JAVA PROGRAMMING

FACULTY: SCHOOL OF TECHNOLOGY AND INFORMATION MANAGEMENT

COURSE: DIPLOMA IN INFORMATION TECHNOLOGY

UNIT CODE: DIT 406

UNIT NAME: JAVA PROGRAMMING

ASSIGNMENT THREE:

NAME: Tom Ngari

REG NO: 21/05079

SUBMITTED TO: Mr. Joseph Kimotho.

DATE: 20th November 2022.

PART I

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

- **Primitive data** types are already defined in Java while **reference data** types can be defined by the user.
- **Primitive data** types specify the size and type of variable values while **reference data** types specify the reference/address of the variable values.

2. Define the scope of a variable (hint: local and global variable)
A **scope** is a region of the program and broadly speaking there are three places where variables can be declared: Outside of all functions which are called global variables. Inside a function or a block which is called local variables, In the definition of function parameters which is called formal parameters.

3. Why is initialization of variables required?
- To avoid run-time errors.
- So that they can be used in a program.

4. Differentiate between static, instance and local variables.

Local variables remain in memory as long as the method executes .
Instance variables remain in memory as long as the object is in memory.
Static variables remain in memory as long as the program executes.

Local variables can be defined within a method or a code block.
Instance variables can be defined outside a method at the class level.
Static variables can be defined outside a method at the class level.

5. Differentiate between widening and narrowing casting in java.

Widening conversions preserve the source value but can change its representation while **narrowing conversion** changes a value to a data type that might not be able to hold some of the possible values.

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	3.4E-38 to 3.4E+38
Double	8	0.0d	-1.8E+308 to +1.8E+308

7. Explain the importance of using Java packages
   - To group related classes.
   - To avoid name conflicts.

8. Explain three controls used when creating GUI applications in Java language.
   - *Label* - Is used to provide a descriptive text string that cannot be changed directly by the user.
   - *Text Field* - Used to get text input from the user into the program for processing.
   - *Button* - Used to execute blocks of code in a program when clicked by the user.
   - *Checkbox* - Used to display options to the user, where the user can select more than one option.

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

Containers can have other containers and components in it while components cannot have other components in them.

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

```
import java.util.*;
import java.util.stream.*;
public class PrintArrays
{
   public static void main(String[] args) {
   //creating the array with 5 items
   Integer[] myArray = {1, 2, 3, 4, 5};

   //print the array starting from last element
      for(int i=myArray.length-1; i>=0; i--) {
          System.out.print(myArray[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 a program might respond to those events.

- *Event* - Is the changing of the state of an object or behavior by performing actions. These actions can be a button click, cursor movement, keypress through keyboard or page scrolling, etc.

Example: - when the user clicks a button, the program can display a dialog box.
- When the user moves the cursor in a container, the cursor can change its shape.

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

- *Polymorphism and encapsulation*

Polymorphism ensures that the proper method will be executed based on the calling object's type. Encapsulation allows you to control access to your object's state, while making it easier to maintain or change your implementation at a later date.

- *Method overloading and method overriding*

Method overloading is used when we want multiple methods providing a similar implementation. However, method overriding is used when we want to add some additional functionality on top of base class implementation.

- *Class and interface*

An object of a class can be created while an object of an interface cannot be created.
Inheritance and polymorphism
Inheritance supports the concept of reusability and reduces code length in object-oriented programming while polymorphism allows the object to decide which form of the function to implement at compile-time as well as run-time.

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

PART II

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

I. *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.

```
public class Example {
   private String str;
   Example(String str) {
   this.str = str;
   }
   public String getName() {
   return str;
   }
   public void setName(String coursename) {
   this.str = coursename;
   }
   public static void main(String[] args) {
   Example obj = new Example("Diploma in IT");
   System.out.println(obj.getName());
   // Here, we can update the name using the setName method.
   obj.setName("Java Programming");
   System.out.println(obj.getName());
   }
}
```

II. Immutable classes.

Explain what is meant by immutable class.

An immutable class is one that cannot change its internal state after it is created.

Write a program that implements the concept of immutable class.

```
public class Example {
   private final String str;
   Example(final String str) {
      this.str = str;
   }

   public final String getName() {
      return str;
   }

      //main method
   public static void main(String[] args) {
   Example obj = new Example("Core Java Programming.");
      System.out.println(obj.getName());
   }
}
```

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

- Immutable classes are thread-safe so you will not have any synchronization issues.
- Immutable classes are good Map keys and Set elements, since these typically do not change once created.
- Immutable classes it easier to write, use and reason about the code (class invariant is established once and then unchanged)
- Immutable classes make it easier to parallelize your program as there are no conflicts among objects.

2. a. Explain what a **StringBuffer class** is as used in Java, the syntax of creating an object of **StringBuffer class** and Explain the methods in the **StringBuffer class**.

StringBuffer is a thread-safe, a sequence of characters that can change.
The syntax of creating a *StringBuffer* object is:
Methods in the *StringBuffer* class:

- length() - used to return the length of the string i.e. total number of characters.
- reverse() - used to return the string in reversed order.
- capacity() - used to return the current capacity.

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:

The program has no output.

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

In the above code we have ast.indexOf('ast'). indexOf() does not take a String argument hence resulting to an error.

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.  }
```

The program does not run because of an error in line 6. "c.append(bfobj1);". The variable "c" was not created.

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: **obmaJ**

This is because the original str1 having "Jambo" has been reversed by the reverse() function and transferred to the str2 variable that is later printed.

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.    }
```

Output:

1 is a digit
a is a lower case Letter

At the first loop, we check if the second value is a digit, a whitespace, an uppercase or lowercase. Since it is "1", then it is a digit, and we print to the console.
We then skip the third value, and check the forth value if it is a digit, a whitespace, an uppercase or lowercase. Since the forth value is "a", then it is a lowercase, and we print to the console.
"I" is incremented two times in the loop.