Design Exercise

This exercise is divided into two parts. The first part is this document, which describes the requirements for a piece of software and several alternative designs. The second part is a set of questions related to the alternative designs. These are available via the Internet, with a link from the course syllabus.

The objective of this exercise is to prepare you for further discussion that we will have in class. As such, you will not be graded on the specifics of your answers; rather, you will receive full credit if you spend at least one but no more than two hours on this exercise, and make a good faith effort to answer each of the questions as you are best able.

Problem description

You are hired by a local video rental store to write software to help with the store operation. This includes things like keeping track of the inventory, the customers, the stock items that are checked out, and accruing customer fines. The store is staffed by employees, who are the only people who will have access to the software. The store rents its items of stock, which includes movies and computer games.

The software should support a set of queries about customers and stock items, as well as a set of transactions that change customer or stock information. The queries that should be supported are of the form:

- What are all movies starring a given Actor?
- Is there a game with a given Title that is available (i.e. not checked out)?
- What are all movies that a particular Customer has checked out, with their due dates?
- What is a Customer's account balance?
- What are all stock items that are overdue?

The transactions are of the form:

- Check out a stock item to a customer
- Check in a stock item from a customer and add fines to their balance, if any
- Add a stock item
- Add a customer

Alternative Designs

Below are four different designs for the video store software. Each design is described in terms of its classes, the responsibilities of each class, and how the classes interact. A one-letter name has been assigned to each design.

Anything described below as attribute X is assumed to be private but with public getX and setX methods. Anything described as an operation is assumed to be a public method.
Design F

Classes

CheckedInGame, CheckedOutGame, CheckedInMovie, CheckedOutMovie, Customer, CommandProcessor, VideoStore.

Class details

CheckedInGame represents a game that is currently in the video store, i.e. it is not currently rented. It has attributes rentalPrice, title, itemID, and platform.

CheckedOutGame represents a game that is currently being rented. It has attributes rentalPrice, title, itemID, and platform, dueDate, and customerWhoIsRenting. Attribute customerWhoIsRenting references a Customer object associated with the person who is currently renting the stock item.

CheckedInMovie represents a movie that is currently in the video store. It has attributes rentalPrice, title, itemID, and actor.

CheckedOutMovie represents a movie that is currently being rented. It has attributes rentalPrice, title, itemID, and actor, dueDate, and customerWhoIsRenting. Attribute customerWhoIsRenting references a Customer object associated with the person who is currently renting the stock item.

Customer has attributes accountBalance, name and customerID.

CommandProcessor has a Vector of CheckedInGame, representing all games currently in the store, a Vector of CheckedOutGame, representing all games currently rented, a Vector of CheckedInMovie, representing all movies currently in the store, a Vector of CheckedOutMovie, representing all movies currently rented, and a Vector of Customer, representing all current customers. CommandProcessor supports the following operations:

findByTitle, which determines whether a stockItem with the given title exists;
findByActor, which returns all movies in which the actor stars;
isCheckedOut, which determines whether a stockItem is checked out;
balance, which returns a given customer’s account balance;
overdueItems, which returns all stock items that are currently overdue;
checkOut, which checks out a stockItem to a customer on a given date;
checkIn, which checks in a stockItem from a customer on a given date;
addCustomer, which adds a customer to the set of store customers;
addStockItem, which adds a new stock item to the store stock;

VideoStore contains the main control. It is responsible for creating the CommandProcessor, taking user commands, passing these commands to the CommandProcessor, and displaying the results returned from the CommandProcessor.
Design Y

Classes

StockItems, Customers, RentedItems, and VideoStore.

Class details

StockItems encapsulates all of the current video store stock items. It has the following attributes:

- float[] rentalPrice;
- String[] title;
- String[] itemID;
- String[] actor;
- String[] platform;
- int itemCount;
- int currentItem;

The value of itemCount is the current number of stock items, while currentItem acts as an iterator on each of the attribute arrays. To find the attributes of any single item, one simply references the same cell in each of the attribute arrays, so that rentalPrice[] refers to the rental price of the ith item and title[] refers to its title. If platform[] is null then the item is a movie, while if actor[] is null then the item is a game. If the array size limit is ever reached, then an array with double the size is allocated for each of the attributes and initialized to the values currently in the attribute arrays. StockItems supports the following operations:

- addStockItem, which adds a new stock item to the store stock;
- findByTitle, which determines whether a stockItem with the given title exists;
- findByActor, which returns all movies in which the actor stars;

Customers encapsulates all of the current customers of the video store. It has the following attributes:

- float[] accountBalance;
- String[] name;
- String[] customerID;
- int customerCount;
- int currentCustomer;

Attributes customerCount and currentCustomer are similar to the corresponding attributes in StockItems, and the ith customer is represented by indexing the attribute arrays by the value i. Customers supports the operations:

- addCustomer, which adds a customer to the set of store customers;
- balance, which returns a given customer’s account balance;

RentedItems encapsulates all of the stock items that are currently rented by a customer. It has the following attributes, which represent each rented item in a manner similar to Customers and StockItems:

- String[] customerID;
- String[] itemID;
- Date[] dueDate;
- int itemCount;
- int currentItem;

RentedItems supports the operations:

- checkOut, which checks out a stockItem to a customer on a given date;
- checkIn, which checks in a stockItem from a customer on a given date;
- isCheckedOut, which determines whether a stockItem is checked out;
- overdueItems, which returns all stock items that are currently overdue;

VideoStore contains the main control. It is responsible for creating StockItems, Customers, and RentedItems, taking user commands, passing these commands to either StockItems, Customers, or RentedItems, and displaying the results.
Design K

Classes

StockItem, Customer, Movie, Game, RentedItem, Map, MapIterator, QueryProcessor, TransactionProcessor, and VideoStore.

Class details

StockItem is the superclass for Movie and Game, and has attributes rentalPrice, title, and itemID.
Customer has attributes accountBalance, name and customerID.
Movie inherits from StockItem, and adds an actor attribute.
Game inherits from StockItem, and adds a platform attribute, indicating the game platform on which the game will operate (e.g. PlayStation, Nintendo, Xbox, etc.).
RentedItem represents a StockItem that is rented by a customer. It has attributes customerID, itemID, and dueDate.
Map represents a collection of objects of type T, each object associated with a key of type k. Only a single object of type T can be associated with any particular key. Map supports the operations:
    find(k), which returns the object of type T associated with key of type k,
    insert(k,T), which inserts an object of type T into the Map associated with a key of type k,
    delete(k), which deletes the object of type T associated with the key of type k.
    iterator(), which returns a MapIterator.
MapIterator is a generalization of a reference to memory. It has operations:
    begin(), which sets the iterator to the first element of the associated Map<k,T>,
    get(), which returns an element of type T at the iterator’s current location,
    hasNext(), which returns a boolean if there are remaining elements in the Map to iterate over,
    next(), which advances the iterator to the next element in the iterated sequence.
QueryProcessor handles queries from the user that don’t result in changes to the customer or stock item data. It has as attributes a reference to Map<String, Customer>, which holds all customers, with the customerID field as the key, and a reference to Map<String, StockItem>, which holds all stock items, with the itemID as the key. It also has a reference to a Vector of RentedItem, which represents all stock items currently rented. The operations it supports are:
    findByTitle, which determines whether a stockItem with the given title exists;
    findByActor, which returns all movies in which the actor stars;
    isCheckedOut, which determines whether a stockItem is checked out;
    balance, which returns a given customer’s account balance;
    overdueItems, which returns all stock items that are currently overdue;
TransactionProcessor handles commands from the user that result in changes to the customer or stock item data. It has the same attributes as the QueryProcessor (i.e. references to the Map of all customers, the Map of all stock items, and the Vector of all rented items). The operations it supports are:
    checkOut, which checks out a stockItem to a customer on a given date;
    checkIn, which checks in a stockItem from a customer on a given date;
    addCustomer, which adds a customer to the set of store customers;
    addStockItem, which adds a new stock item to the store stock;
VideoStore contains the main control. It is responsible for creating Maps of both customers and stock items, taking user commands, passing these commands to the Query or Transaction Processor, and displaying the results.
Design Q

Classes

StockItem, Customer, and VideoStore.

Class details

**StockItem** represents both movies and games, and has attributes rentalPrice, title, itemID, isMovie, platform, actor, dueDate, and customerWhoIsRenting. If isMovie is set to true, then platform is null and actor has a value; similarly, if isMovie is false, then platform has a value and actor is null. If the StockItem is checked out, then dueDate is non-null, and customerWhoIsRenting references a Customer object associated with the person who is currently renting the stock item.

**Customer** has attributes accountBalance, name, customerID, and itemsRenting. Attribute itemsRenting is a Vector of references to StockItem objects, representing the stock items that the customer is currently renting.

**VideoStore** contains the main control. It has a Vector of StockItem, representing all items in stock, and a Vector of Customer, representing all current customers. VideoStore supports the following operations:

- **findByTitle**, which determines whether a stockItem with the given title exists;
- **findByActor**, which returns all movies in which the actor stars;
- **isCheckedOut**, which determines whether a stockItem is checked out;
- **balance**, which returns a given customer’s account balance;
- **overdueItems**, which returns all stock items that are currently overdue;
- **checkOut**, which checks out a stockItem to a customer on a given date;
- **checkIn**, which checks in a stockItem from a customer on a given date;
- **addCustomer**, which adds a customer to the set of store customers;
- **addStockItem**, which adds a new stock item to the store stock;

VideoStore is also responsible for taking user commands, executing the command directly on the appropriate vector or vectors, and displaying the results.