFORMATION FROM OBJECTS. FOR EXAMPLE, CONSIDER A TELEVISION AS AN OBJECT.
- IT HAS A MANUAL STATING HOW TO USE THE TELEVISION.
- HOWEVER, THIS MANUAL DOES NOT SHOW ALL THE TECHNICAL DETAILS OF THE TELEVISION, THUS, GIVING ONLY AN ABSTRACTION TO THE USER
- Examples:
- The first example, let's consider a Car, which abstracts the internal details and exposes to the driver only those details that are relevant to the interaction of the driver with the Car
- The second example, consider an ATM Machine; All are performing operations on the ATM machine like cash withdrawal, money transfer, retrieve mini-statement…etc. but we can't know internal details about ATM
- In Java, we have Access modifiers like public, private and so on are used to implement Abstraction principle.



What is Encapsulation?

- Encapsulation is a process of wrapping the data and methods into a single unit is called encapsulation.
- In OOP, data and methods operating on that data are combined together to form a single unit, which is referred to as a Class.
- DETAILS OF WHAT A CLASS CONTAINS NEED NOT BE VISIBLE TO OTHER CLASSES AND OBJECTS THAT USE IT. INSTEAD, ONLY SPECIFIC INFORMATION CAN BE MADE VISIBLE AND THE OTHERS CAN BE HIDDEN. THIS IS ACHIEVED THROUGH ENCAPSULATION, ALSO CALLED DATA HIDING. BOTH ABSTRACTION AND ENCAPSULATION ARE COMPLEMENTARY TO EACH OTHER.
- examples

- Capsule, it is wrapped with different medicines. In a capsule, all medicine is encapsulated inside a capsule.



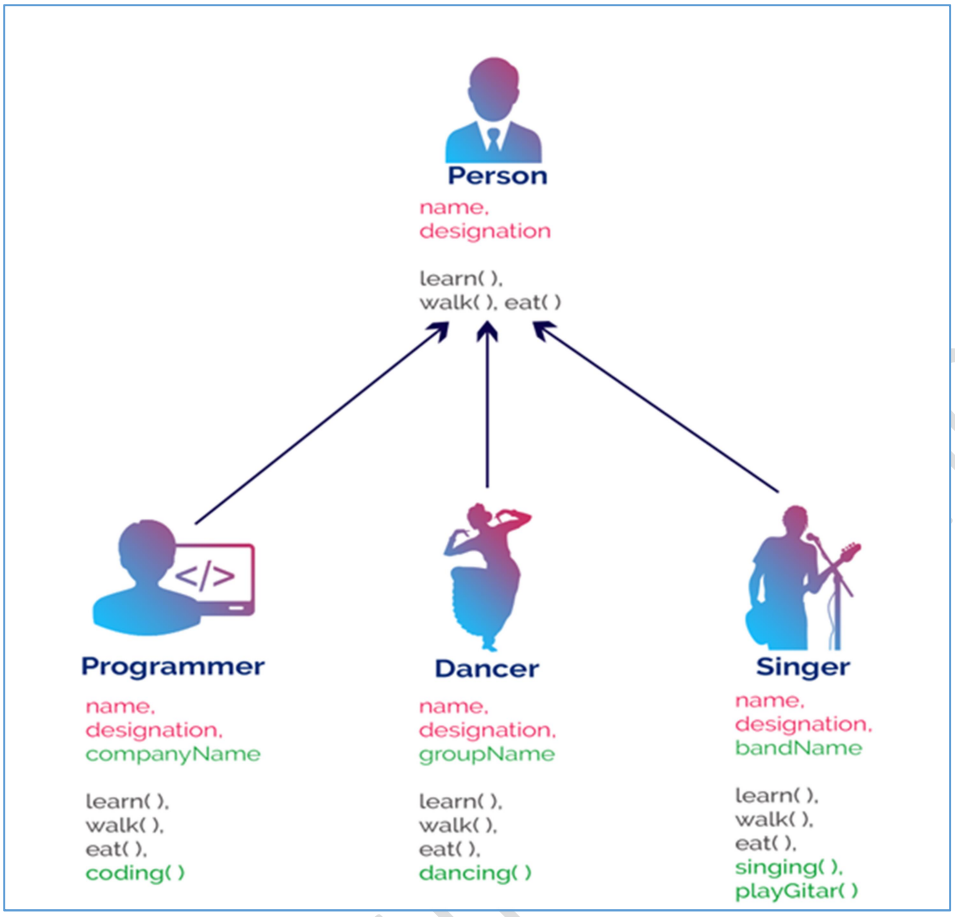
What is the difference between Abstraction and encapsulation?

- Abstraction and Encapsulation in Java are two important Object-oriented programming concept and they are completely different to each other.
- Encapsulation is a process of binding or wrapping the data and the codes that operate on the data into a single entity. This keeps the data safe from outside interface and misuse.
- Abstraction is the concept of hiding irrelevant details. In other words, make the complex system simple by hiding the unnecessary detail from the user.
- Abstraction is implemented in Java using access modifiers, interface and abstract class while Encapsulation is implemented using packages and classes.
- Abstraction solves the problem in the design level. Whereas Encapsulation solves the problem in the implementation level

What is Inheritance?

- INHERITANCE IS THE PROCESS OF CREATING A NEW CLASS BASED ON THE ATTRIBUTES AND METHODS OF AN EXISTING CLASS.
- THE EXISTING CLASS IS CALLED THE BASE CLASS WHEREAS THE NEW CLASS CREATED IS CALLED THE DERIVED CLASS.
- THIS IS A VERY IMPORTANT CONCEPT OF OBJECTORIENTED PROGRAMMING AS IT HELPS TO REUSE THE INHERITED ATTRIBUTES AND METHODS

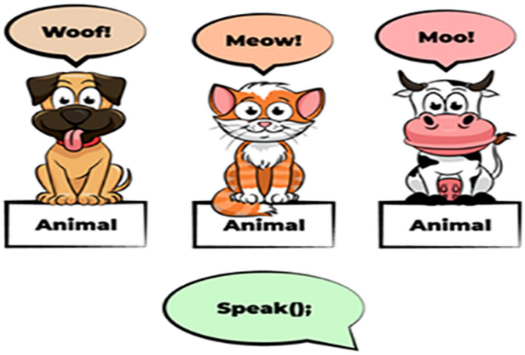
SSM COMPUTER EDUCATION

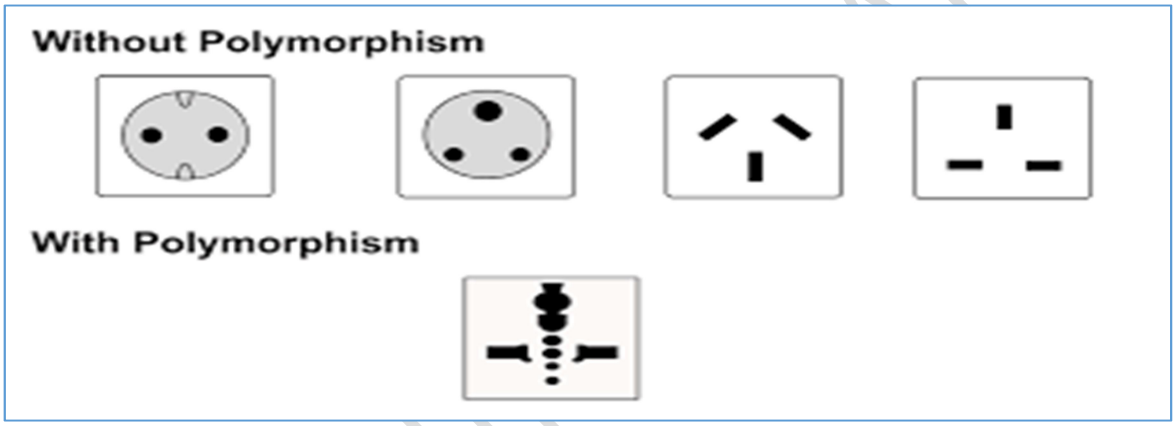


What is Polymorphism?

- POLYMORPHISM IS THE ABILITY TO BEHAVE DIFFERENTLY IN DIFFERENT SITUATIONS.
  IT IS BASICALLY SEEN IN PROGRAMS WHERE YOU HAVE MULTIPLE METHODS DECLARED WITH THE SAME NAME BUT WITH DIFFERENT PARAMETERS AND DIFFERENT BEHAVIOR
- The process of representing one form in multiple forms is known as Polymorphism.
- Different definitions of Polymorphism are:
  - Polymorphism let us perform a single action in different ways.
  - Polymorphism allows you to define one interface and have multiple Implementations
  - We can create functions or reference variables which behaves differently in a different programmatic context.
  - Polymorphism means many forms.
- A real-world example of polymorphism
  - Suppose if you are in a classroom that time you behave like a student,
  - when you are in the market at that time you behave like a customer,
  - when you at your home at that time you behave like a son or daughter,
  - Here one person present in different-different behaviors.

SSSITCOMPUTER EDUCATION





SSIT COMPUTER

SECTION 8: OOPS BASICS (CLASSES & OBJECTS)

- Components of a Class
- Types of Variables and its uses.
- Method Advantages, Categories and Types
- Constructor advantages and its types
- Ways to initialize the Object
- "this" keyword
- Static Block & Instance Block
- Nested classes
- Real-time Practicals
    - Get the real time examples for static variables, final variables and instance variables in your own thinking.
    - Create the Class Student with School name – "Gandhi School", roll_no and name. Create the objects and assign the roll_no and name values using the constructor and display the students using methods

What is a class?

- A class is a collection of objects. Classes don't consume any space in the memory.
- It is a user defined data type that act as a template for creating objects of the identical type.
- A large number of objects can be created using the same class. Therefore, Class is considered as the blueprint for the object.

What is an object?

- An object is a real world entity which have properties and functionalities.
- Object is also called an instance of class. Objects take some space in memory.

For eg .

- Fruit is class and its object s are mango ,apple , banana
- Furniture is class and its objects are table , chair , desk

SSSIT COMPUTER EDUCATION

class is a combination of data members and related methods together.

Data members represents the state of a class.

Ex: htno,number,name,salary,year……(Noun in English)

Methods represents the behaviour of the class.

Example:

accept() --- used to read and store the details.

display() --- used to display the details

factorial() --- perform factorial

How to define a class in java?

```
class  <<name-of-the-class>>
{
        <<data-type>> variable1;
        <<data-type>> variable2;
        .
        .
        .
        <<return-type>> method-name1(arguments){.....}
        <<return-type>> method-name2(arguments){.....}
        <<return-type>> method-name3(arguments){.....}
        .
        .
        .
}
```

How to create an object of a class?

Step 1: create a reference variable of type class

```
class-name  ref-variable;
```

step 2: allocate the memory for the class members.

```
Ref-variable = new class-name();
```

How to access the members of a class?

```
ref-variable.member
```

Members of a class:

1. Non static Data members(Instance variables)
2. Non static Methods
3. Constructors
4. Static data members
5. Static methods
6. Static block

SSIT COMPUTER EDUCATION

7. Non – static or instance block

Methods:

What is a function?

- Sub program
- Used to perform the specific task
- And returns the value to the calling function.

Member function of a class is called Method.

Syntax:

<<access-modifier>><<return-type>><<Method-name>>(list of parameter declarations)

{

.

.

.

[return<<parameter>>;]

}

Write a java program for sum of two complex numbers.

```
class Complex
{
        int real;
        int img;

        void accept(int real,int img)
        {
                this.real = real;
                this.img = img;
```

SSIT COMPUTER EDUCATION


```
    }

    void display()
    {
        System.out.println(real + " +i " + img);
    }

    Complex addition(Complex t=c2)
    {
        Complex temp = new Complex();
        temp.real = real + t.real;
        temp.img = img + t.img;

        return temp;
    }

}

class ComplexAddition
{
    public static void main(String[] args)
    {
        Complex c1 = new Complex();
        Complex c2 = new Complex();
        Complex c3 = new Complex();

        c1.accept(10,15);
        c2.accept(20,25);
```

SSSIT COMPUTER EDUCATION

```
        c3=c1.addition(c2);

        c3.display();


    }
}
```

CONSTRUCTORS

- Is a special method which is used to initialize the state of the object.

How constructor is different with methods?

Constructor	Method
Implicit call	Explicit call
There should not any return type including void	There must be a return type at least void
Constructor name must match with the class name	No rules for naming the method
Executed once in a life span of object	Executed whenever a method is called
Automatically called by the JVM at the time of creating an object.	User has to call explicitely.

Rules for defining a constructor:

1. Constructor name must be similar to class name.
2. Constructor will be called automatically whenever we create object.
3. Constructor never returns any value even void (If we write ant return type then it is treated as ordinary method.)
4. Constructors should not be static (as and when we create an object constructor must be called each and every time).
5. Constructor access specifier may or may not be private. If it is private then an object of the corresponding class is not possible to create in the context of the some other class. But it is possible to create within the same class context.
6. If it is 'not private' then an object of the class can be created within the context of the same class and context of another class.

SSIT COMPUTER EDUCATION

Syntax:

Class <<class-name>>

{

.....

```
<<class-name>>(arguments)
{
.
.
}
```

}

Types of constructors

We have two types of constructors:

1. Default constructor
   Defined by the JVM in the absence of user defined constructor
   Assigns the default values to the members of a class

Member type	Default value
Integer	0
Float	0.0
Character	Space
Boolean	False
Object	Null

Write a java program to demonstrate default constructor?

```
class DefaultConstructor
{
        int num;
        float num1;
        char var2;
        boolean var3;
        String var4;
```

CSSIT COMPUTER EDUCATION


```
        void display()
        {
                System.out.println("Num = " + num);
                System.out.println("Num1 = " + num1);
                System.out.println("Var2 = " + var2);
                System.out.println("Var3 = " + var3);
                System.out.println("Var4 = " + var4);
        }

}

class DefaultConstructorDemo
{
        public static void main(String[] args)
        {
                DefaultConstructor dc = new DefaultConstructor();
                dc.display();
        }
}
```

2. User defined constructor
    a. No argument constructor
    b. Augmented or parameterized constructor
    c. Copy constructor

No Argument Constructor

```
class ConstructorRT
{
        ConstructorRT()
        {
                System.out.println("Hai.....");
        }
}

class ConstructorRTDemo
```

SSIT COMPUTER EDUCATION


```
{
	public static void main(String[] args)
	{
		ConstructorRT crt = new ConstructorRT();
	}
}
```

Program – 2

```
class NoArgConstructor
{
	int num;

	NoArgConstructor()
	{
		num = 100;
	}

	void display()
	{
		System.out.println("Number is:" + num);
	}
}
class NoArgConstructorDemo
{
	public static void main(String[] args)
	{
		NoArgConstructor nac = new NoArgConstructor();
		nac.display();
```

SSSIT COMPUTER EDUCATION


```
        }
}
```

Argumented Constructor or Parameterized Constructor

```
class ConsClass
{
        int var;

        /* Defining the constructor */
        ConsClass(int p)
        {
                System.out.println("Hi....Constructor");
                var = p;
        }

        void display()
        {
                System.out.println("Value = " + var);
        }
}


class ConClassDemo
{
        public static void main(String[] args)
        {
                System.out.println("Hai....");
                ConsClass cs = new ConsClass(100);
```

SSSIT COMPUTER EDUCATION


```
            System.out.println("Object created....");
            cs.display();
        }
}
```

Write a Java program that accepts and display branch details using augmented constructor.

```
class Branch
{
        int bID;
        String bName;
        String hod;

        Branch(int b,String bName,String hod)
        {
            this.bID = b;
            this.bName = bName;
            this.hod = hod;
        }

        void display()
        {
            System.out.println("Branch ID = " + bID);
            System.out.println("Branch Name = " + bName);
            System.out.println("HOD = " + hod);
        }

}
```

SSSIT COMPUTER EDUCATION

```
import java.util.Scanner;
class BranchDemo
{
	public static void main(String[] args)
	{
		Scanner scan = new Scanner(System.in);

		System.out.println("Enter ECE Branch ID:");
		int bID = scan.nextInt();
		System.out.println("Enter ECE HOD Name:");
		String hod = scan.next();

		Branch ece = new Branch(bID,"ECE",hod);
		ece.display();

	}
}
```

What happens if the constructor is declared as private?

- We can't create an object outside of the class.

What happen if we gave a return type to the constructor?

- It will become method.

```
class ConstructorRT
{
	// JVM treats as a method
	  void ConstructorRT()
	{
		System.out.println("Hai.....");
	}
```

SSIT COMPUTER EDUCATION


```
}

class ConstructorRTDemo
{
        public static void main(String[] args)
        {
                ConstructorRT crt = new ConstructorRT();
                crt.ConstructorRT();
        }
}
```

How JVM identifies whether is it methd or constructor?

With the support of return type.

Polymorphism.

What is Polymorphism?

- It is one of the object oriented principle.
- Providing multiple implementations to a method or constructor is called polymorphism.

These are of two types:

1. Compile time polymorphism
   a. Constructor Overloading
   b. Method overloading
2. Run time polymorphism
   a. Method overriding

Constructor Overloading

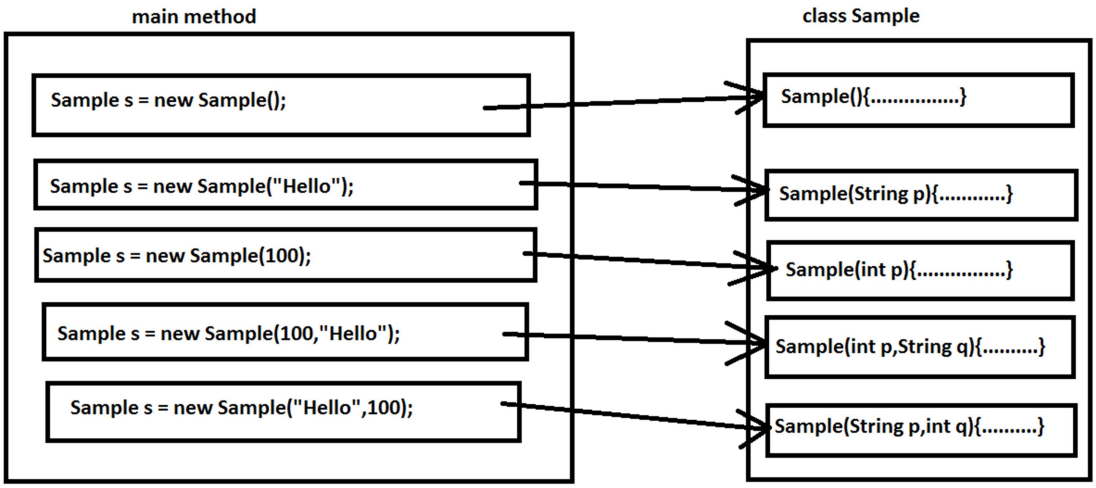
Can we define more than one constructor in a class?

SSIT COMPUTER EDUCATION

- Yes. We can define multiple constructors in a class. Such a concept is called Constructor Overloading.
- Constructor overloading is a type of polymorphism.

How to define multiple constructors in a class?

- We can define multiple constructors in class by changing:
    1. Type of arguments
    2. Number of arguments
    3. Sequence of arguments.



Write a Java program to demonstrate constructor overloading.

```
class SimpleInterest
{
        int prin;
        int term;
        int rate;

        SimpleInterest()
        {
                System.out.println("No Arg. Constructor");
                prin = 1000;
```


```
        term = 1;
        rate = 10;
    }

    SimpleInterest(int prin)
    {
        System.out.println("One Arg. Constructor");
        this.prin = prin;
        term = 2;
        rate = 20;
    }

    SimpleInterest(int prin,int term)
    {
        System.out.println("Two Arg. Constructor");
        this.prin = prin;
        this.term = term;
        rate = 30;
    }

    SimpleInterest(int prin,int term,int rate)
    {
        System.out.println("Three Arg. Constructor");
        this.prin = prin;
        this.term = term;
        this.rate = rate;
    }
```

SSSIT COMPUTER EDUCATION


```
        void display()
        {
                int res = (prin*term*rate)/100;

                System.out.println("Simple Interest is:" + res);
        }


}
class SimpleInterestDemo
{
        public static void main(String[] args)
        {
        SimpleInterest s1 = new SimpleInterest();// No arg. constructor
        SimpleInterest s2 = new SimpleInterest(2000);// One arg. constructor
        SimpleInterest s3 = new SimpleInterest(3000,3);// Two arg. constructor
SimpleInterest s4 = new SimpleInterest(4000,4,40);// Three arg. constructor

                s1.display();
        s2.display();
        s3.display();
        s4.display();
        }
}
```

Is it possible to call another constructor from a constructor?

- Yes. Using method form of this keyword.
- Syntax:

SSSIT COMPUTER EDUCATION

- this(<<parameters>>); → to call argumented constructor
- this() → to call no argument constructor

- This concept is called as constructor chaining.

Note: this() must be the first statement in the constructor definition.

Method Overloading

Is it possible to provide multiple implementations for a method without changing the name of the method?

- Yes. We can provide multiple implementations without changing the method name. This concept is called Method overloading.
- This is another type of polymorphism.
- We can provide multiple implementations by changing:
  1. Type of arguments
  2. Number of arguments
  3. Sequence of arguments.

Write a Java program to demonstrate Method overloading.

```
class MethodOverloading
{
	void display()
	{
		System.out.println("No Arg. Method");
	}

	void display(int p)
	{
		System.out.println("one Arg. Method with int param.");
	}

	void display(String p)
	{
```

SSIT COMPUTER EDUCATION

```
            System.out.println("one Arg. Method with String param.");
        }

        void display(int p,String q)
        {
            System.out.println("Two Arg. Method with int and string");
        }
        void display(String p,int q)
        {
            System.out.println("Two Arg. Method with String and int");
        }

}

class MethodOverloadingDemo
{
        public static void main(String[] args)
        {
            MethodOverloading mov = new MethodOverloading();
            mov.display("Hai...");
            mov.display("Hai..",150);
            mov.display(200,"bye...");
            mov.display();
            mov.display(1000);
        }
}
```

Copy Constructor

```
class CopyConstructor
```

SSSIT COMPUTER EDUCATION


```
{
        int num;

        CopyConstructor(int p)
        {
                num = p;
        }

        /* Copy constructor */
        CopyConstructor(CopyConstructor temp)
        {
                num = temp.num;
        }

        void display()
        {
                System.out.println("Number is:" + num);
        }
}

class CopyConstructorDemo
{
        public static void main(String[] args)
        {
                CopyConstructor cc = new CopyConstructor(100);

                //CopyConstructor cc1 = cc;
                CopyConstructor cc1 = new CopyConstructor(cc);
```

SSSIT COMPUTER EDUCATION


```
            cc1.num = 0;

            cc.display();
            cc1.display();
            System.out.println(cc);
            System.out.println(cc1);
        }
}
```

SSIT COMPUTER EDUCATION

Second approach--- With the support of object

Step 1: create the object of a class

<<class-name>><<ref-variable>> = new <<class-name>>(parameters);

Step 2: call the static method using ref – variable

<<ref-variable>>.<<method-name>>(parameters);

Note:

1. All static methods works at application level where as non – static methods works at object level
2. We can access both static and non – variables from non – static methods. Whereas we can access only static variables from static methods.

SSSIT COMPUTER EDUCATION

STATIC VARIABLES

If we declare an instance variable using STATIC keyword, then it is called as static variable

Syntax:

<<access-modifier>> static <<data-type>><<variable>> = <<value>>;

How to access a static variable?

Instance variable	Static variable

Class Sample
{
Intnum = 100;
.
.
.
}
 	
Class SampleDemo
{
Public static void main(String args[])
{
	Class Sample
	{
	Static intnum = 100;
	.
	.
	.
	.
	}
	
	Class SampleDemo
	{
	Public static void main(String args[])

Sample s = new Sample();
System.out.println(s.num);
}
}
	{
	System.out.println(Sample.num);
	Approach – 1
	
	Sample s = new Sample();
	System.out.println(s.num);
	Approach – 2
	}
	}

Note:

All non – static variables holds separate memory allocation in every object. This is the meaning of Object level attribute.

Every static variable allocates a common memory allocation to all the objects. This is the meaning of Class level attributes.

Static variables acts like a global variables for all the objects.

```
classJavaCourse
{
        static String instName;
        static String facultyName;
        String studName;
        intstudID;
}
```

Is it recommended to initialize the static variables in a constructor?

NO. Whenever we are creating an object, the static variable updated value resets to the default value.

STATIC BLOCK

Static block is used to initialize the default values to all static variables.

SSSIT COMPUTER EDUCATION

Syntax:

```
static
{
.
.// Initialize the static variables
.
}
```

When the static block executes?

At the time of loading the class into JVM, the static block executes.

When a class loads into JVM?

Two scenarios:

1. Java <<class-name>>
2. At the time creating the first object of that class.

What are the difference constructor and a static block?

Constructor	Static block
Is a Method	It is a block What is a block? Set of statements enclosed between curly braces.
Executes at the time of creating an object	Executes at the time of loading a class
We can pass default values at run time	We must provide the default values at compile time
We can initialize both instance variables and static variables	We can't initialize instance variables. We are allowed to initialize only static variables.

Can we define multiple static blocks in a class?

SSSIT COMPUTER EDUCATION

YES.

In which sequence the static blocks will be executed if we define more than once?

It executes the sequence in which we defined.

Or

In the order which we followed in class a definition

Instance Block:

- Set of statements enclosed between curly braces without using static keyword is called instance block
- It executes whenever we are creating an object before constructor.

Syntax:

```
{
.
// Set of statements
.
}
```

Instance block is used to initialize both static and instance variables.

SSIT COMPUTER EDUCATION

SECTION 9: OOPS CONCEPTS – INHERITANCE

- Inheritance and its advantages
- The “extends” keyword
- Types of Inheritance
    - Single Inheritance
    - Multilevel Inheritance
    - Multiple Inheritance
    - Hierarchical Inheritance
    - Hybrid Inheritance
- Difference between IS-A and HAS-A relationship
- Use of “super” keyword and forms of “super” keyword
- Real-time Practicals
    - Try to get the real time scenarios for all kinds of inheritance in your own thinking.
    - In College , We are having 3 kinds of Teachers. For Ex: English Teacher, Tamil Teacher, Maths Teacher. These Teachers having different main subjects but their designation is ‘Teacher’, college is ‘Nehru College’ and functionality is ‘teaching’. Achieve this using Inheritance and display the details of 3 teachers.

What is Inheritance?

- It is one of the main object oriented principle.
- Extending the features of existed class to new class is called Inheritance.
- Existed class is called as Super class
- Newly defined class is called Sub class
- It follows Reusability mechanism.

How to implement Inheritance in Java?

Using Extends keyword to the new class, we can implement inheritance concept in java.

Syntax:

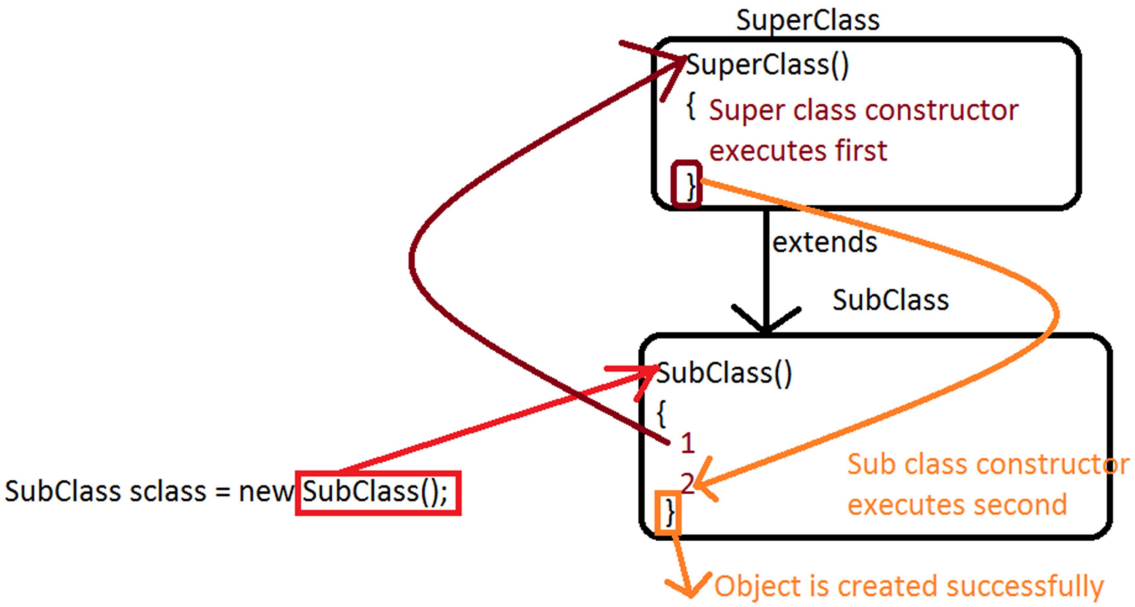
SSIT COMPUTER EDUCATION

```
Class <<new-class>> extends <<existed-class>>
{
// definition of new class
}
```

Note:

- Only variables and methods are extended from super class to sub class.
- Constructor and other properties are the own properties of super class.

Execution sequence of Constructor in Inheritance:



How to pass the arguments from sub class constructor to its super class constructor?

Using method form of super(), we can pass the arguments from sub class constructor to its immediate super class constructor.

What is method form of super()?

- Method form of super is used to pass the arguments from child class constructor to parent class constructor.
- Syntax:

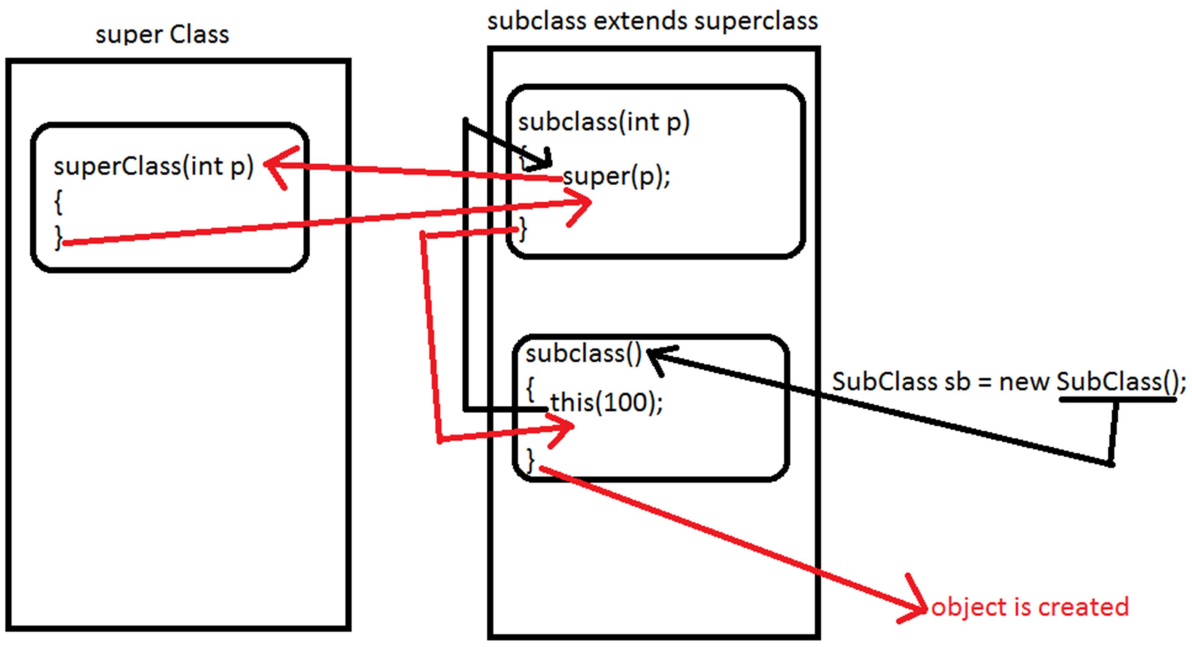
  super(arg1,arg2,.....); → used to call argumented constructor
  super() -→ used to call no argument constructor.

- If we want to call super class constructor from sub class, then super() must be the first statement in child class constructor.

What is the difference between this() and super()?

this()	super()

Used to call another constructor with in the class
this(arg1,arg2,.....); //Argumented
this(); // No argument
	Used to call its immediate super class constructor
	super(arg1,arg2,.....);// Argumented
	super(); // No argument



Is it possible to use this() and super() in one constructor?

NO. because if we want to either this() or super(), then it must be the first statement in the constructor definition.

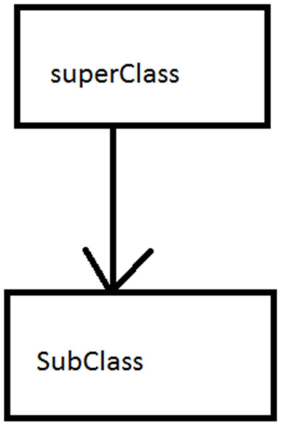
Types of Inheritances

According to OOPS, there are FIVE different types of inheritances:

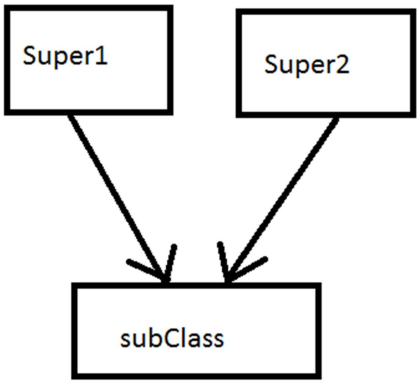
1. Single inheritance
2. Multiple Inheritance
3. Multi-level Inheritance
4. Hierarchal Inheritance
5. Hybrid Inheritance

But, Java supports only THREE Inheritances:

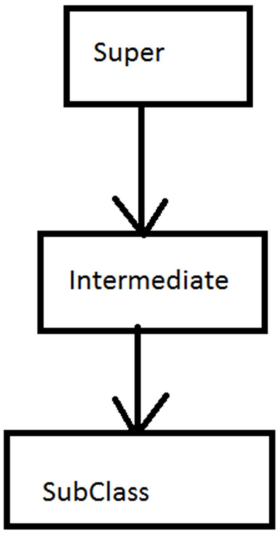
1. Single Inheritance
2. Multi-level Inheritance
3. Hierarchal Inheritance



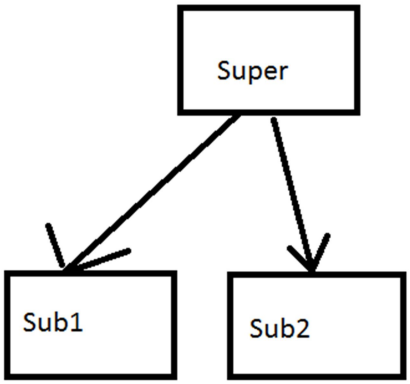
single Inheritance



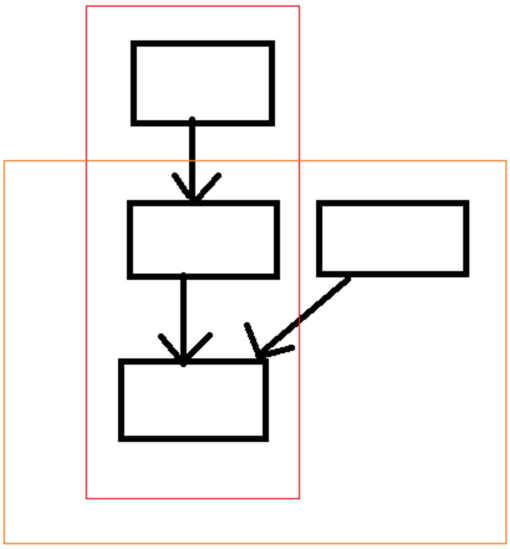
Multiple Inheritance



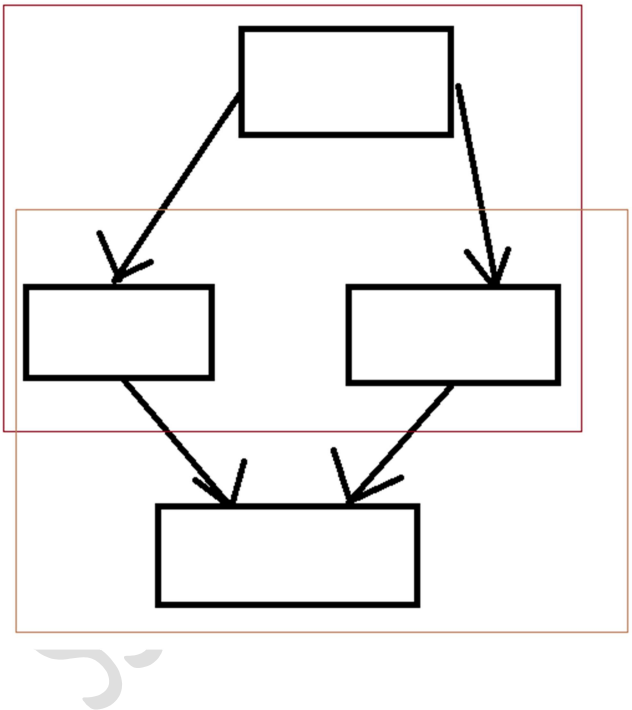
Multi Level Inheritance



Hierarichal Inheritance

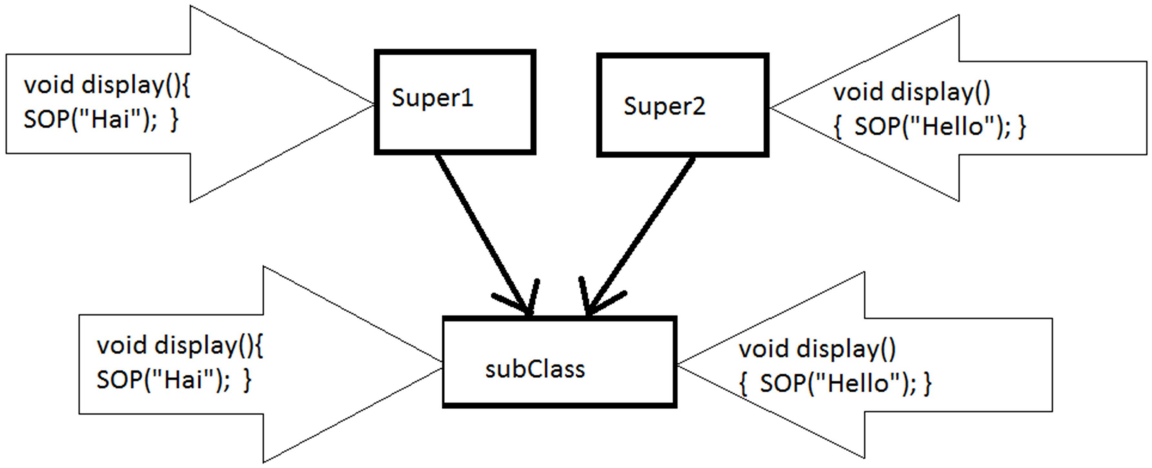


Hybrid Inheritance - 1



Hybrid Inheritance - 2

Why Java doesn't support multiple inheritance directly(using classes)?



Multiple Inheritance

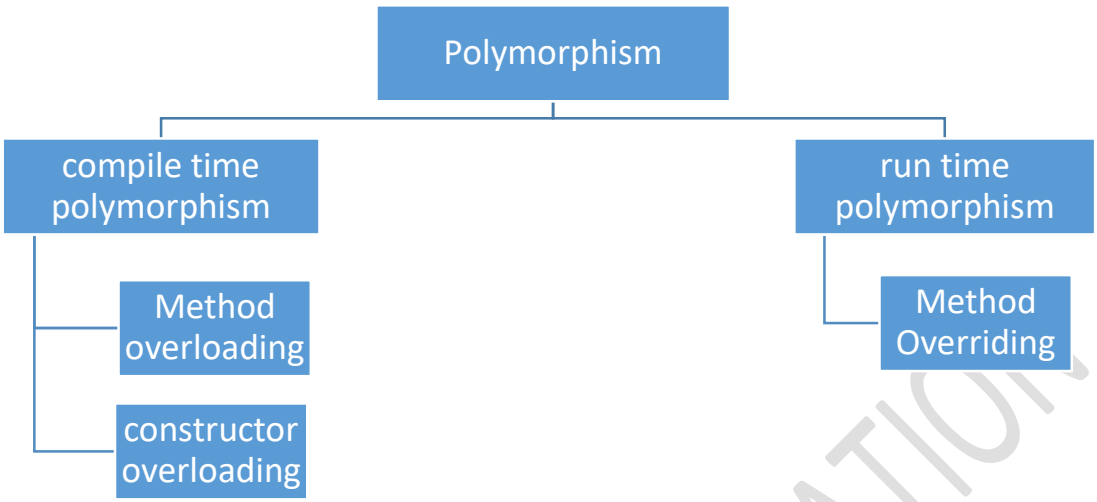
```
SubClass sb = new SubClas();
sb.display();
```

Why Java doesn't support Hybrid Inheritance using classes?

- Hybrid inheritance is the combination of Multiple and Multi level and Hierarichal inheritance.
- As java doesn't support multiple inheritance. It even won't support hybrid inheritance.

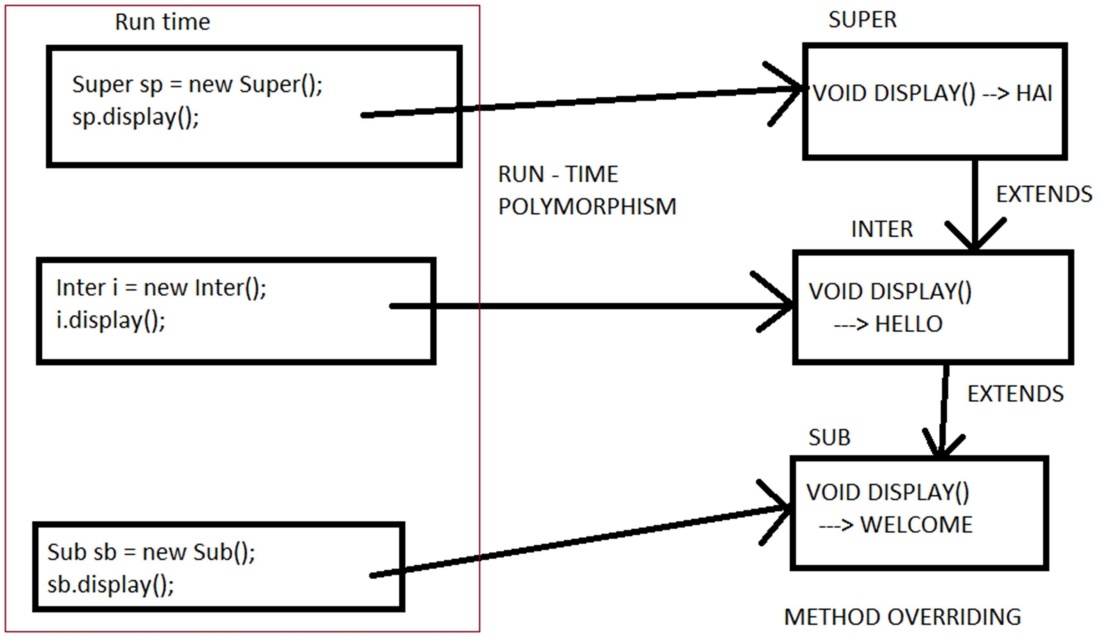
Polymorphism:

SSIT COMPUTER



Method Overriding

- Providing the multiple implementations to methods
- in different classes
- within the hierarchy
- without changing the method signature



What is the difference between Method Overloading and Method overriding?

Method Overloading	Method Overriding
Different implementations with in the class	Different implementations in different classes with in the hierarchy
Signature must be different	Signature must be same
Execution is depends on method call	Always sub class implementation will be executed.
Compile time polymorphism	Run time polymorphism

Rules for overriding:

1. Method signature must be same.
2. If sub class access modifier must have same or higher priority thaa super class access modifiers.

Access modifier used in Super class	Access modifier used in Sub class
Private	Private
Default	Default --- same Protected Public
Protected	Protected --- same Public
Public	Public --- same

- Can we overload main method?
  YES. If we overload, JVM always searches for the method that was defined as public static void main(String[] …)

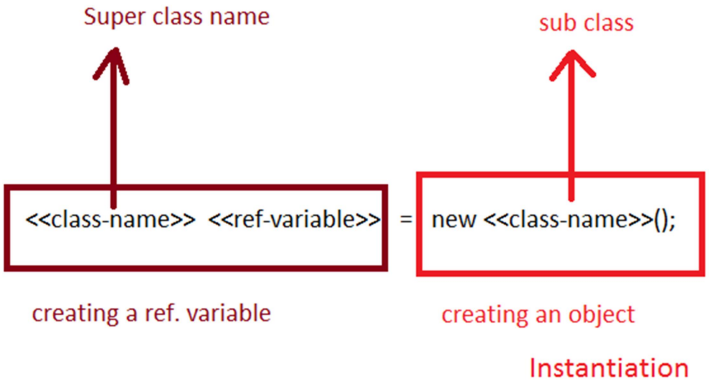
SSJT COMPUTER EDUCATION

- Can we override main method? Or can we override static method?

  YES. But this concept is called METHOD HIDING. Because overriding is applicable at object level where as static methods are at class level.

- What is Data hiding?

Super Class reference – sub class object



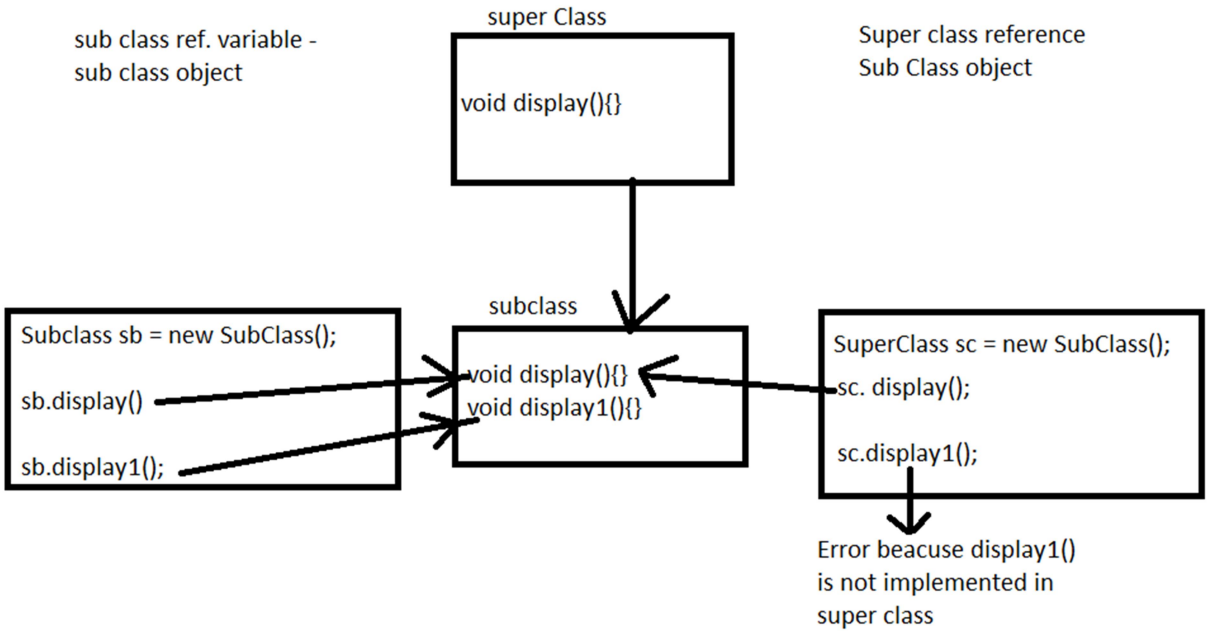
In the normal scenario,

- Whenever we called any method, first JVM checks whether the implementation is available in sub class or not.

- If the method is implemented in sub class, then its implementation will be executed.
- If the method is not implemented in sub class, then JVM checks whether the method is implemented in super class or not.
- If it is implemented then JVM executes the super class method.
- If not implemented then JVM throws a compile time error.

In Super class reference variable – sub class object,

- Whenever we called any method, first JVM checks whether the method is implemented in super class or not.
- If it is implemented in super class, then its sub class definition is executed.
- If it is not implemented in super class, then JVM raises compile time error message.



What is the purpose of Super?

We can use Super in two ways:

1. Method form of super(super())
2. Super keyword

super() is used to pass an arguments to the immediate super class constructor.

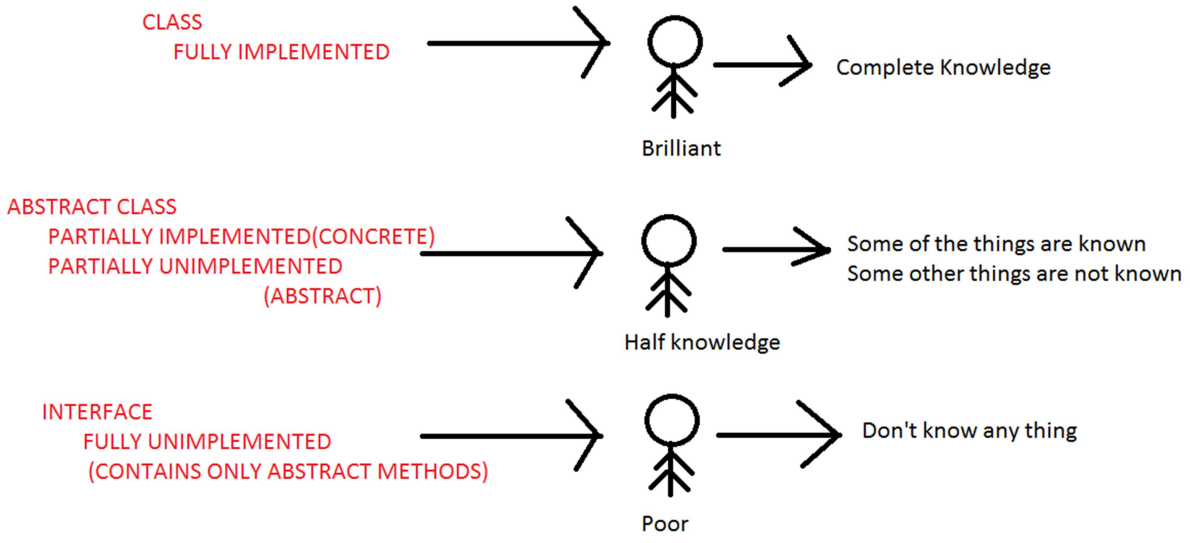
super keyword is used to access the methods or variables defined in super class.

SSIT COMPUTER EDUCATION

SECTION 10: OOPS CONCEPTS – ABSTRACTION

- Abstraction and its advantages
- Abstract Class and Abstract Methods
- The “abstract” keyword
- Implementation of Abstract Methods
- Uses of Abstract Classes
- Interface and its advantages
- ‘implements’ keyword
- Achieve Multiple Inheritance with Interface
- Difference between Abstract Class and Interface
- Real-time Practicals
    - Eating and traveling functionalities are mandatory for all Animals. But food and traveling paths are different. Display the food and traveling paths for different animals. (use interface)
    - All the banks should have a rate of interest . But it’s varied from bank to bank. Display the rate of interestfor all banks. (use interface)

Different types of classes in Java



Difference between class, Abstract Class and Interfaces

Class	Abstract Class	Interface
instance variables	instance variables	static variables
instance methods	instance methods	abstract methods
constructors	constructors	
static variables	static variables	
static methods	static methods	
static block	static block	
instance block	instance block	
	ABSTRACT METHOD	

In Java, We have two different types of Methods:

1. Concrete methods
2. Abstract methods

Concrete method	Abstract method
Implemented method or any method with definition	Any method without definition or an un implemented method

void display()
{
System.out.println(“Hello…”);
}
	abstract void display();

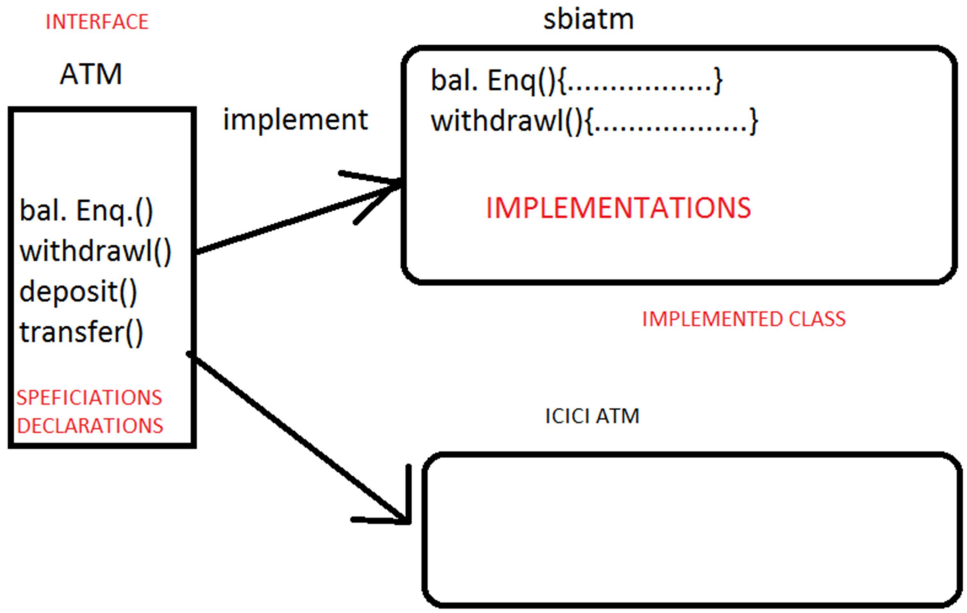
1. Every unimplemented method must be declared using “abstract” keyword.
2. If a class contains at least one abstract method, then such a class must be declared as abstract class using “abstract” keyword.

SSIT COMPUTER EDUCATION

3. Every unimplemented method in super class must be implemented in sub class.
4. We can't create an object for abstract class using "new" keyword. If we create an object then it gives "Instantiation Exception".
5. An abstract class always acts like a super class.

What is interface?

- Contains variables and methods.
- All variables are static and final by default
- All methods are public and abstract by default.
- Using interface, we can provide the set of specifications and those specifications are implemented by the sub classes.



How to define an interface?

Interface <<interface-name>>

{

.

.

}

Ex:

Interface Car

{

IntnoDoors=4;

IntnoTyres = 4;

void features();

}

Note: every interface creates his own .class file

How to implement the features of interface to a class?

Using a keyword “implements”, we can apply the features of an interface to a class.

class<<sub-class>> implements <<interface-name>>

{

.

// override the abstract methods

}

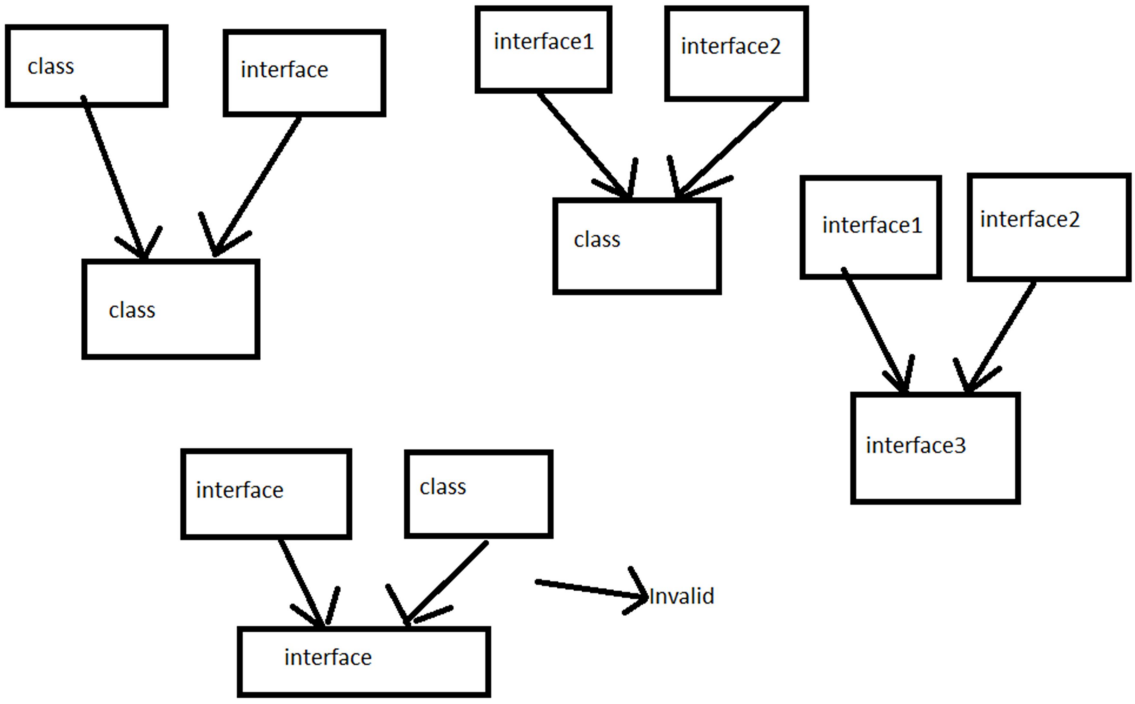
SSSIT COMPUTER EDUCATION


```
Class MaruthiCar implements Car
{
  Public void features()
{
System.out.println("Features of Maruthi Car…");
}
}
Class HyundaiCar implements Car
{
   Public void display()
{
System.out.println("These are the features of Hyundai Car");
}
}
```

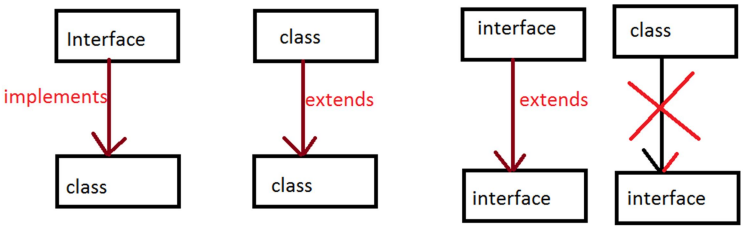
How to implement Multiple Inheritance in Java?

We can implement multiple inheritances in Java using interfaces. Because all interfaces contains only abstract methods.

SSSIT COMPUTER EDUCATION



What is the difference between extends and implements keywords?



Final keyword

In Java, we can final keyword in three ways:

1. To declare constant variables
2. To declare constant methods
3. To declare constant class

How to declare constant variables?

final<<data-type>><<variable-name>>[=<<default-value];

1. instance variables
2. static variables

If we declare any variable using final keyword, then JVM won't allowsto modify the value.

How to apply a final keyword to the methods?

If we apply a final keyword to any method, then it is not eligible for overriding (avoid overriding).

Can we declare an abstract method using final keyword?

Can we declare an abstract method using final keyword?

We can't declare an abstract method using final keyword. Bcz final method must be implemented. Whereas an abstract method is unimplemented method.

Final class

SSSIT COMPUTER EDUCATION

If we declare any class using final keyword, then JVM won't allow to extend the properties of that class to any other class.(avoid inheritance)

SSSIT COMPUTER EDUCATION

SECTION 11: OOPS CONCEPTS – ENCAPSULATION ALONG WITH PACKAGES

- What is package and its advantages
- Types of packages
- Static Import
- Access Modifiers
- Encapsulation
- Real-time Practical's
    - We are having Class AOne. In this class create static variables and static methods. And try to access these variables and methods in Package B classes without using class name and creating an object.
    - Create the class Account with account number and account balance as private members. Set the Account number and deposit the amount. Show the Account Balance. Write the Program to achieve this using Data Encapsulation.

SSIT COMPUTER EDUCATION

SECTION 12: STRINGS and Object Class

- Object Class
- String,
- String Buffer,
- String Builder
- Real-time Practicals
    - Count the number of vowels, consonants, special charaters in the following String: "-God is Great!-"

Object Class

Sl.no	Methods	Signature
1	getClass()	Public final Class getClass();
2	hashCode()	public int hashCode();
3	equals()	public boolean equals(Object)
4	clone()	Protected object clone() throws CloneNotSupportedException
5	toString()	public String toString()
6	notify()	public final void notify();
7	notifyAll()	public final void notifyAll();
8	wait()	public final void wait() throws InterruptedException
		public final void wait(long) throws InterruptedException
		public final void wait(long,int) throws InterruptedException
9	finalize()	protected void finalize() throws Throwable

Various methods in java.lang.Object(Super class for all java classes)

1. getclass()
   Used to find the name of the class
   Syntax: object_name.getClass()

2. hashcode()
   Returns the hascode value of the object.
   Syntax: object_name.hashCode()
   To get in hexa decimal format
   Integer.toHexaString(object_name.hashCode());

```
publicclassClassDemo {
    void display()
```

SSIT COMPUTER EDUCATION

```
	{
		System.out.println("Hello");
	}

}

publicclass Demo1 {

	publicstaticvoid main(String[] args) {
		ClassDemocd1 = newClassDemo();
		System.out.println("getClass = " + cd1.getClass().getName());
		System.out.println("HashCode = " + cd1.hashCode());
		System.out.println("Hexa Decimal format = " +
Integer.toHexString(cd1.hashCode()));
		System.out.println("CD1 = " + cd1);

	}

}
```

3. equals()
   a. Compares the given object to this object.
   b. Syntax: object_name.equals(another_object_name);

   Note:

   - We can override equals method.
   - If we override equals method then we must override hashcode().

The general contract of hashCode is:

- Whenever it is invoked on the same object more than once during an execution of a Java application, the *hashCode* method must consistently return the same integer, provided no information used in *equals* comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
- If two objects are equal according to the *equals(Object)* method, then calling the *hashCode* method on each of the two objects must produce the same integer result.
- It is *not* required that if two objects are unequal according to the *equals(java.lang.Object)* method, then calling the *hashCode* method on

SSIT COMPUTER EDUCATION

each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables.

Program: 2(Without Overriding equals Method)

```
packageWithoutEqualsOverriding;

publicclass Student {
    inthtno;
    String name;
    Student(inthtno,Stringname)
    {
            this.htno = htno;
            this.name = name;
    }

}
packageWithoutEqualsOverriding;

publicclassStudentDemo {

    publicstaticvoid main(String[] args) {
            Student stud1 = newStudent(101,"abc");
            Student stud2 = newStudent(101,"abc");
            Student stud3 = stud1;

            if(stud1.equals(stud2))
                    System.out.println("Both are same");
            else
                    System.out.println("Both are not same");

            if(stud1.equals(stud3))
                    System.out.println("Both are same");
            else
                    System.out.println("Both are not same");


    }

}
```

Program 3(by overriding equals method)

```
packageWithEqualsOverriding;
```

SSSIT COMPUTER EDUCATION


```
publicclass Student {
    inthtno;
    String name;
    Student(inthtno,Stringname)
    {
            this.htno = htno;
            this.name = name;
    }

    publicboolean equals(Student s)
    {
            if(htno==s.htno&&name == s.name)
                    returntrue;
                    else
                            returnfalse;
    }

}
packageWithEqualsOverriding;

publicclassStudentDemo {

    publicstaticvoid main(String[] args) {
            Student stud1 = newStudent(101,"abc");
            Student stud2 = newStudent(101,"abc");
            Student stud3 = stud1;

            if(stud1.equals(stud2))
                    System.out.println("Both are same");
            else
                    System.out.println("Both are not same");

            if(stud1.equals(stud3))
                    System.out.println("Both are same");
            else
                    System.out.println("Both are not same");



    }


}
```

4. Clone()

SSSIT COMPUTER EDUCATION

a. creates and returns the exact copy (clone) of this object.
b. Syntax: object_name.clone()

Note:

- If we want to apply clone() method on any object, then the class which we got an object must implement Clonable interface
- If not implemented then we will get CloneNotSupportedException.
- Clonable is a marker interface(marker interface means interface without any properties).
- Other way creating an exact copy is using new operator.

```
class Student implements Cloneable{

int rollno;

String name;

 Student(int rollno,String name){

this.rollno=rollno;

this.name=name;

}

public static void main(String args[]){

try{

Student stud1=new Student(101,"abc");


Student stud2=(Student)stud1.clone();


System.out.println(stud1.rollno+" "+stud1.name);

System.out.println(stud2.rollno+" "+stud2.name);


}catch(CloneNotSupportedException c){}
```

SSIT COMPUTER EDUCATION

```
}

}
```

5. toString()
   a. returns the string representation of this object.

6. Wait()

wait(long timeout)	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
wait(long timeout,intnanos)	causes the current thread to wait for the specified miliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
wait()	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).

7. notify()

public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.

8. finalize()
   a. is invoked by the garbage collector before object is being garbage collected.

SSM COMPUTER EDUCATION

String, String Buffer and String Builder

String:

- It is a collection of characters
- In Java, String is represented in 3 ways:
  - String class
  - StringBuffer class

- StringBuilder class

- Whenever we create a string, it internally creates a character array.
- This class is defined in java.lang package
- String class is immutable means once it was created, the contents can't be changed.

Various constructors in String

- public java.lang.String();
- public java.lang.String(java.lang.String);
- public java.lang.String(char[]);

String constant

These are stored in an area called String constant pool in method area.

String variable="value";

Int [] a = {1,2,3,4,5};

a.length;

String str = new String("a");

Str.isEmpty();

2+""

""+2

Sop(2)===sop(String.valueOf(2));

Methods in String

1. public int length(); ---- returns the length of the string
2. public Boolean isEmpty(); ---- returns true if empty otherwise false

3. valueof()

SSIT COMPUTER EDUCATION

a. public static java.lang.String valueOf(boolean);
b. public static java.lang.String valueOf(char);
c. public static java.lang.String valueOf(int);
d. public static java.lang.String valueOf(long);
e. public static java.lang.Stringv alueOf(float);
f. public static java.lang.String valueOf(double);

4. comparision
   a. comparing of two string can be done in following ways:
      i. equals method
         - public boolean equals(java.lang.Object);
         - this is the method in object class overridden in String class
         - used to compare the string with the specified string.

      ii. equalsIgnoreCase
         - public booleanequalsIgnoreCase(java.lang.String);
         - to compare the string with the specified string with ignore case

      iii. compareTo()
         - public intcompareTo(java.lang.String);
         - compares the strings in a lexicographically.

Return value	
0	If two strings are same
>0	If string1 > string2
<0	If String1 < String2

      iv. compareToIgnoreCase()

      publicintcompareToIgnoreCase(java.lang.String);

         - public intcompareTo(java.lang.String);
         - compares the strings in a lexicographically with ignore case.

SSSIT COMPUTER EDUCATION

Return value	
0	If two strings are same
>0	If string1 > string2
<0	If String1 < String2

v. == operator
   1. Compare the object references or hashcodes.

Note:

Difference between == and equals method.

5. Concatenation of two strings
   a. Adds the specified string at the end of this string.
   b. Can be done in two ways:
      i. Concat method
         publicjava.lang.Stringconcat(java.lang.String);
      ii. + operator
6. Extracting the characters from the string
   a. charAt()

      public char charAt(int);

      used to retrieve the character at specified position

      Example: java program that reads a string and count the number of vowels.

   b. Induce a sub string from the given string
      publicjava.lang.String substring(int);
      publicjava.lang.String substring(int, int);

7. Converting the case

   publicjava.lang.String toLowerCase();

   publicjava.lang.String toUpperCase();

8. Replace strings

   publicjava.lang.String replace(char, char);

SSIT COMPUTER EDUCATION

publicjava.lang.StringreplaceFirst(java.lang.String, java.lang.String);

publicjava.lang.StringreplaceAll(java.lang.String, java.lang.String);

publicjava.lang.String replace(java.lang.CharSequence, java.lang.CharSequence);

9. Split
   Public java.lang.String[] split(java.lang.String, int);
   Public java.lang.String[] split(java.lang.String);

Other methods

1. void getChars(char[], int);
2. public byte[] getBytes();
3. public Boolean contentEquals(java.lang.StringBuffer);
4. public Boolean contentEquals(java.lang.CharSequence);
5. public Boolean startsWith(java.lang.String);
6. public Boolean endsWith(java.lang.String);
7. public int hashCode();
8. public int indexOf(int);
9. public int lastIndexOf(int);
10. public int indexOf(java.lang.String);
11. public int lastIndexOf(java.lang.String);
12. public java.lang.String trim();
13. public java.lang.String toString();
14. public char[] toCharArray();

StringBuffer

- Like a string
- String buffer is mutable(can be modified)

Differences between String and StringBuffer

String	StringBuffer
Immutable	Mutable
Methods in String are not synchronized	Methods in StringBuffer are synchronized
Not threadsafe	ThreadSafe

SSIT COMPUTER EDUCATION

Created in String constant pool(method area)	Created in heap area
Shared	Not shared
Recomonded for single threaded application	Safe for the use of muti threaded application

Constructors in String Buffer

1. StringBuffer(): creates an empty string buffer with the initial capacity of 16.
2. StringBuffer(String str): creates a string buffer with the specified string.
3. StringBuffer(int capacity): creates an empty string buffer with the specified capacity as length.

To add the contents to the existed String Buffer

1. public synchronized java.lang.StringBuffer append(java.lang.Object);
2. public synchronized java.lang.StringBuffer append(java.lang.String);
3. public synchronized java.lang.StringBuffer append(java.lang.StringBuffer);
4. public synchronized java.lang.StringBuffer append(char[]);
5. public synchronized java.lang.StringBuffer append(char[], int, int);
6. public synchronized java.lang.StringBuffer append(boolean);
7. public synchronized java.lang.StringBuffer append(char);
8. public synchronized java.lang.StringBuffer append(int);
9. public synchronized java.lang.StringBuffer append(long);
10. public synchronized java.lang.StringBuffer append(float);
11. public synchronized java.lang.StringBuffer append(double);

To insert the value at the given position(insert)

1. public synchronized java.lang.StringBuffer insert(int, char[], int, int);
2. public synchronized java.lang.StringBuffer insert(int, java.lang.String);
3. public synchronized java.lang.StringBuffer insert(int, char[]);
4. public java.lang.StringBuffer insert(int, boolean);
5. public synchronized java.lang.StringBuffer insert(int, char);
6. public java.lang.StringBuffer insert(int, int);
7. public java.lang.StringBuffer insert(int, long);
8. public java.lang.StringBuffer insert(int, float);
9. public java.lang.StringBuffer insert(int, double);

capacity is used to return the current capacity of the string buffer

SSIT COMPUTER EDUCATION

public synchronized int capacity();

public synchronized StringBuffer replace(intstartIndex, intendIndex, String str):
is used to replace the string from specified startIndex and endIndex.

public synchronized StringBuffer delete(intstartIndex, intendIndex):
is used to delete the string from specified startIndex and endIndex.

public synchronized StringBuffer reverse():
is used to reverse the string.

public void ensureCapacity(intminimumCapacity):
is used to ensure the capacity at least equal to the given minimum.

public char charAt(int index):
is used to return the character at the specified position.

public String substring(intbeginIndex):
is used to return the substring from the specified beginIndex.

public String substring(intbeginIndex, intendIndex):

is used to return the substring from the specified beginIndex and endIndex.

String Builder

- Java StringBuilder class is used to create mutable (modifiable) string.
- The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized.
- It is available since JDK 1.5.

Important Constructors of StringBuilder class

1. StringBuilder(): creates an empty string Builder with the initial capacity of 16.
2. StringBuilder(String str): creates a string Builder with the specified string.
3. StringBuilder(int length): creates an empty string Builder with the specified capacity as length.

Important methods of StringBuilder class

SSIT COMPUTER EDUCATION

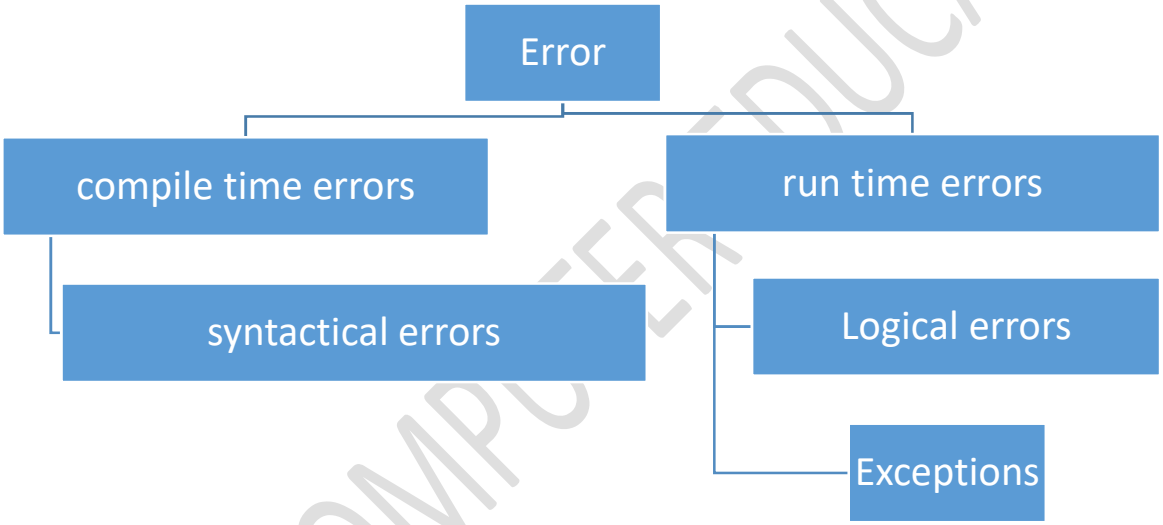
Method	Description
public StringBuilder append(String s)	is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public StringBuilder insert(int offset, String s)	is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public StringBuilder replace(intstartIndex, intendIndex, String str)	is used to replace the string from specified startIndex and endIndex.
public StringBuilder delete(intstartIndex, intendIndex)	is used to delete the string from specified startIndex and endIndex.
public StringBuilder reverse()	is used to reverse the string.
public int capacity()	is used to return the current capacity.
public void ensureCapacity(intminimumCapacity)	is used to ensure the capacity at least equal to the given minimum.
public char charAt(int index)	is used to return the character at the specified position.
public int length()	is used to return the length of the string i.e. total number of characters.
public String substring(intbeginIndex)	is used to return the substring from the specified beginIndex.
public String substring(intbeginIndex, intendIndex)	is used to return the substring from the specified beginIndex and endIndex.

SSIT COMPUTER EDUCATION

SECTION 13: EXCEPTION HANDLING

- What is Exception and itstypes
- How to handle exception?
- Multiple catch blocks
- Finally block

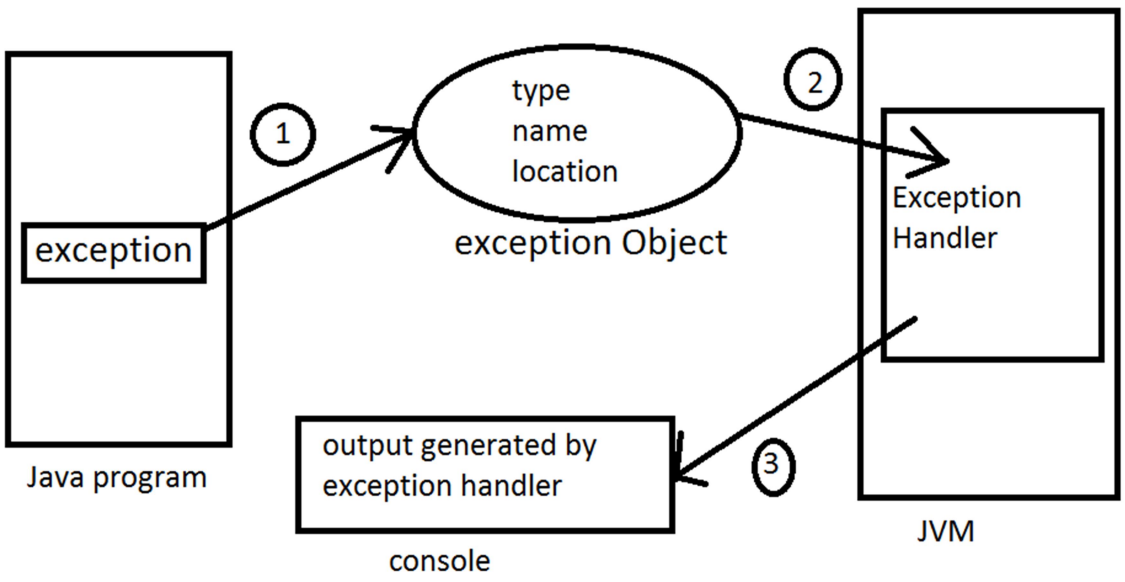
- Difference between throw and throws
- Custom Exception
- Real-time Practicals
    - Write a program to generate the Null Pointer Exception and Number Format Exception. Handle these exceptions with multiple catch blocks.
    - Create the custom exception called “AgeExceedException”. Get the age from user. If age exceeds 35 then throw the AgeExceedException with message “You are not eligible candidate to apply for this position”.



SSIT COMPUTER EDUCATION

What is Exception?

- Any unexpected situation in the program is called Exception.
- Whenever an exception is raised in the program, the program terminates abnormally.
- In order to terminate the program successfully, we have to handle the exception.
- Handling an exception by the developer is called Exception handling.

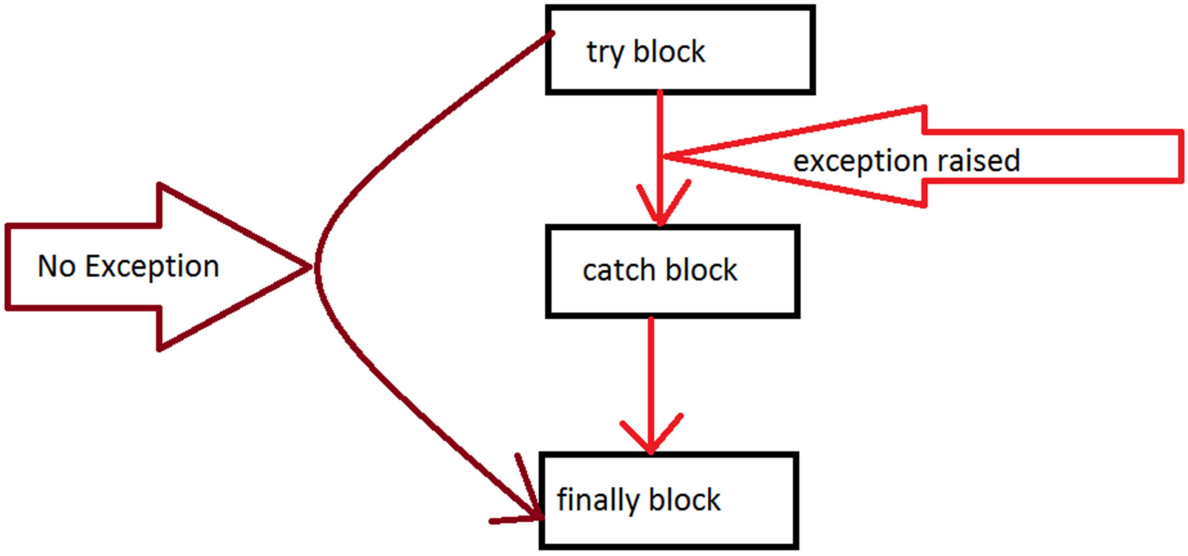


How to Handle Exception in Java?

Java has 5 keywords to handle an exception:

1. try
2. catch
3. finally
4. throw
5. throws

SSIT COMPUTER EDU



what is try?

- Is a block.
- Contains set of statements that may possible to raise an exception.

Syntax:

```
try
{
  // statements that may possible to raise an exception
}
```

What is purpose Catch?

- Like a method form.
- Contains set of statements to be executed if an exception is raised.
- This is called as exception handler.
- Catch is one of the predefined block in which is used to write the block of the statements which provides user friendly error messages by suppressing system error messages.
- An appropriate catch block will be executed an exception occurs in the try block.
- At any point of time only one catch block will executed out of multiple catch blocks.

SSIT COMPUTER EL

- In one catch block one can also write try and catch blocks.

Syntax:

```
catch(<<exception-type>><<exception-ref>>)
{
  // exception handling statements
}
```

What is the purpose Finally block?

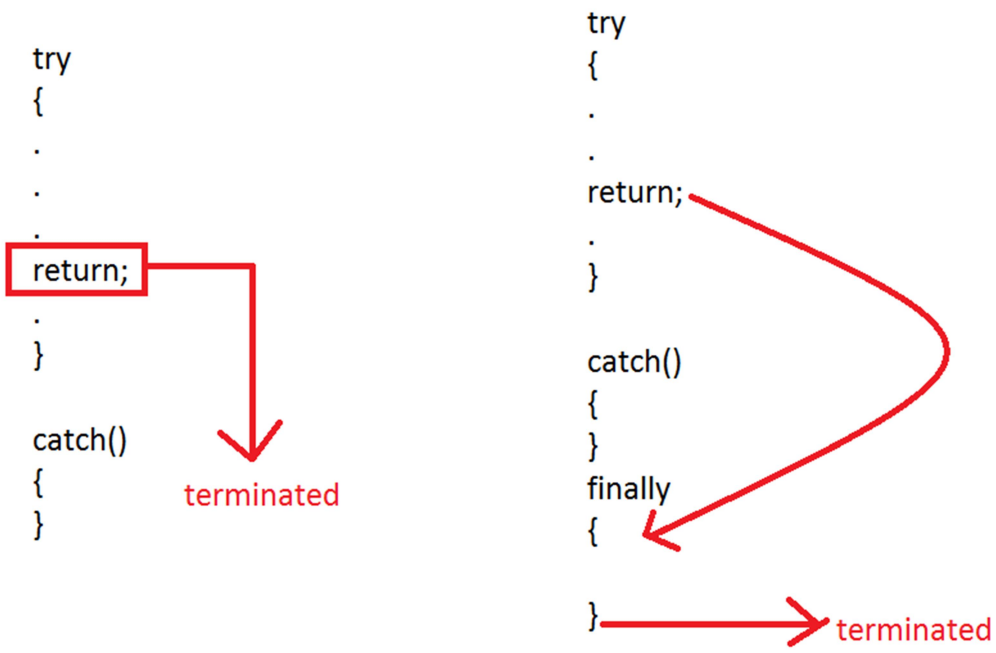
- Contains the set of statements to be executed irrespective of an exception is raised or not.
- It is one of the predefined block in which we write the block of statements which will terminates the resources of files, database which are obtained in the try block. In generally finally block contains **resource relinquishing logic (terminating logic).**
- Finally block will executes compulsorily.
- Writing finally block in java program is optional
- A finally block contain try and catch blocks.
- Per java program it is highly recommended to write one finally block.

Syntax:

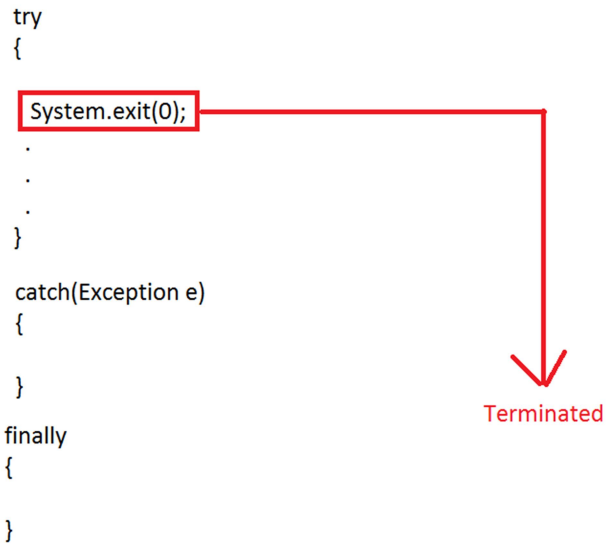
```
finally
{
  // set of statements
}
```

SSSIT COMPUTER EDUCATION

What is the importance of finally block in exception handling?



When finally block won't executes?



Note:

1. A try block must follow either catch block or finally block.(in between no other statements are allowed)

```
try
{
}
catch()
{
}
```

```
try
{
}
finally
{
}
```

```
try
{
}
catch()
{
}
finally
{
}
```

In the first and third combinations, developer is handling exception.

Where as in second scenario, the JVM is handling exception.

In all the three scenario's, the program will be terminated successfully.

What is the difference between final, finally, finalize?

Final	Finally	Finalize
Key word	Block	Method
Used to declare constants	Contains the set of statements to be executed irrespective of an exception is raised or not	Contains the set of statements to be executed before the object is collected by the Garbage collector

What is the significance of finally block in exception handling.

1. Main is a method. So, there is a chance of having return statement.
2. If return statement is present then the program terminates their itself.

SSIT COMPUTER EDUCATION

3. If return statement is present and finally block is defined, then the control transfers from return statement to finally block and executes the entire finally block before terminating the program.

What is the purpose of throw keyword?

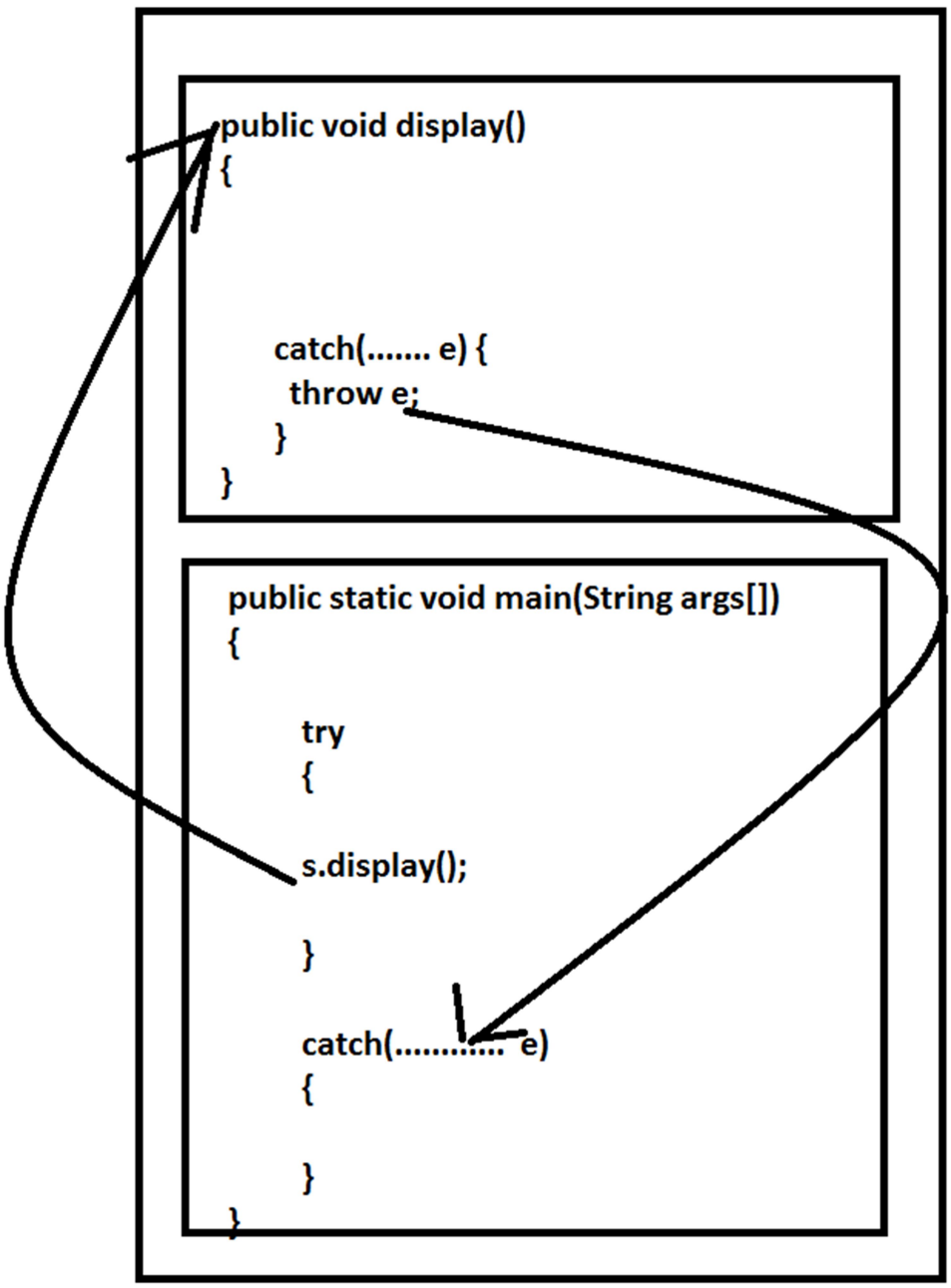
- Is used to pass the exception information from called function to calling function.

Syntax:

throw<<exception-reference-object>>;

SSIT COMPUTER EDUCATION

Sample


public void display()
{
catch(..... e) {
throw e;
}
}
public static void main(String args[])
{
try
{
s.display();
}
catch(..... e)
{
}
}

What is the purpose of throws keyword?

- Is used to declare the exception in method signature.

Syntax:

<<return-type>> <<method-name>>(<<parameters>>) throws <<exception-type(s)>>

{

// Method Defination

}

What is the difference between throw and throws?

Throw	Throws
throw is used to explicitly throw an exception.	throws is used to declare an exception.
throw is followed by an instance.	throws is followed by class.
throw is used within the method.	throws is used with the method signature.
cannot throw multiple exception	can declare multiple exception e.g. public void method()throws IOException,SQLException.

Is it possible to write multiple catch blocks?

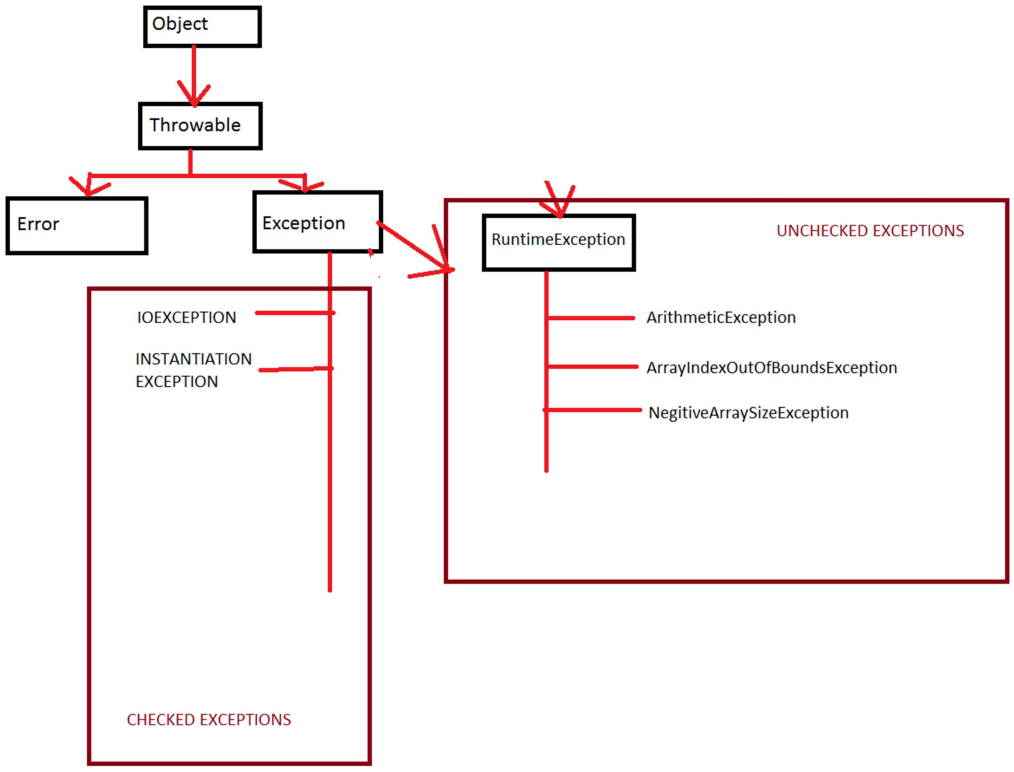
- Yes. We can define multiple catch blocks where each catch block can handle some type of exception.
- If an exception is raised then the control goes to the respective catch block.

How many try or finally blocks we can define in a method?

- In any method, there must be only one try block and one finally. But we can define multiple catch blocks.
- We can define a try block inside another try block or catch block or finally block.

SSSIT COMPUTER EDUCATION

Types of Exceptions



Object class is the super most class for all java classes.

Throwable class

- predefined super class for all exceptions in java
- used for deciding what types of exceptions occurs in the java program
- It is extended to two sub classes:
    - Error ---
        - unexpected situation that can't be handled by the JVM.

- super class for all asynchronous exceptions
- for dealing with hardware and external problems

- Exception ---
  - unexpected situation that can be handled by the JVM.
  - super class for all synchronous exceptions(specially for checked).
  - Exceptions are of two types:
    - Checked exception
    - Un checked exception

What is the difference between checked and un checked exceptions?

Checked exception	Un checked exception
List of classes that are extending from Exception class	List of classes that are extending from Runtime Exception class
All Checked exceptions must be handled by the user otherwise compiler throws Exception at compile time Ex: IOException CloneNotSupportedException ……	Handling the unchecked exceptions is optional to the user. If the user is not handled then it was handled by the JVM if exception raises. Ex: ArithmeticException ArrayIndexOutOfBoundsException… …….

Runtime Exception class

- It is super class for all synchronous unchecked exceptions.

User defined exceptions

How to define User defined exception?

Step 1:

Extend the exception class from either Throwable or Error or Exception or RunnableException

class <<exception-name>> extends Throwable Error or CheckedException
class <<exception-name>> extends Error --→ Error(can't handle by JVM)
class <<exception-name>> extends Exception --→ Checked Exception (can handle by JVM)
class <<exception-name>> extends RuntimeException

SSSIT COMPUTER EDUCATION

--→ Unchecked Exception(can handle by JVM)

Step 2:

Override toString() method or Pass the exception message to super class using super keyword.

```
class <<exception-name>> extends <<exception-type>>
{
      public String toString()
       {
            return <<exception-message>>;
       }
}
```

```
class <<exception-name>> extends <<exception-type>>
{
       <<exception-name>>(<<arguments>>)
      {
          super <<expression>>;
      }
}
```

Note:

- Whenever the user is defined an exception, then the user must provide the information about the situation to throw the defined exception.
- If user defined exception class is extending from either Throwable or Exception, then it is treated as a checked exception.
- If the same class is extending from RuntimeException, then it is treated as unchecked exceptions.

In how many ways we can display the exception information?

We have three ways:

1. System.out.println(<<exception-object-reference>>);
2. <<exception-object-ref>>.getMessage();
3. <<exception-object-ref>>.printStackTrace();

SSSIT COMPUTER EDUCATION

SSIT COMPUTER EDUCATION

SECTION 14: MULTITHREADING

- What is multithreading and advantages.
- What is Thread?
- Life Cycle of a Thread.
- Thread Classes and Methods
- "Runnable" Interface
- Creating Thread
- Working with multiple Threads.
- Java Synchronization
- Interthread Communication
- Dead Lock
- Real-time Praticals
    - Create 3 threads. Each thread has to print Multiplication tables (2 tables, 3 tables, 4 tables) simultaneously.
    - Print 5 Divisible and 2 divisible up to 50 using synchronization.

What are the differences between program and process?

Program	**Process**
1. A program is set of optimized instructions. 2. Program always resides in secondary memory for a long time until we delete. 3. Based on the duration of the existence of a program is treated as static Object.	1. A program is under exception is known as process. 2. Process resides in main memory for limited span of time until execution complete. 3. Based on the duration of the existence of a program is treated as dynamic Object.

What is a process?

- A program under execution is called as process.
- Ex: opening an MS-word
- If multiple programs are running simultaneously, then it is called as multi-processing environment.
- Ex: in order to run a Java program we need to open the following programs:
    - Command prompt
    - Editor like notepad

What is a thread?

- Part of the program is called as Thread.
- Running multiple threads in a program is called as Multi-threading.

SSIT COMPUTER EDUCATION

- Ex: while printing the document, we can do some modifications that won't reflect in print.
- One of the important features of Java that made the language popular is Multi-threading.

As real world developer in any language we may develop two types of applications. They are

1. Process based applications	**Thread based applications**
1. Process based applications always provides single flow of control. 2. All the applications of C,C++, Pascal, COBOL e.t.c., are comes under process based applications. 3. Context switch is more. 4. Because of more context switch process based applications takes more execution time. 5. All the process based applications are treated as heavy weight components. 6. Process based applications provide only sequential execution but not concurrent execution. 7. For each sub program in process based applications there exists a separate address space.	1. Thread based applications always provides and contains multiple flow of control. 2. All the applications of Java, .NET e.t.c., are comes under process based applications. 3. Context switch is less. 4. Because of less context switch process based applications takes more execution time. 5. All the Thread based applications are treated as heavy light weight components. 6. Thread based applications provide both sequential and concurrent execution. 7. In Thread based applications irrespective of number of sub programs there exists a single address space.

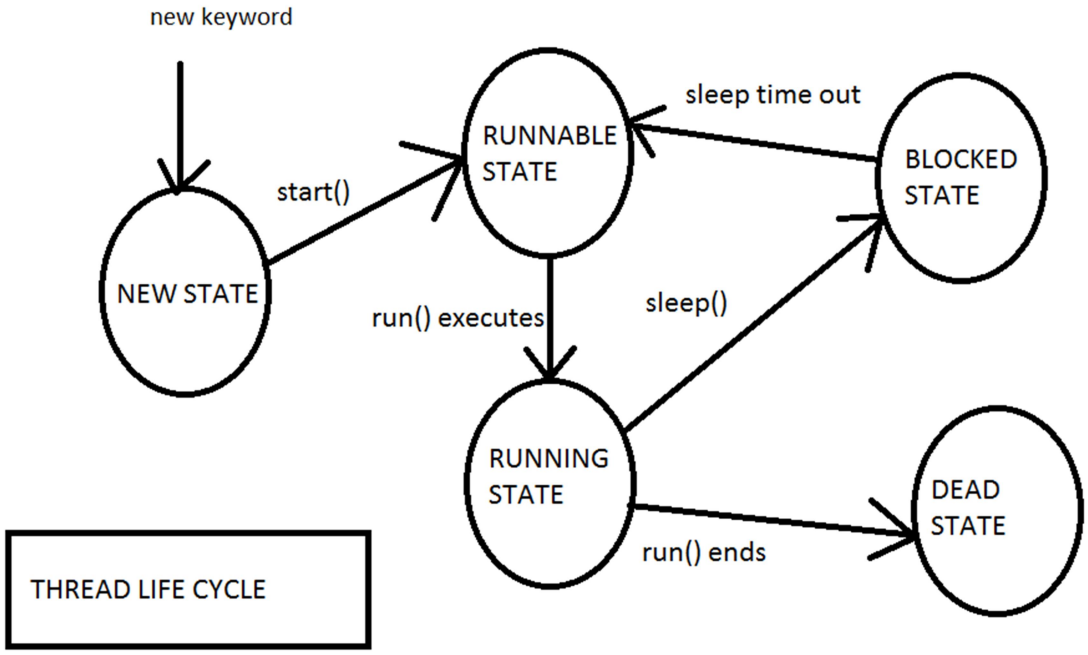
Thred Life Cycle:

Whenever we write a multi threaded program there is possibility of existing many number of Threads. When multi threading program start executing, the threads of that program will undergo five states of Thread. They are

1. New State
2. Ready State or Runnable State
3. Running State
4. Waiting State or Sleep State
5. Halted State or Dead State

Thread State Chart Diagram:

SSIT COMPUTER EDUCATION



New State: A new state is one in which Thread is created and it is about to enter in main memory.

Ready: A Ready state is one in which Thread is entered into the main memory. Memory space is created and first time waiting for CPU.

Running State: A Running State is one which the Thread is under the control of CPU in Java programming Threads of Java executing concurrently.

If Threads are executing hour by hour or minute by minute or second by second that type of Thread execution is known as sequential execution. If threads are executing millisecond by millisecond that type of Thread execution is known as Concurrent execution because no human being calculations may not be able to differentiable of milliseconds of time hence in Java programming all the Threads of our program executed by the CPU concurrently by following Round robin algorithm with the difference of milliseconds time.

Waiting state: A state of the Thread is said to be waiting state if and only if it satisfies the following factors.

a) Remaining **CPU burst time** (The amount of time required by the Thread from the CPU for its complete execution is known as CPU burst time).
b) Suspending the currently executing the Thread.
c) Making the currently executing Thread to sleep for a period of time in terms milliseconds.
d) Making the currently executing Thread to wait for a period of time in terms of milliseconds.
e) Making the currently executing Thread to wait without specifying the period of time.

SSIT COMPUTER LC

f) Joining the currently executing Threads after completion of execution.

Halted State: A Halted State is one in which Thread has completed its execution in general as long as the Thread present in ready, running and waiting states whose execution state is true and these states are known as **in memory states.** As long as Threads are present in NEW and HALTED States whose execution status is false and these states are known as **out memory states.**

In How many ways we can create a Thread?

In Java, we can create a thread in two ways:

1. Extends Thread class
2. Implements Runnable interface

How to define a Thread class using Thread Class?

Step 1: extends from Thread class

Step 2: override public void run() method.

How to execute the thread which is extending thread class?

Step 1: create an object of the class which is defined in the above algorithm

Step 2: call start method using thread class object reference.

```
class MyThread extends Thread
{
        public void run()
        {
                for(int i=1;i<=5;i++)
                {
                        System.out.println(i);
                        try
                        {
                                sleep(1000);
                        }
                        catch (InterruptedException e)
                        {
```

SSIT COMPUTER EDUCATION

```
                        System.out.println(e);
                    }
                }
            }

}

class ExtendsThreadDemo
{
        public static void main(String[] args)
        {
                MyThread mt = new MyThread();
                mt.start();//--->internally calls run() method

        }
}
```

Implements Runnable Interface

How to define a Thread class using runnable interface?

Step 1: implements from Runnable interface

Step 2: override public void run() method.

How to execute the thread which is implementing runnable interface?

Step 1: create an object of the class who is implementing runnable interface class

Step 2: create an object of thread class and pass the above object as an argument.

Step 3: call start method using thread class object reference.

```
public class ServerBackup implements Runnable{

        public void run()
```

SSSIT COMPUTER EDUCATION


```
	{
		for(int i=0;i<5;i++)
		{
			System.out.println("Taking server backup.....");

			try
			{
				Thread.sleep(1000);
			}
			catch(InterruptedException e)
			{
				System.out.println(e);
			}
			System.out.println("Back Up completed....");
		}
	}


}

public class ServerBackupDemo {

	public static void main(String[] args) {

		ServerBackup sb = new ServerBackup();
		Thread t = new Thread(sb);
		t.start();

	}
```

SSJT COMPUTER EDUCATION

}

Some of the methods defined in Thread class

Public final void setName(String)

Public final String getName()

The above methods are used for setting the user friendly name to the Thread and obtaining same thing from the Thread.

EX: Thread t1=new Thread()

String tname=t1.getName();

System.out.println(tname); // Thread 0

T1.setName("Java th");

Tname=t1.getName();

System.out.println(tname);//Java th

Public final void setPriority(int)

Public final int getPriority()

The above methods are used for setting and getting the priority of the Thread.

EX: Thread t1=new Thread();

Int pv=t.getPriority();

System.out.println(pv);

T1.setPriority(Thread.MAX_PRIORITY)

Pv=t1.getPriority();

System.out.println(pv); //100→ MAX_PRIORITY

Public final void start():

This method is used for transferring the Thread from new state to ready state, provides internal services of multithreading and automatically calls run() which is containing the logic of the Thread.

Note: The internal service of the multithreading represents providing concurrency, synchronization, inter thread communication e.t.c.,

1. **Public void run():**

CSSIT COMPUTER EDUCATION

i. It is one of the **non final method** present in Thread class for providing logic of the Thread. Originally run () defined in Thread class with null body method.

ii. As a Java programmer to provide a logic of the Thread, run() must be overridden in the context of our derived class by extending java.lang.Thread class.

iii. This method will be called automatically by start () upon the Thread class Object we apply start (). That is run () can't be called by the Java programmer directly because that particular Thread never get the services of multithreading.

While we are overriding run() of the Thread class either we write block of statements which provides the logic of the Thread or we call user defined method which contains the logic of the Thread.

```
public class ToStringDemo {


    public static void main(String[] args) {
        ServerBackup sb = new ServerBackup();
        Thread t = new Thread(sb);


        System.out.println("Result is:" + t);


        /*


        Output: Thread[Thread-0,5,main]
        Thread-0: default name assigned to the thread by JVM
              to retrieve the thread Name ----> getName()
                    to set the new name ---> setName()


        5 : priority Number.
          every thread contains some number called priority Number
            that explains the execution sequence.
```

SSSIT COMPUTER EDUCATION

0-4 ---> MIN_PRIORITY

5 ---> NORM_PRIORITY

6-10 ---> MAX_PRIORITY

default priority of any thread is 5.

to retrieve the priority ---> getPriority()

to set the new priority ---> setPriority()

main: Name of the thread group.

ThreadGroup tg = new ThreadGroup();

Thread t = new Thread(tg);

t is the member of tg group.

getThreadGroup().getName() used to retrieve the thread group name.

```
*/
t.setName("Server backup Thread");
t.setPriority(10);

System.out.println("Result is" + t);
System.out.println("Thread Name is" + t.getName());
System.out.println("Thread Priority is:" + t.getPriority());
System.out.println("Thread group Name is:" +
                                t.getThreadGroup().getName());

}
```

SSSIT COMPUTER EDUCATION


```
}
```

Public boolean isAlive(): This method determines the execution states of the Thread. This method returns true provided Thread is present in ready, running and waiting state. This method returns false when the Thread is in NEW and HALTED states.

```
class isAliveDemo
{
	public static void main(String[] args)
	{
		MyThread mt = new MyThread();
		System.out.println("Is Alive.....:" + mt.isAlive());
		mt.start();
		System.out.println("Is Alive.....:" + mt.isAlive());
		try
		{
			mt.join();
		}
		catch (InterruptedException e)
		{
			System.out.println(e);
		}

		System.out.println("Is Alive.....:" + mt.isAlive());

	}
}
```

public final void join() throws java.lang.InterruptedException:

This method is used for joining threads which are completed their execution as a single Thread. In another words this method joins the completed Thread group name collects all the combined joined Threads and handover to garbage

SSSIT COMPUTER EDUCATION

Collector this approach improves the performance of multi Threading applications. The default environment of Java says as and when individual Threads are completed their execution individually collected Thread group name and handover to garbage collector which is not a recommended process.

Interrrupted exception occurs, interrupted exception never occurs in single/standalone applications but there is a possibility of occurring in Client/server environment. Lets assume n Threads are stated their execution and n-1 Threads are completed their execution and joined. Still nth Thread is executing due to mis memory management of serverside Operating System, Thread group name is **trying** collect n-1 joined Threads and handovering to garbage collector even through nth Thread is under execution. In this point of time JVM generates a predefined exception called java.lang.Interrupted Excetion with respective nth Thread. Interuppted Exception is one of the CheckedException.

```
public class JoinMethodDemo {


        public static void main(String[] args) {
                ServerBackup bkp = new ServerBackup();
                Thread t = new Thread(bkp);
                t.start();


                ServerBackup bkp1 = new ServerBackup();
                Thread t1 = new Thread(bkp1);
                t1.start();


                try
                {
                        t.join(3000);
                }
                catch(InterruptedException e)
                {
                        System.out.println(e);
                }
```

SSSIT COMPUTER EDUCATION

```
            System.out.println("Bye....Main Thread");

        }

}
```

How to create Multiple Threads?

```
public class NumberGenerator implements Runnable{

        Thread t;
        public NumberGenerator(String tName,int priority) {
                t = new Thread(this);
                t.setPriority(priority);
                t.setName(tName);
                t.start();
        }

        public void run()
        {
                for(int i=1;i<=5;i++)
                {
                        System.out.println(t.getName() + "::" + i);

                        try
                        {
                                Thread.sleep(1000);
                        }

                        catch(InterruptedException e)
```

SSIT COMPUTER EDUCATION

```
                {
                        System.out.println(e);
                }
        }
    }

}

public class NumberGeneratorDemo {

    public static void main(String[] args) {
        NumberGenerator ng = new NumberGenerator("th1",10);
        NumberGenerator ng1 = new NumberGenerator("th2",5);
        NumberGenerator ng2 = new NumberGenerator("th3",1);

    }

}
```

Synchronization Techniques:

1. Synchronization is one of the distinct feature of multithreading to eliminate inconsistent results.
2. The concept of Synchronization must be always applied on common tasks for getting consistent results that is if multiple Threads are performing common task then those Threads generates inconsistent result. To avoid this inconsistent result it is recommended to apply Synchronization concept.

DEF: The mechanism of allowing only one Thread among multiple Threads into the area which is sharable/common to perform read and write operations known as Synchronization.

Need of Synchronization:

SSBT COMPUTER EDUCATION

1. Let us assume variable balance whose initial value is 0. Let us assume there exists two types of Threads t1 and t2 want to update or deposit `10 and if `20 to the balance variable respectively.
2. Both Threads shared execution and completed if verify the value of the balance variable then it contains either 10 or 20. Which is one of the inconsistent result.
3. To avoid this inconsistent result we must apply the concept of **Synchronization.**

Synchronization concept applied:

1. Threads t1 and t2 started their execution (Assume t1 starts first and t2 starts later with the different of milliseconds of time). Thread t1 gets the value of balance(0) and balance variable value will be locked by JVM until t1 completes its execution.
2. Maintenance if Thread t2 trying to access the balance variable value then the JVM made the Thread t2 to wait until t1 completes its execution.
3. Once the Thread t1 completes its execution balance variable value(10) will be unlocked and given to t2 and once again balance variable value will be locked.
4. Thread t2 completed its execution, balance variable value(30) will be unlocked. Finally it we observer the value of the balance then it contains consistent result(30).
5. Hence during the Synchronization concept the process of locking and unlocking will be performed by the JVM until all Threads completed their execution.
6. Thread synchronization techniques are divided into two types they are

i. Synchronization Methods
ii. Synchronized blocks.

i. **Synchronized Methods:** If any ordinary method is accessed by more than one Thread then the ordinary method generates an inconsistent result. To avoid this inconsistent result the definition of ordinary method must be preceded by a keyword synchronized for making the method as Synchronized. Based on Synchronized keyword we write before the methods, they are classified into two types they are

a) Synchronized instance methods.
b) Synchronized static methods.

a) Synchronized instance methods: If an Ordinary instance method is accessed by multiple Threads then there is possibility of getting inconsistent result. To avoid this inconsistent result, the ordinary method must be made as synchronized by using synchronized keyword.

Syn:

Synchronized ReturntypeMethodName(list of formal params if any)

{

SOFTJUMPUFFEDUCATION

Block of Statement(s);

}

If we make an ordinary instance method as synchronized then the **object** of corresponding class will be locked by JVM.

```
//package SynchronizedKeyword;


public class ATMCounter {
	/* synchronized keyword or synchronized method*/
	public  synchronized void withdrawl(String name)
	{
		System.out.println(name + "entered inside ATM Counter");
		System.out.println(name + "is counting money");
		try
		{
			Thread.sleep(1000);
		}
		catch(InterruptedException e)
		{
			System.out.println(e);
		}
		System.out.println(name + "is leaving ATM Counter");
	}
}

/*

synchronized <<return-type>> <<method-name>>(Parameters)
{
	.
	.
```

SSSIT COMPUTER EDUCATION


```
}
*/
//package SynchronizedKeyword;

public class Customer implements Runnable{
	Thread t;
	ATMCounter atm;

	Customer(String nm,ATMCounter at)
	{
		t = new Thread(this,nm);
		atm = at;
		t.start();
	}
	public void run()
	{
		atm.withdrawl(t.getName());
	}
}
```

```
//package SynchronizedKeyword;

public class MainMethod1 {

	public static void main(String[] args) {
		ATMCounter atm = new ATMCounter();
```

SSSIT COMPUTER EDUCATION

```
        Customer c1 = new Customer("customer1",atm);
        Customer c2 = new Customer("customer2",atm);
        Customer c3 = new Customer("customer3",atm);
        Customer c4 = new Customer("customer4",atm);
    }

}
```

Synchronized static methods: If an ordinary static method accessed by multiple Threads then there is possibility of getting in consistent result. To avoid this inconsistent result the ordinary static method definition must be made as synchronized by using synchronized keyword.

Syn:

Synchronized static method name(list of formal params if any)

{

Block of statement(s);

}

If we make any ordinary static method as synchronized then corresponding class will be locked.

```
//package SynchronizedKeyword;


public class ATMCounter {
    /* synchronized keyword or synchronized method*/
    public static synchronized void withdrawl(String name)
    {
        System.out.println(name + "entered inside ATM Counter");
        System.out.println(name + "is counting money");
        try
        {
            Thread.sleep(1000);
        }
```

SSIT COMPUTER EDUCATION

```
            catch(InterruptedException e)
            {
                System.out.println(e);
            }
            System.out.println(name + "is leaving ATM Counter");
        }
}

/*

synchronized  static <<return-type>> <<method-name>>(Parameters)
{
        .
        .
}
*/
//package SynchronizedKeyword;

public class Customer implements Runnable{
        Thread t;
        ATMCounter atm;

        Customer(String nm,ATMCounter at)
        {
            t = new Thread(this,nm);
            atm = at;
            t.start();
        }
        public void run()
```

SSIT COMPUTER EDUCATION

```
	{
		atm.withdrawl(t.getName());
	}
}
//package SynchronizedKeyword;

public class MainMethod1 {

	public static void main(String[] args) {
		/* Class level locking demo */
		ATMCounter atm = new ATMCounter();
		ATMCounter atm1 = new ATMCounter();
		ATMCounter atm2 = new ATMCounter();
		ATMCounter atm3 = new ATMCounter();
		Customer c1 = new Customer("customer1",atm);
		Customer c2 = new Customer("customer2",atm1);
		Customer c3 = new Customer("customer3",atm2);
		Customer c4 = new Customer("customer4",atm3);

	}

}
```

Synchronized blocks:

Synchronized blocks are an alternative synchronization for achieving consistent results instead of using synchronized methods. Synchronized blocks must be always return ordinary instance methods and ordinary instance methods inherited non static methods for achieving the consistent result.

Synchronized(object of current class)

{

SSSIT COMPUTER EDUCATION

Block of statement(S);

}

Synchronized blocks always locks object of the class but not class because we override instance methods only but not recommended to static methods.

Need of Synchronized blocks: If a derived class inherits any method from an interface and if the inherited ordinary instance method is accessed by multiple Threads with respect to the 1derived class Object then the inherited ordinary instance method generates inconsistent result.

To avoid this inconsistent result, as a derived class programmer, we may attempt to write synchronized keyword before ordinary inherited instance method. This attempting is not possible because the derived class programmer does not have any privileges to change the prototype because interface methods. We are unable to solve inconsistent problem with concept of Synchronized methods. Hence to solve the problem of Synchronized methods and to get consistent result we have a concept synchronized blocks.

```
//package SynchronizedBlockDemo;


public class Library {


	public void readingBook(String nm)
	{
		System.out.println(nm + "entered inside library");


		/* Synchronized block */
		synchronized(this)
		{
			System.out.println(nm + "reading the book");
			try
			{
				Thread.sleep(1000);
			}
			catch(InterruptedException e)
```

SSSIT COMPUTER EDUCATION


```
                {
                        System.out.println(e);
                }
                System.out.println(nm + "is leaving the library");
        }


    }

}



/*

/*

<<return-type>> <<method-name>>(Parameters)
{

    synchrnoized(this)
    {

  }
    .
    .
}
*/
```

SSIT COMPUTER EDUCATION


```
//package SynchronizedBlockDemo;

public class Reader implements Runnable {
	Library lib;
	Thread t;

	Reader(String nm,Library lib)
	{
		t = new Thread(this,nm);
		this.lib = lib;
		t.start();
	}

	public void run()
	{
		lib.readingBook(t.getName());
	}
}

//package SynchronizedBlockDemo;

public class MainDemo {

	public static void main(String[] args) {
		Library lib = new Library();
		Reader r1 = new Reader("reader1",lib);
		Reader r2 = new Reader("reader2",lib);
		Reader r3 = new Reader("reader3",lib);
	}
```

SSSIT COMPUTER EDUCATION

}

Inter Thread Communication: ITC of Java is one of the specialized concept of inter process communication of Operating System.

1. ITC applications of Java are the fastest applications in the real industry compare to any of the applications in any language.
2. The real time implementations of ITC are

a) Implementation of real world server software by third party vendors.
b) Implementation of universal protocols such as http, FTP, SMTP e.t.c.,

Def: The mechanism of exchanging the data/information among multiple Threads is known as Inter Thread Communication (OR)

If the output of first Thread is given as input to second Thread, the output of second Thread is given to the input to Third Thread e.t.c., then the communication between first, second and third Threads are known as Inter Thread Communication. In order to develop Inter Thread Communication based applications we use the methods of java.lang.Object class. These methods are known as Inter Thread Communication.

Methods in java.lang.Object:

1. **public void wait(long milliseconds) throws interruptedException:** This method is used for making the Thread to wait for a period of time in terms of milliseconds. If the wailting time completed, automatically the Thread will be entered into Ready state from waiting state. This method is not recommended to use by Java programmer for making the Thread to wait for an amount of time because Java programmer can't decide the CPU burst time of previous Thread to wait.
2. **public void wait():** This method is used for making the Thread to wait without specifying any time.
3. **public void notify():** This method is used for transferring one Thread at a time from waiting state to Ready state.
4. **public void notifyAll():** This method is used for transferring all the Threads into the form waiting state. The classical example of itc are

i) Producer – Consumer problems
ii) Dining – Philosophers problem
iii) Readers – writers problem
iv) Barbershop Problem

```
public class BankAccount {


    int money;

    boolean flag=false;
```

SSITCOMPUTER EDUCATION


```
public synchronized void deposit(String name)
{
        if(flag)
        {

        System.out.println(name + "is going to waiting state");
                try {
                        wait();

                } catch (InterruptedException e) {

                        e.printStackTrace();
                }
        }

        money=1;
        flag=true;
        System.out.println(name + "is deposited Money");
        notify();

}
public synchronized void withdraw(String name)
{
        if(!flag)
                try {
                        System.out.println(name + "is going to waiting state");
                        wait();
                } catch (InterruptedException e) {
```

SSIT COMPUTER EDUCATION

```
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }

                money=0;
                flag=false;
                System.out.println(name + "is withdrawn Money");
                notify();
        }

}
//package ITC;

public class Reciever implements Runnable {
        BankAccount account;
        Thread t;

        Reciever(String nm,BankAccount acc)
        {
                t = new Thread(this,nm);
                t.start();
                account = acc;
        }

        public void run()
        {
                for(int i=1;i<=5;i++)
                        account.withdraw(t.getName());
        }
```

SSIT COMPUTER EDUCATION

```
}
//package ITC;

public class Sender implements Runnable{

	BankAccount account;
	Thread t;
	Sender(String nm,BankAccount acc)
	{
		t = new Thread(this,nm);
		t.start();
		account = acc;
	}

	public void run()
	{
		for(int i=1;i<=5;i++)
			account.deposit(t.getName());
	}
}
class BankCustomerITC
{
	public static void main(String[] args)
	{
		BankAccount acc = new BankAccount();
		Sender s = new Sender("Sender",acc);
		Reciever r = new Reciever("Reciever",acc);
```

SSIT COMPUTER EDUCATION


```
        }
}
```

SSIT COMPUTER EDUCATION

SECTION 15: UTIL PACKAGE

- Regular Expression
- String Tokenizer
- Date Classes
- Real-time Praticals
    - Get the String “Java *programming* language” using the Scanner Class.
    - Split the String by using delimiter(*) and display the tokens and count the number of tokens using String Tokenizer.
    - Get the String “Java is my favorite language. Java is my profession” from user. Search and display the text java and index position of java using regular expression.

SSIT COMPUTER EDUCATION

SECTION 16: UTIL PACKAGE – COLLECTIONS FRAMEWORK

- Java Collections Introduction
- The "Collection" Interface
- List, Set and Map Interfaces
- Cursors in java and its differences
- Difference between the implementation classes of List, Set and Map
- Real-time Praticals
    - Remove duplicate words and characters from String using Collections.
    - Get the employee names from the database and store the employee
    - names in List. Display the names in ascending order and descending order using Listiterator.

- What is an Array
    - An Array is collection of homogenous Elements.
    - Every element in an array is identified with an Index or Position
- Limitations of Array
    - Array is Fixed in Size.
    - Homogenous Elements.
    - No Underlying Data structure.
- What is Collection
    - Group of Individual objects as a single Entity.
- What is Collection Framework
    - A collections framework is a unified architecture for representing and manipulating collections.
    - All collections frameworks contain the following:
        - **Interfaces:**
            - These are abstract data types that represent collections.

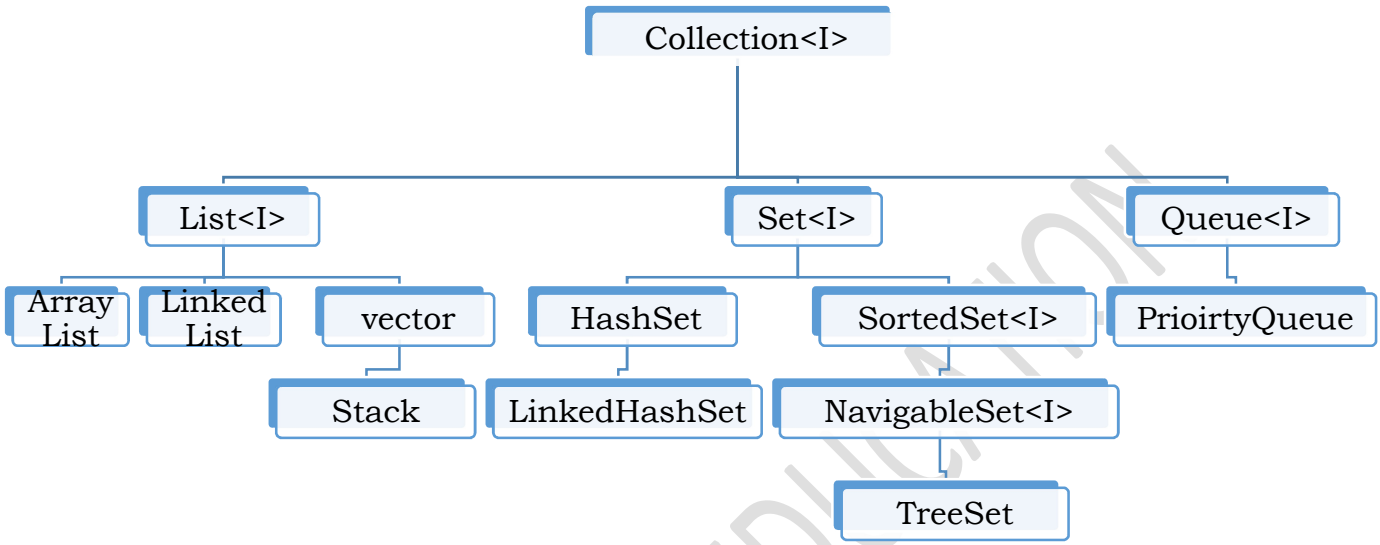
SSSIT COMPUTER EDUCATION

- Interfaces allow collections to be manipulated independently of the details of their representation.
- In object-oriented languages, interfaces generally form a hierarchy.

- **Implementations i.e. Classes:**
  - These are the concrete implementations of the collection interfaces.
  - In essence, they are reusable data structures.
- **Algorithms:**
  - These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces.
  - The algorithms are said to be polymorphic: that is, the same method can be used on many different implementations of the appropriate collection interface.

How the Collection Hierarchy is designed?

SSIT COMPUTER EDUCATION



What are the interfaces involved in Collection framework?

The following are the NINE interfaces of Collection Frame work

1. Collection
2. List
3. Set
4. Sortedset
5. Navigableset
6. Queue
7. Map
8. Sortedmap
9. Navigable Map

Methods in Collection Interface

- Boolean add(object)

SSIT COMPUTER EDUCATION

- Boolean addAll(Collection)
- Void clear()
- Boolean contains(Object)
- Boolean containsAll(collection)
- Boolean Equals(object)
- Boolean isEmpty()
- Boolean remove(Object)
- Boolean removeAll(collection)
- Object[] toArray()
- Iterator iterator()
- List Interface
- Methods in List Inteface
- Void add(index,obj)
- boolean addAll(Index,Collection)
- Object get(index);
- Int indexOf(Object);
- Int LastIndex(Object)
- ListIterator ListIterator();
- ListIterator ListIterator(index)
- Object remove(Index)
- Object set(Index,Object);
- List subList(start-index,end-index);

Array List - Class

- Underlying Data structure is Resizable or Growable array.
- Insertion order is preserved

SSIT COMPUTER EDUCATION

- NULL values are allowed.
- Hetrogeneous Objects are allowed.
- Duplicates are allowed.
- Constructor
- **ArrayList**()
  - Constructs an empty list with an initial capacity of ten.
  - Once it was reached to the maximum capacity then it uses following formula:
  - **ArrayList**(int initialCapacity)
  - Constructs an empty list with the specified initial capacity.
  - **ArrayList**(Collection c)
  - Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Vector - Class

- Underlying Data structure is Resizable or Growable array.
- Insertion order is preserved
- NULL values are allowed.
- Hetrogeneous Objects are allowed.
- Duplicates are allowed.
- Every method in Vector is Synchronized. So, it is ThreadSafe.
- **Vector**() Constructs an empty vector so that its internal data array has size 10 and its standard capacity increment is zero.
- **Vector**(Collection c) Constructs a vector containing the elements of the specified collection, in the order they are returned by the collection's iterator.

SSSIT COMPUTER EDUCATION

- **Vector**(int initialCapacity) Constructs an empty vector with the specified initial capacity and with its capacity increment equal to zero.
- **Vector**(int initialCapacity, int capacityIncrement) Constructs an empty vector with the specified initial capacity and capacity increment.

Difference between vectors and arraylist

Array List	Vector
ArrayList increases by half of its size when its size is increased.	Vector doubles the size of its array when its size is increased.
Not Thread Safe	Thread Safe
Non Legacy in JDK1.2 Version	Legacy Collection from JDK1.0
ArrayList is way faster than Vector.	Since Vector is synchronized and thread-safe it pays price of synchronization which makes it little slow.

Stack

- It is a sub-class of Vector
- Follows LIFO mechanism
- Defined as:
  - public class Stack<E> extends Vector<E>
- **Constructor is:**
  - **Stack**() Creates an empty Stack.
- Stack Methods
- boolean **empty**()
  - Tests if this stack is empty.
- Object **peek**()
  - Looks at the object at the top of this stack without removing it from the stack.

SSIT COMPUTER EDUCATION

- Object **pop**()
  - Removes the object at the top of this stack and returns that object as the value of this function.
- Object **push**(E item)
  - Pushes an item onto the top of this stack.
- int **search**(Object o)
  - Returns the 1-based position where an object is on this stack.

Linked List - Class

- Underlying Data structure is DOUBLE LINKED LIST.
- Insertion order is preserved
- NULL values are allowed.
- Heterogeneous Objects are allowed.
- Duplicates are allowed.
- Preferable when the Insert and delete operations are in the middle.
- Constructor and Description
  - **LinkedList**() Constructs an empty list.
  - **LinkedList**(Collection c) Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
- Methods in Linked List
  - Void addFirst(Object o)
  - Void addLast(Object o)
  - Object removeFirst();
  - Object removeLast();
  - Object getFirst();
  - Object getLast();

SSSIT COMPUTER EDUCATION

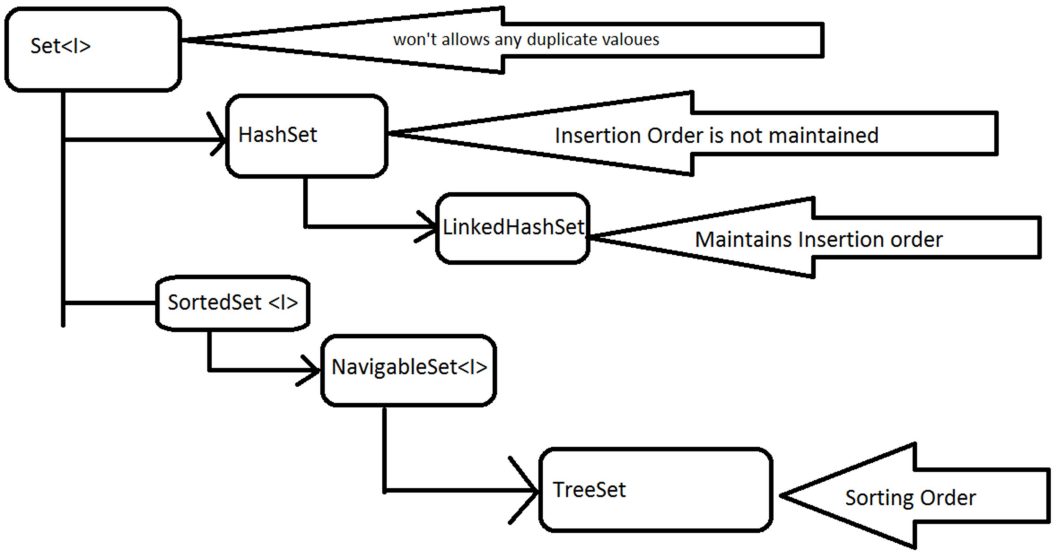
- Make synchronized methods
- Under list Interface,
  - the classes LinkedList and ArrayList are not synchronized, to make them Synchronized use
  - public static List synchronizedList(List);
  - For example:
    - ArrayList al = new ArrayList();
    - List l1 = Collections.synchronizedList(al);
    - LinkedList ll = new LinkedList();
    - List l2 = Collections.synchronizedList(l2);
- ArrayList and Vector implements Random Access interface but LinkedList won't implements Random access interface
- All three classes implements:
  - Serializable
  - Clonable
-

Note:

- Collection framework can be used to Store and transfer the objects.
- To do this requirement, every collection framework implements two interfaces:
  - Serializable
  - Clonable

Set Interface

SSSIT COMPUTER EDUCATION

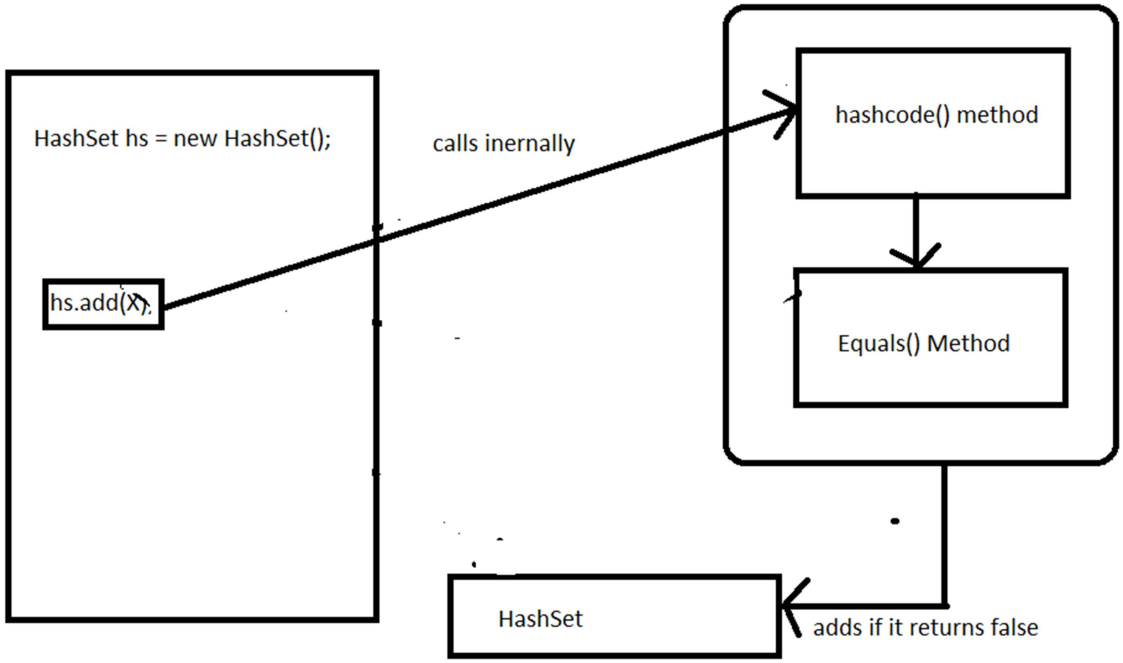


HASH SET

- CLASS IMPLEMENTED IN 1.2
- FOLLOWS HASH DATA STRUCTURE
- constructors
- **HashSet**()
  - Constructs a new, empty set; the backing HashMap instance has default initial capacity (16) and load factor (0.75).
  - **HashSet**(Collection<? extends E> c)
  - Constructs a new set containing the elements in the specified collection.
  - **HashSet**(int initialCapacity)
  - Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and default load factor (0.75).
  - **HashSet**(int initialCapacity, float loadFactor)

SSSIT COMPUTER

– Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and the specified load factor.



LinkedHashSet

- Child class of HashSet
- Introduced in 1.4
- Insertion order is preserved
- Hashtable and doublelinkedlist are the data structures
- LINKED HASHSET IS THE BEST APPLICATION FOR CACHE RELATED APPLICATIONS WHERE DUPLICATES ARE NOT ALLOWED.

DIFFERENCES BETWEEN HASHSET AND LINKED HASHSET

HASHSET	LINKEDHASHSET
UNDERLYING DATA STRUCTURE IS HASH TABLE	UNDERLYING DATA STRUCTURE IS HASH TABLE AND LINKED LIST
INSERTION ORDER IS NOT PRESERVED	INSERTION ORDER IS PRESERVED
INTRODUCED IN JAVA 1.2	INTRODUCED IN JAVA 1.4

SSIT COMPUTER

SORTED SET

- IT IS AN INTERFACE
- IMPLEMENTED IN JAVA 1.2
- FOLLOWS DEFAULT ORDERING OR CUSTOMIZING ORDERING.
- TREE SET
- UNDERLYING DATA STRUCTURE IS BALANCED TREE
- DUPLICATES ARE NOT ALLOWED.
  - IF TRYING TO STORE THEN ADD METHOD RETURNS FALSE.
- HETROGENEOUS ELEMENTS ARE NOT ALLOWED WHEN NATURAL ORDERING IS FOLLOWING IF THE ORDER IS CUSTOMIZED THEN WE CAN PROVIDE HETROGENOUS.
  - IF TRIES TO STORE RAISES CLASS CAST EXCEPTION
- INSERTION ORDER IS NOT PRESERVED.
- Based on sorting order
- Null is allowed only once.
  - IF NULL IS TRYING TO STORE IN NON-EMPTY SET THEN NULL POINTER EXCEPTION IS ARAISED
- Constructor and Description
- **TreeSet**() Constructs a new, empty tree set, sorted according to the natural ordering of its elements.
- **TreeSet**(Collection<? extends E> c) Constructs a new tree set containing the elements in the specified collection, sorted according to the *natural ordering* of its elements.
- **TreeSet**(Comparator<? super E> comparator) Constructs a new, empty tree set, sorted according to the specified comparator.
- **TreeSet**(SortedSet<E> s) Constructs a new tree set containing the same elements and using the same ordering as the specified sorted set.
- NATURAL ORDERING

SSIT COMPUTER EDUCATION

- JVM INTERNALLY CALLS

  CompareTo(object1 , Object2)

which is defined in Comparable class.

- Comparable
- Defined in **java.lang** package
- One method called **compareTo()**
- Used to sort the keys using natural or default ordering.
- All Wrapper class and string class are implemented by Comparable Interface.
- Drawbacks of Comparable
- Can be applied to classes that are implemnting **Comparable** Interface.
- Follows natural or default ordering.
- Comparator Interface
- Defined in java.util package.
- Used for customized sorting order.
- Contains two Methods:
  - public int compare(Object o1, Object o2)
  - public boolean equals()
- In order to define compare Method, the class should implement Comparator Interface.
- Implementation of equals() method in this interface is optional. Because it is already in object class.
- examples
- Class MyComparator implements Comparator
- {
- public int compare(Object o1,Object o2)

SSIT COMPUTER EDUCATION


```
{
    Integer i1 = (Integer)o1;
    Integer i2 = (Integer)o2;
 .
.
.
}
return i1.compareTo(i2); // ascending  order
return - i1.compareTo(i2); // descending order
return i2.compareTo(i1); // descending order
return -i2.compareTo(i1); // Ascending order
return -1   // reverse of the Insertion order
return +1 // insertion order is preserved
return 0 // first key is stored.
```

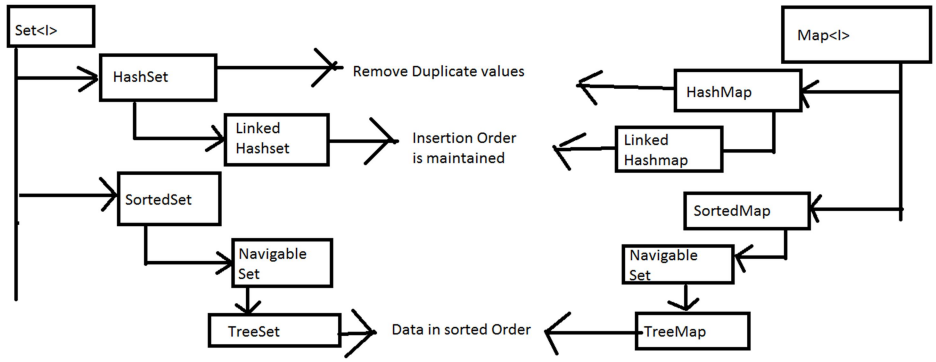
- NAVIGABLE SET
- It is an Interface
- Implemented in Java 1.6
- Child interface of Sorted set.
- Set s = Collections.synchronizedSet(new HashSet(...));
- Set s = Collections.synchronizedSet(new LinkedHashSet(...));
- SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));
- Queue Interface
- public interface Queue<E> extends Collection<E>
- Follows FIFO mechanism
- Null values are not allowed. If given, raises NullPointerException

SSIT COMPUTER EDUCATION

- Methods of Queue Interface
- boolean **add**(E e)
  - Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success and throwing an IllegalStateException if no space is currently available.
- boolean **offer**(E e)
  - Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions.
- E **element**()
- Retrieves, but does not remove, the head of this queue. Throws Exception when element is not available
- E **peek**()
  - Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
- E **poll**()
  - Retrieves and removes the head of this queue, or returns null if this queue is empty.
- E **remove**()
  - Retrieves and removes the head of this queue. Throws Exception
- Priority Queue
- An unbounded priority queue based on a priority heap
- It follows Natural Ordering defined at comparator interface or definition that was provided at compile time.
- Null values are not allowed here.
- constructors
- Methods in Dequeue

SSIT COMPUTER EDUCATION

Map Interface



Various methods in Map Interface

- If we want to represent group of objects in a key – value pair, then we must go to MAP interface.
- Duplicate keys are not allowed.
- But values may duplicate.
- Combination of key and its associated value is called as an ENTRY.
- **Commonly used methods of Map interface:**
- **public Object put(object key,Object value):** is used to insert an entry in this map.
- **public void putAll(Map map)**:is used to insert the specifed map in this map.
- **public Object remove(object key):i**s used to delete an entry for the specified key.
- **public Object get(Object key):**is used to return the value for the specified key.
- **public boolean containsKey(Object key)**:is used to search the specified key from this map.

SSM COMPUTER EL

- **public boolean containsValue(Object value)**:is used to search the specified value from this map.
- **public Set keySet()**:returns the Set view containing all the keys.
- **public Set entrySet()**:returns the Set view containing all the keys and values

HASHMAP

- DATASTRUCTURE IS HASH TABLE
- DUPLICATE KEYS ARE NOTALLOWED
- INSERTION ORDER IS NOT PRESERVED AND VALUES ARE STORED BASED ON HASH CODE OF THE KEY
- NULL KEY IS ALLOWED ONLY ONCE AND NULL VALUES ARE NOT ALLOWED.
- HETROENEOUS KEYS AD VALUES ARE ALLOWED
- HASH MAP METHODS ARE NOT SYNCHRONIZED
- NULL IS ALLOWED FOR KEY AS WELL AS VALUE
- Introduced in java 1.2
- Map m = Collections.synchronizedMap(new HashMap(...));

Linked HashMap

- Child interface of HASHMAP
- HASH MAP WON'T PRESERVES THE ORDER OF INSERTION
- INTRODUCED IN 1.2
- LINKED HASHMAP PRESERVES THE ORDER OF INSERTION
- INTRODUCED IN 1.4

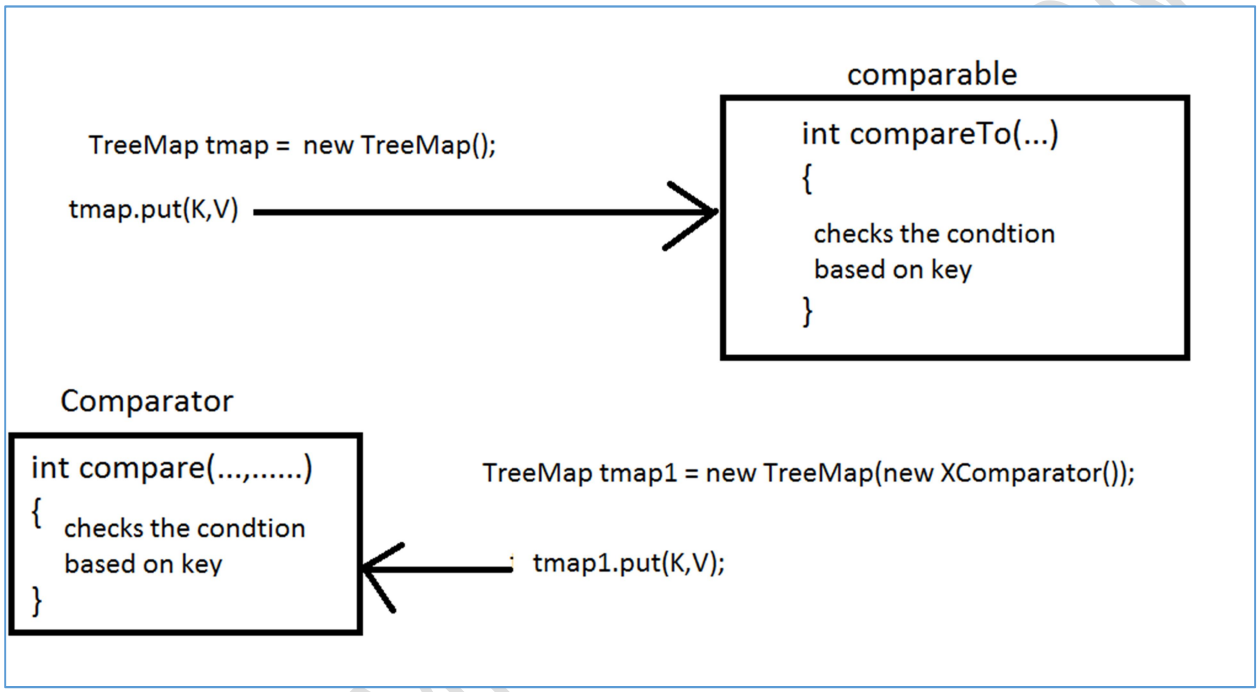
SortedMap

NAVIGABLE MAP

SSSIT COMPUTER EDUCATION

- It is an Interface
  - ALL THE METHODS ARE USED FOR NAVIGATION PURPOSE
- Implemented in Java 1.6
- Child interface of Sorted Map.

TreeMap



CURSORS IN JAVA

1. ENUMERATION --- 1.1
2. ITERATOR -- 1.2
3. LIST ITERATOR -- 1.2

ENUMERATION

- APPLICABLE ONLY FOR LEGACY CLASSES.
- EXAMPLE:
  - VECTOR
  - STACK

- LIMITATIONS OF ENUMERATION
  - Can be applied for Legacy Methods. i.e. not a Universal Cursor.
  - Read only operations are only possible.

ITERATOR

- IT IS A UNIVERSAL CURSOR.
- ALLOWS DELETE OPERATION WHILE ITERATING.
- LIMITATIONS OF ITERATOR
  - UNI DIRECTIONAL I.E. FORWARD DIRECTION
  - ONLY READ AND DELETIONS ARE POSSIBLE BUT ADDITION AND MODIFICATION IS NOT POSSIBLE.

LIST ITERATOR

- CHILD INTERFACE OF ITERATOR.
- IT IS BIDIRECTIONAL.
- ALLOWS ADDITION ,MODIFICATION AND DELETE OPERATIONS WHILE ITERATING.
- LEGACY MEMBERS IN JAVA

Legacy Members of Java

1. ENUMERATON -- INTERFACE
2. DICTIONARY -- ABSTRACT CLASS
3. VECTOR -- CLASS
4. STACK -- CLASS
5. HASHTABLE -- CLASS
   - HASH TABLE METHODS ARE SYNCHRONIZED
   - NULL NOT ALLOWED IN KEY AS WELL AS VALUE
   - Legacy Member of Java
     - ie. Introduced in Java 1.1

SSIT COMPUTER EDUCATION

6. PROPERTIES -- CLASS

SSIT COMPUTER EDUCATION

SECTION 17: GENERICS IN COLLECTIONS FRAMEWORK

- Generics in Java
- Types of Generics
- Uses and Limitations of Generics
- Custom objects with Generic collections
- Sorting in Collections
- Real-time Praticals
    - Create the class employee with name, employee_number, phone. And store the employee objects in generic collections.
    - Sort the custom object generic collectionby employee name.

SSIT COMPUTER EDUCATION

SECTION 18: IO PACKAGE

- Stream and its types
- Read, Write and Copy Files
- Serialization
- Marker Interface
- “transient” keyword

SSIT COMPUTER EDUCATION

SECTION 19: JAVA UPDATED FEATURES AD OOPS MISC

- Varargs
- Autoboxing and Autounboxing
- EnumType

Varargs

Autoboxing and Autounboxing

EnumType

SSIT COMPUTER EDUCATION

SECTION 20: LATEST JAVA VERSION FEATURES

- Default Methods and static methods in interface
- Functional Programming and Lambda Expressions
- Predefined Functional Interfaces
  - Predicates
  - Functions
  - Supplier
  - Consumer
- Optional Class
- forEach Method in iterable interface
- Streams

Static Methods in interfaces

- Upto JAVA7 version, interfaces are able to allow only abstract methods.
- From JAVA8 onwards, interfaces allows to define static methods.
- The main intention of introducing static methods in interfaces is to improve sharability.
- In interfaces, if we declare static methods then we must provide implementation part for that method at the same declaration.
- If we declare static methods inside an interface then we are able to access that static method by using the respective interface name only, not possible to access with interface reference variable, implementation class name and implementation class reference variable.

Example:

```
package com.sssit.online;

public interface ATM {

	static boolean validatePin(int pin) {
		if(pin==1234)
			return true;
		return false;
	}

	String withdrawl(int amount);
}


package com.sssit.online;

public class SavingsAccount implements ATM{

	int availableBalance = 1000;
```

```
	@Override
	public String withdrawl(int amount) {

		if(availableBalance-amount>1000)
			return "Success...";

		return "InSufficient Balance......";
	}

}
package com.sssit.online;

public class SalaryAccount implements ATM{

	int availableBalance = 1000;

	@Override
	public String withdrawl(int amount) {

		if(availableBalance-amount>0)
			return "Success...";

		return "InSufficient Balance......";
	}

}

package com.sssit.online;

public class ATMDemo {

	public static void main(String[] args) {

		SavingsAccount myAccount = new SavingsAccount();

		if(ATM.validatePin(1234))
			System.out.println(myAccount.withdrawl(1000));
		else
			System.out.println("Invalid Pin........");

		//System.out.println(myAccount.validatePin(1234));

	}
}
```

SSSIT COMPUTER EDUCATION

Default Methods in interfaces

- Till JAVA7 version, interfaces are able to include only abstract methods.
- From JAVA8 onwards, interfaces supports default methods, which includes initial / default implementation.

- In general, in java applications, we will use interfaces to declare services and we will give an option to implement these services to some other module members or some other thirds party vendors, in this context, to provide initial / default implementation for the service methods inside the interfaces then we have to use "default methods".
- If we declare default methods inside the interfaces then we are able to reuse that default implementation or we may override the default implementation as per our application requirement.
- To declare default methods inside the interfaces we have to use "**default**" keyword.

Example:

```
package com.sssit.online;

public interface Vehicle {

	default String numberOfWheels()
	{
		return "Two";
	}

}
```

```
package com.sssit.online;

public class FourWheller implements Vehicle {

	public String numberOfWheels()
	{
		return "Four";
	}

}
```

```
package com.sssit.online;

public class TwoWheller implements Vehicle {

}
```

```
package com.sssit.online;

public class DefaultMethodApp {

	public static void main(String[] args) {

		TwoWheller bike = new TwoWheller();
		FourWheller car = new FourWheller();
```

SSSIT COMPUTER EDUCATION

```
		System.out.println("NO. of Wheels for bike: " + bike.numberOfWheels());
		System.out.println("NO. of Wheels for Car: " + car.numberOfWheels());

		// TODO Auto-generated method stub

	}

}
```

Functional Interfaces:

- Any interface with exactly one abstract method then that interface is called as Functional Interface.
- Some of the examples of Functional Interfaces are:
  - java.lang.Runnable
  - java.util.Comparable

If we want to declare user defined interface as functional interface then we have to use the following annotation.

```
@FunctionalInterface
public interface <<Interface-Name>> {

}
```

In a Functional Interfaces, we can write any no of static methods and any no of default methods, but only one abstract method.

```
package com.sssit.online;

@FunctionalInterface
public interface ValidateRule {

	boolean validate(String str);

}
```

Lambda Expressions

- Lambda Expressions is a feature derived from Lambda Calculas, It was introduced in 1930's in Maths, by getting the advantages of Lambda Calc.
- Advantages of Lambda Expressions in a Java Application
  - ➔ It will reduce Code.
  - ➔ It can be used as an alternative for Anonymous Inner classes.
  - ➔ It can be passed as parameters to the methods.
  - ➔ It will introduce Function /Procedure oriented programming in Java
  - ➔ It will be used as an Object.
  - ➔ It provides replacement for the implementation classes of the Functional interfaces.

What is Lambda Expression?

- Lambda Expression is an anonymous Function, it is a normal function, it does not include name, return type and access modifiers.

Syntax:

(list-of-parameters) -> {<<Function-body>>}

```
package com.sssit.online;

public class ValidationDemo {

	public static void main(String[] args) {

		ValidateRule length = (str) -> str.length()>8 ? true : false;

		System.out.println(length.validate("abcdefghijkl"));
		System.out.println(length.validate("abc"));

		ValidateRule splSymbol = str -> str.contains("@");
		System.out.println(splSymbol.validate("abc@gmail.com"));
		System.out.println(splSymbol.validate("abc"));
	}

}
```

- In Lambda Expression body, if we are using only one statement then curly braces are optional.
- In Lambda Expression , If the paramter types predicted by the compiler automatically depending on the situation then parameter data types are optional
- If we want to access Lambda Expressions then we have to use Functional interface.
- In Lambda Expression, if only one parameter is existed then it is optional to provide parenthesis [()]

Note: If we want to use "return" keyword in Lambda Expression Body then curly braces are mandatory, if we remove return keyword then curly braces are not required and provide the value at right side of the expression without using curly braces then JVM will return the provided value from the Lambda Expression.

Examples:

Application 1:

```
package com.sssit.online.app1;

@FunctionalInterface
public interface Message {

	void wish();

}
package com.sssit.online.app1;

public class MessageDemo {

	public static void main(String[] args) {


		Message hiMsg = ()-> System.out.println("Hi....");
		hiMsg.wish();

	}

}
```

Application – 2

```
package com.sssit.online.app2;

@FunctionalInterface
public interface ValidateRule {
```

SSSIT COMPUTER

```
	boolean validate(String str);

}
package com.sssit.online.app2;

public class ValidationDemo {

	public static void main(String[] args) {

		ValidateRule length = (str) -> str.length()>8 ? true : false;

		System.out.println(length.validate("abcdefghijkl"));
		System.out.println(length.validate("abc"));

		ValidateRule splSymbol = str -> str.contains("@");
		System.out.println(splSymbol.validate("abc@gmail.com"));
		System.out.println(splSymbol.validate("abc"));

	}

}
```

Application – 3

```
package com.sssit.online.app3;

@FunctionalInterface
public interface Display {

	void displayArray(Object[] values);

}
package com.sssit.online.app3;

public class MessageDemo {

	public static void main(String[] args) {

		Integer numValues[] = {1,2,3,4,5};
		String strValues[] = {"One","Two","Three","Four","Five"};

		Display arrayDisplay = obj -> {
			for(Object value : obj)
				System.out.println(value);
		};

		System.out.println("Integer Array Values.......");
		arrayDisplay.displayArray(numValues);

		System.out.println("String Array Values.......");
		arrayDisplay.displayArray(strValues);
	}
}
```

Application – 4

SSSIT COMPUTER EDUCATION

```
package com.sssit.online.app4;

import java.util.Collection;

public interface Display {

	static void displayCollection(Collection values) {

		for(Object value : values)
			System.out.println(value);

	}

}
package com.sssit.online.app4;

public class Student {

	Integer htno;
	String name;

	public Student() {
		super();
		// TODO Auto-generated constructor stub
	}

	public Student(Integer htno, String name) {
		super();
		this.htno = htno;
		this.name = name;
	}
	public Integer getHtno() {
		return htno;
	}
	public void setHtno(Integer htno) {
		this.htno = htno;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}

	@Override
	public String toString() {
		return "Student [htno=" + htno + ", name=" + name + "]";
	}



}

package com.sssit.online.app4;
```

SSSIT COMPUTER EDUCATION

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;



public class StudentApp {

	public static void main(String[] args) {



		ArrayList<Student> studList = new ArrayList<Student>();
		studList.add(new Student(110,"stud10"));
		studList.add(new Student(105,"stud5"));
		studList.add(new Student(108,"stud8"));
		studList.add(new Student(101,"stud1"));
		studList.add(new Student(106,"stud6"));

		Display.displayCollection(studList);

		Comparator<Student> htnoComparator =
				(s1,s2) -> s1.getHtno().compareTo(s2.getHtno());

		Collections.sort(studList,htnoComparator);

		System.out.println("After Sort.........");
		Display.displayCollection(studList);







	}

}
```

Application – 5

```
package com.sssit.online.app5;

import java.util.TreeSet;

public class StringBufferDemo {
	public static void main(String[] args) {
		StringBuffer sb1 = new StringBuffer("AAA");
		StringBuffer sb2 = new StringBuffer("B");
		StringBuffer sb3 = new StringBuffer("CCCC");
		StringBuffer sb4 = new StringBuffer("DD");
		StringBuffer sb5 = new StringBuffer("EEEEE");

		TreeSet<StringBuffer> ts = new TreeSet<>((s1, s2) -> {
			int val = 0;
			int length1 = s1.length();
```

SSSIT COMPUTER EDUCATION

```
                int length2 = s2.length();
                if (length1 < length2) {
                        val = -100;
                } else if (length1 > length2) {
                        val = 100;
                } else {
                        val = 0;
                }
                return -val;
        });

        ts.add(sb1);
        ts.add(sb2);
        ts.add(sb3);
        ts.add(sb4);
        ts.add(sb5);
        System.out.println(ts);
    }
}
```

Application - 6

```
package com.sssit.online.app6;

public class ThreadDemo {

    public static void main(String[] args) {
        new Thread(()->System.out.println("Hi....")).start();
    }

}
```

Function:

- Function is a Functional interface provided by JAVA 8 in a java.util.function.Function package.
- It includes apply(--) method, it will take an input parameter of any data type and it will return a value of any data type[not only boolean].

```
public interface Function<T, R>{
public R apply(T t);
}
```

Examples:

```
package com.sssit.online.app1;

import java.util.function.Function;
```

```
public class FunctionApp1 {

	public static void main(String[] args) {


		Function<String, Integer> strLength = (str) -> str.length();

		System.out.println("Length of ABC is:" + strLength.apply("abc"));

		Function<String, String> convertUpper = (str) -> str.toUpperCase();
		System.out.println("Result is:" + convertUpper.apply("abc"));


	}

}
```

```
package com.sssit.online.app2;

import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.function.Function;

public class EncryptPassword {

	public static void main(String[] args) {


		Function<String, String> encryptPasswordSha256 =
				(password) -> {
					MessageDigest md;

					try {
						md = MessageDigest.getInstance("SHA-
256");
						byte[] hashInBytes =
md.digest(password.getBytes(StandardCharsets.UTF_8));

						// bytes to hex
					  StringBuilder sb = new StringBuilder();
					  for (byte b : hashInBytes) {
					      sb.append(String.format("%02x", b));
					  }
					  return sb.toString();
					} catch (NoSuchAlgorithmException e) {
						// TODO Auto-generated catch block
						e.printStackTrace();
					}

					return "";


				};
```

SSSIT COMPUTER EDUCATION

```
		String actualPassword = "SSSIT";
		String encryptedPassword =
				encryptPasswordSha256.apply(actualPassword);

		System.out.println("Actual Password = " + actualPassword);
		System.out.println("Encrypted Password = " + encryptedPassword);



	}

}
package com.sssit.online.app3;

import java.util.function.Function;

public class FunctionApp3 {

	public static void main(String[] args) {

		Function<String, Integer> func = x -> x.length();

	  Function<Integer, Integer> func2 = x -> x * 2;

	  Integer result = func.andThen(func2).apply("SSSIT");

	  System.out.println(result);


	}

}
package com.sssit.online.app4;

import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.function.Function;

public class ListToMapDemo {

   public static void main(String[] args) {

	ListToMapDemo obj = new ListToMapDemo();

	  List<String> list =
			Arrays.asList("node", "c++", "java", "javascript");


	  Map<String, Integer> map = obj.convertListToMap(list, x -> x.length());

	  System.out.println(map);    // {node=4, c++=3, java=4, javascript=10}
```

SSSIT COMPUTER EDUCATION

```
    }

    public <T, R> Map<T, R> convertListToMap(List<T> list, Function<T, R> func) {

        Map<T, R> result = new HashMap<>();
        for (T t : list) {
            result.put(t, func.apply(t));
        }
        return result;

    }
}
```

```
package com.sssit.online.app5;

import java.util.function.BiFunction;

public class BiFunctionDemo {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		BiFunction<String, String, String> fullName =
				(str1,str2) -> str1+" " + str2;

		System.out.println("Full Name = " +
				fullName.apply("First", "Second"));

	}

}
```

Predicate:

- Predicate is a predefined functional interface provided by JAVA8 defined in a package java.util.function.Predicate.
- Predicate functional interface is having test() method, it will take an input parameter and it will check the provided condition and it will return the result of the conditional expression , that is, true or false value.

```
public interface Predicate{
public boolean test(T t)
}
```

- In Java applications, if we want to test a condition and if we want to return the result of test case then prepare Lambda Expression for

Predicate interface and use that Lambda expression in Java application, it will reduce our explicit condition checking and return statements.

Examples:

```
package com.sssit.online.app1;

import java.util.function.Predicate;

public class PredicateDemo {

	public static void main(String[] args) {

		Predicate<String> passwordStrength =
				(pass) -> pass.length()>8;

		System.out.println("abcd...." + passwordStrength.test("abcd"));
		System.out.println("abcdefghi...." +
					passwordStrength.test("abcdefghi"));


	}

}
package com.sssit.online.app2;

import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;

public class PredicateDemo2 {

	public static void main(String[] args) {

		List<Integer> list =
				Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

		Predicate<Integer> evenCond = x -> x%2==0;

		//System.out.println("Even Sum = " + evenSum(list, x -> x%2==0));
		System.out.println("Even Sum = " + sumDigits(list, evenCond));
		System.out.println("Odd Sum = " + sumDigits(list, evenCond.negate()));

	}

	public static int sumDigits(List<Integer> lst, Predicate<Integer> cond)
	{
		int sum = 0;
		for(Integer value : lst)
		{
			if(cond.test(value)) {
				sum += value;
				System.out.println(value);
			}
```

SSSIT COMPUTER EDUCATION

```
		}

		return sum;

	}

}
package com.sssit.online.app3;

import java.util.function.Predicate;

public class PredicateDemo3 {

	public static void main(String[] args) {

		Predicate<String> mailIdValidation =
				(mailId) -> mailId.length()>0;
		Predicate<String> splSymbol =
				(mailId) -> mailId.indexOf("@")>1;

		System.out.println("abcd...." +
				mailIdValidation.and(splSymbol).test("abcd"));
		System.out.println("abcdefghi...." +
				mailIdValidation.and(splSymbol).test("abcd@gmail.com"));


	}

}
```

Usage:

- Predicate is mainly for Condition checking and Function is mainly for processing over the data.

Supplier

- Supplier is functional interface which does not take any argument and produces result of type T.
- It has a functional method called T get()
- As Supplier is functional interface, so it can be used as assignment target for lambda expressions.

```
public interface Supplier<T>{
public T get();
}
```

Examples

```
package com.sssit.online.app1;

public class Student {

		private int id;
		private String name;


		public Student(int id, String name) {
			super();
			this.id = id;
			this.name = name;

		}

		public int getId() {
			return id;
		}

		public void setId(int id) {
			this.id = id;
		}

		public String getName() {
			return name;
		}

		public void setName(String name) {
			this.name = name;
		}



		@Override
		public String toString() {
			return "Student [id=" + id + ", name=" + name +  "]";
		}


}
```

```
package com.sssit.online.app1;

import java.util.function.Supplier;

public class SupplierDemo {

	public static void main(String[] args) {

		Supplier<String> supplier= ()-> "SSSIT";
		System.out.println(supplier.get());
```

SSSIT COMPUTER EDUCATION

```
		Supplier<Student> studentObject = () -> new Student(101,"Student1");
		System.out.println(studentObject.get());

	}

}
```

Application – 2

```
package com.sssit.online.app2;

import java.util.Random;
import java.util.function.Supplier;

public class OTPGeneratorDemo {

	public static void main(String[] args) {


		Supplier<Integer> otpNumber = ()-> {
			return new Random().nextInt(999999);
		};

		System.out.println("OTP = " + otpNumber.get());
		System.out.println("OTP = " + otpNumber.get());
		System.out.println("OTP = " + otpNumber.get());



	}

}
```

Consumer

- Consumer<T> is an in-built functional interface introduced in Java 8 in the java.util.function package.
- Consumer can be used in all contexts where an object needs to be consumed,i.e. taken as input, and some operation is to be performed on the object without returning any result.

```
public interface Consumer<T>{
   public void accept (T t)
}
```


```
package com.sssit.online.app1;

import java.util.function.Consumer;

public class ConsumerApp1 {

	public static void main(String[] args) {

		Consumer<String> print = x -> System.out.println(x);
		 print.accept("java");

	}

}
```

```
package com.sssit.online.app2;

import java.util.Arrays;
import java.util.List;
import java.util.function.Consumer;

public class ConsumerApp2 {
	public static void main(String[] args) {

		List<String> list = Arrays.asList("a", "ab", "abc");

		forEach(list, (String x) -> System.out.println(x.length()));

	}

	static <T> void forEach(List<T> list, Consumer<T> consumer) {
		for (T t : list) {
			consumer.accept(t);
		}
	}

}
```

SSSIT COMPUTER EDUCATION

Optional Class

- Java 8 introduced an optional class in the “java.util” package.
- “Optional” is a public final class and is used to deal with NullPointerException in the Java application.
- Using Optional, you can specify alternate code or values to run.
- By using Optional you don’t have to use too many null checks to avoid nullPointerException.
- You can use the Optional class to avoid abnormal termination of the program and prevent the program from crashing.

- The Optional class provides methods that are used to check the presence of value for a particular variable.

The following program demonstrates the use of the Optional class.

```
package com.sssit.online.app1;


import java.util.Optional;

public class OptionalApp1 {
	public static void main(String[] args) {
		String[] str =
{"First","Second","Third","Four","Five",null,null,null,null,null};
		Optional<String> checkNull =
				Optional.ofNullable(str[5]);
		if (checkNull.isPresent()) {
			System.out.print(checkNull);
		 } else
		   System.out.println("string is null");


		Optional<String> value =
			Optional.ofNullable(str[3]);

	   if (value.isPresent()) {
	    System.out.print(value);
	} else
	  System.out.println("string is null");
	    }
}
```

In this program, we use the “ofNullable” property of the Optional class to check if the string is null. If it is, the appropriate message is printed to the user.

forEach Method in iterable interface

In Java 8, we have a newly introduced forEach method to iterate over collections and Streams in Java.

```
public void forEach(Consumer<? super E> action)
```

Note: This method throws NullPointerException if the specified action is null.

```
package com.sssit.online;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class ForEachApp1 {

	public static void main(String[] args) {


		List<String> strList = Arrays.asList("One","Two","Three","Four","Five");

		System.out.println("List of String are.....");
		strList.forEach(str -> System.out.println(str));
	}

}


package com.sssit.online.app2;

import java.util.HashMap;
import java.util.Map;

public class ForEachApp2 {

	public static void main(String[] args) {



	Map<String, Integer> items = new HashMap<>();
	items.put("A", 10);
	items.put("B", 20);
	items.put("C", 30);
	items.put("D", 40);
	items.put("E", 50);
	items.put("F", 60);

	items.forEach((k,v)->System.out.println("Item : " + k + " Count : " + v));


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

	items.forEach((k,v)->{
		if(v>30)
			System.out.println("Item : " + k + " Count : " + v);
	});

	}

}

package com.sssit.app3;
```

SSSIT COMPUTER EDUCATION

```
import java.util.Map;

public class ForEachApp3 {

	public static void main(String[] args) {


		Map<String, Integer> items = null;
		items.forEach((k,v)->System.out.println("Item : " + k + " Count : " +
v));
		//The above line gives Null Pointer Exception.

	}

}
```

Stream API

- In Java applications, Collection objects are able to represent a group of Other objects as a single entity.
- But, Stream API is a set of predefined Classes and Interfaces, which are used to process the elements which are represented in the form of Collection objects.
- The complete predefined classes and interfraces which are required for Stream API are provided by JAVA8 version in the form of java.util.stream package.

If we want to use Stream API in Java applications then we have to use the following steps.

1. Create Stream object from a particular Collection
2. Configure Stream object either with filter or with map.
3. Processing the elements.

Step 1: Create Stream object from a particular Collection

```
public Stream stream();

	Ex: Stream s = al.stream();
```

Step 2: Configure Stream object

There are two ways to configure Stream object.

1. By using Filter

- Filtering mechanism will use a Predicate to filter the elements, for this we have to use the following method from java.util.stream.Stream.

```
public Stream filter(Predicate p)
```

2. By Using Map

- Map mechanism will use a Function to process the elements, for this we have to use the following method from java.util.stream.Stream

```
public Stream map(Function f)
```

Step 3: Processing the elements.

After getting all the elements in the form of Stream we have to process that elements by using the following methods.

1. collect(-)
2. count(-)
   a. It will count the no of elements which are existed in Stream.
3. sorted(-)
   a. It able to keep all the elements in a particular Sorting order.
      Note: By using sorted() method we are able to provide both natural sorting or customized sorting.
4. min(-)
   a. It will return min value from Stream
5. max(-)
   a. It will return min value from Stream
6. forEach(-)
   a. It able to return max value from Stream
7. Collectors(-)


```
package com.sssit.online.filter;

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class FilterDemo {

        public static void main(String[] args) {

      List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

      List<Integer> collect = list.stream().filter(x -> x > 5).collect(Collectors.toList());

      System.out.println(collect);

   }

}
```

```
package com.sssit.online.map;

import java.util.Arrays;
import java.util.List;
import java.util.function.Function;
import java.util.stream.Collectors;

public class MapDemo {

       public static void main(String[] args) {

         List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

         List<Integer> collect = list.stream()
                   .map(x -> x * 10).collect(Collectors.toList());

         Function<Integer,Integer> squareNumber = x -> x * x;
         List<Integer> squares = list.stream()
                   .map(squareNumber).collect(Collectors.toList());

         System.out.println(collect);
         System.out.println("List of Squares = ");
         squares.forEach(x -> System.out.println(x));


    }

}
```

SSSIT COMPUTER

```
package com.sssit.online.ops;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;
import java.util.function.Function;
import java.util.stream.Collectors;

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

		List<Integer> list = Arrays.asList(5,1,6,40,30,15,8,4);

		Optional<Integer> max = list.stream().max((x,y)->x.compareTo(y));
		Optional<Integer> min = list.stream().min((x,y)->x.compareTo(y));

		System.out.println("Maximum value = " + max.get());
		System.out.println("Minimum value = " + min.get());


	}

}

package com.sssit.online.sorting;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class SortDemo {

	public static void main(String[] args) {

		List<Integer> list = Arrays.asList(5,1,6,40,30,15,8,4);

		List<Integer> ascendingOrder =
list.stream().sorted().collect(Collectors.toList());
		ascendingOrder.forEach(x -> System.out.println(x));

		List<Integer> descendingOrder = list.stream().sorted((x,y)->y.compareTo(x))
				.collect(Collectors.toList());
		System.out.println("Descending Order.....");
		descendingOrder.forEach(x -> System.out.println(x));

	}

}


package com.sssit.online.count;

import java.util.Arrays;
import java.util.List;
```

SSSIT COMPUTER EDUCATION


```
public class CountDemo {

	public static void main(String[] args) {

		List<Integer> list = Arrays.asList(5,1,5,4,1,5,3,3);

		list.stream().distinct().forEach(value->{
			System.out.println(value + " Repeated "
			+ list.stream().filter(x->x==value).count() + " times");

		});
	}

}
```

```
package com.sssit.online.lst2map;

import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.function.Function;
import java.util.stream.Collectors;

public class ListToMap {

	public static void main(String[] args) {


		List<String> names =
				Arrays.asList("First", "Second", "Third",
				   "Four", "Five");

		Map<Object, Object> namesMap = names.stream().
				collect(Collectors.toMap(k->k, v -> v.length()));
		namesMap.forEach((k,v)-> System.out.println("Length of  " + k + " is: "
+ v));


		List<Integer> list = Arrays.asList(5,1,5,4,1,5,3,3);
		Map<Integer, Long> mp = list.stream().
				collect(Collectors.groupingBy(Function.identity(),
Collectors.counting()));
		mp.forEach((k,v)-> System.out.println(k + " repeated " + v + " times"));
	}
}
```

Date Time API

SSSIT COMPUTER EDUCATION

Date Time API from JAVA8 version is providing simplified approach to represent date and time in JAVA applications.

JAVA8 version has provided the complete predefined library in the form of "java.time" package.

java.time package has provided the following predefined classes to represent date in java applications.

java.time.LocalDate
java.time.LocaTime
java.time.LocalDateTime

Example:

```
package java8features;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;

public class Test {
public static void main(String[] args)throws Exception {
LocalDate date = LocalDate.now();
System.out.println(date);
int day = date.getDayOfMonth();
int month = date.getMonthValue();
int year = date.getYear();
System.out.println(day+"-"+month+"-"+year);
System.out.println();

LocalTime time = LocalTime.now();
System.out.println(time);
int hour = time.getHour();
int mnt = time.getMinute();
int scn = time.getSecond();
int nscn = time.getNano();
System.out.println(hour+":"+mnt+":"+scn+":"+nscn);
System.out.println();

LocalDateTime dt = LocalDateTime.now();
System.out.println(dt);
int day1 = dt.getDayOfMonth();
int month1 = dt.getMonthValue();
int year1 = dt.getYear();
int hour1 = dt.getHour();
int mnt1 = dt.getMinute();
int scn1 = dt.getSecond();
int nscn1 = dt.getNano();
```


```
System.out.println(day1+"-"+month1+"-
"+year1+":"+hour1+":"+mnt1+":"+scn1+":"+nscn1);


}
}
```

SSIT COMPUTER EDUCATION