Assignment 1

Aims:

- Practice how to apply a systematic object-oriented design process
- Gain experience in implementing an object-oriented program with multiple interacting classes
- Learn more about the Java class libraries

Due Date: Break Week, Friday, April 6, 11:59 p.m.

Value: 10%

Cinema Booking System

In this assignment, you will implement a prototype system that could serve as the "back-end" for a cinema booking system. Customers can make, change and delete cinema bookings. Each booking has an ID number and is made for a given number of **adjacent** seats in a particular cinema at a particular time (assume all times are in the format HH:MM and they are all on the same day). A booking request is either granted in full or is completely rejected by the system; there are no bookings partially filled.

Assessment will be based on the design of your program in addition to correctness. You should submit at least a UML class diagram used for the design of your program, i.e. not generated from code afterwards.

All input will be a sequence of lines of the following form, where, in addition, a comment (starting with a '#' character) can appear at the end of a line. Your program should be able to process and discard such comments (this includes whole lines that consist only of a comment).

```
Cinema <cinema> <row> <seats>
        # specify that cinema <cinema> has a <row> row that has <seats> seats
Session <cinema> <time> <movie>
        # specify that there is a session showing <movie> in <cinema> at <time>
Request <id> <cinema> <time> <tickets>
        # booking request <id> is for <cinema> at <time> for <tickets> tickets
Change <id> <cinema> <time> <tickets>
        # change booking <id> to be for <cinema> at <time> for <tickets> tickets
Cancel <id>
        # cancel booking <id> (if it exists) and free up seats
Print <cinema> <time>
        # print record of all bookings for <cinema> at <time>
```

All cinema seat rows and showings will be declared before any other commands are issued. Rows in cinemas will be declared in order (from the front of the cinema) and will have a single word alphabetic name such as "A", "B", "AA", etc. (though rows may not always start at "A"). Seats in each row are numbered starting from 1 up to the number of seats in the row. Thus each seat has a unique name (for that cinema) such as "A1", "B2", etc. Sessions are input after all the cinemas have been declared, and may be given in any order.

A booking is for a specified number of seats in a given session of a given cinema (assume the session and cinema are valid). The system should make a booking if it is possible to assign that number of seats **together** in any row of the cinema. To remove ambiguity, booking requests and changes are fulfilled as follows: each row is checked (in the order in which they appear in the input file) to determine if there are sufficient adjacent empty seats to satisfy the request, and if so the first available seats (starting from seat 1) are assigned to the booking. Note that all seats must be from the same row in order to satisfy a request. The output for a booking or change should give the range of seats assigned to the new booking. Printing the bookings in a session should output the name of the film followed by the bookings on each row. See below for more detail.

Create all your Java source files in the **default package** (**not** a package like "ass1"). Call your main Java file `CinemaBookingSystem.java`. Read input from a file whose name is passed as an argument to the main method in the call to `java CinemaBookingSystem` and print output to `System.out`. For machine marking, the output will be redirected to a text file that will be compared to the expected output (so do not print out extra spaces or lines, etc.) **and remember to close the input file**. You can assume that booking ids are unique. For bookings and changes, print out `Booking` or `Change`, followed by the ID and the seat range of the new booking (the start and end seat separated by '-', or just the seat name if there is only one seat). If a booking request cannot be fulfilled, print out `Booking rejected`, for a change that cannot be made, `Change rejected`, and for a cancellation that cannot be done, `Cancel rejected`. Printing the bookings in a session should output the name of the film on one line followed by the bookings for each row on separate lines, with the row name followed by ':' and the ranges of seat numbers booked (or just the seat number if there is only one seat) separated by commas. See the examples below.

To read input from a text file (whose name should be passed as a **command line** argument to java, e.g. `java CinemaBookingSystem input.txt`), use code such as:

```
Scanner sc = null;
try
{
    sc = new Scanner(new File(args[0]));    // args[0] is the first command line argument
    // Read input from the scanner here
}
catch (FileNotFoundException e)
{
    System.out.println(e.getMessage());
}
finally
{
    if (sc != null) sc.close();
```

```
}
```

Sample Input

Below is an example of the input form and meaning. Note that you will have to submit at least three input test files with your assignment. These test files should include one or more comments to specify what scenario is being tested (see Requests 2, 3 and 5 for illustration). However, the following sample input includes many comments added only to explain the input format; your input test files do not need to contain this many comments.

```
Cinema 1 A 15  # Row A of cinema 1 has 15 seats
Cinema 1 B 20  # Row B of cinema 1 has 20 seats
Cinema 2 B 5   # Row B of cinema 2 has 5 seats
Cinema 2 X 5   # Row X of cinema 2 has 5 seats
Session 1 09:00 Toy Story    # Toy Story is showing at the 9:00am session in cinema 1
Session 1 14:30 Ratatouille  # Ratatouille is showing at 2:30pm session in cinema 1
Session 2 09:00 Up           # Up is showing at the 9:00am session in cinema 2
Request 1 1 09:00 10 # Request 1 is for 10 tickets for the 9:00am session in cinema 1
                     # Assign seats 1-10 of row A
                     # Output Booking 1 A1-A10
# Test if there are not sufficient seats in the first row, seats will be allocated from the second row
Request 2 1 09:00 12 # Request 2 is for 12 tickets for the 9:00am session in cinema 1
                     # Assign seats 1-12 of row B
                     # Output Booking 2 B1-B12
# Test that bookings are rejected if there are no available seats
Request 3 1 09:00 10 # Request 3 is for 10 tickets for the 9:00am session in cinema 1
                     # Assign no seats
                     # Output Booking rejected
Request 4 1 14:30 10 # Request 4 is for 10 tickets for the 2:30pm session in cinema 1
                     # Assign seats 1-10 of row A
                     # Output Booking 4 A1-A10
# Test that multiple bookings can share a row
Request 5 1 14:30 4  # Request 5 is for 4 tickets for the 2:30pm showing in cinema 1
                     # Assign seats 11-14 of row A
                     # Output Booking 5 A11-A14
Request 6 2 09:00 3  # Request 6 is for 3 tickets for the 9:00am session in cinema 2
                     # Assign seats 1-3 of row B
                     # Output Booking 6 B1-B3
Change 1 1 14:30 7   # Change booking 1 to 7 tickets for the 2:30pm session in cinema 1
                     # Assign seats 1-7 of row B
                     # Output Change 1 B1-B7
Cancel 6             # Cancel booking 6
                     # Deassign seats 1-3 of row B for the 9:00am session in cinema 2
                     # Output Cancel 6
Print 1 14:30        # Print the bookings for the 2:30pm session in cinema 1
                     # Print the title of the film showing in that session followed by the bookings in each row
                     # Rows are printed in the order they are declared in the input, one line for each row
                     # Print the row name, ':' and the seat ranges booked (in seat order), separated by commas
                     # Rows with no bookings are not printed out at all
```

Sample Output

The output corresponding to the above input is as follows:

```
Booking 1 A1-A10
Booking 2 B1-B12
Booking rejected
Booking 4 A1-A10
Booking 5 A11-A14
Booking 6 B1-B3
Change 1 B1-B7
Cancel 6
Ratatouille
A: 1-10,11-14
B: 1-7
```

Submission

- Submit all your files using the following command:

  ```
  give cs2511 ass1 *.java *.pdf *.txt
  ```

- Your submission should include:
  - All your `.java` source files (there is no need to include `.class` files: your Java files will be recompiled on the CSE machine)
  - A `.pdf` file containing your design documents (a UML class diagram and, optionally, other diagrams necessary to understand your design)
  - A series of `.txt` files (at least three) that you have used as input files to test your system (each including comments to indicate the scenarios tested), and the corresponding `.txt` output files (call these `input1.txt, output1.txt, input2.txt, output2.txt,` etc.)

- When your files are submitted, a test will be done to ensure that your Java files compile on the CSE machine (**take note of any error messages printed out**)

- Check that your submission has been received using the command:

```
2511 classrun -check ass1
```

Assessment

Marks for this assignment are allocated as follows:

- Correctness (automarked): 60%
- Design (30%) and programming style (10%): 40%

Late penalty: 2 marks per day or part-day late off the mark obtainable for up to 3 (calendar) days after the due date

Assessment Criteria

- Correctness: Assessed on standard input tests, using calls of the form:

```
java CinemaBookingSystem input1.txt > output1.txt   # Output to System.out is redirected to
output1.txt
```

- Design: Adherence to object-oriented design principles, clarity of UML diagrams and conformance of UML diagrams to code
- Programming style: Adherence to standard Java programming style, understandable class and variable names, adequate Javadoc and comments

Plagiarism

Remember that ALL work submitted for this assignment must be your own work and no code sharing or copying is allowed. You may use code from textbooks or the Internet only with suitable attribution of the source in your program. You should **carefully** read the UNSW policy on academic integrity and plagiarism, noting, in particular, that *collusion* (working together on an assignment, or sharing parts of assignment solutions) is a form of plagiarism.