Java Programming Language

# Introduction to Java­

* Java- developed by James Gosling and his friends in 1991.­
* The first version of Java was released in 1995-96 by Sun -Microsystems Organisation in USA.
* Java is class based-Object oriented programming language.
* Class Based🡪 We cannot create any program without creating a class

# Features of Java

* Entirely based on Object
* Platform independent
* Simple Language
* Secured Language
* Portable Language
* Complied & Interpreted Language
* Robust Language
* High Performance Language🡪Byte Code very fast in execution
* Dynamic Language
* Very Popular Language

# JDK, JRE & JVM

* JDK- Java Development Kit🡪provide tools to work with java language
* JRE- Java Runtime Environment🡪provide environment for our programs to run🡪 JRE internally contains JVM
* JVM- Java Virtual Machine🡪it is a software 🡪it interprets our program and execute our program.

# Install & Download -JDK & Eclipse

# Basic Syntax of Java Program

class className{

main(){

System.out.println(“Hello World”);

}}

# Class

* Class is collection of Variables(data members) and methods(member functions).
* Method-piece of code which perform certain functions.Method is a part of a class.
* Syntax:

class A

{

variables

methods

}

# Object

* Object is copy of a class.
* Purpose🡪 to access variables and methods of class.
* Syntax:

class className

{

variables(a,b,c)

methods(m1,m2,m3)

}

* Object creation:

className c = new className();

c.a;// calling variable

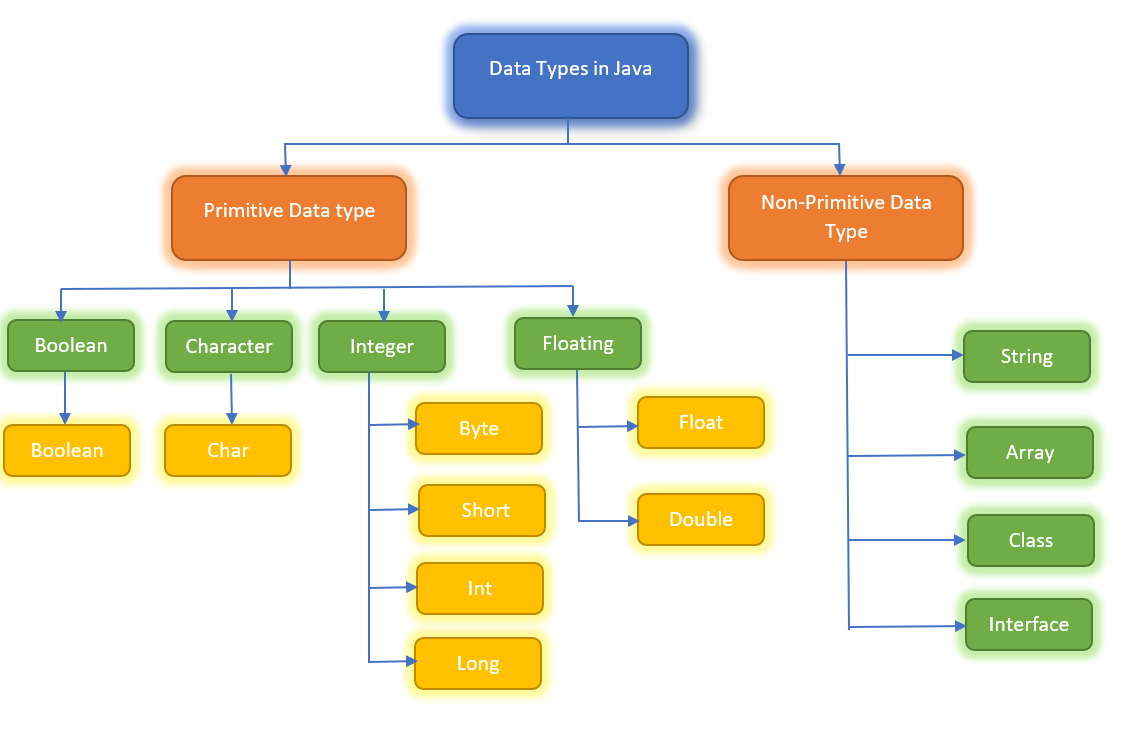
S.o.p(c.a);

c.m2();//calling methods

* We can create any number of objects of a particular class.
* Different objects will be assigned different memory allocations.
* No two objects of same class will be related to each other.
* But variables and methods of class will be available for all its objects

# Data Types

* Specify Values stored in the variables.
* Types



# Variables

* A variable is a container which holds the value while the Java program is executed.
* A variable is assigned with a data type.
* It is user-defiend (name given by user)
* Variables value can be changed (thus called variables).

Eg:

int a=100;

a=a+10; //110

float a = 100.5f; // int dt🡪 float dt.

Types:

* Local Variables:
* Instance / Global Variables:
* Static Variables:

1.Local Variables:

* Declared inside method body.

method()

{

int a;

}

* Declared inside method parameter.

method(int a)

{

S.o.p(a);

}

* We can use this variable only within that method in which it is created and the other methods in the class aren't even aware that the variable exists.
* A local variable cannot be defined with "static" keyword

2.Global/Instance variable:

* Declared inside class but outside of method.

class A

{

int a;

m1()

{

}

m2()

{

}

}

* + It is not declared as static.
  + It is called an instance variable because its value is instance-specific and is not shared among instances.

3.Static Variable:

* Declared using static k/w

class A

{ static int a; }

Variable Class:

|  |
| --- |
| **package** Day3;  **public** **class** Variables {  **public** **static** **void** main(String[] args) {    Variables var = **new** Variables();  var.method1();    }    **int** a=100; //GV      **void** method1() {  // int a=10;  **int** b=10;//LV    System.***out***.println(b);  System.***out***.println(a);//GV  }    **void** method2() {    **int** b =60;  **int** c=20;  System.***out***.println(b);//100  System.***out***.println(a);//100  System.***out***.println(c);//20    }    **void** method3(**int** d) {      System.***out***.println(a);//100  System.***out***.println(d);//      }      }  **class** Animal{    **void** method1() {  }    **void** method2() {  }  } |

Static k/w Program:

|  |
| --- |
| **package** Day3;  **public** **class** staticVariable {  **public** **static** **void** main(String[] args) {  staticVariable sv = **new** staticVariable();  System.***out***.println(sv.a);  // System.out.println(staticVariable.a);// classname.GV  System.***out***.println(staticVariable.*s*);// classname.staticVar  System.***out***.println(sv.*s*);// obj.staticVar  System.***out***.println(*s*);    staticVariable.*method3*();      }    **int** a=100;//GV    **static** **int** *s*=20;    **void** method1() {    **int** c=30;//LV  System.***out***.println(c);  }    **void** method2() {    **int** d=30;  System.***out***.println(d);  }    **static** **void** method3() {    **int** e=50;  System.***out***.println(e);  }    } |

# Methods

* Method is piece or block of code which perform certain task.
* It takes input from the user and gives output.
* Method runs only when we call the method.

Types:

1.Predefined Methods🡪

Defined by Java.

Eg: print(); , sort(); , NextInt(); etc.

2.User Defined Methods🡪

Definded by the user **.**

These methods are modified according to the requirement.

Eg: add() , show() , display() etc.

Syntax:

A diagram of a method declaration

Description automatically generated with medium confidence

Why create methods:

1.Re-Usability.

2.Readability.

# Access-Specifiers

Define accessability of the methods and variables in our class.

Types:

**1.private :**

* Contains private keyword
* accessible only within the class in which it is present.

**2.default :**

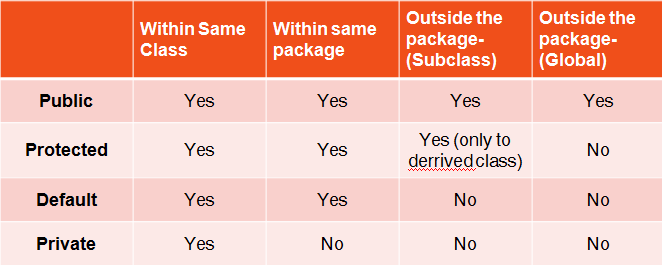
* if no access specifier mentioned , it is default.
* Accessible only within the package🡪 all its classes.
* Restricted outside the package.

**3.protected:**

* Accessible within the package.
* If we want to access it outside the package, need to use inheritance concept(using subclass).

**4.public:**

* Accessible from anywhere in the class , package , oustside the package .



|  |
| --- |
| **package** Day3Methods;  **public** **class** accessSpecifiers1 {  //1.What are the various access specifiers provided in Java --- Lab Sessions  //2.Write a Java program to illustrate private and public access specifiers.--- Lab Sessions    **private** **int** a=10;    **int** b=20;    **protected** **int** c=30;    **public** **int** d=40;    **private** **void** method1(){    System.***out***.println("this is private method1");  System.***out***.println(a);  System.***out***.println(b);  System.***out***.println(c);  System.***out***.println(d);    }      **void** method2(){    System.***out***.println("this is default method2");  System.***out***.println(a);  System.***out***.println(b);  System.***out***.println(c);  System.***out***.println(d);  }    **protected** **void** method3(){    System.***out***.println("this is protected method3");  System.***out***.println(a);  System.***out***.println(b);  System.***out***.println(c);  System.***out***.println(d);  }    **public** **void** method4(){    System.***out***.println("this is public method4");  System.***out***.println(a);  System.***out***.println(b);  System.***out***.println(c);  System.***out***.println(d);  }  **public** **static** **void** main(String[] args) {    accessSpecifiers1 as = **new** accessSpecifiers1();  as.method1();  as.method2();  as.method3();  as.method4();    }  } |

# Return Type

* Define the return type of the method.
* It decides what data if any, the method will be returning.
* The method should contain return keyword if the method has specific return type.
* If the method does not return anything then the return type is void.

Eg:

int m1(){

int a=10;

return a;

}

|  |
| --- |
| **package** Day3Methods;  **public** **class** returnType {    **void** method1(){    **int** a=10;  System.***out***.println(a);//10  }    **int** method2(){ //20    **int** a =1000;  **return** a;    }  **boolean** method3(){    **boolean** f=**false**;  **return** f;    }    String method4(){    String s= "Java";  **return** s;    }    **int** method5(**int** c){    **return** c; //35    }    **public** **static** **void** main(String[] args) {    returnType rt = **new** returnType();    rt.method1();    System.***out***.println(rt.method2());    **int** r=rt.method2();//1000  System.***out***.println(r);//1000    **boolean** b = rt.method3();  System.***out***.println(b);    String d=rt.method4();  System.***out***.println(d);    **int** x=rt.method5(35);  System.***out***.println(x);  }  } |

# Parameterised & Non Parameterised methods

1.Non -Parameterised method: does not pass parameters

class A{

m1(){

sop(“Java”);

}

Calling method:

Obj.m1();

2. Parameterised method: pass parameters

class A{

m1(int a){

sop(a);

}

Calling method: Obj.m1(10);//10

|  |
| --- |
| **package** Day3Methods;  **public** **class** parameterisedMethod {  **void** method1(){  **int** a=50;  a=100;  System.***out***.println(a);  }    **void** method2(**int** b, **int** c){  **int** z=b+c;  System.***out***.println(z);    }    **void** method(**int** b , **int** c){    **int** z=b+c;  System.***out***.println(z);  }    **void** method3(**int** c , String s , **boolean** d){    System.***out***.println(c);//50  System.***out***.println(s);//Java  System.***out***.println(d);      }  **public** **static** **void** main(String[] args) {    parameterisedMethod pm = **new** parameterisedMethod();  // pm.method1();  // pm.method2();  pm.method2(2,3);  pm.method(2,3);  pm.method3(50, "Java" , **false**);    }  } |

# **Memory in Java**

1. Stack Memory:

* Stack stores the data that is short-lived.
* It includes local primitive variables, references of heap objects, and methods in execution
* Every method has its own stack memory.

1. Heap Memory:

* It stores objects and JRE classes.
* Whenever we create objects it occupies space in the heap memory while the reference of that object creates in the stack.

# String Class

* String is basically an object that represents sequence of char values.
* Non-Primitive Data-Type.
* It is a final Class.
* Ways to create String:

1.String Literal: String a= “Java”;

String b=”Java”;

2.New K/W: String s = new String(“Java”);

String x= new String(“Java”);

* String Pool Areas:

|  |  |
| --- | --- |
| Constant Pool Area | Non-Constant Pool Area |
| * No new k/w | 1.New k/w |
| * Duplicates not allowed | 2.Duplicated allowed |
| * Memory-Fixed-No new memory   allocated | 3. Memory- Not Fixed-Expands when new k/w used. |

# String Methods

* **charAt()** method
* **concat()** method
* **contains()** method
* **endsWith()** method
* **startsWith()** method
* **equals()** method
* **equalsIgnoreCase()** method
* **indexOf()** method
* **isEmpty()** method
* **length()** method
* **replace()** method
* **replaceAll()** method
* **split()** method
* **substring()** method
* **toLowerCase()** method
* **toUpperCase()** method
* **trim()** method

|  |
| --- |
| **public** **class** string {  **public** **static** **void** main(String[] args) {    String a="Java";  String b="Java";  System.***out***.println(a==b); //true -->// duplication not allowed  String s1=**new** String ("String");  String s2= **new** String("String");  System.***out***.println(s1==s2);//false --> // duplication allowed      } |

|  |
| --- |
| **public** **class** stringMethods1 {  **public** **static** **void** main(String[] args) {    //charAt() method  String name="JavaClass";  System.***out***.println(name.charAt(4));      **char** ch=name.charAt(4);//returns the char value at the 4th index --> C    System.***out***.println(ch); //C        //concat() method  // 1. Java Program to Concatenate Two Strings------Lab Sessions  String s1="java string";    s1.concat("is immutable");    System.***out***.println(s1);    s1=s1.concat(" is expilicitly immutable ");  System.***out***.println(s1);    System.***out***.println("");    //contains() method    String s="This is Java Class";  System.***out***.println(s.contains("This is")); //true  System.***out***.println(s.contains("Java")); //T  System.***out***.println(s.contains("Cla")); //T  System.***out***.println(s.contains("Manual")); //F    System.***out***.println("");    }  } |

|  |
| --- |
| **public** **class** stringMethods2 {  **public** **static** **void** main(String[] args) {    //endsWith() method  String s="java morning session";  System.***out***.println(s.endsWith("n")); //T  System.***out***.println(s.endsWith("ion")); //T  System.***out***.println(s.endsWith("morning")); //F    System.***out***.println("");      //startsWith() method  String str="java morning session";  System.***out***.println(str.startsWith("j"));//T  System.***out***.println(str.startsWith("m"));//F  System.***out***.println(str.startsWith("java")); //T    System.***out***.println("");      } |

|  |
| --- |
| **public** **class** stringMethods3 {  **public** **static** **void** main(String[] args) {      // indexOf() method  String s1 = "indexOf method";    **int** index1 = s1.indexOf("m"); //8 index  System.***out***.println("index of m is "+index1);    **int** index2 = s1.indexOf("e");  System.***out***.println("index of x is "+index2);    **int** index3 = s1.indexOf("etho"); //3  System.***out***.println("index of etho is "+index3);    System.***out***.println("");      // isEmpty() method    String e1="";  String e2="java";    System.***out***.println(e1.isEmpty());//T  System.***out***.println(e2.isEmpty()); //F    System.***out***.println("");      // length() method    String str1 = " Welcome To Java ";  **int** size1 = str1.length();  System.***out***.println(size1);      String str2 = "Welcome To Java";  **int** size2 = str2.length();  System.***out***.println(size2);      }  } |

|  |
| --- |
| **public** **class** stringMethods4 {  **public** **static** **void** main(String[] args) {    // replace() method    String s1="This is my Java Class";  String replaceStr=s1.replace('a','z');  System.***out***.println(replaceStr);    System.***out***.println("------------");    String s2="This is my Java Class";  String replaceStr2=s2.replace("my","our");  System.***out***.println(replaceStr2);    System.***out***.println("------------");    // replaceAll() method    String str = "JavaClass";  System.***out***.println(str);  System.***out***.println("------------");    str = str.replaceAll("", " ");  System.***out***.println(str);  System.***out***.println("------------");    str = str.replaceAll(" ", "./");  System.***out***.println(str);  System.***out***.println("------------");    // split() method    String s="Java Split method";    String[] words=s.split(" ");    **for**(String w:words){    System.***out***.println(w);    }  }  } |

|  |
| --- |
| **public** **class** stringMethods5 {  **public** **static** **void** main(String[] args) {  //equals() method  String s1="javaClass";  String s2="JAVACLASS";  String s3="MorningJava";  String s4="javaClass";    System.***out***.println(s1.equals(s2));//f  System.***out***.println(s1.equals(s3));//f  System.***out***.println(s1.equals(s4)); //t  System.***out***.println("");    //equalsIgnoreCase() method  String j1="javaClass";  String j2="JAVACLASS";  String j3="MorningJava";  String j4="javaClass";    System.***out***.println(j1.equalsIgnoreCase(j2));//t  System.***out***.println(j1.equalsIgnoreCase(j3));//f  System.***out***.println(j1.equalsIgnoreCase(j4)); //t  }    } |

|  |
| --- |
| **public** **class** stringMethods6 {  **public** **static** **void** main(String[] args) {    //substring() method  String s="javaClass";  System.***out***.println(s.substring(2,7)); // 7th index is exclusive// vaCla  System.***out***.println(s.substring(2)); // from 2nd index //vaClass    //toLowerCase() method  String s1="JAVA CLASS stRIng";  String s1lower=s1.toLowerCase();  System.***out***.println(s1lower);  System.***out***.println(s1);    //toUpperCase() method  String s2="Java ClasS stRIng";  String s2UPPER=s2.toUpperCase();  System.***out***.println(s2UPPER);    //trim() method  String s3=" Java Class ";  System.***out***.println(s3);  String trim=s3.trim();  System.***out***.println(trim);  }  } |

# Packages

* A package is group of classes , interfaces .
* It is just like a folder .
* Types:
* Pre-defined: java.lang , java.util, java.io , etc
* User-Defined: package p1 , package newPackage etc.

# Constructors

* It is a special type of a method.
* It has name same as that of the classname.
* Every java class has a constructor.
* If user does not create the constructor , the java creates default constructor during compilation of the class.
* The constructor is automatically called when we create the object of the class.
* The constructor does not have any return type, not even void.
* It is called constructor because it constructs the values at the time of object creation

Syntax:

class A

{

A( ){

}

}

Types Of Constructors:

* Default / Non- Parameterized Constructor:-does not have any paramaeter

class A

{

A(){

}

}

* Parameterised Constructor:Parameterised Constructor

class A

{

A(int a){

}

}

|  |
| --- |
| **public** **class** constructor1 {  **public** **static** **void** main(String[] args) {  constructor1 c = **new** constructor1();  c.show();  }  **int** a;  String s;  **boolean** b;    constructor1() {  a = 10;  s = "constructor";  b = **false**;  }    **void** show() {  a = 100;  s = "Java";  b = **true**;    System.***out***.println(a + " " + s + " " + b);  } |

|  |
| --- |
| **public** **class** defaultConstructor {  // 1. Java Program to illustrate the default constructor------Lab Sessions    **int** a;  String b;    defaultConstructor(){    a=10;  b="Java";    }    **void** display(){    System.***out***.println(a+" "+b);  }  **public** **static** **void** main(String[] args) {    defaultConstructor d = **new** defaultConstructor();  d.display();    }  } |

|  |
| --- |
| **public** **class** parameterisedConstructor {    //Java Program to illustrate the parameterized constructor------Lab Sessions  **int** id;  String name;    parameterisedConstructor(**int** a, String b){  id=a;  name=b;    }    **void** display(){    System.***out***.println(id+" "+name);  }    **public** **static** **void** main(String[] args) {  parameterisedConstructor d =**new** parameterisedConstructor(10,"John");  d.display();    parameterisedConstructor d1 =**new** parameterisedConstructor(20,"Mary");  d1.display();    }  } |

# Operators

* Arithmetic Operators

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Name | Description | Example |
| + | Addition | Adds together two values | x + y |
| - | Subtraction | Subtracts one value from another | x - y |
| \* | Multiplication | Multiplies two values | x \* y |
| / | Division | Divides one value from another | x / y |
| % | Modulus | Returns the division remainder | x % y |
| ++ | Increment | Increases the value of a variable by 1 | ++x |
| -- | Decrement | Decreases the value of a variable by 1 | --x |

|  |
| --- |
| **public** **class** ArithmeticOperators {  **public** **static** **void** main(String[] args) {    ArithmeticOperators ao = **new** ArithmeticOperators();  ao.method1();  // ao.method2();    }    **void** method1(){    **int** x=10;  **int** y=40;  **int** sum;    sum=x+y;    //+ operator  System.***out***.println(sum);//50    //- operator  **int** sub;  sub=x-y;    System.***out***.println(sub);//-30    //\* operator  **int** m;  m=x\*y;  System.***out***.println(m);    // '/' operator  **int** d;  d=y/x;  System.***out***.println(d);    // '%' operator    **int** mod;  mod=y%x;  System.***out***.println(mod);  **int** i;  i=++x;  System.***out***.println(i);// x=10 , x++ = 11    **int** dec;  dec=--y;  System.***out***.println(dec);    } |

* Relational Operators

|  |  |  |
| --- | --- | --- |
| **Condition** | **Operator** | **Example** |
| Is equal to | == | int num=1; (num==1) |
| Is not equal to | != | int num=1; (num!=2) |
| Is less than | < | int num=0; (num<1) |
| Is less than or equal to | <= | int num=1; (num<=1) |
| Is greater than | > | int num=2; (num>1) |
| Is greater than or equal to | >= | int num=1; (num>=1) |
|  |  |  |

|  |
| --- |
| **public** **class** relationalOperators {  **public** **static** **void** main(String[] args) {      **int** var1 = 5;  **int** var2 = 10;  **int** var3 = 5;    //== operator  System.***out***.println("var1 == var2: " + (var1 == var2));//5==10//f  System.***out***.println("var1 == var3: " + (var1 == var3));//5==5//t    // > operator  System.***out***.println("var1 > var2: " + (var1 > var2));//5>10//f  System.***out***.println("var3 > var1: " + (var3 > var1));//5>5//f    // < operator  System.***out***.println("var1 < var2: " + (var1 < var2));//5<10//t  System.***out***.println("var2 < var3: " + (var2 < var3));//10<5//f    // >=/<= operator  System.***out***.println("var1 >= var2: " + (var1 >= var2));//5>=10 //f  System.***out***.println("var2 >= var3: " + (var2 >= var3));// 10>=5 //t    System.***out***.println("var1 <= var2: " + (var1 <= var2));//5<=10 //t  System.***out***.println("var2 <= var3: " + (var2 <= var3));// 10<=5 //f    //!= Operator  System.***out***.println("var1 != var2: "+ (var1 != var2)); //5!=10 //t  System.***out***.println("var1 != var3: "+ (var1 != var3)); // 5!=5 // f    }  //  } |

* Logical Operators
* **AND Operator**( **&&**) – a && b
* **OR Operator** ( **||**) – a || b
* **NOT Operator** (**!**) – a!=b

|  |
| --- |
| **public** **class** LogicalOperators {  **public** **static** **void** main(String[] args) {    **int** a = 10, b = 20, c = 20;  // && operator    System.***out***.println(a > b && b == c); // 10>20 and 20==20 // t+t=t    // || operator    System.***out***.println(a < b || b != c);//10<20 or 20!=20 // t+f=t    // ! operator    System.***out***.println(!(a<b)); //!(10<20)-->!true -->returns false  System.***out***.println(!(a>b)); //!(10>20)-->!false -->returns true    }  } |

# Java Control Statement

Java provides two types of control flow statements:

1.Decision Making statements

1. 🡪 if statements-if , if else , if else ladder
2. 🡪 switch statement

2.Loop statements

* + 🡪 do while loop
  + 🡪 while loop
  + 🡪 for loop
  + 🡪 for-each loop

# Decision Making statements

1.Decision Making statements

1. 🡪 if statements
2. 🡪 switch statement

1. if Statement

Use the if statement to specify a block of Java code to be executed if a condition is true.

**Syntax:**

if (condition)

{// Executes this block if condition is true}

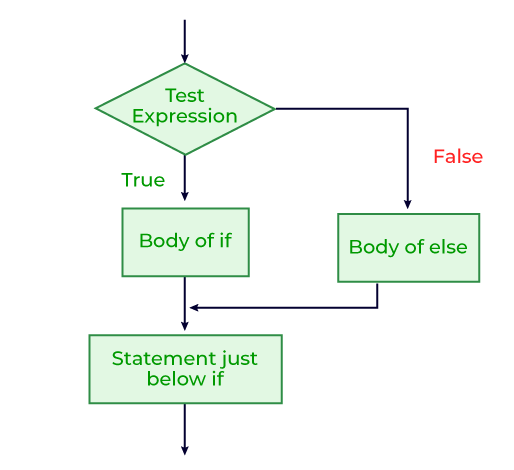
A picture containing text, screenshot, diagram, line

Description automatically generated

|  |
| --- |
| **public** **class** ifStatement {  **public** **static** **void** main(String[] args) {  **int** i = 100;    **if** (i > 15)  {  System.***out***.println("inside if - block");    }    System.***out***.println("Outside if-block");    }    } |

2. if else Statement

If- else together represents the set of Conditional statements in Java that are executed according to the condition which is true.



**Syntax:**

if (condition)

{

// Executes this block if condition is true

}

else

{

// Executes this block if condition is false

}

|  |
| --- |
| **public** **class** ifElseStatement\_Program\_1 {  **public** **static** **void** main(String[] args) {  **int** i = 10;  **if** (i < 15) {  System.***out***.println("if - block");  } **else** {  System.***out***.println("else - block");  }  System.***out***.println("Outside if-else block");  System.***out***.println("Outside if-else block");  System.***out***.println("Outside if-else block");  System.***out***.println("Outside if-else block");  System.***out***.println("Outside if-else block");  System.***out***.println("Outside if-else block");    }  } |

3. if else Ladder Statement

* **if-else-if ladder** is used to decide among multiple options.
* The if statements are executed from the top down.
* As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed.
* If none of the conditions is true, then the final else statement will be executed.
* **Syntax:**

if (condition1)

{

// Executes this block if condition is true

} else if (condition2)

{

// Executes this block if condition is true

}

else if (condition3)

{

// Executes this block if condition is true

}

.

.

.

else

{

// Executes this block if all above conditions are false

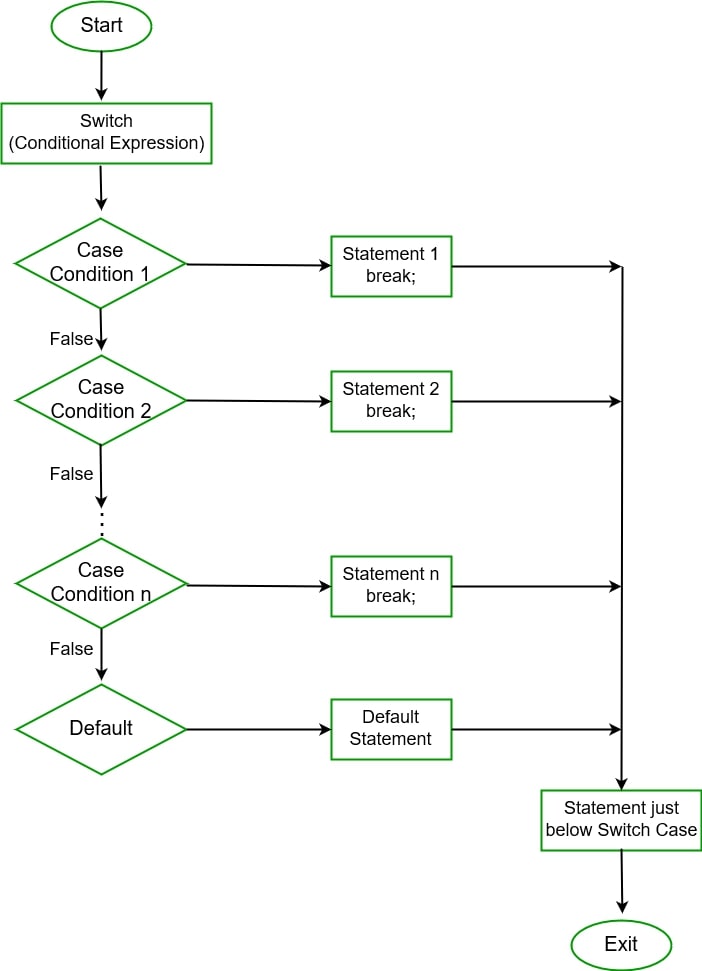
}

A black background with green rectangles

Description automatically generated with low confidence

4. Switch Case Statement

* Executes one statement from multiple conditions.



# Loop statements

Loop statements

* + 🡪 do while loop
  + 🡪 while loop
  + 🡪 for loop
  + 🡪 for-each loop

**1. while Loop:**

* **Syntax:**

while (condition)

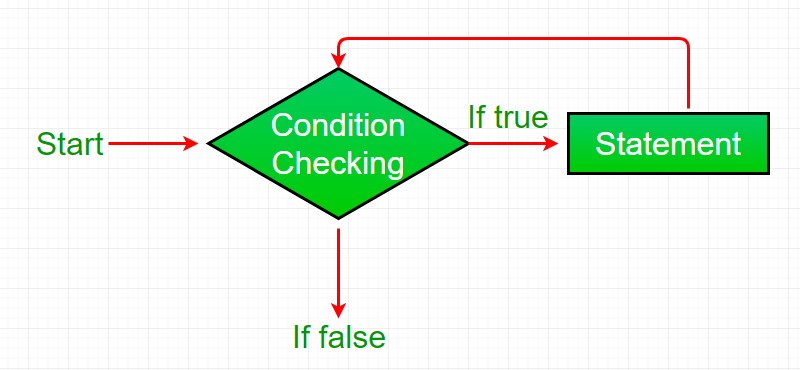
{

// statements

update\_expression;

}

* **Flow Chart:**



**2.Do while Loop:**  **Exit control loop**.

* **Syntax:**

do{

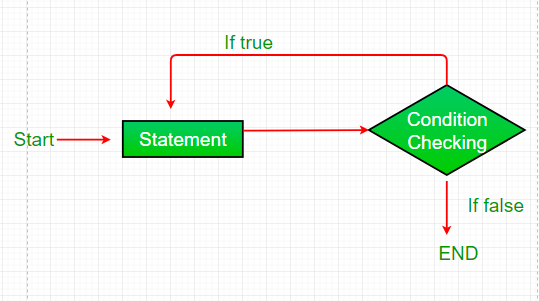
// Loop Body

Update\_expression

}

while (condition); // Condition check

* **Flow Chart:**



**3.for Loop:**

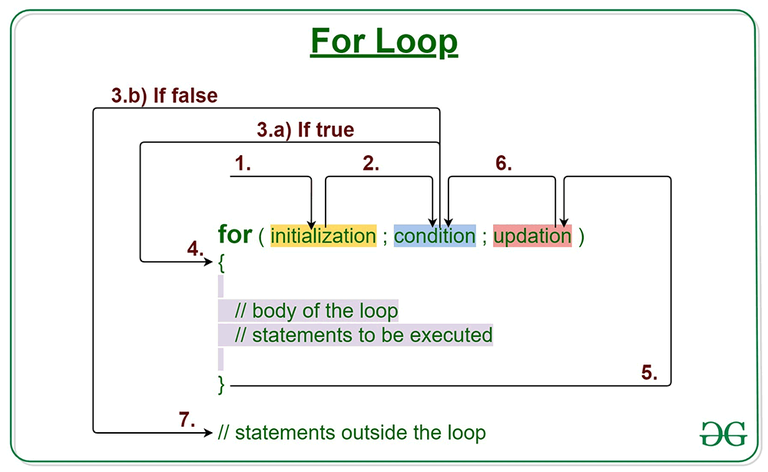
* **Syntax:**

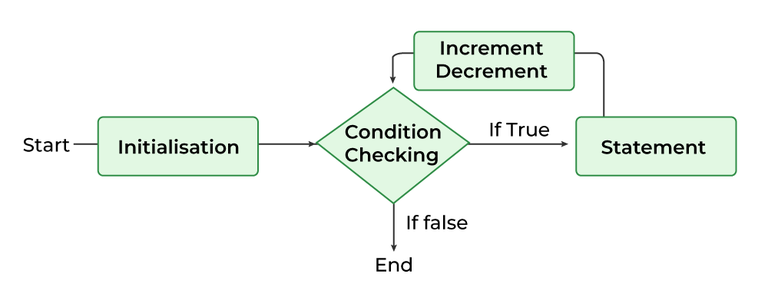
for (initialization; test condition; update exp)

{

// body of the loop

}





**4.for each - Loop:**

* Used to iterate over an array or a Collections class
* **Syntax:**

for (type var : array)

{

statements using var;

}

# Array

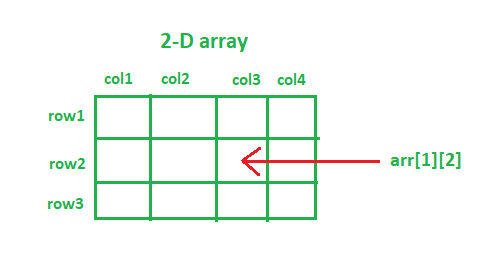
* Array is an object in java which contains similar type of data in a continous memory location.
* We can store only a fixed set of elements in a Java array.
* Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.
* Syntax:

Eg: int a[] ; / int[] a;

* Declaration :
* int a[] = new int [5];
* int a[]= { 10, 20 , 30 , 40 ,50};
* Types:
* Single Dimensional Array



* Multi Dimensional Array



# OOP’s Principles:

# 1.Inheritance:

* When one class (child , sub-class) access all the properties of another class(parent ,super class).
* Achieved using extends keyword.
* Use-Reusability of data
* We cannot access private members.
* Since all properties present in child/sub-class , we create object of subclass.
* Method – Overriding possible using inheritance
* Syntax:

class parent

{ }

class child **extends** parent

{ }

* Types:
* Single Level inheritance
* Multi Level inheritance
* Multiple Level inheritance
* Hybrid/ Hierarchial inheritance

### Super k/w:

🡪 Refers to objects of super / parent class.

🡪When we have used super class and sub class variables and methods of same name , we use super k/w to avoid confusion between super class and sub class variables and methods.

Super 🡪 variable , method , constructor

# 2.Polymorphism

* Same Object performing different roles.
* Types:

1.Compile Time Polymorphism: exists at time of compilation. Thus role of Compiler comes here.Achieved through Method Overloading.

Method Overloading:Whenever a class contains more than one method with same name and different types of paramaters

Same Class 🡪 different methods 🡪same name🡪different parameters

2.Run Time Polymorphism: exists at the time of execution of program. Thus no role of Compiler but of JVM.Achieved through Method Over-riding.

Method Overriding: Whenever we create a method in Parent and Child Class using same method name and parameters.

Note:We first need to perform inheritance using extends k/w before perform method Overiding.

Eg:Class A{

Void show(){

}

Class B extends A{

Void show(){ }

Rules to remember:

While runtime , JVM first search for method in parent class 🡪 if No then give Compilation error.

# 3.Encapsulation:

* Wrapping the variables and methods in a single unit.(class).
* Note: declare class var and methods- private

# 4.Abstraction:

* Process of hiding the implementation details from the user but showing just the highlighted set of services provided to the user.
* Advantage: Security and enhancements.
* Achieved through –

1.Abstract Class

2. Interface

## Abstract Class:

* Contains abstract k/w in its declaration.
* We cannot create object of abstract class , but can make reference variable.
* It may or may not contain abstract methods(methods no implementation), thus can include abstract and non-abstract methods as well.
* To use abstract class we need to inherit it to sub-class k/a Concrete class.(using extends k/w).

## Interface

* It is like a class containing only abstract methods
* Achieved using implements k/w.
* Implementaion of the abstract methods are in implementation class , using implements k/w.
* Interface methods by default are public and abstract.
* Interface variables are by default public , static and final.
* Use: as a deal between client and the developer.
* We can achieve multiple inheritance using Interface in java.

### This K/w:

* Used when global and local variable/methods have same name.
* Thus to avoid confusion of JVM during runtime , we used this k/w.
* Also used when we want ot call the default constructor of it’s own class.

`

### Final K/w:

* It is a modifier which provide restrictions to variable , method and class
* Final variable🡪 once declared final , we cannot change the assigned value.
* Final Method🡪once declared final , we cannot override the method.
* Final Class🡪 once declared final , final class cannot be extended or inherited to child class.

### TypeCasting

* Converting one datatype to another datatype.
* ***Types***🡪

1. Implicit-

* Small dt to big dt.­­
* JavaC does it for you automatically.

1. Explicit-

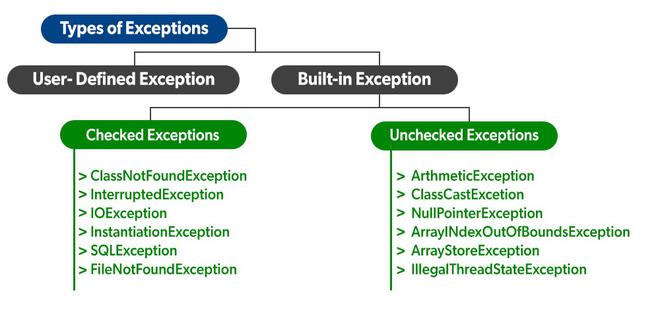
* Big dt to small dt.
* By default , JavaC does not allow explicit typecasting.
* We have to explicitly typecast.
* Data loss may happen.

# ­­ Exceptions & Errors

|  |  |
| --- | --- |
| Exceptions | Errors |
| Caused by our program | Caused due to lack of system resources |
| Recoverable 🡪 using try -catch block , throw k/w | Not-recoverable |
| Classified-Checked & Un-Checked | Only Un-Checked type |
| Both are Child Classes of throwable super class | |
| Both occur at runtime | |

# Exception Handling

* Exception 🡪 unexpected / unwanted / abnormal situation that occurred during runtime
* Exception Handling🡪 done to maintain the flow of our program execution irrespective of exceptions.
* **Types of Exceptions**:



* **Checked Exceptions:**

**🡪checked** exceptions are also known as **compileTime** execeptions.

🡪Checked exceptions are those which need to be taken care at compile time.

* **Un-Checked Exceptions:**

**🡪unchecked** exceptions are also known as **runtime** exceptions.

🡪Unchecked exceptions are those which need to be taken care at runtime.

* **Techniques to handle exceptions**:

1.try – catch block

🡪finally block

2.throw k/w

3.throws k/w

## Try-Catch block

**Try Block –**

* contains risky / error suspecting code.
* If exception encountered here, it will directly enter the catch block
* Thus the remaining code in the try block willl not be executed.
* Therefore, we write only the code that gives exception and not the whole code inside the try block

**Catch Block –**

* Handles the exception thrown by the try block.
* Note: Catch block executed only when exception occurs in try block.
* Thus , it does not execute if there is no exception thrown inside the try block.

## Finally Block

* This block is executed compulsorily , irrespective of exception occurrence.

**Difference between final , finally and finalize?**

* Final k/w: variable , method , class
* Finally{ } block🡪 try , catch , finally block
* Finalize () method🡪
* **finalize() method** of **Object Class ,**is a method that the Garbage collector always calls just before the deletion/destroying the object which is eligible for Garbage Collection to perform clean-up activity.
* Clean-up activity means closing the resources associated with that object like Database Connection, Network Connection, or we can say resource de-allocation
* Note: Every java class is a subclass of predefined Object class

## Throw k/w

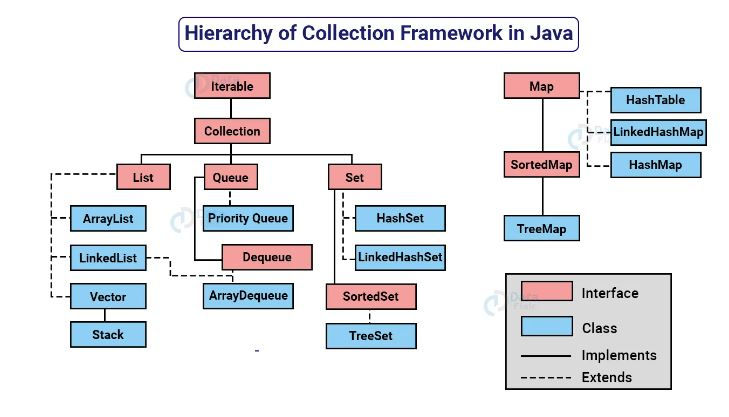
* Throw k/w is used to explicitly throw an exception from a method or any block of code.
* We can throw either checked and unchecked exceptions.
* The throw keyword is mainly used to throw custom exceptions.

## Throws k/w

* throws is a keyword in Java that is used in the signature of a method to indicate that this method might throw one of the listed type exceptions.
* The caller(JVM) to these methods has to handle the exception using a try-catch block.
* use the throws keyword to delegate the responsibility of exception handling to the caller (It may be a method or JVM) then the caller method is responsible to handle that exception.

## Collections Framework

* Set of Pre-defined classes and interfaces in Java.
* Helps to perform different data structure operations like: sorting , searching , traversing ,storing and processing data efficiently.



* Need For Collections Framework : To overcome drawbacks of Array

1. We can store homogeneous data only in Array.
2. Fixed size of Array-->thus not growable in nature.
3. Array does not have underline data structure.(it cannot use readymade methods).

## Collection Interface

* **Collections interface** 🡪 Parent Interface contains various methods which can be directly used by all the collections which implement this interface:
* *add(Object)* 🡪This method is used to add an object to the collection.
* *addAll(Collection c)*🡪This method adds all the elements in the given collection to this collection.
* *clear()🡪* This method removes all of the elements from this collection.
* *contains(Object o)* 🡪This method returns true if the collection contains the specified element.
* *containsAll(Collection c)🡪* This method returns true if the collection contains all of the elements in the given collection.
* *equals(Object o)*🡪 This method compares the specified object with this collection for equality.
* *isEmpty()🡪* This method returns true if this collection contains no elements.
* *iterator()*🡪 This method returns an iterator over the elements in this collection.
* *max()* 🡪This method is used to return the maximum value present in the collection.
* *remove(Object o)🡪* This method is used to remove the given object from the collection. If there are duplicate values, then this method removes the first occurrence of the object.
* *removeAll(Collection c)* 🡪 This method is used to remove all the objects mentioned in the given collection from the collection.
* *size()*🡪 This method is used to return the number of elements in the collection.
* *toArray()🡪* This method is used to return an array containing all of the elements in this collection.

## List Interface

* Child interface of the collection interface.
* Insertion order of the objects is preserved.
* Allows duplicate data to be present in it. (index helps in identification)
* Implemented by various classes like ArrayList, Linked List
* Since all the subclasses implement the list, we can instantiate a list object with any of these classes.

List <T> al = new ArrayList<> ();

List <T> ll = new LinkedList<> (); //Where T is the type of the object

### **Operations in a Java List Interface**

* **Operation 1:**Adding elements to List class using add() method
* **Operation 2:**Updating elements in List class using set() method
* **Operation 3:** Searching for elements using indexOf(), lastIndexOf()methods
* **Operation 4:**Removing elements using remove() method
* **Operation 5:**Accessing Elements in List class using get() method
* **Operation 6:**Checking if an element is present in the List class using contains() method

### **Adding elements to List class using** **add() method**

**Parameters:**  It takes 2 parameters:

* **add(Object):** This method is used to add an element at the end of the List.
* **add(int index, Object):** This method is used to add an element at a specific index in the List

### **Updating elements in List class using set() method**

Takes an index and the ­ which needs to be inserted at that index.

### **Searching for elements using indexOf(), lastIndexOf methods**

* indexOf() method returns the index of the first occurrence of a specified element in the list
* lastIndexOf() method returns the index of the last occurrence of a specified element.

### **Removing elements using remove() method**

To remove an element from a list.

**Parameters:**

* **remove(Object):** This method is used to simply remove an object from the List. If there are multiple such objects, then the first occurrence of the object is removed.
* **remove(int index):** Since a List is indexed, this method takes an integer value which simply removes the element present at that specific index in the List. After removing the element, all the elements are moved to the left to fill the space and the indices of the objects are updated.

### **Accessing Elements in List class using get() method**

* In order to access an element in the list, we can use the get() method, which returns the element at the specified index

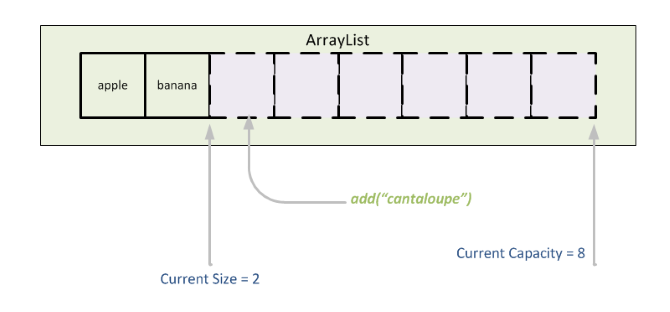
**Parameters:**

* get(int index): This method returns the element at the specified index in the list.

### **Checking if an element is present in the List using contains() method**

* This method returns true if the specified element is present in the list, otherwise, it returns false.

## ArrayList Class



1. ArrayList is Resizable Array or Growable Array.
2. Default size is 10.
3. Duplicates Are Allowed.
4. Insertion Order is Preserved.
5. Heterogeneous objects are allowed.
6. Null insertion is possible.

Advantages of ArrayList

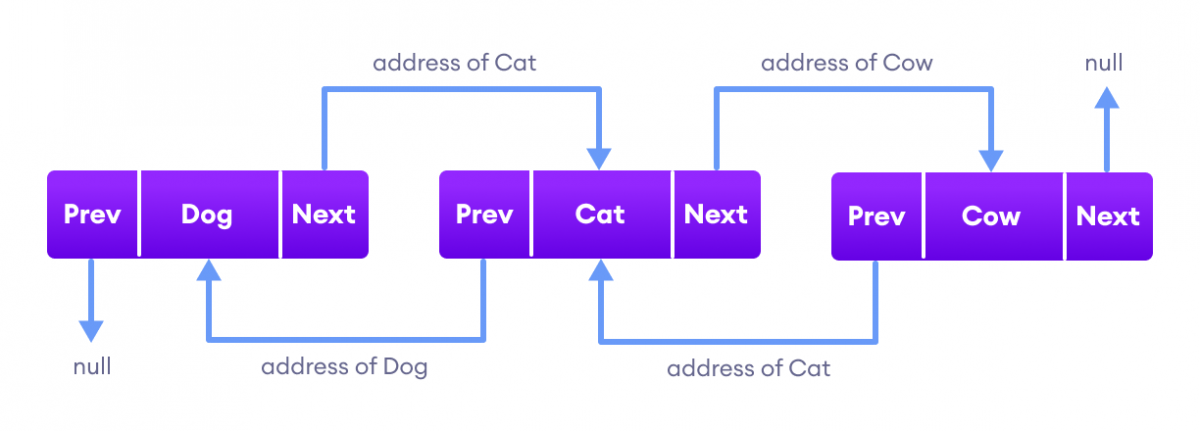
1. Dynamic size: ArrayList can dynamically grow and shrink in size, making it easy to add or remove elements as needed.
2. Fast access
3. Ordered collection
4. Supports null values

Disadvantages of ArrayList

1. Slower than arrays for certain operations, such as inserting elements in the middle of the list.
2. Increased memory usage: as it needs to maintain its dynamic size and handle resizing.
3. Performance degradation: as the number of elements in the list increases, especially for operations such as searching for elements or inserting elements in the middle of the list.

## LinkedList Class

1. Default size -N/A- an empty list is created when a LinkedList is initialized.
2. Duplicates Are Allowed.
3. Insertion Order is Preserved.
4. Null insertion is possible.
5. Based on doubly linked data structure.



Advantages of using LinkedList in Java:

1. Dynamic size
2. Efficient Insertions and Deletions
3. Flexible Iteration
4. Insertion / Deletion Performance-faster🡪 node link are connected / disconnected and no element shifting takes place.

Disadvantages of using LinkedList in Java:

1. Retrieval Performance: slower performance than ArrayList , because we have to traverse the list to reach the desired element, whereas with ArrayList, you can simply access the desired element using an index.
2. Memory overhead: LinkedList requires more memory than ArrayList because each element requires additional memory for the links to its predecessor and successor elements.

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |
| 5) The memory location for the elements of an ArrayList is contagious. | The location for the elements of a linked list is not contagious. |
| 6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList. | There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized. |

## **Set Interface**

* Child interface of the collection interface
* Unordered collection of objects .
* Duplicate values cannot be stored.
* Used when we wish to avoid the duplication of the objects and wish to store only the unique objects.
* Implemented by various classes like HashSet, TreeSet, LinkedHashSet, etc.
* Since all the subclasses implement the set, we can instantiate a set object with any of these classes.

Set<T> hs = new HashSet<> ();

Set<T> lhs = new LinkedHashSet<> ();

Set<T> ts = new TreeSet<> ();//Where T is the type of the object.

## HashSet Class

1. Default size -16
2. Duplicates Not Allowed.
3. Insertion Order is Not -Preserved.
4. Null insertion is possible.
5. Heterogeous data allowed
6. Elements entered using Hashcode(No Index concept)🡪faster performance.
7. Has not methods of its own , just implements set interface methods.
8. Direct Sorting / shuffling not possible.(convert to Arraylist first , then perform sorting).
9. Load Factor : 75%.

## LinkedHashSet Class

Only differences🡪

1. Insertion Order is -Preserved
2. Based on Hashtable + LinkedList Set.

**Queue Interface Methods**🡪

* add(element): Adds an element to the rear of the queue. If the queue is full, it throws an exception.
* offer(element): Adds an element to the rear of the queue. If the queue is full, it returns false.
* remove(): Removes and returns the element at the front of the queue. If the queue is empty, it throws an exception.
* poll(): Removes and returns the element at the front of the queue. If the queue is empty, it returns null.
* element(): Returns the element at the front of the queue without removing it. If the queue is empty, it throws an exception.
* peek(): Returns the element at the front of the queue without removing it. If the queue is empty, it returns null.

## **Map Interface**

* Independent interface.
* A map is a data structure that supports the key-value pair for mapping the data.
* Doesn’t support duplicate keys because the same key cannot have multiple mappings, however, it allows duplicate values in different keys.
* A map is useful if there is data and we wish to perform operations on the basis of the key.
* This map interface is implemented by various classes like HashMap, HashTable
* Since all the subclasses implement the map, we can instantiate a map object with any of these classes.

Map<T> hm = new HashMap<> ();

Map<T> tm = new TreeMap<> ();//Where T is the type of the object.

## HashMap Class

1. Insertion Order is -Not Preserved
2. Based on Hashcode data structure.
3. Duplicate keys🡪 not allowed.
4. Duplicate values🡪allowed.
5. Null key🡪 allowed only once.
6. Null Values🡪 multiple allowed.
7. Preferred when🡪 more number of search operations
8. HashMap is Non -Synchronised🡪 multiple threads allowed at a time🡪 not thread safe🡪 performance fast

## HashTable Class

Similarity with HashMap🡪

1. Based on Hashcode data structure.
2. Insertion Order is -Not Preserved
3. Duplicate keys🡪 not allowed.
4. Duplicate values🡪allowed.

Differences with HashMap🡪

1.HashTable is Synchronised🡪Only one thread can access the method at a time🡪 thus thread safe🡪 performance compromised

2.Duplicate keys🡪 not allowed.

3.Duplicate values🡪allowed.

4.Null key🡪 not allowed.

5.Null Values🡪 not allowed.

Hashtable class has since been considered obsolete and its use is generally discouraged.

## File Handling