Section 1

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

Primitive data types are the most basic data types available in Java. They include boolean, char, byte, short, int, long, float, and double. These data types hold their values directly in memory. Reference data types, on the other hand, store references (addresses) to the actual data stored in memory, rather than the data itself. Examples of reference data types are objects, arrays, and strings. The key difference is that primitive types store the actual value, while reference types store the memory address where the value is located.

# 2. Define the scope of a variable (hint: local and global variable)

The scope of a variable determines where in the program a variable can be accessed or modified. A local variable is declared within a method or block and can only be accessed within that method or block. It is not accessible outside of its scope. A global variable, often referred to as a class or instance variable in object-oriented programming, is declared outside of all methods but within a class, and can be accessed by any method in the class.

# 3. Why is initialization of variables required?

Initialization of variables is required to assign an initial value to the variable before it is used in operations. If a variable is not initialized, it might contain garbage data, leading to unpredictable behavior or errors in the program. In some programming languages, like Java, local variables must be initialized before use, whereas class variables are automatically initialized to default values.

# 4. Differentiate between static, instance and local variables.

Static variables are class-level variables that are shared among all instances of a class. They are declared using the static keyword and retain their values between method calls. Instance variables are non-static variables that are unique to each instance of a class. They represent the attributes or state of an object. Local variables are declared within a method or block and are only accessible within that method or block. They are not visible to other parts of the class.

# 5. Differentiate between widening and narrowing casting in Java.

Widening casting in Java is the process of converting a smaller data type to a larger data type. For example, converting an int to a long. This is done automatically by the compiler as there is no risk of data loss. Narrowing casting, on the other hand, is converting a larger data type to a smaller data type, like converting a double to an int. This requires explicit casting by the programmer because there is a risk of losing precision or data.

# 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' | '\u0000' to '\uffff' |
| Byte | 1 | 0 | -128 to 127 |
| Short | 2 | 0 | -32,768 to 32,767 |
| Int | 4 | 0 | -2^31 to 2^31-1 |
| Long | 8 | 0L | -2^63 to 2^63-1 |
| Float | 4 | 0.0f | -3.4E+38 to 3.4E+38 |
| Double | 8 | 0.0d | -1.8E+308 to 1.8E+308 |

# 7. Define class as used in OOP

In Object-Oriented Programming (OOP), a class is a blueprint or template for creating objects. It defines a set of properties (attributes) and behaviors (methods) that the objects created from the class will have. A class encapsulates data for the object and methods to manipulate that data, promoting code reusability and modular design.

# 8. Explain the importance of classes in Java programming.

Classes in Java programming are fundamental to the object-oriented paradigm. They allow for the creation of objects, which are instances of classes. Classes promote code reuse through inheritance and polymorphism, enable encapsulation of data, and provide a structure for organizing and managing complex software projects. By using classes, developers can create modular, scalable, and maintainable code.