📌 BUỔI 4: LỚP VÀ ĐỐI TƯỢNG (PHẦN 2)

⏳ **Thời lượng**: 3 tiết

🎯 **Mục tiêu**:

- Hiểu **đóng gói và che giấu thông tin (Encapsulation)** trong OOP.
- Phân biệt **kiểu dữ liệu nguyên thủy và kiểu đối tượng** trong Java.
- Hiểu rõ **cơ chế phép gán và truyền tham số** trong Java.
- Sử dụng hiệu quả từ khóa **this, static** và khái niệm **package**.

🔷 Phần 1: Giới thiệu & Ôn tập

1. **Mục tiêu buổi học:** Tóm tắt nội dung cần đạt được.
2. **Ôn tập Buổi 3:** *Class, Object, Constructor, phạm vi truy cập, từ khóa this.*
3. **Tại sao cần Đóng gói & Che giấu thông tin?**
4. **Tình huống thực tế:**

   Quản lý thông tin tài khoản ngân hàng, làm sao để bảo vệ dữ liệu?

🔷 Phần 2: Đóng gói và Che giấu thông tin

1. **Định nghĩa Đóng gói (Encapsulation)**
2. **Lợi ích của đóng gói**: *Bảo vệ dữ liệu, dễ bảo trì, hạn chế lỗi…*
3. **Mức độ truy cập trong Java:** private, public, protected, default.
4. **Ví dụ về Đóng gói** – So sánh code có và không có private.
5. **Getter & Setter – Tại sao cần thiết?**
6. **Thực hành nhanh**: Viết class BankAccountsử dụng privatevà getter/setter.
7. **Thảo luận**: Nếu không có getter/setter, hệ thống có thể gặp vấn đề gì?

🔷 Phần 3: Kiểu Dữ liệu Nguyên thủy vs Đối tượng

1. **Phân biệt kiểu dữ liệu nguyên thủy (int, double…) và đối tượng (String, Array…)**
2. **Kiểu tham trị (pass-by-value) vs Tham chiếu (pass-by-reference)**
3. **Ví dụ minh họa – Phép gán giá trị kiểu nguyên thủy**
4. **Ví dụ minh họa – Phép gán đối tượng, tại sao cả hai thay đổi cùng lúc?**
5. **Bài tập nhanh:** Viết chương trình so sánh int và Integer.

🔷 Phần 4: Cơ chế Phép gán & Truyền tham số

1. **Phép gán trong Java: Khi nào dữ liệu bị thay đổi?**
2. **Truyền tham trị vs Truyền tham chiếu – So sánh chi tiết**
3. **Ví dụ truyền tham số với kiểu nguyên thủy**
4. **Ví dụ truyền tham số với đối tượng – Tại sao thay đổi giá trị?**
5. **Thực hành nhanh:**
   Viết chương trình thử nghiệm truyền tham trị và tham chiếu

🔷 Phần 5: this, static, package

1. **Từ khóa this trong Java – Khi nào cần dùng?**
2. **Ví dụ this– Phân biệt biến instance và biến cục bộ**
3. **Từ khóa static– Dùng để làm gì?**
4. **Ví dụ static– Đếm số lượng sinh viên được tạo**
5. **Giới thiệu package– Cách tổ chức mã nguồn chuyên nghiệp**
6. **Ví dụ sử dụng package trong dự án lớn**
7. **Bài tập thực hành tổng hợp:** Viết class Employeesử dụng private,static, this.
8. **Tổng kết & Giao bài tập về nhà**

🎯 Slide 1: Mục tiêu buổi học:

✓ Hiểu **đóng gói (Encapsulation)** và che giấu thông tin trong OOP.

✓ Phân biệt **kiểu dữ liệu nguyên thủy và kiểu đối tượng** trong Java.

✓ Hiểu rõ **cơ chế phép gán, truyền tham số, từ khóa this, static, package**.

✓ Thực hành với các bài tập **thực tế & ứng dụng OOP**.

🎯 Slide 2: Mục tiêu Buổi Học

Sau buổi học này, sinh viên sẽ:

✓ **Hiểu rõ về Đóng gói (Encapsulation)** và cách bảo vệ dữ liệu trong Java.

✓ **Phân biệt được kiểu dữ liệu nguyên thủy và kiểu đối tượng** trong Java.

✓ **Nắm vững cơ chế phép gán, truyền tham số**, tránh lỗi khi lập trình.

✓ **Sử dụng từ khóa this, static** để quản lý dữ liệu và bộ nhớ hiệu quả.

✓ **Biết cách tổ chức mã nguồn bằng package** để xây dựng dự án chuyên nghiệp.

✓ **Thực hành với các bài tập ứng dụng, giúp nắm chắc kiến thức OOP.**

🎯 Slide 3: Ôn tập kiến thức buổi trước

Trước khi học nội dung mới, hãy cùng ôn lại các kiến thức đã học trong **Buổi 3 - Lớp và Đối tượng (Phần 1):**

1. **Lớp (Class) và Đối tượng (Object)**
   Lớp là bản thiết kế, **đối tượng** là thực thể cụ thể được tạo ra từ lớp.
   Ví dụ: *Lớp Student chứa thuộc tính name, age, phương thức display().*

2. **Constructor & Overloading Constructor**
   - **Constructor** là phương thức đặc biệt được gọi khi tạo đối tượng.
   - **Overloading** giúp tạo nhiều constructor với các tham số khác nhau.

3. **Phạm vi truy cập trong Java (public, private, protected, default) Giúp kiểm soát quyền truy cập vào dữ liệu của lớp.**
4. **Từ khóa this**
   Dùng để tham chiếu đến đối tượng hiện tại trong lớp.

📌 Câu hỏi kiểm tra nhanh:

🧐 Bạn có thể trả lời các câu hỏi sau không?

✓ **1. Lớp và đối tượng khác nhau như thế nào?**

✓ **2. Constructor hoạt động ra sao?**

✓ **3. Từ khóa this dùng để làm gì?**

🎯 Slide 4: Giới thiệu về Đóng gói (Encapsulation)

🔷 Đóng gói (Encapsulation) là gì?

✓ Là một trong bốn đặc trưng quan trọng của Lập trình Hướng Đối Tượng.

✓ **Che giấu dữ liệu** của đối tượng và chỉ cho phép truy cập thông qua các phương thức được xác định trước.
✓ **Ngăn chặn truy cập trực tiếp vào dữ liệu**, giúp bảo mật và bảo trì dễ dàng hơn.

🔷 Lợi ích của đóng gói:

✅ **Bảo vệ dữ liệu** – Ngăn chặn sửa đổi trái phép.

✅ **Kiểm soát cách dữ liệu được truy cập** thông qua các phương thức setter/getter.

✅ **Giúp mã nguồn dễ bảo trì** và mở rộng hệ thống dễ dàng.

🔷 **Ví dụ thực tế:**

🧐 **Bạn có muốn người khác thay đổi số dư tài khoản ngân hàng của mình một cách tùy tiện không?**
💡 Đó là lý do **đóng gói dữ liệu** quan trọng trong lập trình!

🎯 Slide 5: Mức độ Truy cập trong Java

🔷 **Java cung cấp 4 mức độ truy cập để bảo vệ dữ liệu trong lớp:**

Mức độ truy cập	Trong cùng lớp	Trong cùng package	Lớp con (khác package)	Bất kỳ đâu
private	✅ Có	❌ Không	❌ Không	❌ Không
default	✅ Có	✅ Có	❌ Không	❌ Không
protected	✅ Có	✅ Có	✅ Có	❌ Không
public	✅ Có	✅ Có	✅ Có	✅ Có

🔷 **Giải thích ngắn gọn:**

✓ private: Chỉ có thể truy cập trong chính lớp đó.

✓ default: Có thể truy cập trong cùng package.

✓ protected: Cho phép truy cập trong package và lớp con.

✓ public: Cho phép truy cập từ bất kỳ đâu.

🔷 **Ví dụ minh họa:**

```
class Student {

    private String name;   // Chỉ truy cập được trong lớp Student
    protected int age;     // Truy cập được trong package & lớp con
    public String school; // Có thể truy cập từ bất kỳ đâu
}
```

📌 **Câu hỏi thảo luận:**

🧐 **1. Tại sao không nên khai báo tất cả biến là public?**

🧐 **2. Khi nào nên sử dụng protectedthay vì private?**

🎯 Slide 6: Ví dụ về Đóng gói trong Java

📌 **Nội dung chính:**

🔷 **Triển khai đóng gói bằng private và getter/setter**

✓ **Dữ liệu được bảo vệ** bằng từ khóa private.

✓ **Chỉ truy cập dữ liệu qua phương thức getter và setter**.

🔷 **Ví dụ minh họa:**

```
class BankAccount {
    private double balance; // Không thể truy cập trực tiếp từ bên ngoài
    // Constructor
    public BankAccount(double initialBalance) {
        if (initialBalance >= 0) {
            this.balance = initialBalance;
        }
    }

    // Getter để lấy số dư
    public double getBalance() {
        return balance;
    }

    // Setter để cập nhật số dư
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        }
    }
}
```

🔷 **Giải thích:**

✓ balanceđược khai báo **private → Chỉ truy cập thông qua phương thức**.

✓ **Phương thức deposit()kiểm tra dữ liệu đầu vào**, tránh lỗi nạp tiền âm.

✓ **Dữ liệu được bảo vệ, tránh thay đổi trực tiếp từ bên ngoài!**

📌 **Câu hỏi thảo luận:**

🧐 **1. Điều gì xảy ra nếu không dùng private?**

🧐 **2. Nếu muốn thêm tính năng rút tiền (withdraw()), cần làm gì?**

🎯 **Slide 7: Thực hành – Viết Code Đóng gói**
Yêu cầu:

✓ Viết một lớp Studentđể quản lý thông tin sinh viên.

✓ Sử dụng **đóng gói (private)** để bảo vệ dữ liệu.

✓ Tạo **getter & setter** để truy xuất dữ liệu một cách an toàn.

🔷 Gợi ý Code:

```
class Student {

    private String name;
    private int age;

    // Constructor
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Getter
    public String getName() {
        return name;
    }

    // Setter
    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        if (age > 0) {
            this.age = age;
        }
    }
}
```

🔷 Thử nghiệm code:

```
public class Main {

    public static void main(String[] args) {
        Student s1 = new Student("An", 20);
        System.out.println("Tên: " + s1.getName());

        s1.setAge(22);
```

```
            System.out.println("Tuổi mới: " + s1.getAge());
        }
    }
```

📌 **Câu hỏi thảo luận:**

🧐 **1. Tại sao namevà agecần là private?**

🧐 **2. Nếu bỏ setter, chương trình có chạy đúng không?**

🤔 **3. Làm sao để ngăn chặn việc nhập tuổi âm (age < 0)?**

🎯 Slide 8: Kiểu Dữ liệu Nguyên thủy vs Kiểu Đối tượng

🔷 1. Kiểu Dữ liệu Nguyên thủy (Primitive Data Types)

✔ Lưu trữ **giá trị thực tế**.

✔ Hiệu suất cao, chiếm ít bộ nhớ hơn.

✔ Không có phương thức đi kèm.

✔ Ví dụ: int, double, char, boolean.

🔷 2. Kiểu Đối tượng (Reference Data Types)

✔ Lưu trữ **địa chỉ (tham chiếu) đến vùng nhớ** chứa dữ liệu.

✔ Có thể có nhiều phương thức đi kèm.

✔ Ví dụ: String, Array, Scanner, Student(lớp tự định nghĩa).

🔷 3. So sánh Kiểu Nguyên thủy vs Đối tượng

Đặc điểm	**Kiểu Nguyên thủy**	**Kiểu Đối tượng**
Lưu trữ	Giá trị thực	Địa chỉ vùng nhớ
Kích thước	Nhỏ, cố định	Lớn, phụ thuộc vào dữ liệu
Tốc độ	Nhanh	Chậm hơn do truy cập gián tiếp
Có phương thức?	❌ Không	✅ Có (ví dụ: String.length())

🔷 **Ví dụ minh họa:**

```
// Kiểu nguyên thủy
int a = 10;
int b = a; // Copy giá trị

b = 20;
System.out.println(a); // Kết quả: 10 (a không bị thay đổi)

// Kiểu đối tượng
Student s1 = new Student("An", 20);
Student s2 = s1; // Copy địa chỉ

s2.setName("Bình");
System.out.println(s1.getName()); // Kết quả: Bình (s1 cũng bị thay đổi)
```

📌 **Câu hỏi thảo luận:**

🧐 **1. Tại sao int không bị thay đổi khi gán b = a nhưng Student lại thay đổi?**

🧐 **2. Khi nào nên dùng kiểu nguyên thủy, khi nào nên dùng kiểu đối tượng?**

🎯 **Slide 9: Cơ chế Phép gán trong Java**

🔷 **1. Phép gán với kiểu dữ liệu nguyên thủy**

✓ Gán **giá trị thực tế** từ biến này sang biến khác.

✓ **Hai biến độc lập**, thay đổi biến này **không ảnh hưởng** đến biến kia.

🔷 **2. Phép gán với kiểu đối tượng**

✓ Gán **tham chiếu (địa chỉ vùng nhớ)** từ biến này sang biến khác.

✓ **Hai biến trỏ đến cùng một đối tượng**, thay đổi biến này **ảnh hưởng đến biến kia**.

🔷 **3. Ví dụ minh họa:**

📌 **Phép gán với kiểu nguyên thủy:**

```
int a = 10;
int b = a; // Sao chép giá trị của a
b = 20;
System.out.println(a); // Kết quả: 10 (a không bị thay đổi)
```

📌 **Phép gán với kiểu đối tượng:**

```
class Student {
```

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

public class Main {
    public static void main(String[] args) {
        Student s1 = new Student("An");
        Student s2 = s1; // Sao chép tham chiếu

        s2.name = "Bình";
        System.out.println(s1.name); // Kết quả: Bình (s1 cũng bị thay đổi)
    }
}
```

📌 **Câu hỏi thảo luận:**

🧐 **1. Tại sao intkhông bị thay đổi khi gán b = a, nhưng Studentlại thay đổi khi gán s2 = s1?**
🧐 **2. Nếu muốn sao chép một đối tượng mà không bị ảnh hưởng bởi thay đổi, ta phải làm thế nào?**

🎯 **Slide 10: Cơ chế Truyền tham số trong Java**

🔷 **Java sử dụng cơ chế "Truyền tham trị" (Pass by Value)**

✓ **Với kiểu nguyên thủy (int, double, char...)**

→ **Truyền bản sao của giá trị**, thay đổi bên trong phương thức **không ảnh hưởng** đến biến gốc.

✓ **Với kiểu đối tượng (String, Array, Student...)**

→ **Truyền bản sao của tham chiếu (địa chỉ vùng nhớ)**, thay đổi nội dung **sẽ ảnh hưởng đến đối tượng gốc**.

🔷 **Ví dụ minh họa:**

📌 **Truyền tham số với kiểu nguyên thủy:**

```
public class Example {

    static void changeValue(int x) {
        x = 20; // Thay đổi giá trị trong phương thức
```

```
    }

    public static void main(String[] args) {
        int a = 10;
        changeValue(a);
        System.out.println(a); // Kết quả: 10 (a không bị thay đổi)
    }
. }
```

📌 **Truyền tham số với đối tượng:**

```
class Student {
    String name;
    Student(String name) {
        this.name = name;
    }

    static void modify(Student s) {
            s.name = "Bình"; // Thay đổi giá trị của đối tượng
    }

    public static void main(String[] args) {
        Student s1 = new Student("An"); modify(s1);
        System.out.println(s1.name); // Kết quả: Bình (s1 bị thay đổi)
    }
}
```

📌 **Câu hỏi thảo luận:**

🧐 **1. Tại sao a không thay đổi trong ví dụ đầu tiên, nhưng s1 lại thay đổi trong ví dụ thứ hai?**

🧐 **2. Nếu muốn tránh thay đổi giá trị của đối tượng khi truyền vào phương thức, ta phải làm thế nào?**

🎯 **Slide 11: Từ khóa thistrong Java**

🔷 **1. This là gì?**

✓ this là một từ khóa đặc biệt trong Java, đại diện cho **đối tượng hiện tại**.

✓ Dùng để phân biệt **biến instance** và **biến cục bộ** trong phương thức hoặc constructor.

🔷 **2. Khi nào cần dùng this?**

✓ Khi **tên biến cục bộ trùng với tên biến instance**.

✓ Khi cần **gọi constructor khác trong cùng lớp**.

✓ Khi cần **truyền chính đối tượng hiện tại vào phương thức khác**.

🔷 3. Ví dụ minh họa:

📌 Phân biệt biến instance và biến cục bộ:

```
class Student {
    private String name;

    // Constructor sử dụng `this`
    public Student(String name) {
        this.name = name; // `this.name` là biến instance, `name` là biến cục bộ
    }

    public void display() {
        System.out.println("Tên sinh viên: " + this.name);
    }
}
```

📌 Dùng thisđể gọi constructor khác:

```
class Student {

    private String name;
    private int age;

    public Student(String name) {
        this(name, 18); } // Gọi constructor khác trong cùng lớp
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

📌 Truyền thisvào phương thức khác:

```
class Student {

    void show(Student s) {
        System.out.println("Đối tượng: " + s);
    }

    void display() {
        show(this); // Truyền chính đối tượng hiện tại
    }
```

```
}
```

📌 **Câu hỏi thảo luận:**

🧐 **1. Nếu bỏ thistrong constructor, chương trình có chạy đúng không?**

🧐 **2. Khi nào thisthực sự cần thiết?**

🧐 **3. Bạn có thể viết một chương trình đơn giản sử dụng thiskhông?**

📌 **Hình ảnh minh họa (gợi ý):**

🖼 **Sơ đồ minh họa thistrỏ đến đối tượng hiện tại trong bộ nhớ.**

📌 **Tiếp theo, chúng ta sẽ tìm hiểu về từ khóa statictrong Java!** 🚀

🎯 **Slide 12: Từ khóa statictrong Java**

🔷 **1. staticlà gì?**

✓ staticlà từ khóa trong Java dùng để khai báo **biến, phương thức hoặc khối lệnh thuộc về lớp**, không thuộc về đối tượng cụ thể.
✓ **Biến staticđược chia sẻ giữa tất cả các đối tượng của lớp** thay vì mỗi đối tượng có một bản sao riêng.

🔷 **2. Khi nào cần dùng static?**

✓ Khi muốn **tạo biến dùng chung** giữa tất cả đối tượng (ví dụ: biến đếm số lượng đối tượng).
✓ Khi muốn **tạo phương thức có thể gọi mà không cần tạo đối tượng** (ví dụ: phương thức tiện ích như Math.pow()).

🔷 **3. Ví dụ minh họa:**

📌 **Sử dụng biến static:**

```
class Student {

    private String name;
    private static int count = 0; // Biến dùng chung tăng

    public Student(String name) {
        this.name = name;
        count++; // Mỗi khi tạo đối tượng mới, biến count
    }

    public static int getStudentCount() {
```

```
        return count;
    }
}

public class Main {
    public static void main(String[] args) {
        Student s1 = new Student("An");
        Student s2 = new Student("Bình");

        System.out.println("Số lượng sinh viên: " + Student.getStudentCount()); // Output: 2
    }
}
```

📌 **Sử dụng phương thức static:**

```
class MathUtils {

    public static int square(int x) {
        return x * x;
    }
}

public class Main {
    public static void main(String[] args) {
        System.out.println(MathUtils.square(5)); // Gọi mà không cần tạo đối tượng
    }
}
```

📌 **Câu hỏi thảo luận:**

🧐 **1. Nếu bỏ statictrong count, kết quả có thay đổi không?**

🧐 **2. Khi nào nên dùng phương thức staticthay vì phương thức thông thường?**

🧐 **3. Có thể sử dụng thistrong phương thức statickhông? Vì sao?**

🎯 **Slide 13: Giới thiệu về packagetrong Java**

🔷 **1. packagelà gì?**

✓ packagelà một nhóm các lớp (class) và giao diện (interface) có liên quan, giúp **tổ chức mã nguồn khoa học và dễ quản lý**.

✓ Giúp **tránh xung đột tên lớp**, đặc biệt khi làm việc với các dự án lớn.

🔷 2. Lợi ích của package:

✅ **Giúp tổ chức code gọn gàng**, dễ tìm kiếm.

✅ **Kiểm soát truy cập tốt hơn** giữa các lớp.

✅ **Tái sử dụng mã nguồn dễ dàng**, có thể import các package khi cần.

🔷 3. Cách sử dụng packagetrong Java:

📌 Tạo một package:

```
package mypackage; // Định nghĩa package


public class Student { private
        String name;

        public Student(String name) {
                this.name = name;
        }

        public void display() {
                        System.out.println("Tên sinh viên: " + name);
        }
}
```

📌 Sử dụng lớp từ package khác:

```
import mypackage.Student; // Import package


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

📌 Câu hỏi thảo luận:

🧐 **1. Điều gì xảy ra nếu không khai báo package?**

🧐 **2. Tại sao cần importkhi sử dụng lớp từ package khác?**

🧐 **3. Bạn có thể tự tạo một package và sử dụng nó trong chương trình không?**

🎯 Slide 14: Thực hành – Tạo và Sử dụng package trong Java

📌 Bài tập thực hành:

✓ **Mục tiêu:** Giúp sinh viên hiểu cách tổ chức mã nguồn bằng package.

✓ **Yêu cầu:**

- **Tạo một package universitychứa lớp Student.**
- **Dùng lớp Student trong một chương trình khác bằng cách import package.**

🔷 1. Bước 1: Tạo package và lớp Student

📌 Tạo tệp Student.javatrong thư mục university

```
package university; // Định nghĩa package
public class Student {
    private String name;

    public Student(String name) {
        this.name = name;
    }

    public void display() {
        System.out.println("Tên sinh viên: " + name);
    }
}
```

🔷 2. Bước 2: Sử dụng Studenttrong chương trình khác

📌 Tạo tệp Main.javađể sử dụng lớp Student

```
import university.Student; // Import package

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

📌 Câu hỏi thảo luận:

🧐 **1. Nếu không dùng import, làm sao để gọi lớp Studenttừ package university?**

🧐 **2. Có thể có nhiều package trong cùng một dự án không?**

🧐 **3. Nếu đổi thư mục chứa file Student.java, chương trình có chạy đúng không?**

🎯 Slide 15: Tổng kết Kiến thức Buổi Học

📌 Tiêu đề Slide:

TỔNG KẾT BUỔI HỌC

📌 Nội dung chính:

🔷 1. Đóng gói (Encapsulation)

✓ Bảo vệ dữ liệu bằng private, chỉ truy cập thông qua **getter & setter**.

✓ Kiểm soát truy cập bằng **các mức độ truy cập** (private, public, protected, default).

🔷 2. Kiểu Dữ liệu Nguyên thủy vs Kiểu Đối tượng

✓ **Nguyên thủy (int, double)**: Lưu giá trị trực tiếp, nhanh hơn.

✓ **Đối tượng (String, Student)**: Lưu tham chiếu đến vùng nhớ trên Heap.

🔷 3. Phép gán và Truyền tham số

✓ Kiểu nguyên thủy **truyền giá trị → Không bị ảnh hưởng khi thay đổi**.

✓ Kiểu đối tượng **truyền tham chiếu → Bị ảnh hưởng nếu thay đổi giá trị bên trong phương thức**.

🔷 4. Từ khóa this

✓ **Tham chiếu đến đối tượng hiện tại**, dùng để phân biệt biến instance và biến cục bộ.

✓ Có thể dùng this()để gọi constructor khác trong cùng lớp.

🔷 5. Từ khóa static

✓ Dùng để khai báo **biến, phương thức dùng chung** cho tất cả đối tượng.

✓ **Phương thức static**có thể gọi mà không cần tạo đối tượng.

🔷 6. packagetrong Java

✓ Giúp **tổ chức code gọn gàng**, tránh xung đột tên lớp.

✓ **Dùng import**để sử dụng lớp từ package khác.

📌 **Câu hỏi kiểm tra nhanh:**

🧐 **1. statickhác gì so với this?**

🧐 **2. Khi nào cần sử dụng privatethay vì public?**

🧐 **3. Làm sao để truyền tham số mà không làm thay đổi đối tượng gốc?**

🎯 **Slide 16: Bài tập Tổng hợp – Lập trình OOP**
📌 **Yêu cầu bài tập:**

✓ Viết chương trình **quản lý nhân viên (Employee)** với các yêu cầu sau:

Sử dụng Encapsulationđể bảo vệ dữ liệu.

Áp dụng thisđể tham chiếu đến biến instance.
Dùng staticđể đếm số lượng nhân viên được
tạo. **Tổ chức mã nguồn trong package**.

🔷 **1. Bước 1: Tạo lớp Employeetrong package company**

📌 **File Employee.java**

```
package company;

public class Employee {
    private String name; private
    double salary;
    private static int count = 0; // Biến static để đếm số nhân viên

    // Constructor
    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
        count++; // Tăng số lượng nhân viên khi tạo đối tượng
    }

    // Getter & Setter
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getSalary() {
        return salary;
```

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

    // Phương thức static để lấy số lượng nhân viên
    public static int getEmployeeCount() {
        return count;
    }

    // Hiển thị thông tin nhân viên
    public void display() {
        System.out.println("Tên: " + name + ", Lương: " + salary);
    }
}
```

🔷 2. Bước 2: Viết chương trình chính để sử dụng Employee

📌 File Main.java

```
import company.Employee; // Import package


public class Main {
    public static void main(String[] args) {
        Employee e1 = new Employee("An", 5000);
        Employee e2 = new Employee("Bình", 7000);

        e1.display();
        e2.display();
        System.out.println("Tổng số nhân viên: " +
        Employee.getEmployeeCount());
    }
}
```

📌 Câu hỏi thảo luận:

🧐 1. Tại sao countphải là static?

🧐 2. Nếu không có this, chương trình có hoạt động đúng không?

🧐 3. Nếu muốn sắp xếp nhân viên theo mức lương, cần làm gì thêm?

CÂU HỎI TRẮC NGHIỆM

📌 Mức độ 1 - Cơ bản (1 - 7)

🎯 **Mục tiêu:** Kiểm tra kiến thức cơ bản về lớp, đối tượng, 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: Nếu 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

📌 **Mức độ 2 - Trung bình (8 - 14)**

🎯 **Mục tiêu:** Kiểm tra khả năng hiểu và ứng dụng Overloading constructor, từ khóa this, phạm vi truy cập.

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

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: 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 (15 - 20)**

🎯 **Mục tiêu:** Kiểm tra khả năng vận dụng kiến thức OOP vào bài toán thực tế.

Câu 13: Hai phương thức có cùng tên, cùng danh sách tham số nhưng khác kiểu trả về có phải Overloading không?

A. Có

B. Không

📌 Mức độ 2 - Trung bình (tiếp tục từ câu 14)

🎯 **Mục tiêu:** Kiểm tra khả năng hiểu về phạm vi truy cập, static, và Overriding.

Câu 14: Biến staticcó thể được truy cập bằng cách nào?

A. Chỉ thông qua đối tượng của lớp

B. Chỉ có thể được truy cập từ phương thức static

C. Có thể truy cập bằng tên lớp hoặc đối tượng

D. Chỉ có thể được truy cập từ lớp cha

Câu 15: Khi nào nên sử dụng finalvới phương thức **trong Java?**

A. Khi muốn cho phép ghi đè phương thức

B. Khi muốn ngăn chặn ghi đè phương thức

C. Khi muốn tạo constructor mặc định

D. Khi muốn phương thức có thể thay đổi nội dung trong lớp con

📌 **Mức độ 3 - Nâng cao (16 - 20)**

🎯 **Mục tiêu:** Kiểm tra khả năng vận dụng kiến thức OOP vào bài toán thực tế.

Câu 16: Điều gì xảy ra nếu lớp con có một phương thức trùng tên với phương thức của lớp cha mà không có @Override?

A. Java sẽ báo lỗi

B. Chương trình vẫn chạy nhưng không ghi đè phương thức lớp cha

C. Lớp con không thể gọi phương thức của lớp cha

D. Lớp con sẽ tự động gọi phương thức của lớp cha thay vì phương thức của mình

Câu 17: Interface trong Java KHÔNG thể chứa thành **phần nào sau đây?**

A. Hằng số

B. Phương thức static

C. Phương thức default

D. Constructor

Câu 18: Đâu là cách đúng để gọi constructor của lớp **cha từ lớp con?**

A. super();

B. this();

C. super.constructor();

D. ParentClass();

Câu 19: Điều gì xảy ra nếu một lớp abstract KHÔNG có bất kỳ phương thức abstract nào?

A. Java sẽ báo lỗi

B. Lớp đó vẫn hợp lệ

C. Không thể tạo đối tượng từ lớp đó

D. B và C đều đúng

Câu 20: Điều gì xảy ra nếu một lớp cha có phương thức **private và lớp con cố gắng ghi đè nó?**

A. Java sẽ báo lỗi

B. Lớp con có thể ghi đè bình thường

C. Lớp con sẽ có một phương thức mới mà không liên quan đến lớp cha

D. Không có tác động gì, nhưng phương thức lớp con sẽ không hoạt động

📌 **Tổng kết**

🔷 **20 câu hỏi trắc nghiệm** đã được thiết kế từ **cơ bản đến nâng cao**, giúp sinh viên:

✅ **Hiểu khái niệm cốt lõi của OOP** (Class, Object, Constructor, Encapsulation).

✅ **Phân biệt được Overloading vs Overriding**.

✅ **Sử dụng this, static, superhợp lý**.

✅ **Áp dụng kiến thức OOP vào các bài toán thực tế.**

BÀI TẬP TỰ LUYỆN TẬP

📌 Bài tập 1: Đóng gói dữ liệu (Encapsulation)

Đề bài:

Viết một lớp BankAccountvới các thuộc tính sau:

private double balance(số dư tài khoản)

private String ownerName(tên chủ tài khoản)

Yêu cầu:

- ✓ Viết constructor để khởi tạo tài khoản.
- ✓ Viết gettervà setterđể truy cập số dư nhưng không cho phép số dư âm.
- ✓ Viết phương thức deposit(double amount)để nạp tiền vào tài khoản.
- ✓ Viết phương thức withdraw(double amount)để rút tiền nhưng không cho phép rút quá số dư.

📌 Gợi ý:

Sử dụng privateđể bảo vệ dữ liệu.

Dùng getterđể lấy số dư, setterđể kiểm soát thay đổi.

📌 Bài tập 2: Phân biệt Kiểu Dữ liệu Nguyên thủy vs **Đối tượng**

Đề bài:

Viết chương trình để kiểm tra sự khác biệt giữa kiểu nguyên thủy (int) và kiểu đối tượng (Integer).

Yêu cầu:

- ✓ Khai báo int a = 10;và Integer b = new Integer(10);
- ✓ Viết phương thức changeValue(int x, Integer y) làm thay đổi giá trị của x và y.
- ✓ Gọi phương thức changeValue() trong main()và in ra kết quả.

📌 Gợi ý:

Kiểu nguyên thủy **truyền giá trị** → Không thay đổi sau khi ra khỏi phương thức.

Kiểu đối tượng **truyền tham chiếu** → Có thể bị thay đổi nếu tham chiếu đến cùng một đối tượng.

📌 Bài tập 3: Sử dụng từ khóa this

Đề bài:

Viết lớp Studentvới các thuộc tính:

```
private String name;
private int age;
```

Yêu cầu:

- ✓ Viết constructor nhận tham số namevà age.
- ✓ Dùng thisđể phân biệt biến instance và biến tham số.
- ✓ Viết phương thức display() để in ra thông tin sinh viên.

📌 Gợi ý:

Dùng this.name = name; để gán giá trị từ tham số vào biến instance.

📌 Bài tập 4: Sử dụng static

Đề bài:

Viết lớp Employeeđể quản lý số lượng nhân viên trong công ty.

Yêu cầu:

✓ Khai báo biến private static int count = 0; để đếm số nhân viên.

✓ Mỗi khi tạo một nhân viên mới, biến count tăng lên.

✓ Viết phương thức public static int getEmployeeCount() để lấy số lượng nhân viên.

📌 Gợi ý:

static giúp biến count được chia sẻ giữa tất cả các đối tượng.

📌 Bài tập 5: Tạo và sử dụng package

Đề bài:

✓ Tạo một package school.

✓ Trong package school, tạo lớp Teacher với thuộc tính name và phương thức teach().

✓ Tạo lớp Main bên ngoài package, dùng import school.Teacher; để truy cập lớp Teacher.

📌 Gợi ý:

Khai báo package school; ở đầu file Teacher.java.
Dùng import school.Teacher;trong Main.java.

📌 Tổng kết

📝 **5 bài tập trên giúp sinh viên:**

✅ Hiểu rõ về **Đóng gói (Encapsulation)**.

✅ Phân biệt **Kiểu nguyên thủy vs Đối tượng**.

✅ Biết cách sử dụng **this, static, package**.

✅ Luyện tập **viết chương trình OOP hoàn chỉnh**.

BÀI TẬP LUYỆN TẬP

📌 Bài tập 1: Tạo lớp đơn giản (Dễ)

Đề bài:

Viết một lớp Carvới các thuộc tính sau:

private String brand(hãng xe)

private int year(năm sản xuất)

Yêu cầu:

- ✓ Viết constructor để khởi tạo brand và year.
- ✓ Viết phương thức displayInfo() để in thông tin xe ra màn hình.
- ✓ Trong main(), tạo một đối tượng Carvà hiển thị thông tin.

📌 Gợi ý:

Sử dụng thisđể phân biệt biến instance và biến tham số.

System.out.println()để hiển thị thông tin.

📌 Bài tập 2: Đóng gói (Encapsulation) (Trung bình)

Đề bài:

Viết lớp BankAccountđể mô phỏng tài khoản ngân hàng với các thuộc tính:

private String ownerName(tên chủ tài khoản)
private double balance(số dư tài khoản)

Yêu cầu:

- ✓ Viết constructor để khởi tạo tài khoản.
- ✓ Viết gettervà setterđể truy cập số dư nhưng không cho phép số dư âm.
- ✓ Viết phương thức deposit(double amount)để nạp tiền.
- ✓ Viết phương thức withdraw(double amount)để rút tiền nhưng không cho phép rút quá số dư.

📌 **Gợi ý:**

Dùng private để bảo vệ dữ liệu.

Kiểm tra amount > 0 trước khi nạp hoặc rút tiền.

📌 **Bài tập 3: static và Biến dùng chung (Khó hơn một chút)**

Đề bài:

Viết lớp Employee để quản lý số lượng nhân viên trong công ty.

Yêu cầu:

- ✓ Khai báo biến private static int count = 0; để đếm số nhân viên.
- ✓ Khi tạo một nhân viên mới, biến count tăng lên.
- ✓ Viết phương thức public static int getEmployeeCount() để lấy số lượng nhân viên hiện có.
- ✓ Trong main(), tạo một số nhân viên và hiển thị số lượng nhân viên.

📌 **Gợi ý:**

static giúp biến count được chia sẻ giữa tất cả đối tượng.

📌 **Bài tập 4: Kế thừa (Inheritance) và super(Khó)**

Đề bài:

Viết chương trình quản lý sinh viên với 2 lớp:

Lớp Person có thuộc tính namevà phương thức display().

Lớp Student kế thừa Person, thêm thuộc tính studentId.

Yêu cầu:

- ✓ Studentsử dụng super(name) để gọi constructor của Person.
- ✓ Ghi đè phương thức display() để hiển thị cả name và studentId.
- ✓ Trong main(), tạo một sinh viên và hiển thị thông tin.

📌 Gợi ý:

Dùng super() để gọi constructor của lớp cha. Ghi đè phương thức display().

📌 Bài tập 5: Tạo và sử dụng package(Rất khó)

Đề bài:

✓ Tạo một package university.

✓ Trong package university, tạo lớp Professorvới thuộc tính name và phương thức teach().
✓ Tạo lớp Mainbên ngoài package, dùng import university.Professor; để truy

cập lớp Professor.

📌 Gợi ý:

package university; ở đầu file Professor.java.

Dùng import university.Professor;trong Main.java.

📌 Tổng kết

📝 **5 bài tập này giúp sinh viên luyện tập:**

✅ **Hiểu về Đóng gói (Encapsulation)**.
✅ **Sử dụng staticđể quản lý dữ liệu dùng chung**.
✅ **Áp dụng Inheritanceđể tái sử dụng mã nguồn**.
✅ **Tổ chức mã nguồn theo packagechuyên nghiệp**.

BÀI TẬP DỰ ÁN

📌 Dự án 1: Hệ thống Quản lý Sinh viên (Dễ - Trung bình)

🎯 Mục tiêu:

✓ Thực hành **Đóng gói (Encapsulation), Kế thừa (Inheritance), staticvà ArrayList**.

✓ Sinh viên **tổ chức chương trình theo mô hình hướng đối tượng**.

✓ Luyện tập làm việc nhóm: **phân công vai trò, kiểm thử code**.

📌 Yêu cầu bài toán:

Viết chương trình quản lý danh sách sinh viên, với các lớp sau:

1 Lớp Person(Cha của Student)

📌 Thuộc tính:

private String name(Tên)

private int age(Tuổi)

📌 Phương thức:

public Person(String name, int age): Constructor

public void display(): Hiển thị thông tin

2 Lớp Student(Kế thừa từ Person)

📌 Thuộc tính:

private String studentId(Mã sinh viên)

private double gpa(Điểm trung bình)

private static int studentCount = 0;(Số lượng sinh viên)

📌 Phương thức:

public Student(String name, int age, String studentId, double gpa): Constructor

public void display(): Hiển thị thông tin sinh viên (Ghi đè display()từ Person).

public static int getStudentCount(): Trả về số lượng sinh viên hiện có.

3 Lớp StudentManager(Quản lý danh sách sinh viên)

📌 Phương thức:

public void addStudent(Student s): Thêm sinh viên vào danh sách.

public void removeStudent(String studentId): Xóa sinh viên theo mã.

public void displayAll(): Hiển thị danh sách tất cả sinh viên.

4 Lớp Main(Chạy chương trình)

Cho phép thêm, xóa, và hiển thị danh sách sinh viên bằng Scanner.

📌 **Gợi ý:**

Dùng ArrayList<Student>để quản lý danh sách sinh viên.

Sử dụng staticđể đếm số lượng sinh viên.

📌 **Phân công nhóm:**

👩💻 **Người 1**: Viết lớp Person, Student.

👨💻 **Người 2**: Viết StudentManager, Main.

📌 Dự án 2: Hệ thống Quản lý Thư viện (Khó hơn)

🎯 **Mục tiêu:**

✓ Thực hành **Encapsulation, Inheritance, Overriding, static, ArrayList, package**.
✓ Giúp sinh viên **tư duy về mô hình OOP trong ứng dụng thực tế**.

✓ Luyện tập làm việc nhóm: **lập kế hoạch, phân công vai trò, viết báo cáo ngắn**.

📌 Yêu cầu bài toán:

Viết chương trình quản lý thư viện, với các lớp sau:

1 Lớp Book(Quản lý sách)

📌 **Thuộc tính:**

private String title(Tiêu đề)

private String author(Tác giả)

private boolean isBorrowed(Trạng thái mượn)

📌 **Phương thức:**

public Book(String title, String author): Constructor public
void borrowBook(): Đánh dấu sách là "đã mượn". public
void returnBook(): Đánh dấu sách là "có sẵn". public void
displayInfo(): Hiển thị thông tin sách.

---

2 Lớp Member(Quản lý thành viên thư viện)

📌 Thuộc tính:

private String name(Tên thành viên)

private ArrayList<Book> borrowedBooks(Danh sách sách đã mượn)

📌 Phương thức:

public Member(String name): Constructor public
void borrowBook(Book b): Mượn sách. public
void returnBook(Book b): Trả sách.

---

public void displayMemberInfo(): Hiển thị thông tin thành viên và sách đã mượn.

3 Lớp LibraryManager(Quản lý thư viện)

📌 Phương thức:

public void addBook(Book b): Thêm sách vào thư viện.

public void removeBook(String title): Xóa sách khỏi thư viện.

public void displayAllBooks(): Hiển thị danh sách sách.

---

4 Lớp Main(Chạy chương trình)

📌 Chức năng:

✓ Thêm/xóa sách vào thư viện.

✓ Tạo thành viên và cho phép mượn/trả sách.

✓ Hiển thị danh sách sách và thành viên.

📌 Gợi ý:

Dùng ArrayList<Book>để quản lý sách.

Dùng ArrayList<Member>để quản lý thành viên.

Kiểm tra isBorrowed == truetrước khi mượn sách.

📌 Phân công nhóm:

👩💻 **Người 1**: Viết Book, Member.

👨💻 **Người 2**: Viết LibraryManager, Main.

📌 Tổng kết

🔷 **Dự án 1 (Quản lý Sinh viên)**: **Dễ - Trung bình**, giúp sinh viên làm quen với **OOP,**

static, ArrayList.

🔷 **Dự án 2 (Quản lý Thư viện)**: **Khó hơn**, yêu cầu sinh viên **tư duy OOP sâu hơn, sử dụng Overriding, package**.