**OOP Lab 004 Report**

**9. Constructors of whole classes and parent classes**

Which classes are aggregates of other classes? Checking all constructors of whole classes if they initialize for their parts?

**• Aggregates:**

* **Store** aggregates Media.
* **Cart** aggregates Media.
* **CompactDisc** aggregates Track.

**• Store Class**

* **Attributes**: Likely contains a collection of Media objects.
* **Constructor**: Initializes the list of Media.
* **Aggregation**: The Store class aggregates Media objects because Media can exist independently of the Store.

**• Cart Class**

* **Attributes**: Likely contains a collection of Media objects.
* **Constructor**: Initializes the list of Media.
* **Aggregation**: The Cart class aggregates Media objects for the same reason as Store.

**• Disc Class**

* **Attributes**: May contain additional details like length and director.
* **Constructor**: Sets properties for Disc, and indirectly via inheritance, initializes Media attributes.
* **Aggregation**: Aggregates no separate objects but inherits from Media.

**• CompactDisc Class**

* **Attributes**: Contains a List<Track> and an artist.
* **Constructor**: Likely initializes the List<Track>.
* **Aggregation**: The CompactDisc aggregates Track because Track instances can exist independently of a CompactDisc.

**• Track Class**

* **Attributes**: Title and length.
* **Constructor**: Initializes these properties.
* **Aggregation**: Not an aggregate class since it contains no other objects.

**• DigitalVideoDisc Class**

* **Attributes**: Inherits Disc attributes and methods.
* **Constructor**: Sets properties specific to DigitalVideoDisc and initializes inherited ones.
* **Aggregation**: None - it directly inherits from Disc.

**10. If the passing object is not an instance of Media, what happens?**

**equals**() would return false if the object supplied to it wasn't an instance of **Media** or **Track**. In the event that type checking is not handled, this prevents **ClassCastException** and guarantees type safety.

**12. Sort media in the cart**

**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 Media class should implement the Comparable interface since we want to define a default ordering for media objects.

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

A screen shot of a computer program

Description automatically generated

If the title is the same, then it would compare by cost (item with higher cost should come first

* **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, it's really challenging because a class can only have one natural ordering according to the Comparable interface. I believe that Comparator should be used in place of several ordering rules, such as sorting by cost or title.

* **Suppose the DVDs have 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?**

**A computer screen shot of a program code

Description automatically generated**

I believe we can override the compareTo() method in the DigitalVideoDisc class because DVDs have different ordering criteria than other media types. The altered compareTo() function will then compare the DVDs.

**Test result:**  
A screen shot of a computer program

Description automatically generated