Introduction to Computers and Engineering Problem Solving 1.00 / 1.001 Fall 2004

Problem Set 4

Due: 11AM, Friday October 15, 2004

Problem 1: Movie & Game Rental Store (3) [70 points]

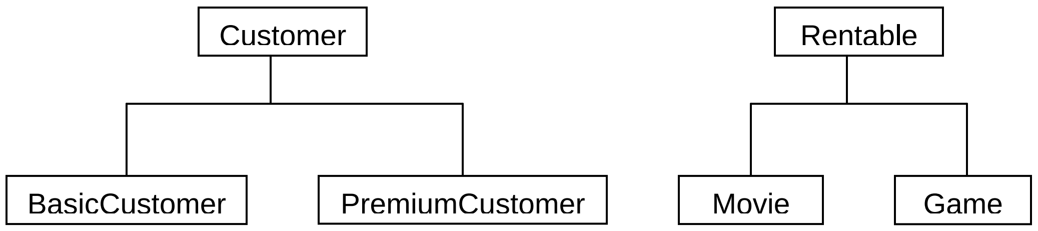
Introduction

With plenty of copies in stock, now some customers want to keep more movies and games. Besides, it looks like you will make more profit this way. So you have decided to introduce two distinguished rental plans: basic and premium. The **basic plan** will allow customers to keep **3 movies and 2 games** while the **premium plan** customers can rent up to **6 movies and 4 games**.

To make the necessary changes to your software, you will use the concept of 'inheritance'. See the class hierarchy diagram below for the relationships among classes. By convention, classes connected to another class by a line and below another class inherit from the class above them. Similarly, classes connected to an interface by a line and placed below that interface should implement that interface.

You will implement two concrete classes, `BasicCustomer` and `PremiumCustomer`, which extend the **abstract** class `Customer`. Also, you will introduce the **interface** `Rentable` for the Movie and Game classes.

Class Hierarchy



Assignment

Create files to implement seven Java classes and interfaces:
`Customer.java`, `BasicCustomer.java`, `PremiumCustomer.java`,
`Rentable.java`, `Movie.java`, `Game.java`, and `ProblemSet4.java`.

1. `Customer.java`
   - Please note that `Customer` is an abstract class. What keyword do you need to create an abstract class?
   - Define a data member, ‘name’, to hold the name of customer.
   - Write a single-argument constructor that sets the name of the customer.
   - Complete getter/setter methods for the ‘name’ data member.
   - Write 4 abstract methods for 1) renting a movie, 2) returning a movie, 3) renting a game, and 4) returning a game. How is an abstract method different from a concrete method?
   - Write a concrete `printInfo()` method, which prints out the name of customer.

2. `BasicCustomer.java`
   - Please note that `BasicCustomer` is a subclass that extends the class `Customer`. What keyword do you need?
   - Define 2 data members, ‘rentedMovies’ and ‘rentedGames’, using instances of `ArrayList` class to store references to the instances of movies and games that a customer has rented.
   - Define 2 `final` variables `MAXMOVIE` and `MAXGAME`, which are the maximum number of movies and games that basic customers can keep.
   - Write a single-argument constructor, which should not only initialize the class data members, but also invoke the constructor of class `Customer`.
   - Write a single-argument public method for renting a movie. What are the necessary conditions for renting a movie? This method needs to check first if customer already has this movie. Don’t forget to update the availability of the movie as well.
   - Write a single-argument public method for returning a movie. Again, your program needs to update the availability of a movie.
   - Complete public methods for renting and returning a game.
   - Write a `printInfo()` method, which prints out the name of customer and the title of all the movies and games he/she has. Please note that, to output the name of customer, you **should** use the `printInfo()` method of `Customer` class.

3. `PremiumCustomer.java`
   - Similar to `BasicCustomer.java`, but with different values for the maximum number of movies and games a customer can keep.

4. `Rentable.java`
   - Please note that `Rentable` is an `interface`. What is the definition of `interface`? What keyword do you need?
   - Define an `abstract` method `isAvailable()`.

- Define `abstract` methods `rentItem()` and `returnItem()`.

5. `Movie.java`
    - Update `Movie.java` from Problem Set 3, which should `implement` the `Rentable interface`.

6. `Game.java`
    - Similar to `Movie.java`.

7. `ProblemSet4.java`
    - Write a `main()` method as described below to test the other classes and interfaces you have created.
    - First, create 2 customers, 3 movies, and 3 games.
    - Invoke `printInfo()` methods on all 8 objects. Availabilities of all the movies and games should be true at this point.
    - Run a few transactions so that some of the movies and games are rented out to some of the customers.
    - Invoke `printInfo()` methods again on all the instances of the objects. Check the output to make sure all the information is up-to-date.
    - Finally, try few more transactions and invoke the `printInfo()` methods. Is the print-out correct?

Problem 2: Generalization of Fibonacci [30 points]

Introduction

Generalized Fibonacci numbers are defined as series of nonnegative integers produced by the following rule:

$F(0) = 0$
$F(1) = 1$
$F(n) = a \times F(n-1) + b \times F(n-2) \quad \text{if } n > 1$

where *a* and *b* are positive integers.

Thus, the first few Generalized Fibonacci numbers when *a = 2* and *b = 3* are

0, 1, 2, 7, 20, 61, 182 …

Assignment

Create a file to implement `Recursion.java`.

1. Static method `generalFibonacci()`

- Write a static method `generalFibonacci()` that takes 3 arguments: values of *a*, *b*, and *n*.
- Implement this method in a recursive way so that the method invokes itself.
- What are the base conditions when *n = 0* and *n = 1*?

2. `main()` method
   - Ask users for the values of *a*, *b*, *n* using the `showInputDialog()` method of `JOptionPane` class.
   - Run the `generalFibonacci()` method and print the output for some specific inputs you choose.

Turn In

- Turn in **electronic** copies of **all source code** (`.java` files). No printed copies are required.
- Place a **comment** with your full name, Stellar username, tutorial section, TA's name, and assignment number at the beginning of all `.java` files in your solution.
- Remember to **comment your code**. Points will be taken off for insufficient comments.
- Place all of the files in your solution into a **single zip file**. Submit this single zip file on the 1.00 Web site under the appropriate section and problem set number. For directions, see **HowTo: Submit Homework** on the 1.00 Web site.
- Your solution is **due at 11AM**. Your uploaded files should have a time stamp of no later than 11AM on the due date.
- **Do not** turn in compiled byte code (`.class` files) or backup source code (`.java~` files).

Penalties

- **30% off** If you turn in your problem set **after 11AM on Friday** but **before 11AM on the following Monday**.
- **No Credit** If you turn in your problem set **after 11AM on the following Monday**.