Assignment 3: Object-oriented design and Java

The goal of this assignment is to demonstrate your understanding of object-oriented design using Java. You are to create an implementation for the InventoryMan API (explained below) that provides services for managing an inventory of books and music for a flat (or any group of people). Unlike Assignment 2, the API is specified as a Java interface. Also unlike Assignment 2, there are no restrictions on what classes you use in your implementation. You are, however, required to provide an implementation that provides a reasonable use of polymorphism, enums, abstract classes, and checked exceptions.

You must submit all your source code and a short explanation of your design. More details are below.

Due	1 May 2359 (Confirmed)
Worth	9%
Submission	Assignment DropboxLinks to an external site., 1 archive file

Restrictions

Do not use Java 8 (or later) features, in particular Lambda expressions. Lambda expressions are for a style of design known as "functional programming". This course is about object-oriented design, not functional programming. Submissions that use Java 8 (or later) features will not be marked.

Updates

The intent is to provide a "realistic" system to implement within the constraints of a university course assignment. While I have endeavoured to describe everything I think you need to know, the complexity of the assignment may mean I've missed something, or some requirements are ambiguous. Feel free to seek clarification from me for anything you are unsure of (ideally referring to this text or the description of `inventoryman.InventoryMan`). Any changes or clarifications will be posted here.

- Date checking only requires that the date conforms to the required format (yyyy-mm-dd). Nothing more sophisticated is required. (20200407)

Client/Server architectures

Many systems now have so-called "client/server" architectures, that is, they are divided into two parts — the "client", and the "server". The client typically provides the user interface, that is, what the user interacts with. The server stores that data and does the main processing. The client and server communicate with each other, typically (but not required to be) over a network, and now usually its over the internet. Many mobile apps use this architecture, in part to reduce the load on the mobile device, but there are other advantages such as portability — if a new mobile device comes out it is much easier to recode just the client for the new device rather than recode the entire application.

In order for the client to communicate with the server, it must follow the rules that the server expects for communication. The rules make up the "interface" for the server.

An adequate (**but non authoritative**) description can be found on Wikipedia (Links to an external site.).

Application Programming Interface (API)

An Application Programming Interface (API) provides an abstraction for building a class of applications, meaning any instance of that class can be developed without having to understand the low-level details required for such applications. For example, the "Java API" supports Java programmers building Java applications. Using it means Java programs can be built much more quickly than if it were not available. In fact it is really a number of APIs for different *problem domains*, such as accessing file systems, creating graphical user interfaces, managing security, and so on.

In principle any API can have multiple implementations. For example, the Java API has an implementation by Oracle and there is also the open-source implementation Open JDK (Links to an external site.).

In a client/server system, the client communicates with the server through a defined interface. This means there can be multiple implementations of the server, provided they all understand the same interface. And so the interface the client expects and the server provides can also be thought of as an API (although it is not common to do so).

Assignment 2 was effectively building the server side for the visitor management system, where `visitorman.VisitorMan` provided the API for the services of the system, and any "client" (e.g. `Checker`) had to use this API to get those services. `VisitorMan` could have been a Java `interface` rather than an unimplemented class, except at the time that assignment came out you hadn't seen interfaces yet.

An adequate (**but non authoritative**) description can be found on Wikipedia (Links to an external site.)

Flat Inventory Manager

For this assignment, you must provide an implementation for an API that provides some services for managing an inventory of books and music for a flat (or any group of people). Unlike Assignment 2, the API is specified as a Java interface.

The reason for only implementing the API is because this way we avoid the complexities of how to provide a good user interface (which will be partially addressed in the second half of the course, and in more depth in SOFTENG350). The reason for implementing only some services is because even what seems like a small system can require a lot of development, so we need to scale what you have to do to fit in the time allocated to the assignment.

Requirements

Functionality

I have provided a **Java interface** `inventoryman.InventoryMan` for the server for the inventory management system. The functionality requirement for the assignment is to provide an implementation for this interface. The name of the class that implements this interface must be `inventoryman.InventoryManImpl`. The details of the required functionality are provided in the JavaDoc for this interface (`InventoryMan.html`).

In keeping with the intent that we are dealing with an interface that is for communication across a network, `inventoryman.InventoryMan` is expressed in terms of `String` or lists of `String`. This makes it a little awkward to do certain things (e.g. error reporting and handling). While an interface for a real server would not look exactly like this, it would have the some of the same awkwardness.

There are a number of possible error cases that could be checked for (e.g. an item is added that has the same creator, title, and format as an item already in the inventory, the cost is incorrect formatted, the format is invalid). *You are only responsible for the errors mentioned in the JavaDoc.*

Understanding of Object-Oriented Design

Your implementation must be a reasonable object-oriented design. Evidence for how reasonable your design is should be provided by describing the appropriate *domain model*, how your design relates to this model, and any other information you think will help demonstrate that your domain model is appropriate and that your design is consistent with it.

Other evidence you should provide is by demonstrating how your design sensibly uses polymorphism. You must explain where (i.e. where in the code) your implementation demonstrates polymorphism and explain how it does so.

Understanding of Java

Your implementation should show at least one reasonable use of each of the following Java features:

- `enum`
- abstract class
- checked exception

Resources

On Canvas (Files) you fill find the following

`InventoryMan.java`

The Java interface you are to implement. *Do not change this in any way*.

`InventoryMan.html`

The JavaDoc for the interface (generated from `InventoryMan.java`)

`Checker.java`

A client. This is provided for your convenience, to help you confirm your can execute your implementation, and can be used to provide some simple testing. It does not, however, provide complete testing. You may change this as much as you like, for example extending it to do more complete testing. You must not submit this.

The marking of your submission will be done partially by an automatic marking system that will test your implementation. It will access your implementation in a manner similar to this client. If the marking system cannot execute your implementation, you will not receive any marks for it.

Deliverables

You must

- implement `inventoryman.InventoryMan` with a class named `inventoryman.InventoryManImpl`
- demonstrate use of polymorphism
- demonstrate use of enums
- demonstrate use of abstract classes
- demonstrate use of checked exceptions
- describe the domain model for the problem domain on which your design is based and how you have met the requirements.

You should submit the following:

- Your source code, properly organised into the directory structure corresponding to the package structure. Do not include anything unnecessary such as .class files or temporary files. Do not just include your Eclipse project in your submission as that will include unnecessary files. Mac uses, learn how to remove (of avoid adding) `.DS_Store` folders and the like. If the markers cannot compile your code without having to make changes to what you have handed in, you could lose up to 25% of your mark.
- A report describing how you have met the requirements. You do not need to discuss what every class does, in fact just a description of every class will not be sufficient. What you should do is provide a description of the domain model and how your design relates to it. This does not have to be very elaborate. Just describe the main concepts, explain why they are the main concepts, and how those concepts are represented in your design, and how many objects you expect to be created (which may be presented in terms of how the API is used).

You report should also explain how you have used the require Java features (polymorphism, enums, abstract classes, checked exceptions) and justified their use.

I do not expect more than a page. Diagrams are acceptable.

Submitting the Assignment

Failure to following instructions may result in a mark of zero for the assignment.

These instructions are intended to reduce the cost of doing the marking so that that markers do not have to waste time trying to figure out what you have submitted and so can spend more time providing feedback. If you cannot follow instructions, you don't deserve to become an engineer.

The deliverables are due on via the Assignment DropboxLinks to an external site..

Submit exactly one file. Use only zip or tar (possibly with gzip) in creating this file. I recommend the following layout (assuming a PDF file for the report):

```
src/

  pkg1/

    Foo.java (for class pkg1.Foo)

  ... etc ...

softeng251_<username>_a02_report.pdf
```

That is, at the top level there if your report and a directory containing your source code. The source code must be organised according to your package structure so that it can be compiled without having to change anything. Your justification must be submitted in a format that is universally readable. For all practical purposes this means PDF (postscript is ok, as is ASCII. Word documents or any other format are not acceptable).

Assessment

This assignment is worth 9% of your overall grade for SOFTENG 251 and will be marked out of a total of 30 as follows:

Executable 12 marks

To get full marks, your submission must have no obvious faults. If the markers cannot figure out how to use your submission, you will lose marks. If your submission does not conform to what has been specified, you will lose marks. *Indicative* marks for individual parts of the interface are given in the JavaDoc.

Source code 4 marks

You are expected to follow good design and coding practises, as discussed in class. Your mark for this part will be based on an assessment as to how well you have done this. You must follow the code conventions used in the course and provide appropriate comments. If the markers cannot compile your submitted code, you will get zero.

Design 6 marks

The main criteria is whether your design shows you understand how to do object-oriented design. At minimum, this means having a bunch of classes from which an appropriate number of objects get created, and those objects then interact to get the desired result. Your report should explain how your design achieves this.

There is no one "perfect" design and so there are any number of designs that will get you full marks. There are, however, also any number of poor designs that will not get full marks. Characteristics of such poor designs are:

- A design that does not match the problem domain very well.
- A single class (or very small number of classes). There are plenty of reasonable classes that could be used to meet the requirements. You should have no difficulty creating a design with more than 5 classes.
- Overuse of static. While there are some limited situations where static makes sense (main and perhaps one or two others), it can also be a sign of not understanding object-oriented design as it means not needing objects. If you use static, when quite clearly a non-static option existed, then it will count against you.
- Classes that have responsibilities that are not related to each other, or classes where it is not clear what abstraction they represent.

Part of the assessment of your design will be based on how well you have justified your design decisions. You may choose a design that (for example) the markers do not like, but if you explain your design decisions well you can still get full marks. However, a design that the markers agree with but which is not well justified may not get full marks.

Java features 8 marks

You must demonstrate appropriate use of:

- Polymorphism (4 marks)
- enum (1 mark)
- abstract class (1 mark)
- checked exception (2 marks)

Your report should explain how you have met this requirement.

Final Comments

Other things to think about:

  - Design choices
    - What likely classes will you need?
    - What state do you need to keep track of?
    - How will you represent this state?
    - What are all the scenarios you need to deal with?
    - What cases will you need to test?
    - How will you do the testing?
    - How can you make the testing easier?
  - You are also entirely welcome to use anything from the Java Standard Library. Classes and interfaces that are worth looking at include:
    - java.lang.String
    - java.util.List
    - java.util.Set
    - java.util.Map
    - java.util.ArrayList
    - java.util.HashSet
    - java.util.HashMap

Use of any of the date or calendar support provided in the Java Standard Library is **strongly discouraged**. You are likely to spend more time trying to figure out how to make it work than if you implemented what you need yourself.