2021-2022 Semester 2

CT548 Object Oriented Design and Development

Individual Assignment 4

Objective	Design and implement using a test-driven development approach an object-oriented solution to a challenge involving design patterns.
Lecturer	Dr Adrian Clear Email: adrian.clear@nuigalway.ie
Marks Awarded	This assignment is marked out of **30 marks** and is worth **30%** of the overall grade for the module.
Submission	The deadline for submission is **23:59** on ~~**Thursday, March 31st**~~ **Friday, April 1st, 2022**. Your attempt should be submitted through Blackboard. Please provide a zip file including • A Jar file containing your solution to Task 3 and 4 (see instructions in Appendix 2 below). • A Word or PDF file containing your design documents (Task 1 and 2) and answers to questions in Task 5.
Late Submission or No Submission Policy	Late assignments will incur a **penalty** depending on the delay, as follows: - 96% if submission is within 1 hour of the deadline - 92% if submission is between 1-3 hrs after the deadline - 80% pts if submission is between 3-24 hrs after the deadline - 60% pts if submission is between 24-48 hrs after the deadline after 48 hrs: submissions not accepted (no points awarded) After **48 hours** past the deadline, submissions will no longer be accepted. Multiple submissions are possible. Note that the **last submission will be graded** whether submitted before or after the deadline.

Task description

Your task is to design and implement a solution in Java to the challenge below using good practices of object-oriented development. Your code should be fully documented with Javadoc. Your Javadoc should describe the responsibilities of each class and describe each method. You are required to:

1. Create **class diagram** based on the challenge description below.
2. Create **a sequence diagram** for realising the following use case:
    a. **adding** an entry to the bibliography
3. Use test-driven development. At a minimum, use equivalence and boundary testing to create **unit test cases** for
    a. Validating a digital object identifier (doi) of an academic article (see requirements for this in the next section)
    b. Loading a bibliography from a BibTeX file (note that you do not need to implement loading from file functionality but you should have test cases for this scenario)
4. Develop an **object-oriented Java solution** to the challenge described below.
5. Explain the significance of the factory pattern **and** the strategy pattern in the solution. What changes would be required to extend the system to support a console-based interface instead of a GUI-based one? What changes would be required to extend the system to include a new type of publication (e.g. a conference publication that includes a "proceedings" field)?

Challenge: a bibliography management system

The application

Your challenge is to implement **a bibliography management tool** in Java which stores details about academic publications. A bibliography can be composed of many different types of publications. The publication types that your application is required to support are academic books, journal articles, and technical reports. Details of these different kinds of publications can be found in Appendix 1. However, many other types are possible and so the application should be extensible to new types being added.

GUI interface

This version of the software has a GUI-based user interface. When you run your project, the user should be presented with GUI elements to perform the following use cases:

```
- Load a BibTeX file to the bibliography
- Save your bibliography to a file
- Add an entry to the bibliography
- Delete an entry from the bibliography
- View the bibliography entries
- Quit
```

Adding an item: This functionality should support the specification of which publication types to add to the bibliography, i.e., : `article`, `techreport`, or `book`

Depending on their selection of "article", "techreport", or "book", the application should allow the user to enter the details required (documented in Appendix 1) using an appropriate form, implemented using a modal Dialog[1], and store them appropriately.

Deleting and item: This functionality should support the specification of a cite-key for a publication to delete (note: a cite-key is the first field following the "{" in the BibTeX representation. For the example in 1.1, the cite-key is "`hawking1988`"). If the cite-key corresponds to a publication stored in the application, it should be removed.

Viewing the bibliography: If this option is chosen, all of the publications should be printed in Harvard-style format. You have the option of printing them using GUI elements or to the console. Each publication should be separated by a blank line and they should be printed in order of the year they were published, with the most recent publication being printed first.

Saving to file: This functionality should support the specification of a file name. All of the publications stored in the system should then be written to a file of that name. The contents of the file should be in BibTeX format, with a blank line between each publication.

Loading from a file: You're not required to implement the loading of contents from a file but you should test the scenario that a user loads from a file by creating a stub from an interface called BibFileReader that has one method:

```
public void readFromFile(String filename);
```

Quit: If this option is chosen, the application should terminate.

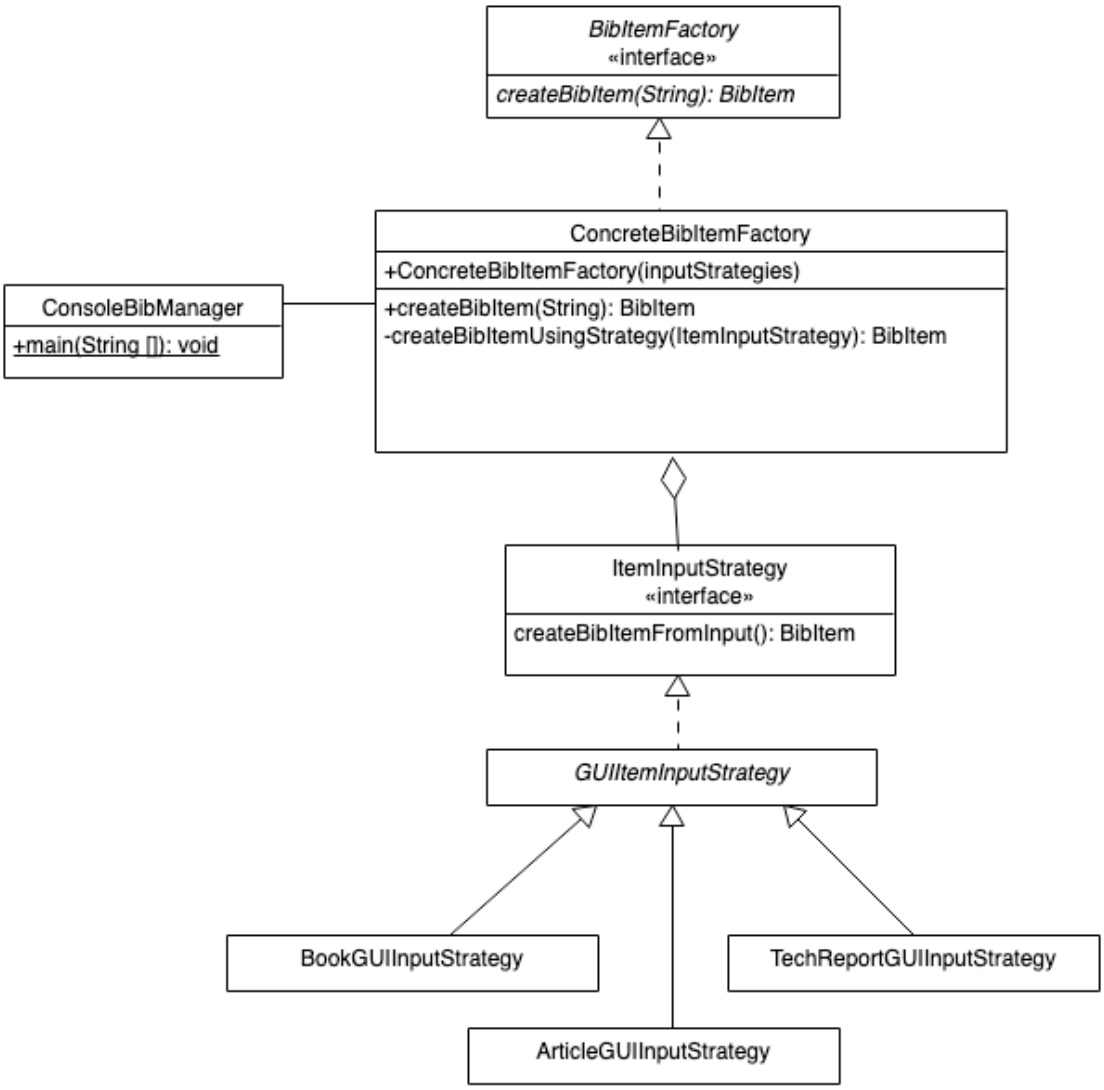
Bibliography entry requirements

[1] See https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html for an overview of Java Dialogs.

- All of the fields in the publication types should be non-null
- Apart from the year and the doi in academic articles, there are no further constraints on the contents of these.
- The year should be a number.
- The doi string for an article should be prefixed with "`https://doi.org/`". The rest of the string consists of two parts separated by a "/". The part before the "/" consists of two numbers separated by a ".". The first number is exactly two digits long and the second number is exactly four digits long. The part after the "/" should be non-empty. An example doi string is `https://doi.org/10.1007/3-540-47910-4_21`

Design

- Your application will need to represent each different type of publication. A publication type should be responsible for generating both a Harvard-style representation and a BibTeX representation of itself.
- The application should store publications in a data structure that that allows for efficient removal of items by their cite-key and sorts them according to the year they were published.
- The class that maintains the data structure to store publications should be a singleton.
- The creation of publications should be handled by a single Factory class.
- The three types of publication types in Appendix 1 are only some of the types of publications that can be represented in BibTeX. Use the strategy pattern along with the Factory pattern to make your application open and extensible to other possible types of publication.
- Hint: the UML diagram below illustrates the application of the factory pattern and strategy pattern for the application (note that this is only a partial diagram of the application in order to illustrate the patterns). In the diagram, GUIBibManager is the main class where the application is run. It is associated with a particular instance of ConcreteBibItemFactory that is used to create new bibliography items from user input. The factory is associated with an ItemInputStrategy for each of the publication types it can create and it determines the correct strategy to use to input the bibliography item depending on the publication type.


BibItemFactory
«interface»
createBibItem(String): BibItem
ConcreteBibItemFactory
+ConcreteBibItemFactory(inputStrategies)
+createBibItem(String): BibItem
-createBibItemUsingStrategy(ItemInputStrategy): BibItem
ConsoleBibManager
+main(String []): void
ItemInputStrategy
«interface»
createBibItemFromInput(): BibItem
GUIItemInputStrategy
BookGUIInputStrategy
TechReportGUIInputStrategy
ArticleGUIInputStrategy

Appendix 1: Publication types

The application is required to support three types of publications. Two different formats are shown for each publication type: a Harvard style print-friendly version that can be used for adding to academic reports written in a word processor like MS Word; and a BibTeX version which is a style-independent format used for storing references and typically used with the LaTeX document preparation system.

A1.1 Academic books

An example of an academic book in Harvard style:

```
Hawking, Stephen. (1988). A Brief History of Time: From the Big Bang to
Black Holes. Bantam.
```

An example of an academic book in the BibTeX format:

```
@book{hawking1988,
author    = "Hawking, Stephen",
title     = "A Brief History of Time: From the Big Bang to Black
Holes",
year      = 1988,
publisher = "Bantam"
}
```

A1.2 Academic articles

An example of an academic article in Harvard style:

```
Gamma, Erich and Helm, Richard and Johnson, Ralph and Vlissides,
John. (1993). Design Patterns: Abstraction and Reuse of Object-
Oriented Design. ECOOP' 93 --- Object-Oriented Programming.
https://doi.org/10.1007/3-540-47910-4_21.
```

An example of an academic article in the BibTeX format:

```
@article{gamma93ecoop,
author="Gamma, Erich and Helm, Richard and Johnson, Ralph and
Vlissides, John",
title="Design Patterns: Abstraction and Reuse of Object-Oriented
Design",
year="1993",
journal="ECOOP' 93 --- Object-Oriented Programming",
doi=" https://doi.org/10.1007/3-540-47910-4_21"
}
```

A1.3 Technical reports

An example of a technical report in Harvard style:

> Cummins, Ronan and O'Riordan, Colm. (2004). Evolving, Analysing and Improving Global Term-Weighting Schemes in Information Retrieval. National University of Ireland, Galway.

An example of a technical report in the BibTeX format:

```
@techreport{Cummins:2004:071204,
author = "Cummins, Ronan and O'Riordan, Colm",
title = "Evolving, Analysing and Improving Global Term-Weighting Schemes in Information Retrieval",
year = "2004",
institution = "National University of Ireland, Galway"
}
```

Appendix 2: Creating a Jar file in Eclipse.

1. Right-click the source (“src”) folder of your project and select “Export”.
2. In the dialog box select Java → JAR File (not runnable JAR file). Click Next.
3. Make sure your project folder is checked. You can uncheck the .classpath and .project files. Make sure the “Expore Java source files and resources” is checked.
4. Click Browse and select a location and name for your Jar file.
5. Click Finish.

You can check it worked correctly by importing your Jar file to a new project.

1. Create a new Java Project.
2. Right-click the “src” folder and click “import”.
3. Select General → Archive file.
4. Click Browse and locate your Jar file.
5. Click Finish.
6. Make sure your files and packages have been correctly imported in the Package Explorer.