**Java Task 3**

1.What is the difference between primitive data type and reference data type in Java?

**a primitive variable's information is stored as the value of that variable, whereas a reference variable holds a reference to information related to that variable**.

1. Define the scope of variables (Hint:Local and Global variable)

**A global variable exists in the program for the entire time the program is executed. A local variable is created when the function is executed, and once the execution is finished, the variable is destroyed.**

1. why is initialization of variables required?

**To prevent bugs**

1. Differentiate between static,instance and local variable

Local variable instance variable Static variable

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Defined within a method or a code block | Defined outside a method at the class level | | | | | | | | Defined outside a method at the class level |
| Is only accessible in the method/code block where it is declared | | | | | Is accessible throughout the class | | | | Is accessible throughout the class |
| Remains in memory as long as the method executes | | Remains in memory as long as the object is in memory | | | | Remains in memory as long as program executes | | | |
| Does not require any special keyword | | | Does not require any special keyword but any access specifier (private, protected or public) can be specified. Typically, private or protected is used | | | | Requires the static keyword to be specified. In addition, any access specifier (private, protected or public) can be specified. Typically, public is used | | |
| Requires to be initialized before it is used | | | | Is given default value based on its data type, so does not require to be initialized before it is used | | | | Is given default value based on its data type, so does not require to be initialized before it is used. | |

5.Differentiate between widening and narrowing casting in java

**Widening takes place when a *smaller primitive type value* is automatically accommodated in a *larger/wider primitive data type but* Narrowing a *wider/bigger primitive type value* to a *smaller primitive type value*.**

6.the following table shows data type, its size, default value and the range. Filling in the missing values.

|  |  |  |  |
| --- | --- | --- | --- |
| **TYPE** | **SIZE (IN BYTES)** | **DEFAULT** | **RANGE** |
| boolean | 1 bit | **false** | true, false |
| Char | 2 | '\u0000' | ‘\0000’ to ‘\ffff’ |
| Byte | 1 | 0 | -27 to +27-1 |
| Short | 2 | 0 | -215 to +215-1 |
| Int | 4 | 0 | -231 to +231-1 |
| Long | 8 | 0L | - **-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807**. |
| Float | 4 | 00.0f | 1.40129846432481707e-45 to 3.40282346638528860e+38 (positive or negative). |
| Double | 8 | 0.0d | -1.8E+308 to +1.8E+308 |

**7.**Explain the importance of using Java packages

**1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.**

**2) Java package provides access protection.**

**3) Java package removes naming collision.**

**4) Packages provide reusability of code .**

**5)To bundle classes and interfaces.**

**6)We can crate our own Package or extend already available Package.**

8.Explain three controls used when creating GUI applications in Java language.

**Labels-**It is generally used to display some informative text on the GUI which is not changed during runtime.

**Text fields-** is a basic text control that enables the user to type a small amount of text.

**Buttons-**enables to get the expected interactive feedback from the system following a particular command.,

9.Explain the difference between containers and components as used in Java.

**The class Component is the abstract base class for the non-menu user-interface controls of AWT,a component represents an object with graphical representationwhereas the class Container is the superclass for the containers of AWT,the container object can contain other AWT components.**

10.Write a Java program to reverse an array having five items of type int.

**import** java.util.\*;

**import** java.util.stream.\*;

**public** **class** Main

{

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

    Integer[] intArray = {15,25,35,45,55,65,75,85,95};

  //print array starting from first element

    System.out.println("Original Array:");

**for**(**int** i=0;i<intArray.length;i++)

         System.out.print(intArray[i] + "  ");

    System.out.println();

    //print array starting from last element

    System.out.println("Original Array printed in reverse order:");

**for**(**int** i=intArray.length-1;i>=0;i--)

         System.out.print(intArray[i] + "  ");

    }

}

11.Programs written for a graphical user interface have to deal with “events.” Explain what is meant by the term event.

Give at least two different examples of events, and discuss how program mightrespond to those events.

**Event is an object that is created when something changes within a graphical user interface example if a user clicks on a button, clicks on a combo box, or types characters into a text field, etc., then an event triggers, creating the relevant event object.**

12.Explain the difference between the following terms as used in Java programming.

**-Polymorphism allows program code to have different meaning or functions while encapsulation is the process of keeping classes private so they cannot be modified by external codes**.

**-Overriding is when the method signature (name and parameters) are the same in the superclass and the child class while overloading is when two or more methods in the same class have the same name but different parameters**

**-Class describes the behaviour of the object and interface contains those behaviours which were assigned and defined by the class.**

**-Inheritance is one in which a new class is created that inherits the features from the already existing class Whereas polymorphism is that which can be defined in multiple forms.**

**13.**Using examples, explain the two possible ways of implementing polymorphism. Show your code in java.

****Method overloading** is the process that can create multiple methods of the same name in the same class, and all the methods work in different ways examples**

class Shapes{ public void area()

{ System.out.println("Find area "); }

public void area(int r) {

System.out.println("Circle area = "+3.14\*r\*r); }

public void area(double b, double h) { System.out.println("Triangle area="+0.5\*b\*h); }

public void area(int l, int b) { System.out.println("Rectangle area="+l\*b); } }

class Main { public static void main(String[] args)

{ Shapes myShape = new Shapes(); // Create a Shapes object myShape.area(); myShape.area(5); myShape.area(6.0,1.2); myShape.area(6,2);

} }

**Method overriding is the process when the subclass or a child class has the same method as declared in the parent class.**

**Example**

class Vehicle{ //defining a method void run(){

System.out.println("Vehicle is moving");} } //Creating a child class class Car2 extends Vehicle{ //defining the same method as in the parent class void run(){

System.out.println("car is running safely");}

public static void main(String args[]){

Car2 obj = new Car2();//creating object obj.run();//calling method

} }

14.With relevant examples, explain the following concepts as used in Java programming.

a. Mutable classes.

Explain what is meant by mutable class

**A mutable class is** **one that can change its internal state after it is created**

Write a program that implements the concept of mutable class

import java.util.Date;

public class TestMain {

public static void main(String args[]){

ImmutableClass im =

ImmutableClass.createNewInstance(100, "test", new Date());

System.out.println(im);

tryModification(im.getImmutableField1(), im.getImmutableField2(), im.getMutableField());

System.out.println(im);

System.out.println(test);

}

private static void tryModification(Integer fld1, String fld2,

Date mutableField){

fld1 = 304;

fld2 = "Test Changed";

mutableField.setDate(10);

}

}

B.Immutable classes.

Explain what is meant by immutable class

**define objects which, once created, never change their value**.

Write a program that implements the concept of immutable class

// Java Program to Create An Immutable Class

// Importing required classes

**import** java.util.HashMap;

**import** java.util.Map;

// Class 1

// An immutable class

**final** **class** Student {

    // Member attributes of final class

**private** **final** String name;

**private** **final** **int** regNo;

**private** **final** Map<String, String> metadata;

    // Constructor of immutable class

    // Parameterized constructor

**public** Student(String name, **int** regNo,

                   Map<String, String> metadata)

    {

        // This keyword refers to current instance itself

**this**.name = name;

**this**.regNo = regNo;

        // Creating Map object with reference to HashMap

        // Declaring object of string type

        Map<String, String> tempMap = **new** HashMap<>();

        // Iterating using for-each loop

**for** (Map.Entry<String, String> entry :

             metadata.entrySet()) {

            tempMap.put(entry.getKey(), entry.getValue());

        }

**this**.metadata = tempMap;

    }

    // Method 1

**public** String getName() { **return** name; }

    // Method 2

**public** **int** getRegNo() { **return** regNo; }

    // Note that there should not be any setters

    // Method 3

    // User -defined type

    // To get meta data

**public** Map<String, String> getMetadata()

    {

        // Creating Map with HashMap reference

        Map<String, String> tempMap = **new** HashMap<>();

**for** (Map.Entry<String, String> entry :

**this**.metadata.entrySet()) {

            tempMap.put(entry.getKey(), entry.getValue());

        }

**return** tempMap;

    }

}

// Class 2

// Main class

**class** GFG {

    // Main driver method

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

    {

        // Creating Map object with reference to HashMap

        Map<String, String> map = **new** HashMap<>();

        // Adding elements to Map object

        // using put() method

        map.put("1", "first");

        map.put("2", "second");

        Student s = **new** Student("ABC", 101, map);

        // Calling the above methods 1,2,3 of class1

        // inside main() method in class2 and

        // executing the print statement over them

        System.out.println(s.getName());

        System.out.println(s.getRegNo());

        System.out.println(s.getMetadata());

        // Uncommenting below line causes error

        // s.regNo = 102;

        map.put("3", "third");

        // Remains unchanged due to deep copy in constructor

        System.out.println(s.getMetadata());

        s.getMetadata().put("4", "fourth");

        // Remains unchanged due to deep copy in getter

        System.out.println(s.getMetadata());

    }

}

c. Explain the situations where mutable classes are more preferable than immutable classes when writing a Java program.

Immutable classes may waste memory and garbage collector time as changing of data will generate wasted instances.

1. Explain what a String buffer class is as used in Java

**is a peer class of String that provides much of the functionality of strings.**

the syntax of creating an object of StringBuffer class

**StringBuffer sb = new StringBuffer();**

Explain the methods in the StringBuffer class

**-StringBuffer Class append() Method-**The append() method concatenates the given argument with this String**.**

**-StringBuffer insert() Method-**The insert() method inserts the given String with this string at the given position.

**-StringBuffer replace() Method-**The replace() method replaces the given String from the specified beginIndex and endIndex.

**-StringBuffer delete() Method-**The delete() method of the StringBuffer class deletes the String from the specified beginIndex to endIndex.

b. Write the output of the following program.

class Myoutput

1. {
2. public static void main(String args[])
3. {
4. String ast = "hello i love java";
5. System.out.println(ast.indexOf('e')+" "+ast.indexOf('ast')+" "+ast.lastIndexOf('l')+" "+ast .lastIndexOf('v'));
6. }
7. }

ANSWER :6 4 6 9

c. Explain your answer in (2b) abo

SOLUTION :

indexof('c') and lastIndexof('c') are pre defined function which are used to get the indexof first and last occurrence ofthe character pointed by c in the given array

.Output:$ javac

Output.java$ java

Output 6 4 6 9

With explanation, write the output of the following program.

class Myoutput

1. {
2. public static void main(String args[])
3. {
4. StringBuffer bfobj = new StringBuffer("Jambo");
5. StringBuffer bfobj1 = new StringBuffer(" Kenya");
6. c.append(bfobj1);
7. System.out.println(bfobj);
8. }
9. }

e. With explanation, write the output of the following program.

class Myoutput

1. {
2. public static void main(String args[])
3. {
4. StringBuffer str1 = new StringBuffer("Jambo");
5. StringBuffer str2 = str1.reverse();
6. System.out.println(str2);
7. }

Output: obmaJ

f. With explanation, write the output of the following program.

**class Myoutput**

1. {
2. class output
3. {
4. public static void main(String args[])
5. {
6. char c[]={'A', '1', 'b' ,' ' ,'a' , '0'};
7. for (int i = 0; i < 5; ++i)
8. {
9. i++;
10. if(Character.isDigit(c[i]))
11. System.out.println(c[i]+" is a digit");
12. if(Character.isWhitespace(c[i]))

OUTPUT :

1 is a digit

a is a lower case Letter