CORE JAVA

HISTORY OF JAVA

- James Gosling initiated the Java language project in June 1991 for use in one of his many set-top box projects.

- The language, initially called Oak after an oak tree that stood outside Gosling's office, also went by the name Green and ended up later being renamed as Java, from a list of random words.

- Sun released the first public implementation as Java 1.0 in 1995. It promised **Write Once, Run Anywhere**(WORA).

- Oracle acquired Sun Microsystems in 2010, and since that time Oracle's hardware and software engineers have worked side-by-side to build fully integrated systems and optimized solutions designed to achieve performance levels that are unmatched in the industry.

C++ vs JAVA

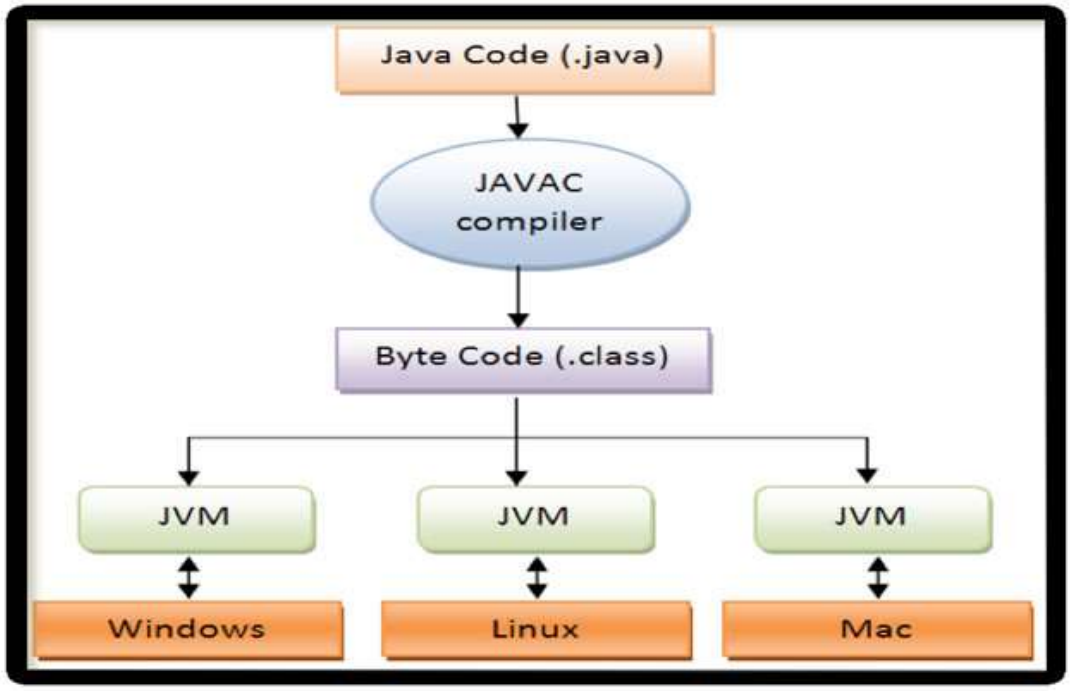
C++	JAVA
1. C++ is not a purely object-oriented programming	1. Java is purely an object-oriented programming language
2. Pointers are available in C++.	2. We cannot create and use pointers in Java
3. Allocating memory and de-allocating memory is the responsibility of the programmer.	3. Allocation and de-allocation of memory will be taken care of by JVM.
4. Operator overloading is available in C++.	4. It is not available in Java.
5. There are 3 access specifiers in C++: private, public and protected	5. Java supports 4 access specifiers: private, public, protected, and default.
6. There are constructors and destructors in C++.	6. Only constructors are there in Java. No destructors are available in this language

WHY JAVA?

- Java is easy to learn
- Java is object-oriented
- Java is platform-independent.
- Java is distributed.
- Java is secure.
- Java is robust.
- Java is multithreaded

WHAT IS JVM?

- JVM (Java Virtual Machine) is an abstract machine.
- It is a specification that provides runtime environment in which java bytecode can be executed.
- JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).



How Much Java you need to learn for a good Selenium Scripting ?

- Java Programming essentials
- OOP’s concept
- Control Statements
- Looping statements
- Arrays Concepts
- Threads and Multithreading Concepts
- Java Collections Framework

JAVA Building Blocks

- Objects
- Class
- Methods
- Instance Variables
- Case Sensitivity
- Class Names
- Method Names
- public static void main(String args[])

First Java Application

- File Name=Class Name
- Syntax:-

```
class ClassName
{
        public static void main(String[] args)
        {
                //Lines of Code1
        }
}
```

- Example:- class MyFirstProg (MyFirstProg.java)

```
{
        public static void main(String[] args)
        {
                System.out.println("This is my First Java Program");
        }
}
```

Why Standard Syntax for Java main() Method?

- public static void main(String[] args)

  - ❖ public -> Access Specifier.
  - ❖ static -> Access Modifier.
  - ❖ void -> Return Type of Method.
  - ❖ main -> Name of the Method. (Starting Point of Execution).
  - ❖ String[] args -> Parameter of main method and accepts command line arguments during program execution.

Access Specifier

- Access Specifiers are used to specify the scope for a class and members of class.
- Helps in providing security to class and members of class. (Restrict Access).
- Types of access:

Types of Access for Class and Members of Class	Keyword provided by Java to support the access
Private Access (Not Applicable for Class)	private
Default Access	No Keyword
Protected Access (Not Applicable for Class)	protected
Public Access	public

static Keyword

- “static” Keyword is used for declaring the members of the class. (variables, methods or blocks).
- static members belong to the class.
- static members can be accessed without creating an object.
- static members cannot access non- static members .
- static keyword can be used with following members of class:
  - Variables
  - Methods

Static Variables

- Variables which are declared with “static” keyword are called static variables.
- It belongs to class and do not belong to instance or object.
- Only one copy of static variable is present and it belongs to class.
- Similar to Global variable, if value is changed it affects globally.
- Static Variables can be accessed inside class directly.
- Eg: public static int i=2;

Static Methods

- Methods which are declared with “static” keyword are called static methods.
- Static methods can be accessed without creation of objects.
- It cannot access instance methods.
- Eg: public static int method1()

```
{
    //Line of Codes
    return x;
}
```

Data Types/ Return Types

- Java has 3 types of data types:
  - **Primitive** (byte ,short, int, long, float, double, char and boolean).
  - **Derived Types** (arrays).
  - **User Defined Types** (class, subclass, abstract class, interface, enumerations and annotations).

byte

- Byte data type is an 8-bit signed two's complement integer.
- Minimum value is -128 (-2^7)
- Maximum value is 127 (inclusive)(2^7 -1)
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.
- Example: byte a = 100 , byte b = -50

short

- Short data type is a 16-bit signed two's complement integer.
- Minimum value is -32,768 (-2^15)
- Maximum value is 32,767 (inclusive) (2^15 -1)
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an int
- Default value is 0.
- Example: short s = 10000, short r = -20000

Primitive Data Types

int

- Int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648.(-2^31)
- Maximum value is 2,147,483,647(inclusive).(2^31 -1)
- Int is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0.
- Example: int a = 100000, int b = -200000

long

- Long data type is a 64-bit signed two's complement integer.
- Minimum value is -9,223,372,036,854,775,808.(-2^63)
- Maximum value is 9,223,372,036,854,775,807 (inclusive). (2^63 -1)
- This type is used when a wider range than int is needed.
- Default value is 0L.
- Example: long a = 100000L, long b = -200000L

float

- Float data type is a single-precision 32-bit IEEE 754 floating point.
- Float is mainly used to save memory in large arrays of floating point numbers.
- Default value is 0.0f.
- Float data type is never used for precise values such as currency.
- Example: float f1 = 234.5f

double

- double data type is a double-precision 64-bit IEEE 754 floating point.
- This data type is generally used as the default data type for decimal values, generally the default choice.
- Double data type should never be used for precise values such as currency.
- Default value is 0.0d.
- Example: double d1 = 123.4

Primitive Data Types

char

- char data type is a single 16-bit Unicode character.
- Minimum value is '\u0000' (or 0).
- Maximum value is '\uffff' (or 65,535 inclusive).
- Char data type is used to store any character.
- Example: char letterA ='A'

boolean

- boolean data type represents one bit of information.
- There are only two possible values: true and false.
- This data type is used for simple flags that track true/false conditions.
- Default value is false.
- Example: boolean one = true

Java Identifiers

- Names used for classes, variables and methods are called as Identifiers.
- Naming Conventions used by Java Programmers:
  - Any Identifier should have 1st character as a letter(A to Z or a to z), dollar symbol ($) and underscore(_).
  - 1st character cannot be a digit.
  - It cannot have space.
  - It cannot be a keyword.

Valid Identifiers	Invalid Identifiers
abc123	123abc
$test	if
_123	&abc
AVGTEMP	abc 123

Variables

- Variables is a named memory location that maybe assigned a value.
- Value in a variable can be either changed or modified.
- Syntax: datatype variablename=value;
- Example: int i=10;
- There are three kinds of variables in Java:
  - ✓ Method Local variables.
  - ✓ Instance variables.
  - ✓ Class/static variables.

Method Local variables

- Variables are declared anywhere inside method are known as Method Local Variables.
- Default Values are not assigned to Local Variables.
- Variables comes into life when they declared/initialized and ends when method completes.
- Example:-

```
public void add(int a,int b)
{
   int c; //Method Local Variable
   c=a+b;
   System.out.println("Sum is" +c);
}
```

Instance Variables

- Variables which are declared at class level is known as Instance Variables.
- Instance Variables come into life when the object gets created and destroyed when object is dereferenced.
- Number of Instance Variable=Number of Objects created.
- It belongs to objects.
- It can be accessed inside the class directly or outside the class with the help of "." (dot) operator.
- "." operator should be used with reference variable after it is initialized by creating an object.
- Example:

```
MyFirstProg obj=new MyFirstProg ();
obj.add(2,3);
```

Static Variables

- Variables which are declared with “static” keyword are called static variables.
- It belongs to class and do not belong to instance or object.
- Only one copy of static variable is present and it belongs to class.
- Similar to Global variable, if value is changed it affects globally.
- Static Variables can be accessed inside class directly.
- Eg: public static int i=2;

Arrays in Java

- An array as a collection of variables of the same type.
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.
- Declaring Array Variables:
  - dataType[] arrayRefVar; // preferred way.

  or
  - dataType arrayRefVar[]; // works but not preferred way.
- Creating Arrays:

  dataType[] arrayRefVar = new dataType[arraySize];

  ❖ Alternatively you can create arrays as follows:

  dataType[] arrayRefVar = {value0, value1, ..., valuek}

Operators in Java

- Arithmetic Operators (+ , - ,* , / , %).
- Assignment Operator (= , += , -= , *= , /= , %=).
- Relational Operator (> , < , >= , <= , == , =).
- Logical Operator (&& , ||).
- Increment/Decrement Operator (++ , --).
- Ternary Operator (?:).
- new
- instanceof
- Bitwise Operator (>> , << , & , | , ~ , ^)

Arithmetic Operators

- Used for doing mathematical operations.
- Combining arithmetic operator with 2 or more operands is known as ***Arithmetic Expression.***
- Result of arithmetic expression is integer or floating value.
- **Example:**

```
int i=a+b;
float f=a-b;
int m=a*b;
float n=a/b;
int o=a%b;
```

Assignment Operator

- Used for assigning values from right side operand to the left side operand.
- All assignment operators are binary operators.
- When you are using assignment operator, you should use the same type of operand or both the operands should be of same data type.
- To assign value to different data type, Casting should be done. (Casting might be Implicit or Explicit).
- **Example:**

```
a=b;
a+=2;
b-=3;
c*=2;
d/=5;
e%=2;
```

Relational Operator

- All Relational operator are binary operators.
- Result of relational expression is a ***boolean*** value.
- **Example:**

```
int a=10,b=20,x=20,y=30,m=10,n=10;
boolean res=a>b;          //false
boolean res=a<b;          //true
boolean res=x>=y;         //false
boolean res=x<=y;         //true
boolean res=(m==n);       //true
boolean  res=(m!=n);      //false
```

Logical Operator

- There are 3 Logical operators used in Java. They are:
  - ✓ && (Logical AND). {Binary Operator}
  - ✓ || (Logical OR). {Binary Operator}
  - ✓ ! (Logical NOT). {Unary Operator}
- Operands of logical operator must be of boolean type.
- Logical operators form logical expression whose result is a boolean value.

A	B	A && B	A || B	!A
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

Increment/Decrement operator

- Both are unary operator.
- “++” operator is an increment operator, which increments the operand value by 1.
- “--” operator is an decrement operator, which decrements the operand value by 1.
- A prefix operator (used before operand) first adds/subtracts 1 to the operand and then result is assigned to the variable on the left.
  - Example:

```
int i=5;
System.out.println(++i);
Output: 6
```

```
int j=5;
System.out.println(--j);
Output: 4
```

- A suffix operator (used after operand) first assigns the value to the variable on left and then adds/subtract 1 from the operand.
  - Example:

```
int m=5;
System.out.println(m++);
Output: 5
```

```
int n=5;
System.out.println(n--);
Output: 5
```

Ternary Operator

- Used to perform some simple conditional check.
- First Condition will be evaluated and if condition is true then the true block value will be assigned to variable. If condition is false, then false block value will be assigned to variable.
- **Syntax:** Var=(condition)?TrueBlock:FalseBlock;
- **Example 1:**

```
int a=5,b=10;
Max=(a>b)?a:b; //Max=10    False Block Executed
```

- **Example 2:**

```
int a=10, b=5;
Max=(a>b)?a:b; //Max=10    True Block Executed
```

new Operator

- Used for creating instances/objects of classes.
- The **new** operator dynamically allocates memory for an object.
- It has this general form: ***classname var* = new *classname*( );**

  - *var* is a variable of the class type being created.
  - The *classname* is the name of the class that is being instantiated.
  - The class name followed by parentheses specifies the *constructor* for the class.

- **Example:** Box b1=new Box();
  Box b2=new Box(5,3,4);

instanceof keyword

- **instanceof operator** is used to test whether the object is an instance of the specified type.
- **Example:**

```
class Simple1
{
        public static void main(String args[]){
        Simple1 s=new Simple1();
        System.out.println(s instanceof Simple1);//true
}
}
```

- It returns a boolean value.(either true or false).

Control Statements

- All Statements are executed sequentially. In some situations, we may want to change the execution of statements based on certain conditions or to repeat a group of statements until certain specified condition is met.

- Following are the control statements:
  - if/else
  - switch/case
  - for
  - while and do..while();
  - return

if/else

if

- It is used to perform conditional checks.
- **Syntax:**

```
if(condition)
{
        Statement1;
}
Statement X;
```

- **Example:**

```
int a=10;
if(a==10)
{
        System.out.println("Value of a is 10");
}
System.out.println("Displayed regardless of condition passed or failed");
```

If..else

- If the condition is true, then statements inside if block is executed.
- If the condition is false, then statements inside else block is executed.

- **Syntax:**

```
if(condition)
{
        Statement 1;
}
else
{
        Statement 2;
}
Statement X;
```

Example:

```
if(a>10)    //Consider a=5
{
        S.o.p("a is greater than 10");
}
else
{
        S.o.p("a is less than 10");
}
S.o.p("I will be displayed regardless of any condition ");
```

if/else

if/else if…else if/else

- If the condition is true, then statements inside if block is executed.
- If the condition is false, then statements inside else block is executed.
- **Syntax:**

```
if(condition1)
{
        Statement 1;
}
else if(condition2)
{
        Statement 2;
}
else
{
        Statement 3;
}
Statement X;
```

Example:

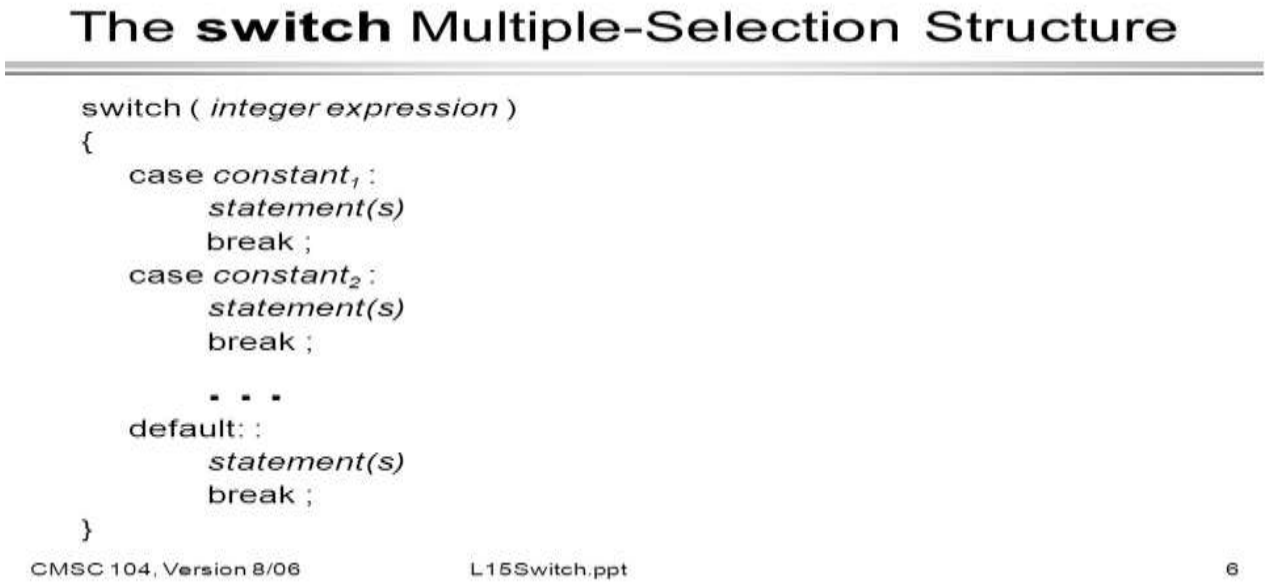
```
if(a==1)    //Consider a=2
{
        S.o.p("Value of a is 1");
}
else if(a==2)
{
        S.o.p("Value of a is 2");
}
else
{
        S.o.p("Value of a is: "+a);
}
Statement X;
```

switch/case

- switch statement is used to execute a particular set of statements among multiple conditions.
- Alternative to complicated if else if ladder conditions.
- Expression type in switch must be of any of the following data type:
  - byte
  - short
  - int
  - char
  - enum (java 1.5)
  - String (java 1.7)
- Expression type in switch must not be of any of the following data type:
  - long
  - float
  - double
  - boolean

switch/case (Contd..)

- **break** is optional. Every case must end with break statements which will help to terminate or transfer the control outside of switch block.
- If no break is used then execution will continue to next case.
- **default** is optional. If present it will execute only if the value of the expression does not match with any of the case and then control goes to statement X.



switch Example

```
switch ( day )
{
    case 0:  printf ("Sunday\n");
             break;
    case 1:  printf ("Monday\n");
             break;
    case 2:  printf ("Tuesday\n");
             break;
    case 3:  printf ("Wednesday\n");
             break;
    case 4:  printf ("Thursday\n");
             break;
    case 5:  printf ("Friday\n");
             break;
    case 6:  printf ("Saturday\n");
             break;
    default: printf ("Error -- invalid day.\n");
             break;
}
```

Is this structure more efficient than the equivalent nested if-else structure?

CMSC 104, Version 8/06 L15Switch.ppt 12

for (Iterative Statements)

for

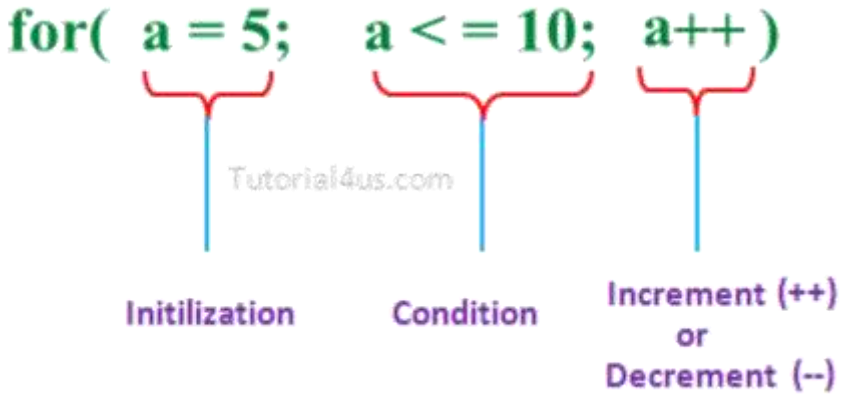
- It is used to execute a set of statements multiple times.
- Syntax: for(initialization;condition;increment/decrement)

```
{
        Statement 1;
            .
            .
        Statement n;
}
```

for (Contd...)

- Example:

```
for(int a=5;a<=10;a++)
{
        System.out.println("I will be executed 6 times");
 }
```

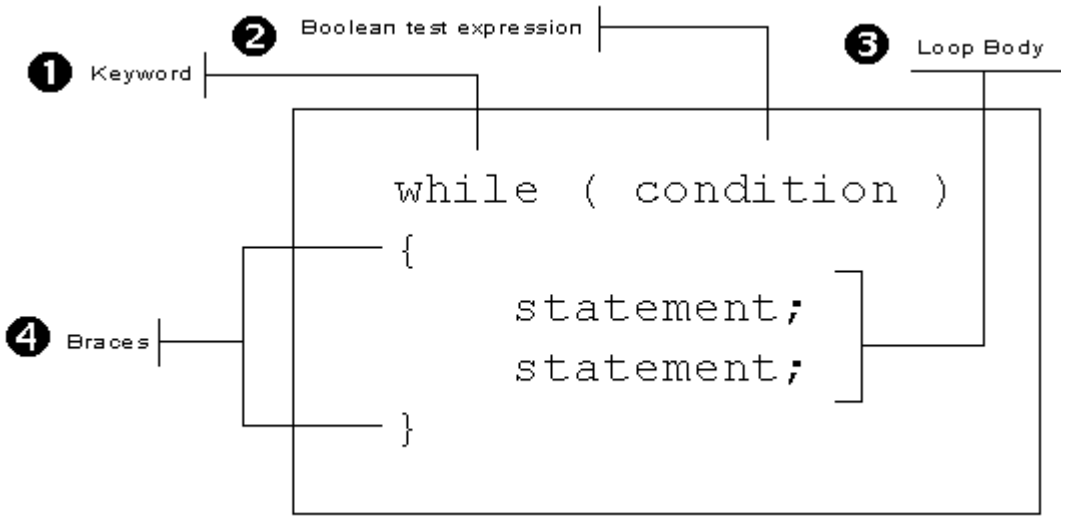


- **Note:** In for loop, 1st time: Initialization -> Condition -> Statements -> Increment/Decrement.

  From Next Round of Iteration : Condition -> Statements -> Increment/Decrement.

while

- It is an entry controlled loop.
- In while Statement, 1st condition will be checked.
- When condition is true, loop will be repeated (repetitive execution).
- When condition is false, control comes out of loop.
- **Syntax:**



- **Example:**

The while Statement

- An example of a while statement:

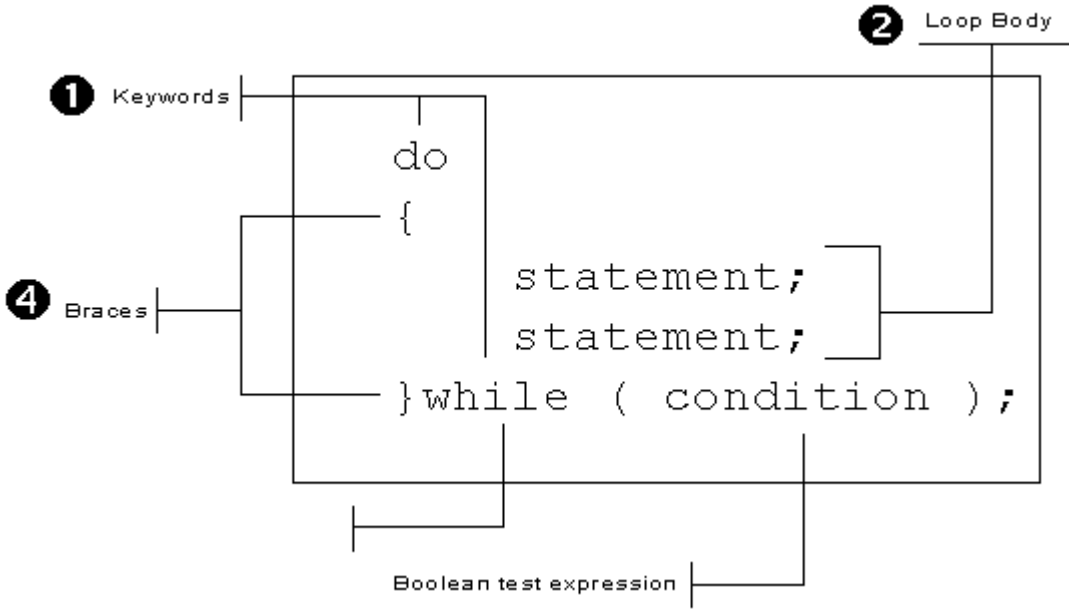
```
int count = 1;
while (count <= 5){
   System.out.println (count);
   count++;
}
```

- **If the condition of a `while` loop is false initially, the statement is never executed**
- **Therefore, the body of a `while` loop will execute zero or more times**

5-5

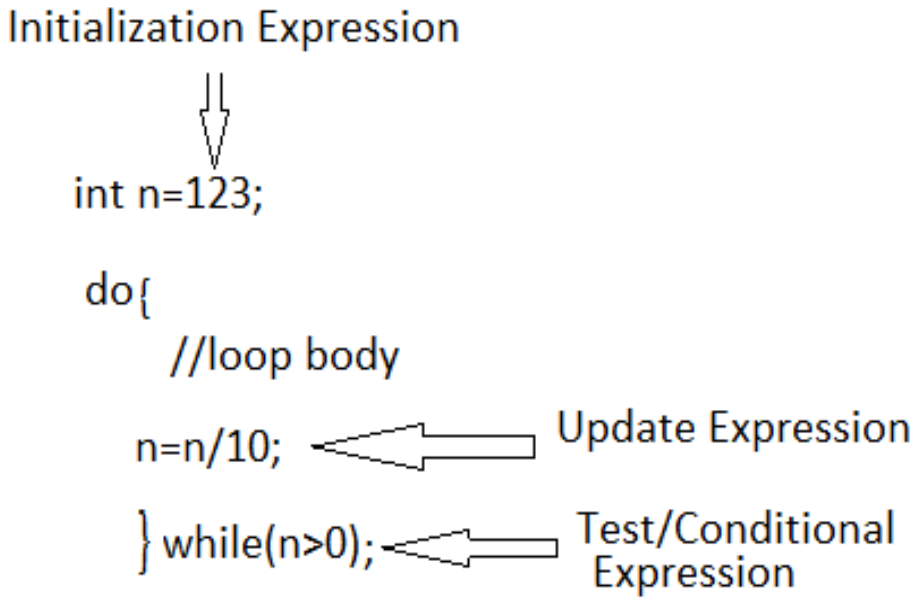
do/while

- It is an exit controlled loop.
- Unlike while, first all statements inside block are executed and then condition is verified.
- In while, when the condition is false for the first time then the statement inside the block will be executed for 0 times, whereas in do while statement will be surely executed for at least one time.
- **Syntax:**



do while Statement

- Example:



return Keyword

- “return” keyword terminates the execution in a method and returns the control to caller method.

- There are 2 main use of *return* statement. They are:

    - It immediately terminates the execution of the callee method and returns the control from callee method to caller method.
    - It is used to return a value from callee to caller method.

Java Methods

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

How to write a Method in Java?

Syntax:

```
modifier returntype methodname(param0,….,paramN)
{
        //Business Logic
}
```

Example:

```
//Method with No parameters
public int add()
{
        int a,b,c;
        c=a+b;
        return c;
}
```

```
//Method with Parameters
public int add(int a,int b)
{
        int c;
        c=a+b;
        return c;
}
```

Constructor

- Constructor is a **special method** which has the **same name** as **class name**.
- It does not have return type even void also.
- Specially designed to initialize the instance variable.
- Executed only once during creation of every new object.
- An object can never be created without invoking a constructor.
- Example:-public class SecondProg

```
{   //Default Constructor
    SecondProg()
    {


    }
}
```

```
public class SecondProg(int a,int b)
{   //Parametrized Constructor
        SecondProg(int a,int b)
        {
                this.a=a;
                this.b=b;
        }
}
```

Object Creation

- There are three steps when creating an object from a class:
  - **Declaration:** A variable declaration with a variable name with an object type.
  - **Instantiation:** The 'new' keyword is used to create the object.
  - **Initialization:** The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Example: Object Creation

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

How to Access Instance Variables and Methods?

- Instance variables and methods are accessed via created objects.

- Below are the steps followed to access instance and variables:

1. First create an object
   ObjectReference = new Constructor();

2. Now call a variable as follows
   ObjectReference.variableName;

3. Now you can call a class method as follows
   ObjectReference.MethodName();

Example: Object Creation and Invocation

```
public class Puppy{
  int puppyAge;
  public Puppy(String name){
    // This constructor has one parameter, name.
    System.out.println("Name chosen is :" + name );
  }
   public void setAge( int age ){
      puppyAge = age;
  }
public int getAge( ){
      System.out.println("Puppy's age is :" + puppyAge );
      return puppyAge;
  }
```

Example: Object Creation and Invocation (Contd..)

```
public static void main(String []args){
    /* Object creation */
    Puppy myPuppy = new Puppy( "tommy" );

    /* Call class method to set puppy's age */
    myPuppy.setAge( 2 );

    /* Call another class method to get puppy's age */
    myPuppy.getAge( );

    /* You can access instance variable as follows as well */
    System.out.println("Variable Value :" + myPuppy.puppyAge );
  }
}
```

this Keyword

- this is a reference variable which refers to current object.
- It is used to access the members of the class within the same class.
- It is used to access instance members only. It cannot be used to access static members of class.
- It can be used to access the following members of the class:
    a) Instance Variables.
    b) Constructors.
    c) Methods.

this Keyword(Contd..)

“this” keyword in Instance Variables

- “this” keyword is used to resolve the conflicts between instance variables and local variables.
- This type of conflict arises either in constructor or methods.

“this” keyword in Constructors

- this() can be used to invoke another constructor from constructor within the same class.
- this() can be used for invoking current class constructor.
- this() will be used for doing constructor chaining within the same class.
- When you are using this() to invoke a constructor, then it should be the first statement inside the constructor.

“this” keyword in Methods

- this can be used to invoke another method from one method.

Object Oriented Concepts

- There are 4 main pillars of Object Oriented Programming:-

  - ❑ Encapsulation
  - ❑ Inheritance
  - ❑ Abstraction
  - ❑ Polymorphism

Encapsulation

- *Encapsulation* in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as single unit.
- In encapsulation the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class, therefore it is also known as *Data Hiding*.
- Example:

```
public class EncapsulationDemo {
    private int ssn;
    private String empName;
    private int empAge;
   //Getter and Setter methods
    public int getEmpSSN()          public void setEmpSSN(int ssn)
   {                                {
            return ssn;                     this.ssn=ssn;
   }                                }                                  }
```

Inheritance

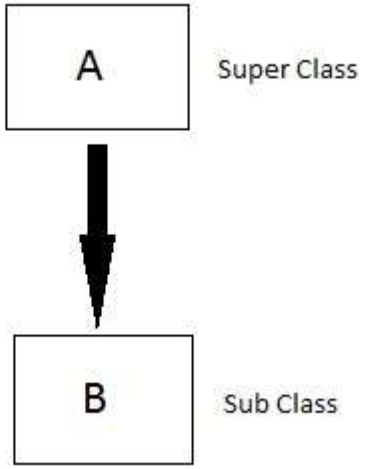
- It is a concept of inheriting the properties of one class (super class) into another class (sub class) when there “IS” a relationship between classes.
- A class should be closed for modification but open for extension. We can add additional features to an existing class without modifying it.
- Existing class -> Super Class and New class -> Sub Class.
- All members of super class will be inherited into sub class (except constructor and private members can be accessed directly in sub class).
- Super classes are generalized classes but subclasses are specialized classes.

Types of Inheritance

1. Simple Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance
5. Hybrid Inheritance

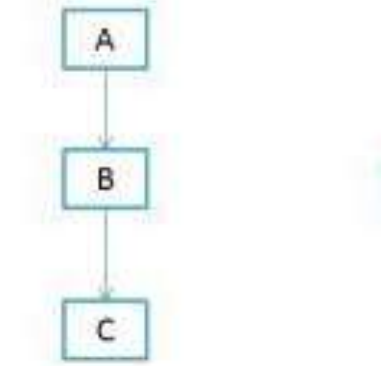
Simple Inheritance

- There will be exactly 1 super class and 1 sub class i.e. subclass will get the functionality from exactly only 1 super class.
- It is supported by class data type



Multilevel Inheritance

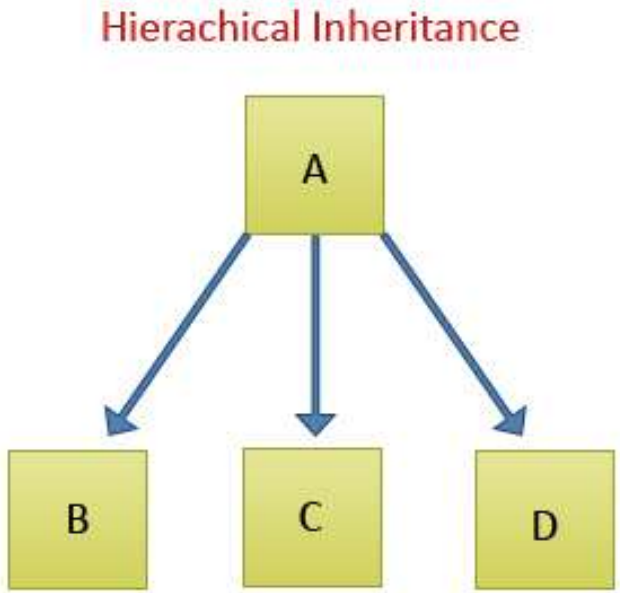
- 1 super class can have many sub class.
- 1 sub class can have many indirect super classes i.e. one is direct super class, all remaining are sub class.
- It is supported by class data type.



Multilevel Inheritance

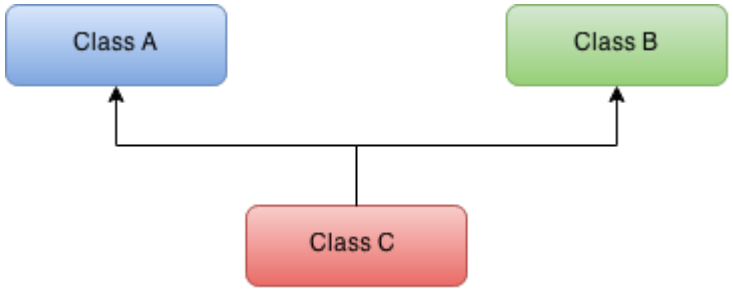
Hierarchical Inheritance

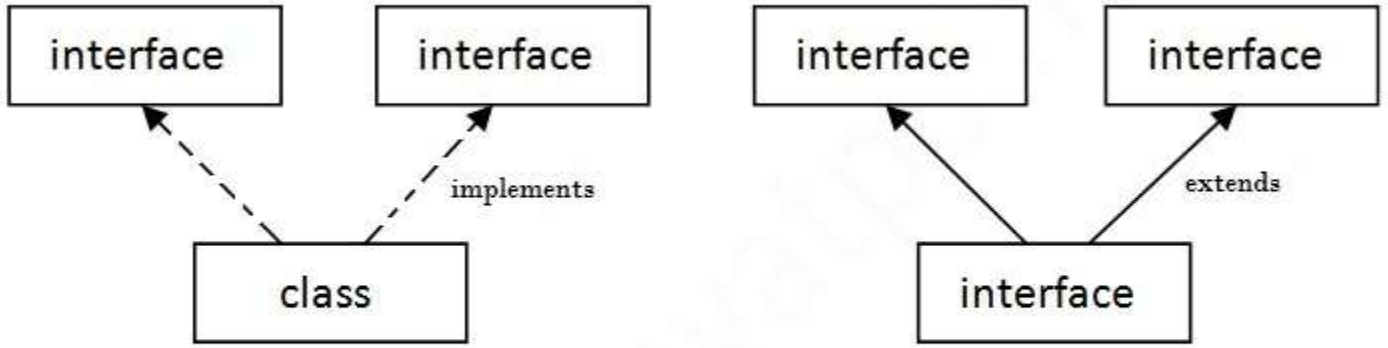
- 1 Super class have many direct sub class.
- It is supported by class data type.



Multiple Inheritance

- 1 sub class can have many direct super class.
- **Multiple Inheritance is not supported by Java with class data type**. But it is supported with ***interface*** data type.

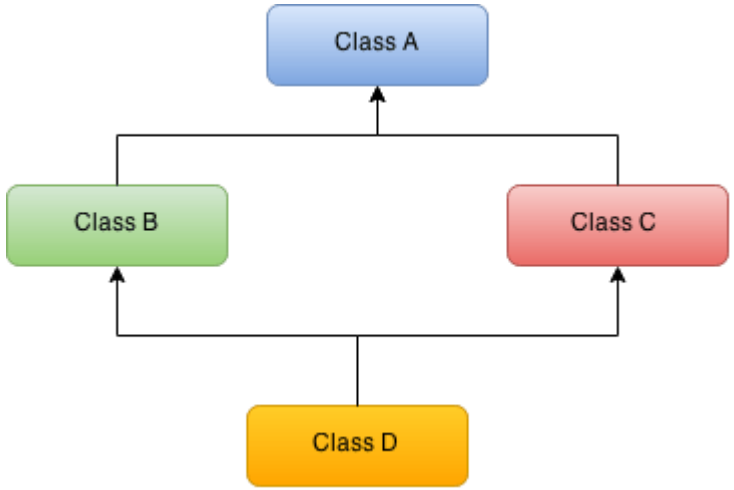




Multiple Inheritance in Java

Hybrid Inheritance

- It is the combination of multiple, multilevel and hierarchical inheritance.
- It is not supported by java with class data type. But it is supported with interface data type.



extends Keyword

- **extends** is the keyword used to inherit the properties of a class.
- **Syntax:**

```
class Super{
        .....
        .....
}
class Sub extends Super{
        .....
        .....
}
```

Super Keyword

- super is a keyword in java which should always be used in subclasses.
- super is used to access the instance members of the super class from sub class.
- super is used to access the following members of the super class from sub class.
  - Variables
  - Constructors
  - Methods

Super in Access Variables

- “super” keyword is used to resolve conflicts between super instance variables and subclass instance variables or local variables.

- It is generally used to access the super class instance variable in subclass and super class instance variables.

Super in Constructor

- super() is used to access the constructor of super class in subclass constructor.
- It is used for constructor chaining.
- It should be used in the first line of the subclass constructor.
- Constructors are executed in the order of derivation, from super class to sub class.
- It is designed in this way because while creating a sub class object the super class is initialized first, then sub
- super() can be called with arguments if you want to invoke super class parametrized constructor.
- If we don’t write super() in subclass then a default super() is added in the first line of subclass constructor during runtime by JVM.

Super to access methods

- It is used to access the method of super class in overridden method of subclass.

- The super class method is generally invoked in the subclass method for adding extra logic to the method of super class without actually modifying it.

Abstraction

- Providing essential properties and operations of an object by hiding internal things is called as *Abstraction*.
- Where there is encapsulation, there is abstraction.
- Abstraction is the quality of dealing with ideas rather than events.
- For Example, when you consider the case of e-mail, complex details such as what happens soon you send an e-mail, the protocol your email server uses are hidden from the user, therefore to send an e-mail you just need to type the content, mention the address of the receiver and click send.

Abstract Class

- It is a user-defined data type which can be used for implementing OOP.
- It is used to do partial implementation and setting the standards for subclasses to complete the implementation.
- A class that is declared abstract is called ***abstract*** class.
- Abstract class may contain both concrete and abstract methods. But if a class has an abstract method, then the class should be declared as abstract.
- An abstract method is a method i.e. declared without an implementation (without braces and followed by a semicolon).
    - Eg:- abstract void add(int x ,int y);

- If a class include abstract methods, the class itself must be declared abstract:
    - Eg:-
```
public abstract class Hello {
abstract void add(int x,int y);
}
```

Abstract Class (Contd...)

- An abstract method should not be declared as static and final.
- An abstract class also should not be declared final.
- Abstract classes cannot be instantiated, but we can declare a reference variable of it.
- When an abstract class is subclassed, the subclass must override the abstract method and provide implementation for all of the abstract methods in its parent class. If it does not, the subclass must also be declared as abstract.
- An abstract class can have the following members:
    - Static variables, blocks and methods.
    - Instance variables, blocks and methods. (Abstract class can have state).
    - Constructor.

Polymorphism

- One form behaving differently in different situations is called ***polymorphism***.
- It is extensively used in implementing ***Inheritance***.
- It helps in **static** and **dynamic** binding.
- Following concepts demonstrate different types of polymorphism in java.

    1) **Method Overloading**

    2) **Method Overriding**

Method Overloading

- It is possible to define two or more methods of same name in a class, provided that there argument list or parameters are different. This concept is known as ***Method Overloading***.
- **Example:-**

Ret type= void **Param = int**	**Ret type= int** **Param = int**	**Ret type= void** **Param = double**

void add(int a,int b)
{
System.out.println((a+b));
}
	int add(int a,int b)
	{
	int c=a+b;
	System.out.println(c);
	return c;
	}
		void add(double a, double b)
		{
		
		System.out.println((a+b));
		
		}

d.add(2.5,4.6); d.add(3,0); int x=d.add(2,2); // 3 different stmts which will invoke which method?

Rules for Method Overloading

1.Overloading can take place in the same class or in its sub-class.

2.Constructor in Java can be overloaded

3.Overloaded methods must have a different argument list.

4.Overloaded method should always be the part of the same class(can also take place in sub class), with same name but different parameters.

5.The parameters may differ in their type or number, or in both.

6.They may have the same or different return types.

7.It is also known as ***Compile Time Polymorphism***.

Method Overriding

- Implementing the super class method in the subclass with same name, same signature (number, order and type of parameters) and same return type is called ***Method Overriding.***
- Method Overriding is done to add/change functionality to the method present in super class.
- Method Overriding helps for achieving dynamic polymorphism.
- Only instance methods and inherited methods can be overridden.
- private, static and final methods cannot be overridden.

Super Class	Sub Class

void display()
{
S.o.p("Display method in super class");
}
	void display()
	{
	S.o.p("Display method in sub class");
	}

Dynamic Polymorphism

- Binding subclass object with super class reference variable is called as ***Dynamic polymorphism***.
- A super class reference variable can refer to subclass object but what members can be accessed from subclass object depends upon the type of super class reference variable.
- But if the methods are overridden it always depends on the type of subclass object, the super class reference variable is referring to. This concept is called ***Dynamic Polymorphism***.
- Following rules to achieve dynamic polymorphism:
  - There must be method overriding.
  - Subclass object must be assigned to super class reference variable.
- It means “when a subclass object is assigned to a super class reference variable, the super class reference will call subclass overridden method”.

interface

- It is a user defined data type which can be used for implementing OOP.
- It does not contain any implementation.
- It is used for setting the standards for the subclasses to do the implementation.
- It contains only 2 members:
  - public static final variables.
  - public abstract methods.
- It cannot contain any other following members:
  - Instance variables.
  - Static and instance methods.
  - Concrete methods.
  - Constructor.

Interface (Contd...)

- **Syntax:-**

```
interface name
{
        datatype variableName=value;
        returntype methodname(parameter list);
}
```

- **Example:-**

```
interface Hello
{
        int x=10;
        int add(int x,int y);
}
```

Interface (Contd..)

- Each method in an interface is also implicitly public and abstract.
- All variables declared in an interface are by default public static final variables and have to be initialized during declaration.
- When a class is implementing the interface, then that class must override all the abstract method in the interface otherwise the class should be declared as abstract.
- When a class is implementing interface, than all final static variables of the interface will be inherited into subclass.

Implements Keyword

- “implements” keyword is used to interlink class and interface.
- Example:-

```
interface Hello
{
        int x=10;
        int add(int x,int y);
}
class HelloImpl implements Hello
{
        public int add(int x,int y)
        {
                int z=0;
                z=x+y;
                return z;
        }
}
```

Interfaces (Contd..)

- Interface supports multiple inheritance. One class can implement may interfaces. One interface can extend any number of interfaces.
- It supports dynamic polymorphism. It provides very high level of decoupling and abstraction.
- It supports abstraction to a greater extent.
- Interfaces cannot be instantiated, but it can have a reference variable.
- It is open completely for implementation and does not contain any implementation.

Packages

- Packages are containers or are used for storing all type of java user defined data types and sub packages.
- Package is used for organizing set of related classes and interfaces.
- Packages are used for storing the following types:-
  - Class
  - Subclass
  - Abstract class
  - Interface
  - Annotation
  - Enumeration
  - Sub Packages

Packages (Contd...)

- Packages are created with the keyword called “package”.
- The package statement should be the first line in the source file.
  - 2 package statements are not allowed in same source file.
- **Syntax:-** package packagename;
- **Example:-** package a;

  package a.b;

  package com.sun;
- Packages are used to avoid naming conflicts between 2 classes. (when both class have same name).
- java & javax are 2 main packages which are provided in java.

“import” Statement

- import is a keyword in java.
- import statement should be used immediately after package statement and before any class definition.
- Generally a class has stored in one package has to be used in another package with a fully qualified class name.
- Fully qualified class name means using a class with its package name:
  - java.util.LinkedList lnk=new java.util.LinkedList;
- import keyword helps in importing a class stored in one package to be used in another package.
  - To import all the classes inside the util package:
    - **import java.util.*;**
  - To import particular class inside the util package:
    - **import java.util.LinkedList;**

Exception Handling

- Exception: Runtime error which causes the program to terminate but it can be handled.

- Exception Handling: Mechanism of handling runtime error and maintaining the normal execution of the program.

- There are 5 keywords used in Exception handling in java:-
  - try
  - catch
  - finally
  - throws
  - throw

Exception Handling (Contd..)

try

- In a method, write the code which is subject to create an exception or error inside a try block.
- try should be used with catch or finally or both.

catch

- catch block is used to catch the exception created in try block and fix the error.
- catch must be used with try.
- One try block can have multiple catch statements.

try/catch

Syntax:-

```
try
{
    //statements that may cause an exception
}
catch (exception(type) e(object))
{
    //error handling code
}
```

Example:-

```
 try
{
            int a=5;
            res=a/0;
            S.o.p("Result is: "+res);
}
catch(Exception e)
{
            S.o.p("Exception Caught");
}
```

try/multiple catch

Syntax:-

```
try
{
    //statements that may cause an exception
}
catch (exception(type1) e(object))
{
    //error handling code
}
catch (exception(type2) e(object))
{
   //error handling code
}
```

Example:-

```
 try
{
            int a=5;
            res=a/0;
            S.o.p("Result is: "+res);
}
catch(ArithmeticException e)
{
S.o.p("Exception Caught in catch1");
}
catch(Exception e)
{
S.o.p("Exception Caught in catch2");
}
```

finally block

- If we want to execute some statements without fail, write the code in finally block.
- It is used to clear method level resources.
- finally is guaranteed to execute, whether exception is raised in try block or not even if no correct catch block is found with try.
- return statement in try/catch block doesn’t stop finally block execution.
- Only System.exit() method can stop the execution of finally block.

try/finally

Syntax:-

```
try
{
    //statements that may cause an exception
}
finally
{
//clear method level resource code
}
```

try/catch/finally

Syntax :-

```
 try
{
   //statements that may cause an exception
}
catch (exception(type1) e(object))
{
//Exception caught
}
finally
{
                //clear method level resource code
}
```

throws

- It is used at method level to propagate the exception object to the caller method.
- It is mandatory to use ***throws*** if the method is throwing checked exceptions to the caller method.
- ***Example:-***

```
void mymethod(int num)throws IOException, ClassNotFoundException
{
        if(num==1)
                throw new IOException("Exception Message1");
        else
                throw new ClassNotFoundException("Exception Message2");
}
```

throw

- throw is a key word used to deliberate create an exception object inside a method.
- If catch block is present in the same method then it will be caught.
- If catch block is not present, then that exception object is thrown to the caller method.
- throw keyword can be used with only subclasses of class Throwable.

throw example

```
static void checkEligibilty(int stuage, int stuweight){
    if(stuage<12 && stuweight<40) {
       throw new ArithmeticException("Student is not eligible for registration");
    }
    else {
       System.out.println("Entries Valid!!");
    }
 }
```

Difference between throw and throws

throw	throws
1. Java throw keyword is used to explicitly throw an exception	1. Java throws keyword is used to declare an exception.

2. void m(){
throw new
ArithmeticException("sorry");
}
	2. void m()throws ArithmeticException{
	//method code
	}

3. Checked exception cannot be propagated using throw only.	3. Checked exception can be propagated with throws.
4. Throw is followed by an instance.	4. Throw is followed by a class.
5. Throw is used within the method.	5. Throws is used with the method signature.
6. You cannot throw multiple exceptions.	6. You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

Thread.sleep()

Example:-

```
try
{
   Thread.sleep(1000);              //1000 milliseconds is one second.
}
catch(InterruptedException ex)
{
   Thread.currentThread().interrupt();
}
```

- **To halt an execution for few milliseconds, we make use of sleep() in Thread class.**

Scanner

- It is present in java.util package.

- In this class, various methods have been provided to read the values in the required format from various devices like keyboard,file etc.

- When reading a file the Scanner class breaks the input into tokens and uses whitespaces as delimiters.
  - nextInt():- reads as integer values.
  - nextDouble():- reads as double values.
  - next():- reads as String values.

- Some of the methods present in Scanner class to read values are :-

Collections

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

  - **Implementations:** 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. In essence, algorithms are reusable functionality.

Types of Collection

- There are 3 types of Collection. They are:
  - **Set:** No Duplicates; No Index; Random allocation of memory.
  - **List:** Duplicates allowed; Proper allocation/arrangement
  - **Maps:** The Map interface maps unique keys to values.

Types of Set

- There are 3 types of Set. They are:
  - ✓**Hash Set:** No duplicates; No index; Random allocation of memory.
  - ✓**Linked Hash Set:** No duplicates; Proper allocation/arrangement.
  - ✓**Tree Set:** No duplicates; Ascending order rearrangement; Implements a set stored in a tree.

Types of List

- There are 2 types of List. They are:
  - ✓**Linked List:** Index starts with 0. addFirst(), addLast(), remove(object), remove(key), removeFirst(),removeLast().
  - ✓**Array List:**
    - Array List are created with an initial size.
    - When the size is exceeded, the collection is automatically enlarged.
    - When objects are removed, the array may be shrunk.

ArrayList

- ArrayList is a variable length Collection class.
- ArrayList allows you to use Generics to ensure type-safety
- ArrayList stores values sequentially, but the values can be accessed randomly.
- Values will be stored in the same order as inserted.
- The elements in ArrayList can be accessed by Iterator and ListIterator.
- ArrayList elements can be accessed randomly because it is implementing RandomAccess marker interface.

Map

- The Map interface maps unique keys to values.
- A key is an object that we use to retrieve a value at a later date.
- Given a key and a value, we can store the value in a Map object.
- After the value is stored, we can retrieve it by using its key.
- No duplicate keys allowed, but values can be duplicates.
- If keys duplicated, then latest value will be displayed.

Iterator

- Iterator is a collection framework interface.
- We can visit the elements of a collection only in the forward direction.
- We can remove the elements in collection using Iterator.
- It is available to all the classes in collection framework.

ListIterator

- ListIterator is a collection framework interface.
- We can iterate both in forward and reverse direction through a collection.
- We can add, remove and replace the elements of a collection using ListIterator.

Class Array

- Array class is present in java.util package.
- This class provides various methods that are useful when working with arrays.
- Arrays class provides various methods to search, sort, fill etc in an array.