CS410J Project 1: Designing an Appointment Book Application (7 points[1])

In this project you will create the fundamental `Appointment` and `AppointmentBook` classes that you will work with for the duration of the course.

Goals: Extend classes that you did not write and perform more complex command line parsing

The `edu.pdx.cs410J` package contains two abstract classes, `AbstractAppointment` and `AbstractAppointmentBook`. For this project you will write two concrete classes in your `edu.pdx.cs410J.`*login* package: `Appointment` that extends `AbstractAppointment` and `AppointmentBook` that extends `AbstractAppointmentBook`[2]. Each of your classes must implement all of the abstract methods of its superclass.

An `AppointmentBook` belongs to a particular owner and consists of multiple `Appointment`s. An `Appointment` has some description (such as "Have lunch with Lisa"), begins at a given time[3] and ends at a given time. For this assignment, all of this data should be modeled with `String`s. You may ignore the `getBeginTime` and `getEndTime` methods.

You should also create a `Project1` class that contains a `main` method that parses the command line, creates an `AppointmentBook` and an `Appointment` as specified by the command line, adds the `Appointment` to the `AppointmentBook`, and optionally prints a description of the `Appointment` by invoking its `toString` method[4]. Your `Project1` class should have the following command line interface[5]:

```
usage: java edu.pdx.cs410J.<login-id>.Project1 [options] <args>
  args are (in this order):
    owner                   The person whose owns the appt book
    description             A description of the appointment
    beginTime               When the appt begins (24-hour time)
    endTime                 When the appt ends (24-hour time)
  options are (options may appear in any order):
    -print                  Prints a description of the new appointment
    -README                 Prints a README for this project and exits
  Date and time should be in the format: mm/dd/yyyy hh:mm
```

Note that multi-word arguments should be delimited by double quotes. For instance the `owner` argument could be `"Brian Griffin"`. However, the dates and times should **not** be quoted (they are two separate command line arguments). The following dates and times are valid: `7/15/2016 14:39` and `06/2/2016 1:03`[6].

[1] 6 for code, 1 for POA

[2] Be aware that you should **not** modify any of my code. When I test your code I will use my version of the code, not yours. In fact, the `Submit` program will not allow you to submit my code. Remember that the `Submit` program can submit more than one file at a time.

[3] Your program should accept times and dates that have already occurred.

[4] Note that `Appointment`'s `toString` method is inherited from `AbstractAppointment`. You do not need to override it.

[5] You can learn more about the `README` option in the "Documenting Your Code for CS410J" handout on the course's website.

[6] That is, the month and the day can be expressed as either 1 or 2 digits. The year should always be four digits.

Error handling: Your program should exit "gracefully" with a user-friendly error message under all reasonable error conditions. Examples of such conditions include

- Something is missing from the command line or there are extraneous command line arguments
- The format of the day or time is incorrect or the description is empty

The class files for classes in the `edu.pdx.cs410J` package can be found in `/u/whitlock/jars/cs410J.jar`

You should submit `Project1.java`, `Appointment.java`, and `AppointmentBook.java` using the submit program. You can learn more about the `Submit` program in the "Instructions for submitting projects for CS410J" handout on the course's website.

To get you started with the project, there is a Maven archetype for the Appointment Book project.

```
$ mvn archetype:generate \
  -DarchetypeCatalog=https://dl.bintray.com/davidwhitlock/maven/ \
  -DarchetypeGroupId=edu.pdx.cs410J \
  -DarchetypeArtifactId=apptbook-archetype
Define value for groupId: : edu.pdx.cs410J.<login-id>
Define value for artifactId: : apptbook
Define value for version:  1.0-SNAPSHOT: :
Define value for package:  edu.pdx.cs410J.<login-id>: :
Confirm properties configuration:
groupId: edu.pdx.cs410J.<login-id>
artifactId: apptbook
version: 1.0-SNAPSHOT
package: edu.pdx.cs410J.<login-id>
 Y: : Y
```

The archetype creates the `Project1` class and a class for testing it, `Project1Test`

```
+- apptbook/
   +- pom.xml  (Dependencies and reporting configuration)
   +- src/
      +- main/  (program source code)
         +- java/
            +- edu/pdx/cs410J/login-id/
               +- Appointment.java
               +- Project1.java
         +- javadoc/ (files for JavaDoc)
            +- edu/pdx/cs410J/login-id/
               +- package.html
      +- test/  (unit tests)
         +- java/
            +- edu/pdx/cs410J/login-id/
               +- AppointmentTest.html
         +- javadoc/ (files for test JavaDoc)
            +- edu/pdx/cs410J/login-id/
               +- package.html
```

```
      +- it/  (integration tests)
         +- java/
            +- edu/pdx/cs410J/login-id/
               +- Project1IT.java
```

The project should compile and run out-of-the-box. The ‘verify‘ phase compiles all of the source code, runs the unit tests, creates the jar file, and runs the integration tests.

```
$ mvn verify
```

The archetype configures a bunch of cool reports to run against your project.

```
$ mvn site
```

Open `target/site/index.html` and view the reports generated for your project.

The jar file created by the archetype is an “executable jar” that runs your `Project1` main class.

```
$ java -jar target/apptbook-1.0-SNAPSHOT.jar -README
```