**OOP Lab 003 Report**

1. **Updated Class Diagram and Use Case Diagram**

A diagram of a computer network

Description automatically generated

Updated Class Diagram

A diagram of a network

Description automatically generated

Updated Use Case Diagram

1. **Working with method overloading**

**Question:** *Try to add a method* ***addDigitalVideoDisc*** *which allows to pass an arbitrary number of arguments for* ***dvd****. Compare to an array parameter. What do you prefer in this case?*

****

With the array parameter, you can just simply check the length of the ***dvdList*** array to ensure the total number of DVDs in the cart does not exceed the maximum. This provides a more straightforward way to enforce the cart's capacity limit.

1. **Passing parameter**

**Is JAVA a Pass by Value or a Pass by Reference programming language?**

* Java is a pass-by-value programming language. For example, if you pass an object into a method in Java *(****swap(DVD dvd1, DVD dvd2)***, the method only receives the address values that point to the dvd1 and dvd2 objects in memory. So, if you try to swap the objects by doing ***tmp = dvd1****;* ***dvd1 = dvd2****;* ***dvd2 = tmp***, it won't work
* This is because the method is only changing the values of the local variables **dvd1** and **dvd2**, which does not affect the original objects.

A screenshot of a computer program

Description automatically generated

After the call of ***swap(jungleDVD, cinderellaDVD)*** , why does the title of these two objects remain?

* After calling ***swap(jungleDVD, cinderellaDVD)***, the titles of these two objects remain the same. As mentioned earlier, the ***swap()*** method is only manipulating the local variables, not the original ***jungleDVD*** and ***cinderellaDVD*** objects.

After the call of ***changeTitle(jungleDVD, cinderellaDVD.getTitle())*** why is the title of the JungleDVD changed?

* After calling **changeTitle(jungleDVD, cinderellaDVD.getTitle()),** the title of the **jungleDVD** object is changed. In this case, the **changeTitle**() method is passed a reference to the **jungleDVD** object, so when the title of the **dvd** object (which is the same as **jungleDVD**) is modified, it changes the title of the original **jungleDVD** object, since they both reference the same object in memory.

1. **Debugging Java in Eclipse**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

A screenshot of a computer

Description automatically generated

1. **Classifier Member and Instance Member**

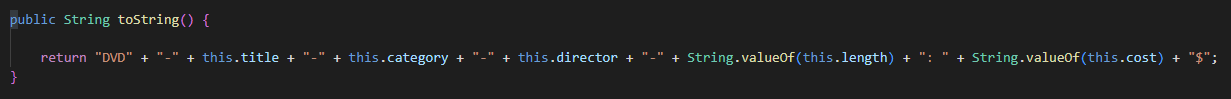
**A screen shot of a computer program

Description automatically generated**

1. **Open the Cart Class**

Write a ***toString()*** method for the ***DigitalVideoDisc*** class. What should be the

return type of this method?



* The method should return a String

A screen shot of a computer program

Description automatically generated

A screenshot of a computer screen

Description automatically generated

1. **Implement the Store Class**

**A screen shot of a computer program

Description automatically generated**

Testing:

A screenshot of a computer program

Description automatically generated

**String, StringBuilder and StringBuffer**

**A screen shot of a computer program

Description automatically generated**

String concatenation very slow when handle a really big file

**A screen shot of a computer program

Description automatically generated**

Much faster and more efficient with StringBuffer

The main difference between StringBuffer and StringBuilder in Java is that StringBuffer is thread-safe, while StringBuilder is not.

StringBuffer is synchronized, which means its methods can be safely called from multiple threads without the need for external synchronization. This ensures that the operations on the string buffer are atomic and thread safe.