**Java Programming Assignment**

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

Primitive data types are the most basic data types available in Java. They are predefined by the language and named by a reserved keyword e.g. byte, short, int, long, float, double, char, and Boolean, while, Reference data types are used to refer to objects. They are not predefined like primitive types and are created by the user.

e.g. All classes, arrays, and interfaces.

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

**Local Variable:** A variable declared within a method, constructor, or block.

**Scope**: It is only accessible within the method, constructor, or block where it is declared.

It is created when the method, constructor, or block is entered and destroyed upon exit.

**Global Variable:** A variable declared inside a class but outside any method.

**Scope**: Accessible within any non-static method of the class.

Exists as long as the instance of the class exists.

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

(a). Preventing Garbage Values: Ensures that the variable has a definite value before it is used, preventing unpredictable behavior.

(b). Avoiding Compilation Errors: In many programming languages like Java, using an

uninitialized variable leads to compilation errors.

©. Improving Code Readability and Maintainability: Makes the code clearer and easier to understand by indicating the initial state of the variable.

1. Differentiate between static, instance and local variables.

**Static Variables**: Shared among all instances of a class, belong to the class, exist for the duration of the program, accessed using the class name.

**Instance Variables**: Belong to each instance of a class, separate copy for each instance, exist for the lifetime of the instance, accessed through object instances.

**Local Variables**: Declared within a method, constructor, or block, limited scope to where they are declared, exist only for the duration of the method or block execution, accessed directly within their scope.

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

**(i). Widening Casting (Automatic Type Conversion)**: Converting a smaller data type to a larger data type. This type of casting is done automatically by the Java compiler and is also known as implicit casting. E.g. byte to short, short to int

**(ii). Narrowing Casting:** Converting a larger data type to a smaller data type. This type of casting must be explicitly performed by the programmer using a cast operator. E.g. double to float

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 | **false** | true, false |
| Char | 2 | \U0000 | ‘\0000’ to ‘\ffff’ |
| Byte | **8 bits (1 byte)** | 0 | -**128 to 127.** |
| Short | 2 bytes | 0 | -215 to +215-1 |
| Int | 4 | 0 | -2^31 to 2^31-1 |
| Long | 8 bytes | 0L | -2^63 to 2^63-1 |
| Float | 4 | 00.0f | 1.4E-45 to 3.4028235E+38 |
| Double | 8 | 0.0d | -1.8E+308 to +1.8E+308 |

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

* a **class** is a blueprint or template for creating objects (instances) of a particular type. It serves as a foundational concept that encapsulates data (attributes) and behaviors (methods) that objects of the class can exhibit.

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

(I). A Blueprint for objects- This allows for the creation of multiple objects that share the same structure and behavior, promoting code reusability and consistency.

(ii). Enhances code maintainability and reduces complexity.

(iii). Classes allow programmers to create abstract representations of real-world entities or concepts.

(iv). Classes enable hierarchical relationships among objects through inheritance. This promotes code reuse, as subclasses can extend and specialize behaviors defined in their superclass.

(v). Through classes, Java supports polymorphism, enabling flexibility and extensibility in object-oriented design. Polymorphism allows methods to be invoked dynamically at runtime based on the actual type of object, promoting code flexibility and scalability.