**Java:**

1. Java was developed by Sun Microsystems in the year 1995,James Gosling is known as the father of Java.

2. Java is a programing language .

3. Java can create mobile applications ,web applications , standalone applications.

4. Java is very securable in other languages.

5. Java is object oriented language.

6. Java is independent.

Platform independent:

1 .Platform independent means suppose we have an project on windows operating system ,

That project has many .java files when i compile that project then that time that .java files will

converted to .class file will run in any operating system,this is platform independent..

2. Java is a run time environment

3. It is own platform to run applications

Features :

1. Simple : Java is very easy to learn, and its syntax is simple, clean and easy to understand.

2. Object-Oriented :

3. Portable:

4. Platform independent:

5. Secured:

6. Robust:

7. Architecture neutral:

8. Interpreted:

9. High Performance:

10. Multithreaded:

11. Distributed: 1 . It is a software that runs on multiple computers connected to a network at the same time .

2 . There are two types of concepts java can implement distributed applications.

1. Remote Method Invocation RMI

2. Enterprise Java Beans EJB

12. Dynamic:

CLASS:

1. Class is a group of obj.

2. Class is a wrapping of datamembers and memberfunctions/methods as a single unit .

3. In a class we can write variables and methods.

Syntax:

class <class name>{

variables;

methods;

}

OBJECT:

1. An entity that has state and behavior is known as an object

2. Object is created through new keyword mainly e.g. Student s1=new Student();

3. Obj is a realworld entity

4. Obj is a blue print of class

5. With out class obj does not exist.

Example:

public class Employe {

String name;

int id;

public String run(){

String name="mani";

return name;

}

public static void main(String args[])

{

Employe e1=new Employe();

e1.name="sai";

e1.id=12;

System.out.println(e1.name);

System.out.println(e1.id);

System.out.println(e1.run());

}

}

JVM:

1. It is a specification that provides a runtime environment in which Java bytecode can be executed ,and also

( I ) Loads code

( II ) Verifies code

( III ) Executes code

( IV ) Provides runtime environment.

2. JVM platform dependent because the configuration of each OS is different from each other.

JRE:

1. The JRE is a set of software tools which are used for developing Java applications.

2. It is used to provide the runtime environment. It is the implementation of JVM. It contains a set of libraries + other files .



JDK:

1.JDK is used to develop Java applications.It contains JRE + development tools.



VARIABLES:

1.A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

2.Variable is a name of memory location.

3.There are three types of variables in java

1) Local Variable: 1. A variable declared inside the body of the method is called local variable.

2. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

3. Access modifiers cannot be used for local variables.

2) Instance Variable: 1. A variable declared inside the class but outside the body of the method, is called instance variable.

2. It is called instance variable because its value is instance specific and is not shared among instances.

3. Access modifiers can be given for instance variables.

3) Static variable: 1. A variable which is declared as static is called static variable.

2. It cannot be local. You can create a single copy of static variable and share among all the instances of the class.

3. Memory allocation for static variable happens only once when the class is loaded in the memory.

4. Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.

5. Static variables are created when the program starts and destroyed when the program stops.

JAVA DATA TYPES:-

1.Byte:-

1. Byte data type is an 8-bit integer type

2. Minimum value is -128

3. Maximum value is 128

4 .Byte data is used to save the pace in large arrays.mainly inplace of integers

5. it is 4 times smaller than int

2. Short:-

1 .short data type is an 16-bit integer type

2. Minimum value is -32768

3. Maximum value is 32767

4. short data is used to save the pace in large arrays.mainly in place of integers

5. it is 2 times smaller than int

3. Int:-

1. Int data type is a 32-bit integer type

2. Minimum value is -2^32

3. Maximum value is 2^31 -1

4. int is generally used data type for integer values .

4.Long:-

1. Long data type is a 64-bit integer type

2. Minimum value is -9,223,372,036,854,775,808.(-2^63)

3. Maximum value is 9,223,372,036,854,775,807 (inclusive). (2^63 -1)

4. This type is used when a huge range than int is needed.

5. Default value is 0L.

5. Float:-

1. Float data type is a 32-bit float type

2. Float is used to save memory in large arrays..

3. Default value is 0.0f.

4. Example: float f1 = 234.5f

6. Double:-

1. double data type is a double-precision 64-bit float type

2. This is generally used as the default data type for decimal values.

3. Default value is 0.0d.

4. Example: double d1 = 123.4

7. Char:-

1. char data type is a single 16-bit character.

2. Minimum value is 0

3 .Maximum value is 65,535

4. Char data type is used to store any character.

5. Example: char letterA ='A'

8. Boolean:-

1. boolean data type size is one bit

2. There are only two possible values: true and false.

3. This data type is used for simple flags that track true/false conditions.

4. Default value is false.

5. Example: boolean one = true

CONSTRUCTOR:

1. Constructor is a special type of method that is used to initialize the value of obj while creating only .

2. Constructor name must be the same as its class name only.

3. There is no return type of constructor .

4. There are two types of constructors in Java:

Default constructor (Zero argument constructor)

Parameterized constructor (User defined constructor)

5. If you can create any Parameterized constructor you should create Zero argument constructor .

6. If you are not create any Zero argument constructor ,then compiler will create Default constructor .

Syntax:-

<class\_name>( )

{

EXAMPLE:

1)Default Constructor:- Default constructor means no parameters to constructor.

\*If we do not create any parameterized Constructor then compiler creates default Constructor.

eg:

class

{

data members;

no constructors.

}

-> This is provided by java compiler

2)Parameterized Constructors:-A Constructor that have parameter is known as Parameterized Constructors.

eg:-

class Employee{

int empNo;

string empName;

float empSalary;

Employee(int no)

{

empNo=no;

}

Employee(int no,string name)

{

empNo=no;

empName=name;

}

Employee(int no,string name,float sal)

{

empNo=no;

empName=name;

empSalary=sal;

}

Employee(){

{

}

class Test{

public static void main(String [] arg){

Employee first\_Employee=new Employee(10,"ramu",100000);

//It is valid

System.out.println("firt\_Employee no is :"+first\_Employee.empNo+"\nSalry is:"+first\_Employee.empSalary);

Employee second\_Employee=new Employee("ramu", 20,10000);

// it is not valid

//because in this one there is no order of value assigning

Employee third\_Employee=new Employee(10);

// it is valid

System.out.println("third employee no is"+third\_Employee.empNo);

}

}

Example:

package com.test.Constructor;

public class Bag {

String bag\_name;

int bag\_zips;

String bag\_colour;

Bag(String bag\_name,int bag\_zips){

this.bag\_name=bag\_ name;

this.bag\_zips=bag\_zips;

}

Bag(String bag\_name,int bag\_zips,String bag\_colour){

this.bag\_name=bag\_name;

this.bag\_zips=bag\_zips;

this.bag\_colour=bag\_colour;

}

Bag( ){ //zero argument constructor

}

public static void main(String args[]){

Bag b=new Bag("americantoorist",2 );

Bag b2=new Bag("priority",4,"black");

//Bag b3=new Bag();

System.out.println(b.bag\_name+" "+b.bag\_zips);

System.out.println(b2.bag\_name+" " +b2.bag\_zips+" "+b.bag\_colour);

}

}

}

STATIC :

1. The static keyword in java is used for memory management.

2. We can apply static keyword with variables, methods, blocks and nested classes

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

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

5. The static variable can be used to refer to the common property of all variables.

example:

package com.test.Static;

public class Employe {

String employe\_name;

int employe\_id;

public static String employe\_company="PYDAH";

public static void main(String args[]){

Employe e=new Employe();

e.employe\_name="siva";

e.employe\_id=112441;

System.out.println(e.employe\_name);

System.out.println(e.employe\_id);

System.out.println(e.employe\_company);

Employe e1=new Employe();

System.out.println(e1.employe\_company);

}

}

THIS :

1.This is a keyword which is used to represent the current class variables and methode

2.this can be used to refer current class instance variable.

3.this can be used to invoke current class method (implicitly)

4.this() can be used to invoke current class constructor.

5.this can be passed as an argument in the method call.

6.this can be passed as argument in the constructor call.

7.this can be used to return the current class instance from the method.

In java, this is a **reference variable** that refers to the **current object**.



EXAMPLE:

class Student{

int rollno;

String name;

float fee;

Student(int rollno,String name,float fee){

this.rollno=rollno;

this.name=name;

this.fee=fee;

}

void display(){System.out.println(rollno+" "+name+" "+fee);

}

}

public static void main(String args[]){

Student s1=new Student(111,"ankit",5000f);

Student s2=new Student(112,"sumit",6000f);

s1.display();

s2.display();

}

SUPER:

1.The super keyword in Java is a reference variable which is used to refer immediate parent class object.

2.super can be used to refer immediate parent class instance variable.

3.super can be used to invoke immediate parent class method.

4.super() can be used to invoke immediate parent class constructor.

5.To call methods of the superclass that is overridden in the subclass.

6.To access attributes (fields) of the superclass if both superclass and subclass have attributes with the same name.

7.To explicitly call superclass no-arg (default) or parameterized constructor from the subclass constructor.

ENCAPSULATION:

1. Encapsulation is a process of declaring the Data members and member functions as a private.

2. By using encapsulation we are going to provide security for the class.

3. If i declare any variables and methods as a private we can't use those at out side of the class.

4. But java main theme was reusabulity,but here we are breaking that concept.

5. If we want to use those variables at out side of the class means we have to provide setters and getters for that variables .

6. It is a information hiding concept .

7. If one class has 10 variables if i can declare 10variables has a private that class has tighly encapsulated class .

Exmple:

package com.test.Encapsulation;

public class Shirt {

public String Shirt\_colour;

private String shirt\_brand;

private int shirt\_code;

public String getShirt\_brand() {

return shirt\_brand;

}

public void setShirt\_brand(String shirt\_brand) {

this.shirt\_brand = shirt\_brand;

}

public int getShirt\_code() {

return shirt\_code;

}

public void setShirt\_code(int shirt\_code) {

this.shirt\_code = shirt\_code;

}

public static void main(String args[]){

Shirt s=new Shirt();

s.Shirt\_colour="red";

s.shirt\_brand="jocky";

s.shirt\_code=2563245;

System.out.println(s.Shirt\_colour+" "+s.shirt\_brand+" "+s.shirt\_code);

Roadster u= new Roadster();

u.setShirt\_code(123456);

System.out.println(u.getShirt\_code());

u.setShirt\_brand("hailanders");

System.out.println(u.getShirt\_brand());

System.out.println(s.Shirt\_colour);

}

}

class Roadster extends Shirt{

}

INHERITANCE:

1.Inheritance is a concept which one class acquires all the properties and methods from a super class .

2. By using extends key word we are going to achieve inheritance concept.

3. By using inheritance concept we can create parent,child relation between two classes

4. You can reuse variables and methods from super class .

5. You can add new methods and fields in your current class also .

6. The main theam of inheritance was code reusabulity .

Syntax:-

class subclass extends superclass{

fieldes ;

methods;

}

Example:

public class Toothpaste {

public String toothpaste\_name;

String toothpaste\_colour;

int toothpast\_id;

public static void main(String[] args) {

Toothpaste t=new Toothpaste();

t.toothpaste\_name="glizer";

t.toothpaste\_colour="white";

t.toothpast\_id=123456;

System.out.println(t.toothpaste\_name+" "+t.toothpaste\_colour+" "+t.toothpast\_id);

Colgate c=new Colgate();

c.toothpaste\_name="meswak";

c.toothpaste\_colour="red";

c.toothpast\_id=456123;

System.out.println(c.toothpaste\_name+" "+c.toothpaste\_colour+" "+c.toothpast\_id);

System.out.println(t.toothpast\_id);

}

}

class Colgate extends Toothpaste{

}

**MEHOD OVERRIDING**:

1. I have some method declare in super class, in that method some functionality is there,i am not satisfying that functionality,i am going to be write new functionality in that method what ever i want.

2. Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.

3. Method overriding is used for runtime polymorphism

4. The method must have the same name as in the parent class

5. Method overriding concept will be existing in inheritance concept .

6. Method over riding can be override child class only not parent class .

7. Method overriding is the example of run time polymorphism.

Example:

package com.test.MethodoverRiding;

public class Pen {

public void pen\_writing(){

System.out.println("20 day to write");

}

public static void main(String args[]) {

Cello c=new Cello();

c.pen\_writing();

}

}

class Cello extends Pen{

public void pen\_writing(){

System.out.println("25 day to write");

}

}

METHOD OVERLOADING:

1. Method overloading means we can create multiple methods with same name in a class but different in parameters, it is known as Method Overloading.

2. By changing the no of parameters.by changing the order of parameters, incresing the parameters

3. Method overloading increases the readability of the program.

4. Method overloading is the example of compile time polymorphism.

EXAMPLE:

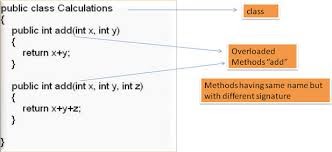
Different ways to overload the method,There are two ways to overload the method in java

1) By changing number of arguments

2) By changing the data type

1)By changing the no. of arguments:-

In this example, we have created two overloaded methods, first sum method performs addition of two numbers and second sum method performs addition of three numbers.



class Calculation {

void sum(int a, int b) {

System.out.println(a + b);

}

void sum(int a, int b, int c) {

System.out.println(a + b + c);

}

public static void main(String args[]) {

Calculation obj = new Calculation();

obj.sum(10, 10, 10);

obj.sum(20, 20);

}

}

Output:30

40

2) By changing data type of argument:-

In this one, we have created two overloaded methods that differs in data type. The first sum method receives two integer arguments and second sum method receives two double arguments.

class Calculation2 {

void sum(int a, int b) {

System.out.println(a + b);

}

void sum(double a, double b) {

System.out.println(a + b);

}

public static void main(String args[]) {

Calculation2 obj = new Calculation2();

obj.sum(10.5, 10.5);

obj.sum(20, 20);

}

}

Output:21.0

40

Example:

public class Brush {

String brush\_name;

String brush\_colour;

int brush\_cost;

public String brushtotal\_details(String brush\_name,String brush\_colour){

String brushtotal\_details=brush\_name+brush\_colour;

return brushtotal\_details;

}

public String brushtotal\_details(String brush\_name,String brush\_colour,int brush\_cost){

String brushtotal\_details=brush\_name+brush\_colour+ brush\_cost;

return brushtotal\_details;

}

public static void main(String[] args) {

Brush b=new Brush();

b.brushtotal\_details("colgate", "red");

System.out.println(b.brushtotal\_details("colgate", "red"));

b.brushtotal\_details("zigzag", "white", 50);

System.out.println( b.brushtotal\_details("zigzag", "white", 50));

}

POLYMORPHISM:

1. Polymorphism in Java is a concept by which we can perform a single action in different ways.

2. There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

Upcasting

If the reference variable of Parent class refers to the object of Child class, it is known as upcasting.

EXAMPLE:

package com.test.Polymorphism;

public class Tv {

void tv\_cost(){

System.out.println("20000 rupies");

}

public static void main(String args[]){

Tv t;

t=new Sovea();

t.tv\_cost();

t=new Tv();

t.tv\_cost();

}

}

class Sovea extends Tv{

void tv\_cost(){

System.out.println("2 lac");

}

}

FINAL:

1.The final keyword in java is used to restrict the user.

2.The java final keyword can be used

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

EXAMPLE:

[ There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because

final variable once assigned a value can never be changed.]

class Bike9{

final int speedlimit=90; //final variable

void run() {

speedlimit=400;

}

public static void main(String args[]){

Bike9 obj=new Bike9();

obj.run();

}

}

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

EXAMPLE:

class Bike{

final void run(){

System.out.println("running");

}

}

class Honda extends Bike{

void run(){

System.out.println("running safely with 100kmph");

}

public static void main(String args[]){

Honda honda= new Honda();

honda.run();

}

}

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

EXAMPLE:

final class Bike{}

class Honda1 extends Bike{

void run(){

System.out.println("running safely with 100kmph");

}

public static void main(String args[]){

Honda1 honda= new Honda1();

honda.run();

}

}

ABSTRACTION:

1. Abstraction is a process of hiding the implementation details and showing only method name to the user.

2. A class which is declared with the abstract keyword is known as an abstract class .

3. In abstract class you can declare

I . Absrtact methods (method with out the body).

II . Non-abstract methods &concrete methods (method with the body).

EXAMPLE:

package com.test.Abstractclass;

abstract class Animal {

abstract void animal\_sound();

public void sleep(){

System.out.println("only relize co");

}

public static void main(String[] args) {

Animal p=new Pig();

p.animal\_sound();

p.sleep();

}

}

class Pig extends Animal{

void animal\_sound() {

System.out.println("uhhuhhh");

}

}

package com.test.Abstractclass;

// ANOTHER CLASS

public class Dog extends Pig {

public static void main(String args[]){

Dog t=new Dog();

t.animal\_sound();

t.sleep();

}

}

INTERPHACE:

1. An interface is declared by using the interface keyword.

2. The interface in Java is a mechanism to achieve abstraction.

3. There can be only abstract methods in the Java interface, not method body.

4. It is used to achieve abstraction and multiple inheritance in Java.

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

Syntax:

interface <interface\_name>{

// declare constant fields

// declare methods that abstract

// by default.

}



EXAMPLE:

**interface** Bank{

**float** rateOfInterest();

}

**class** SBI **implements** Bank{

**public** **float** rateOfInterest(){

**return** 9.15f;

}

}

**class** PNB **implements** Bank{

**public** **float** rateOfInterest(){

**return** 9.7f;

}

}

**public** **static** **void** main(String[] args){

Bank b=**new** SBI();

System.out.println("ROI: "+b.rateOfInterest());

}



EXAMPLE:

**interface** Printable{

**void** print();

}

**interface** Showable{

**void** show();

}

**class** A7 implements Printable,Showable{

**public** **void** print(){

System.out.println("Hello");

}

**public** **void** show(){

System.out.println("Welcome");

}

**public** **static** **void** main(String args[]){

A7 obj = **new** A7();

obj.print();

obj.show();

 }

}

ABSTRACTIN:

1.Abstract class does not support multiple inheritance.

2.A class can extend only one abstract class.

3.An abstract class can have both abstract and non-abstract methods.

4.It can have methods with implementations.

INTERFACE:

1. interface support multiple inheritance

2.A class can implement more than one interface.

3.An interface can have only abstract methods.

4.It provides absolute abstraction and cannot have method implementations.

**STRING:**

**String** is basically an object that represents sequence of char values.

**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.

.

String class implements  Serializable, Comparable and CharSequence  interfaces



## 4. CharSequence Interface: The CharSequence interface is used to represent the sequence of characters. String, [StringBuffer](https://www.javatpoint.com/StringBuffer-class) and [StringBuilder](https://www.javatpoint.com/StringBuilder-class) classes implement it. It means, we can create strings in java by using these three classes.



5.The java.lang.String class is used to create a string object.

6. there are two ways to create String object:

By string literal

By new keyword

**Note: String objects are stored in a special memory area known as the "string constant pool".**

# Java String concat():

The java string concat() method combines specified string at the end of this string. It returns combined string. It is like appending another string

EXAMPLE:

public class ConcatExample3 {

public static void main(String[] args) {

String str1 = "Hello";

String str2 = "Javatpoint";

String str3 = "Reader";

// Concatenating Space among strings

String str4 = str1.concat(" ").concat(str2).concat(" ").concat(str3);

System.out.println(str4);

// Concatenating Special Chars

String str5 = str1.concat("!!!");

System.out.println(str5);

String str6 = str1.concat("@").concat(str2);

System.out.println(str6);

O/P:

Hello Javatpoint Reader

Hello!!!

Hello@Javatpoint

# Immutable String in Java

# In java, string objects are immutable. Immutable simply means unmodifiable or unchangeable.

Once string object is created its data or state can't be changed, but a new string object is created.

EXAMPLE:

. class Testimmutablestring{

public static void main(String args[]){

String s="Sachin";

s.concat(" Tendulkar");

// we are new obj was created .we are not create any reference variable//

//concat() method appends the string at the end (here we are not creating obj.)

System.out.println(s); //will print Sachin because strings are immutable objects.

}

}

O/P:

Sachin

//EXAMPLE:2:

class Testimmutablestring1{

public static void main(String args[]){

String s="Sachin";

s=s.concat(" Tendulkar");

System.out.println(s);

}

}

O/P:

Sachin Tendulkar

**java string comparison:**

We can compare string in java on the basis of content and reference.

It is used in authentication (by equals() method), sorting (by compareTo() method), reference matching (by == operator) etc.

There are three ways to compare string in java:

**Java String** **equals():**

The java string equals() method compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true

The String equals() method overrides the equals() method of Object class.

**Signature:**

**public boolean** equals(Object anotherObject)

**Returns**

**True** if characters of both strings are equal otherwise **False**.

**EXAMPLE:**

public class EqualsExample {

public static void main(String[] args) {

String s1 = "javatpoint";

String s2 = "javatpoint";

String s3 = "Javatpoint";

System.out.println(s1.equals(s2)); // True because content is same

if (s1.equals(s3)) {

System.out.println("both strings are equal");

}else System.out.println("both strings are unequal");

}

}

O/P:

true

both strings are unequal

**String compare by equals() method**

public boolean equals(Object another) compares this string to the specified object.

public boolean equals IgnoreCase(String another) compares this String to another string, ignoring case.

class Teststringcomparison2{

public static void main(String args[]){

String s1="Sachin";

String s2="SACHIN";

System.out.println(s1.equals(s2)); //false

System.out.println(s1.equalsIgnoreCase(s2)); //true

}

}

O/P:

false

true

**2) String compare by == operator**

The = = operator compares references not values.

**EXAMPLE:**

class Teststringcomparison3{

public static void main(String args[]){

String s1="Sachin";

String s2="Sachin";

String s3=new String("Sachin");

System.out.println(s1==s2); //true (because both refer to same instance)

System.out.println(s1==s3); //false(because s3 refers to instance created in nonpool)

}

}

O/P: true

false

# Java String compareTo():

The java string compareTo() method compares the given string with current string lexicographically. It returns positive number, negative number or 0.

It compares strings on the basis of Unicode value of each character in the strings.

If first string is lexicographically greater than second string, it returns positive number (difference of character value). If first string is less than second string lexicographically, it returns negative number and if first string is lexicographically equal to second string, it returns 0.

if s1 > s2, it returns positive number

if s1 < s2, it returns negative number

if s1 == s2, it returns 0

**String Concatenation in Java**

In java, string concatenation forms a new string that is the combination of multiple strings.

There are two ways to concat string in java:

1. By + (string concatenation) operator

2. By concat() method

**1) String Concatenation by + (string concatenation) operator**

Java string concatenation operator (+) is used to add strings.

EXAMPLE:

class TestStringConcatenation1{

public static void main(String args[]){

String s3="sai"+"siva";

System.***out***.println(s3);

o/p:

saisiva

Example:

String s8=25+52+"sai"+32+25;

System.***out***.println(s8);

o/p:

77sai3225

**Note**: After a string literal, all the + will be treated as string concatenation operator

**2. by Concat() method** : The String concat() method concatenates the specified string to the end of current string.

**Syntax:**

public String concat(String another)

**EXAMPLE**:

String s9="sai";

String s10="kumarreddy";

System.***out***.println(s9.concat(s10));

**o/p**:

saikumarreddy

**Substring in Java**

1. A part of string is called substring. In other words, substring is a subset of another string.

2. In case of substring startIndex is inclusive and endIndex is exclusive.

**Note:** Index starts from 0.

You can get substring from the given string object by one of the two methods:

1**. public String substring(int startIndex):** This method returns new String object containing the substring of the given string from specified startIndex (inclusive).

**2. public String substring(int startIndex, int endIndex):** This method returns new String object containing the substring of the given string from specified startIndex to endIndex.

In case of string:

1. startIndex: inclusive

2. endIndex: exclusive

**EXAMPLE**:

String s6="saikumarreddy";

System.out.println(s6.substring(1));

System.out.println(s6.substring(2,8 ));

**O/P**:

aikumarreddy

ikumar

**Java String class methods**

The java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.

Java String is a powerful concept because everything is treated as a string if you submit any form in window based, web based or mobile application.

Let's see the important methods of String class.

**Java String toUpperCase() and toLowerCase() method:**

The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method into lowercase letter.

**EXAMPLE**:

String s6="saikumarreddy";

System.out.println(s6.toUpperCase());

System.out.println(s6.toLowerCase());

**O/P**:

SAIKUMARREDDY

saikumarreddy

**Java String trim() method:**

The string trim() method eliminates white spaces before and after string.

**NOTE:**

The string trim() method doesn't omits middle spaces.

**EXAMPLE:**

String s6=" saikumarreddy ";

System.out.println(s6.trim());

**O/P**:

Saikumarreddy

**Java String startsWith() and endsWith() method**

**EXAMPLE:**

String s6=" saikumarreddy ";

System.out.println(s6.startsWith(" sai"));

System.out.println(s6.endsWith("y"));

**O/P**:

true

false

**Java String charAt() method:**

The string charAt() method returns a character at specified index.

**EXAMPLE**:

String s6="saikumarreddy ";

String s6=" saikumarreddy ";

System.out.println(s6.charAt(4));

System.out.println(s6.charAt(10));

**O/P:**

u

e

**Java String length() method:**

The string length() method returns length of the string.

**EXAMPLE:**

String s6="saikumarreddy";

System.out.println(s6.length());

**O/P**:

13

**Java String intern() method:**

Java String valueOf() method

**Java String replace() method:**

The string replace() method replaces all occurrence of first sequence of character with second sequence of character.

**EXAMPLE**:

String s7="I am a boy. I am learning java programing.";

String replaceString=s7.replace("I","Me");

System.out.println(replaceString);

**o/p**:

Me am a boy. Me am learning java programing.

**String :**

1. string is basically an object that represents sequence of char values.

2.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.

3. String class implements Serializable, Comparable and CharSequence interfaces.

4. String class is immutable.

5. String is slow and consumes more memory when you concat too many strings because every time it creates new instance.

6. the object created as a string is stored in the Constant string pool.every immutable object in java is thread safe,that implies string is also thread safe. string can not be used by two threads simultaneously.

7. since once assigned can not be changed.

8. string can be concatenated by using "+" operator or concat() method.

**StringBuffer:**

1. String buffer is mutable.

2. StringBuffer is fast and consumes less memory when you concat strings.

3. StringBuffer is less efficient than StringBuilder.

4. StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.

5. the object created through string buffer is stored in the heap .

6. string buffer can be concatenated by using append().

7. string buffer performance is low.

**String builder**:

1. string builder is mutable.

2. StringBuilder is more efficient than StringBuffer.

3. StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.

4. The StringBuilder class is introduced since JDK 1.5.

5. the object created through string buffer is stored in the heap .

6. string builder performance is high.

**Java StringBuffer class**

**What is mutable string:**

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

**1) StringBuffer append() method:**

The append() method concatenates the given argument with this string.

**EXAMPLE:**

class StringBufferExample{

public static void main(String args[]){

StringBuffer sb1=new StringBuffer("apple");

sb1.append("java");

System.out.println("append "+sb1);

}

}

**O/P:**

append applejava

**2) StringBuffer insert() method:**

The insert() method inserts the given string with this string at the given position.

**EXAMPLE:**

class StringBufferExample2{

public static void main(String args[]){

StringBuffer sb1=new StringBuffer("apple");

sb1.insert(1, "program");

System.out.println("insert "+sb1);

}

}

**O/P:**

insert aprogrampple

**3) StringBuffer replace() method:**

The replace() method replaces the given string from the specified beginIndex and endIndex.

**EXAMPLE:**

class StringBufferExample3{

public static void main(String args[]){

StringBuilder sb=new StringBuilder("saikumarreddy");

sb.replace(2,8,"siva");

System.out.println("replace "+sb);

}

}

**O/P:**

replace sasivareddy

**4) StringBuffer delete() method:**

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

**EXAMPLE:**

class StringBufferExample4{

public static void main(String args[]){

StringBuilder sb=new StringBuilder("saikumarreddy");

sb.delete(3,5);

System.out.println("delete "+sb);

}

}

**O/P:**

delete saimarreddy

**5) StringBuffer reverse() method:**

The reverse() method of StringBuilder class reverses the current string.

**EXAMPLE:**

class StringBufferExample5{

public static void main(String args[]){

StringBuilder sb=new StringBuilder("saikumarreddy");

sb.reverse();

System.out.println("reverse "+sb);

}

}

**O/P:**

reverse ydderramukias

**How to create Immutable class?**

There are many immutable classes like S1tring, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable.

We can also create immutable class by creating final class that have final data members as the example given below:

**Example to create Immutable class**

In this example, we have created a final class named Employee. It have one final datamember, a parameterized constructor and getter method.

public final class Employee{

final String pancardNumber;

public Employee(String pancardNumber){

this.pancardNumber=pancardNumber;

}

public String getPancardNumber(){

return pancardNumber;

}

}

The above class is immutable because:

1. The instance variable of the class is final i.e. we cannot change the value of it after creating an object.

2. The class is final so we cannot create the subclass.

3. There is no setter methods i.e. we have no option to change the value of the instance variable.

These points makes this class as immutable.

**Java String join():**

The java string join() method returns a string joined with given delimiter. In string join method, delimiter is copied for each elements.

In case of null element, "null" is added. The join() method is included in java string since JDK 1.8.

There are two types of join() methods in java string.

**Signature:**

The signature or syntax of string join method is given below:

1. public static String join(CharSequence delimiter, CharSequence... elements)

and

2. public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)

**Parameters**

delimiter : char value to be added with each element

elements : char value to be attached with delimiter

EXAMPLE:

public class StringJoinExample{

public static void main(String args[]){

String join=String.join("-","sai", "is", "a boy");

System.out.println(join);

String date=String.join("/", "04","03","2020");

System.out.println("join "+date);

String time=String.join(":", "07","32");

System.out.println("join "+time);

}

}

O/P:

sai-is-a boy

join 04/03/2020

join 07:32

**EXCEPTION HANDLING**

**Exception:**

Exception is an unwanted, unexpected event that disrupts normal flow of the program is known as exception

----------

**Exception Handling:**

Exception handling is defining alternative ways to continue rest of the program normally

-----------

1. Every thread jvm will create separate stack. Every method call performed by the thread one entry will be stored in the corresponding stack.

2. Once the method execution completed corresponding entry will be removed.

3. After execution all methods calls now stack become empty before terminating thread jvm will destroy the corresponding stack this mechanism is known as runtime stack mechanism.

each entry is calls activation record or stack frame.

**Types of Java Exceptions:**

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

exception is checked exception or unchecked exception.

Compulsory every exception should occurs at runtime only. there is no chance of occurring any exception at compile time those errors are syntactical errors

**1. Checked Exception**: A checked exception is an exception that is checked (notified) by the compiler at compilation-time, these are also called as compile time exceptions. These exceptions cannot simply be ignored, the programmer should take care of (handle) these exceptions.

**2. Unchecked Exception:** An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.

**3. Error:** 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.

**EXCEPTION HANDLING:**

1. avoiding abnormal termination of program

when exception is rised the program will be terminated abnormally.

**You will use five keywords mainly:**

1. try

2. catch

3. finally

4. throw

5. throws

**try:**

1. It is block of statements.

2. In this we have to write the code in which we are expecting the exceptions.

3. If an exception occurs at the particular statement of try block, the rest of the block code will not execute.

4. So, it is recommended not to keeping the code in try block that will not throw an exception.

5. Java try block must be followed by either catch or finally block.

**catch:**

1. It is block of statements

2. Handles exceptions raised in try block

3. Without try block you should not write the catch block.

4. Java catch block is used to handle the Exception by declaring the type of exception within the parameter.

5. The declared exception must be the parent class exception ( i.e., Exception) or the generated exception type.

6. The catch block must be used after the try block only. You can use multiple catch block with a single try block.

**Syntax:**

try {

-------

-----

}

catch(Exception e)

{

------

----------

}

1. **Checked exceptions**: it is also known as compile time exceptions.

Ex:

I/O exception

File not found exception

Sql exception

2. **Unchecked exceptions**: these exceptions are not checked at the compile time.it is also known as run time exceptions.

Example of run time exceptions is:

I) **ArithmeticExceptions:**

/ By zero.

ex: 25/0 such type of exception called ArithmeticException.

II) **NumberFormatException:**

String str=Hello;

Converting string to integer

"5" is string converted in to integer "5" , but

" hello" can be converted into

int num=Integer.parscInt(str);

III) **ArrayIndexOutOfBoundException:**

Array means number (multiple) of elements, access those elements that can be done through INDEX values, if the index values OUTOFBOUND then these type of exceptions can be occurred.

Int a[ ]=new int[5];// hear index range values are 0 to 4

a [6]=15; these are out of boundary because above you can take the range 0 to 4

so in this such case ArrayIndexOutOfBoundException willbe raised.

**IV) NullPointerException:**

String str=null;

syso(str.length());

in the try block we have to write the code we are expecting the exceptions and catch block we have to write the all those exceptions.

ArithmeticExceptions:

NumberFormatException:

ArrayIndexOutOfBoundException:

NullPointerException:

those are available in throwable class

EXAMPLE:

package com.test.ExceptionHandling;

public class ExceptionExample {

public static void main(String[] args) {

int b=7/2;

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

try{

int a=5/0;

}

catch(ArithmeticException e){

System.out.println(e);

}

try{

int e[]=new int[5];

e[6]=4;

}

catch(ArrayIndexOutOfBoundsException e){

System.out.println(e);

}

try{

String str="siva";

int num=Integer.parseInt(str);

}

catch(NumberFormatException e)

{

System.out.println(e );

}

try{

String str=null;

String str1="siva";

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

System.out.println(str1.length());

}

catch(NullPointerException e){

System.out.println(e);

}

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

}

}

O/P:

rises......

java.lang.ArithmeticException: / by zero

java.lang.ArrayIndexOutOfBoundsException: 6

java.lang.NumberFormatException: For input string: "siva"

java.lang.NullPointerException

exceptionnnnn......

**3. Errors:**

I) logic errors

II) Syntax errors

III) I/o errors

**NESTED TRY BLOCK:**

try block with in two try blocks is known as NESTED TRY BLOCK

SYNTAX:

try {

try{

//statemente to execute

}

catch(exception e)

{

System.out.println( e);

}

try{

int a[]=new int[5];

a[5]=4;

}

}

**THROW:**

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

2. Checked exception cannot be propagated using throw only.

3. Throw is followed by an instance.

4. Throw is used within the method.

5. You cannot throw multiple exceptions

6. SYNTAX:

void m(){

throw new ArithmeticException("sorry");

}

**THROWS:**

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

2. Checked exception can be propagated with throws.

3. Throws is followed by class.

4. Throws is used with the method signature.

5. You can declare multiple exceptions e.g. public void method() throws IOException, SQLException.

6. SYNTAX:

void m()throws ArithmeticException{

//method code

}

**FINAL:**

1. Final is a keyword.

2. Final is used to apply restrictions on class, method and variable.

3. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.

4. Final method is executed upon its call.

EXAMPLE:

public class Exception4 {

static int a=5;

final int b=63;

public static void main(String[] args) {

Exception4 e=new Exception4();

e.a=52;

e.b=25;//hear b s final we cannot change the value

}

}

**FINALLY:**

1. In Java “finally” is a block that is always associated with the try/catch block.

2. The “finally” block executes after the try/catch block and before the code following try/catch block.

3. The “finally” block will execute whether the exception is thrown or not.

4. When an exception is thrown, and no catch block matches the exception even then, the “finally” block is executed.

5. When a method returns to the caller from inside the try/catch block via an uncaught exception or an explicit return statement,

the “finally” block gets executed

6. just before the method returns to the caller.

7. The “finally” block is used to clean up the resources or free the memory used in “try” block.

8. The “finally” block is optional, but it is a good practice to write finally block after the try/catch block.

EXAMPLE:

public class Exception4 {

public static void main(String[] args) {

try{

int a[]=new int[5];

a[6]=4;

} catch(ArrayIndexOutOfBoundsException a){

System.out.println("bounce the exception");

}

finally{

System.out.println("final can be exicutes");

}

}

}

OUTPUT:

bounce the exception

final can be exicutes

**FINALIZE:**

1. Finalize is a method.

2. Finalize is used to perform clean up processing just before object is garbage collected.

3. finalize() method executes just before the destruction of the object.

4. An object may be holding some non-java resources such as file handle; then it must be freed before it is destroyed.

5. The finalize method is declared as protected so that it cannot be accessed from outside the class.

6. This method is always called before the garbage collection.

7. SYNTAX:

Protected void finalize ( )

{

// finalization code here

}

EXAMPLE:

public class Exception4 {

public void finalize(){

System.out.println("finalizing is calling");

}

public static void main(String[] args) {

Exception4 e=new Exception4();

e.finalize();

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

}

}

OUTPUT:

finalizing is calling

final.....

IO stream:

System.in: reads the data from keyboard

System.out: display the data on screen

System.err: display the error msgs

those are available in java.lang packages.

import java.lang.\*;//\* each and every class which belongs to the lang package will be accessed.

output:

system.out.print: every thing will be printed in a single line

system.out.println: output will be printed in multiple lines

**Input:**

**1. Scanner class:** reads the data from key board. this is available in UTIL package.

these Scanner class can implements UTIL package cam be implemented.

every classs name will be starting with capital letters

Import java.util.Scanner;

the import java.util.Scanner, scanner class can be imported to our program.

Import java.util.\*;

whatever the classes available in util package everything we can use this program

METHODES:

next( ): it is used to read string

nextInt( ): read integer

next Float( ): read flat value

nextDouble( ): read double value

In order to access the methods first one obj should crated for the scanner class and through that obj by using the dot(.) operator we have to use these methods.

Scanner sc // sc is the obj( instance) of Scanner class

sc.next( )

sc.nextInt( )

sc.nextFloat( )

sc.nextDouble( )

Creation of obj: classname objname=new constructor( )

Scanner sc=new Scanner(System.in)

**2. BufferReader class**: reads the data from key board or file(existing file).this is available in IO package. the buffer reader reads the characters from char inputstream

**InputStreamReader**: it will reades bites and decode to char set. this id done with the help of system.in

**File Reader**: it is used to read the data from files.

All the data to save to buffer and use buffer reader to access or read the data

Those are two methods read the input to the run time

import java.io.\*;

We want to implement to the BufferReader we can import to the IO package

METHODS:

Will reads only char/strings

read( ): reads single char

readline( ): reads multiple char// simply it is a string

Integer. parseInt( --)//use read( ) and readline( ) methods

Float.parseFloat(-- )//use read( ) and readline( ) methods

InputStreamReader ir=new InputStreamReader(System.in)

BufferedReader br=new BufferedReader( ir)

int a=Integer.parseInt( br .readLine( ) ) ;

float b=Float.parseFloat(br.readline( ) ) ;

string str=br.readline( ) ;

FileReader fr=new FileReaer("D:\\java\\sai.txt");

BufferedReader br=new BufferedReader( fr)

**COLLECTONS**

**Collections in Java**

The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many

interfaces

(Set, List, Queue, Deque) and

classes

( ArrayList,Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

**Hierarchy of Collection Framework:**

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



**ARRAYLIST**:

1. Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

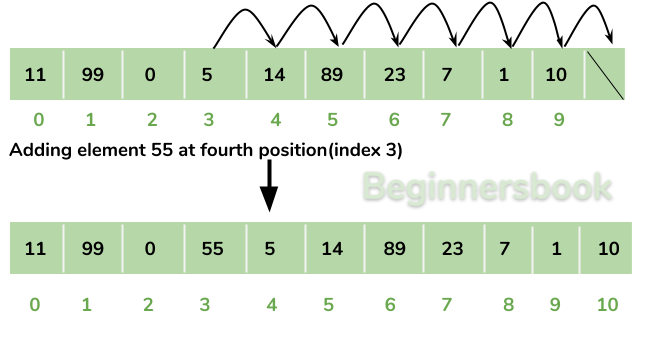
2. Java ArrayList class can contain duplicate elements.

3. Java ArrayList class maintains insertion order.

4. Java ArrayList class is non synchronized.

5. Java ArrayList allows random access because array works at the index basis.

6. In Java ArrayList class, manipulation is slow because a lot of shifting needs to occur if any element is removed from the array list.





**How to create an ArrayList?**

ArrayList<String> alist=new ArrayList<String>();

ArrayList<Integer> list=new ArrayList<Integer>();

**How to add elements to an ArrayList?**

We add elements to an ArrayList by using add() method, this method has couple of variations, which we can use based on the requirement.

EXAMPLE:

import java.util.ArrayList;

public class Student {

public static void main(String args[]){

ArrayList al=new ArrayList();

al.add("ramu");

al.add("siva");

al.add("mani");

al.add(2);

System.out.println(al);

al.add(2, "sai");

System.out.println(al);

}

}

O/P:

[ramu, siva, mani, 2]

[ramu, siva, sai, mani, 2]

**Change an element in ArrayList**

We can use the set method to change an element in ArrayList. We provide the index and new element,

this method then updates the element present at the given index with the new given element.

EXAMPLE:

import java.util.ArrayList;

public class Student {

public static void main(String args[]){

ArrayList al=new ArrayList();

al.add("ramu");

al.add("siva");

al.add("mani");

al.add(2);

System.out.println("list of elements" +al);

al.add(2, "sai");

al.add("siva");

al.add("mom");

System.out.println("after adding"+al);

//set(int index, Object o): Used for updating an element. It replaces the element present at the specified index with the object o.

obj.set(1, "dad"); //

al.set(1,"dad");

System.out.println("change the element"+al);

}

}

O/P:

list of elements[ramu, siva, mani, 2]

after adding[ramu, siva, sai, mani, 2, siva, mom]

change the element[ramu, dad, sai, mani, 2, siva, mom]

**How to remove elements from ArrayList**

We use remove() method to remove elements from an ArrayList, Same as add() method, this method also has few variations.

EXAMPLE:

package com.practice.collections;

import java.util.ArrayList;

public class Student {

public static void main(String args[]){

ArrayList al=new ArrayList();

al.add("ramu");

al.add("siva");

al.add("mani");

al.add(2);

System.out.println("list of elements" +al);

al.add(2, "sai");

al.add("siva");

al.add("mom");

System.out.println("after adding"+al);

al.set(1,"dad");

System.out.println("after change the element"+al);

//remove(Object o): Removes the object o from the ArrayList.

obj.remove("sai");

This statement will remove the string “sai” from the ArrayList.

remove(int index): Removes element from a given index.

obj.remove(3); //

al.remove("sai");

System.out.println(" after remove elements"+al);

al.remove(3);

System.out.println(" after remove elements"+al);

}

}

O/P:

list of elements[ramu, siva, mani, 2]

after adding[ramu, siva, sai, mani, 2, siva, mom]

after change the element[ramu, dad, sai, mani, 2, siva, mom]

after remove elements[ramu, dad, mani, 2, siva, mom]

after remove elements[ramu, dad, mani, siva, mom]

**LINKEDLIST**

**Java LinkedList class**

1. Java LinkedList class uses a doubly linked list to store the elements.

2. It provides a linked-list data structure.

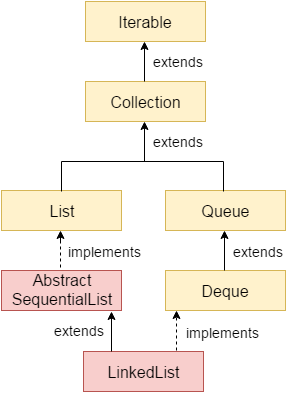
3. It inherits the AbstractList class and implements List and Deque interfaces.

4. Java LinkedList class can contain duplicate elements.

5. Java LinkedList class maintains insertion order.

6. Java LinkedList class is non synchronized.

7. In Java LinkedList class, manipulation is fast because no shifting needs to occur.



**Doubly Linked List**

In the case of a doubly linked list, we can add or remove elements from both sides.



EXAMPLE:

package com.practice.collections;

import java.util.LinkedList;

public class Book {

public static void main(String[] args) {

LinkedList ll=new LinkedList();

ll.add("a");

ll.add("b");

ll.add("c");

ll.add(2);

ll.add("d");

ll.add("e");

ll.add(1);

System.out.println(ll);

LinkedList ll2=new LinkedList();

ll2.addFirst("aa");

ll2.addLast("bb");

ll2.add(2, "cc");

ll.addAll(ll2);

System.out.println("after adding all elements" +ll);

ll.remove("aa");

ll.remove(2);

System.out.println("after removing elements" +ll);

ll.removeFirst();

System.out.println(ll);

ll.removeFirstOccurrence("b");

ll.removeLastOccurrence("cc");

System.out.println(ll);

}

}

O/P:

[a, b, c, 2, d, e, 1]

after addingall elements[a, b, c, 2, d, e, 1, aa, bb, cc]

after removing elements[a, b, 2, d, e, 1, bb, cc]

[b, 2, d, e, 1, bb, cc]

[2, d, e, 1, bb]

**How Java Iterator Works Internally?**

In this section, we will try to understand how Java Iterator and it’s methods works internally.

Let us take the following LinkedList object to understand this functionality.

**EXAMPLE:**

List<String> names = new LinkedList<>();

names.add("E-1");

names.add("E-2");

names.add("E-3");

.

.

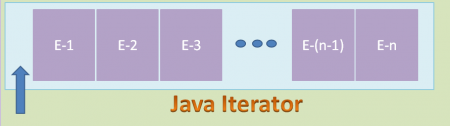
.

names.add("E-n");

//Now create an iterator object on List object as shown below//

Iterator<String> namesIterator = names.iterator();

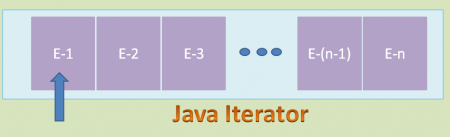
Let us assume “namesIterator” iterator looks like below:



Here Iterator’s Cursor is pointing to the before first element of the List. Now we run the following code snippet.

namesIterator.hasNext();

namesIterator.next();



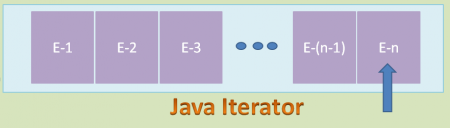
When we run the above code snippet, Iterator’s Cursor points to the first element in the List as shown in the above diagram. Then run the following code snippet.

namesIterator.hasNext();

namesIterator.next();



When we run the above code snippet, Iterator’s Cursor points to the second element in the List as shown in the above diagram. Do this process to reach the Iterator’s Cursor to the end element of the List.



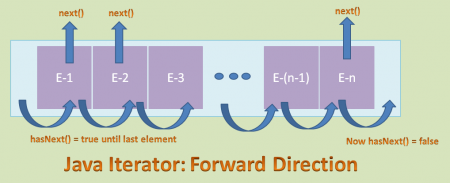
After reading the final element, if we run the below code snippet, it returns “false” value.

namesIterator.hasNext();



As Iterator’s Cursor points to the after the final element of the List, hasNext() method returns false value.

After observing all these diagrams, we can say that Java Iterator supports only Forward Direction Iteration as shown in the below diagram. So it is also know as Uni-Directional Cursor.



**Advantages of Java Iterator**

1. We can use it for any Collection class.

2. It supports both READ and REMOVE operations.

3. It is an Universal Cursor for Collection API.

4. Method names are simple and easy to use them.

**Limitations** **Java Iterator**

1. In CRUD Operations, it does NOT support CREATE and UPDATE operations.

2. It supports only Forward direction iteration that is Uni-Directional Iterator.

3. Compare to Spliterator, it does NOT support iterating elements parallel that means it supports only Sequential iteration.

4. Compare to Spliterator, it does NOT support better performance to iterate large volume of data.