## Basics of any programing language:

Let’s try to understand basics of any programing language before moving to Java:

Below are the common / basic futures of any programming language:

1. Data Types
2. Variables
3. Operators
4. Conditional Statements
5. Control Statements (loops)
6. Arrays
7. Functions / Methods

**Data Types:**

Data types are used to allocate memory and specify what type of data you are using in the program.

All the data types are keywords which are having built in meaning. Below are the few data types in Java.

1. **int**: which is used to store integer value and it will not accept any decimal value. It allocates 4 bytes of memory.
2. **long**: which is used to store integer value and it will not accept any decimal value. It allocates 8 bytes of memory
3. **float**: which is used to store decimal value. It allocates 4 bytes of memory
4. **double**: which is used to store decimal value. It allocates 8 bytes of memory
5. **char**: which is used to store single character and it use to take only one byte.
6. **Short:** which use to take 2 bytes of memory and stores integer value.
7. **Boolean:** which takes only true or false
8. **Byte:** which takes only byte memory

**String**: is not a data type but it is predefined class. It allows to store set of characters.

**Variables:** Variables are nothing but in which data is stored. It is nothing but memory location name by which you can refer your data.

Declarition and initilization of variable

datatype variablename=value

Ex: int a=10;

int b=20

int c=a+b

Rules to follow for declaring the variable name:

1. Should start with lower case and first letter of the second word should start with captial letter. i.e. firstName, orderNumber
2. Can start with \_ (under score)
3. Can start with $
4. First letter cannot be a capital letter
5. It can contain numbers but not in the middle
6. No other special symbols allowed

**Operators:**

1. Arithmetic:

1. +
2. -
3. \*
4. /
5. % (moduls) which gives remainder

2. Relation Operator:

1. >
2. <
3. >=
4. <=
5. !=

3. Equality Operator / comparison:

1. == which checks for equality

4. Concatenation Operator:

1. +: is used to value with string or string with value

Ex: String a="Ashok"

int b=10

a+b=Ashok10

b+a=10Ashok

5. Logical Operators

1. &&(And operator)
2. ||(OR operator)
3. ! (logical not operator)

**Conditional statements**: are used to execute set of statements based on the condition satisfaction.

1. if
2. else
3. else if
4. nested if

if:

Syn: if(condition)

{

statements;

}

This block of statements will be executed only if condition is true.

if-else:

syn: if(condition)

{

statements;

}

else

{

statements;

}

else-if

syn: if(condition)

{

statements;

}

else if(condition)

{

statements;

}

else

{

statements;

}

Nested if: writing if inside if

Example: Here we are writing one if inside another if condition itself.

If(conditions)

{

If(condition)

{

Statements;

}

}

**Switch**: Unlike if-then and if-then-else statements, the switch statement can have a number of possible execution paths. A switch works with the byte, short, char, and int.

Syn:

Switch(expression)

{

case “a”:

statements;

break;

case “b”:

statements;

break;

default:

statements;

}

Note: Here “break” is a keyword which breaks the block in which it is used. Here it comes outside switch after executing any of the case statements. If we do not write break here it will continue to execute remaining case statements also.

**Control statements (loops)**: are used to execute set of statements as per defined number of times. All the programming languages have below 3 types of loops.

* 1. while
  2. for
  3. do-while

While Loop:

Syntax:

initialization;

while (condition)

{

statements;

incrimination / decrementation of number

}

Example:

int i=1;

while(i<=10)

{

statements;

i++ OR i=i+1;

}

From above code statements inside the loop will be repeated for 10 times.

You can implement an infinite loop using the while statement as follows:

boolean b=true;

while (b){

// your code goes here

if(login==)

b=false

}

Note: You need to make sure make b value set to false to avoid infinite loop

For Loop:

Syntax:

for(initialization; condition; incrimination / decrementation of number)

{

statements;

}

Example:

For(int i=1; i<=10;i++)

{

Statements;

}

do-while syn:

do

{

statements;

incrimination / decrementation of number

}while(condition);

The difference between do-while and while is that do-while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once

Example:

do

{

Statements;

incrimination / decrementation of number

}while(i<=10);

**Arrays**: Arrays are nothing but set of similar elements stored in single variable.

If we take normal variable (int x;) it will take / store only one value at a time and we cannot store more than one value.

Let’s say I would like to write a program where I want to store student rule numbers for college in which there are 1000 students available. If I use normal variable I have to declare thousand variables like below:

int rolno1=1;

int rolno2=2;

int rolno3=3

And it goes on till

int rolno1000=1000;

Here you are spending more time in declaring variable rather than writing actual program. So to avoid this we can go for using arrays where you can declare only one array variable and store 1000 values in this variable.

**Declaration of Arrays:**

Data type arrayname []

Example: int rolno[];

**Allocating Memory for Arrays:**

int rolno[] = new int[1000];

From above line we are declaring rolno variable as an array and we specifying size as 1000 which means rolno variable can be used to store 1000 values in one array variable.

We are using “new” here new is a key word which basically used to allocate the memory. We will discuss about “new” key word in coming topics in detail.

**Initializing / assigning the values for Arrays:**

In above we understand that we have allocated size of array as 1000 and we can story 1000 integer values in the variable called rolno.

Let’s assign values for array: arrays index always starts with zero

rolno[0]=1;

rolno[1]=2;

rolno[2]=3;

rolno[3]=4;

And it goes on….. Till

rolno[999]=1000;

Since arrays starts with indexing zero last value will be stored always size – 1 which is 1000-1= 999

Types of array

1. Single dimension array: Example int rolno[] = new int[1000];
2. Multi dimension array: Example int x[][]=new int[10][20];

**Methods / Functions:** Method is a set or group of statements which can be used to repeat any number of times. Methods are written for specific task / purpose. Methods have following:

1. Method declaration (it is not required from java and C++ & needed in C language)

2. Method calling (goes to method definition), if we don’t call function definition will not be executed

3. Method definition (here you write the code for which purpose method is written)

Recursive method: A method which is calling itself is known as recursive method.

Syntax: return type functionname(parmeters list)

{

Code

}

Void add()

{

Code

}

Types of functions

1. Pre-defined: These functions are which are already defined and we can reuse these functions. Example: main
2. User defined: Which we are going to define / write to our self

**OOPS**: If any programming language that supports below futures then that language is known as OOPS language.

1. Class
2. Object
3. Polymorphism
4. Inheritance
5. Encapsulation

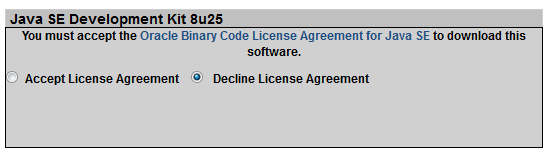
We will discuss each of these topics in the below.

**Installation of Java:**

1. Open <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
2. Click on JDK Download



1. Click on “Accept License Agreement”



1. Select Operating System to which you are downloading Java (Windows)

**Core Java Basics:**

1. What is Class
2. What is Object
3. Constructors
4. Polymorphism
   1. Method Overloading
   2. Method Overriding
5. Constructor Overloading
6. Static Variables / Methods / Block
7. Inheritance & Types of Inheritance
8. Super & This
9. Abstract class & Abstract methods
10. Interfaces
11. Packages
12. Access Modifiers
13. Exception Handling
14. Collections Interfaces
15. Final, Finalize and finally
16. Strings

**Class:**

A class is a group of data members and functions which has common properties.

Syn: class Class\_Name

{

Data members(variables)

Data functions

}

Ex: class Car

{

String steering;

String break;

int wheels;

void run()

{

}

void reverseGear()

{

}

}

**Objects:**

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical.

Object is an instance of a class. So object is the instance(result) of a class.

Syn: Class\_Name objname = new Class\_Name();

Ex: Car maruthi = new Car();

Car Skoda = new Car();

**Constructors:**

Constructor is a special type of method which has same name of the class.

Constructor is invoked at the time of object creation and no need to call the constructors like functions. Constructors will not return a value not even void also.

If there is no constructor in a class, compiler automatically creates a default constructor

There are basically two rules defined for the constructor.

* Constructor name must be same as its class name
* Constructor must have no explicit return type

Types of constructors:

1. Default constructor (no-arguments constructor)
2. Parameterized constructor

Default Constructor:

Constructor that have no parameter is known as default constructor.

Syntax of default constructor:

class\_name()

{

Initilization statements;

}

Parameterized constructor:

A constructor that have parameters is known as parameterized constructor.

Syntax of Parameterized constructor:

class\_name(data type variable name, data type variable name)

{

Initialization statements;

}

Why use parameterized constructor?

Parameterized constructor is used to provide different values to the distinct objects.

**Polymorphism**:

Means many forms. More than one method will have same name.

Polymorphism is achieved in two ways

1. Compile time polymorphism
2. Run time polymorphism

Compile time polymorphism: is achieved using method overloading and constructor overloading

Method overloading:

If a class have multiple methods by same name but different parameters, it is known as Method Overloading.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as add1(int,int) for two parameters, and add2(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs. So, we perform method overloading to figure out the program quickly.

Advantages of method overloading: Method overloading increases the readability of the program

Different ways to overload the method

1. By changing number of arguments
2. By changing the data type

Method Overloading is not possible by changing the return type of the method

Can we overload main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading.

**Constructor Overloading:**

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Run time polymorphism: is achieved using method overloading and this concept is implemented using inheritance. We will discuss this once inheritance is completed.

**Static key word in java:**

The static keyword is used in java mainly for memory management. We may apply static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

* variable (also known as class variable)
* method (also known as class method)
* block

Static variable:

If you declare any variable as static, it is known static variable.

The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees,college name of students etc.

The static variable gets memory only once in class area at the time of class loading.

Advantage of static variable:

It makes your program memory efficient (i.e it saves memory).

Suppose there are 500 students in my college, now all inx x xstance data members will get memory each time when object is created. All student have its unique rollno and name so instance data member is good. Here, college refers to the common property of all objects.If we make it static,this field will get memory only once

Static method:

If you apply static keyword with any method, it is known as static method

A static method belongs to the class rather than object of a class.

A static method can be invoked without the need for creating an instance of a class.Static methods called by classname. ex classname.methodName();

Static method can access only static data member and can change the value of it.

Restrictions for static method:

* The static method cannot use non static data member or call non-static method directly.
* this and super keywords cannot be used in static context.

Static block:

Is used to initialize the static data member.

It is executed before main method at the time of class loading

Why main method is static?

Because object is not required to call static method if it were non-static method, jvm create object first then call main() method that will lead the problem of extra memory allocation

**Inheritance:**

Inheritance is a mechanism in which one object acquires all the properties and behaviors of parent object.

The idea behind inheritance is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you reuse (or inherit) methods and fields, and you add new methods and fields to adapt your new class to new situations.

Why use Inheritance?

* For Method Overriding (So Runtime Polymorphism).
* For Code Reusability.

Syntax of Inheritance

class Subclass-name extends Superclass-name

{

//methods and fields

}

The keyword extends indicates that you are making a new class that derives from an existing class. In the terminology of Java, a class that is inherited is called a superclass. The new class is called a subclass.

Types of Inheritance

1. Single
2. Multilevel
3. Multiple
4. Hybrid

Java does not support multiple inheritance however it can implemented using Interfaces.

Single Inheritance:

class Father

{

//methods and fields

}

class Child extends Father

{

//methods and fields

}

Multilevel Inheritance:

class GrandFather

{

}

class Father extends GrandFather

{

//methods and fields

}

class Child extends Father

{

//methods and fields

}

**Using Super:**

When invoking a superclass version of an overridden method the super keyword is used.

**Abstract Class :**

A class which contains the abstract keyword in it’s declaration is classed abstract class.

Eg. Abstract Class A(){

}

Note: 1. We can not create object of abstract class.

2.It’s may or may not contains abstract methods .

3.It can have abstract and not-abstract methods .

4.To use on abstract class , you have to inherite it from subclass.

5.If a class contains partial implementation then we should declare a class as abstract.

**Abstract Methods**: A method which contain abstract modifier at the time of declaration in called as abstract method.

-It can only be used in abstract class.

-It doesn’t contain any body “{ }” and always ends with “;”.

-Abstract methods must be overridden in sub-class otherwise it will also become a abstract class.

**Interfaces:**

What is interface ?

Ans : An interface is a blue print of class or just like a class .which contains only static ,constant and abstract methods .

To achieve interface java provides a keyword called implements.

Note: 1.Interface methods are by default public and abstract.

2. Interface variables are bydefault public +static +constant.

3. Interface methods must be overwrite inside the implementing class.

4.Interface nothing but deles between client and developer.

================================================

An interface is a blueprint of a class. It has static constants and abstract methods.

The interface is a mechanism to achieve fully abstraction in java. There can be only abstract methods in the interface. It is used to achieve fully abstraction and multiple inheritance in Java

It cannot be instantiated just like abstract class.

Why use Interface?

•It is used to achieve fully abstraction.

•By interface, we can support the functionality of multiple inheritance.

The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.

Multiple inheritance in Java by interface:

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.

Note: A class implements interface but One interface extends another interface

**Package in Java:**

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

Package can be categorized in two form, built-in package and user-defined package. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Advantage of Package:

•Package is used to categorize the classes and interfaces so that they can be easily maintained.

•Package provids access protection.

Creation of Package:

Package packageName; // this package contains class called Example

Class Example

{

Data & Methods

}

How to access package from another package?

There are three ways to access the package from outside the package.

1.import packagename.\*; It will import all the class and interfaces to this package. It is recommended not to use this as it will impact on performance as it as to load all the class.

2.import packagename.classname;

3.fully qualified name.

Using packagename.\*:

If you use package.\* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

Example: import packageName.\*

Using packagename.classname:

If you import package.classname then only declared class of this package will be accessible.

Example: import of packageName.ClassName

Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

**Access Modifiers:**

The access modifiers specifies accessibility (scope) of a datamember, method, constructor or class.

There are 4 types of access modifiers:

1.private

2.default

3.protected

4.public

Private:

The private access modifier is accessible only within class.

Simple example of private access modifier

class A

{

private int x=40;

private void method1()

{

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

}

}

public class Simple

{

public static void main(String args[])

{

A obj=new A();

System.out.println(obj.data);//Compile Time Error

obj.method1();//Compile Time Error

}

}

Role of Private Constructor:

If you make any class constructor private, you cannot create the instance (object) of that class from outside the class. For example:

class A

{

private A(){

}//private constructor

void msg(){System.out.println("Hello java");}

}

public class B

{

public static void main(String args[])

{

A obj=new A();//Compile Time Error

}

}

Note: A class cannot be private or protected except nested class.

Default:

If you don't use any modifier, it is treated as default by default. The default modifier is accessible only within package.

Example of default access modifier

In this example, we have created two packages pack1 and pack2. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

//save by A.java

package pack1;

class A

{

void method1(){System.out.println("Hello");}

}

//save by B.java

package pack2;

import pack1.\*;

class B

{

public static void main(String args[])

{

A obj = new A();//Compile Time Error

obj.method1();//Compile Time Error

}

}

In the above example, the scope of class A and its method method1() is default so it cannot be accessed from outside the package.

Protected:

The protected access modifier is accessible within package and outside the package but through inheritance only.

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

Example of protected access modifier

In this example, we have created the two packages pack1 and pack2. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

//save by A.java

package pack1;

public class A

{

protected void method1()

{

System.out.println("Hello");

}

}

//save by B.java

package pack2;

import pack1.\*;

class B extends A

{

public static void main(String args[])

{

A obj = new A();

obj.method1();

}

}

Output:Hello

Public:

The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

Example of public access modifier

//save by A.java

package pack;

public class A

{

public void msg()

{

System.out.println("Hello");

}

}

//save by B.java

package mypack;

import pack.\*;

class B

{

public static void main(String args[])

{

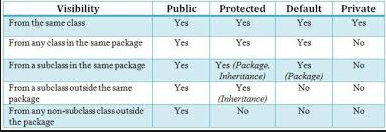
A obj = new A();

obj.msg();

}

}

Access modifiers by a simple table.



**Exception Handling:**

The exception handling is one of the powerful mechanism provided in java. It provides the mechanism to handle the run time errors so that normal flow of the application can be maintained.

Exception

* Dictionary Meaning: Exception is an abnormal condition.
* In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Advantage of Exception Handling:

The core advantage of exception handling is that normal flow of the application is maintained. Exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

1.statement 1;

2.statement 2;

3.statement 3;

4.statement 4;

5.statement 5;

6.statement 6;

7.statement 7;

8.statement 8;

9.statement 9;

10.statement 10;

Suppose there is 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the code will be executed. That is why we use exception handling.

Types of Exception:

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

Checked exceptions: A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.

Runtime exceptions: A runtime exception is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.

Errors: These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Common scenarios of Exception Handling where exceptions may occur:

There are given some scenarios where unchecked exceptions can occur. They are as follows:

1. Scenario where Arithmetic Exception occurs: If we divide any number by zero, there occurs an Arithmetic Exception.

int a=50/0;//Arithmetic Exception

1. Scenario where NullPointerException occurs: If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

String s=null;

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

1. Scenario where NumberFormatException occurs: The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

String s="abc";

int i=Integer.parseInt(s);//NumberFormatException

1. Scenario where ArrayIndexOutOfBoundsException occurs: If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

int a[]=new int[5];

a[10]=50; //ArrayIndexOutOfBoundsException

Five keywords used in Exception handling:

1.try

2.catch

3.finally

4.throw

5.throws

Syntax of try with catch block

try

{

statements;20/0

}

catch(Exception\_class\_Name reference)

{

handle exception here;

}

Syntax of try with finally block

try

{

statements;

}

finally

{

handle exception here;

}

Rules:

* At a time only one Exception is occurred and at a time only one catch block is executed.
* All catch blocks must be ordered from most specific to most general i.e. catch for ArithmeticException must come before catch for Exception.

Finally block:

The finally block is a block that is always executed. It is mainly used to perform some important tasks such as closing connection, stream etc.

Note:Before terminating the program, JVM executes finally block(if any).

Note:finally must be followed by try or catch block.

Why use finally block?

finally block can be used to put "cleanup" code such as closing a file,closing connection etc

Note: For each try block there can be more catch blocks, but you can have only one finally block.

Note: The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).

Throw:

The throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exception.

Throws keyword

The throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Syntax of throws keyword:

void method\_name() throws exception\_class\_name

{

statements;

}

Differences between throw keyword and throws keyword

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

**Collections:** Collections in java is a framework that provides an architecture to store and manipulate the group of objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

**What is Collection in java**

Collection represents a single unit of objects i.e. a group.

**What is framework in java**

* provides readymade architecture.
* represents set of classes and interface.

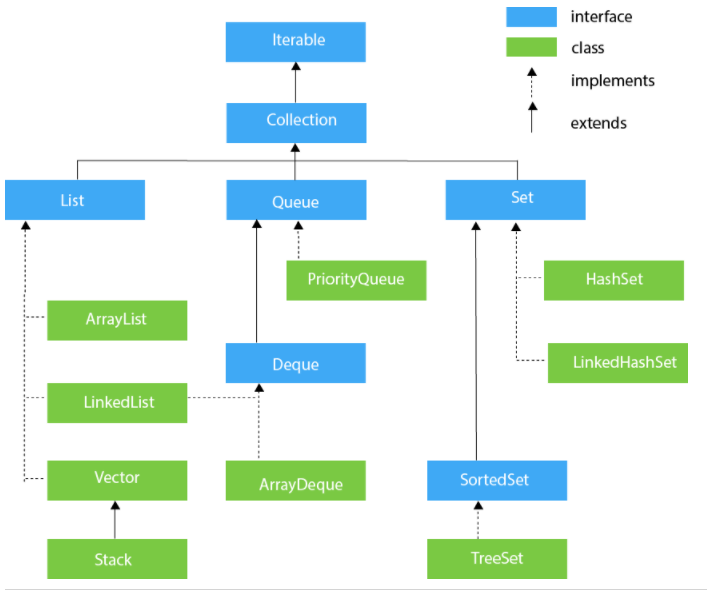
**What is Collection framework**

Collection framework represents a unified architecture for storing and manipulating group of objects. It has:

1. Interfaces and its implementations i.e. classes

**Hierarchy of Collection Framework**

Let us see the hierarchy of collection framework.The **java.util** package contains all the classes and interfaces for Collection framework.



**Methods of Collection interface**

There are many methods declared in the Collection interface. They are as follows:

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | public boolean add(Object element) | is used to insert an element in this collection. |
| 2 | public boolean addAll(Collection c) | is used to insert the specified collection elements in the invoking collection. |
| 3 | public boolean remove(Object element) | is used to delete an element from this collection. |
| 4 | public boolean removeAll(Collection c) | is used to delete all the elements of specified collection from the invoking collection. |
| 5 | public boolean retainAll(Collection c) | is used to delete all the elements of invoking collection except the specified collection. |
| 6 | public int size() | return the total number of elements in the collection. |
| 7 | public void clear() | removes the total no of element from the collection. |
| 8 | public boolean contains(Object element) | is used to search an element. |
| 9 | public boolean containsAll(Collection c) | is used to search the specified collection in this collection. |
| 10 | public Iterator iterator() | returns an iterator. |
| 11 | public Object[] toArray() | converts collection into array. |
| 12 | public boolean isEmpty() | checks if collection is empty. |
| 13 | public boolean equals(Object element) | matches two collection. |
| 14 | public int hashCode() | returns the hashcode number for collection. |

### Iterator interface

|  |
| --- |
| Iterator interface provides the facility of iterating the elements in a forward direction only. |

#### Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | public boolean hasNext() | It returns true if the iterator has more elements otherwise it returns false. |
| 2 | public Object next() | It returns the element and moves the cursor pointer to the next element. |
| 3 | public void remove() | It removes the last elements returned by the iterator. It is less used. |

# Java ArrayList class

* Java ArrayList class uses a dynamic array for storing the elements. It extends AbstractList class and implements List interface.
* Java ArrayList allows random access because array works at the index basis.
* In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

### List vs ArrayList in Java

|  |  |
| --- | --- |
| LIST | ARRAYLIST |
| List is an Interface. | ArrayList is a Class. |
| [List interface extends the Collection framework.](https://www.geeksforgeeks.org/collections-in-java-2/) | ArrayList extends AbstractList class and implements List interface. |
| List cannot be instantiated. | ArrayList can be instantiated. |
| List interface is used to create a list of elements(objects) which are associated with their index numbers. | ArrayList class is used to create a dynamic array that contains objects. |
| List interface creates a collection of elements that are stored in a sequence and they are identified and accessed using the index. | ArrayList creates an array of objects where the array can grow dynamically. |

## Set Interface

Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

Set can be instantiated as:

1. Set<data-type> s1 = **new** HashSet<data-type>();
2. Set<data-type> s2 = **new** LinkedHashSet<data-type>();
3. Set<data-type> s3 = **new** TreeSet<data-type>();

## HashSet

HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.

## Set Interface

Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

Set can be instantiated as:

1. Set<data-type> s1 = **new** HashSet<data-type>();
2. Set<data-type> s2 = **new** LinkedHashSet<data-type>();
3. Set<data-type> s3 = **new** TreeSet<data-type>();

## HashSet

HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.

Final variable:

If you make any variable as final, you cannot change the value of final variable (It will be constant).

Final method:

If you make any method as final, you cannot override it.

Final class:

If you make any class as final, you cannot extend it.

Is final method inherited?

final method is inherited but you cannot override it.

Can we initialize blank final variable?

Yes, but only in constructor.

**Final Keyword in Java:**

The final keyword in java is used to restrict the user. The final keyword can be used in many context. Final can be

* Final variable
* Final method
* Final class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

**Strings:**

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

|  |  |  |
| --- | --- | --- |
| No. | Method | Description |
| 1 | char charAt(int index) | returns char value for the particular index |
| 2 | [int length()](https://www.javatpoint.com/java-string-length) | returns string length |
| 3 | [static String format(String format, Object... args)](https://www.javatpoint.com/java-string-format) | returns a formatted string. |
| 4 | [static String format(Locale l, String format, Object... args)](https://www.javatpoint.com/java-string-format) | returns formatted string with given locale. |
| 5 | [String substring(int beginIndex)](https://www.javatpoint.com/java-string-substring) | returns substring for given begin index. |
| 6 | [String substring(int beginIndex, int endIndex)](https://www.javatpoint.com/java-string-substring) | returns substring for given begin index and end index. |
| 7 | [boolean contains(CharSequence s)](https://www.javatpoint.com/java-string-contains) | returns true or false after matching the sequence of char value. |
| 8 | [static String join(CharSequence delimiter, CharSequence... elements)](https://www.javatpoint.com/java-string-join) | returns a joined string. |
| 9 | [static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)](https://www.javatpoint.com/java-string-join) | returns a joined string. |
| 10 | [boolean equals(Object another)](https://www.javatpoint.com/java-string-equals) | checks the equality of string with the given object. |
| 11 | [boolean isEmpty()](https://www.javatpoint.com/java-string-isempty) | checks if string is empty. |
| 12 | [String concat(String str)](https://www.javatpoint.com/java-string-concat) | concatenates the specified string. |
| 13 | [String replace(char old, char new)](https://www.javatpoint.com/java-string-replace) | replaces all occurrences of the specified char value. |
| 14 | [String replace(CharSequence old, CharSequence new)](https://www.javatpoint.com/java-string-replace) | replaces all occurrences of the specified CharSequence. |
| 15 | [static String equalsIgnoreCase(String another)](https://www.javatpoint.com/java-string-equalsignorecase) | compares another string. It doesn't check case. |
| 16 | [String[] split(String regex)](https://www.javatpoint.com/java-string-split) | returns a split string matching regex. |
| 17 | [String[] split(String regex, int limit)](https://www.javatpoint.com/java-string-split) | returns a split string matching regex and limit. |
| 18 | [String intern()](https://www.javatpoint.com/java-string-intern) | returns an interned string. |
| 19 | [int indexOf(int ch)](https://www.javatpoint.com/java-string-indexof) | returns the specified char value index. |
| 20 | [int indexOf(int ch, int fromIndex)](https://www.javatpoint.com/java-string-indexof) | returns the specified char value index starting with given index. |
| 21 | [int indexOf(String substring)](https://www.javatpoint.com/java-string-indexof) | returns the specified substring index. |
| 22 | [int indexOf(String substring, int fromIndex)](https://www.javatpoint.com/java-string-indexof) | returns the specified substring index starting with given index. |
| 23 | [String toLowerCase()](https://www.javatpoint.com/java-string-tolowercase) | returns a string in lowercase. |
| 24 | [String toLowerCase(Locale l)](https://www.javatpoint.com/java-string-tolowercase) | returns a string in lowercase using specified locale. |
| 25 | [String toUpperCase()](https://www.javatpoint.com/java-string-touppercase) | returns a string in uppercase. |
| 26 | [String toUpperCase(Locale l)](https://www.javatpoint.com/java-string-touppercase) | returns a string in uppercase using specified locale. |
| 27 | [String trim()](https://www.javatpoint.com/java-string-trim) | removes beginning and ending spaces of this string. |
| 28 | [static String valueOf(int value)](https://www.javatpoint.com/java-string-valueof) | converts given type into string. It is an overloaded method. |

**Encapsulation : Is mechanism through which we can wrapping the data members and member and member method of class in a single unit called encapsulation .**

**Note : 1.Declare class variable as private**

**2.Declare the class as public**