A blue logo on a black background

Description automatically generated with low confidence

*LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG*

|  |  |
| --- | --- |
| **Program Code** |  |
| **Issue/Revision** | **x/y** |
| **Effective date** | **14/10 /2024** |

**Final Exam**

**Thời gian thi:150 phút**

Họ và tên: Trần Quang Huy

Mã lớp:

KCG-03

**Lý Thuyết**

**Bài 1**

****

**Đóng gói (Encapsulation)**: Là cơ chế gói dữ liệu và các phương thức thao tác trên dữ liệu lại với nhau trong một đối tượng. Điều này giúp che giấu các chi tiết bên trong của đối tượng khỏi bên ngoài, chỉ cung cấp các phương thức cần thiết để tương tác. Ví dụ, các thuộc tính của một lớp có thể được đặt là private, chỉ có thể được truy cập qua các phương thức getter và setter.

**Kế thừa (Inheritance):** Là khả năng tạo ra một lớp mới dựa trên lớp đã có, giúp tái sử dụng mã nguồn và giảm thiểu việc viết lại mã. Lớp con (subclass) thừa hưởng thuộc tính và phương thức từ lớp cha (superclass) và có thể mở rộng hoặc ghi đè chúng.

**Đa hình (Polymorphism):** Là khả năng sử dụng các đối tượng có kiểu lớp khác nhau qua cùng một interface. Đa hình cho phép một đối tượng có thể được biểu diễn theo nhiều cách khác nhau. Ví dụ, phương thức cùng tên nhưng có thể thực hiện những hành động khác nhau khi áp dụng cho các đối tượng thuộc các lớp khác nhau.

**Trừu tượng (Abstraction):** Là khả năng ẩn đi các chi tiết phức tạp và chỉ hiện ra những phần cần thiết. Lớp trừu tượng và giao diện (interface) là những cách để thực hiện tính trừu tượng trong OOP.

****

**Bài 2**

**Overloading (Nạp chồng)**: Là khi hai hoặc nhiều phương thức có cùng tên nhưng khác nhau về tham số (số lượng hoặc kiểu dữ liệu) trong cùng một lớp. Ví dụ, bạn có thể có phương thức add(int a, int b) và add(double a, double b).

**Overriding (Ghi đè)**: Là khi lớp con định nghĩa lại một phương thức đã được định nghĩa trong lớp cha. Phương thức ghi đè phải có cùng tên, cùng kiểu trả về và cùng tham số với phương thức trong lớp cha. Overriding giúp cung cấp một phiên bản cụ thể của phương thức cho lớp con.

**VD**

class Animal {

void sound() {

System.out.println("Animal makes a sound");

}

}

class Dog extends Animal {

@Override

void sound() {

System.out.println("Dog barks");

}

}

**Bài 3**

**Đóng gói** giúp bảo vệ trạng thái bên trong của một đối tượng khỏi sự can thiệp và sửa đổi từ bên ngoài, đảm bảo sự toàn vẹn của dữ liệu. Nó chỉ cho phép truy cập đến dữ liệu thông qua các phương thức công khai, điều này giúp kiểm soát và bảo vệ các giá trị của thuộc tính một cách an toàn.

class Person {

private String name;

private int age;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

if (age > 0) {

this.age = age;

}

}

}

**Bài 4**

**Interface**: Là một tập hợp các phương thức trừu tượng không có phần thực thi. Một lớp có thể triển khai nhiều interface, điều này giúp Java hỗ trợ tính đa kế thừa. Interface chỉ định rõ những gì mà lớp phải thực hiện nhưng không cung cấp cách làm điều đó.

**Abstract class**: Là một lớp có thể có cả phương thức trừu tượng (không có phần thực thi) và phương thức đã được thực thi. Lớp con thừa kế từ abstract class phải triển khai các phương thức trừu tượng.

**Khác biệt**:

Abstract class có thể có thuộc tính, còn interface chỉ có các phương thức trừu tượng.

Lớp có thể thừa kế từ một abstract class, nhưng có thể triển khai nhiều interface.

**Bài 5**

**Creational Patterns** (Mẫu thiết kế khởi tạo): Giúp tạo đối tượng một cách linh hoạt và tự động, thay vì tạo trực tiếp. Ví dụ: Factory Method, Abstract Factory, Singleton.

**Structural Patterns** (Mẫu thiết kế cấu trúc): Giúp tổ chức các lớp và đối tượng để tạo ra các cấu trúc phức tạp. Ví dụ: Adapter, Composite, Proxy.

**Behavioral Patterns** (Mẫu thiết kế hành vi): Giúp quản lý cách các đối tượng tương tác và giao tiếp với nhau. Ví dụ: Observer, Strategy, Command.

Các nhóm này giúp phân loại và tổ chức các mẫu thiết kế để giải quyết các vấn đề khác nhau trong thiết kế phần mềm.

**Bài 6**

**Factory Method**: Là mẫu thiết kế cung cấp một interface để tạo ra đối tượng nhưng cho phép lớp con quyết định đối tượng nào sẽ được khởi tạo. Điều này giúp giảm sự phụ thuộc vào các lớp cụ thể.

**Abstract Factory**: Là mẫu thiết kế cung cấp một giao diện để tạo ra một nhóm các đối tượng liên quan hoặc phụ thuộc lẫn nhau mà không cần xác định các lớp cụ thể. Nó hữu ích khi hệ thống cần làm việc với nhiều nhóm đối tượng.

**Khi nào sử dụng**:

**Factory Method**: Khi bạn chỉ cần tạo ra một đối tượng từ một lớp cụ thể và muốn để việc quyết định lớp được khởi tạo cho các lớp con.

**Abstract Factory**: Khi bạn cần tạo ra nhiều loại đối tượng liên quan và muốn tách biệt việc tạo ra các đối tượng này.