**Brief Introduction About Java**

History of Java:-

* Java programming language was originally developed by “sun Microsystems” which was initiated by James Gosling and released in 1995.
* Java is guaranteed to be Write Once, Run Anywhere(WORA).
* Originally designed for small, embedded systems in electronic appliances like step-top boxes.
* Firstly, it was called “Greentalk” by “James Gosling” , and file extension was .gt
* After that, it was called oak and it was developed as apart of the “Green Project”.
* Why oak? Oak is a symbol of strength and chosen as a national tree of many countries like U.S.A, France, Germany, Romania, etc.
* In 1995, Oak was renamed as “Java” because it was already a trademark by “oak Technologies”.

What is Java uses:-

* Mobile App Development
* Desktop GUI Applications
* Web-Based Applications
* Gaming Applications
* Big Data Technologies
* Distributed Applications
* Cloud-based Applications
* IOT Applications

Features of Java:-

* Object-Oriented
* Multithreading
* Interpreted
* Platform Independent
* Portable
* Distributed
* Architectural Neural
* Dynamic
* Robust
* High Performance
* Secured
* Simple

Java Architecture:-

Java File

Machine object code

Byte code



Data Types in Java:-

A data type is a set of values and a set of operations defined on those values.

**Syntax**:-Datatype-name variable name = values;

* **They are divided into two types**:-

1. Primitive Data type
2. Non-primitive Data type

1)**Primitive Data type**:-

**Again primitive divided into two types** :-

1)Boolean

2)Numeric

1)Boolean(1 byte – 8 bits):-

* In Boolean we will store only two types of values either true or false
* True is defined by 1, false is defined by 0.

Syntax:- Boolean variableName = booleanValue;

Ex:- Boolean a = true;

2)Numeric:-

Again Numeric divided into 2 types

1. Character
2. Integer

1)**Character(2 bytes – 16bits**):-

It will stores only single character of value.

Syntax:- char VariableName = charValue;

Ex:- char name = ‘a’;

2)Integer:-

Integer divided into 2 types

1. Integral
2. Floating-Point

1)**Integral**:-

Integral will stores the integer values.

There are four datatypes in Integral

1. Byte
2. Short
3. Int
4. Long
5. Byte(1 byte – 8 bits):-

The smallest integer type is byte.

**Syntax**:-

Byte variableName = value;

Ex:- byte a = 10;

1. **s**hort(2 bytes – 16 bits**)**:-

It is probably the least used java type.

**Syntax**:-

Short variableName = value;

Ex:-

Short c = 6;

1. int(4 bytes – 32 bits) :-

It is mostly used data type.

Syntax:-

Int variableName = value;

Ex:-

Int a = 10;

1. long(8 bytes – 64 bits):-

It is the largest value in Integrals.

**Syntax**:-

Long variableName = value;

Ex:-

Long a = 10;

2)Floating-point:-

Floating point data type is used to store decimal type of values

1) Float

2) Double

1)Float(4 bytes – 32 bits) :-

Float will store decimal values.

**Syntax**:-

float variableName = value;

Ex:-

float a = 1.0;

2)double(8 bytes – 64 bits) :-

double will store decimal values.

**Syntax**:-

double variableName = value;

Ex:-

double a = 1.0;

2)Non-primitive:-

* + - * String
      * Array

Etc.

Reserved Memory Allocation :-

Int age = 20;



Datatype variable value



RAM

Reserved memory allocation



Variables in Java **:-**

A variables is a name assigned to a value that is stored inside the system memory.

Types of Variables :-

There are three types of variables In java

1. Local variable
2. Instance variable
3. Static variable
4. Local variable :-

Declare inside method.

Ex:-

Class main{

Void add(){

Int a = 10; //local variable

}

}

1. Instance Variable:-

Declare within class out of method.

Ex:- class main{

Int a = 10; //instance variable

Void add(){

}

}

1. Static Variable:-

Variables which is declare with the static keyword

Syntax:- static datatype variableName = value;

Ex:-

Static int a = 10;

Keywords In java:-

In java programming language, a keyword is any one of 67 reserved words, that have a predefined meaning in the language. Because of this programmers cannot use keywords in some context. From these keywords 16 are contextually reserved.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Int | Abstract | throw | Throws | New | Void |
| Long | Class | Try | Finally | Public | Volatile |
| byte | If | Catch | Default | Protect | Native |
| Short | For | Extends | Switch | Private | Synchronized |
| Floating | Else if | Import | Case | Strictfp | Enum |
| Double | Else | Implements | Const | This | Transient |
| Char | While | Break | Goto | Interface | Enum |
| String | Do while | Continue | instanceOf | package | Final |
| super | return |  |  |  |  |

Java Naming conventions :-

* Java Naming convention is a rule to follow as you decide
* While developing software in java for good maintenance and readability of code.

Class:-

* First should be uppercase.
* It should be noun

Ex:- Employee

Interface:-

* First letter should be Uppercase
* It should be adjective

Ex:- Sport

Method:-

* First letter should be lowercase
* Followed word first letter should be uppercase

Ex:- changedGer

Variable:-

* First letter should be lowercase
* Followed word first letter should be uppercase

Ex:- changedGer

**Package**:-

* First letter should be lowercase
* Divide words by inserting dot(.)

Ex:- java.util.lang

Methods in java:-

* Method is a collection of statements that performs some specific task.
* “code Reusability”, code optimization.

Without Return type:-

public void calculateSum(){

int c = a + b;

System.out.println( c );

}

With Return type:-

public void CalculateSum(int a, int b){

int c = a + b;

return c;

}

System . out . pritnln()



It is a class present Java.lang.package

In Java, there are four types methods:-

1. User – defined methods
2. Standard library methods
3. Abstract methods
4. Static methods
5. User-defined methods :-

We can create our own method based on our requirement.

**Ex**:-

int a = 10;

int b = 20;

public void main (){

sum = a + b;

}

Main(); //calling main method (user-defined method)

* Here main is user – defined method which is provided by user

1. Standard library methods:-

These are built-in methods in java that are available to use.

**Ex**:-

String name = “Valli”;

String name1 = “lohit”;

If (name.equals(name2)){

System.out.println(name);

}

else{

System.out.println(name1):}

* Here equals is predefined method which provided in java libraries

1. Abstract Methods:-

These are the methods that do not contains the body of the method and implements inside the abstract class.

Ex:-

Abstract class Bike{ //abstract class

Abstract void run(); //abstract

}

Class Honda extends Bike{

Void run(){

System.out.println(“Running safely”);

}

public static void main(String args[]){

Bike obj = new Honda();

Obj.run():

}

}

* Here abstract method will doesn’t had any body
* To access method we should extend that with another class.

1. Static methods:-

These are methods that are accessible without any instance of the class.

Ex:-

Public class GFG{

Static int a = 40;

Int b = 50;

void simpleDisplay(){

System.out.println(a);

System.out.println(a);

}

static Void staticDisplay(){

System.out.println(a);

}

Public static void main(String args[])

{

GFG obj = new GFG();

Obj.simpleDisplay();

staticDisplay();

}

}

Methods with/ without parameters and return**:-**

There are four types of with/without parameters and return value:-

1. With param with return value
2. With param without return value
3. Without param with return value
4. Without param with return value
5. With param with return value**:-**

public void main(int a, int b){

c = a + b;

return c;

}

1. With param without return value:-

public void main(int a, int b){

c = a + b;

System.out.println( c );

}

1. Without param with return value:-

public void main(){

c = a + b;

return c;

}

1. Without param with return value:-

Public void main(){

C = a + b;

System.out.println( c );

}

Java user input Scanner:-

The Scanner class is used to get user input and it is found in the java.util.package(Scanner).

**Input Type:-**

* nextBoolean():-

Read a Boolean value from the user

* nextByte()**:**-

Read a byte value from the user

* nextDouble()**:**-

Reads a double value from the user

* nextFloat()**:**-

Reads a float value from the user

* nextInt():-

Reads a int value from the user

* nextLine()**:**-

Reads a string value from the user

* nextLong()**:**-

Reads a long value from the user

* nextShort()**:**-

Reads a short value from the user

Type Casting in java:-

Type casting is when you assign a value of one primitive data type to another type.

Syntax:-

Datatype variableName = (datatype)VariableToConvert

There are two casting directions:-

1. Narrowing (larger to smaller type)
2. Widening(smaller to larger type)

1)Narrowing(Larger to smaller type):-

Converts a larger datatypes to small datatypes

Double->float->long->int->char->short->byte

2)Widening casting:-

Converts a smaller datatypes to large datatypes

Byte -> short ->char ->int->long->float->double

Constructor in java:-

* Constructor name must be same as it’s class name.
* Constructor must have no explicit return type
* When object is involved automatically it will call default constructor.

Type of java constructors:-

1. Default constructor
2. Parameterized constructor
3. Default constructor(No-arg constructor):-

The default constructor in java initializes the data members of the class to their default values such as 0 for int.

Example:-

Public Class Fruits{  
 Fruits(){

System.out.println(“I had many

fruits”);

}

Public static void main(String args[]){

Fruits obj = new Fruits();

}

}

Output:-

I had many fruits

1. Parameterized constructor:-

* A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

* The parameterized constructor is used to provide different values to distinct object however, you can provide the same values also.

Example:-

Class Student {

Int id;

String name;

Student(int id, String name)

{

this.id = id;

this.name = name;

}

Void display(){  
 System.out.println(id+” “+name);

}

Public static void main(String args[]){  
 Student s1 = new Student(101,”valli”);

S1.display();

}

}

Output:-

101, valli

Constructor overloading**:-**

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

Example:-

Public Class main{  
 int id;

String name;

Int age;

Main(int id, String name){  
 this.id = id;

this.name = name;

}

Main(int id, String name, int age){

this.id = id;

this.name = name;

this.age = age;

}

Void display(){

System.out.println(“ID: “+id);

System.out.println(“Name: “+name);

System.out.println(“Age: “+age);

Public static void main(String args[]){

Main obj = new main(1, valli);

Main obj1 = new main(2, lohi, 12);

Obj.display();

Obj1.display();

}

}

Output:-

ID: 1

Name: valli

ID: 2

Name: Lohi

Age: 12

**Copy Constructors:-**

‘This’ Keyword reference in java:-

‘this’ is a reference variable that refers to the current object

Without this keyword reference Example:-

Class Test{

Int a = 100;

Int b = 200;

Test(int I, int n){

a = I;

b = n;

}

}

With this keyword reference Example:-

Class Test{

Int a = 100;

Int b = 200;

Test(int a, int b){

this.a = a;

this.b = b;

}}

OOP’S in java:-

* OOP defines object oriented programming language.
* Object means a real world entity such as pen, chair, table etc. object -oriented programming is a methodology to design a program using classes and objects.
  + - * Object
      * Class
      * Inheritance
      * Polymorphism
      * Abstraction
      * Encapsulation

Class in java:-

* A class is a blueprint for creating objects.
* A class is declared only once
* The computer doesn’t allocate memory when you declare a class

Syntax:-

Class <class\_name> {  
 //field;

//method;

}

Structure of java class syntax:-

Access modifier class Name



Public class Main{

Int a; // class body

Int b;

Public void run(){ //class instance methods

}

}

Java main function syntax:-

Declaring the main() method:-

Public static void main(String args[]){

//code for main() method

}

Public:-

It’s the access specifier means from every where we can access it.

Static:-

Access modifier means we can call this method directly using a class name without creating an object of it.

Void**:**-

It’s the return type.

Main:-

method name.

String[] args:-

It accepts only String type if arguments.

Types of classes in java:**-**

There are seven types of classes in java

1. Static class
2. Final class
3. Abstract class
4. Concrete class
5. Singleton class
6. POJO Class
7. Inner class
8. Static class:-

* In java, static is a keyword that manage objects in the memory.
* The static object belongs to the instead of the instance of the class.
* We can make a class static if and only if is a nested class.
* We can also say that static classes are known as nested classes.
* It means that a class that is declared as static within another class is known as a static class.
* Nested static class does not require reference to the outer class
* The purpose of a static class is to provide the outline of it’s inherited class.

The properties of the static class are:-

* The class has only static members.
* It cannot access the member(non-static) of the outer class.
* We cannot create an object of the static class.

Example:-

public class StaticClassExample {

private static String str = “Java”;

public static class staticDemo{

public void show(){

System.out.println(str);

} }

public static void main(String args[]){

StaticClassExample.staticDemo obj = new

StaticClassExample.staticDemo();

Obj.show():

}

}

1. Final class:-

* The word final means that cannot be changed.
* The final class in java can be declared using final keyword.
* Once we declare a class as final, the values remain the same throughout the program.
* The purpose of the final class is to make the class immutable like String class.it also prevents the class from being sub-classed

Example:-

Final class A{

Void printmsg(){

System.out.println(“Base class method is executed”);

}  
 }

Class B extends A{

Void printmsg(){

System.out.println(“Derived class method is executed”);

}

}

Public Class FinalClassExample{

Public static void main(String args[]){

B obj = new B();

Obj.printmsg():

}

}

Error:-

cannot inherit from final A and B extends A

1. Abstract class:-

* An abstract class Is a that is declared with the keyword abstract.
* The class may or may not contain abstract methods.
* We cannot create an instance of an abstract class but it can be a subclass.
* These classes are incomplete, so to complete the abstract class we should extends the abstract classes to a concrete class.

Example:-

Abstract class example{

Int a = 10;

Int b = 20;

Public abstract void add()

}

Public class operations extends example{

Public void add(){

System.out.println(“sum of a and b is : “a+b);

Public static void main(String args[]){  
example obj = new obj();

Obj.add();

}

}

Output:-

Sum of a and b is: 30

1. Concrete Class:-

* These are the regular java classes.
* A derived class that provides the basic implementations for all of the methods that are not already implemented in the base class is known as a concrete class.

Example:-

Public class concreteClassExample

{

Static int product(int a, int b){

Return a \* b;

}

Public static void main(String args[]){

Int p = product(6,8);

System.out.println(“Product of a and b is:”+p);

}

}

1. Singleton class:-

* A class that has only an object at a time is known as a singleton class.
* Still, if we are trying to create an instance a second time, that newly created instance points to the first instance.

1. POJO Class:-

* In java, POJO stands for plain old java object.
* A java class that contains only private variables, setter and getter is known as POJO class.
* It is used to define java object that increase the reusability and readability of a java program.

1. It does not extends the predefined classes such as arrays, HttpServlet, etc.
2. It cannot contain pre-specified annotations.
3. It cannot implement pre-defined interface.
4. It is not required to add any constructor.
5. All instance variables must be private
6. The getter/setter methods must be public

Example:-

Class pojoDemo{

Private double price = 10.0;

Public double getPrice(){

Return price;

}

Public double setPrice(int price){

This.price = price;

}

}

Public class pojoclassExample{

Public static void main(String args[]){

pojoclassExample obj = new pojoclassExample();

System.out.println(“The price of an article

is”+obj.getprice()+”Rs.”);

}

}

1. Inner class:-

* Java allows us to defined a class within a class and such classes are known as nested classes.
* It is used to group the classes logically and to achieve encapsulation.
* The outer class members can be accessed by the inner class.

Syntax:-

Class OuterClass{

//code

Class nestedClass{

//code

}

}

The nested classes are two types:-

1) Static

2) Non-static

1. Static class:-

* A class that is static and nested is called a static nested class.
* It interacts with the instance member of its outer class.
* We can create an object of the static nested class

Example:-

OuterClass.staticNestedClass nestedObject = new OuterClass.staticNestedClass();

1. Non-static nested class:-

* Non-static nested classes are called inner classes.

Types of inner classes:-

Java provides the two types of inner classes

1. Local classes or method local inner class
2. Anonymous classes or Anonymous inner class
3. Local inner class:-

Wrapper class:-

* In java, the term wrapper class represents a collection of java classes that objectify the primitive type of java.
* It means that for each primitive type there is a corresponding wrapper class.
* The wrapper classes are used to perform the conversion from primitive type to an object and vice-versa

Example:-

Public class WrapperClassExample{

Public static void main(String args[]){

Byte x = 0;

Byte byteobj = new byte(x);

Int y = 23;

Integer intobj = new Integer(y);

Char c = ‘m’;

Character charobj = c;

System.out.println(“Byte object byteobj: “+byteobj);

System.out.println(“Integer object intobj: “+intobj);

System.out.println(“Character object byteobj: “+Charobj);

}

}

Objects in java:-

An object in java is the most fundamental unit of the object-oriented programming paradigm an object in java consists of the following

1) Identity

2) Behavior

3) State

Objects:-

* An object is a copy of a class.
* An object is a physically entity
* The keyword used in “new”.
* Object can create any number of objects using one single class.

Syntax:-

<ClassName> objectName = new className();



Class Name

Public static void main(String args[]){

<ClassName> objectName = new className();

<objectName>.<methodName>

<objectName>.<variableName>

}

Example:-

Class main{

Public void ex(){

}

}

Public class example{

Public static void main(String args[]){

Main obj = new main();

Obj.ex();

}

}

Java Heap vs stack:-

Heap :-

* It is used to store object
* JVM will throw Java.lang.outOfMemoryError, if no more heap space.
* Memory size is bigger than stack.

Stack:-

* It is used to store local variables function call.
* JVM will throw Java.lang.stackoverFlowError, if no memory left in stack.
* Memory stack is small



Heap

Obj

object

Stack

Obj(ref)

a = 10

b = 20

Public void method(){

Int a = 10;

Int b = 20;

Class obj = new class();

}

Inheritance:-

* It is possible to inherit attributes and methods from one class to another class. We can reuse field.
* Super-class:-

The class whose features are inherited.

* Sub-class:-

The class that inherits the other class

* Reusability:-

Inheritance supports the concepts of “Reusability”.

Extends Keyword:-

Extends is the keyword used to inherit the properties of a class.

Types of Inheritance:-

1. Single inheritance
2. Multilevel inheritance
3. Hierarchical inheritance
4. Multiple Inheritance
5. Hybrid Inheritance
6. Single inheritance**:-**

When a class inherits another class, it is known as single inheritance

A



B

Example:-

Class Flowers{

Void display(){

System.out.println(“Many types of flowers”);

}

}

Class rose extends flowers{

Void roseStatement(){

System.out.println(“Rose”):

}

}

Class TestInheritance{  
 public static void main(String args[]){

rose obj = new rose();

obj.display();

Obj.roseStatememt();

}

}

Output:-

Many types of flowers

Rose

1. Multilevel inheritance:-

When there is a chain of inheritance, it is known as multilevel inheritance.

A

Base class



B

Intermediatory class



C

Derived class

Example:-

Class Flowers{

Void flow(){

System.out.prinltn(“Flowers:- ”);

}

Class rose extends Flowers{

Void red(){

System.out.prinltn(“Rose”);

}

Class YellowRose extends rose{  
 Void yellow(){

System.out.prinltn(“Yellow rose”);

}

Class PinkRose extends YellowRose {  
 Void pink(){

System.out.prinltn(“Pink rose”);

}

Class BlackRose extends PinkRose {  
 Void Black(){

System.out.prinltn(“Black rose”);

}

}

Public class TestInheritance{

Public static void main(String args[]){

BlackRose obj = new BlackRose();

Obj.flow();

Obj.red();

Obj.yellow();

Obj.pink();

Obj.black();

}

}

Output:-

Flowers:-

Rose

Yellow rose

Pink rose

Black rose

1. Hierarchal Inheritance:-

When two or more classes inherits a single class, it is known as hierarchal inheritance



D

A

C

B

Example:-

Class flower{

Void flow(){

System.out.println(“Flower”){

}

}

Class Redrose extends flower{

Void red(){

System.out.println(“Red rose”){

}

}

Class Pinkrose extends flower{

Void pink(){

System.out.println(“pink rose”){

}

}

Public class TestInheritance{

Public static void main(String args[]){

Redrose obj = new Redrose();

Pinkrose obj1 = new Pinkrose()

Obj.flow();

Obj.red();

Obj.flow();

Obj.pink();}

}

Output:-

Flower

Red rose

Flower

Pink rose

4)Multiple inheritance/Hybrid Inheritance:-

To reduce the complexity and simplify the language, multiple inheritance and hybrid inheritance is not supported in java.

Example for multiple inheritance:-

Class A{

Void msg()

{

System.out.println(”hello”)}

}

Class B{

Void msg(){ System.out.println(“Welcome”)

}

Class C extends A,B{

Public static void main(String args[]){

C obj = new c();

Obj.msg();

}

}

Output:-

Compile time error

Polymorphism in java:-

* Polymorphism in java is a concept by which we can perform a single action in different ways
* There are two types of polymorphism

1) compile-time

2) Run-time

1)Compile-time polymorphism:-

* It Is also known as static polymorphism.
* Method overloading is an example of compile time polymorphism.

Example:-

Public int sum(int x, int y){

return (x+y);

}

Public int sum(int x, int y, int z){

Return (x+y+z);

}

2)Run-time polymorphism:-

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

Example:-

class Animal{

void cat(){

System.out.println(“I am ani”);

}

}

Class cat extends animal{

Public int cats(){

System.out.println(“I am cat”);

}

}

Abstraction in java:-

* A class which is declared with the abstract keyword is known as an abstract class in java.
* Abstract class cannot have object
* It can have abstract and non-abstract methods.
* It can have constructors and static methods also.

Abstraction in java:-

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

Ways to achieve Abstraction :-

There are two ways to achieve abstraction in java

1. Abstract class
2. Interface
3. Abstract class:-

A class which is declared as abstract is known as an abstract class.

Example:-

Abstract class shape{

Abstract void draw(){

}

}

Class Rectangle extends shape{

Void draw(){

System.out.println(“Draw rectangle”);

}

Public static void main(String args[]){

Example obj= abstractExample();

Obj.main();

}

}

1. Interface:-

In interface we will have abstract methods. By using interface we can hide data.

Example:-

Public interface example{

Public static void add(){}

Public static void sub(){}

Public static void mul(){}

Public static void div(){}

}

Public class main extends example{

Public static void add(){

Int a = 10;

Int b = 20;

Int sum = a + b;

System.out.println(“sum: “+sum);

}

Public static void sub(){}

Public static void mul(){}

Public static void div(){}

Public static void main(String args[]){

Main.add();

}

}

Output:-

Sum: 30

Encapsulation in java:-

* Encapsulation simply means binding objects state(fields) and behavior(methods)) together.
* If a data member is private it means it can only be accessed within the same class
* No outside class can access private data members.

How to implements encapsulation in java:-

* Make the instance variable private so that they cannot be access directly from outside the class.
* We can only set and get values of these variables through the methods of the class.
* The java bean class is the example of a fully encapsulated class

Example:-

Public class encapsulationexample{

private int ssn;

private String name;

private int empage;

Public int getEmpssn(){

return ssn;

}

Public String getEmpName(){

return empName;

}

Public int getEmpAge (int newValue){

return empAge;

}

Public int setEmpssn(String newValue){

Ssn = newValue;

}

Public int setEmpName(int newValue){

empName = newValue;

}

Public int setEmpAge(){

empAge = newValue;

}

}

Public class encapsTest{

Public static void main(String args[]){

Encapsulationexample obj = new encapsulationexample();

Obj.setEmpssn(101);

Obj. setEmpName (“vali”);

Obj. setEmpAge (21);

System.out.pritnln(“Employee SSN: “+Obj.setEmpssn);

System.out.pritnln(“Employee Name:

“+Obj.setEmpName);

System.out.pritnln(“Employee Age: “+Obj.setEmpAge);

}

}

Access modifiers in java:-

* Access modifier restrict the access of a class, constructors , data members and methods in another class.

There are four types of access modifiers in java:-

1. Default
2. Public
3. Private
4. Protected
5. Default:-

The access level of a default modifiers is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

1. Public:-

The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package

1. Private:-

The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

1. Protected:-

The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

Java Access Modifiers:-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access  Modifier | Within  class | Within  package | Outside  Package  By  Subclass  only | Outside  package |
| private | Y | N | N | N |
| Default | Y | Y | N | N |
| Protected | Y | Y | Y | N |
| Public | Y | Y | Y | y |

Operators in java:-

Operators in java is a symbol that is used to perform operations

There are many types of operators:-

1. Arithmetic
2. Assignment
3. Comparison
4. Logical
5. Ternary
6. Instance

Common values for examples:-

a = 10

b = 20

1)Arithmetic operators:-

|  |  |  |
| --- | --- | --- |
| Operators | Uses | Example |
| + | Addition | C = a + b->30 |
| - | Subtraction | C = a – b->-10 |
| \* | Multiplication | C = a \* b->200 |
| / | Division | C = a / b->2 |
| % | module | C = a % b ->0 |
| ++ | Increment | a++->11 |
| -- | decrement | a-- = 9 |

2)Assignment operators:-

|  |  |
| --- | --- |
| Operators | Example |
| =+ | a+=b->a = a+b |
| =- | a-=b->a = a-b |
| =\* | a\*=b->a = a\*b |
| =/ | a/=b->a = a/b |
| = | a=b->a = b |

3)comparison operators:-

|  |  |  |
| --- | --- | --- |
| Operators | Uses | Example |
| == | Equals to | a==b ->false |
| != | Not equals to | a!=b ->true |
| <= | Lesser than or equals to | a<=b ->true |
| >= | Greater than or equals to | a>=b ->false |
| > | Lesser than | a>b ->false |
| < | Greater than | a<b ->true |

4)Logical operators:-

|  |  |  |
| --- | --- | --- |
| Operators | Uses | Example |
| && | AND(two statements should be true) | a!=b && a<=b  true |
| || | OR(any one statement should be true) | a!=b || a>=b  true |
| ! | NOT(if statement is true prints false, vise-versa) | a==b  true |

5)Ternary operators:-

* Java ternary operator is used as one line replacement for if – then – else statement

Syntax:-

Variable : (condition)? Expression1 :expression 2

Example:-

Int c : (a!=b)? true: false

Output:

true

6)Instance operators:-

The java instanceof operator is used to test whether the object is an instance of the specified type(class or subclass or interface).

Example:-

Class simple{

Public static void main(string args[]){

Simple s = new simple();

System.out.println(s instanceOf simple);

}

}

Loops in java:-

The java for loop is used to iterate a part of the program several times.

There are three types of loops:-

1. for loop
2. while loop
3. do-while loop
4. for loop:-

Example:-

Public class example{

Public static void main(string args[]){

for(int i = 0; i<=10; i++){

System.out.println(i);

}

}

}

Output:-

1 2 3 4 5 6 7 8 9

2)for – each loop:-

Syntax:-

for(type var : array){

\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

}

Example:-

Int arr[] = {1,2,3,4,5,6,7}

for(int I : arr){

System.out.println(i);

}

Output:-

1 2 3 4 5 6 7

3)while loop:-

Public class whileExample{

Public static void main(String args[]){

i = 1;

while(i<=10){

System.out.println(i);

i++;

}

}

4)do-while loop:-

Syntax:-

do{

}

While(condition)

Example:-

Public class doWhileExample{

Public static void main(String args[]){

i= I;

do{

System.out.println(i);

I++;

}

While(i<=10)

}

}

Switch-cases:-

* The java switch statement executes one statement from multiple conditions.
* It is like if-else-if ladder statements

Syntax:-

Switch(expression){

Case value1:

Break;

Case value2:

Break

………..

Default:

Code to be executed if all cases are not matched

Example:-

Int value;

System.out.println(“Enter value: ”);

Value = sc.nextInt();

Switch(value){

Case 1:

System.out.println(1);

break;

Case 2:

System.out.println(2);

break;

Case 3:

System.out.println(3);

break;

Case 4:

System.out.println(4);

break;

default:

System.out.println(“Default value”);

Output:-

Enter value : 2

2

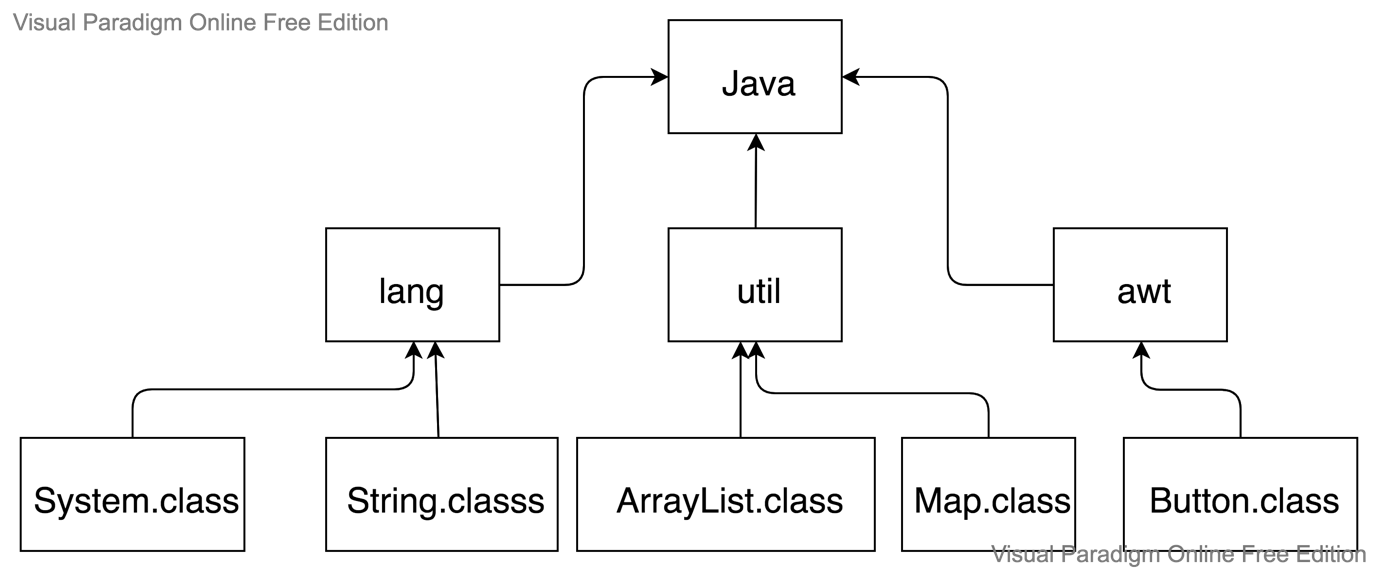
Java Package:-

* A java package is a group of similar types of classes, interface, and sub-package.
* Package in java can be categorized in two form built-in package and use-defined package.

Advantages of java package:-

* Java package is used to categorize the classes and interfaces so that they can be easily maintained
* Java package provides access protection
* Reusability





Java Strings:-

* String is a sequence of character
* Java.lang.string class is used to create a string object.

We can create string in two ways:-

1)String literals

2)By new keyword

1. String literals:-

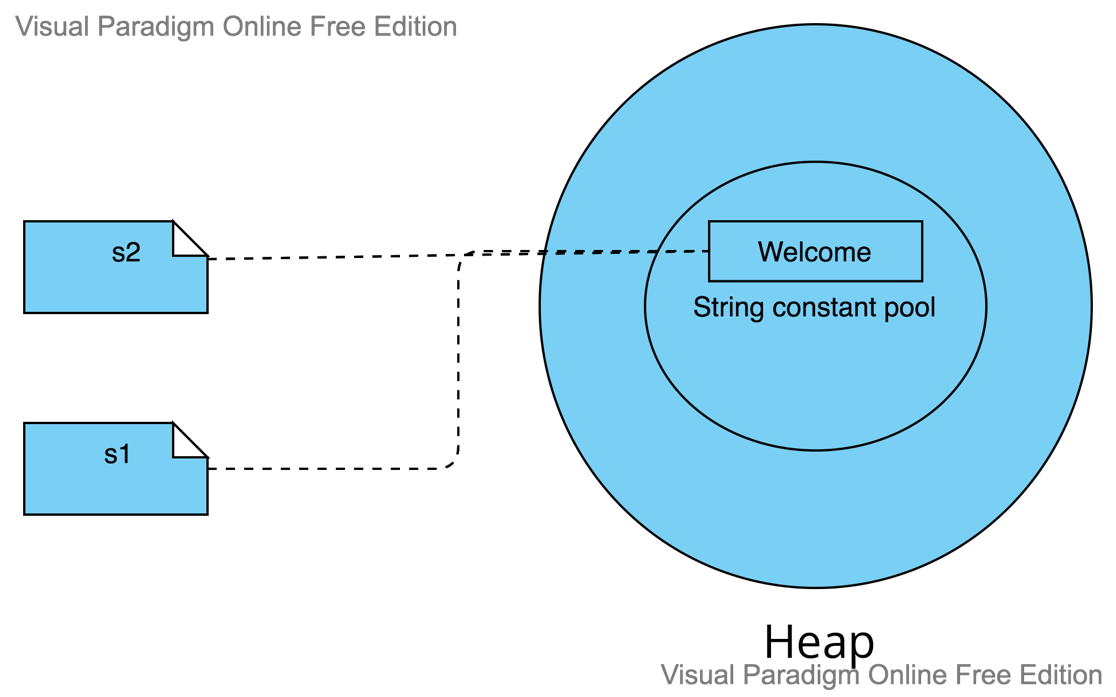
String s = “name”;

* Each time you create a string literal, the JVM

checks the “String constant pool” first. If the string already exists in the pool, a reference to the pooled instance is returned

String s1 = “welcome”;

String s2 = “welcome”;



1. By new keyword:-

String s = new String(“Welcome”):

* In such case, JVM will create a new string object in normal(non-pool) heap memory.
* The variable s will refer to the object in a heap (non-loop)

Example:-

char ch[] = {‘s’,’t’,’r’,’i’,’n’,’g’}

String s2 = new String(ch);

//converting arr to str

String s3 = new String(example); //creating java string by new keyword

Immutable String:-

In java String is immutable(unmodifiable) or (Unchangeable).

String :- Build in function

1. Equals()
2. Concat()
3. charAt(indexNumber)
4. equalsIngnoreCase()
5. toUpperCase()
6. toLowerCase()
7. trime()
8. substring()
9. length()
10. endwith()
11. isEmpty()

Example:-

String s1 = “valli ”;

String s2 = “lohi”;

System.out.println(“Equals: ” +s1.equals(s2));

System.out.println(“Concatenate: “ +s1.concat(s2));

System.out.println(“CharAt: ” +s1.charAt(3));

System.out.println(“EqualsIgnoreCase:” +s1.equalsIgnoreCase(s2));

System.out.println(“To Upper case:“ +s1.toUpperCase());

System.out.println(“To lower case: “ +s1. toLowerCase());

System.out.println(“Trime: “ +s1.trime());

System.out.println(“Substring: “ +s1.substring(0,3));

System.out.println(“Length: “ +s1. Length);

System.out.println(“Endswith: “ + s1. Endwith(“li”));

System.out.println(“IsEmpty: “ + s1. isEmpty());

Output:-

Equals: false

Concatenate: vallilohi

CharAt: l

EqualsIgnoreCase: false

To Upper case: VALLI

To lower case: valli

Trime: valli

Substring: val

Length: 5

Endswith: true

IsEmpty: false

String Buffer:-

* Java StringBuffer class is used to create mutable(modifiable) string
* The StringBuffer class in java is a same as String class except it is mutable i.e;
* Allows only one thread at a time.

Methods in StringBuffer:-

1. Append()
2. Insert()
3. Replace()
4. Delete()
5. Reverse()
6. Delete()
7. Methods in string
8. etc…

Example:-

StringBuffer s1 = new StringBuffer(“valli”);

StringBuffer s2 = new StringBuffer(“lohi”);

System.out.println(“Append: “ +s1.append(“lohi”));

System.out.println(“Insert: “ +s1.insert(2, “a”));

System.out.println(“Replace: “ +s1.replace(“l”, “a”));

System.out.println(“Reverse: “ +s1.reverse());

Output:-

Append: vallilohi

Insert: vaalli

Replace: vaaai

Reverse: illav

String builder:-

It is same as StringBuffer but it allows multiple threads at a time.

Exception handling:-

An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program’s instructions.

There are two types of errors:-

1. Compile-time errors
2. Run-time errors

1. Compile-time errors:-

Compile time error can be again classified again into two types

1. Syntax errors
2. Semantic errors
3. Syntax errors:-

Instead of declaring int a; you mistakenly declared it as in a; for which compiler will thrown an error.

1. semantic errors:-

The semantic error can arises using the wrong variable or using wrong operator or doing operation in wrong order.

Example:-

Int a = “hello”;

String s = “a”;

Int a = 5 – s;

2)Runtime error:-

* A runtime error is called an exception error . It is any event that interrupts the normal flow of program execution.
* Example for exception are, arithmeticexception, nullpointerException, DividedByZeroException, etc.

An exception can occur for many different reasons, including the following:-

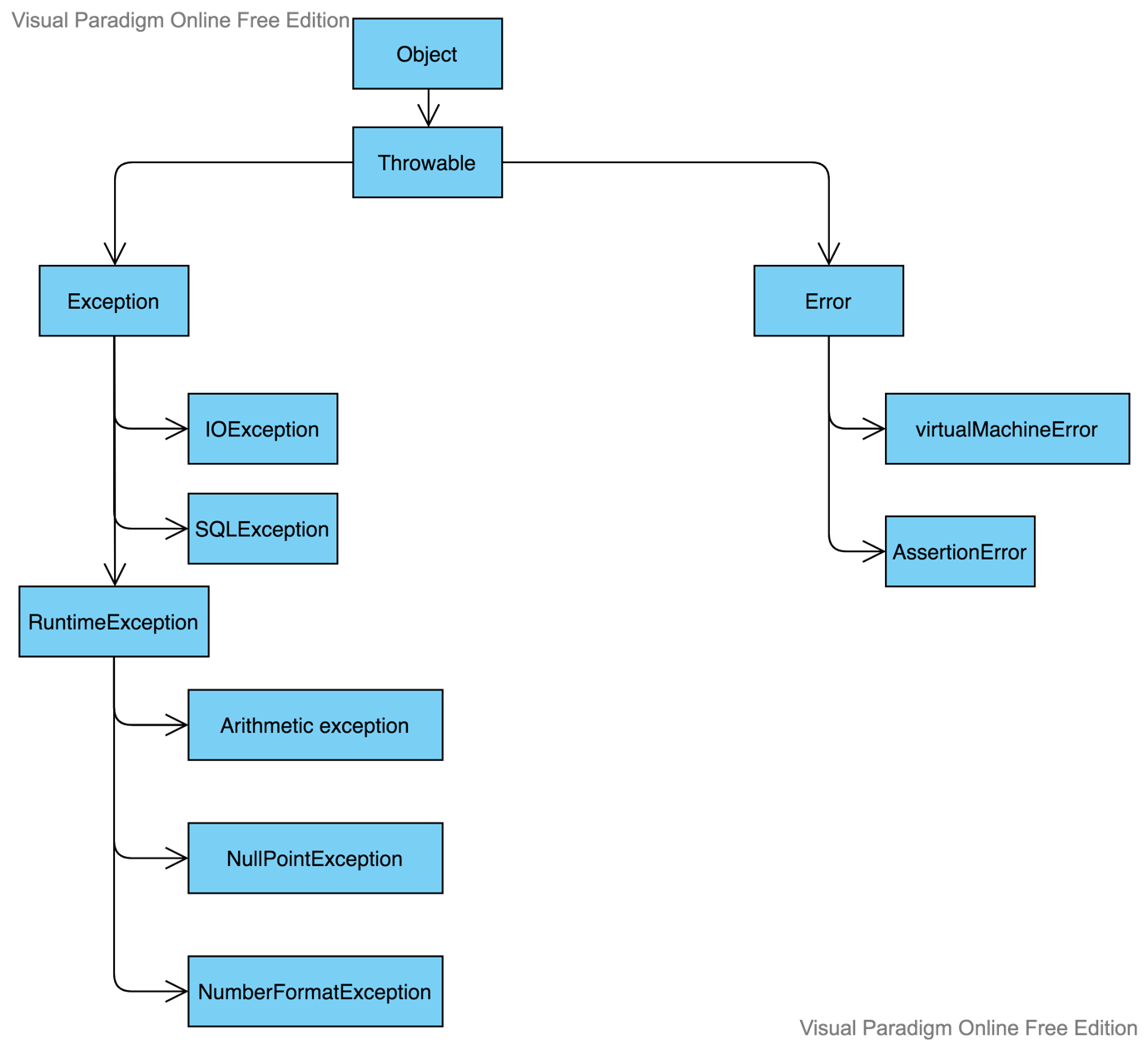
* A user has entered invalid data.
* A file object cannot be closed properly.
* A network connection has been lost in the middle of communication or the JVM has run out of memory.

Advantages of exception handing :-

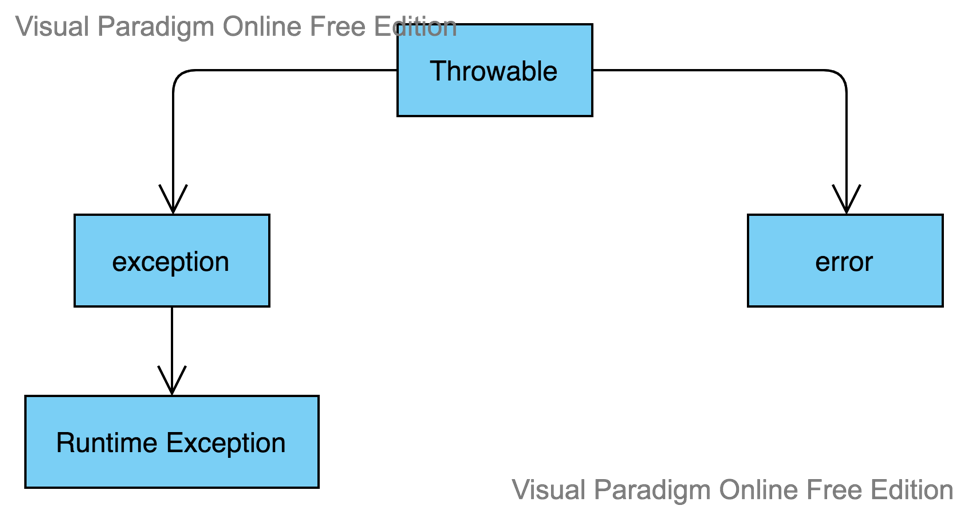
Exception handling is to maintain the normal flow of the application.

What is exception handling:-

Exception handling is a mechanism to handle runtime errors such as classNotFound, IO, sql, Remote



Hierarchy of exception classes:-



Types of Exceptions in java:-

1. Checked exception
2. Unchecked exceptions
3. Error
4. Checked exception:-

The classes that extends throwable class except runtime exception and error are known as checked exception

* IOException
* SQLException etc.

1. Unchecked Exception:-

The classes that extends RuntimeException are known as unchecked exception.

* ArithmeticException
* NullPointerException
* ArrayIndexOutOfBoundsException
* NumberFormateException etc.

1. Error:-

Error is irrecoverable e.g

* outOfMemoryError
* VirtualMachineError

Use of try-catch block in exception handling :-

Five keywords using in exception handling

* Try
* Catch
* Finally
* Throw
* throws

1. Try/catch Block:-

Try:-

Java try block is used to enclose the code that might throw an exception. It must be used within the method. If an exception occurs at the particular statement in the try block, the rest of the block code will not execute.

Catch:-

Each catch block is an exception handler that handles that handles the type of exception indicated by its argument

Syntax:-

Try{

//protected block

}

Catch(ExceptionName e){

//catch block

}

Example:-

Int I =10;

Try{

If(100/0){

System.out.println(10)

}

}Catch(ArithmeticException e){

System.out.println(error);

}

System.out.println(“rest of the code”);

Multiple catch blocks:-

Syntax:-

Try{

//protected block

}

Catch(ExceptionName e){

//catch block

}

Catch(ExceptionName1 e1){

//catch block

}

Catch(ExceptionName2 e2){

//catch block

}

The finally keyword:-

* The finally keyword is used to create a block of code that follows a try block.
* A finally block of code always executes, whether or not an exception has occurred.

Syntax:-

Try{

//protected code

}

Catch(ExceptionName e){

//catch block

}

Catch(ExceptionName1 e1){

//catch block

}

Catch(ExceptionName2 e2){

//catch block

}

Finally{

//The finally block always executes

}

printStackTrace():-

The method printStackTrace() helps us to find the exact error location and name of method causing the error in our code.

Throw vs Throws:-

|  |  |
| --- | --- |
| Throw | Throws |
| 1. Java throw keyword is used to explicitly throw an exception | 1)Java throw keyword is used to declare and exception |
| 1. Checked exception cannot be propagated using throw only | 2)Checked exception can be propagated with throws |
| 1. Throw is followed by an instance | 3)Throws is followed by class |
| 1. Throw is used within the method | 4)Throws is used with the methods signature |
| 1. You cannot throw multiple exception | 5)you can declare multiple exception  Ex:-  Public void method() throws IOException, SQLException |

Throws example:-

Class TestNumber{

Public static void main(String args[])throws Exception{

String s = null;

System.out.println(s.length());

}

}

Throw: user-define exception:-

* We have to create our own exception using throw keyword

Example:-

Public class TestThrow{

Public static void checkNum(int num){

If(num<1){

Throw new ArithmeticException(“\n Number is negative, cannot calculate square”);

}

Else{

S.O.P(“Square of” +num+ “is”+(num \* num));

}

}

Public static void main(String args[]){

testThrow obj = new testThrow();

obj.checkNum(-3);

System.out.println(“Rest of the code”);

}

}

Collections in java:-

What is collection?

* Collection in java can be referred to an object that collects multiple elements into a single unit. It is used to store, fetch and manipulate data.
* The java collection framework is a collection of interface and classes which helps in storing and processing the data efficiently.
* The java.util package contains all the classes and interfaces for collection framework.

Advantages of collections:-

* Increases performance by providing high-performance.
* Increases productivity.
* Reduce operational time.
* Versatility to work with current collection as well.

Types of collections:-

1. List:- ArrayList, LinkedList, Vector
2. Set:- Hashset, TreeSet, LinkedHashset
3. Map:- HashMap, TreeMap, HashTable, LinkedHashmap

List:-

1)ArrayList:-

* ArrayList class implements list interface and it is based on a array data structure. It is widely used because of the functionality and, flexibility it offers
* Can contain duplicate elements
* Maintain insertion order.

Java Non-Generic vs Generic Collection:-

Before Generic, we need to type cast:-

ArrayList list = new ArrayList();

After Generic, we don’t need to typecast the object:-

List<String> list = new ArrayList<String>();

Methods:-

1. Add(object o)
2. Add(int index, object o)
3. Remove(object o)
4. Set(int index, object o)
5. Int size()
6. Boolean contains(object o)
7. Clear()

Example:-

ArrayList list = new ArrayList();

List.add(1);

List.add(“Valli”);

List.add(2);

List.add(“Roopa”);

List.add(3);

List.add(“Lohit”);

System.out.println(list);

Output:-

[1, valli, 2, Roopa, 3, Lohit]

2)LinkedList:-

* LinkedList elements are not search in contiguous locations like arrays, they are linked with each other using pointer.
* add(), addFirst(), addLast() method to add the element.

Example:-

LinkedList<String> list = new LinkedList<String>();

list.add(“Valli”);

list.add(“Lohi”);

list.add(“Roopa”);

list.add(“Siddhu”);

list.add(“Durga”);

list.add(“Srinu”);

System.out.println(list);

Output:-

Valli, lohi, Roopa, Siddhu, Durga, Srinu

3)Vector:-

It is similar to ArrayList but it is poor in performance

Set:-

1. Hashset Class:-

* Use hashset to store the element.
* Contains unique elements only.
* Maintain insertion unorder only.

Example:-

Hashset list = new HashSet();

list.add(1);

list.add(“Valli”);

list.add(2);

list.add(“Lohit”);

list.add(2);

System.out.println(list);

O/P:-

[1, 2, Lohi, Valli]

2)TreeSet class:-

* Contains unique elements only like hashset.
* Maintain ascending order.

Example:-

Treeset<String> a1 = new Treeset<String>();

a1.add(“valli”);

a1.add(“Roopa”);

a1.add(“Lohit”);

System.out.println(a1);

Output:-

Lohit

Roopa

valli

3)LinkedHashSet:-

* Java linkedHashSet class contains unique elements only like Hashset.
* Java LinkedHashSet class maintain insertion order.

Example:-

LinkedHashset<String> set = new LinkedHashSet();

Set.add(“One”);

Set.add(“Two”);

Set.add(“Three”);

Set.add(“Four”);

Set.add(“Five”);

Set.add(“Six”);

System.out.println(set);

Output:-

One, two, three, four, five, six

Map Interface:-

* A map contains values based on the key i.e. key and value pairs
* Each pair is known as entry

Methods of entry interface:-

Public object getKey():- is used to obtain key

Public object getValue():- is used to obtain value

1)HashMap class:-

* A hashmap contains values based on the key
* It contains only unique element

Example:-

HashMap m = new HashMap();

m.put(“Key1”, 10);

m.put(“Key2”, 20);

m.put(“Key3”, 30);

m.put(“Key4”, 40);

m.put(“Key5”, 50);

m.put(“Key6”, 60);

m.put(“Key7”, 70);

System.out.pritnln(m);

Output:-

{key1 : 10, key2 : 20, key3 : 30, key4 : 40, key5 : 50, key6 : 60, key7 : 70}

1. TreeMap:-

* Keys maintain ascending order.
* Java TreeMap cannot have a null key but can have multiple null values.

Example:-

TreeMap<Integer, String> a1 = new TreeMap<Integer, String>();

a1.put(100, “Valli”);

a1.put(103, “Lohi”);

a1.put(102, “Roopa”);

a1.put(101, “Siddhu”);

a1.put(104, “Durga”);

System.out.println(a1);

Output:-

100, Valli

101, Siddhu

102, Roopa

103, Lohi

104, Durga

1. Hashtable:-

* Java Hashtable class implements a hashtable, which maps keys to values. It inherits Dictionary class and implements the map interface.
* Java hashtable class contains unique elements.
* It doesn’t maintain any insertion order.
* Java hashtable class doesn’t allow null key or value.

Example:-

Hashtable<Integer,String> hm = new

Hashtable<Integer,String>();

hm.put(100, “valli”);

hm.put(101, “Roopa”);

hm.put(102, “Laxmi”);

System.out.println(hm);

Output:-

{101 = Roopa, 102 = Laxmi, 100: Valli}

4)LinkedHashMap:-

* Java LinkedHashMap contains values based on the key
* Java LinkedHashMap contains unique elements.
* Java LinkedHashMap may have one null key and multiple null values.
* Java linkedHashMap maintains insertion order.

Example:-

LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>();

map.put(100,”amit”);

map.put(101,”vijay”);

map.put(102,”Rahul”);

System.out.println(map);

System.out.pritnln(map.keySet());

System.out.println(map.values());

Output:-

[100 = amit, 101 = vijay, 102 = Rahul]

[100, 101, 102]

[amit, vijay, Rahul]

Two ways to iterate the elements of collection in java:-

* By iterator interface
* By for-each loop

Iterator Interface:-

* Iterator interface provides the facility of iterating the elements in forward direction only.

Methods of iterator interface:-

* **Public Boolean hasNext()** if returns true if iterator has more elements
* **Public object next(**) if return the elements and moves the cursor pointer to the next elements.

Example:-

ArrayList a1 = new ArrayList();

a1.add(“Valli”);

a1.add(“Roopa”):

a1.add(“Lohit”);

a1.add(“Siddhu”);

System.out.println(a1);

Iterator itr = a1.iterator();

While(itr.hasNext()){

System.out.println(itr.next());

}

Output:-

Valli

Roopa

Lohit

Siddhu

Iterating the elements of collection by for-each loop:

Example:-

ArrayList<string> a1 = new ArrayList<String>();

a1.add(“Valli”);

a1.add(“Roopa”);

a1.add(“Siddhu”);

a1.add(“Lohit”);

for(String str : a1)  
 System.out.println(str);

Output:-

Valli

Roopa

Siddhu

Lohit

Input And Output in java:-

* Input and output used to process the input and produce the output based on the input
* Java.io.package
* Stream:- A Stream is a sequence of data

Three Streams are created for us automatically:-

1. system.out:- standard output stream
2. system.in:- standard input stream
3. System.err:- Standard error stream

OutputStream:-

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



Output Stream



Java application

1100100100

1100110100



InputStream:-

* Java applications uses an input stream to read data from a source.



Java application

1100110100

1100100100

Method of output stream:-

1. write(int)
2. write(byte[])
3. flush()
4. close()

Method of InputStream class:-

1)read()

2)available()

3)close()

OutputStream:-



outputStream

InputStream :-



InputStream



FileInputStream and FileOutputStream:-

* FileInputStream and FileOutputStream classes are used to read and write data In file.

Example for FileOutputStream:-

try{

FileOutputStream fout = new FileOutputStream(“D:// iofile//abc.txt”);

String s = “Sachin Tendulkar”;

Byte b[] = s.getBytes();

fout.write(b);

fout.close();

System.out.println(“success”);

}catch(exception e){

System.out.println(e);

}

Output:-

Success

Example for FileInputStream:-

try{

FileInputStream fin = new FileInputStream(“D://iofile//abc.txt”);

Int I ;

While(i=fin.read()!=-1)

System.out.println((char)i);

fin.close();

}catch(Exception e){

System.out.println(e):

}

SequenceInputStream class:-

SequenceInputStream class is used for data from multiple streams.

Example:-

FileInputStream fin1 = new

FileInputStream(“f1.txt”);

FileInputStream fin2 = new

FileInputStream(“f2.txt”);

SequenceInputStream sis = new SequenceInputStream(fin1, fin2);

Int I;

While((i=sis.read())!=-1){

System.out.println((char)i);

BufferedOutputStream class:-

* BufferedOutputStream used an internal buffer. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.
* It is required if you have connected the one stream with another.

Example:-

FileOutputStream fout = new FileOutputStream(“D://iofile//fu.txt”);

BufferedOutputStream bin = new BufferedOutputStream(“fout”);

String s = “Using Buffered write to fu.txt”;

byte b[] = s.getBytes();

bout.write(b);

bout.flush();

bout.close();

BufferedInputStream:-

Example:-

FileInputStream fin = new

FileInputStream(“D://iofile//fu.txt”);

BufferedInputStream bout = new bufferedInputStream(fin);

Int I;

While(i=bil.read())!=-1)

System.out.println((char)i);

fin.close();

Java File Operations:-

|  |  |  |
| --- | --- | --- |
| Operations | Methods | Package |
| To create file | createNewFile() | Java.io.file |
| To read file | read() | Java.io.fileReader |
| To write file | write() | Java.io.FileWriter |
| To delete file | delete() | Java.io.file |

Create a new file:-

File file = new File(“E:/Java/newFile1.txt”);

try{

Boolean value = file.createNewFile();

If(value){

System.out.println(“The new file is created”);

}

Else{

System.out.println(“The file already exists”);

}

}

Write a new file:-

try {

FileWriter fw = new FileWriter(“D:/Java/newFile.txt”);

fw.write(“My name is Valli”);

fw.flush();

fw.close();

}

Read content in file:-

Try{

FileReader fr = new FileReader(“D:/Java/newFile.txt);

Int I;

While(i=fw.read()!=-1)

System.out.println((char)i);

}

Reading data from keyboard:-

There are ways to read data from keyboard

* InputStreamReader
* Scanner
* Console

InputStreamReader:-

Example:-

InputStreamReader r = new InputStreamReader(System.in);

BufferedReader br = new BufferedReader(r);

System.out.println(“Enter your name: “);

String name = br.readline();

Java serialization and deserialization:-

* Serialization is a process of converting an object into a sequence of byte so that it can be written into a file, transported through a network or stored into database.
* The reverse process of creating object from sequence of byte is called deserialization
* Java.io.sereialization
* Java.io.externalizable
* ObjectInputStream
* ObjectOutputStream

Example:-

try{

Studentinfo s1 = new StudentInfo(“Valli”, 104, “110044”);

FileOutputStream fos = new FileOutputStream(“student.txt”);

ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeobject(s1);

oos.flush();

oos.close();

}

Deserialization:-

try{

FileInputStream fis = new FileInputStream(“student.txt”);

ObjectInputStream ois = new ObjectInputStream(fis);

si = (studentInfo) ois .readObject();

}

Java String Tokenizer:-

* The java.util.StringTokenizer class allows you to break a string into tokens.
* It is simple way to break a string.
* It is a legacy class of Java.

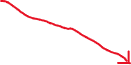
Hello walking techie how are you



String Tokenizer



Token



Hello Walking Techie how are you

Methods in string tokenizer:-

1. Boolean hasMoreTokens()
2. String nextToken()
3. String nextToken(String delim)
4. Boolean hasMoreElements()
5. Object nextElement()

Example:-

Public class simple{

Public static void main(String args[]){

StringTokenizer st = new StringTokenizer(“My name is Krishna”,” “);

While(st.hasMoreTokens()){

System.out.println(st.nextToken());

}

}

}

Multi-threading in java:-

* Multithreading in java is a process of executing multi threads simultaneously
* A thread is a lightweight sub-process, the smallest unit of processing
* Multiprocessing and multithreading, both are used to achieve multitasking
* However, we use multithreading than multiprocessing because threads use a shared memory area.
* They don’t allocate separate memory area so save memory, and context-switching between the threads takes less time than process.
* Java multithreading is mostly used in games, animation etc.

Advantages of java multithreading:-

* It doesn’t block user because threads are independent and you can perform multiple operations at the same time.
* You can perform many operations together . so it saves time.
* Threads are independent, so it doesn’t effects other threads if an exception occurs in a single thread.

Multitasking:-

* Multitasking is a process of executing multiple tasks simultaneously.
* We can multitasking to utilize the cpu.
* Multitasking can be achieved in two ways.

1. Process-based multitasking
2. Thread-based multitasking
3. Process-based multitasking:-

* A process is heavyweight.
* Cost of communication between the process is high.
* Switching from one process to allocate requires some time for saving and loading registers, memory maps, updating, lists etc.

1. Thread-based multitasking:-

* Thread share the same address space
* A thread is lightweight
* Cost of communication between the thread is low.

What is thread in java:-

* A thread is a lightweight process, the smallest unit of processing. It is a separate path of execution.
* Threads are independent.
* If there occurs exception in one thread, it doesn’t effect other threads. It uses a shared memory area.

os



* As shown in the above figure, a thread is executed inside the process.
* There is context -switching between the threads.
* There can be multiple processes inside the os, and one process can have multiple threads

Java thread class:-

* Java provides thread class to achieve thread programming.
* Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends object class and implements runnable interface

Methods used in threads:-

1. Start():- It is used to start the execution of the thread.
2. Run():- It is used to do an action for a thread
3. Sleep():-It Is used sleep a thread for the specified amount of time.
4. currentThread():- It returns a references to the currently executing thread object.
5. Join():- It wait for a thread to die.
6. getpriority():- it returns the priority of the thread.
7. setpriority():- it changes the priority of the thread.
8. getName():- It returns the name of the thread.
9. setName():- It changes the name of the thread.
10. getId():- It returns the id of the thread.
11. isAlive():- It tests if the thread is alive.
12. yield():- It causes the currently executing thread object to pause and allow other threads to execute temporarily
13. suspend():- It Is used to suspend the thread.
14. resume():- It is used to resume the suspended thread.
15. stop():- It is used to stop the thread.
16. destroy():- It is used to destroy the thread group and all of it’s subgroups.
17. isDaemon():- A daemon thread is a background service thread which runs as a low priority thread performs background operations like garbage collections.
18. setDaemon():- It marks the thread as daemon or user thread.
19. interrupt():- It interrupts the thread.
20. Isinterrupted():- It tests whether the thread has been interrupted.
21. interrupted():- It test whether the current thread has been interrupted.
22. activeCount():- It returns the number of active threads in the current threads groups.
23. checkAccess():- It determines if the currently running thread has permission to modify the thread.
24. holdLock():- It returns true if and only if the current thread holds the monitor lock on the specified object
25. dumpstack():- It is used to print a stack trace of the current thread to the standard error stream.
26. enumerate():- It is used to copy every active threads thread group and it’s subgroup into the specified array.
27. getstate():- It is used to return the state of the thread.
28. getThreadGroup():- It is used to return the thread group to which this thread belongs.
29. toString():- It is used to return a string representation of this thread, including the threads name priority, and thread group.
30. notify():- It is used to give the notification for only one thread which is waiting for a particular.
31. notifyAll():- It is used to give the notification to all waiting threads of a particular object.
32. setContextClassLoder():- I t sets the contextClassLoder for the thread.
33. getContextClassLoader():- It gets the contextClassLoader for the thread.

Life cycle of a thread:-

In java, a thread always exists in any one of the following states. These states are

1. new
2. active
3. blocked/waiting
4. timed waiting
5. terminated
6. New:- Whenever a new thread is created, it is always in the new state.
7. Active:- When a thread invokes the start() method, it moves from the new start to allocate state.
   * + 1. Runnable:-

A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time.

* + - 1. Running:-

When the thread gets the cpu, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.

3)Blocked or waiting:-

Whenever a thread is inactive for a span of time then, either the thread is in the blocked state or is in the blocked state or is in the waiting state.

4)Timed waiting:-

Sometimes, waiting for leads to starvation. For example, a thread(it’s name Is A) has entered the critical section of a code and it is not willing to leave that critical section. In such a scenario another thread(B) has to wait forever, which lead to starvation. To avoid such scenario, a timed waiting state is given to thread B.

5)Terminated:-

1. A thread reaches the terminated state because of the following reasons:-

* When thread has finished it’s job, then it exists or terminated normally
* Abnormal termination:- It occurs when some unusual events such as an unhandled exception, or segmentation fault.

2)A terminated thread means the thread is no more in the system. In other words, the thread is dead, and there is no way one can respond(active after kill) the dead thread.



Implementation of thread state:-

* In java, one can get the current state of a thread using the thread.getState() method. The java.lang.thread.state class of java provides the constant Enum to represent the state of a thread.

Public static final thread.state NEW

* It represent the first state of a thread that is the new state.

Public static final thread.state RUNNABLE

* It represents the runnable state. It means a thread is waiting in the queue to run.

Public static final thread.state BLOCKED

* It represents the blocked state. In this state, the thread is waiting to acquire a lock.

Public static final thread.state WAITING

* It represents the waiting state. A thread will go to this state when, it invoke the object.wait() method, or thread.join() method with no timeout. A thread in the waiting state is waiting for another thread to complete it’s task.

Public static final thread.state TIME-WAITING

* It represents the timed waiting state. The main difference between waiting and time waiting is the time constraint. A thread invoking the following methods.
  + - * Sleep
      * Join with timeout
      * Wait with timeout
      * Parkunit
      * parkNanos

Public static final thread.state TERMINATED

* It represents the final state f a thread that is terminated or dead.

How to create thread in java:-

There are two ways to create a thread:-

1. By extending a thread class
2. By implementing a runnable interface

Thread class:-

Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends objects class and implements runnable interface.

Commonly used constructors of thread class:-

* Thread()
* Thread(String name)
* Thread(Runnable r)
* Thread(Runnable r, String name)

Runnable interface:-

The runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

1. Public void run():- is used to perform action for a thread

Starting a thread:-

The start() method of thread class is used to start a newly created thread. It performs the following tasks:-

* A new thread starts(with new callstack)
* The thread moves from new state to runnable state.
* When the thread gets a chance to execute, it’s target run() method will run.

Example:-

class creatingThread extends Thread{

public void run(){

System.out.println(“Thread is created”);

}

Public static void main(String args[]){

creatingThread obj = new creatingThread();

obj.start();

}

}

Java Thread example by implementing runnable interface:-

Class multi implements Runnable{

Public void run(){

System.out.println(“Thread is running”);

}

Public static void main(String args[]){

multi m1 = new multi();

Thread t1 = new Thread(m1);

t1.start():

}

}

Using the thread class:- Thread(String name)

Public class MyThread{

Public static void main(String args[]){

Thread t = new Thread(“My first thread”);

t.start();

String str = t.getName();

System.out.println(str);

}

}

Using the thread class: Thread(Runnable r, String name)

Public class MyThread implements Runnable{

Public void run(){

System.out.println(“Now the thread is running”);

}

Public static void main(String args[]){

Runnable r1 = new myThread();

Thread th1 = new Thread(r1, “My Thread”);

th1.start();

String str = th1.getName();

System.out.println(str);

}

}