**ĐẠI HỌC BÁCH KHOA HÀ NỘI**
**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

BÁO CÁO THỰC HÀNH

# BÀI THỰC HÀNH 2

Họ và tên: NGUYỄN TRÍ NAM

MSSV: 20226093

Lớp: Việt Pháp

GVHD: Đàm quang tuấn

Hà Nội 10/2024

# 1. Bài toán đặt ra

Yêu cầu: Thiết kế hệ thống mới cho dự án AIMS. (Chỉ có 1 loại phương tiện DVD)

# 2. Yêu cầu hệ thống

## 2.1. Đối với Customer:

- Duyệt danh sách các DVD có sẵn
- Tìm kiếm DVD theo: Tiêu đề, danh mục, giá cả
- Xem thông tin chi tiết của 1 DVD
- Thêm DVD vào giỏ hàng
- Xem giỏ hàng
- Sắp xếp DVD theo tiêu đề hoặc chi phí
- Cập nhật số lượng DVD trong giỏ hàng
- Đặt hàng

## 2.2. Đối với Store Manager:

- Đăng nhập, kiểm tra quyền
- Xem danh sách các đơn hàng đang được xử lý
- Xem chi tiết đơn hàng
- Thêm mới vào/ xoá ra danh sách sản phẩm

# 3. Use Case Diagram


*Figure 1: Use Case Diagram*

## 4. Class Diagram

4.1. Nội dung, các đối tượng

4.1.1. AIMS (Đối tượng chính):

Phương thức:

+ void main()

4.1.2. Cart

Thuộc tính:

- qtyOrdered: int

- MAX_NUMBER_ORDER: int

- itemOrdered: DigitalVideoDisc[]

Phương thức:

+ addDigitalVideoDisc(disc: DigitalVideoDisc): int

+ totalCost(): float

+ removeDigitalVideoDisc(disc: DigitalVideoDisc): int

4.1.3. DigitalVideoDisc

Thuộc tính:

- title: String

- category: String

- director: String

- length: int

- cost: float

Phương thức:

+ DigitalVideoMusic(title: String, category: String, director: String, length: int, cost: float): void

+ getTitle(): String

+ getCategory(): String

+ getDirector(): String

+ getLength(): int

+ getCost(): float

4.2. Hình ảnh

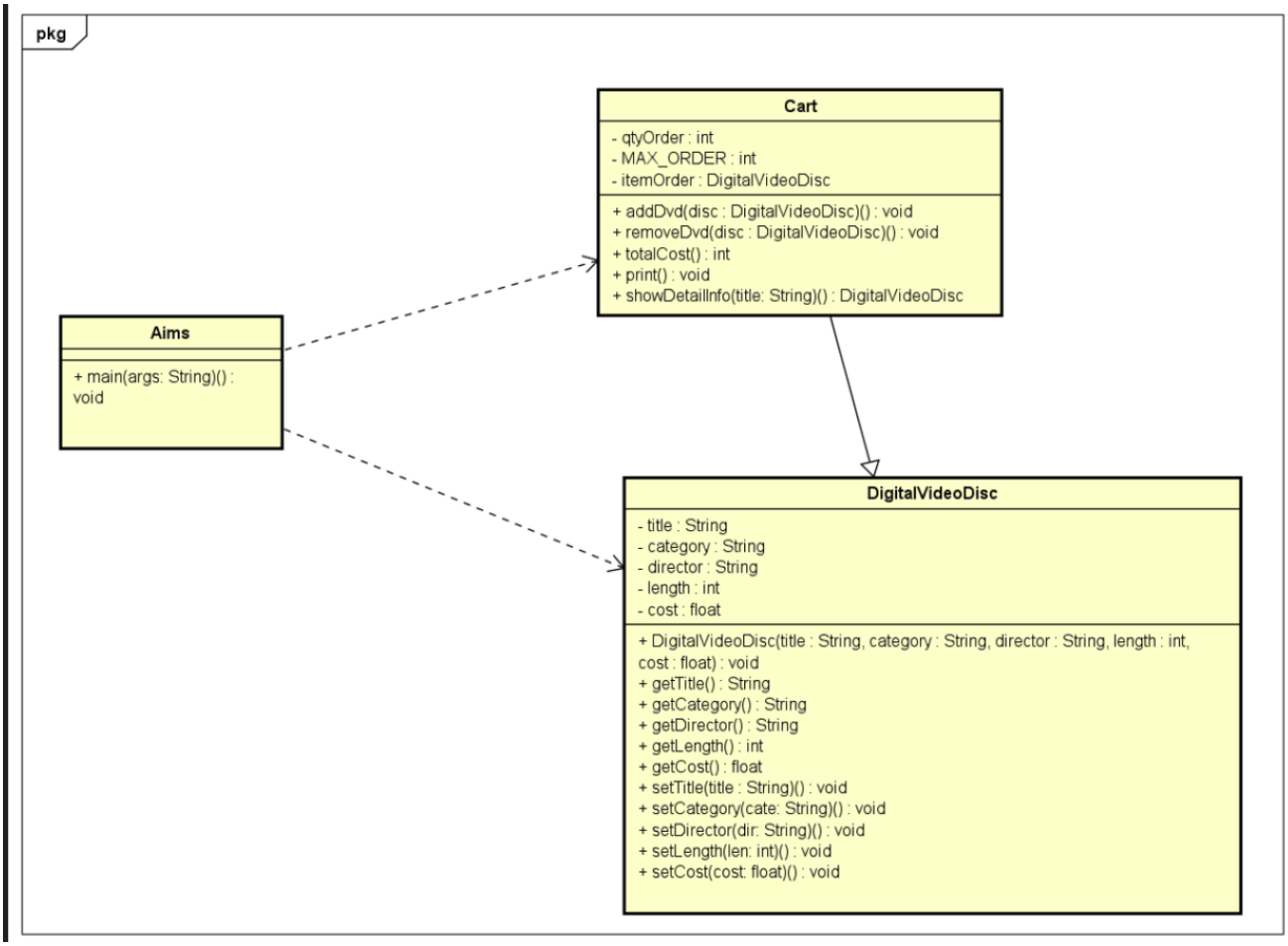



*Figure 2: Class Diagram*

# 5. Source Code

## 5.1. Aims Class

```
package Tido;
public class Aims {

	public static void main(String[] args) {
		// Create a new cart;
		Cart anOrder = new Cart();

		//Create new dvd objects and add them to the cart
		DigitalVideoDisc dvd1 = new DigitalVideoDisc("The Lion King", "Animation", "Roger Allers", 87, 19.95);
		anOrder.addDigitalVideoDisc(dvd1);

		DigitalVideoDisc dvd2 = new DigitalVideoDisc("The Lion King2", "Animation", "Roger Allers", 90, 20.95);
		anOrder.addDigitalVideoDisc(dvd2);

		// print total cost
		System.out.println("Total cost is: ");
		System.out.println(anOrder.totalCost());

	}

}
```

*Figure 3: Aims Class*

## 5.2. DigitalVideoDisc Class

```
package Tido;
public class DigitalVideoDisc {
	private String title;
	private String category;
	private String director;
	private int length;
	private float cost;
	public String getTitle() {
		return title;
	}
	public String getCategory() {
		return category;
	}
	public String getDirector() {
		return director;
	}
	public int getLength() {
		return length;
	}
	public float getCost() {
		return cost;
	}
	public DigitalVideoDisc(String title) {
		this.title = title;
	}
	public DigitalVideoDisc(String category, String title, float cost) {
		this.title = title;
		this.category = category;
		this.cost = cost;
	}
	public DigitalVideoDisc(String director, String category, String title, float cost) {
		this.title = title;
		this.category = category;
		this.cost = cost;
		this.director = director;
	}
	public DigitalVideoDisc(String title, String category, String director, int length, float cost) {
		this.title = title;
		this.category = category;
		this.cost = cost;
		this.director = director;
		this.length = length;
	}
}
```

## 5.3. Cart Class

```
package Tido;

public class Cart {
        public static final int MAX_NUMBERS_ORDERED = 20;
        private DigitalVideoDisc itemOrdered[] = new DigitalVideoDisc[MAX_NUMBERS_ORDERED];

private int qtyOrdered = 0;

public void addDigitalVideoDisc(DigitalVideoDisc disc) {
        if(qtyOrdered < 20) {
                itemOrdered[qtyOrdered] = disc;
                qtyOrdered++;
                System.out.println("Disc has been added");
        }
        else {
                System.out.println("Cart is full");
        }
}

public void removeDigitalVideoDisc(DigitalVideoDisc disc) {
        for(int i = 0; i < qtyOrdered; i++) {
                if(itemOrdered[i].getTitle() == disc.getTitle()) {
                        for(int j = i; j<qtyOrdered; j++) {
                                itemOrdered[j] = itemOrdered[j+1];
                        }
                        qtyOrdered--;
                        System.out.println("Disc has been removed");
                        break;
                }
        }
}

public float totalCost() {
        float total = 0;
        for(int i = 0; i < qtyOrdered; i++) {
                total+= itemOrdered[i].getCost();
        }
        return total;
}
}
```

## 6. Reading Assignment: When should accessor methods be used?

*Figure 4: Getter and Setter Methods*

Read the following article and find the best possible answer to the above question: Holub, Allen. “Why getter and setter methods are evil” JavaWorld, 5 Sep. 2003, https://www.infoworld.com/article/2073723/why-getter-and-setter-methods-are-evil.html

You should expand your research to other sources as well. For the response, give a summary of your findings in the form of a mindmap. You can draw this mind map by hand and take a picture of your work or use any online tools. In both cases, the accepted format for the image file is one of the following: .png, .jpg, .jpeg and .pdf.

## 7. Answer the question

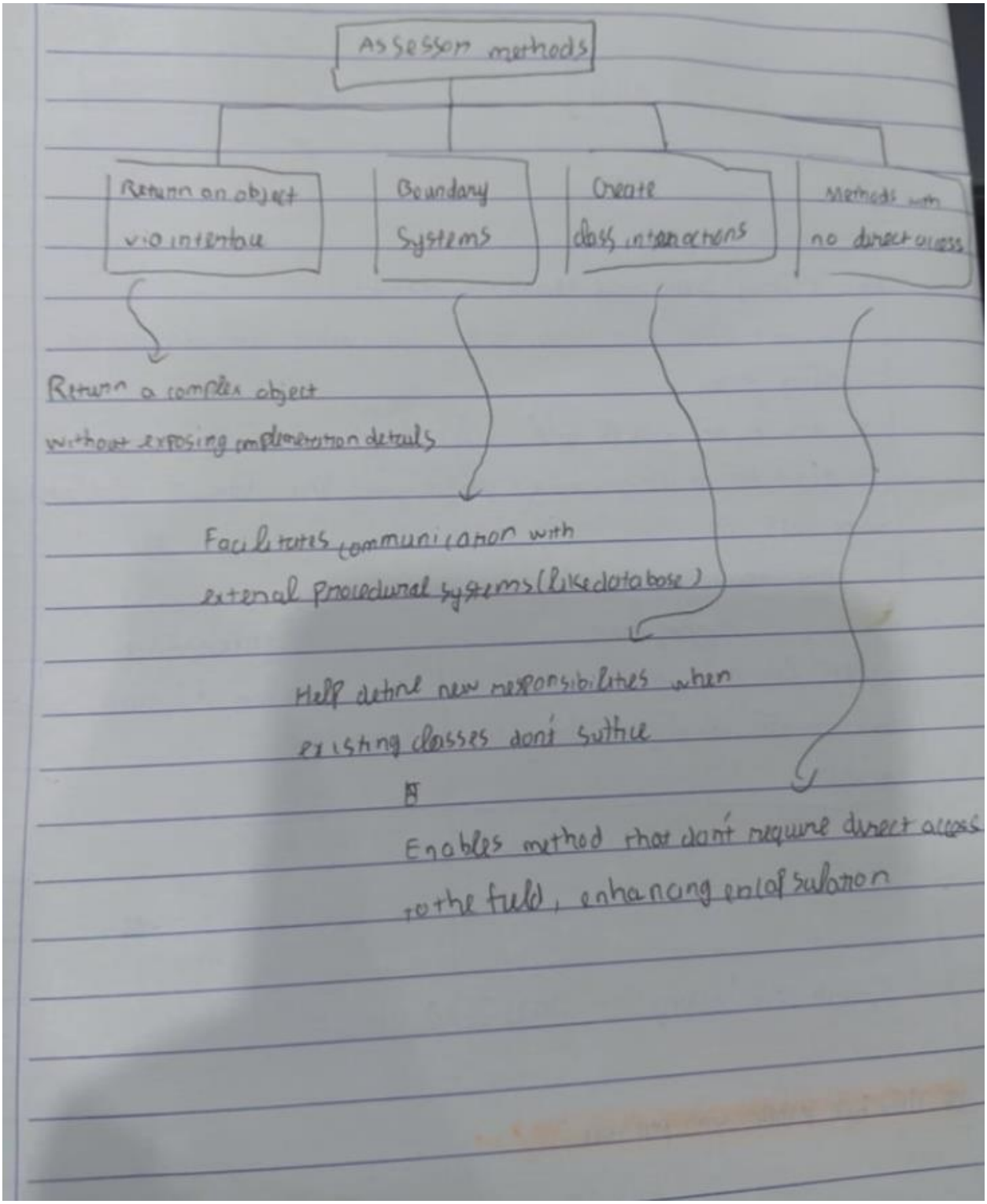
Câu hỏi: If you create a constructor method to build a DVD by title then create a constructor method to build a DVD by category. Does JAVA allow you to do this?

Tạm dịch: Nếu bạn tạo một phương thức khởi tạo để xây dựng một DVD theo tiêu đề, sau đó tạo một phương thức khởi tạo để xây dựng một DVD theo thể loại. Java có cho phép bạn làm điều này không?

Trả lời: Có, Java cho phép bạn làm điều này thông qua tính năng **nạp chồng phương thức (method overloading)**. Bạn có thể tạo nhiều constructor cho cùng một lớp với các bộ giá trị khởi tạo khác nhau dựa trên các tham số mà nó nhận.

Ví dụ:

```
public class DVD {
    private String title;
    private String category;
    public DVD(String title) {
        this.title = title;
    }
    public DVD(String category) {
        this.category = category;
    }
}
```


Accessor methods
Return an object via interface
Boundary Systems
Create class interactions
Methods with no direct access
Return a complex object without exposing implementation details
Facilitates communication with external procedural systems (like database)
Help define new responsibilities when existing classes don't suffice
Enables method that don't require direct access to the field, enhancing encapsulation