**Task Three**

1. **Explain the differences between primitive and reference data types.**

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

1. **Define the scope of a variable (hint: local and global variable)**

Scope of a variable is the part of the program where the variable is accessible

1. **Why is initialization of variables required.**

To enable the program recognize your values and run according to the values.

1. **Differentiate between static, instance and local variables.**

Local variable is only accessible in the method/code block where it is declared where as instance variables are accessible throughout the class while static variable is accessible throughout the class

1. **Differentiate between widening and narrowing casting in java.**

Widening Casting is the conversion of a smaller data type to the larger type size while Narrowing Casting involves converting a larger data type to a smaller size type.

1. **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 | 1 | 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 | - |
| Float | 4 | 00.0f | −3.40282347×1038 to  3.40282347×1038 |
| Double | 8 | 0.0d | -1.8E+308 to +1.8E+308 |

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

It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, and annotations are related.

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

**AWT API** which wasas introduced in JDK 1.0. Most of the AWT UI components have become outdated and should be switched by newer Swing UI components.

**Swing API**, a much more inclusive set of graphics libraries that enhances the AWT, was introduced as part of Java Foundation Classes (JFC) after the release of JDK 1.1. JFC consisting of; Swing, Java2D, Accessibility, Internationalization.

**JavaFX,** which was integrated into JDK 8, was meant to replace Swing. JavaFX was moved out from the JDK in JDK 11, but still available as a separate module.

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

A container is a window-like component that can contain other components while a component is an object, like a button or a scroll bar.

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

public class ReverseArray {

public static void main(String[] args) {

//Initialize array

int [] arr = new int [] {1, 2, 3, 4, 5};

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

for (int i = 0; i < arr.length; i++) {

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

}

System.out.println();

System.out.println("Reverse array: ");

//Loop through the array in reverse order

for (int i = arr.length-1; i >= 0; i--) {

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

}

}

}

1. **Programs written for a graphical user interface have to deal with “events.”**

**Explain what is meant by the term event.**

The change in the state of an object in java.

**Give at least two different examples of events, and discuss how a program might**

**respond to those events.**

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

**Polymorphism and encapsulation**

Encapsulation is Packaging data and methods together while polymorphism is existing in many forms. Variables, functions, and objects can exist in multiple forms in Java.

**method overloading and method overriding**

Method overriding occurs when the method signature is the same in the superclass and the child class but method overloading occurs when two or more methods in the same class have the same name but different parameters.

**class and interface**

A class can contain constructors. An Interface cannot contain constructors.

**inheritance and polymorphism**

Inheritance is one in which a new class is created that inherits the properties of the already exist class while polymorphism is that in which we can perform a task in multiple forms or ways.

1. **sing examples, explain the two possible ways of implementing polymorphism. Show your**

**code in java.**

**OVERLOADING METHOD**

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);

}

}

Is the process that can create multiple methods of the same name in the same class, and all the methods work in different ways. Method overloading occurs when there is more than one method of the same name in the class.

**METHOD OVERRIDING**

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

}

}

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

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

**a. Mutable classes.**

**Explain what is meant by mutable class**

It is an object which can be modified after it is created.

**Write a program that implements the concept of mutable class**

**b. Immutable classes.**

public class JtpExample {

private String s;

JtpExample(String s) {

this.s = s;

}

public String getName() {

return s;

}

public void setName(String coursename) {

this.s = coursename;

}

public static void main(String[] args) {

JtpExample obj = new JtpExample("JavaTpoint");

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

// You can update the name by adding a few characters using the setName method.

obj.setName("Java Training");

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

}

}

**Explain what is meant by immutable class**

It is an object which can be modified after creation.

**Write a program that implements the concept of immutable class**

class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

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

Mutable classes are used where ther is need for modification while immutable classes are used in instances where objects shouldn’t be changed.

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

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

**the syntax of creating an object of StringBuffer class**

StringBuffer ob = new StringBuffer(String);

**Explain the methods in the StringBuffer class**

**append() method**

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

**insert() method**

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

**replace() method**

The replace() method replaces the given string from the specified beginIndex and endIndex-1.

**delete() method**

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex-1

**reverse() method**

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

**capacity() method**

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity\*2)+2.

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. }

OUTPUT;

**c. Explain your answer in (2b) above.**

**d. 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. }

**OUTPUT**; Jambo Kenya

**Explanation**; StringBuffer class is used to create mutable String objects. The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed therefore output becomes Jambo Kenya.

**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. }
8. }

**OUTPUT**; ombaJ

**Explanation**; StringBuffer class is used to create mutable String objects. The StringBuffer class in Java is the same as String class except it is mutable therefore output becomes **ombaJ** In this case bcause the string is reversed with the code; **System.out.println(str2);**

**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]))
13. System.out.println(c[i]+" is a Whitespace character");
14. if(Character.isUpperCase(c[i]))
15. System.out.println(c[i]+" is an Upper case Letter");
16. if(Character.isLowerCase(c[i]))
17. System.out.println(c[i]+" is a lower case Letter");
18. i++;
19. }
20. }
21. }