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.

```
class Book {
    String title, author;
    double price;


    public Book() {
        this.title = "Unknown";
        this.author = "Unknown";
        this.price = 0.0;
    }


    public Book(String title, String author, double price) {
        this.title = title;
        this.author = author;
        this.price = price;
    }

    public void displayDetails() {
        System.out.println("Title: " + title + ", Author: " + author + ", 
Price: " + price);
    }

    public static void main(String[] args) {
        Book book1 = new Book();
        Book book2 = new Book("Java Programming", "John Doe", 499.99);

        book1.displayDetails();
        book2.displayDetails();
    }
}
```

2. Write a `Circle` class with a `radius` attribute. Use constructor chaining to initialize `radius` with default and user-provided values.

```
class Circle {
    double radius;


    public Circle() {
        this(1.0); // Calls the parameterized constructor
    }


    public Circle(double radius) {
        this.radius = radius;
    }

    public double calculateArea() {
        return Math.PI * radius * radius;
    }

    public void display() {
        System.out.println("Radius: " + radius + ", Area: " +
calculateArea());
    }

    public static void main(String[] args) {
        Circle c1 = new Circle();
        Circle c2 = new Circle(5.5);

        c1.display();
        c2.display();
    }
}
```

3. Create a Person class with a copy constructor that clones another person's attributes.

```
class Person {
    String name;
    int age;


    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }


    public Person(Person other) {
        this.name = other.name;
        this.age = other.age;
    }

    public void display() {
        System.out.println("Name: " + name + ", Age: " + age);
    }

    public static void main(String[] args) {
        Person p1 = new Person("Alice", 25);
        Person p2 = new Person(p1); // Using copy constructor

        p1.display();
        p2.display();
    }
}
```

4. **Hotel Booking System**: Create a `HotelBooking` class with attributes `guestName`, `roomType`, and `nights`. Use default, parameterized, and copy constructors to initialize bookings.

```
class HotelBooking {
    String guestName, roomType;
    int nights;


    public HotelBooking() {
        this.guestName = "Guest";
        this.roomType = "Standard";
        this.nights = 1;
    }


    public HotelBooking(String guestName, String roomType, int nights) {
        this.guestName = guestName;
        this.roomType = roomType;
        this.nights = nights;
    }


    public HotelBooking(HotelBooking other) {
        this.guestName = other.guestName;
        this.roomType = other.roomType;
        this.nights = other.nights;
    }

    public void displayDetails() {
        System.out.println("Guest: " + guestName + ", Room Type: " +
roomType + ", Nights: " + nights);
    }

    public static void main(String[] args) {
        HotelBooking h1 = new HotelBooking();
        HotelBooking h2 = new HotelBooking("John Doe", "Deluxe", 3);
        HotelBooking h3 = new HotelBooking(h2);

        h1.displayDetails();
        h2.displayDetails();
        h3.displayDetails();
    }
}
```

5. **Library Book System**: Create a `Book` class with attributes `title`, `author`, `price`, and `availability`. Implement a method to borrow a book.

```
class LibraryBook {
    String title, author;
    double price;
    boolean available = true;

    public LibraryBook(String title, String author, double price) {
        this.title = title;
        this.author = author;
        this.price = price;
    }

    public void borrowBook() {
        if (available) {
            available = false;
            System.out.println(title + " has been borrowed.");
        } else {
            System.out.println(title + " is not available.");
        }
    }

    public void displayDetails() {
        System.out.println("Title: " + title + ", Author: " + author + ", 
Price: " + price + ", Available: " + available);
```

```
    }

    public static void main(String[] args) {
        LibraryBook book1 = new LibraryBook("Java Basics", "James Gosling",
300);
        book1.displayDetails();
        book1.borrowBook();
        book1.displayDetails();
    }
}
```

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

```
class CarRental {
    String customerName, carModel;
    int rentalDays;
    double pricePerDay = 500.0;

    public CarRental(String customerName, String carModel, int rentalDays)
{
        this.customerName = customerName;
        this.carModel = carModel;
        this.rentalDays = rentalDays;
    }

    public double calculateTotalCost() {
        return rentalDays * pricePerDay;
    }

    public void displayDetails() {
        System.out.println("Customer: " + customerName + ", Car: " +
carModel + ", Days: " + rentalDays + ", Total Cost: " +
calculateTotalCost());
```

```
    }

    public static void main(String[] args) {
        CarRental rental = new CarRental("Alice", "Toyota", 5);
        rental.displayDetails();
    }
}
```

.

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.

```
class Product {
    String productName;
    double price;
    static int totalProducts = 0;

    public Product(String productName, double price) {
        this.productName = productName;
        this.price = price;
        totalProducts++;
    }

    public void displayProductDetails() {
        System.out.println("Product: " + productName + ", Price: " +
price);
    }

    public static void displayTotalProducts() {
        System.out.println("Total Products: " + totalProducts);
    }

    public static void main(String[] args) {
        Product p1 = new Product("Laptop", 60000);
        Product p2 = new Product("Phone", 25000);
        p1.displayProductDetails();
        p2.displayProductDetails();
        Product.displayTotalProducts();
    }
}
```

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.

```
class Course {
    String courseName;
    int duration;
    double fee;
    static String instituteName = "ABC Institute";

    public Course(String courseName, int duration, double fee) {
        this.courseName = courseName;
        this.duration = duration;
        this.fee = fee;
    }

    public void displayCourseDetails() {
        System.out.println("Course: " + courseName + ", Duration: " +
duration + " months, Fee: " + fee + ", Institute: " + instituteName);
    }
```

```java
    public static void updateInstituteName(String newName) {
        instituteName = newName;
    }

    public static void main(String[] args) {
        Course c1 = new Course("Java", 6, 10000);
        Course c2 = new Course("Python", 5, 12000);

        c1.displayCourseDetails();
        c2.displayCourseDetails();

        Course.updateInstituteName("XYZ Academy");

        c1.displayCourseDetails();
        c2.displayCourseDetails();
    }
}
```

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.

```java
class Vehicle {
    String ownerName;
    String vehicleType;
    static double registrationFee = 5000.0; // Class variable

    public Vehicle(String ownerName, String vehicleType) {
        this.ownerName = ownerName;
        this.vehicleType = vehicleType;
    }

    public void displayVehicleDetails() {
        System.out.println("Owner: " + ownerName + ", Vehicle Type: " +
vehicleType + ", Registration Fee: " + registrationFee);
    }

    public static void updateRegistrationFee(double newFee) {
        registrationFee = newFee;
    }

    public static void main(String[] args) {
        Vehicle v1 = new Vehicle("John", "Car");
        Vehicle v2 = new Vehicle("Alice", "Bike");

        v1.displayVehicleDetails();
        v2.displayVehicleDetails();

        Vehicle.updateRegistrationFee(6000.0);

        v1.displayVehicleDetails();
        v2.displayVehicleDetails();
    }
}
```

---

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.

```java
class Student {
    public int rollNumber;
    protected String name;
    private double CGPA;


    public Student(int rollNumber, String name, double CGPA) {
        this.rollNumber = rollNumber;
        this.name = name;
        this.CGPA = CGPA;
    }

    public void setCGPA(double CGPA) {
        this.CGPA = CGPA;
    }

    public double getCGPA() {
        return CGPA;
    }

    public void displayStudent() {
        System.out.println("Roll Number: " + rollNumber + ", Name: " + name
+ ", CGPA: " + CGPA);
    }
}


class PostgraduateStudent extends Student {
    String specialization;
    public PostgraduateStudent(int rollNumber, String name, double CGPA,
String specialization) {
        super(rollNumber, name, CGPA);
        this.specialization = specialization;
    }

    public void displayPGStudent() {
        System.out.println("Roll Number: " + rollNumber + ", Name: " + name
+ ", Specialization: " + specialization);
    }

    public static void main(String[] args) {
        PostgraduateStudent pgStudent = new PostgraduateStudent(101,
"Alice", 9.1, "Data Science");
        pgStudent.displayPGStudent();
    }
}
```

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.

```
class BookLibrary {
```

```java
    public String ISBN;
    protected String title;
    private String author;

    public BookLibrary(String ISBN, String title, String author) {
        this.ISBN = ISBN;
        this.title = title;
        this.author = author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public String getAuthor() {
        return author;
    }

    public void displayBook() {
        System.out.println("ISBN: " + ISBN + ", Title: " + title + ",
Author: " + author);
    }
}


class EBook extends BookLibrary {
    public EBook(String ISBN, String title, String author) {
        super(ISBN, title, author);
    }

    public void displayEBook() {
        System.out.println("E-Book ISBN: " + ISBN + ", Title: " + title);
    }

    public static void main(String[] args) {
        EBook eBook = new EBook("123456", "Java Programming", "John Doe");
        eBook.displayEBook();
    }
}
```

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.

```java
class BankAccount {
    public long accountNumber;
    protected String accountHolder;
    private double balance;

    public BankAccount(long accountNumber, String accountHolder, double
balance) {
        this.accountNumber = accountNumber;
        this.accountHolder = accountHolder;
        this.balance = balance;
    }

    public void deposit(double amount) {
        balance += amount;
        System.out.println("Deposited: " + amount + ", New Balance: " +
balance);
    }

    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawn: " + amount + ", Remaining
Balance: " + balance);
        } else {
            System.out.println("Insufficient balance!");
        }
    }

    public void displayBalance() {
        System.out.println("Account Number: " + accountNumber + ", Balance:
" + balance);
    }
}

class SavingsAccount extends BankAccount {
    public SavingsAccount(long accountNumber, String accountHolder, double
balance) {
        super(accountNumber, accountHolder, balance);
    }

    public void displaySavingsAccount() {
        System.out.println("Account Holder: " + accountHolder + ", Account
Number: " + accountNumber);
    }

    public static void main(String[] args) {
        SavingsAccount sa = new SavingsAccount(987654321, "Alice", 5000.0);
        sa.displaySavingsAccount();
        sa.deposit(1000);
        sa.withdraw(2000);
    }
}
```

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`.

```
class Employee {
    public int employeeID;
    protected String department;
    private double salary;

    public Employee(int employeeID, String department, double salary) {
        this.employeeID = employeeID;
        this.department = department;
        this.salary = salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    public void displayEmployee() {
        System.out.println("Employee ID: " + employeeID + ", Department: "
+ department + ", Salary: " + salary);
    }
}


class Manager extends Employee {
    public Manager(int employeeID, String department, double salary) {
        super(employeeID, department, salary);
    }

    public void displayManager() {
        System.out.println("Manager ID: " + employeeID + ", Department: " +
department);
    }

    public static void main(String[] args) {
        Manager manager = new Manager(201, "IT", 80000.0);
        manager.displayManager();
        manager.setSalary(90000.0);
        manager.displayEmployee();
    }
}
```