**********MixTechSolutions**********

**********A New Way to Change your Life**********

Java:

DataType:

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

Primitive data types:

The primitive data types include boolean, char, byte, short, int, long,

float and double.

Non-primitive data types:

The non-primitive data types include Classes, Interfaces, and Arrays.

Java Keywords:

Java keywords are also known as reserved words. Keywords are particular words that act as a key to a code.

These are predefined words by Java so they cannot be used as a variable or object name or class name.

Ex: Abstract,break,continue etc.

Java Variables

A variable is a container which holds the value while the Java program is executed.

A variable is assigned with a data type.

Types of Variables

There are three types of variables in Java:

local variable

instance variable

static variable

1) Local Variable

A variable declared inside the body of the method is called local variable.

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

that the variable exists.

2) Instance Variable

A variable declared inside the class but outside the body of the method,

is called an instance variable.

3) Static variable

A variable that is declared as static is called a static variable.

Ex:

```
public class A
{
   static int m=100;//static variable
   int data=50;//instance variable
   void method()
   {
      int n=90;//local variable
   }
   public static void main(String args[])
   {

   }
}
```

Java provides three types of control flow statements.

Decision Making statements

if statements

switch statement

Loop statements

do while loop

while loop

for loop

for-each loop

Jump statements

break statement

continue statement

1) If Statement:

In Java, the "if" statement is used to evaluate a condition.

Syntax:

```
If(condition)
{
//code
}
```

2) if-else statement

The if-else statement is an extension to the if-statement,

which uses another block of code, i.e., else block.

The else block is executed if the condition of the if-block is evaluated as false.

```
If(condition)
{
//if condition is true this block will executed
}
else
{
//if condition is false this block will executed



}
```

3) if-else-if ladder:

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true.

We can also define an else statement at the end of the chain.

```
If(condition1)
{
//this block of code will executed
}
else if(condition2)
{
//this block of code will executed
}
else if(condition3)
{
//this block of code will executed
}
….
…
else
{
//this will executed if all conditions are false
}
```

4. Nested if-statement

In nested if-statements, the if statement

can contain a if or if-else statement inside another if or else-if statement.

Ex:

```
If(condition1)
{
//this is outer if condition
If(condition2)
{
//this is inner if condition
}

}
```

Switch Statement:

In Java, Switch statements are similar to if-else-if statements.

The switch statement contains multiple blocks of code called cases

And a single case is executed based on the variable which is being switched.

The switch statement is easier to use instead of if-else-if statements.

It also enhances the readability of the program.

Syntax:

Ex:

```
public class SwitchExample {
public static void main(String[] args) {
    int number=20;
    switch(number){
    case 10: System.out.println("10");
    break;
    case 20: System.out.println("20");
    break;
    case 30: System.out.println("30");
    break;
```

```
    default:System.out.println("Not in 10, 20 or 30");
    }
}
}
```

Loop:

Loop Statements In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order.

In Java, we have three types of loops that execute similarly.

1) **for loop**
2) **While Loop**
3) **Do While**

Type of for Loop:

1) Simple for loop

Syntax:

Initialization: i=0

Condition : i<10 or i>10

Increment/decrement: i++ /i--

```
for(int i=0;i<condition;i++)
{


}
```

2) for each loop /advanced for loop

```
for(data type variable: variable which hold or store value)
{
 System.out.println(variable);
```

```
}
```

3) Labeled for loop

```
 A:for(int i=initalization;i<condition;i++/i--)
  {

   break A;
  }
```

2) while loop

```
while(condition)
{

//syntax
}
```

3) do-while loop

do

{

//code..

}

while(condition);

ex:

```
public class Test10 {

    public void loop5()
    {

        int i=0;
        do
        {
        System.out.println(i);
        i++;
        }

        while(i==0);


    }

    public static void main(String[] args) {
        Test10 t1=new Test10();
```

```
        t1.loop5();
    }
}
```

Jump statements

break statement --> used to break statement

continue statement ---> used to continue statement

OOPs (Object-Oriented Programming System)

Object-Oriented Programming is a methodology to design a program using classes and objects.

Object

Class

Inheritance

Polymorphism

Abstraction

Encapsulation

Object:

Object means a real-world entity such as a pen, chair, table, computer, watch, etc.

Class:

It is collection or group of objects.

Inheritance:

Inheritance in Java is a mechanism in which one object

Acquires all the properties and behaviors of a parent object.

Why use inheritance in java

For Method Overriding (so runtime polymorphism can be achieved).

For Code Reusability.

The syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

The extends keyword indicates that you are making a new class that derives from an existing class.

The meaning of "extends" is to increase the functionality.

ex:

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

Types of Inheritance:

Single

Multilevel

Hierarichal

Multiple

Hybrid

Polymorphism in Java

Polymorphism in Java is a concept by which we can perform a single action

in different ways. Polymorphism is derived from 2 Greek words:

poly and morphs. The word "poly" means many and "morphs" means forms.

So polymorphism means many forms.

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.

Method Overloading:

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

Different ways to overload the method

There are two ways to overload the method in java

By changing number of arguments

By changing the data type

Method Overriding in Java:

If subclass (child class) has the same method as declared

in the parent class, it is known as method overriding in Java.

Abstract class in Java

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

It can have abstract and non-abstract methods (method with the body).

Abstraction in Java

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

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message.

You don't know the internal processing about the message delivery.

There are two ways to achieve abstraction in java

Abstract class (0 to 100%)

Interface (100%)

Abstract class:

Ex:

```
abstract class Bike{
  abstract void run();
```

```
}
class Honda4 extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

Interface:

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

The interface in Java is a mechanism to achieve abstraction

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

It is used to achieve abstraction and multiple inheritance in Java

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

It is used to achieve abstraction.

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

It can be used to achieve loose coupling.

Ex:

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

Encapsulation:

Encapsulation in Java is a process of wrapping code and data together

into a single unit,for example, a capsule which is mixed of several medicines.

Getter and setter are used to achieve this.

Ex:

```
//A Java class which is a fully encapsulated class.
//It has a private data member and getter and setter methods.
package com.javatpoint;
public class Student{
//private data member
private String name;
//getter method for name
public String getName(){
return name;
}
//setter method for name
public void setName(String name){
```

this.name=name

}

}

Constructor:

In Java, a constructor is a block of codes similar to the method.

It is called when an instance of the class is created.

There are two types of constructors in Java:

Default constructor (no-arg constructor)

Parameterized constructor

Ex:

```
public class Test9 {

    public Test9()
    {
```

```
        System.out.println("Default
Constructor");
    }

    public Test9(int a,int b)
    {
        System.out.println("Paramterized
constructor");
        System.out.println(a+b);
    }


    public static void main(String[] args) {

        Test9 t1=new Test9();

        Test9 t2=new Test9(2,3);


    }

}
```

static keyword

The static keyword in Java is used for memory management mainly.

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

The static keyword belongs to the class than an instance of the class.

The static can be:

Variable (also known as a class variable)

Method (also known as a class method)

Static Block

```
public class Test10 {

    public static int a=10; // static variable

    public static void addition(int a,int b,int
c)
    {

        System.out.println("This is static
method");
        System.out.println(a+b+c);
    }

    static
    {

        System.out.println("This is static
block");
    }
```

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

    }
}
```

Notes:

If you have to call static method or static variable Then

Syntax:

=➔ **ClassName.static variable or method name**

Means by using class name you can static variable or static method.

this keyword in Java :

this keyword

There can be a lot of usage of Java this keyword.

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

this can be used to refer current class instance variable.

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

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

Ex:

```
public class Test11 {

    int a=10;

    public Test11()
    {

        System.out.println("Default
constructor");
    }

    public Test11(int a,int b)
    {
        this();
        this.a=a;
        System.out.println(a+b);
    }

    public static void main(String[] args) {

         Test11 t1=new Test11();
        System.out.println(t1.a);
```

```
    }
}
```

Super Keyword in Java

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

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword:

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

super can be used to invoke immediate parent class method.

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

```
class Animal{
public Animal()
{
System.out.println("Super class constructor");
}
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
public Dog()
{
super();
System.out.println("Child class constructor");
}
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
super.eat();
bark();
}
}
```

```
class TestSuper2{

public static void main(String args[]){

Dog d=new Dog();

d.work();

}}
```

Final Keyword:

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

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

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

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

Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it. For Example:

```
class Bike{

final int a=100;
```

```
  final void run(){
System.out.println("running...");
}
}
class Honda2 extends Bike{
   public static void main(String args[]){
    new Honda2().run();
System.out.println(“ Final variable ”+a);
   }
}
```

Access Modifier:

There are four types of Java access modifiers:

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

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

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

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

Difference between Abstract Class VS Interface:

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.

3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Java Array:

Java array is an object which contains elements of a similar data type. Additionally, the elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Types of Array in java

There are two types of array.

- **Single Dimensional Array**
- **Multidimensional Array**

- Single Dimensional Array

Syntax to Declare an Array in Java

```
1.  dataType[] arr; (or)
2.  dataType []arr; (or)
3.  dataType arr[];
```

Ex:

```
1.  class Testarray1{
2.  public static void main(String args[]){
3.  int arr[]={33,3,4,5};
4.  //printing array using for-each loop
5.  for(int i:arr)
6.  System.out.println(i);
7.  }}
```

- **Multidimensional Array**

Example to instantiate Multidimensional Array in Java

```
int[][] arr=new int[3][3];//3 row and 3 column
```

Example to initialize Multidimensional Array in Java

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

Example of Multidimensional Java Array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

```
//Java Program to illustrate the use of multidimensional array
class Testarray3{
public static void main(String args[]){
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array
for(int i=0;i<3;i++){
 for(int j=0;j<3;j++){
  System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}
}}
```

String:

String is a class and it is immutable sequence of character.

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.**

Contains:

➔**This method to check partial String data from another String.**

```
public class Test10 {

    public static void main(String[] args) {
        String s1="Hello I am Anil";

        String s2="Hello";

        if(s1.contains(s2))
        {
            System.out.println("Match Found!!");
        }

        else
        {
```

```
            System.out.println("Please check
condition is false");
        }

    }
```

ToLowerCase:➔ Convert any String to lowercase

ToUpperCase: ➔ Convert any String to Upper case

SubString: ➔ cut small or full or required part from given String

Ex:

```
public class Test9 {

    public static void main(String[] args) {

        String s2="HELLO";

        String s3="i m here";

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

        System.out.println(s2.toLowerCase());
```

```
        String s4=s3.substring(2,5);
        System.out.println(s4);

    }

}
```

trim: ➔ This method is used to remove whitespaces from starting and end of character.

length:➔ This method is used to check length.

Ex:

```
public class Test7 {

    public static void main(String[] args) {
        String s1=" Hello Anil  ";

        System.out.println(s1.length());
        String s2=s1.trim();

        System.out.println(s2.length());

    }
```

}

Equals:➔ This method is used to compare the content with case sensitive, and if match found return true otherwise false.

== ➔ this method check address and if both are same address and same content with case sensitive then return true otherwise false.

Ex:

```
public class Test3 {

    public static void main(String[] args) {


        String s1="Hello";

        String s2="Hello";


        String s3=new String("Welcome");
        String s4=new String("Welcome");


        System.out.println(s1==s2);
```

```
        System.out.println(s3==s4);


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

        System.out.println(s3.equals(s4));


    }
```

Concat:➔ This method is used to append one string to another.

Ex:

```
public class Test2 {

    public static void main(String[] args) {
        String s1="hello";
        s1.concat("Anil");

        System.out.println(s1);



    }
```

```
}
```

IsEmpty:➔ This method is used to check given String is empty or not.

Ex:

```
public class Test14 {


    public static void main(String[] args) {


        String s1="";

        if(s1.isEmpty())
        {

            System.out.println("String is
empty");

        }

        else
        {
            System.out.println("String is not
blank!");
        }

    }
```

replace:➔ This method is used to replace character from given string.

Ex:

```
public class Test17 {

    public static void main(String[] args) {
        String s1="Hello I Am Anil";

        String s2=s1.replace("Anil",
"Gurpreet");
     System.out.println(s2);

    }

}
```

Split:➔This method is used to split any String on the basis of condition.

Ex:

```
1.  public class SplitExample{
2.  public static void main(String args[]){
3.  String s1="java string split method  ";
4.  String[] words=s1.split(" ");//splits the string based on whitespace
5.  //using java foreach loop to print elements of string array
6.  for(String w:words){
```

```
7.          System.out.println(w);
8.          }
9.          }}
```

Difference between String and StringBuffer

No.	String	StringBuffer
1)	The String class is immutable.	The StringBuffer class is mutable.
2)	String is slow and consumes more memory when we concatenate too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when we concatenate t strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.
4)	String class is slower while performing concatenation operation.	StringBuffer class is faster while performing concatenation operation.
5)	String class uses String constant pool.	StringBuffer uses Heap memory

StringBuffer vs String Builder

No.	StringBuffer	StringBuilder
1)	StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is *less efficient* than StringBuilder.	StringBuilder is *more efficient* than StringBuffer.
3)	StringBuffer was introduced in Java 1.0	StringBuilder was introduced in Java 1.5

Exception:

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.

Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

1. Checked Exception
2. Unchecked Exception
3. Error

Difference between Checked and Unchecked Exceptions

1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Keywords

Java provides five keywords that are used to handle the exception. The following table describes each.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Try-Catch Example:

Ex:

```
1.  public class JavaExceptionExample{
2.   public static void main(String args[]){
3.    try{
4.      //code that may raise exception
5.      int data=100/0;
6.    }catch(ArithmeticException e){System.out.println(e);}
7.    //rest code of the program
8.    System.out.println("rest of the code...");
9.   }
10. }
```

Throw example:

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;

public class Test1 {

    public static void main(String[] args)
throws FileNotFoundException {


        FileInputStream file=new
FileInputStream("E:\\abc.txt");

    }
```

```
}
```

throw example:

```
import java.io.FileNotFoundException;

public class Test1 {

    public static void main(String[] args)
throws FileNotFoundException {

        int age=19;
        if(age>18)
        {
            System.out.println("Valid age for
Voting");
        }
        else
        {

            throw new ArithmeticException("Not
Valid Age.!!");

        }

    }
```


Difference between final, finally and finalize

The final, finally, and finalize are keywords in Java that are used in exception handling. Each of these keywords has a different functionality. The basic difference between final, finally and finalize is that the **final** is an access modifier, **finally** is the block in Exception Handling and **finalize** is the method of object class.

Along with this, there are many differences between final, finally and finalize. A list of differences between final, finally and finalize are given below:

Sr. no.	Key	Final	finally	finalize
1.	Definition	final is the keyword and access modifier which is used to apply restrictions on a class, method or variable.	finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not.	finalize is the method in Java which is used to perform clean up processing just before object is garbage collected.
2.	Applicable to	Final keyword is used with the classes, methods and variables.	Finally block is always related to the try and catch block in exception handling.	finalize() method is used with the objects.
3.	Functionality	(1) Once declared, final variable becomes constant and cannot be modified. (2) final method cannot be overridden by sub class. (3) final class cannot be inherited.	(1) finally block runs the important code even if exception occurs or not. (2) finally block cleans up all the resources used in try block	finalize method performs the cleaning activities with respect to the object before its destruction.
4.	Execution	Final method is executed only when we call it.	Finally block is executed as soon as the try-catch block is executed. It's execution is not dependant on the exception.	finalize method is executed just before the object is destroyed.

File Handling:

In Java, 3 streams are created for us automatically. All these streams are attached with the console.

1) System.out: standard output stream

2) System.in: standard input stream

3) System.err: standard error stream

Java FileOutputStream Class

Java FileOutputStream is an output stream used for writing data to a file.

If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class.

Method	Description
protected void finalize()	It is used to clean up the connection with the file output stream.
void write(byte[] ary)	It is used to write **ary.length** bytes from the byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream.

void write(int b)	It is used to write the specified byte to the file output stream.
FileChannel getChannel()	It is used to return the file channel object associated with the file output stream.
FileDescriptor getFD()	It is used to return the file descriptor associated with the stream.
void close()	It is used to closes the file output stream.

Ex:

```
1.  import java.io.FileOutputStream;
2.  public class FileOutputStreamExample {
3.     public static void main(String args[]){
4.          try{
5.           FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
6.           String s="Welcome to javaTpoint.";
7.           byte b[]=s.getBytes();//converting string into byte array
8.           fout.write(b);
9.           fout.close();
10.          System.out.println("success...");
11.         }catch(Exception e){System.out.println(e);}
12.    }
13. }
```

Java FileInputStream Class

Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc.

Java FileInputStream class methods

Method	Description
int available()	It is used to return the estimated number of bytes that can be read from the input stream.
int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to **b.length** bytes of data from the input stream.
int read(byte[] b, int off, int len)	It is used to read up to **len** bytes of data from the input stream.
long skip(long x)	It is used to skip over and discards x bytes of data from the input stream.
FileChannel getChannel()	It is used to return the unique FileChannel object associated with the file input stream.
FileDescriptor getFD()	It is used to return the FileDescriptor object.
protected void finalize()	It is used to ensure that the close method is call when there is no more reference to the file input stream.
void close()	It is used to closes the stream.

Ex:

```
1.  import java.io.FileInputStream;
2.  public class DataStreamExample {
3.      public static void main(String args[]){
4.          try{
```

```
5.                 FileInputStream fin=new FileInputStream("D:\\testout.txt");
6.                 int i=0;
7.                 while((i=fin.read())!=-1){
8.                  System.out.print((char)i);
9.                 }
10.                fin.close();
11.               }catch(Exception e){System.out.println(e);}
12.              }
13.             }
```


Collection:

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

List

Set

Queue

And classes

For List we have two Classes that implements List interface

ArrayList

LinkedList

For Set we have three Classes that implements Set interface

HashSet

LinkedHashSet

TreeSet

For Queue we have One Classes that implements Queue interface

PriorityQueue

We have one more Map Interface that is not directly related to Collection.

Map Interface:

Three Classes implement this interface.

HashMap

LinkedHashMap

TreeMap

Basis	Array	ArrayList
Definition	An **array** is a dynamically-created object. It serves as a container that holds the constant number of values of the same type. It has a contiguous memory location.	The **ArrayList** is a class of Java **Collections** framework. It contains popular classes like **Vector, HashTable**, and **HashMap**.
Static/ Dynamic	Array is **static** in size.	ArrayList is **dynamic** in size.
Resizable	An array is a **fixed-length** data structure.	ArrayList is a **variable-length** data structure. It can be resized itself when needed.
Initialization	It is mandatory to provide the size of an array while initializing it directly or indirectly.	We can create an instance of ArrayList without specifying its size. Java creates ArrayList of default size.
Performance	It performs **fast** in comparison to ArrayList because of fixed size.	ArrayList is internally backed by the array in Java. The resize operation in ArrayList slows down the performance.
Primitive/ Generic type	An array can store both **objects** and **primitives** type.	We cannot store **primitive** type in ArrayList. It automatically converts primitive type to object.
Iterating Values	We use **for** loop or **for each** loop to iterate over an array.	We use an **iterator** to iterate over ArrayList.
Type-Safety	We cannot use generics along with array because it is not a convertible type of array.	ArrayList allows us to store only **generic/ type, that's why it is type-safe.**
Length	Array provides a **length** variable which	ArrayList provides the **size()** method

	denotes the length of an array.	to determine the size of ArrayList.
Adding Elements	We can add elements in an array by using the **assignment** operator.	Java provides the **add()** method to add elements in the ArrayList.
Single/ Multi-Dimensional	Array can be **multi-dimensional**.	ArrayList is always **single-dimensional**.

4) What is the difference between ArrayList and Vector?

No.	ArrayList	Vector
1)	ArrayList is not synchronized.	Vector is synchronized.
2)	ArrayList is not a legacy class.	Vector is a legacy class.
3)	ArrayList increases its size by 50% of the array size.	Vector increases its size by doubling the array size.
4)	ArrayList is not ?thread-safe? as it is not synchronized.	Vector list is ?thread-safe? as it?s every method is synchronized.

No.	ArrayList	LinkedList
1)	ArrayList uses a dynamic array.	LinkedList uses a doubly linked list.
2)	ArrayList is not efficient for manipulation because too much is required.	LinkedList is efficient for manipulation.
3)	ArrayList is better to store and fetch data.	LinkedList is better to manipulate data.
4)	ArrayList provides random access.	LinkedList does not provide random access.

5)	ArrayList takes less memory overhead as it stores only object	LinkedList takes more memory overhead, as it stores the object as well as the address of that object.

5) What is the difference between ArrayList and LinkedList?

6) What is the difference between Iterator and ListIterator?

Iterator traverses the elements in the forward direction only whereas ListIterator traverses the elements into forward and backward direction.

No.	Iterator	ListIterator
1)	The Iterator traverses the elements in the forward direction only.	ListIterator traverses the elements in backward and forward directions both.
2)	The Iterator can be used in List, Set, and Queue.	ListIterator can be used in List only.
3)	The Iterator can only perform remove operation while traversing the collection.	ListIterator can perform ?add,? ?remove,? and ?set? operation while traversing the collection.

7) What is the difference between Iterator and Enumeration?

No.	Iterator	Enumeration
1)	The Iterator can traverse legacy and non-legacy elements.	Enumeration can traverse only legacy elements.

2)	The Iterator is fail-fast.	Enumeration is not fail-fast.
3)	The Iterator is slower than Enumeration.	Enumeration is faster than Iterator.
4)	The Iterator can perform remove operation while traversing the collection.	The Enumeration can perform only traverse operation on the collection.

8) What is the difference between List and Set?

The List and Set both extend the collection interface. However, there are some differences between the both which are listed below.

- The List can contain duplicate elements whereas Set includes unique items.
- The List is an ordered collection which maintains the insertion order whereas Set is an unordered collection which does not preserve the insertion order.
- The List interface contains a single legacy class which is Vector class whereas Set interface does not have any legacy class.
- The List interface can allow n number of null values whereas Set interface only allows a single null value.

9) What is the difference between HashSet and TreeSet?

The HashSet and TreeSet, both classes, implement Set interface. The differences between the both are listed below.

- HashSet maintains no order whereas TreeSet maintains ascending order.
- HashSet impended by hash table whereas TreeSet implemented by a Tree structure.
- HashSet performs faster than TreeSet.
- HashSet is backed by HashMap whereas TreeSet is backed by TreeMap.

10) What is the difference between Set and Map?

The differences between the Set and Map are given below.

- Set contains values only whereas Map contains key and values both.
- Set contains unique values whereas Map can contain unique Keys with duplicate values.
- Set holds a single number of null value whereas Map can include a single null key with n number of null values.

---

11) What is the difference between HashSet and HashMap?

The differences between the HashSet and HashMap are listed below.

- HashSet contains only values whereas HashMap includes the entry (key, value). HashSet can be iterated, but HashMap needs to convert into Set to be iterated.
- HashSet implements Set interface whereas HashMap implements the Map interface
- HashSet cannot have any duplicate value whereas HashMap can contain duplicate values with unique keys.
- HashSet contains the only single number of null value whereas HashMap can hold a single null key with n number of null values.

---

12) What is the difference between HashMap and TreeMap?

The differences between the HashMap and TreeMap are given below.

- HashMap maintains no order, but TreeMap maintains ascending order.
- HashMap is implemented by hash table whereas TreeMap is implemented by a Tree structure.
- HashMap can be sorted by Key or value whereas TreeMap can be sorted by Key.
- HashMap may contain a null key with multiple null values whereas TreeMap cannot hold a null key but can have multiple null values.

---

13) What is the difference between HashMap and Hashtable?

No.	HashMap	Hashtable
1)	HashMap is not synchronized.	Hashtable is synchronized.
2)	HashMap can contain one null key and multiple null values.	Hashtable cannot contain any null key or null value.
3)	HashMap is not ?thread-safe,? so it is useful for non-threaded applications.	Hashtable is thread-safe, and it can be shared between various threads.
4)	4) HashMap inherits the AbstractMap class	Hashtable inherits the Dictionary class.

14) What is the difference between Collection and Collections?

The differences between the Collection and Collections are given below.

- The Collection is an interface whereas Collections is a class.
- The Collection interface provides the standard functionality of data structure to List, Set, and Queue. However, Collections class is to sort and synchronize the collection elements.
- The Collection interface provides the methods that can be used for data structure whereas Collections class provides the static methods which can be used for various operation on a collection.

15) What is the difference between Comparable and Comparator?

No.	Comparable	Comparator
1)	Comparable provides only one sort of sequence.	The Comparator provides multiple sorts of sequences.
2)	It provides one method named compareTo().	It provides one method named compare().
3)	It is found in java.lang package.	It is located in java.util package.
4)	If we implement the Comparable interface, The actual class is modified.	The actual class is not changed.