**NAME: JOB GABRIEL  
REG. NO: 21/04321  
TASK 3 ASSIGNMENT**

1. **Differences between primitive and reference data types  
   Primitive** data types’ information is stored as the value of a given variable, whereas a **reference** data type holds reference information related to that variable. Practically, reference data types are the objects in Java.
2. **Definition of the scope of a variable (local and global variable)  
   Global scope variables** are declared outside any function. They can be accessed and modified from any function.  
   **Local scope variables** are declared inside a function with the **var** keyword. They cannot be accessed or modified outside the function declaration.
3. **Why initialization of variables is required**When declaring a variable, it points to a piece of memory. Accessing the value of the variable will give you the contents of that piece of memory. Until the variable is initialized that piece of memory could contain anything.
4. **Differentiation between static, instance, and local variables  
   Static variables** are essentially global variables. Static variables are simply a single copy of the variable created and shared amongst all objects of class level.  
   **An instance variable** is a variable that is specific to a certain object. It is declared within the curly braces of the class but outside any method.  
   **A local variable** is a variable that is declared within the body of a method.
5. **Differentiation between widening and narrowing casting in java  
   Widening casting** is the process of a lower data type to a higher data type. It’s done automatically while **narrowing casting** is the conversion of a higher data type to a lower data type.
6. **Filling in the missing values**

|  |  |  |  |
| --- | --- | --- | --- |
| **TYPE** | **SIZE (IN BYTES)** | **DEFAULT** | **RANGE** |
| boolean | 1 bit | **false** | true, false |
| Char | 2 | **\0000** | ‘\0000’ to ‘\ffff’ |
| Byte | **8** | 0 | -27 to +27-1 |
| Short | **16** | 0 | -215 to +215-1 |
| Int | 4 | **0** | -231 to +231-1 |
| Long | **8** | 0L | **-263 to 263-1** |
| Float | 4 | 00.0f | **3.40282347 x 1038 to 1.40239846 x 10-45** |
| Double | 8 | **0.0d** | -1.8E+308 to +1.8E+308 |

1. **Importance of using java packages**Java packages are used to categorize classes and interfaces so that they can be easily maintained.  
   To provide access protection.  
   To remove naming collision.
2. **Three controls used while creating GUI in Java language**- JButton  
   - JCheckBox  
   - JRadioButton
3. **Difference between containers and components used in java  
   Containers** are the interface between a component and a low-level, platform-specific functionality that supports the component.  
   **Components** are fundamental user interface objects in java. It’s simply everything on display in a java application.
4. // It will be displayed in form of a code

package number10;

import java.util.Arrays;

public class Number10 {

public static void main(String[] args) {

int[] my\_array1 = {

1, 2, 3, 4, 5};

System.out.println("Original array : "+Arrays.toString(my\_array1));

for(int i = 0; i < my\_array1.length / 2; i++)

{

int temp = my\_array1[i];

my\_array1[i] = my\_array1[my\_array1.length - i - 1];

my\_array1[my\_array1.length - i - 1] = temp;

}

System.out.println("Reverse array : "+Arrays.toString(my\_array1));

}

}

1. **Explanation of events written for graphical user interface  
   An event** simply means an object that represents a user’s interaction with a GUI component.   
   Examples are: Pressing a button, Inputting characters in a textbox, clicking or dragging a mouse.  
   **The following is how a program might respond to an event**The program loads and then waits for user input events.  
   As each event occurs, the program runs a particular code to respond.  
   The overall flow of what code is executed is determined by the series of events that occur.  
   Contrast with application- or algorithm-driven control where the program expects input data in a pre-determined order and timing – Typical of large non-GUI applications like web crawling, payroll, and batch simulation.
2. **Explanation of terms in Java  
   Polymorphism and Encapsulation  
   Polymorphism** is the ability of a class to provide different implementations of a method, depending on the object that is passed to the method. **Encapsulation** is the process by which data (variables) and the code that acts upon them (methods) are integratedas a single unit.  
   **Method overloading and method overriding**Method overloading is when multiple methods of the same name and class are created, and all methods work in different ways. Method overriding occurs when a subclass has the same method as the parent class.  
   **Class and interface**  
   A **class** is a template used to create objects and to define object data types and methods. An **interface** is a reference type in java. It is a collection of abstract methods which is similar to a class. A class implements an interface, thereby inheriting the abstract methods of the interface.  
   **Inheritance and polymorphism  
   Inheritance** is a concept that acquires the properties from one class to another class while **Polymorphism** is the ability of a class to provide different implementations of a method, depending on the object that is passed to the method.
3. //Example Will be coded

package marks;

import java.util.Scanner;

public class grade {

static int java;

static int networking;

static int maths;

static int sum;

static int average;

public static void grade() {

Scanner input= new Scanner (System.in);

System.out.println("Enter the marks for java programming:");

java = input.nextInt();

System.out.println("Enter the marks for networking:");

networking = input.nextInt();

System.out.println("Enter the marks for maths:");

maths = input.nextInt();

sum = java + networking + maths ;

average = sum/3;

System.out.println(" The average is:"+ average);

}

public static void main(String[] args) {

grade();

}

}

**PART TWO: TASK 3**

1. **Explanation of the following concepts used in Java**a**. Mutable classes**  
   These are classes that can change their state after the class is created.  
     
   // Example will be coded

package myoutput;

public class Myoutput {

private String name;

Myoutput(String name) {

this.name = name;

}

public String getName() {

return name;

}

// this setter can modify the name

public void setName(String name) {

this.name = name;

}

public static void main(String[] args) {

Myoutput obj = new Myoutput("Job");

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

// update the name, this object is mutable

obj.setName("Gabriel");

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

}

}

1. **Immutable classes**  
   It simply means once an object is created, we cannot change its content.  
     
   // Example will be coded

package myoutput;

public final class Myoutput {

private String name;

Myoutput (String name) {

this.name = name;

}

public String getName() {

return name;

}

//no setter

public static void main(String[] args) {

Myoutput obj = new Myoutput("Job");

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

// there is no way to update the name after the object is created.

// obj.setName("Gabriel");

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

}

}

1. **Situations where mutable classes are preferable to immutable classes**

* In mutability, you can change the fields after object initialization, whereas immutability does not allow you to do so.
* A mutable object refers to the variable, whereas you can think of an immutable object as the value.
* Mutable classes generally have a method to modify the object’s state. Immutable classes do not have such a method.
* Mutable have getter and setter accessors, whereas immutable are read-only and have only getter accessors.
* The process of creating a mutable class is simple and comprises two steps:  
  – Creating a method that modifies the object.  
  – Defining the Getter and Setter methods.
* The process of creating an immutable class is a bit complex and comprises various steps:  
  – Declaring the class as sealed so no other class could inherit it.  
  – Make all the fields private so no other class can access them directly.  
  – Defining the Getter method while skipping the Setter.  
  – Make all the mutable fields read-only so they can be assigned only once.
* To change the state of an immutable object, you need to redeclare and redefine the object that adds an overhead over the compiler and causes inefficient use of memory. You can change the mutable objects without recreating them, which is a more efficient way to utilize the memory.
* Immutability becomes costly if the objects are large.

1. **String Buffer class**This is a class used to create mutable String objects.

**Syntax of creating StringBuffer class**class SBuffer {  
public static void main(String[] args)  
{  
StringBuffer s = new StringBuffer("IamStringB");  
int a = s.length();  
int b = s.capacity();  
System.out.println("Length of string IamStringB =" +a);  
System.out.println("Capacity of string IamStringB =" +b);  
}  
}

**Methods in String Buffer class**

* **Append ()** – Used to add text at the end of the existing text.
* **Length ()** – Used to find the length of a StringBuffer.
* **Capacity ()** – Used to find the total allocated capacity.
* **charAt ()** – Returns the char value in this sequence at the specified index.
* **Delete ()** – Deletes a sequence of characters from the invoking object.
* **deleteCharAt ()** – Deletes the character at the index specified by loc.
* **ensureCapacity ()** – Ensures capacity is equal to the given minimum.
* **Insert ()** – Inserts text at the specified index position.
* **Reverse ()** – Reverse the character with StringBuffer object.
* **Replace ()** – Replace one set of characters with another set inside a StringBuffer object.

1. It will not produce an output because there’s an error.
2. Because the program does not recognize the input "+ast.indexOf('ast')+" in the words “ hello I love java”.
3. There will be no output because the variable “c” cannot be located in the program.
4. The output will be obmaJ. Because the word “Jambo” has been reversed.
5. The output will be:

1 is a digit

a is a lowercase letter

This is because the program recognizes 1 as an integer and it recognizes a as a lowercase letter.

// This assignment was done by 21/04321 – Job Gabriel.