Andrew Guibert

SE339

Homework 3

The most notable design property of my Rental Store program is the inheritance tree used, as well as collecting common code into abstract classes wherever possible.

I started by creating two interfaces: Rentable and Sellable.

Since most products can be either sold or rented, I created an abstract class:

public abstract class AbstractProduct implements Rentable, Sellable

In this abstract class I implemented common functionality that would be the same for all products:  
getName()

isNewRelease()

daysOverdue()

getDaysRented()

getFrequentRenterPoints()

Some methods are intended to have functionality specific to implementations, so I made these methods abstract, thus forcing the concrete class to implement them:  
getRentalCost()

getSellCost()

In order to distinguish between a concrete class that is intended to be used for Renting or Selling purposes, there were two constructors provided on each concrete class.

One was for a sellable product, which did not take any daysRented argument:

public BluerayMovie(String name, boolean isNewRelease) {

super(name, isNewRelease);

}

The other constructor was for a rentable product, which additionally provided the number of days the product was rented, as well as a fixed rental period for that concrete product:

public BluerayMovie(String name, boolean isNewRelease, int daysRented) {

super(name, isNewRelease, daysRented, 4 /\* day rental period \*/);

}

To make the rental store work with these new types, I added sellable methods to my Transaction class:

public void addSellable(Sellable arg)

On the transaction class, I kept two separate lists of products:

private Set<Rentable> rentals = new HashSet<Rentable>();

private Set<Sellable> sales = new HashSet<Sellable>();

When the statement was generated, it was easy to iterate over each set and generate unique statement fields based on the if the product was a rental or a sellable object.