# Assignment Module 1 Description

*By Michael Floerchinger*

05/05/2017

Create an inventory system for a multimedia store such as Borders (remember Borders?).

The system will manage an inventory comprised of CDs, DVDs, and books. The system will

allow the user to enter new items into the system, search for and edit existing items,

and delete items. These operations are often collectively referred to as CRUD

(Create, Retrieve, Update, and Delete). The input and output can be accomplished by using

the System.in and System.out objects. 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.

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).

Your application will run as runnable jar file.

### **Deliverables**

There are four deliverables that must be submitted. These deliverables are as follows:

1. A brief Software System Requirements Specification document. This should reiterate the project

description above, and provide enough additional specifications to facilitate design and implementation.

2. A brief Software System Design Document that identifies the key abstractions of your application,

and describes the relationships between them.

3. Your application as a runnable jar.

4. IMPORTANT! Your source code as well as capture images of your application in operation within a

separate zip file.

# Deliverables

## **Software System Requirements Specification Document**

Key abstractions of the inventory application, and the relationships between the classes, objects, and methods

### **Overview**

The media inventory application contains the following classes:

* InventoryProgramMainApp
* InventoryProgramModel
* InventoryProgramView
* InventoryProgramController
* InventoryProgramPersistance
* InventoryProgramStoreOperations
* InventoryItems
* InventoryCD
* InventoryDVD
* InventoryBOOK
* Observer
* Subject

**InventoryProgramMainApp**

InventoryProgramMainApp contains the main method for the inventory system. It wires the InventoryProgramView, InventoryProgramController, and InventoryProgramModel together by ensuring the controller has a reference to the model, the view has a reference to controller using setModel/getModel setter and getter methods, implemented in all the MVC classes. The view method start() initializes the application.

**InventoryProgramModel – InventoryProgramView – InventoryProgramController**

**InventoryProgramView** uses Scanner to process user input. The start() method starts the application which uses the switch method to present to a user a menu list of options for LIST, CREATE, UPDATE, DELETE or QUIT. The InventoryProgramView calls methods in the InventoryProgramController class for access to all the applications methods and functions. The application also uses the Observer Pattern to notify the view when an change has occured in the data layer of the application. This envokes the update() method which retrieves String buffers from the Model layer.

**InventoryProgramController** implements the controller methods which provides an abstraction from the view to the model. The controller receives calls from the view and invokes methods in the model, referencing the InventoryProgramStoreOperations interface for the activities for LIST, CREATE, UPDATE, DELETE

**InventoryProgramModel** implements the interface methods from the InventoryProgramStoreOperations class, and in doing so invokes all the method calls for interaction with the data store persistence layer. This extends the Properties class, which persists datastore values to a properties file. The model also holds the returns from the InventoryProgramPersistance as String buffers which are returned to the view through implementing the Observer pattern to notify the view a change as occurred in the data store.

**InventoryProgramPersistance - InventoryProgramStoreOperations**

**InventoryProgramPersistance** class extends the Properties class where key:value pairs are stored to a properties file on the local filesystem. During object instantiation, the constructor checks for the presence of the file located in the filesystem. Only if not present does it create a new file. The key:value pairs are single entry items, differentiated by itemType, CD, DVD, Book. The key:value pairs are delimited Strings using the ',' as the delimiter. The InventoryProgramPersistance also maintains the item counter for the different itemTypes, by incrementing the count for each item used, and storing the next available itemNum value back to the properties file.

**InventoryProgramStoreOperations** class is implemented as a interface thereby providing common methods used by the application for the method operations for LIST, CREATE, UPDATE and DELETE functions within the application.

**Subject - Observer**

**Subject** class implements the Observer design pattern by implementing a notification system to notify any registered "observer" views of changes that occured within the data structure. Subject is extended by the InventoryProgramModel class. The Subject class implements the methods required for maintaining an list of all the observers, thereby providing the required methods to add (Observers), setObserverState(), getObserverState() and execute(). The execute() updates the Observer ArrayList which provides an update() method for the view to retrieve the updated results.

**Observer** class implments an abstract method update() which is extended by the InventoryProgramModel. This is called as a result of a completed change at the data store layer, once the results have been returned to the model as String buffers containing the results of the preceding method.

**InventoryItems - InventoryCD InventoryDVD InventoryBOOK**

**InventoryItems** implements an abstract class which is extended by the **InventoryCD**, **InventoryDVD**, **InventoryBOOK** classes. This provides the uniqueness that differentiate the data elements for the itemTypes in the data store. The data store table describes the data element specifics:

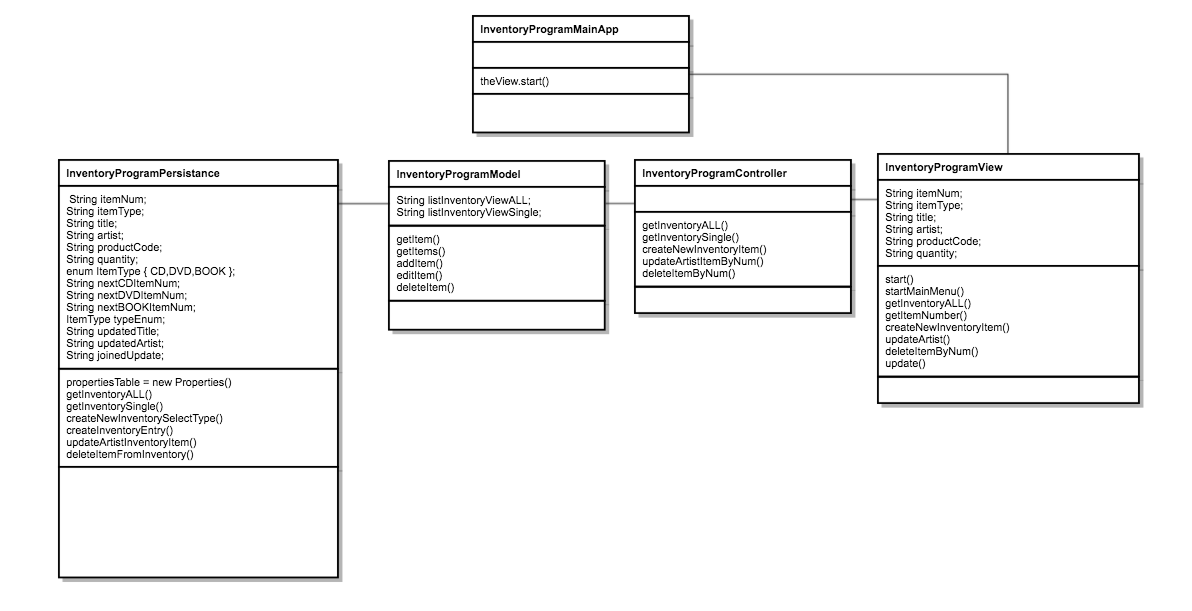
|  |  |  |  |
| --- | --- | --- | --- |
| ***Data Store Elements*** | ***InventoryCD*** | ***InventoryDVD*** | ***InventoryBOOK*** |
| title |  |  |  |
| artist | artist | studio | author |
| productCode | productCode | UPCCode | ISBN |
| quantity |  |  |  |
|  |  |  |  |

**ItemType**

**Enum ItemType** declares the elements CD, DVD, BOOK representing the media type used in the data store. The enum is used to differentiate between the submitted media types when creating a new inventory item.

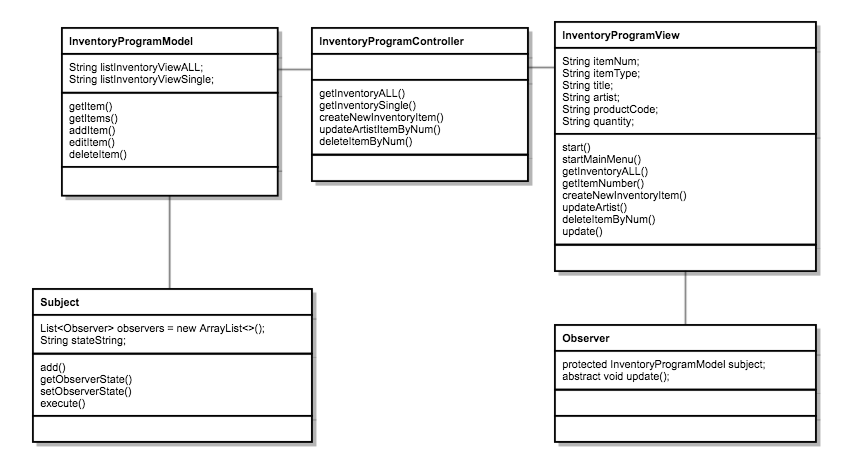
## Class Diagrams

**MVC Structure**



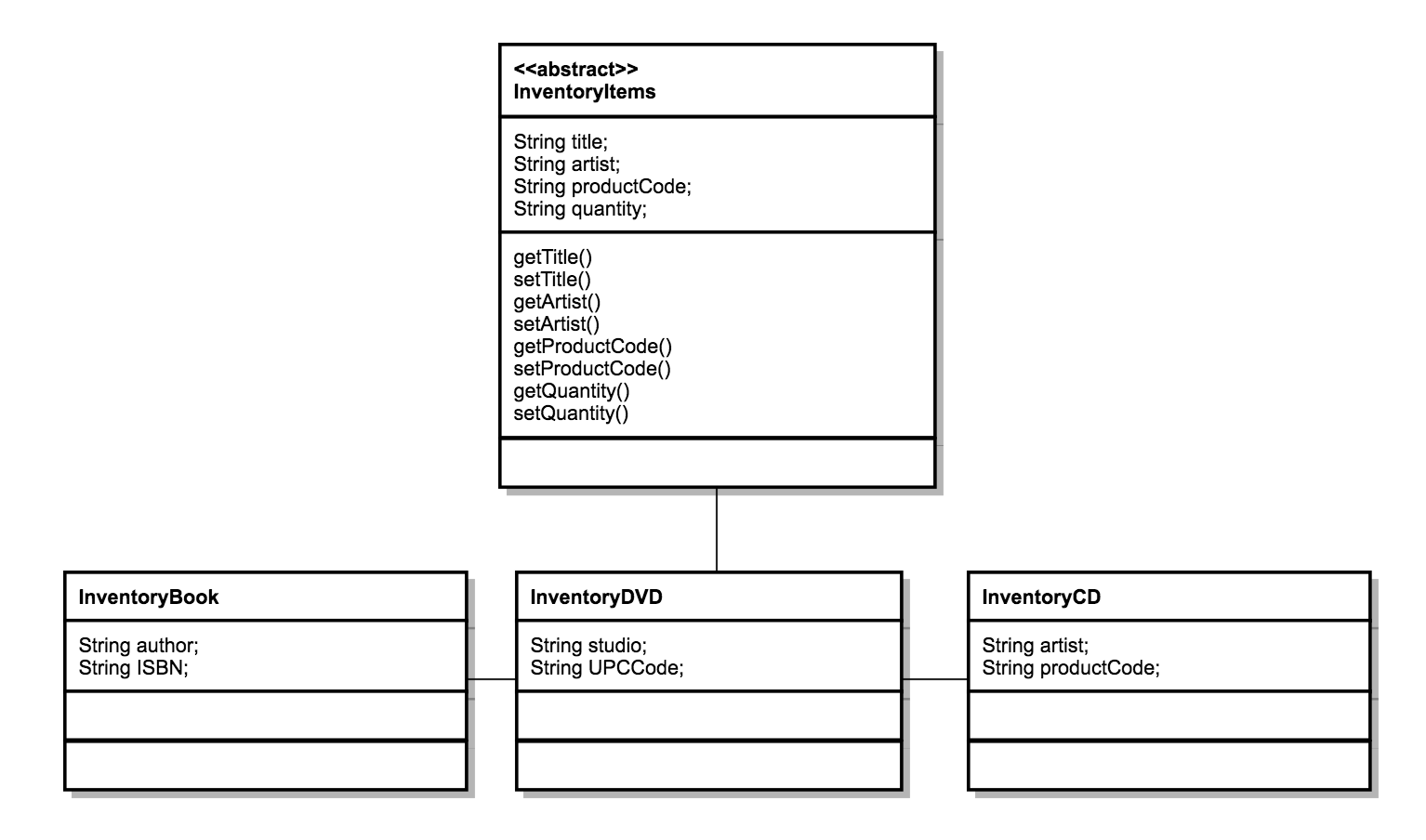
The Model–View–Controller (MVC) applies the software architectural pattern for implementing separated interfaces for this application. It divides the **InventoryProgram** application into three interconnected parts in order to separate internal representations of information from the ways that information is presented to and accepted from the user. The view component theView implements the text based System.in / System.out interface for presenting information to or obtaining information from the user. The additional component in this view provides the MVC implementation in relation to the persistence layer.

**Observer View Pattern**



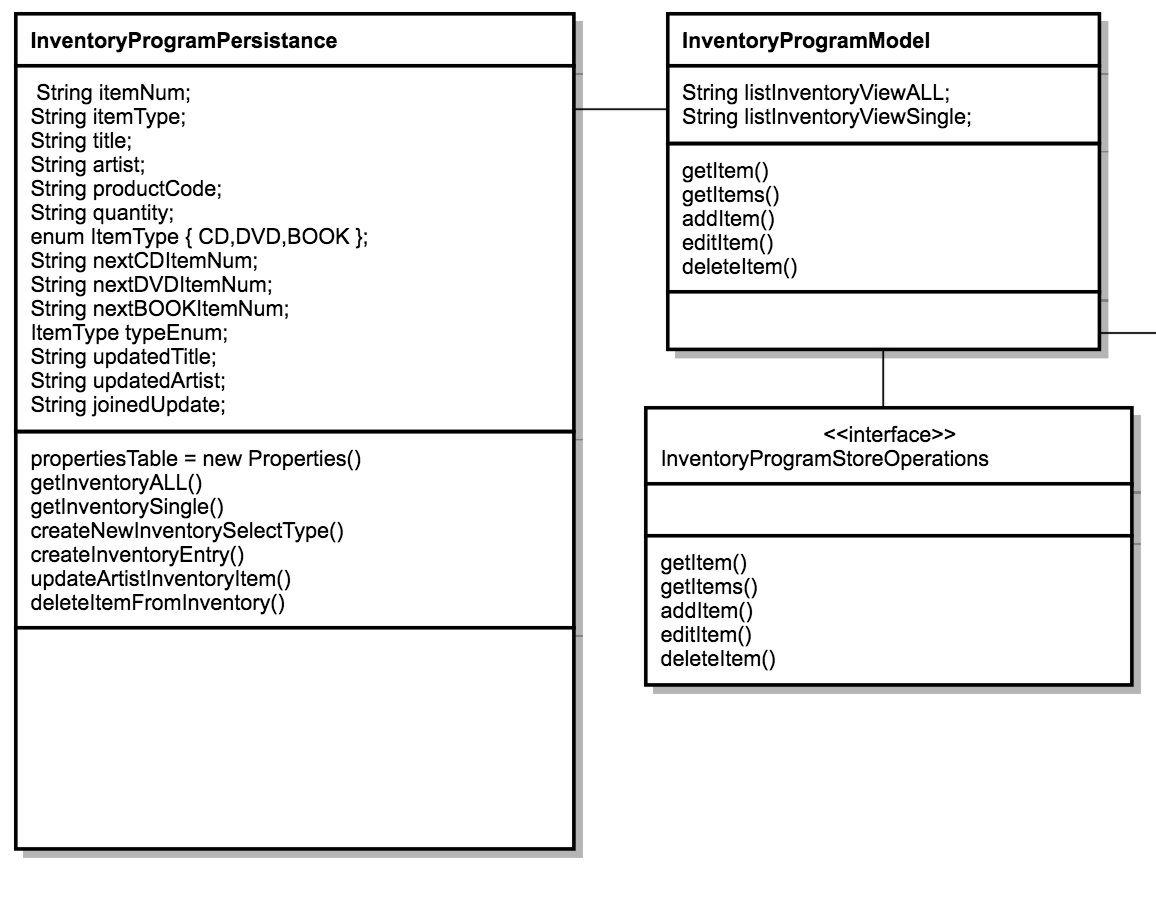
This **InventoryProgram** implements the Observer View Design Pattern, in that the implementation registers the “theView” in the Subject class’s array. Any time changes are applied to the data storage layer, the execute() method is executed which in-turn notifies the view through the update() that the results of recent activities has resulted in new data String Buffer data elements being available in the “theModel” instance variables as “InventoryViewALL” or “InventoryViewSingle” buffers.

## **Inventory Structure**



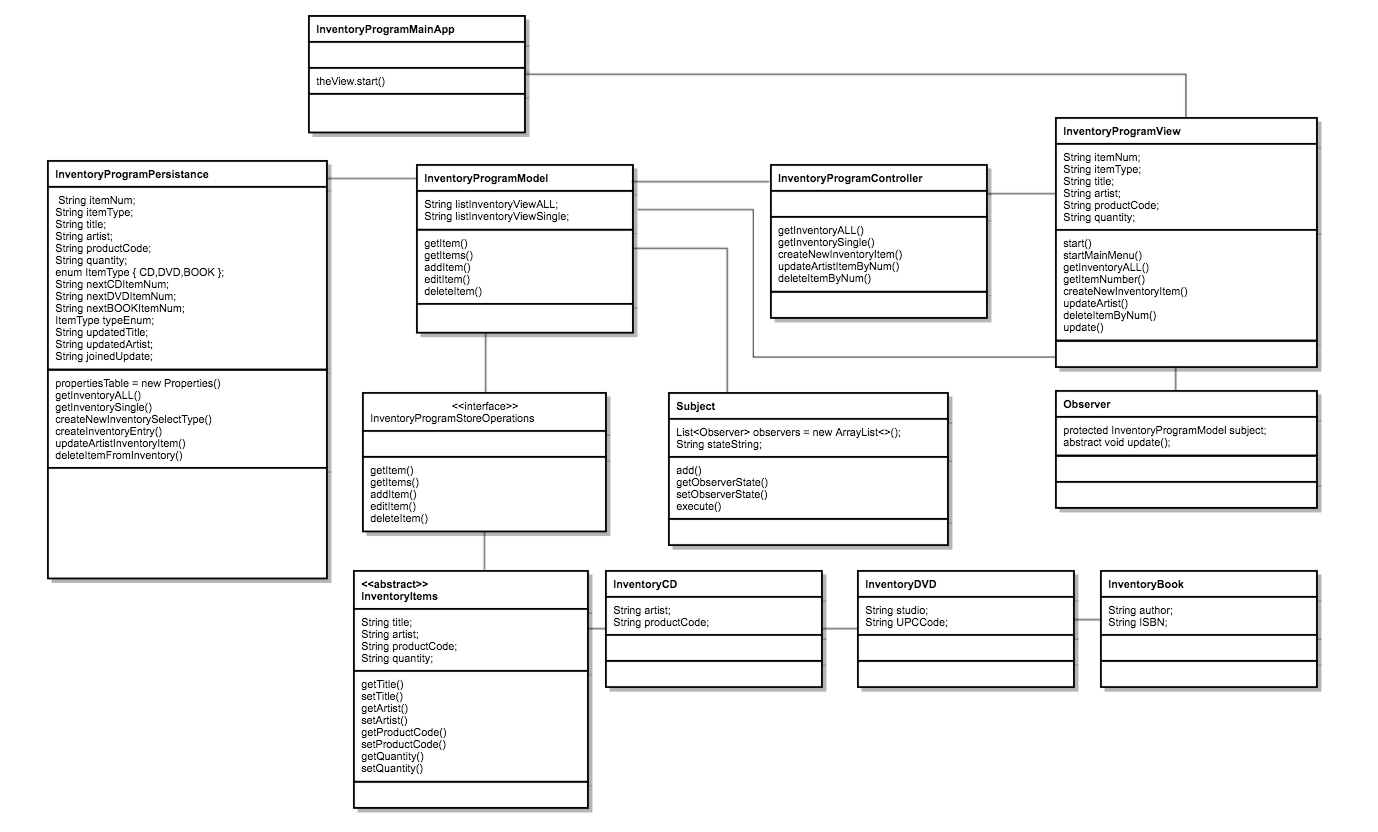
The **Inventory Structure** documents the polymorphic inheritance between the abstract InventoryItems class and the child classes for the CD, DVD, and BOOK classes. Each of the classes provides similar functions, while providing uniqueness for the individual data elements, for the creator of the item, and the corresponding numerical product registration strings.

**Persistence Layer**



The persistence layer shows the implementation of the related classes which provide the data inventory storage. This is the “theModel” of the MVC, which provides the interface to both the persistence layer for storing the data, and writing to the filesystem, adhering to the data model interface of the data operations through the **InventoryProgramStoreOperations** class

C**omplete Application Overview**

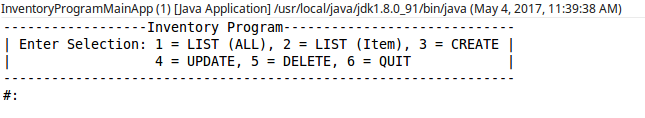


The above diagram, whilst inaccurate for detailing the classes, methods and data elements as a class diagram or data flows. It does however serves to provide an overview of how the different classes connect and what methods interface to each other.

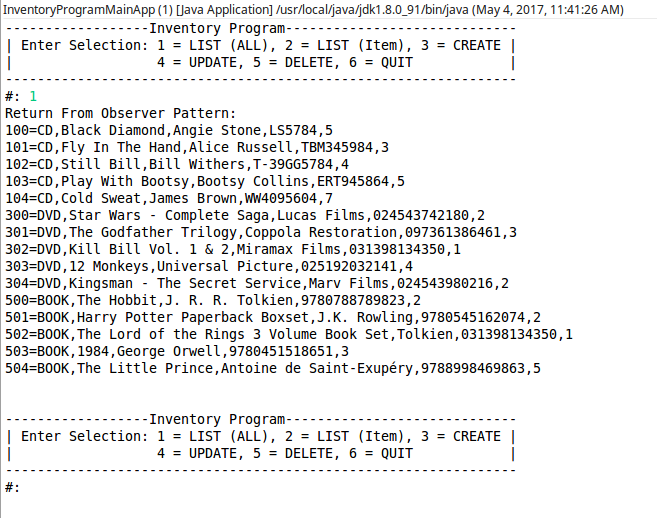
## Application Operations

The following provides screenshots and descriptions of the application operations

***Item 1: Main Menu***

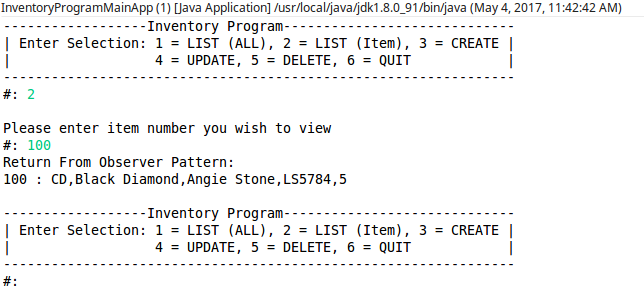
*Description:* Displays main menu

***Item 2: Option 1, LIST (ALL)***

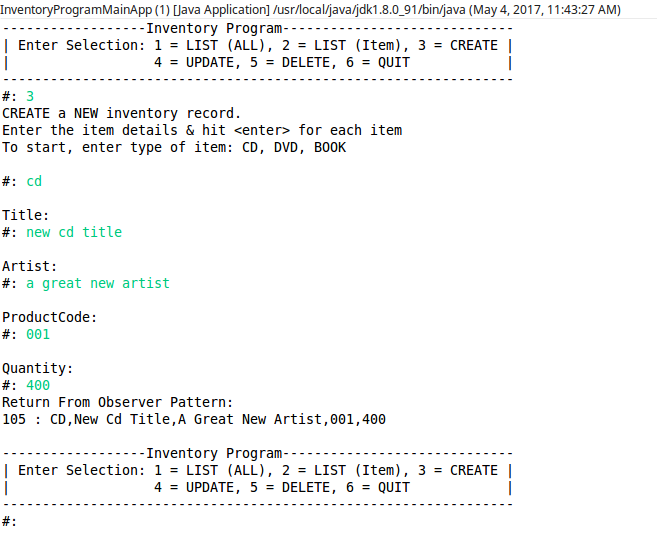


*Description:* Lists all items stored in properties file

***Item 3: Option 2, LIST (Item)***

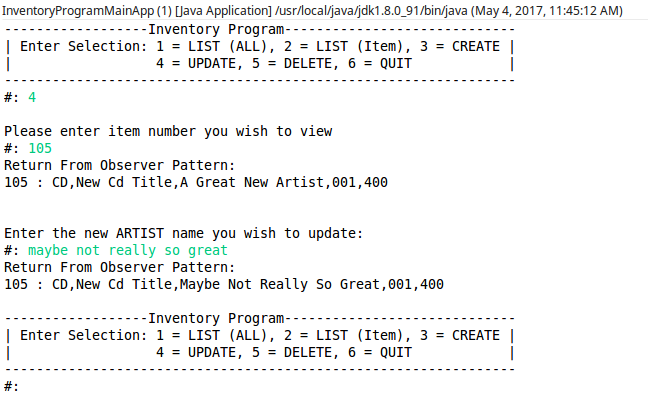
*Description:* Lists single item stored in properties file by selecting the item number from the main List (All) menu option

***Item 4: Option 3, CREATE***



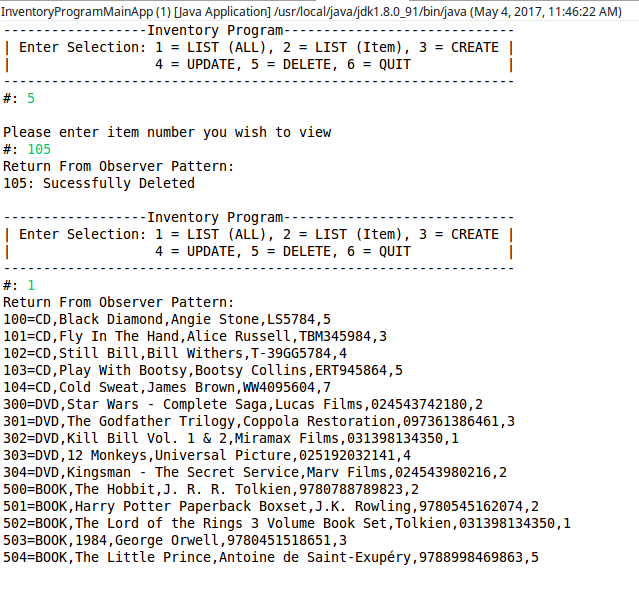
*Description:* Create item, entering “type” creates a unique record as either CD, DVD, BOOK

***Item 5: Option 4, UPDATE***



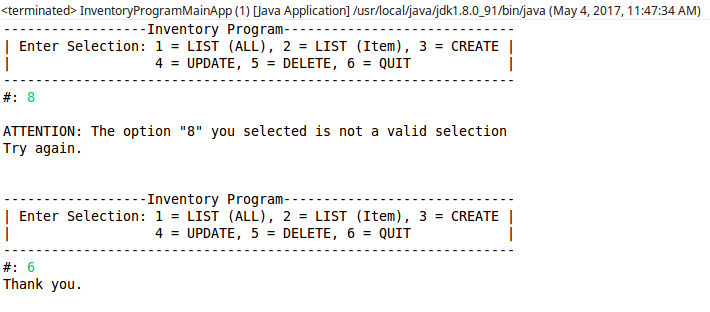
*Description:* Update item, by selecting item number

***Item 6: Option 5, DELETE***



*Description:* Delete item by selecting item number. Returns a “successful” message. Listing the data store shows item is no longer available.

***Item 7: Option 7, QUIT***



*Description:* Quit menu returns notification that you have ended the program with a “Thank you.”. Also submitting an invalid option selection returns an error and requests a user enter a valid option.