Project Description

Create an inventory application for a multimedia store such as Amazon (AmazonLite). The system will manage an inventory comprised of CDs, DVDs, and books. Each inventory item should, at a minimum, include a unique identifier (i.e., a "GUID"), an item description and a quantity. The system will allow the user to enter new items into the system, search for existing items, and edit existing items, and delete specified items. These operations are often respectively referred to as CRUD (Create, Retrieve, Update, and Delete) operations. User input and output can be accomplished by using the System.in and System.out objects. This application will implement the MVC component architecture. Our course shell contains explanatory material regarding MVC in the Discussion Board (Assignment 1 threads), as well as in the Resources page. You are strongly advised to not consult or use external resources regarding the MVC architecture. We are using the "classical" MVC architecture. Looking at other approaches to MVC invariably confuse students and impede learning. By the end of this course, your understanding of MVC will have jelled sufficiently that you'll be able to interpret and analyze other approaches to MVC, but until then, it is best that students focus solely on the approach detailed with our course shell. The user interface code will be modularized as a View (or Delegate) component, such that it can easily be swapped out for a different View component. The initial view will be a text-based console view. It is recommended that you use the Scanner class to manage keyboard input. The underlying inventory representation will be modularized as a Model component such that it can easily be swapped out for a different Model. The inventory will be persistent, and it will be maintained using the Properties class, and the operation of the Properties functionality must be fully contained, encapsulated, and hidden within the Model component. You must utilize a double-dispatch relationship between the Model and the View components.

The View class will include "View" as part of its name. The Controller class will include "Controller" as part of its name. The Model class will include "Model" as part of its name. You must use an appropriately named "Inventory" or "App" class as your main method class. The code for your main method is provided in the Assignment 1 discussion thread entitled 'More on MVC… and What Your "main" Method Will Be'.

Your application will make use of inheritance, polymorphism, and at least one interface definition. It will also make use of at least one enumeration (enum type).

Feedback (wrong double-dispatch between Model and the View)

Your Model makes a lot of assumptions about the View that it is working with. The idea with MVC is that these components really don't know about each other. They only know about their respective responsibilities. The Model only knows about how to services from from the View (via the Controller), and how to call the View's notify() method (which you don't provide) so that the View can then call back upon the Model to get the information necessary to refresh its presentation to the user. What I've just described is the double-dispatch relationship between the Model and the View. Additionally, your Model component has the following code:

```
//method setSelectedOperation(int operation) sets operation in model
// and updates view wiht an integer
public void setSelectedOperation( int operation) {
    if (operation == 1)
        currentOperation = toPerform. CREATE;
    else if (operation == 2)
        currentOperation = toPerform. RETRIEVE;
    else if (operation == 3)
        currentOperation = toPerform. UPDATE;
    else if (operation == 4)
        currentOperation = toPerform. DELETE;
    view. newView(operation);
   }
//method performOperation() performs CRUD operations
public void performOperation(){
    if ( currentOperation == toPerform. CREATE)
        this. createItem( getInputString1());
    else if ( currentOperation == toPerform. RETRIEVE)
        this. retrieveItem( getInputString1());
    else if ( currentOperation == toPerform. UPDATE){
        this. updateItem( getColumnIndex(), getInputString1(), getInputString2());
      }
    else if ( currentOperation == toPerform. DELETE)
        this. deleteItem( getInputString1());
   }
}
```

So, your Model is directing the operations of the View. But, the Model shouldn't know anything about the View! The logic that you have here in your Model is View logic!