Contents

BUỔI 7: THỰC HÀNH TÍNH ĐÓNG GÓI TRONG JAVA

I. Mục tiêu buổi học

Sau buổi học này, sinh viên có thể:

✅ Áp dụng tính đóng gói trong lập trình hướng đối tượng (OOP).
✅ Hiểu và sử dụng các mức độ truy cập (private, public, protected, default).
✅ Sử dụng **getter** và **setter** để kiểm soát truy cập dữ liệu.
✅ Viết chương trình Java có tính đóng gói để quản lý thông tin an toàn.

II. Nội dung thực hành trên lớp

🎯 Phần 1: Ôn tập & Bài tập cơ bản

📌 Ôn tập nhanh

- Sinh viên giải thích lại khái niệm **tính đóng gói** (Encapsulation).
- Phân biệt các mức truy cập: private, public, protected, default.
- Thảo luận: Tại sao cần sử dụng **getter** và **setter**?

📌 Bài tập 1: Viết lớp Student có tính đóng gói

- Viết lớp Student với các thuộc tính **private** (name, id, GPA).
- Cung cấp getter và setter cho từng thuộc tính.
- Kiểm tra tính hợp lệ: **GPA chỉ nhận giá trị từ 0.0 đến 4.0**.
- Viết chương trình main() để nhập, cập nhật và hiển thị thông tin sinh viên.

📌 Thảo luận sau bài tập

- Nếu không dùng private, dữ liệu có thể bị thay đổi ngoài mong muốn không?
- Nếu bỏ setter, sinh viên có thay đổi được thông tin không?

🎯 Phần 2: Bài tập nâng cao

📌 Bài tập 2: Quản lý tài khoản ngân hàng (BankAccount)

- Viết lớp BankAccount với các thuộc tính:
    - private String accountNumber
    - private double balance
- Viết getter cho accountNumber, balance.

- Viết setter cho balance nhưng kiểm tra:
    - **Không cho phép số dư âm**.
    - **Chỉ cho phép rút tiền nếu số dư đủ**.
- Viết chương trình main() kiểm thử.

📌 **Mở rộng bài tập**:

- Thêm phương thức deposit(amount), withdraw(amount).
- Viết menu cho phép nhập từ bàn phím.

Thảo luận:

- Nếu không dùng getter/setter, có thể thay đổi số dư trực tiếp không?

🎯 **Phần 3: Bài tập dự án nhỏ**

📌 **Bài tập 3: Quản lý nhân sự (Employee Management)**

- Viết lớp Employee với các thuộc tính **private**:
    - name, salary, id (tự động tăng).
- Viết getter cho name, salary, id.
- Viết setter cho name, salary (**lương không thể âm**).
- Dùng **static** để đếm số nhân viên.
- Viết chương trình main() tạo danh sách nhân viên.

📌 **Mở rộng bài tập**:

- Thêm lớp Department chứa danh sách nhân viên.
- Thêm phương thức tính **tổng lương phòng ban**.

Thảo luận:

- Khi nào nên sử dụng private, protected, public?

III. Bài tập về nhà

1) 20 câu hỏi trắc nghiệm (Về tính đóng gói, getter/setter, phạm vi truy cập).
2) Bài tập tự luyện

- Viết lớp Book có thuộc tính **private**: title, author, price.

- Viết lớp Car có thuộc tính **private**: brand, model, year.
- Viết lớp Laptop có thuộc tính **private**: brand, CPU, RAM.

3) Bài tập luyện tập

- Viết lớp Person có getter/setter và kiểm tra tuổi hợp lệ.
- Viết lớp Library quản lý danh sách sách bằng **ArrayList**.
- Viết lớp Store quản lý sản phẩm và tổng doanh thu.
- Viết lớp Order chứa danh sách sản phẩm và tổng tiền hóa đơn.
- Viết lớp University quản lý danh sách sinh viên.

4) Bài tập dự án: Xây dựng hệ thống Quản lý Khách Sạn

- Quản lý danh sách khách hàng bằng OOP.
- Sử dụng private, getter/setter, ArrayList.
- Hiển thị thông tin khách hàng & số phòng đã đặt.

Tóm tắt

✅ Buổi học giúp sinh viên thực hành tính đóng gói qua các bài tập từ cơ bản đến nâng cao.
✅ Cấu trúc gồm: Ôn tập, bài tập thực hành, bài tập dự án.
✅ Bài tập về nhà giúp sinh viên luyện tập thêm.

PHẦN 1: ÔN TẬP & BÀI TẬP CƠ BẢN

📌 Phần 1: Ôn tập nhanh

1. Hỏi - đáp kiến thức

- **Câu 1:** Tính đóng gói (Encapsulation) là gì?
- **Câu 2:** Lợi ích của tính đóng gói trong lập trình hướng đối tượng?
- **Câu 3:** Các mức độ truy cập trong Java là gì?
- **Câu 4:** Khi nào nên sử dụng private, public, protected, default?
- **Câu 5:** Getter và Setter dùng để làm gì?

2. Giải thích & tổng kết kiến thức

✅ Tính đóng gói (Encapsulation):

- **Định nghĩa:** Giấu thông tin của một đối tượng và chỉ cho phép truy cập thông qua các phương thức xác định.
- **Ví dụ thực tế:** Bạn có muốn ai đó thay đổi số dư tài khoản ngân hàng của bạn trực tiếp không?

✅ Mức độ truy cập trong Java:

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

✅ Getter và Setter:

- **Getter**: Lấy giá trị của biến private.
- **Setter**: Cập nhật giá trị biến private, có thể kiểm soát tính hợp lệ.
- **Lợi ích:**
    - **Bảo vệ dữ liệu** (không bị thay đổi ngoài kiểm soát).
    - **Kiểm tra tính hợp lệ** trước khi cập nhật dữ liệu.

📌 Phần 2: Bài tập 1 - Viết lớp Student có tính đóng gói

Yêu cầu bài toán

Viết lớp Student với các thuộc tính private:

- name (tên sinh viên - kiểu String)
- id (mã sinh viên - kiểu String)
- GPA (điểm trung bình - kiểu double, chỉ nhận giá trị từ 0.0 đến 4.0)

Yêu cầu kỹ thuật:

✓ Sử dụng **getter** và **setter** để kiểm soát truy cập dữ liệu.

✓ Kiểm tra tính hợp lệ khi cập nhật **GPA** (**từ 0.0 đến 4.0**).

✓ Viết chương trình main() để:

- Nhập thông tin sinh viên từ bàn phím.
- Hiển thị thông tin sinh viên.
- Cập nhật điểm GPA.

📌 **Hướng dẫn sinh viên từng bước**

1. Viết lớp Student

📌 **Gợi ý Code:**

```
class Student {
    private String name;
    private String id;
    private double GPA;

    // Constructor
    public Student(String name, String id, double GPA) {
        this.name = name;
        this.id = id;
        setGPA(GPA); // Gọi setter để kiểm tra giá trị hợp lệ}

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

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

```
    public String getId() {
        return id;}

    public double getGPA() {
        return GPA;}

    public void setGPA(double GPA) {
        if (GPA >= 0.0 && GPA <= 4.0) {
            this.GPA = GPA;
        } else {
            System.out.println("Lỗi: GPA phải từ 0.0 đến 4.0!");
        }}

    // Hiển thị thông tin sinh viên
    public void displayInfo() {
        System.out.println("Tên: " + name + ", Mã: " + id + ", GPA: " + GPA);
    }
}
```

2. Viết chương trình main() để nhập & hiển thị thông tin sinh viên

📌 **Gợi ý Code:**

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
    // Nhập thông tin sinh viên
    System.out.print("Nhập tên sinh viên: ");
    String name = scanner.nextLine();

    System.out.print("Nhập mã sinh viên: ");
    String id = scanner.nextLine();

    System.out.print("Nhập điểm GPA: ");
    double GPA = scanner.nextDouble();

    // Tạo đối tượng Student
    Student student = new Student(name, id, GPA);

    // Hiển thị thông tin sinh viên
    System.out.println("\nThông tin sinh viên:");
    student.displayInfo();

    // Cập nhật GPA
    System.out.print("\nNhập GPA mới: ");
    double newGPA = scanner.nextDouble();
    student.setGPA(newGPA);

    // Hiển thị thông tin sau khi cập nhật
    System.out.println("\nThông tin sau khi cập nhật:");
    student.displayInfo();
}}
```

📌 Phần 3: Thảo luận sau bài tập

Câu hỏi 1: Nếu không dùng private, dữ liệu có thể bị thay đổi ngoài mong muốn không?
Câu hỏi 2: Nếu bỏ setter, sinh viên có thay đổi được thông tin không?
Câu hỏi 3: Nếu GPA không có kiểm tra hợp lệ trong setter, điều gì có thể xảy ra?

🎯 Tổng kết phần 1:

✅ **Sinh viên ôn tập lại kiến thức về tính đóng gói.**
✅ **Hiểu rõ về mức độ truy cập private, public, protected, default.**
✅ **Lập trình lớp Student có tính đóng gói.**
✅ **Thực hành viết getter, setter và kiểm tra tính hợp lệ.**
✅ **Thảo luận về lợi ích của tính đóng gói.**

🎯 PHẦN **2: BÀI TẬP NÂNG CAO**

📌 Bài tập 2: Quản lý tài khoản ngân hàng (BankAccount)

Mục tiêu:

✓ Áp dụng **tính đóng gói** để bảo vệ dữ liệu tài khoản ngân hàng.
✓ Viết **getter** và **setter** để kiểm soát truy cập dữ liệu.
✓ Kiểm tra **tính hợp lệ** khi cập nhật số dư tài khoản (balance).
✓ Xây dựng chương trình quản lý tài khoản có menu nhập từ bàn phím.

📌 Phần 1: Hướng dẫn & Ôn tập nhanh

Câu hỏi kiểm tra:
1. Tại sao accountNumber và balance nên là private?
2. Khi nào nên sử dụng getter thay vì truy cập trực tiếp vào biến?
3. Tại sao cần kiểm tra số dư khi rút tiền (withdraw)?

💡 Giảng viên nhận xét + Giải thích:

- Nếu balance không có kiểm tra, người dùng có thể đặt số dư âm.
- Nếu withdraw() không kiểm tra, có thể rút nhiều hơn số tiền trong tài khoản.
- Getter giúp **xem thông tin**, Setter giúp **kiểm soát việc cập nhật dữ liệu**.

📌 Phần 2: Viết lớp BankAccount

🎯 Yêu cầu bài toán:

Viết lớp BankAccount với các thuộc tính:

✓ private String accountNumber – Số tài khoản.

✓ private double balance – Số dư tài khoản.

Yêu cầu kỹ thuật:

✓ getter cho accountNumber, balance.

✓ setter cho balance, kiểm tra:

- **Không cho phép số dư âm.**
- **Chỉ cho phép rút tiền nếu số dư đủ.**

📌 Viết lớp BankAccount

📌 Gợi ý Code:

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

    // Constructor
    public BankAccount(String accountNumber, double balance) {
        this.accountNumber = accountNumber;
        setBalance(balance); // Kiểm tra số dư ban đầu hợp lệ
    }

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

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

    // Setter với kiểm tra tính hợp lệ
    public void setBalance(double balance) {
        if (balance >= 0) {
            this.balance = balance;
        } else {
            System.out.println("Lỗi: Số dư không thể âm!");
        }
    }

    // Hiển thị thông tin tài khoản
    public void displayInfo() {
        System.out.println("Số tài khoản: " + accountNumber + ", Số dư: " + balance);
    }
}
```

📌 **Phần 3: Viết chương trình kiểm thử main()**

📌 **Gợi ý Code:**

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
        // Nhập thông tin tài khoản
        System.out.print("Nhập số tài khoản: ");
        String accountNumber = scanner.nextLine();

        System.out.print("Nhập số dư ban đầu: ");
        double balance = scanner.nextDouble();

        // Tạo đối tượng BankAccount
        BankAccount account = new BankAccount(accountNumber, balance);

        // Hiển thị thông tin tài khoản
        System.out.println("\nThông tin tài khoản:");
        account.displayInfo();

        // Cập nhật số dư (test setter)
        System.out.print("\nNhập số dư mới: ");
        double newBalance = scanner.nextDouble();
        account.setBalance(newBalance);

        // Hiển thị lại thông tin sau khi cập nhật
        System.out.println("\nThông tin sau khi cập nhật:");
        account.displayInfo();
    }
}
```

📌 Phần 4: Mở rộng bài tập

🎯 Bổ sung các phương thức:

✓ deposit(double amount): Nạp tiền vào tài khoản.

✓ withdraw(double amount): Rút tiền (chỉ khi số dư đủ).

📌 Gợi ý Code:

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

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

    public String getAccountNumber() {
        return accountNumber;
    }

    public double getBalance() {
        return balance;
    }

    public void setBalance(double balance) {
        if (balance >= 0) {
            this.balance = balance;
        } else {
            System.out.println("Lỗi: Số dư không thể âm!");
        }
```

```
    }

    // Nạp tiền vào tài khoản
    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("Lỗi: Số tiền nạp phải lớn hơn 0!");
        }
    }

    // Rút tiền từ tài khoản
    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Rút tiền thành công! Số dư còn lại: " + balance);
        } else {
            System.out.println("Lỗi: Số tiền rút không hợp lệ!");
        }
    }

    public void displayInfo() {
        System.out.println("Số tài khoản: " + accountNumber + ", Số dư: " + balance);
    }
}
```

📌 **Phần 5: Viết chương trình main() có menu**

📌 **Gợi ý Code:**

```
import java.util.Scanner;


public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);


        // Nhập thông tin tài khoản
        System.out.print("Nhập số tài khoản: ");
        String accountNumber = scanner.nextLine();


        System.out.print("Nhập số dư ban đầu: ");
        double balance = scanner.nextDouble();


        BankAccount account = new BankAccount(accountNumber, balance);


        int choice;
        do {
            System.out.println("\nMenu:");
            System.out.println("1. Xem thông tin tài khoản");
            System.out.println("2. Nạp tiền");
            System.out.println("3. Rút tiền");
            System.out.println("4. Thoát");
            System.out.print("Chọn thao tác: ");
            choice = scanner.nextInt();
```

```
            switch (choice) {
                case 1:
                    account.displayInfo();
                    break;
                case 2:
                    System.out.print("Nhập số tiền cần nạp: ");
                    double depositAmount = scanner.nextDouble();
                    account.deposit(depositAmount);
                    break;
                case 3:
                    System.out.print("Nhập số tiền cần rút: ");
                    double withdrawAmount = scanner.nextDouble();
                    account.withdraw(withdrawAmount);
                    break;
                case 4:
                    System.out.println("Thoát chương trình.");
                    break;
                default:
                    System.out.println("Lựa chọn không hợp lệ!");
            }
        } while (choice != 4);
    }
}
```

📌 Phần 6: Thảo luận sau bài tập

Câu hỏi 1: Nếu không dùng getter/setter, có thể thay đổi số dư trực tiếp không?
Câu hỏi 2: Nếu bỏ kiểm tra trong withdraw(), điều gì xảy ra?

🎯 **Tổng kết phần 2:**

✅ **Sinh viên thực hành quản lý tài khoản ngân hàng với tính đóng gói.**
✅ **Viết getter, setter, kiểm tra số dư hợp lệ.**
✅ **Xây dựng menu nhập từ bàn phím.**

🎯 PHẦN 3: BÀI TẬP DỰ ÁN NHỎ

📌 Bài tập 3: Quản lý nhân sự (Employee Management)

Mục tiêu:

✓ Áp dụng **tính đóng gói** để bảo vệ dữ liệu nhân viên.
✓ Viết **getter** và **setter** để kiểm soát truy cập dữ liệu.
✓ Dùng **static** để đếm số lượng nhân viên.
✓ Xây dựng chương trình quản lý danh sách nhân viên.

📌 Phần 1: Hướng dẫn & Ôn tập nhanh

Câu hỏi kiểm tra:
1. Tại sao name, salary, id nên là private?
2. Khi nào nên dùng **getter** thay vì truy cập trực tiếp vào biến?
3. Tại sao id nên dùng static để tự động tăng?

💡 Giảng viên nhận xét + Giải thích:

- Nếu salary không có kiểm tra, có thể đặt giá trị âm.
- Nếu không dùng **getter**, không thể đọc dữ liệu an toàn.
- **Dùng static** giúp mỗi nhân viên có **ID duy nhất**, tăng tự động.

📌 Phần 2: Viết lớp Employee

🎯 Yêu cầu bài toán

Viết lớp Employee với các thuộc tính:
✓ private static int count – Đếm số nhân viên.
✓ private int id – Mã nhân viên (tự động tăng).

✓ private String name – Tên nhân viên.

✓ private double salary – Lương nhân viên.

Yêu cầu kỹ thuật:

✓ getter cho id, name, salary.

✓ setter cho name, salary (lương không thể âm).

✓ **ID tự động tăng**, không cần nhập từ bàn phím.

📌 **Viết lớp Employee**

📌 **Gợi ý Code:**

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

    // Constructor
    public Employee(String name, double salary) {
        this.id = ++count; // Tăng ID tự động
        this.name = name;
        setSalary(salary); // Gọi setter để kiểm tra tính hợp lệ
    }

    // Getter
    public int getId() {
        return id;
    }

    public String getName() {
```

```
    return name;
}

public double getSalary() {
    return salary;
}

// Setter với kiểm tra lương hợp lệ
public void setName(String name) {
    this.name = name;
}

public void setSalary(double salary) {
    if (salary >= 0) {
        this.salary = salary;
    } else {
        System.out.println("Lỗi: Lương không thể âm!");
    }
}

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

// Getter cho tổng số nhân viên
public static int getEmployeeCount() {
```

```
        return count;
    }
}
```

📌 **Phần 3: Viết chương trình kiểm thử main()**

📌 **Gợi ý Code:**

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Nhập thông tin nhân viên
        System.out.print("Nhập tên nhân viên: ");
        String name = scanner.nextLine();

        System.out.print("Nhập lương nhân viên: ");
        double salary = scanner.nextDouble();

        // Tạo đối tượng Employee
        Employee employee = new Employee(name, salary);

        // Hiển thị thông tin nhân viên
        System.out.println("\nThông tin nhân viên:");
        employee.displayInfo();

        // Hiển thị tổng số nhân viên
```

```
        System.out.println("\nTổng số nhân viên: " + Employee.getEmployeeCount());
    }
}
```

📌 Phần 4: Mở rộng bài tập - Lớp Department

🎯 Thêm lớp Department để quản lý danh sách nhân viên

✓ private String name – Tên phòng ban.
✓ private ArrayList<Employee> employees – Danh sách nhân viên.
✓ addEmployee(Employee e): Thêm nhân viên vào phòng ban.
✓ calculateTotalSalary(): Tính tổng lương phòng ban.
✓ displayEmployees(): Hiển thị danh sách nhân viên.

📌 Gợi ý Code:

```
import java.util.ArrayList;

class Department {
    private String name;
    private ArrayList<Employee> employees;

    public Department(String name) {
        this.name = name;
        this.employees = new ArrayList<>();
    }

    public void addEmployee(Employee e) {
        employees.add(e);
    }

    public double calculateTotalSalary() {
```

```
        double total = 0;
        for (Employee e : employees) {
            total += e.getSalary();
        }
        return total;
    }

    public void displayEmployees() {
        System.out.println("Phòng ban: " + name);
        for (Employee e : employees) {
            e.displayInfo();
        }
        System.out.println("Tổng lương: " + calculateTotalSalary());
    }
}
```

📌 **Phần 5: Viết chương trình main() có danh sách nhân viên**

📌 **Gợi ý Code:**

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Department department = new Department("IT");

        int choice;
        do {
```

```
System.out.println("\nMenu:");
System.out.println("1. Thêm nhân viên");
System.out.println("2. Hiển thị danh sách nhân viên");
System.out.println("3. Thoát");
System.out.print("Chọn thao tác: ");
choice = scanner.nextInt();
scanner.nextLine(); // Xử lý ký tự xuống dòng

switch (choice) {
    case 1:
        System.out.print("Nhập tên nhân viên: ");
        String name = scanner.nextLine();

        System.out.print("Nhập lương nhân viên: ");
        double salary = scanner.nextDouble();

        Employee e = new Employee(name, salary);
        department.addEmployee(e);
        break;
    case 2:
        department.displayEmployees();
        break;
    case 3:
        System.out.println("Thoát chương trình.");
        break;
    default:
        System.out.println("Lựa chọn không hợp lệ!");
```

```
            }
        } while (choice != 3);
    }
}
```

📌 Phần 6: Thảo luận sau bài tập

Câu hỏi 1: Khi nào nên sử dụng private, protected, public?
Câu hỏi 2: Nếu không dùng static cho id, điều gì xảy ra?
Câu hỏi 3: Nếu không kiểm tra lương trong setSalary(), có thể xảy ra lỗi gì?

🎯 Tổng kết phần 3:

✅ **Sinh viên thực hành quản lý nhân sự với tính đóng gói.**
✅ **Viết getter, setter, kiểm tra lương hợp lệ.**
✅ **Dùng static để quản lý số lượng nhân viên.**
✅ **Xây dựng danh sách nhân viên có menu nhập từ bàn phím.**

PHẦN 4: CÂU HỎI TRẮC NGHIỆM

Phần 1: Tính đóng gói (Encapsulation) (Câu 1 - 5)

Câu 1: Tính đóng gói trong lập trình hướng đối tượng có tác dụng gì?
A. Giúp chia chương trình thành nhiều phần nhỏ hơn
B. Bảo vệ dữ liệu của đối tượng, chỉ cho phép truy cập qua phương thức xác định
C. Giúp tăng tốc độ thực thi chương trình
D. Giúp giảm số lượng đối tượng trong chương trình

Câu 2: Một lớp trong Java thể hiện tính đóng gói bằng cách nào?
A. Khai báo tất cả biến là public
B. Sử dụng private cho biến và getter/setter để truy cập dữ liệu
C. Sử dụng protected để chia sẻ dữ liệu giữa các lớp
D. Không cần sử dụng phạm vi truy cập

Câu 3: Điều gì có thể xảy ra nếu không sử dụng tính đóng gói trong lập trình?
A. Dữ liệu có thể bị thay đổi mà không được kiểm soát
B. Chương trình chạy nhanh hơn

C. Các lớp có thể sử dụng dữ liệu của nhau một cách trực tiếp
D. Không có bất kỳ vấn đề gì

Câu 4: Trong số các phạm vi truy cập sau, phạm vi nào giúp đóng gói dữ liệu tốt nhất?
A. public
B. private
C. protected
D. default

Câu 5: Đâu là một lợi ích chính của tính đóng gói?
A. Tăng tốc độ xử lý dữ liệu
B. Cho phép truy cập dữ liệu trực tiếp từ bất kỳ đâu
C. Giúp kiểm soát cách dữ liệu được truy cập và sửa đổi
D. Giảm số lượng dòng code cần viết

Phần 2: Getter và Setter (Câu 6 - 10)

Câu 6: Getter và setter được sử dụng để làm gì?
A. Kiểm soát quyền truy cập vào các biến private
B. Thay thế các constructor trong Java
C. Tự động tạo ra các phương thức xử lý dữ liệu
D. Gán giá trị trực tiếp vào biến private mà không cần kiểm tra

Câu 7: Phát biểu nào về setter là đúng?
A. Setter chỉ dùng để đọc giá trị biến private
B. Setter có thể kiểm tra tính hợp lệ của giá trị trước khi cập nhật biến
C. Setter luôn có phạm vi truy cập private
D. Setter không thể thay đổi giá trị của biến private

Câu 8: Trong đoạn code sau, điều gì xảy ra nếu gọi setGPA(5.0)?

```
class Student {
   private double GPA;


   public void setGPA(double GPA) {
      if (GPA >= 0.0 && GPA <= 4.0) {
         this.GPA = GPA;
      }
```

```
    }
}
```

A. Giá trị GPA được cập nhật thành 5.0
B. Giá trị GPA vẫn giữ nguyên
C. Chương trình báo lỗi biên dịch
D. Chương trình dừng ngay lập tức

Câu 9: Khi nào nên sử dụng getter thay vì truy cập trực tiếp vào biến?
A. Khi biến có phạm vi truy cập public
B. Khi cần bảo vệ dữ liệu khỏi các thay đổi ngoài mong muốn
C. Khi biến là static
D. Khi không cần kiểm soát truy cập vào biến

Câu 10: Đâu là cách viết đúng cho một getter trong Java?
A.

```
public double GPA() {
    return GPA;
}
```

B.

```
public double getGPA() {
    return GPA;
}
```

C.

```
public double setGPA() {
    return GPA;
}
```

D.

```
public getGPA(double GPA) {
    return GPA;
}
```

Phần 3: Phạm vi truy cập (Câu 11 - 15)

Câu 11: Phạm vi truy cập private cho phép truy cập từ đâu?
A. Chỉ trong lớp hiện tại
B. Trong cùng package
C. Trong lớp con
D. Từ bất kỳ đâu

Câu 12: Nếu một biến có phạm vi protected, biến này có thể được truy cập từ đâu?
A. Chỉ trong lớp hiện tại
B. Trong cùng package và các lớp con ở package khác
C. Chỉ trong các lớp con
D. Từ bất kỳ đâu

Câu 13: Mức truy cập default trong Java có nghĩa là gì?
A. Biến có thể truy cập từ mọi nơi
B. Biến chỉ có thể truy cập từ trong cùng package
C. Biến chỉ có thể truy cập từ lớp con
D. Biến chỉ có thể truy cập từ lớp hiện tại

Câu 14: Khi nào nên sử dụng public cho một biến?
A. Khi muốn biến chỉ có thể truy cập từ trong lớp hiện tại
B. Khi muốn biến có thể được truy cập từ bất kỳ đâu
C. Khi muốn biến có thể được truy cập từ trong package
D. Khi muốn giới hạn quyền truy cập vào biến

Câu 15: Điều gì xảy ra nếu một lớp khai báo biến private nhưng không có getter?
A. Biến có thể được truy cập trực tiếp từ lớp khác
B. Biến không thể được truy cập từ bên ngoài lớp
C. Biến có thể được truy cập từ package
D. Chương trình sẽ báo lỗi

Phần 4: Ứng dụng thực tế (Câu 16 - 20)

Câu 16: Nếu một biến được khai báo là private, làm thế nào để truy cập giá trị của nó?
A. Truy cập trực tiếp trong main()
B. Dùng phương thức getter
C. Biến private không thể truy cập được từ bất kỳ đâu
D. Biến private chỉ có thể được truy cập nếu khai báo static

Câu 17: Điều gì xảy ra nếu setter không kiểm tra dữ liệu trước khi cập nhật biến?
A. Dữ liệu có thể bị thay đổi thành giá trị không hợp lệ

B. Chương trình sẽ báo lỗi biên dịch
C. Không có gì xảy ra
D. Biến sẽ tự động sửa lỗi dữ liệu

Câu 18: Trong một hệ thống quản lý tài khoản ngân hàng, biến balance (số dư tài khoản) nên có phạm vi truy cập nào?
A. public
B. private
C. protected
D. default

Câu 19: Đâu là một ví dụ thực tế của tính đóng gói?
A. Cho phép mọi người thay đổi điểm số của một sinh viên trong hệ thống
B. Bảo vệ thông tin tài khoản ngân hàng khỏi bị truy cập trực tiếp
C. Chia sẻ dữ liệu giữa các lớp con mà không cần phương thức
D. Không cần bảo vệ dữ liệu, chỉ cần đảm bảo chương trình chạy nhanh

Câu 20: Khi nào nên sử dụng cả getter và setter cho một biến?
A. Khi biến cần được truy cập từ bên ngoài nhưng cần kiểm soát việc cập nhật giá trị
B. Khi biến chỉ cần đọc mà không bao giờ thay đổi
C. Khi biến chỉ cần cập nhật nhưng không cần đọc
D. Khi biến không cần được bảo vệ

PHẦN 5: BÀI TẬP TỰ LUYỆN

Mục tiêu

✓ Thực hành **tính đóng gói (Encapsulation)** trong Java.
✓ Sử dụng **getter** để lấy dữ liệu và **setter** để cập nhật dữ liệu an toàn.
✓ Áp dụng **constructor** để khởi tạo đối tượng.
✓ Viết chương trình kiểm thử cho từng lớp.

📌 Bài 1: Viết lớp Book

Yêu cầu bài toán

Tạo một lớp Book có các thuộc tính:

- **title** (Tên sách - kiểu String)
- **author** (Tác giả - kiểu String)
- **price** (Giá sách - kiểu double, không được âm)

Yêu cầu kỹ thuật:
✓ Đặt tất cả thuộc tính ở chế độ **private**.
✓ Viết **constructor** để khởi tạo thông tin sách.
✓ Cung cấp **getter** cho tất cả các thuộc tính.
✓ Cung cấp **setter** cho price, nhưng kiểm tra rằng giá trị không được âm.
✓ Viết phương thức **displayInfo()** để hiển thị thông tin sách.
✓ Trong main(), tạo một đối tượng Book, hiển thị thông tin và kiểm tra cập nhật giá.

💡 **Gợi ý:**

- Khi nhập giá sách, hãy kiểm tra nếu giá nhỏ hơn 0 thì thông báo lỗi.
- Nếu một cuốn sách có giá -100, setter phải ngăn chặn giá trị này.

📌 Bài 2: Viết lớp Car

Yêu cầu bài toán

Tạo một lớp Car có các thuộc tính:

- **brand** (Hãng xe - kiểu String)
- **model** (Mẫu xe - kiểu String)
- **year** (Năm sản xuất - kiểu int, phải lớn hơn 1885)

Yêu cầu kỹ thuật:
✓ Đặt tất cả thuộc tính ở chế độ **private**.
✓ Viết **constructor** để khởi tạo thông tin xe.
✓ Cung cấp **getter** cho tất cả các thuộc tính.
✓ Cung cấp **setter** cho year, nhưng chỉ chấp nhận giá trị lớn hơn 1885.
✓ Viết phương thức **displayInfo()** để hiển thị thông tin xe.
✓ Trong main(), tạo một đối tượng Car, hiển thị thông tin và kiểm tra cập nhật năm sản xuất.

💡 **Gợi ý:**

- Nếu ai đó nhập năm 1800, setter phải hiển thị lỗi và không cập nhật giá trị.
- Khi hiển thị thông tin xe, hãy in ra theo định dạng: "Hãng: Toyota, Mẫu: Camry, Năm SX: 2022"

📌 Bài 3: Viết lớp Laptop

Yêu cầu bài toán

Tạo một lớp Laptop có các thuộc tính:

- **brand** (Hãng sản xuất - kiểu String)
- **CPU** (Loại CPU - kiểu String)
- **RAM** (Dung lượng RAM - kiểu int, phải lớn hơn 0 GB)

Yêu cầu kỹ thuật:
✓ Đặt tất cả thuộc tính ở chế độ **private**.
✓ Viết **constructor** để khởi tạo thông tin laptop.
✓ Cung cấp **getter** cho tất cả các thuộc tính.
✓ Cung cấp **setter** cho RAM, nhưng chỉ chấp nhận giá trị lớn hơn 0 GB.
✓ Viết phương thức **displayInfo()** để hiển thị thông tin laptop.
✓ Trong main(), tạo một đối tượng Laptop, hiển thị thông tin và kiểm tra cập nhật RAM.

💡 **Gợi ý:**

- Nếu nhập RAM = 0, setter phải báo lỗi và không cập nhật.
- Khi hiển thị thông tin laptop, hãy đảm bảo in ra đúng định dạng: "Hãng: Dell, CPU: Intel i7, RAM: 16GB"

📌 Hướng dẫn thực hiện

1. **Viết lớp**:

- Khai báo **biến private** để bảo vệ dữ liệu.
- Viết **constructor** để khởi tạo đối tượng.
- Viết **getter** để lấy giá trị.
- Viết **setter** để cập nhật giá trị với kiểm tra hợp lệ.

2. **Viết displayInfo()**:

- In ra thông tin đối tượng theo định dạng dễ đọc.

3. **Viết main() để kiểm thử**:

- Tạo một đối tượng từ lớp vừa viết.

- Hiển thị thông tin ban đầu.
- Thử cập nhật dữ liệu hợp lệ và không hợp lệ.
- Hiển thị lại thông tin sau khi cập nhật.

🎯 Tổng kết bài tập tự luyện

✅ **Sinh viên thực hành tính đóng gói với 3 lớp đơn giản.**
✅ **Viết getter, setter để bảo vệ dữ liệu.**
✅ **Kiểm tra tính hợp lệ khi cập nhật dữ liệu.**
✅ **Viết chương trình kiểm thử để nhập, hiển thị thông tin.**

PHẦN 6: BÀI TẬP LUYỆN TẬP

Mục tiêu

✓ Luyện tập **tính đóng gói (Encapsulation)** trong Java.
✓ Sử dụng **getter** và **setter** để kiểm soát dữ liệu.
✓ Áp dụng **ArrayList** để quản lý danh sách đối tượng.
✓ Viết chương trình kiểm thử từng bài tập.

📌 Bài 1: Viết lớp Person

Yêu cầu bài toán

Tạo một lớp Person có các thuộc tính:

- **name** (Tên người - kiểu String)
- **age** (Tuổi - kiểu int, phải lớn hơn 0)

Yêu cầu kỹ thuật:
✓ Đặt tất cả thuộc tính ở chế độ **private**.
✓ Viết **constructor** để khởi tạo thông tin.
✓ Cung cấp **getter** cho name và age.
✓ Cung cấp **setter** cho age, nhưng kiểm tra tuổi không thể nhỏ hơn 0.
✓ Viết phương thức **displayInfo()** để hiển thị thông tin.

💡 **Gợi ý:**

- Nếu ai đó nhập age = -5, setter phải hiển thị lỗi và không cập nhật giá trị.
- Khi hiển thị thông tin, hãy đảm bảo in ra đúng định dạng: "Tên: An, Tuổi: 20"

📌 **Bài 2: Viết lớp Library**

Yêu cầu bài toán

Tạo một lớp Library để quản lý danh sách sách. Mỗi sách có các thuộc tính:

- **title** (Tên sách - kiểu String)
- **author** (Tác giả - kiểu String)
- **price** (Giá sách - kiểu double)

Yêu cầu kỹ thuật:

✓ Sử dụng **ArrayList** để lưu danh sách sách.
✓ Viết phương thức **addBook(Book book)** để thêm sách vào thư viện.
✓ Viết phương thức **displayBooks()** để hiển thị danh sách sách.
✓ Viết phương thức **findBook(String title)** để tìm sách theo tên.
✓ Trong main(), thêm một số sách, hiển thị danh sách và thử tìm kiếm sách.

💡 **Gợi ý:**

- Khi tìm sách, nếu không có, hãy in thông báo "Không tìm thấy sách!"
- Khi hiển thị danh sách sách, hãy in theo thứ tự: "Tên sách: XYZ, Tác giả: ABC, Giá: 200.0"

📌 **Bài 3: Viết lớp Store**

Yêu cầu bài toán

Tạo một lớp Store để quản lý danh sách sản phẩm. Mỗi sản phẩm có các thuộc tính:

- **name** (Tên sản phẩm - kiểu String)
- **price** (Giá sản phẩm - kiểu double, không được âm)
- **quantity** (Số lượng - kiểu int, không được âm)

Yêu cầu kỹ thuật:

✓ Sử dụng **ArrayList** để lưu danh sách sản phẩm.
✓ Viết phương thức **addProduct(Product product)** để thêm sản phẩm.

✓ Viết phương thức **displayProducts()** để hiển thị danh sách sản phẩm.
✓ Viết phương thức **calculateTotalRevenue()** để tính tổng doanh thu (price * quantity).
✓ Trong main(), thêm một số sản phẩm, hiển thị danh sách và kiểm tra tổng doanh thu.

💡 **Gợi ý:**

- Khi tính doanh thu, chỉ tính các sản phẩm có số lượng > 0.
- Khi hiển thị danh sách sản phẩm, hãy in ra theo thứ tự: "Tên: XYZ, Giá: 100.0, Số lượng: 10"

📌 Bài 4: Viết lớp Order

Yêu cầu bài toán

Tạo một lớp Order để quản lý danh sách sản phẩm trong đơn hàng.

- Mỗi đơn hàng có một **danh sách sản phẩm** (ArrayList).
- Tổng tiền đơn hàng = tổng price * quantity của tất cả sản phẩm.

Yêu cầu kỹ thuật:
✓ Sử dụng **ArrayList** để lưu danh sách sản phẩm.
✓ Viết phương thức **addProduct(Product product)** để thêm sản phẩm vào đơn hàng.
✓ Viết phương thức **calculateTotalPrice()** để tính tổng tiền hóa đơn.
✓ Viết phương thức **displayOrder()** để hiển thị danh sách sản phẩm trong đơn hàng.
✓ Trong main(), tạo một đơn hàng, thêm sản phẩm vào đơn và hiển thị thông tin.

💡 **Gợi ý:**

- Nếu đơn hàng không có sản phẩm, tổng tiền phải là 0.0.
- Khi hiển thị đơn hàng, in tổng số tiền cuối cùng.

📌 Bài 5: Viết lớp University

Yêu cầu bài toán

Tạo một lớp University để quản lý danh sách sinh viên.

- Mỗi sinh viên có các thuộc tính:
    - **name** (Tên sinh viên - kiểu String)

- **id** (Mã sinh viên - kiểu String)
- **GPA** (Điểm trung bình - kiểu double, từ 0.0 đến 4.0)

Yêu cầu kỹ thuật:
✓ Sử dụng **ArrayList** để lưu danh sách sinh viên.
✓ Viết phương thức **addStudent(Student student)** để thêm sinh viên.
✓ Viết phương thức **displayStudents()** để hiển thị danh sách sinh viên.
✓ Viết phương thức **findStudent(String id)** để tìm sinh viên theo mã.
✓ Trong main(), thêm sinh viên, hiển thị danh sách và thử tìm kiếm sinh viên.

💡 **Gợi ý:**

- Nếu ai đó nhập GPA = 5.0, setter phải hiển thị lỗi và không cập nhật.
- Khi hiển thị danh sách sinh viên, đảm bảo định dạng "Tên: A, Mã: 123, GPA: 3.5"

📌 **Hướng dẫn thực hiện**

1. **Viết lớp**:
   - Khai báo **biến private** để bảo vệ dữ liệu.
   - Viết **constructor** để khởi tạo đối tượng.
   - Viết **getter** để lấy giá trị.
   - Viết **setter** để cập nhật giá trị với kiểm tra hợp lệ.
2. **Viết các phương thức quản lý danh sách**:
   - **Thêm đối tượng** vào danh sách (ArrayList).
   - **Hiển thị danh sách** theo định dạng dễ đọc.
   - **Tính toán giá trị** (doanh thu, tổng tiền hóa đơn, điểm GPA).
3. **Viết main() để kiểm thử**:
   - Tạo một đối tượng từ lớp vừa viết.
   - Thêm dữ liệu vào danh sách.
   - Hiển thị danh sách để kiểm tra dữ liệu.
   - Thử tìm kiếm đối tượng trong danh sách.

🎯 **Tổng kết bài tập luyện tập**

✅ **Sinh viên thực hành với 5 bài từ dễ đến khó.**
✅ **Áp dụng getter, setter và kiểm tra dữ liệu hợp lệ.**
✅ **Luyện tập làm việc với danh sách (ArrayList).**
✅ **Viết chương trình kiểm thử để nhập, hiển thị, tính toán dữ liệu.**

PHẦN 7: BÀI TẬP DỰ ÁN: QUẢN LÝ VIỆN PHÍ BỆNH NHÂN

Yêu cầu bài toán

Bệnh viện ABC cần xây dựng một hệ thống quản lý viện phí bệnh nhân, trong đó:

- Có hai loại bệnh nhân: **Bệnh nhân ngoại trú** và **Bệnh nhân nội trú**.
- Dữ liệu bệnh nhân cần được **đóng gói** để bảo vệ thông tin.
- Sử dụng **getter** và **setter** để kiểm soát dữ liệu.
- Sử dụng danh sách bệnh nhân bằng **ArrayList**.

I. Mô tả lớp & yêu cầu lập trình

1. Lớp BenhNhan (Bệnh nhân - Lớp cha)

- **Thuộc tính** (đóng gói - private):
    - ma (Mã bệnh nhân - kiểu int).
    - hoTen (Họ tên bệnh nhân - kiểu String).
    - tienThuoc (Tiền thuốc - kiểu double).
- **Constructor:** Khởi tạo mã, họ tên, tiền thuốc.
- **Phương thức:**
    - getter/setter cho các thuộc tính.
    - hienThiThongTin(): Hiển thị thông tin bệnh nhân.

2. Lớp BNNgoaiTru (Bệnh nhân ngoại trú - Kế thừa BenhNhan)

- **Thuộc tính riêng:**
    - phiKham (Phí khám bệnh - kiểu double).
    - phiXetNghiem (Phí xét nghiệm - kiểu double).

- **Constructor:** Khởi tạo đầy đủ thông tin (gồm thông tin từ lớp BenhNhan).
- **Phương thức:**
    - getter/setter cho các thuộc tính riêng.
    - vienPhi(): Tính tổng viện phí = tienThuoc + phiKham + phiXetNghiem.

3. Lớp BNNoiTru (Bệnh nhân nội trú - Kế thừa BenhNhan)

- **Thuộc tính riêng:**
    - phiNgay (Phí nằm viện theo ngày - kiểu double).
    - soNgayNamVien (Số ngày nằm viện - kiểu int).
- **Constructor:** Khởi tạo đầy đủ thông tin (gồm thông tin từ lớp BenhNhan).
- **Phương thức:**
    - getter/setter cho các thuộc tính riêng.
    - vienPhi():
        - vienPhi = (tienThuoc * soNgayNamVien) + (phiNgay * soNgayNamVien) + phuPhi.
        - phuPhi = 50 nếu soNgayNamVien < 10, ngược lại phuPhi = 100.

4. **Lớp QuanLyBenhNhan (Quản lý danh sách bệnh nhân)**

- **Danh sách bệnh nhân**: Sử dụng **ArrayList** để lưu danh sách bệnh nhân.
- **Phương thức**:
    - themBenhNhan(BenhNhan bn): Thêm bệnh nhân vào danh sách.
    - hienThiDanhSach(): In danh sách bệnh nhân.
    - timBenhNhanTheoMa(int ma): Tìm bệnh nhân theo mã.
    - timBenhNhanNoiTruPhiCao(double nguongPhi): Hiển thị bệnh nhân nội trú có viện phí cao hơn nguongPhi.

II. Chương trình chính (main())

Viết một chương trình Java với menu cho phép:
1. Nhập thông tin bệnh nhân (ngoại trú hoặc nội trú).
2. Hiển thị danh sách tất cả bệnh nhân.

3. Tìm bệnh nhân theo mã.
4. Tìm bệnh nhân nội trú có viện phí >= 3000.

III. Yêu cầu lập trình

✅ **Sử dụng tính đóng gói** (private, getter, setter).
✅ **Sử dụng kế thừa** (extends).
✅ **Sử dụng danh sách (ArrayList)** để lưu bệnh nhân.
✅ **Viết chương trình có menu để nhập dữ liệu từ bàn phím.**

💡 **Ghi chú:**
📌 Đề bài này được điều chỉnh để không dùng **interface** và **override phương thức** từ interface (chưa học).
📌 Các khái niệm sử dụng chỉ giới hạn trong **buổi 1 - 7** (Lớp, Đối tượng, Đóng gói, Danh sách, Kế thừa).

QUẢN LÝ CODE OOP BẰNG VS CODE VÀ ECLIPSE

1. Quản lý code OOP bằng VS Code

Bước 1: Cài đặt môi trường

Trước khi bắt đầu, hãy đảm bảo bạn đã cài đặt:

1. **JDK (Java Development Kit)** (Nên dùng **JDK 17 hoặc 21**).
2. **Visual Studio Code** (VS Code).
3. **Các extension cần thiết**:
    - **Extension Pack for Java** (bao gồm Java Language Support, Debugger, Test Runner, etc.)

👉 **Mở VS Code**, vào **Extensions (Ctrl + Shift + X)**, tìm và cài đặt Extension Pack for Java.

Bước 2: Tạo Workspace trong VS Code

1. **Mở VS Code**.
2. **Tạo một thư mục mới** trên máy tính để làm workspace, ví dụ:

makefile

CopyEdit

D:\JavaWorkspace

3. **Mở thư mục này trong VS Code** bằng cách:
    - Chọn **File → Open Folder...**
    - Chọn thư mục JavaWorkspace.
4. **Lưu workspace**:
    - Vào **File → Save Workspace As...**
    - Đặt tên, ví dụ JavaWorkspace.code-workspace, rồi lưu vào thư mục D:\JavaWorkspace.

Bước 3: Tạo Project Java

1. **Mở Terminal trong VS Code**:
    - Chọn **View** → **Terminal** (hoặc nhấn Ctrl + ~).
2. **Chạy lệnh để tạo project Java mới**:

mkdir JavaCourse

cd JavaCourse

3. **Khởi tạo project Java** bằng lệnh:

javac --version # Kiểm tra Java đã cài chưa

Nếu Java đã cài, tiếp tục:

code .

- Nếu bạn muốn sử dụng Maven hoặc Gradle, có thể dùng:

mvn archetype:generate

hoặc

gradle init

Bước 4: Quản lý Code bằng Package là Các Buổi Học

1. **Trong VS Code**, mở thư mục JavaCourse, tạo cấu trúc thư mục như sau:

JavaCourse/

├── src/

│ ├── Buoi1/

│ │ ├── Main.java

│ │ ├── Bai1.java

│ │ ├── Bai2.java

│ ├── Buoi2/

│ │ ├── Main.java

│ │ ├── Bai1.java

│ ├── Buoi3/

│ │ ├── ...

├── .vscode/

├── JavaCourse.code-workspace

├── README.md

├── .gitignore

2. **Tạo package đại diện cho từng buổi học**:
    - Trong thư mục src, tạo các package: Buoi1, Buoi2, Buoi3,...
    - Ví dụ **Buổi 1**, tạo file Main.java trong Buoi1 với nội dung:

```
package Buoi1;


public class Main {
    public static void main(String[] args) {
        System.out.println("Chào mừng đến với Buổi 1!");
    }
}
```

    - **Chạy chương trình**:

```
cd src
javac Buoi1/Main.java
java Buoi1.Main
```

Bước 5: Cấu hình Debug và Chạy Code

1. **Mở file .vscode/launch.json** (nếu chưa có, vào **Run → Add Configuration...**).
2. Thêm cấu hình để chạy Java:

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "java",
            "request": "launch",
            "name": "Debug Buoi1",
```

```
            "mainClass": "Buoi1.Main",
            "projectName": "JavaCourse"
        }
    ]
}
```

3. **Chạy chương trình**:
    - Vào **Run** → **Start Debugging** (F5).
    - Chọn cấu hình **Debug Buoi1**.

Tóm tắt

✅ **Tạo workspace** trong VS Code.
✅ **Tạo project Java** với cấu trúc thư mục rõ ràng.
✅ **Sử dụng package** để quản lý từng buổi học.
✅ **Cấu hình VS Code** để biên dịch và chạy code hiệu quả.

2.Quản lý code OOP bằng Eclipse

Hướng dẫn chi tiết để bạn **tạo workspace trong Eclipse**, **tạo project Java**, và **quản lý code bằng các package đại diện cho các buổi học**.

Bước 1: Cài đặt Eclipse và JDK

Trước khi bắt đầu, hãy đảm bảo bạn đã cài đặt:

1. **JDK (Java Development Kit)** (**JDK 17 hoặc 21**).
2. **Eclipse IDE for Java Developers** (Tải từ eclipse.org).

👉 Sau khi cài đặt, mở **Eclipse IDE**.

Bước 2: Tạo Workspace trong Eclipse

1. **Mở Eclipse**, cửa sổ chọn workspace sẽ xuất hiện.
2. **Chọn thư mục để làm workspace**, ví dụ:

D:\JavaWorkspace

3. Nhấn **Launch** để mở Eclipse với workspace này.

Bước 3: Tạo Java Project

1. Vào **File** → **New** → **Java Project**.
2. Đặt tên project, ví dụ: JavaCourse.
3. **JDK Compiler Level**: Chọn phiên bản Java phù hợp (17 hoặc 21).
4. Nhấn **Finish**.

🔷 Eclipse sẽ tạo một thư mục **src/** trong project để chứa mã nguồn.

Bước 4: Tạo Package cho Mỗi Buổi Học

Tạo Package

1. Nhấp chuột phải vào thư mục src của project.
2. Chọn **New** → **Package**.
3. Đặt tên package theo từng buổi học, ví dụ:
    - Buoi1
    - Buoi2
    - Buoi3
    - ...
4. Nhấn **Finish**.

Tạo File Java trong Mỗi Package

1. Nhấp chuột phải vào package **Buoi1** → **New** → **Class**.
2. Đặt tên class là Main.
3. Chọn **public static void main(String[] args)** để tạo hàm main().
4. Nhấn **Finish**.

Ví dụ mã nguồn Buoi1/Main.java:

```
package Buoi1;


public class Main {
```

```
    public static void main(String[] args) {
        System.out.println("Chào mừng đến với Buổi 1!");
    }
}
```

Bước 5: Chạy Chương Trình trong Eclipse

1. Mở file Main.java trong **Package Explorer**.
2. Nhấn **Run** (Ctrl + F11) hoặc chuột phải → **Run As** → **Java Application**.
3. Kết quả sẽ hiển thị trong **Console** của Eclipse.

Bước 6: Cấu hình Debug

1. Mở file Main.java, đặt breakpoint bằng cách nhấn đúp chuột vào lề trái của dòng System.out.println(...).
2. Nhấn **F11** (hoặc vào **Run** → **Debug As** → **Java Application**).
3. Eclipse sẽ dừng tại breakpoint, bạn có thể kiểm tra giá trị biến.

Tóm tắt

✅ **Tạo workspace** trong Eclipse.
✅ **Tạo project Java**.
✅ **Quản lý từng buổi học bằng package**.
✅ **Chạy và debug chương trình** trong Eclipse.