# Best Practices in Constructors

1. **Use this Keyword**:
   * Avoid ambiguity when parameter names are the same as attribute names.
   * Example: this.customerName = customerName;
2. **Keep Logic Simple**:
   * Avoid heavy computations or database calls inside constructors.
3. **Provide Multiple Constructors**:
   * Support various initialization scenarios by overloading constructors.
4. **Encapsulate Logic**:
   * Use private methods (like calculatePrice()) to keep constructors clean.

# 

# Best Practices in Access Modifiers

**Use the Least Privilege**:

* Start with the most restrictive modifier (private) and relax it as needed (protected or public).

**Encapsulation**:

* Always make attributes private and use getters/setters for controlled access.

**Protected Usage**:

* Use protected only when inheritance is required and controlled access is necessary.

**Avoid Overexposure**:

* Limit the use of public to methods or classes that are meant to be accessed by external code.

**Package Access**:

* Use the default (package-private) modifier to restrict access to the same package unless explicitly needed elsewhere.

**Avoid Leaks**:

* Be cautious with exposing mutable objects, like collections, via getters. Return a copy or an unmodifiable view when possible.

# Level 1 Practice Programs

1. Create a Book class with attributes title, author, and price. Provide both default and parameterized constructors.
2. Write a Circle class with a radius attribute. Use constructor chaining to initialize radius with default and user-provided values.
3. Create a Person class with a copy constructor that clones another person's attributes.
4. **Hotel Booking System**: Create a HotelBooking class with attributes guestName, roomType, and nights. Use default, parameterized, and copy constructors to initialize bookings.
5. **Library Book System**: Create a Book class with attributes title, author, price, and availability. Implement a method to borrow a book.
6. **Car Rental System**: Create a CarRental class with attributes customerName, carModel, and rentalDays. Add constructors to initialize the rental details and calculate total cost.

### **1. Instance vs. Class Variables and Methods**

#### **Problem 1: Product Inventory**

Create a Product class with:

* Instance Variables: productName, price.
* Class Variable: totalProducts (shared among all products).
* Methods:
  + An instance method displayProductDetails() to display the details of a product.
  + A class method displayTotalProducts() to show the total number of products created.

#### **Problem 2: Online Course Management**

Design a Course class with:

* Instance Variables: courseName, duration, fee.
* Class Variable: instituteName (common for all courses).
* Methods:
  + An instance method displayCourseDetails() to display the course details.
  + A class method updateInstituteName() to modify the institute name for all courses.

#### **Problem 3: Vehicle Registration**

Create a Vehicle class to manage the details of vehicles:

* Instance Variables: ownerName, vehicleType.
* Class Variable: registrationFee (fixed for all vehicles).
* Methods:
  + An instance method displayVehicleDetails() to display owner and vehicle details.
  + A class method updateRegistrationFee() to change the registration fee.

### **2. Access Modifiers**

#### **Problem 1: University Management System**

Create a Student class with:

* rollNumber (public).
* name (protected).
* CGPA (private).

Write methods to:

* Access and modify CGPA using public methods.
* Create a subclass PostgraduateStudent to demonstrate the use of protected members.

#### **Problem 2: Book Library System**

Design a Book class with:

* ISBN (public).
* title (protected).
* author (private).

Write methods to:

* Set and get the author name.
* Create a subclass EBook to access ISBN and title and demonstrate access modifiers.

#### **Problem 3: Bank Account Management**

Create a BankAccount class with:

* accountNumber (public).
* accountHolder (protected).
* balance (private).

Write methods to:

* Access and modify balance using public methods.
* Create a subclass SavingsAccount to demonstrate access to accountNumber and accountHolder.

#### **Problem 4: Employee Records**

Develop an Employee class with:

* employeeID (public).
* department (protected).
* salary (private).

Write methods to:

* Modify salary using a public method.
* Create a subclass Manager to access employeeID and department.