functions-classes-objects-visibility

Pranjal Aggarwal

Masters from NIT Allahabad
Upcoming Data Analyst at Citi

October 27, 2024

Agenda

- Understanding Functions
- Introduction to Classes
- Working with Objects
- Access Modifiers

Understanding Functions

Functions help us to modularize code by grouping code into reusable blocks. They make code organized, avoid repetition, and enhance readability.

- **Defining a Function**: Use the keyword `void` or return type, followed by the function name and parameters.
- **Calling a Function**: Simply call the function by its name, passing required arguments if necessary.

Example:

- `public int addNumbers(int a, int b) { return a + b; }`
- `int result = addNumbers(5, 10);`

Why Functions?

Functions allow:

- Reusability: Code once, use multiple times.
- Clarity: Each function can perform a specific task, making the program easier to understand.
- Modularity: Breaking code into manageable parts.

Imagine you have a function for calculating the total price of an order, rather than writing the price calculation each time it's needed.

Example:

- `public double calculateTotal(double price, double tax) { return price + tax; }`

Introduction to Classes

A class is a blueprint for creating objects, defining their properties (attributes) and behaviors (methods).

- **Attributes**: Variables that hold data specific to the class.
- **Methods**: Functions defined within a class that perform actions.
- Classes are templates that can be used to create multiple objects with shared characteristics.

Example:

- `class Car { String color; int speed; void drive() {...} }`

How Classes Work

Each class can contain:

- **Constructor**: A special method for creating and initializing objects.
- **Attributes**: Variables declared inside the class (e.g., `color` or `speed` in a `Car` class).
- **Methods**: Actions an object of the class can perform (e.g., `drive` or `brake`).

Creating an object from a class:

- `Car myCar = new Car();`

Working with Objects

An object is an instance of a class, representing real-world entities with attributes and behaviors.

- Objects allow for storing data in the form of attributes.
- They use methods to interact or perform actions.

Example:

- `Car myCar = new Car();`
- `myCar.color = "Red";`
- `myCar.drive();`

Object Behavior

Objects allow:

- **State**: Represented by attributes, like color or size.
- **Behavior**: Defined by methods, like move, drive, or accelerate.

For example, an object of class `Car` can:

- Change `color` attribute to represent different states.
- Call the `drive` method to perform its behavior.

Access Modifiers in Java

Access modifiers control visibility and accessibility of classes, methods, and variables within and outside their classes. The main access levels are:

- **Public**: Accessible from any other class.
- **Private**: Accessible only within the declared class.
- **Protected**: Accessible within the same package or by subclasses.
- **Default** (no modifier): Accessible only within the same package.

Properly using access modifiers helps secure and organize code.

Examples of Access Modifiers

Example Code:

- ```
  public class Car {
     public String model; // Visible everywhere
     private int year; // Only within Car class
     protected String color; // Visible to Car and
  subclasses
     int speed; // Default:  visible within package
  }
  ```

Usage:

- **Public**: To expose essential information to other classes.
- **Private**: To secure data or methods not meant to be accessed directly.
- **Protected**: Allows subclass access for inheritance needs.
- **Default**: Limits access to package-only visibility.
  ```