Tóm tắt nội dung Chương 1: Tổng quan về lập trình hướng đối tượng (Buổi 1 & Buổi 2)
Buổi 1: Giới thiệu lập trình hướng đối tượng

- So sánh **lập trình truyền thống** và **lập trình hướng đối tượng (OOP)**.
- Hiểu các **đặc trưng chính của OOP**: đóng gói, kế thừa, đa hình, trừu tượng.
- Làm quen với cú pháp cơ bản của Java.

Buổi 2: Các khái niệm và kỹ thuật lập trình OOP

- Tìm hiểu **các từ khóa quan trọng** trong Java.
- Phạm vi truy cập (private, public, protected, default).
- Biến, phương thức, constructor và cách sử dụng this.

Hôm nay chúng ta sẽ bắt đầu Chương 2: Lớp và Đối tượng
Trong lập trình hướng đối tượng, **lớp (class) là bản thiết kế**, còn **đối tượng (object) là thực thể cụ thể** được tạo ra từ lớp. Chương này sẽ giúp chúng ta hiểu cách xây dựng và sử dụng **lớp và đối tượng**, cài đặt các phương thức và quản lý dữ liệu trong Java.

Nội dung các buổi tiếp theo
📌 **Buổi 3: Lớp và Đối tượng (Phần 1)**

- Hiểu **lớp và đối tượng** trong Java.
- Tạo và sử dụng **constructor**, **overloading constructor**.
- Phạm vi truy cập, từ khóa this.

📌 **Buổi 4: Lớp và Đối tượng (Phần 2)**

- Tìm hiểu **đóng gói và che giấu thông tin**.
- Phân biệt **kiểu dữ liệu nguyên thủy và đối tượng**.
- Cơ chế phép gán, truyền tham số, this, static, package.

📌 **Buổi 5: Thực hành tổng hợp về Lớp và Đối tượng**

- Ứng dụng toàn bộ kiến thức về **class, object, constructor, encapsulation**.
- Giải quyết bài toán thực tế bằng OOP.

🚀 Chúng ta hãy **bắt đầu Buổi 3 nhé!**

BUỔI 3: LỚP VÀ ĐỐI TƯỢNG (PHẦN 1)

🎯 **Mục tiêu: Sau buổi học này các em sẽ đạt được bốn năng lực sau:**

- Hiểu khái niệm **lớp (class) và đối tượng (object)** trong Java.
- Biết cách **khai báo, cài đặt, sử dụng lớp và đối tượng**.
- Hiểu và sử dụng **Overloading (nạp chồng phương thức)**.
- Hiểu và sử dụng **Constructor (hàm khởi tạo)**.

I. Mở đầu

Trước khi đi vào nội dung Buổi 3, chúng ta cùng điểm lại một số kiến thức quan trọng về **Lớp (Class) và Đối tượng (Object)** đã học trong **Buổi 2**.

1. Lớp (Class) là gì?

- **Lớp** là một khuôn mẫu (blueprint) để tạo ra các đối tượng.
- Lớp chứa **thuộc tính (biến)** và **phương thức (hành vi)** để xử lý dữ liệu.

📌 **Ví dụ:** Định nghĩa một lớp Student:

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

    void displayInfo() {
        System.out.println("Tên: " + name + ", Tuổi: " + age);
    }
}
```

Code trên có lớp Student định nghĩa hai thuộc tính name (tên) và age (tuổi). Phương thức displayInfo() in thông tin sinh viên ra màn hình theo định dạng "Tên: ..., Tuổi: ...".

2. Đối tượng (Object) là gì?

- **Đối tượng** là một thể hiện cụ thể của một lớp.
- Được tạo bằng từ khóa new.

📌 **Ví dụ:** Tạo đối tượng từ lớp Student:

```
public class Main {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.name = "An";
        s1.age = 20;
        s1.displayInfo();
    }
}
```

Code chương trình trên tạo một **đối tượng s1 của lớp Student**, sau đó:

- **s1.name = "An"** → Gán tên cho đối tượng.
- **s1.age = 20** → Gán tuổi cho đối tượng.
- **s1.displayInfo()** → Đối tượng s1 tự in thông tin của chính nó ra màn hình.

🔷 **Kết quả:**

Tên: An, Tuổi: 20

II. LỚP VÀ ĐỐI TƯỢNG TRONG JAVA

Định nghĩa lớp và đối tượng

Lớp (Class): là bản thiết kế (Blueprint).

Đối tượng (Object): là thực thể cụ thể từ lớp.

Ví dụ thực tế: "Lớp là bản vẽ của ngôi nhà, đối tượng là căn nhà cụ thể."

Khai báo lớp và tạo đối tượng trong Java

Cấu trúc lớp:

```
class Student {
    String name;
    int age;}
```

Trong code trên, lớp Student trong Java chứa hai thuộc tính name (tên, kiểu String) và age (tuổi, kiểu int). Lớp này dùng để tạo đối tượng sinh viên nhưng chưa có phương thức xử lý dữ liệu. Nếu không khai báo constructor, Java sẽ tạo một constructor mặc định.

Tạo đối tượng từ lớp (new):

```
public class Main {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.name = "An";
        s1.age = 20;
        System.out.println(s1.name + " - " + s1.age);}}
```

Code trên là chương trình định nghĩa lớp Main với phương thức main(), tạo một đối tượng Student (s1), gán giá trị "An" cho name và 20 cho age, rồi in thông tin sinh viên ra màn hình dưới dạng "An - 20" bằng System.out.println().

Thực hành nhanh:

Viết lớp `Car` có thuộc tính `brand, color`.

Tạo 2 đối tượng `Car` và hiển thị thông tin.

Code chương trình:

```
class Car {
    String brand;
    String
    color;}
public class Main {
    public static void main(String[] args) {
        Car c1 = new Car();
        c1.brand = "Toyota";
        c1.color = "Red";
        System.out.println(c1.brand + " - " + c1.color);}}
```

Code trên là chương trình định nghĩa lớp Car với hai thuộc tính brand (hãng xe) và color (màu xe). Trong main(), tạo một đối tượng Car (c1), gán giá trị "Toyota" cho brand và "Red" cho color, sau đó in ra "Toyota - Red".

📌 Thảo luận:

Sinh viên chạy chương trình và giải thích cách `new` tạo đối tượng.

? Nếu không gán giá trị cho thuộc tính thì giá trị mặc định là gì?

Gợi ý:

- Khi không gán giá trị cho thuộc tính, Java **tự động gán giá trị mặc định**.
- **Constructor** giúp khởi tạo giá trị ngay khi đối tượng được tạo, tránh giá trị mặc định không mong muốn.

📌 **Ví dụ không có constructor (dùng giá trị mặc định):**

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

    void displayInfo() {
        System.out.println("Tên: " + name + ", Tuổi: " + age);
    }
}

public class Main {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.displayInfo(); // 🔷 Output: Tên: null, Tuổi: 0
    }
}
```

Chương trình tạo một đối tượng Student (s1) nhưng không gán giá trị cho thuộc tính. Khi gọi displayInfo(), Java sử dụng **giá trị mặc định**:

- name (kiểu String) → null
- age (kiểu int) → 0

📌 **Kết quả in ra:**

Tên: null, Tuổi: 0

🚀 **Kết luận:** Nếu không có constructor để khởi tạo giá trị, Java sẽ dùng giá trị mặc định cho các thuộc tính.

Bây giờ chúng ta sẽ tìm hiểu về cách gán giá trị ban đầu cho đối tượng mới nhé!

III. CONSTRUCTOR TRONG JAVA

Constructor là Phương thức đặc biệt tự động gọi khi tạo đối tượng. Có **constructor mặc định** và **constructor có tham số**. **Dùng `this` để phân biệt biến lớp và biến tham số. Ví dụ Constructor trong Java:**

```
class Student {
    String name;
    int age;
    // Constructor có tham số
    Student(String name, int age) {
        this.name = name;
        this.age = age;}
    void display() {
        System.out.println(name + " - " + age);}}
public class Main {
    public static void main(String[] args) {
        Student s1 = new Student("An", 20);
        s1.display();}}  // Output: An – 20
```

Code chương trình trên định nghĩa lớp Student với hai thuộc tính name và age, có **constructor có tham số** để khởi tạo giá trị.

- **Constructor Student(String, int)**: Gán giá trị cho name và age bằng this.
- **Phương thức display()**: In thông tin sinh viên.

Trong main(), tạo đối tượng Student (s1) với "An" và 20, sau đó gọi display(), in ra màn hình "An - 20".

📌 **Thực hành nhanh: Viết lớp `Person` có constructor và hiển thị thông tin.**

📌 **Ví dụ mẫu:**

```
class Person {
    String name;
    int age;
    Person(String name, int age) {
        this.name = name;
        this.age = age;}
     void showInfo() {
           System.out.println("Name: " + name + ", Age: " +
                                age);}}
public class Main {
    public static void main(String[] args) {
        Person p1 = new Person("Lan", 25);
        p1.showInfo();}}
```

Chương trình định nghĩa lớp Person với hai thuộc tính name và age, có **constructor có tham số** để khởi tạo giá trị.

- **Constructor Person(String, int)**: Gán giá trị cho name và age bằng this.
- **Phương thức showInfo()**: In thông tin người.

Trong main(), tạo đối tượng Person (p1) với "Lan" và 25, sau đó gọi showInfo(), in ra màn hình "Name: Lan, Age: 25".

📌 **Câu hỏi thảo luận:**

Câu hỏi 1: Constructor có thể có kiểu trả về không?

Gợi ý trả lời:

Không, constructor **không có kiểu trả về**, kể cả void. Java tự động gọi constructor khi đối tượng được tạo mà không cần kiểu trả về.

📌 **Ví dụ sai (Lỗi biên dịch):**

```
class Student {
    int age;
    void Student() { // ❌ Sai: Constructor không thể có kiểu trả về
        age = 18;
    }
}
```

Lớp Student có một phương thức Student(), nhưng do có kiểu trả về void, nên **đây không phải constructor** mà là một phương thức bình thường. **Constructor không có kiểu trả về**, do đó chương trình sẽ sử dụng **constructor mặc định** và age sẽ không được khởi tạo đúng cách.

📌 **Ví dụ đúng:**

```
class Student {
    int age;
    Student() { // ✅ Đúng: Không có kiểu trả về
        age = 18;
    }
}
```

Lớp Student có một **constructor hợp lệ** vì **không có kiểu trả về**. Khi tạo đối tượng, age sẽ được khởi tạo với giá trị 18.

Câu hỏi 2: Có thể gọi constructor khác trong cùng một lớp không? Nếu có, sử dụng từ khóa nào?

Gợi ý trả lời:

Có, có thể gọi constructor khác trong cùng lớp bằng từ khóa this().

📌 **Ví dụ:**

```java
class Student {
    String name;
    int age;

    Student() {
        this("Unknown", 18); // Gọi constructor có tham số
    }

    Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

Lớp Student có **constructor mặc định** gọi **constructor có tham số** bằng this(), giúp tái sử dụng code và gán giá trị mặc định "Unknown" cho name và 18 cho age.

🚀 **Kết luận:** Dùng this() giúp **tái sử dụng constructor** và tránh lặp code.

Câu hỏi 3: Điều gì xảy ra nếu một lớp chỉ có constructor có tham số mà không có constructor mặc định?
Gợi ý trả lời:

- Nếu gọi new ClassName();, chương trình **bị lỗi biên dịch** vì không có constructor mặc định.
- Giải pháp: **Tạo constructor mặc định** để tránh lỗi.

📌 **Ví dụ lỗi:**

```java
class Student {
    String name;

    Student(String name) { // ❌ Chỉ có constructor có tham số
        this.name = name;
    }
}

public class Main {
    public static void main(String[] args) {
        Student s = new Student(); // ❌ Lỗi biên dịch
    }
}
```

Lớp Student chỉ có **constructor có tham số**, nên khi gọi new Student();, Java **không**

tìm thấy constructor mặc định, dẫn đến lỗi biên dịch. Để khắc phục, cần thêm một constructor mặc định.

📌 **Cách khắc phục:**

```
class Student {
    String name;

    Student() {} // ✅ Thêm constructor mặc định
    Student(String name) { this.name = name; }
}
```

Lớp Student được khắc phục bằng cách **thêm constructor mặc định**, cho phép tạo đối tượng Student mà không cần truyền tham số, tránh lỗi biên dịch khi gọi new Student();.

IV. NẠP CHỒNG PHƯƠNG THỨC (OVERLOADING)

Khái niệm Overloading:

Một lớp có thể có nhiều phương thức cùng tên nhưng khác tham số.

Lợi ích:

- Giúp mã nguồn dễ đọc, dễ bảo trì.
- Tránh lặp lại code không cần thiết.

Ví dụ minh họa:

```
class MathUtil {
    int add(int a, int b) {
        return a + b;}
    double add(double a, double b) {
        return a + b;}}
public class Main {
    public static void main(String[] args) {
        MathUtil m = new MathUtil();
        System.out.println(m.add(5, 10));     // Output: 15
        System.out.println(m.add(5.5, 2.5)); }} // Output: 8.0
```

Trong code ví dụ này, chương trình định nghĩa lớp MathUtil với hai phương thức add() dùng **Overloading** (nạp chồng phương thức):

- add(int, int) cộng hai số nguyên.
- add(double, double) cộng hai số thực.

Trong main(), tạo đối tượng MathUtil (m), gọi add(5, 10) trả về 15 và add(5.5, 2.5) trả về 8.0, sau đó in kết quả ra màn hình.

📌 **Thực hành:**

Viết lớp `Calculator` có phương thức `multiply()` cho số nguyên và số thực.

📌 **Code mẫu:**

```
class Calculator {
    int multiply(int a, int b) {
        return a * b;}
    double multiply(double a, double b) {
        return a * b;}}
public class Main {
    public static void main(String[] args) {
        Calculator c = new Calculator();
        System.out.println(c.multiply(3, 4));     // 12
        System.out.println(c.multiply(2.5, 4)); }}  // 10.0
```

Chương trình của code trên định nghĩa lớp Calculator với hai phương thức multiply() dùng **Overloading**:

- multiply(int, int) nhân hai số nguyên.
- multiply(double, double) nhân hai số thực.

Trong main(), tạo đối tượng Calculator (c), gọi multiply(3, 4) trả về 12 và multiply(2.5, 4) trả về 10.0, sau đó in kết quả ra màn hình.

📌 **Thảo luận:**

Câu hỏi 1: Các em hãy cho biết nếu hai phương thức có cùng số tham số nhưng khác kiểu dữ liệu trả về, có phải Overloading không?

Gợi ý trả lời:

Không, vì Overloading yêu cầu **danh sách tham số phải khác nhau** (số lượng hoặc kiểu dữ liệu của tham số). **Chỉ thay đổi kiểu trả về không phải là Overloading** vì Java sẽ không thể phân biệt hai phương thức chỉ dựa vào kiểu trả về.

Code 1:

```
class Test {
   int add(int a, int b) { return a + b; }
   double add(int a, int b) { return a + b; } // ❌ Lỗi: Không thể chỉ khác kiểu trả về
}
```

Lý do code trên bị lỗi:

- Hai phương thức có **cùng số lượng và kiểu tham số (int, int)**, chỉ khác kiểu trả về (int vs double).
- Java **không phân biệt phương thức dựa trên kiểu trả về**, dẫn đến lỗi biên dịch.

Code 2:

```
class Test {
   int add(int a, int b) { return a + b; }
   double add(double a, double b) { return a + b; } // ✅ Hợp lệ: Khác kiểu tham số
}
```

📌 **Lý do code hợp lệ:**

- Hai phương thức có cùng tên nhưng **khác kiểu tham số** (int, int vs double, double),

nên Java có thể phân biệt chúng.

- Đây là Overloading hợp lệ.

🎯 **Kết luận: Chỉ khác kiểu trả về không phải Overloading**, phương thức phải có danh sách tham số khác nhau. 🚀

Câu hỏi 2: Tại sao `multiply()` lại không cần viết hai tên khác nhau?

Gợi ý trả lời:
multiply() không cần đặt tên khác nhau vì **Overloading** cho phép một phương thức có nhiều phiên bản với **cùng tên nhưng khác danh sách tham số**. Java sẽ tự động chọn phương thức phù hợp dựa trên kiểu dữ liệu của đối số truyền vào.

Phân tích hai đoạn code:

✅ **Overloading hợp lệ (Không cần đổi tên phương thức)**

```
class Calculator {
    int multiply(int a, int b) { return a * b; }
    double multiply(double a, double b) { return a * b; } // ✅ Hợp lệ: Khác kiểu tham số
}
```

📌 **Lý do hợp lệ:**

- Hai phương thức **có cùng tên multiply()** nhưng **khác kiểu tham số** (int, int vs double, double), nên Java có thể phân biệt chúng.
- Khi gọi multiply(), Java sẽ chọn phương thức phù hợp dựa trên kiểu đối số truyền vào.

❌ **Nếu đặt hai tên khác nhau (Không cần thiết)**

```
class Calculator {
    int multiplyInt(int a, int b) { return a * b; }
    double multiplyDouble(double a, double b) { return a * b; }
}
```

Lớp Calculator định nghĩa hai phương thức multiplyInt() và multiplyDouble() để nhân hai số nguyên và hai số thực.

📌 Không cần đặt tên khác nhau vì Overloading cho phép sử dụng cùng một tên phương thức nhưng khác danh sách tham số.

✅ Cách tối ưu hơn (sử dụng Overloading):

```
class Calculator {
    int multiply(int a, int b) { return a * b; }
    double multiply(double a, double b) { return a * b; }
}
```

Lớp Calculator sử dụng Overloading, cho phép phương thức multiply() có cùng tên nhưng khác kiểu tham số (int và double). Điều này giúp **mã ngắn gọn, dễ đọc**, và Java tự động chọn phương thức phù hợp khi gọi.

🚀 **Kết luận:** Dùng cùng tên multiply() giúp mã ngắn gọn, dễ đọc và Java tự động

chọn phương thức phù hợp.

V. TỔNG KẾT

📌 **Tóm tắt nội dung chính của bài học:**

✅ **Lớp và đối tượng** là nền tảng của lập trình hướng đối tượng.

✅ **Overloading** giúp tăng tính linh hoạt.

✅ **Constructor** giúp khởi tạo đối tượng tự động.

🎯 **Kết quả mong đợi sau buổi học:**

✅ Sinh viên hiểu cách tạo lớp, đối tượng, constructor, Overloading.

✅ Viết được chương trình Java cơ bản sử dụng lập trình hướng đối tượng.

📌 BÀI TẬP THỰC HÀNH TRÊN LỚP

🎯 **Mục tiêu:**

- Luyện tập khai báo và sử dụng **lớp, đối tượng** trong Java. Áp dụng **constructor**, **overloading** và **phạm vi truy cập**.
- Hiểu và thực hành **các từ khóa `this`, `private`, `public`**.

Bài 1: Khai báo lớp và tạo đối tượng

📌 **Yêu cầu:**

Tạo lớp `Student` với các thuộc tính:

- `id` (số nguyên)
- `name` (chuỗi ký tự)
- `gpa` (điểm trung bình, kiểu `double`)

Viết phương thức `displayInfo()` để in thông tin sinh viên. Trong `main()`, tạo 2 sinh viên và hiển thị thông tin của họ.

📌 **Gợi ý:**

Chương trình định nghĩa lớp Student với ba thuộc tính id, name, gpa và phương thức displayInfo() để in thông tin sinh viên.

- **Phương thức displayInfo()**: Hiển thị id, name và gpa.

Trong main():

- Tạo hai đối tượng Student (s1 và s2).
- Gán giá trị cho từng sinh viên.
- Gọi displayInfo(), in ra:

ID: 101, Name: Nguyen Van A, GPA: 3.5

ID: 102, Name: Tran Thi B, GPA: 3.8

Code chương trình:

```
class Student {
    int id;
    String name;
    double gpa;

    void displayInfo() {
        System.out.println("ID: " + id + ", Name: " + name + ", GPA: " + gpa);
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.id = 101;
        s1.name = "Nguyen Van A";
        s1.gpa = 3.5;
        s1.displayInfo();

        Student s2 = new Student();
        s2.id = 102;
        s2.name = "Tran Thi B";
        s2.gpa = 3.8;
        s2.displayInfo();
    }
}
```

Bài 2: Sử dụng Constructor

📌 Yêu cầu:

- Thêm **constructor có tham số** cho lớp `Student`.
- Trong `main()`, tạo 2 sinh viên bằng constructor và hiển thị thông tin của họ.

📌 Gợi ý:

Chương trình định nghĩa lớp Student với ba thuộc tính id, name và gpa, có **constructor có tham số** để khởi tạo giá trị.

- **Constructor Student(int, String, double)**: Gán giá trị cho id, name, gpa bằng this.
- **Phương thức displayInfo()**: Hiển thị thông tin sinh viên.

Trong main():

- Tạo hai đối tượng Student (s1, s2) với dữ liệu đầu vào.
- Gọi displayInfo(), in ra:

ID: 101, Name: Nguyen Van A, GPA: 3.5

ID: 102, Name: Tran Thi B, GPA: 3.8

Code chương trình:

```
class Student {
    int id;
    String name;
    double gpa;

    // Constructor có tham số
    Student(int id, String name, double gpa) {
        this.id = id;
        this.name = name;
        this.gpa = gpa;
    }

    void displayInfo() {
        System.out.println("ID: " + id + ", Name: " + name + ", GPA: " + gpa);
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Student s1 = new Student(101, "Nguyen Van A", 3.5);
        Student s2 = new Student(102, "Tran Thi B", 3.8);

        s1.displayInfo();
        s2.displayInfo();
    }
}
```

Bài 3: Overloading Constructor

📌 Yêu cầu:

- Viết lớp `Rectangle` với hai constructor:
- Constructor mặc định (khởi tạo `width = 1`, `height = 1`).
- Constructor có tham số (`width`, `height`).
- Viết phương thức `calculateArea()` để tính diện tích hình chữ nhật.
- Trong `main()`, tạo 2 đối tượng `Rectangle` (1 dùng constructor mặc định, 1 có tham số) và in diện tích của chúng.

📌 Gợi ý:

Chương trình định nghĩa lớp Rectangle với hai thuộc tính width (chiều rộng) và height (chiều

cao), có **hai constructor** và phương thức tính diện tích.

- **Constructor mặc định (Rectangle())**: Gán width = 1, height = 1.
- **Constructor có tham số (Rectangle(int, int))**: Gán giá trị từ tham số vào width và height bằng this.
- **Phương thức calculateArea()**: Trả về diện tích (width * height).

Trong main():

- Tạo r1 bằng constructor mặc định (1x1), r2 bằng constructor có tham số (5x10).
- Gọi calculateArea(), in ra:

Diện tích r1: 1
Diện tích r2: 50

Code chương trình:

```
class Rectangle {
    int width, height;

    // Constructor mặc định
    Rectangle() {
        width = 1;
        height = 1;
    }

    // Constructor có tham số
    Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }

    int calculateArea() {
        return width * height;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(); // Dùng constructor mặc định
        Rectangle r2 = new Rectangle(5, 10); // Dùng constructor có tham số

        System.out.println("Diện tích r1: " + r1.calculateArea());
        System.out.println("Diện tích r2: " + r2.calculateArea());
    }
}
```

Bài 4: Đóng gói dữ liệu với `private` và `getter/setter`

📌 Yêu cầu:

- Viết lớp `BankAccount` với các thuộc tính **private**: `accountNumber` (số tài

khoản, kiểu `String`). `balance` (số dư, kiểu `double`).

- Cung cấp phương thức `getter` và `setter` để truy cập dữ liệu. Viết phương thức `deposit()` để nạp tiền vào tài khoản.
- Trong `main()`, tạo tài khoản ngân hàng, thực hiện giao dịch và hiển thị thông tin.

📌 **Gợi ý:**

Chương trình định nghĩa lớp BankAccount với hai thuộc tính **riêng tư** (private) là accountNumber (số tài khoản) và balance (số dư), có **constructor, getter và setter** để quản lý dữ liệu.

- **Constructor (BankAccount(String, double))**: Khởi tạo accountNumber và balance bằng this.
- **Getter (getAccountNumber(), getBalance())**: Trả về giá trị của số tài khoản và số dư.
- **Setter (deposit(double))**: Nạp tiền nếu amount > 0, cập nhật balance, in thông báo, nếu không, báo lỗi.

Trong main():

- Tạo tài khoản acc với số tài khoản "123456789" và số dư 5000.
- Hiển thị thông tin tài khoản.
- Gọi deposit(2000), cập nhật số dư lên 7000, in ra:

Tài khoản: 123456789, Số dư: 5000
Nạp tiền thành công! Số dư hiện tại: 7000

Code chương trình:

```
class BankAccount {
    private String accountNumber;
    private double balance;

    // Constructor
    BankAccount(String accountNumber, double balance) {
        this.accountNumber = accountNumber;
        this.balance = balance;
    }

    // Getter
    public String getAccountNumber() {
        return accountNumber;
    }

    public double getBalance() {
        return balance;
    }
```

```java
    // Setter (Nạp tiền)
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Nạp tiền thành công! Số dư hiện tại: " + balance);
        } else {
            System.out.println("Số tiền nạp phải lớn hơn 0!");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        BankAccount acc = new BankAccount("123456789", 5000);
        System.out.println("Tài khoản: " + acc.getAccountNumber() + ", Số dư: " + acc.getBala
        acc.deposit(2000);
    }
}
```

BÀI TRẮC NGHIỆM

📌 Mức độ 1 - Cơ bản (1 - 10)

🎯 **Mục tiêu**: Kiểm tra kiến thức cơ bản về **class, object, constructor** và **phạm vi truy cập**.

Câu 1: Lớp (class) trong Java là gì?

A. Một thực thể cụ thể trong thế giới thực

B. Một khuôn mẫu dùng để tạo ra đối tượng

C. Một phương thức trong Java

D. Một biến chứa nhiều giá trị

Câu 2: Đối tượng (object) trong Java là gì?

A. Một bản thiết kế để tạo ra lớp

B. Một thể hiện cụ thể của một lớp

C. Một phương thức để truy xuất dữ liệu

D. Một biến toàn cục

Câu 3: Từ khóa nào được dùng để tạo một đối tượng trong Java?

A. `class`

B. `static`

C. `new`

D. `object`

Câu 4: Khi một lớp không có constructor nào được khai báo, Java sẽ...?

A. Không thể tạo đối tượng từ lớp đó

B. Tự động tạo một constructor mặc định

C. Báo lỗi biên dịch

D. Tạo một constructor nhận tham số `null`

Câu 5: Constructor có nhiệm vụ gì?

A. Khởi tạo giá trị cho đối tượng khi được tạo

B. Hủy một đối tượng trong Java

C. Gọi phương thức `main()` trong lớp

D. Kiểm tra kiểu dữ liệu của biến

Câu 6: Phạm vi truy cập nào giúp một thuộc tính chỉ có thể được truy cập trong cùng một lớp?

A. `public`

B. `private`

C. `protected`

D. `default`

Câu 7: Overloading (nạp chồng phương thức) có nghĩa là gì?

A. Ghi đè phương thức của lớp cha trong lớp con

B. Có nhiều phương thức cùng tên nhưng khác tham số trong một lớp

C. Viết lại phương thức `main()` nhiều lần

D. Thay đổi kiểu dữ liệu trả về của phương thức

Câu 8: Constructor có thể có kiểu dữ liệu trả về không?

A. Có, bất kỳ kiểu dữ liệu nào

B. Không, constructor không có kiểu dữ liệu trả về

C. Chỉ có thể là kiểu `void`

D. Chỉ có thể là kiểu `int`

Câu 9: Lệnh nào sau đây tạo một đối tượng hợp lệ của lớp `Student`?

A. `Student s1 = Student();`

B. `Student s1 = new Student();`

C. `Student s1;`

D. `new Student s1();`

Câu 10: Nếu không khai báo phạm vi truy cập, phạm vi mặc định của thuộc tính là gì?

A. `private`

B. `public`

C. `protected`

D. `default`

📌 Mức độ 2 - Trung bình (11 - 15)

🎯 Mục tiêu:

- Kiểm tra khả năng hiểu và ứng dụng **Overloading constructor, từ khóa `this`, phạm vi truy cập**.
- Đánh giá khả năng sinh viên **xử lý lỗi khi sử dụng constructor**.

Câu 11: Đâu là cách đúng để sử dụng từ khóa `this` trong constructor?

A. `this = new Student();`

B. `this.name = name;`

C. `this.Student(name, age);`

D. `this -> name = name;`

Câu 12: Đâu là lợi ích chính của overloading constructor?

A. Giúp tạo nhiều đối tượng với cách khởi tạo khác nhau

B. Giúp ghi đè phương thức `main()`

C. Giúp các lớp con kế thừa phương thức của lớp cha

D. Giúp thay đổi kiểu dữ liệu của constructor

Câu 13: Trong Java, có thể có bao nhiêu constructor trong một lớp?

A. Chỉ một

B. Tối đa hai

C. Không giới hạn số lượng
D. Chỉ có constructor mặc định

Câu 14: Nếu một lớp có constructor có tham số, nhưng không có constructor mặc định, điều gì xảy ra khi gọi `new Student();`?

A. Lỗi biên dịch xảy ra

B. Java tự tạo constructor mặc định

C. Đối tượng được tạo thành công nhưng có giá trị `null`

D. Chương trình chạy nhưng không khởi tạo giá trị

Câu 15: Điều gì xảy ra nếu constructor được khai báo `private`?

A. Không thể tạo đối tượng từ lớp đó bên ngoài lớp

B. Constructor không thể hoạt động

C. Java sẽ báo lỗi biên dịch

D. Constructor chỉ có thể được gọi từ lớp cha

📌 Mức độ 3 - Nâng cao (16 - 20)

🎯 Mục tiêu:

- Kiểm tra khả năng vận dụng kiến thức OOP vào bài toán thực tế.
- Đánh giá khả năng **tư duy logic về Overloading, Encapsulation và Constructor**.

Câu 16: Hai phương thức có cùng tên, cùng danh sách tham số nhưng khác kiểu trả về có phải overloading không?

A. Có

B. Không

Câu 17: Cách nào đúng để gọi một constructor khác trong cùng một lớp?

A. `this(parameters);`

B. `new this(parameters);`

C. `this.Student(parameters);`
D. `Student(parameters);`

Câu 18: Tại sao nên sử dụng `private` cho các thuộc tính?

A. Giúp bảo vệ dữ liệu và tránh truy cập trái phép

B. Giúp phương thức `main()` dễ truy cập hơn

C. Giúp chương trình chạy nhanh hơn

D. Không có tác dụng gì đặc biệt

Câu 19: Phát biểu nào sai về constructor?

A. Constructor có thể `private`

B. Constructor không thể Overloading

C. Constructor không có kiểu trả về

D. Constructor được gọi tự động khi đối tượng được tạo

BÀI TẬP LẬP TRÌNH

Bài 1: Viết lớp `Book`
Thuộc tính: `title`, `author`, `price`. Constructor có tham số.
Phương thức `displayInfo()` để hiển thị thông tin sách. Trong `main()`, tạo 2 sách và hiển thị thông tin.

Bài 2: Viết lớp `Circle`
Thuộc tính `radius` (bán kính, kiểu `double`). Constructor có tham số.
Phương thức `calculateArea()` để tính diện tích. Trong `main()`, tạo 2 hình tròn và hiển thị diện tích.

Bài 3: Viết lớp `Employee` với Overloading Constructor
`Employee` có `name`, `salary`. Overloading constructor:
1 constructor chỉ có `name` (mặc định `salary = 5000`). 1 constructor có cả `name` và `salary`.
Phương thức `displayInfo()` hiển thị thông tin nhân viên.
Trong `main()`, tạo 2 nhân viên với 2 cách khởi tạo khác nhau.