**********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.

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.

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.

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.

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.

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.

for loop

1) Simple for loop

Syntax:

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

}

2) for each 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);

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

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)

Block

this keyword in Java :

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.

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.

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 void run(){System.out.println("running...");}
}
class Honda2 extends Bike{
   public static void main(String args[]){
    new Honda2().run();
   }
}
```

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,**	Interface has **only static and final variables**.

static and non-static 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

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.  }}
```

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	StringBuffer class doesn't override the

	you can compare the contents of two strings by equals() method.	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