MICHELLE KABURA NDIBUI  
SCT221-0500/2023

OCHIENG K. OTIENO

SCT221-0500/2023

Java programming Assignment

Section 1:

1.Primitive vs. Reference Data Types

Primitive Data Type:

The primitive data types are those basic data types that store their values in the memory locations directly. For example int, float, double, char, boolean.

The primitive data types are pre-defined in the language and are of fixed size. They are better in memory point of view.

Operations on the primitives are generally faster since they are based on direct value access with no dereferencing overhead.

Primitive types cannot be null and are always holding some value.

Reference data types:

These types store the references of the actual data or the memory addresses—objects and arrays.

These can be made by the programmer.

Reference types can take a value of null, i.e. they do not point to any object.

They consume more memory bad the extra storage to keep the references as well as the actual objects, their Definitions is sometimes involving extra steps, that is, dereferencing, thereby slightly reducing performance.

2.Scope of Variable

Local Variables:

Declared within a method or block; it is only accessible within that scope. Created and destroyed with method execution.

Global Variables:

Declared outside methods; accessible throughout the program. Exist for the lifetime of the program.

3. Importance of Variable Initialization

* Prevents undefined behavior and errors.
* Ensures correctness by starting with known values.
* Enhances code readability and clarity.

4.Differentiating Static, Instance, and Local Variables

Static Variables:

Belong to the class, shared across all instances, initialized once.

Instance Variables:

Unique to each instance of a class, retain different values per object.

Local Variables:

Declared within a method, accessible only within that method, created and destroyed with method calls

5.Widening vs. Narrowing Casting in Java

Casting of widening:

Widening conversion from smaller type to the bigger type, for example, from int to float. This is a safe operation with no data loss.

Narrowing Casting:

If one is converting from a smaller type to a larger type, such as from float to int, an explicit cast can be used, but of course this can lose data.

5. fill in the missing values in the table for data types, their sizes, default values, and ranges:

| **TYPE** | **SIZE (IN BYTES)** | **DEFAULT** | **RANGE** |
| --- | --- | --- | --- |
| boolean | 1 bit | false | true, false |
| Char | 2 | '\u0000' | '\u0000' to '\uffff' |
| Byte | 1 | 0 | -128 to 127 |
| Short | 2 | 0 | -32,768 to 32,767 (-2¹⁵ to 2¹⁵-1) |
| Int | 4 | 0 | -2,147,483,648 to 2,147,483,647 |
| Long | 8 | 0L | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| Float | 4 | 0.0f | -3.4E+38 to +3.4E+38 |
| Double | 8 | 0.0d | -1.8E+308 to +1.8E+308 |

7.A class is a blueprint of an object. It contains and namespaces data, known as attributes, and methods — known as procedures, functions, or behaviors — that describe the nature and view of the respective objects formed from it.

8.Why are classes important in Java programming?

Modularity: Classes are those parts of a program which divide it into independent modules; it allows ease of understanding and maintenance of code.

Reusability: With the concept of inheritance, code reusability is possible; therefore, it avoids redundancy and saves development time.

Encapsulation: Classes encapsulate the intrinsic state of objects. The access to data is restricted and thus, it enhances the integrity of the data.

Abstraction: Classes support abstraction. Complex behavior can be defined with implementation details hidden.

Polymorphism: Classes support polymorphism. Methods can operate on objects from different classes, and thus, it provides flexibility.

Scalability: Adding new features to classes is relatively easier and has a minimal effect on existing code. This makes the software easier to adapt.