**Report Lab 04: Inheritance and Polymorphism**

# Additional requirements of AIMS

A screen shot of a computer

Description automatically generated

A screen shot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated A screen shot of a computer

Description automatically generated A black background with many colorful lines

Description automatically generated with medium confidence

# Creating the abstract **Media** classA screen shot of a computer program Description automatically generated A screenshot of a computer program Description automatically generated

# Creating the **CompactDisc** class

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated A black screen with colorful text

Description automatically generated

# Create the Playable interface

A black rectangular object with white text

Description automatically generated

# Update the **Cart** class to work with **Media**

A screen shot of a computer

Description automatically generated A screen shot of a computer program

Description automatically generated

# Update the **Store** class to work with **Media**

A screenshot of a computer

Description automatically generated A black rectangle with white text

Description automatically generated

# Constructors of whole classes and parent classes

A diagram of a computer

Description automatically generated

# Unique item in a list

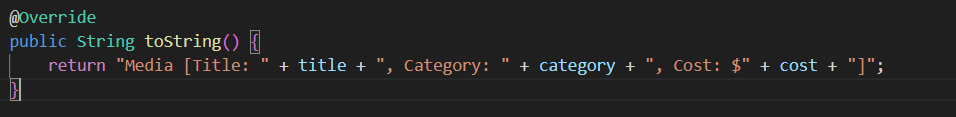
A screen shot of a computer program

Description automatically generated

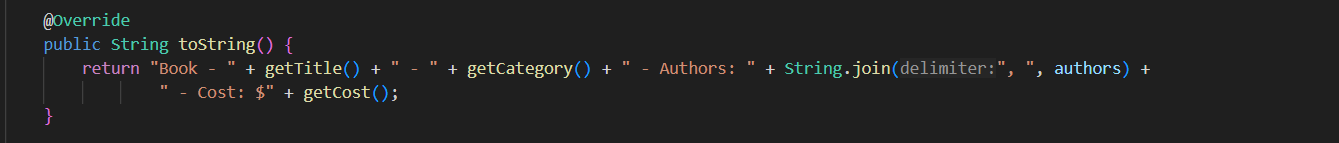
A black background with text

Description automatically generated

# Polymorphism with toString() method

 A computer screen with text

Description automatically generated



A computer screen shot of a black background

Description automatically generated

# Sort media in the cart

A computer screen with text on it

Description automatically generated

A screen shot of a computer code

Description automatically generated

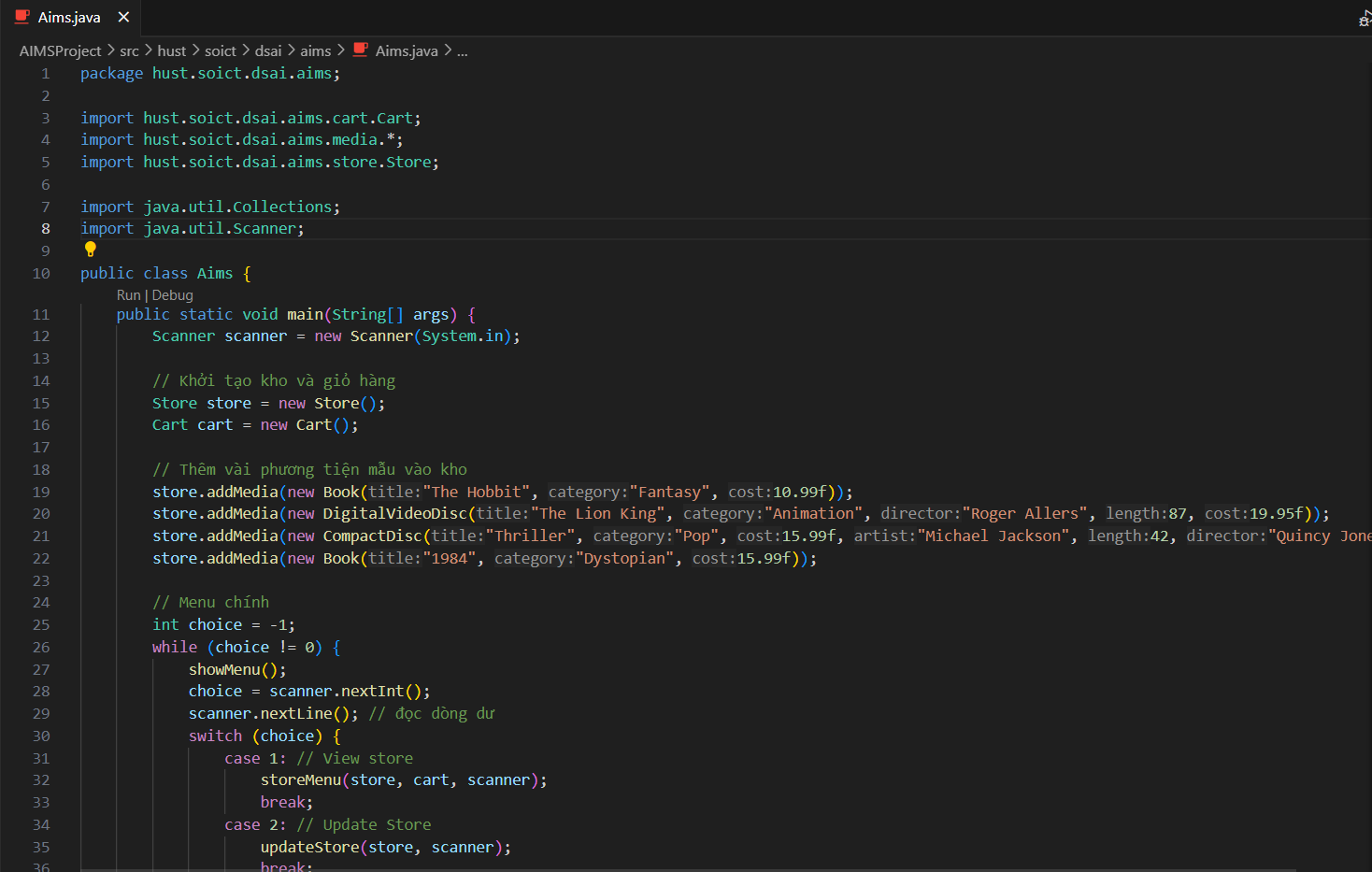
A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# Create a complete console application in the Aims class

 A screen shot of a computer program

Description automatically generated A screen shot of a computer screen

Description automatically generated A screen shot of a computer code

Description automatically generated A screen shot of a computer program

Description automatically generated A screen shot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated A screen shot of a computer program

Description automatically generated

A screen shot of a computer screen

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

# Question

1. Lớp nào nên triển khai interface Comparable?

Bất kỳ lớp nào cần được sắp xếp dựa trên các thuộc tính của nó nên triển khai interface Comparable. Ví dụ, nếu bạn có một lớp gọi là Item đại diện cho các mục trong giỏ hàng, nó nên triển khai Comparable<Item>.

public class Item implements Comparable<Item> {

private String title;

private double cost;

// Constructor, getters, và setters bị bỏ qua để tiết kiệm không gian

@Override

public int compareTo(Item other) {

// Logic so sánh sẽ được thực hiện ở đây

}

}

2. Trong những lớp đó, bạn nên triển khai phương thức compareTo() như thế nào để phản ánh thứ tự mà chúng ta muốn?

Phương thức compareTo() phải trả về một số nguyên cho biết thứ tự của đối tượng hiện tại so với một đối tượng khác:

- Một số nguyên âm nếu đối tượng này nhỏ hơn đối tượng đã chỉ định.

- Zero nếu đối tượng này bằng với đối tượng đã chỉ định.

- Một số nguyên dương nếu đối tượng này lớn hơn đối tượng đã chỉ định.

Nếu bạn muốn sắp xếp các mục trước tiên theo tiêu đề và sau đó theo chi phí, việc triển khai của bạn có thể như sau:

@Override

public int compareTo(Item other) {

int titleComparison = this.title.compareTo(other.title);

if (titleComparison != 0) {

return titleComparison; // Sắp xếp theo tiêu đề

}

return Double.compare(this.cost, other.cost); // Sắp xếp theo chi phí

}

3. Chúng ta có thể có hai quy tắc sắp xếp của mục (theo tiêu đề rồi đến chi phí và theo chi phí rồi đến tiêu đề) nếu chúng ta sử dụng cách tiếp cận interface Comparable không?

Sử dụng interface Comparable, bạn chỉ có thể định nghĩa một thứ tự tự nhiên cho mỗi lớp. Tuy nhiên, nếu bạn cần nhiều quy tắc sắp xếp (như sắp xếp theo tiêu đề rồi đến chi phí và theo chi phí rồi đến tiêu đề), bạn thường sẽ sử dụng interface Comparator thay thế.

Ví dụ:

Comparator<Item> byCostThenTitle = Comparator.comparingDouble(Item::getCost)

.thenComparing(Item::getTitle);

4. Giả sử các DVD có quy tắc sắp xếp khác với các loại phương tiện khác, đó là theo tiêu đề, sau đó là độ dài giảm dần, rồi đến chi phí. Bạn sẽ sửa đổi mã của mình như thế nào để cho phép điều này?

Nếu các DVD có quy tắc sắp xếp cụ thể (theo tiêu đề, sau đó là độ dài giảm dần, rồi đến chi phí), bạn có thể đạt được điều này bằng cách tạo một lớp riêng cho DVD mà triển khai Comparable:

public class DVD implements Comparable<DVD> {

private String title;

private double length; // Giả sử độ dài tính bằng phút

private float cost;

@Override

public int compareTo(Media other) {

if (other instanceof DigitalVideoDisc) {

DigitalVideoDisc dvd = (DigitalVideoDisc) other;

int titleComparision = this.getTitle().compareToIgnoreCase(dvd.getTitle());

if (titleComparision != 0) {

return titleComparision;

}

// so sánh độ dài

int lengthComparision = Integer.compare(dvd.getLength(), this.getLength());

if (lengthComparision != 0)

return lengthComparision;

// so sánh giá

return Float.compare(dvd.getCost(), this.getCost());

}

return 0;

}

}