Object-Oriented Programming

Nguyễn Văn Tiến – 20205195

**Lab 04: Inheritance and Polymorphism**

1. Code update.
2. Create abstract class *Media*, override method *toString()* and create 2 comparators ***COMPARE\_BY\_TITLE\_COST*** and ***COMPARE\_BY\_COST\_TITLE.***

A screen shot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generated

1. Create class *Book* extends *Media*.

A screenshot of a computer program

Description automatically generated

1. Create class *Disc* extends Media, modify class *DigitalVideoDisc* and create *CompactDisc*, both are extends *Disc*.

A screen shot of a computer

Description automatically generated

A screen shot of a computer screen

Description automatically generated

A screen shot of a computer program

Description automatically generated

1. Create class *DataGenerator* to generate data.

A screen shot of a computer program

Description automatically generated

1. Update menu(UI) for Main.

A screenshot of a computer program

Description automatically generated

1. Update usecase diagram and class diagram.
2. Update usecase diagram.

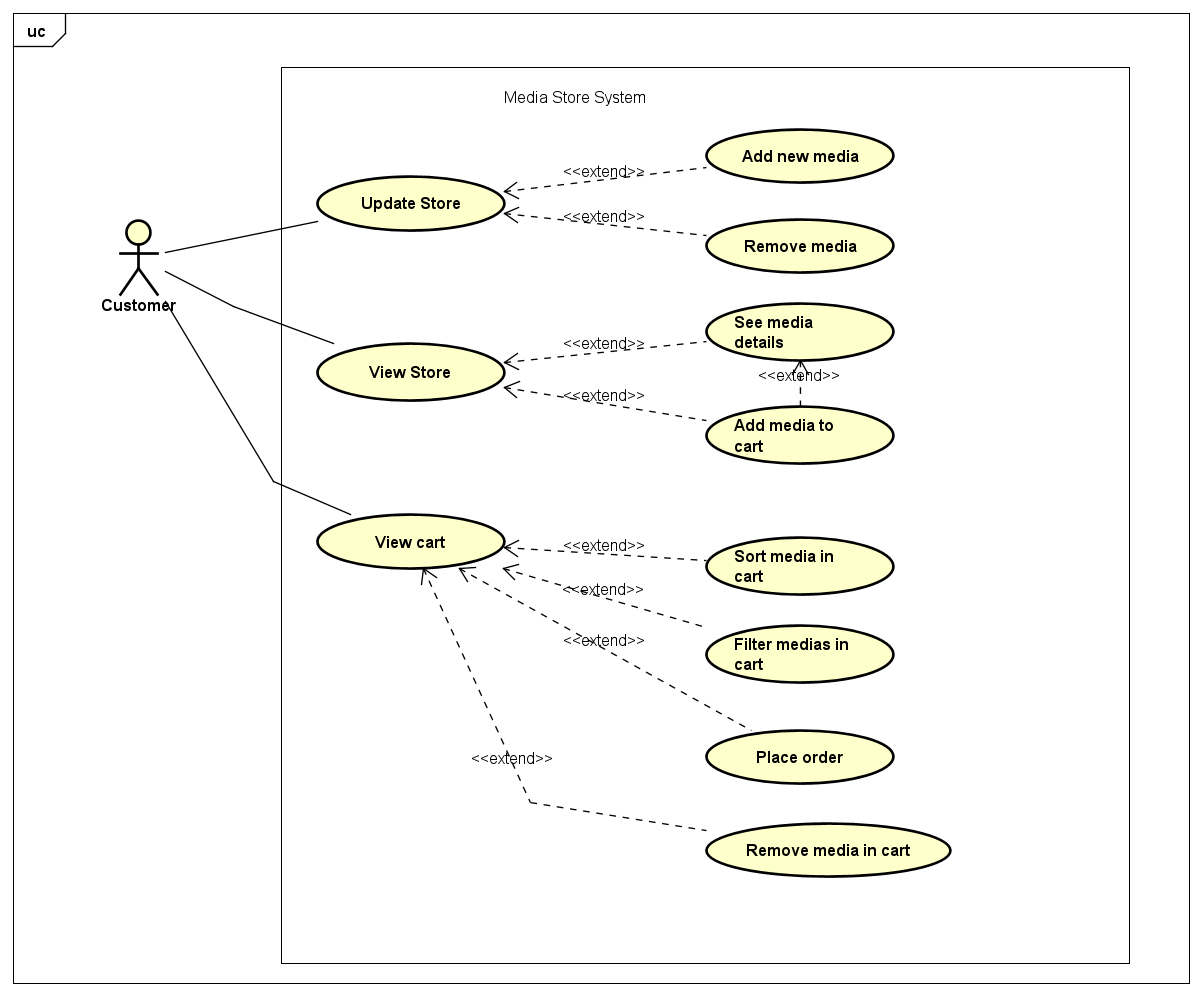


Figure 6: Usecase diagram

1. Update class diagram.

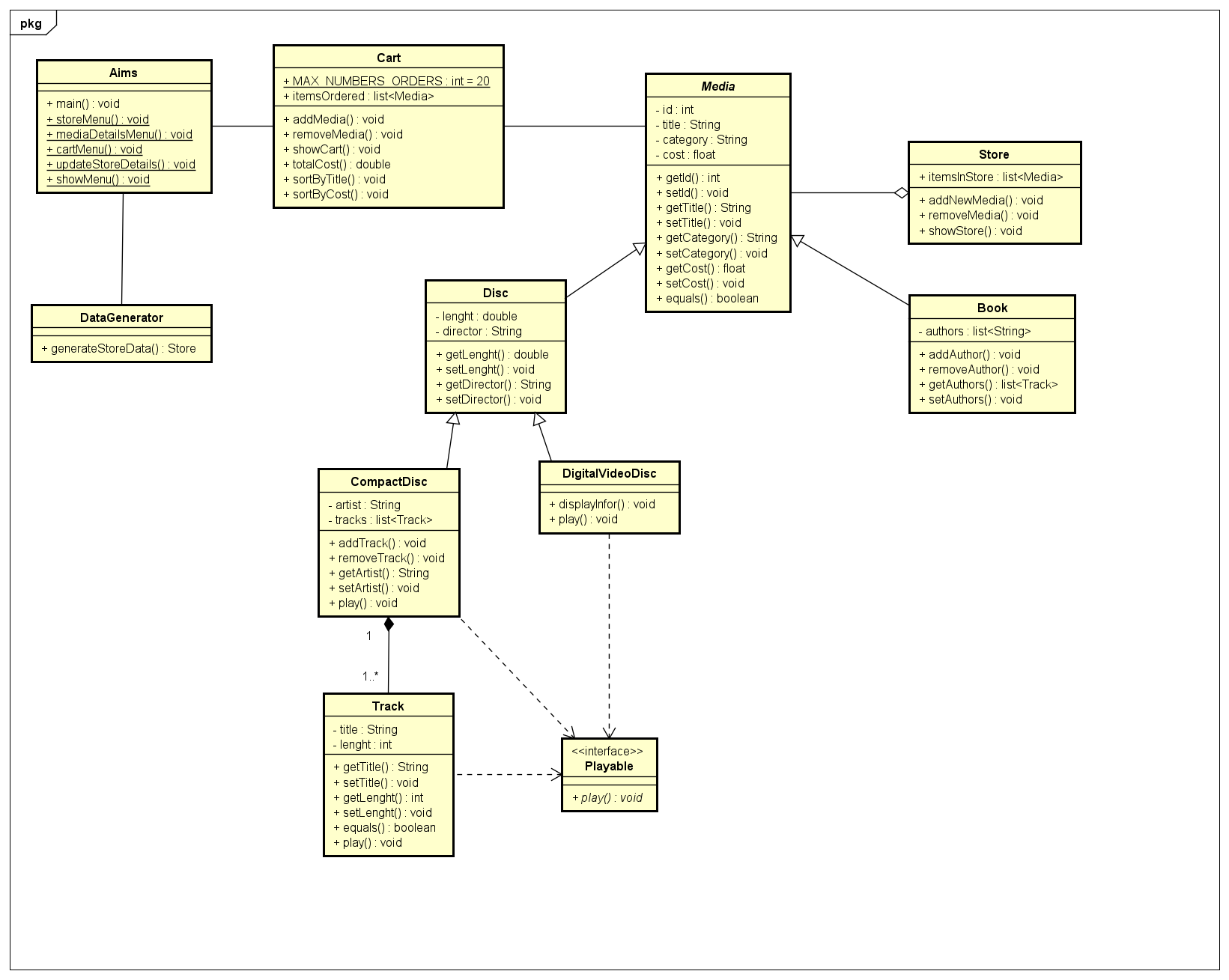


Figure 7: Class diagram

1. Answer the questions.

Question: Alternatively, to compare items in the cart, instead of using Comparator, we can use the Comparable interface and override the compareTo()method. You can refer to the Java docs to see the information of this interface.

Suppose we are taking this Comparable interface approach.

**- What class should implement the Comparable interface?**

The class that represents the object we want to compare should implement the Comparable interface. In this case:

* **The Media class** should implement Comparable<Media> because all media items (e.g., Book, CompactDisc, DigitalVideoDisc) inherit from Media, and we want to compare objects of these subclasses.

**- In those classes, how should you implement the compareTo()method be to reflect the ordering that we want?**

The compareTo() method should reflect the primary and secondary ordering criteria. For example, to compare items **by title first, then by cost**, the implementation could look like this:

@Override

public int compareTo(Media other) {

// Compare by title alphabetically (ignoring case)

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

if (titleComparison != 0) {

return titleComparison;

}

// If titles are equal, compare by cost (ascending)

return Float.compare(this.cost, other.cost);

}

This ensures:

1. **Primary ordering**: Title (case-insensitive, ascending).
2. **Secondary ordering**: Cost (ascending).

**- Can we have two ordering rules of the item (by title then cost and by cost then title) if we use this Comparable interface approach?**

No, the Comparable interface only allows **one natural ordering** to be defined through the compareTo() method.

To have multiple ordering rules (e.g., "by title then cost" and "by cost then title"):

* Use the Comparator interface instead.
* Create separate Comparator implementations for each ordering rule.

For example:

Comparator<Media> byTitleThenCost = Comparator.comparing(Media::getTitle)

.thenComparing(Media::getCost);

Comparator<Media> byCostThenTitle = Comparator.comparing(Media::getCost)

.thenComparing(Media::getTitle);

**- Suppose the DVDs has a different ordering rule from the other media types, that is by title, then decreasing length, then cost. How would you modify your code to allow this?**

To allow DVDs to have a different ordering rule, override the compareTo() method in the **DigitalVideoDisc** class (a subclass of Media). The compareTo() implementation in DigitalVideoDisc would reflect the specific ordering rule for DVDs:

A screen shot of a computer program

Description automatically generated