**ASSIGNMENT**

**RYAN MWENDA**

**22/08977**

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

* **Primitive data types:** These are basic data types provided by the programming language (e.g., int, float, char) and are used to store simple values directly. They are typically stored on the stack memory.
* **Reference data types:** These refer to objects in memory, and variables of reference types store references (addresses) to the actual data stored on the heap. Examples include objects of classes, arrays, etc. Reference types are more complex and require more memory than primitive types.

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

* **Local variable:** Declared inside a method, constructor, or block. Its scope is limited to the block in which it is declared.
* **Global variable (instance variable):** Declared within a class but outside any method, constructor, or block. It is accessible throughout the class and its value is specific to each instance of the class.
* **Global variable (class/static variable):** Declared as static within a class but outside any method, constructor, or block. It belongs to the class rather than any instance and is shared among all instances of the class.

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

* Variables in most programming languages must be initialized before they can be used to ensure predictable behavior and avoid errors. Uninitialized variables can contain garbage values, leading to unexpected results or crashes.
* Initialization sets a default value to a variable, ensuring that the variable has a known state when it is first used.

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

* **Static variables:** Belong to the class rather than instances of the class. They are initialized only once, at the start of the execution, and are shared among all instances of the class.
* **Instance variables:** Also known as non-static or member variables, these belong to individual instances of a class. Each instance has its own copy of instance variables.
* **Local variables:** Declared within a method, constructor, or block. Their scope is limited to the block in which they are declared and they are not accessible outside that block.

1. **Differentiate between widening and narrowing casting in java.**
   * **Widening (implicit) casting:** This happens when you convert a smaller data type to a larger data type. Java does this automatically because there is no loss of information. For example, converting an int to a float.
   * **Narrowing (explicit) casting:** This involves converting a larger data type to a smaller data type, which could potentially lose information. Explicit casting is required and must be done manually because it may lead to loss of precision.
2. **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 | False | True, False |
| Char | 2 | ‘\u0000’ | ‘\u0000’ to ‘\uffff’ |
| Byte | 1 | 0 | -128 to 127 |
| Short | 2 | 0 | -32,768 to 32,767 |
| Int | 4 | 0 | -2,147,483,648 to 2,147,483,647 |
| Long | 8 | 0L | -9,223,372,036,854,775,80,8 to 9,223,372,036,854,775,807 |
| Float | 4 | 00.0F | 3.4028235E+38 to 1.4E-45 |
| Double | 8 | 0.0 | 1.797631348623157E+308 to 4.9E-324 |

1. **Define class as used in OOP**

* In Object-Oriented Programming (OOP), a class is a blueprint or template for creating objects (instances) that define its properties (attributes) and behaviors (methods).
* **Properties (attributes):** These are the data members or fields that describe the state of an object.
* **Behaviors (methods):** These are the functions or operations that an object can perform.
* A class encapsulates data (attributes) and methods (behaviors) into a single unit, providing a way to model real-world entities or abstract concepts in code.

1. **Explain the importance of classes in Java programming**

* **Encapsulation:** Classes in Java enable encapsulation, which means bundling data (attributes) and methods (behaviors) together into a single unit. This helps in organizing code logically and improves code maintainability.
* **Code reusability:** Classes allow you to create reusable components. Once a class is defined, you can create multiple instances (objects) of that class throughout your program.
* **Inheritance:** Java supports inheritance, where one class (subclass or derived class) can inherit attributes and methods from another class (superclass or base class). This promotes code reuse and allows you to model hierarchical relationships.
* **Polymorphism:** Classes and inheritance support polymorphism, where objects of different classes can be treated as objects of a common superclass. This allows for flexibility and extensibility in your code.
* **Modularity:** Classes help in breaking down complex problems into smaller, manageable components. Each class can focus on a specific aspect of functionality, making the overall program structure more modular and easier to understand.
* **Object instantiation:** In Java, objects are instances of classes. Classes define the structure and behavior of objects, and creating instances of classes allows you to work with real data and perform operations based on the class's definition.
* **Standardization:** Java programming encourages the use of classes and objects as a standard way to model and represent entities in software development. This standardization fosters good programming practices and facilitates collaboration among developers.