## SCT221-0843/2022

## ROCK ANNAN SCHOLICKS

# OOP 2 ASSIGNMENT

1. Differences Between Primitive and Reference Data Types

- Primitive Data Types :

- These are the most basic data types in a programming language.

- They hold simple values such as integers, characters, and booleans.

- Examples: `int`, `char`, `boolean`, `float`, `double` in Java.

- Primitive data types are stored directly in the memory stack.

- They are fixed in size, and their values are stored directly in the variable itself.

- Reference Data Types :

- These refer to objects and arrays. A reference type holds a reference (or address) to the actual data.

- They do not store the actual value but a pointer to the memory location where the object or array is stored.

- Examples: `String`, `Array`, `Classes`, `Interfaces` in Java.

- Reference data types are stored in the heap memory, and the reference (or pointer) is stored in the memory stack.

2. Scope of a Variable

- Local Variable :

- A local variable is declared within a method, constructor, or block.

- It is only accessible within that specific method, constructor, or block.

- Once the method execution is complete, the local variable is destroyed.

- Global Variable (or Instance Variable in Object-Oriented languages):

- A global variable is typically declared at the class level, outside of any methods or blocks.

- It can be accessed by all methods within the class.

- Global variables remain in memory for the entire duration of the program and are accessible from any part of the code where they are visible.

3. Why Initialization of Variables is Required

- Initialization ensures that a variable has a defined value before it is used. Uninitialized variables can lead to undefined behavior, as they might contain garbage data.

- In some programming languages, like Java, primitive variables have default values, but it's still good practice to explicitly initialize them to avoid confusion.

- Initializing variables also makes the code more readable and maintainable, as it clearly shows the programmer's intent.

4. Differentiation Between Static, Instance, and Local Variables

- Static Variables :

- Declared with the `static` keyword.

- Belong to the class rather than any specific instance.

- Shared among all instances of a class, meaning if one instance modifies the static variable, all other instances will see the change.

- Memory allocation happens only once when the class is loaded.

- Instance Variables :

- Declared inside a class but outside any method, block, or constructor.

- Each instance of the class has its own copy of the instance variables.

- They represent the attributes of an object and are initialized when an object is created.

- Instance variables are destroyed when the object is destroyed.

- Local Variables :

- Declared within a method, constructor, or block.

- They only exist and are accessible during the execution of the block in which they are declared.

- They do not have default values, so they must be initialized before use.

5. Widening vs. Narrowing Casting in Java

- Widening Casting (Implicit Casting) :

- This is automatically done by the Java compiler when converting a smaller data type into a larger data type.

- No data loss occurs during widening.

- Example: Converting an `int` to a `long` or a `float` to a `double`.

- Syntax: `long l = intVariable;`

- Narrowing Casting (Explicit Casting) :

- This is done manually by the programmer when converting a larger data type into a smaller data type.

- Data loss can occur, as you might be truncating or rounding off the value.

- Example: Converting a `double` to an `int` or a `long` to a `short`.

- Syntax: `int i = (int) doubleVariable;`

### 6. Filling in the Missing Values in the Data Types Table

| **TYPE** | **SIZE (IN BYTES)** | **DEFAULT VALUE** | **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 | ±1.4E-45 to ±3.4E+38 |
| double | 8 | 0.0d | ±4.9E-324 to ±1.8E+308 |

### 7. Definition of Class in Object-Oriented Programming (OOP)

A **class** in Object-Oriented Programming (OOP) is a blueprint or template for creating objects. It encapsulates data for the object (attributes) and methods (functions) that operate on the data. The class defines the properties and behaviors that objects created from the class can have.

### 8. Importance of Classes in Java Programming

**1. Encapsulation:**

* Classes allow you to encapsulate data and the methods that operate on that data into a single unit. This promotes modular design and makes it easier to manage and maintain code.

**2. Reusability:**

* Once a class is defined, it can be reused to create multiple objects with similar properties and behaviors. This reduces code duplication and enhances maintainability.

**3. Abstraction:**

* Classes enable abstraction by hiding the internal implementation details and exposing only the necessary interfaces. This simplifies complex systems and makes them easier to work with.

**4. Inheritance:**

* Classes support inheritance, allowing new classes to be created based on existing ones. This facilitates code reuse and helps in building hierarchical class structures, promoting efficient code organization.

**5. Polymorphism:**

* Polymorphism allows methods to be overridden and objects to be treated as instances of their parent class. This flexibility allows for dynamic method binding, enabling more generic and adaptable code.

**6. Object-Oriented Design:**

* Classes are the fundamental building blocks of object-oriented programming, which is a paradigm that models real-world entities as software objects. This approach makes the design and implementation of complex systems more intuitive and scalable.