CSCI-605 Advanced Object-Oriented Programming Concepts
Homework 7 - Zipper

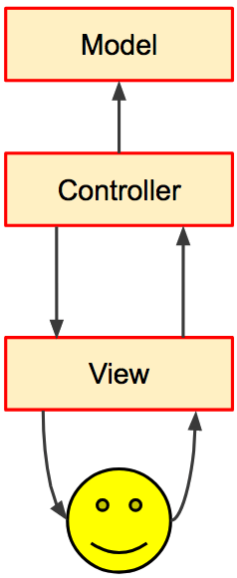
1 Introduction

For this homework you will be writing a simple command line interface (CLI) for a utility that allows users to compress and uncompress archives using ZIP compression. For this homework, you will be responsible for reading the online documentation and learning about the `java.util.ZipOutputStream` and `java.util.ZipInputStream` classes.

When properly implemented, the ZIP files created by your application should work with the standard ZIP utility provided by your operating system, e.g. on a Windows computer you should be able to right click the file and choose the "Extract All..." option from the context menu. Conversely, your application should be able to extract files from ZIP files created by your operating system, e.g. on a Windows computer, if you right click a file and select the "Send to - Compressed (zipped) folder" option, your program will be able to unarchive the file.

So far this semester, when you have used a combination of standard output (System.out) and standard input (Scanner and System.in) to implement user interfaces, usually the code that implements your program's core application logic (the algorithms) and the code the implements the user interface is all mixed up together. This means that if you were to write a different user interface for the same program, you would need to copy or rewrite much of the core application logic.

1.1 Model View Controller



The Model View Controller architectural pattern is a very famous way of organizing your code so that the core application logic (here called the model) and the user interface (here called the view) don't get mixed up together. Implementing the MVC pattern is conceptually easy, but can be difficult in practice.

- **The Model** contains all of the classes that include your core application logic and never includes classes that the user sees or interacts with. For command line interfaces this means your model should not make any calls to `System.out` or `System.in` (except maybe for logging and/or debugging purposes).
- **The View** contains all of the code to present output to the user and to collect input from the user. For command line interfaces, here is where you will use `System.out` to present output directly to the user and `System.in`/`Scanner` to collect input from the user. The user should only ever directly interact with classes in your model.

- **The Controller** is one or more classes that sit between the View and the model. It acts as a layer of separation that keeps the classes in the model and the classes in the view from directly interacting with each other. Each time the user enters a command through the view, the view will make a call to one or more methods on the controller. The controller will in turn make one or more calls on the model. If there is any result or feedback from the model, the controller returns it to the view. The view can then display any necessary output to the user.

For this homework, you will be using the MVC architectural pattern to separate your classes into a model, a view, and a controller. Each MVC component may include one or more classes. In this case you are likely to have several classes in your model, but perhaps only one in your controller and view.

1.2 Provided Files

1. `data` - a folder containing example (images and text) files for you to practice zipping and unzipping with your program.
2. `output` - a folder containing input and output examples. The files named `exampleXX.txt` contain examples of user interaction through the command line interface. Your output should match these files as closely as possible. The files named `commandsXX.txt` contain only the commands used in the example files.

2 Input Specification

Your implementation must support the following commands entered by the user via the command line:

- `help` - displays a list of available commands.
- `add <file path>` - add file to the list of files to be archived when the archive command is executed.
- `archive <file path>` - creates the archive in the specified file and adds each added file as a separate entry.
- `clear` - clears the list of files to archive.
- `files` - prints a list of the files to be archived.
- `list <file path>` - displays a list of files in the specified directory.
- `quit` - quits the Zipper Utility.
- `unarchive <source path> <destination path>` - extracts the archive at the source path and saves the entries into the directory at the destination path.

What follows is an example of a short interactive session.

```
enter command >> list data
Listing files in 'data'...
  D:\ClassMaterials\605Materials\HW\Files\data\books.png
  D:\ClassMaterials\605Materials\HW\Files\data\buttercup.jpg
  D:\ClassMaterials\605Materials\HW\Files\data\cutie.jpg
```

```
  D:\ClassMaterials\605Materials\HW\Files\data\tacos.jpg
  D:\ClassMaterials\605Materials\HW\Files\data\words.txt

enter command >> add data/books.png
data/books.png added to list of files to be archived.
enter command >> add data/buttercup.jpg
data/buttercup.jpg added to list of files to be archived.
enter command >> files
Files to be archived:
  D:\ClassMaterials\605Materials\HW\Files\data\books.png
  D:\ClassMaterials\605Materials\HW\Files\data\buttercup.jpg
enter command >> archive zipper.zip
Archive successfully created: zipper.zip(6387114 bytes)
enter command >> unarchive zipper.zip temp
Extracting the archive 'zipper.zip' and saving entries in directory 'temp'...
  D:\ClassMaterials\605Materials\HW\Files\temp\books.png
  D:\ClassMaterials\605Materials\HW\Files\temp\buttercup.jpg
enter command >> list temp
Listing files in 'temp'...
  D:\ClassMaterials\605Materials\HW\Files\temp\books.png
  D:\ClassMaterials\605Materials\HW\Files\temp\buttercup.jpg

enter command >> quit
Quitting...
```

See the `output` folder in the provided files for additional examples.

3 Architecture

Your implementation will be required to adhere to the *Model-View-Controller* architectural pattern. This section includes more specific details, requirements, and suggestions.

- **Model** - Your model package should contain the classes that implement your application's core logic and are responsible for maintaining the state of the application. The contents of this package should include at least the following at a *minimum*:
  - `ZipperException` - A custom *checked* exception. This is the only exception that the classes in your model should explicitly throw. Many of the classes that you use will throw `IOException` (or one of its subclasses). You should catch and rethrow any of these exceptions as an instance of your custom exception.
  - `Zipper` - A class that is responsible for creating ZIP archives. You will need to use the `java.util.ZipOutputStream` class to help you.
  - `Unzipper` - A class that is responsible for extracting ZIP archives into a directory. You will need to use the `java.util.ZipInputStream` class to help you.
- **View** - Your view package should contain the class(es) needed to implement your user interface. The class(es) in this package are solely responsible for prompting the user, reading any user input (commands entered via `System.in`), and providing any output to the user (through `System.out` and `System.err`). Your user interface should handle

any exceptions that occur without crashing your program; instead, provide the user with appropriate output messages and continue running. The view should contain at least a class named `ZipperUtility`, which will act as the main class for your user interface.

- **Controller** - Your controller package should contain the class(es) that facilitate communication between the view and the model. The model in your application should include several classes when it is finished; you may think of the controller as a simplified interface to your model. The model and the view should not directly depend on each other, but should instead communicate through the controller.

4 Submission

You will need to submit all of your code to the MyCourses assignment before the due date. You must submit your `hw7` folder as a ZIP archive named "`hw7.zip`" (if you submit another format, such as 7-Zip, WinRAR, or Tar, you will not receive credit for this homework). Moreover, the zip file must contain only the java files of your solution. Do not submit the .txt files provided in this homework nor compiled files.

5 Grading

The grade breakdown for this homework is as follows:

- MVC Design 20%
- Functionality 70%
  - Custom Exception Implementation 10%
  - Zipping/Compressing 30%
  - Unzipping/Uncompressing 30%
- Testing 10%