**CMSC203**

**Assignment #6**

Movie Theaters

In 1907 you could watch a film at a Nickelodeon theater for five cents. In fact, that's why it's called a "Nickel"odeon. During the Depression or the 1920s, movies cost about 27 cents. During the swinging '60s, it cost a little less than $1 to catch a flick. The cost was about $4 to go during the 80’s and now you’re laying down a “10-spot” to watch a show.

Here’s some interesting facts about movie theaters:

* The first air-conditioned movie theater was built way back in 1922,
* The first cup holders didn't make it into cinemas until AMC Theaters introduced them in 1981.
* The first shopping mall multiplex was built in 1963. It had two screens and sat 700 people.
* New Jersey was home to the very first drive-in theater, which was built in 1933. The drive-in theater peaked in popularity in the '50s and '60s but there are now less than 1000 of them left in North America.

**Assignment Description**



MOVIE TICKET MANGER

The XYZ Theater has four types of tickets Adult, Child, Employees, MoviePass. The price of the tickets are determined as follows:

Before 6pm 6pm and after IMAX 3D Tax

Adult $10.50 $13.50 +$3.00 +$2.50 +9.6%

Child $5.75 $10.75 +$2.00 +$1.50 +9.6%

A CHILD can only attend G or PG movies

Employee – The first two movies are free and all additional movies are half of the pre-tax Adult price + 9.6% tax.

MoviePass – The first movie is $9.99 (includes tax) and all additional movies are free if:

1. Attend only one movie per day
2. Movie cannot have been seen before
3. Movie cannot be IMAX or 3D.

If any of the 3 rules have not been met, they will be charged at the Adult price.

The Management of the XYZ Theater also wants to print reports of the following:

1. All tickets
2. 3D tickets
3. MoviePass tickets
4. Monthly Sales Report

Create an application that calculates ticket prices and prints out reports needed by the management.

**Concepts tested by this assignment**

* Reading from a file
* ArrayList<>
  + Searching an Arraylist
  + Sorting an Arraylist
* Enumeration
* Inheritance
* Interface
* Polymorphism
  + Abstract classes
  + Overriding methods
  + super()

**Classes**

**Data Element - Ticket**

Create an abstract class called **Ticket** with:

* two abstract methods called **calculateTicketPrice** which returns a double and getId() which returns an int.
* instance variables (one of them is an object of the enumerated type Format) which are common to all the subclasses of Ticket
* toString method, getters and setters and at least 2 constructors, one of the constructors must be the default (no-arg) constructor.

**Data Element – subclasses of Ticket**

Create the following classes for the 4 types of tickets: Adult, Child, Employee, MoviePass.

* may contain additional instance variables appropriate for the type of ticket. They will
* extend from Ticket and define the calculateTicketPrice and getId methods. The Adult and Child classes do not require an id, so getId will return a -1. There should be
* getters and setters for the instance variables, and any other methods that are needed for your design. The
* toString method and constructors for these classes will use the super reference to take advantage of the Ticket constructor and toString methods.
* Use finals to represent constants.

**Enumerated Type – Format**

Create an enumerated type called Format. The valid values will be IMAX and THREE\_D and NONE. (You are not allowed to start enumerated variables with a numeric, so 3D was not an option)

**Data Structure**

* You will be using an Arraylist within your Data Manager to hold the ticket objects.

**Data Manager - MovieTicketManager**

Create a MovieTicketManager class that implements the MovieTicketInterface. It will contain:

* arraylist which holds references to Ticket objects (Adult, Child, Employee and MoviePass all inherit from Ticket).
* methods to add a ticket, calculate the total sales for the month, retrieve the 3D movie tickets, retrieve the MoviePass tickets, retrieve all tickets, and any other methods needed for your design.
* readFile method reads from a file and add tickets to the arrayList of Tickets
* toString method will polymorphically call the toString method for each ticket stored in the arraylist.
* monthlySalesReport method that returns a string with the entire report. Refer to program Screen shots for the format of the String returned from **toString** method.
* suggest two private methods – sortById and sortByDay that will sort the arraylist of Ticket objects either by Id or by Day.
* Follow the Javadoc given you. Private members of the Javadoc are suggestions and are not required.

**GUI Driver (provided)**

* You are provided with a GUI Driver. When the GUI is launched, tickets are added by reading from a file, or manually entered. The fields in the file are separated by colons (:). You can assume all information in the file is correct. The GUI uses an instance of the MovieTicketManager class.
* The user will have the ability to add additional employees through the keyboard. The labels and textfields will change depending on the type of movie patron that is selected to be added.
* A report is displayed of all movie tickets, 3D tickets and MoviePass tickets. All movie tickets and 3D ticket reports are in order by date. MoviePass tickets reports are in numeric order by the moviepass id.
* Data Validation: If a CHILD ticket is being purchased, it can only be a G or PG movie rating. The time must be between 8 and 23 inclusively. The day must be between 1 and 31 inclusively.
* Note: your GUI need not display an icon.

**External Documentation**

* Create Javadoc for all classes that you have created. Upload your entire doc folder.
* Provide a UML diagram with all classes and their relationships.
* Store your files in your GitHub repository and submit a screenshot.

**Testing**

* Ensure that the JUnit tests in MovieTicketManagerTest.java succeed.
* Implement the MovieTicketManagerSTUDENTTest.java test methods.
* Create a Junit test for your Adult and MoviePass classes. Test classes for Child and Employee classes are provided.

The minimum information stored for each ticket:

**Assignment Details**

* Movie name
* Rating (G, PG, PG13, R, or NR)
* Day
* Time
* id (only needed for Employees and MoviePass members)
* Price of ticket
* Type (Adult, Child, Employee, MoviePass )
* Format (IMAX***,*** 3D, NONE)

Reports needed by the Management:

* Print All Tickets (in order by date)
* Print 3D Tickets (in order by date)
* Print MoviePass Tickets (in order by MoviePass Id)
* Monthly Sales Report. Displays the total sales and number of each type of ticket sold that month. Also displays the total of all tickets sold that month

Use Alert boxes to display the reports.

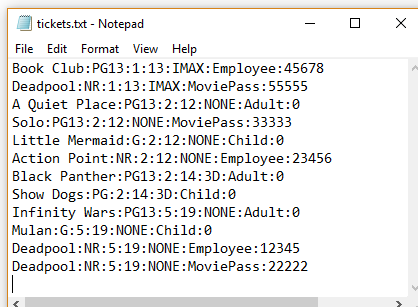
For simplicity sake, we will only be managing tickets for one month at a time. The day will be represented as an int (1-31)

For simplicity sake, movies start at the top of the hour and will be represented as an int in military format. The earliest movie will begin at 8 am (8) and the latest movie will begin at 11pm (23)

Movie ratings will be a string: G, PG, PG13, R, or NR.

INPUT FILES:

Input files that can be read from this application have values that are separated by a “:”.



SORTING:

Here is a simple Insertion Sort that works for arrays of ints:

1. int temp;
2. for (int i = 1; i < array.length; i++) {
3. for (int j = i; j > 0; j--) {
4. if (array[j] < array [j - 1]) {
5. temp = array[j];
6. array[j] = array[j - 1];
7. array[j - 1] = temp;
8. }
9. }

Convert this to work for an ArrayList of Ticket objects for your sortById and sortByDay methods.

When you sort by Day, don’t worry about ordering at a second level (such as time).

EMPLOYEE-45678 IMAX Movie: Book Club Rating: PG13 Day:1 Time: 19 Price: $0.00

MOVIEPASS-55555 IMAX Movie: Deadpool Rating: NR Day:1 Time: 23 Price: $14,80

Or:

MOVIEPASS-55555 IMAX Movie: Deadpool Rating: NR Day:1 Time: 23 Price: $14,80

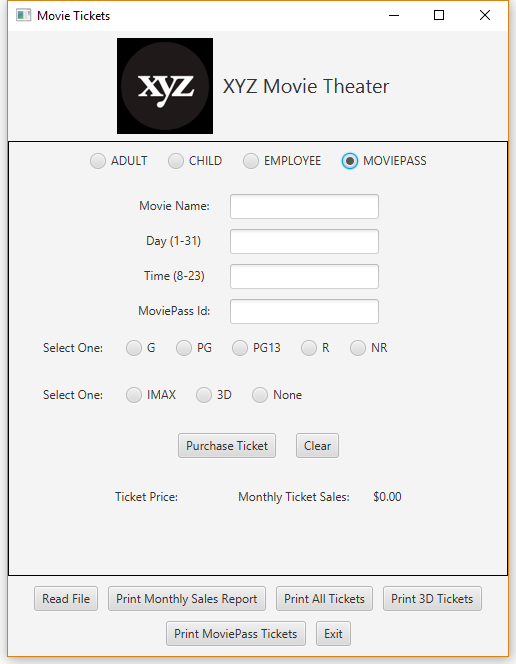
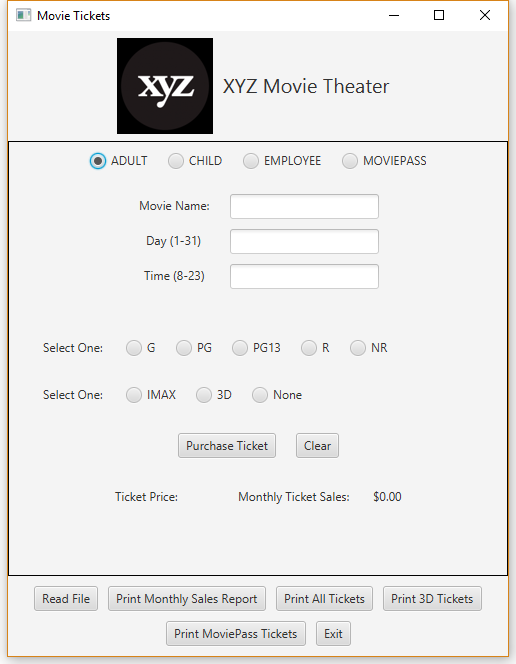
EMPLOYEE-45678 IMAX Movie: Book Club Rating: PG13 Day:1 Time: 19 Price: $0.00

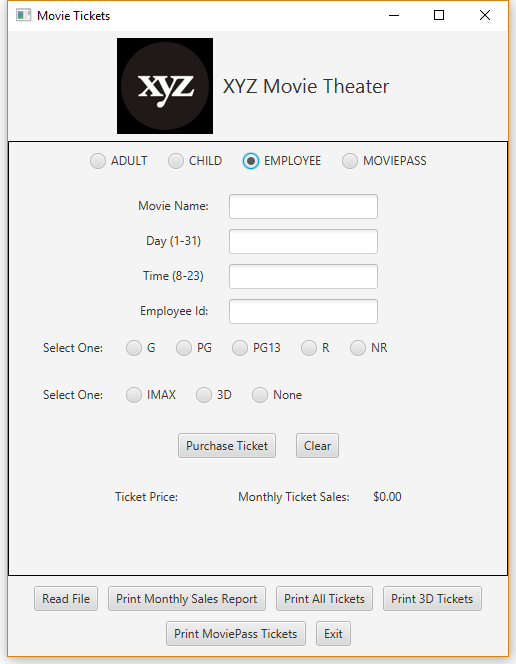
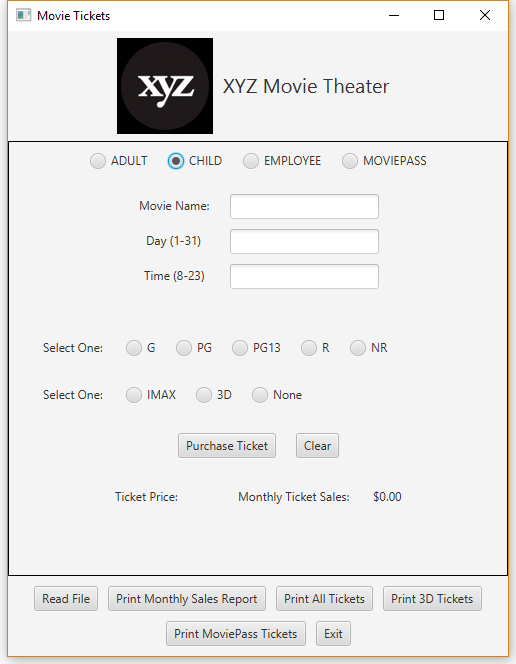
The JUnit test would consider both of these to be correct.

When you order by Id, don’t worry about ordering at a second level (such as day)

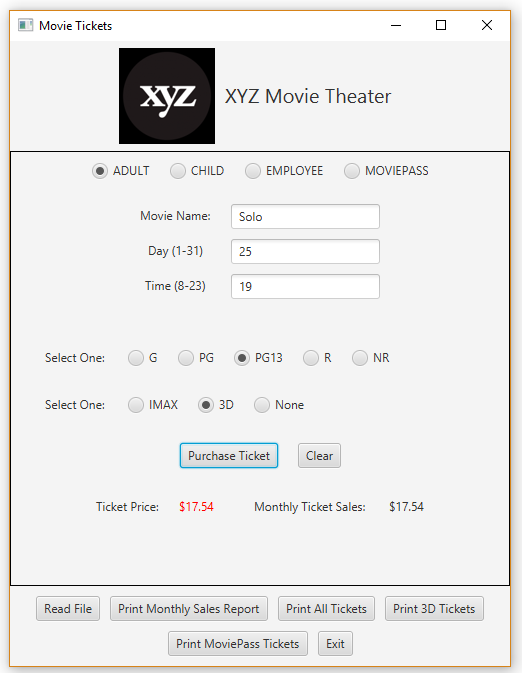
GUI:

Labels and Textfields will change depending on the type of ticket selected:





Ticket Price will be printed in RED to make it easy to find:



**BUTTONS**:

When the Purchase Ticket button is selected:

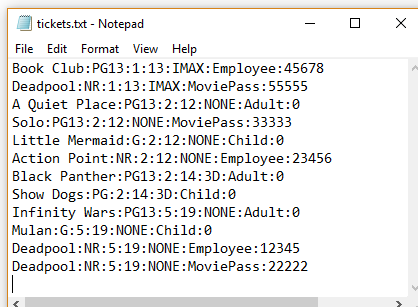
1. Information will be gathered from the textfields and radiobuttons
   1. data validation of day, time and CHILD ticket
2. addTicket(…)
3. Display the price of the ticket

When the Read File button is selected:

1. User selects file with FileChoose
2. readFile(..)
3. Display the Monthly Total Sales

**Based on the following input file:**

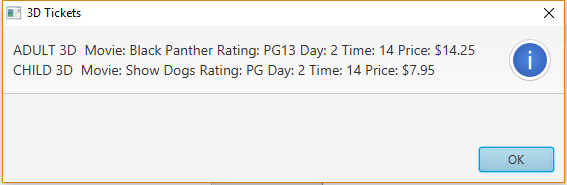
**Examples**



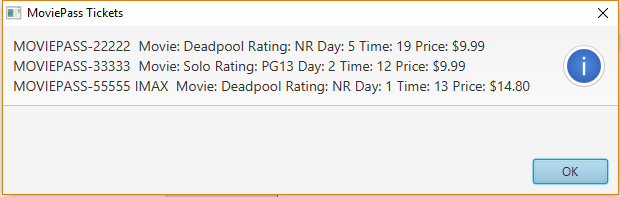
**Result of Print All Tickets: (in order by Day)**



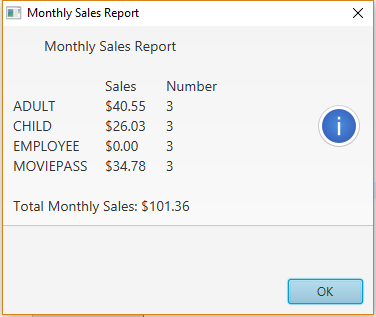
**Result of Print 3D Tickets:**



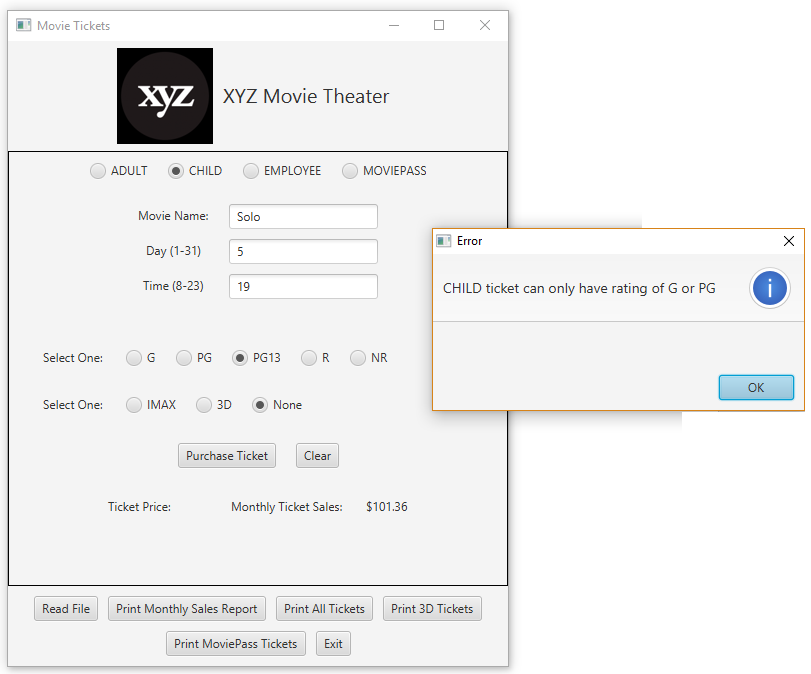
**Result of Print MoviePass Tickets:**



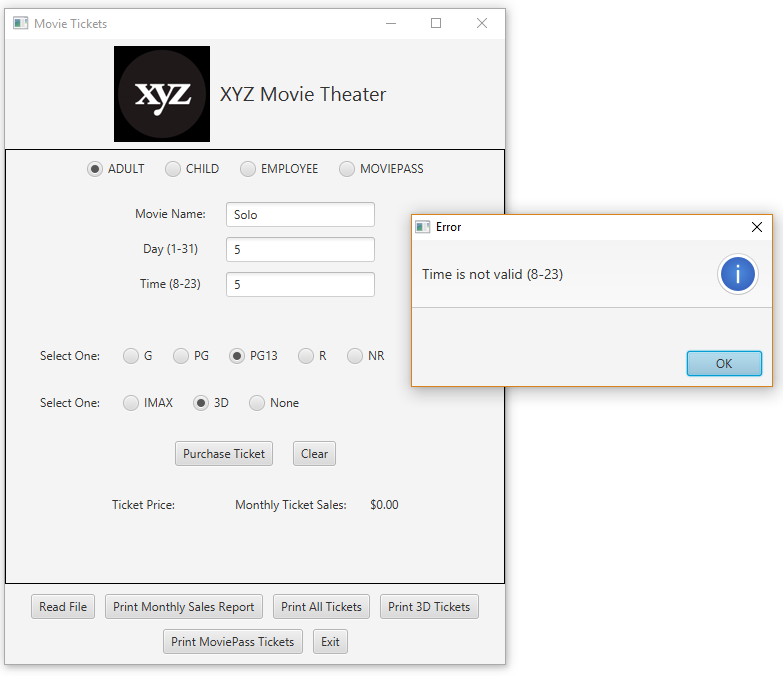
**Result of Monthly Sales Report:**



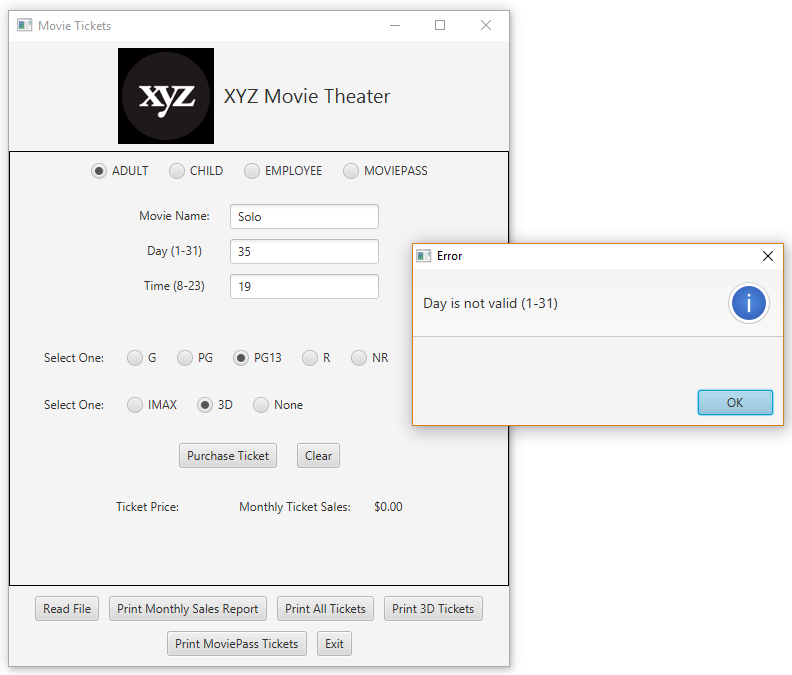
**Error Message - CHILD attempting to purchase a PG13 ticket.**



**Error message – Time not valid**



**Error Message – Day not valid**



**Upload into GitHub**

Upload all input files and all implemented files into GitHub in the repo you created in Lab 1 in a directory named CMSC203\_Assignment6

**Deliverables / Submissions:**

**Deliverables**

Design: UML class diagram with pseudo-code for key methods

Implementation: Submit a compressed file containing the follow (see below): The Java application (it must compile and run correctly); Javadoc files in a directory; a write-up as specified below. Be sure to review the provided project rubric to understand project expectations. The write-up will include:

* UML diagram – latest version
* Any assumptions that you are making for this project
* In three or more paragraphs, highlights of your learning experience. What were you successful at, and what (if anything) were you not able to implement?

**Deliverable format:** The above deliverables will be packaged as follows. Two compressed files in the following formats:

* LastNameFirstName\_Assignment6\_Complete.zip, a compressed file in the zip format, with the following:
  + Write up (Word document) - reflection paragraphs
  + UML Diagram - latest version (Word or jpg document)
  + doc (directory) – Javadoc
    - * subdirectories and other files
      * File1.html (example)
      * File2.html (example)
  + src (directory) – Java and JUnit tests files
    - * File1.java (example)
      * File2.java (example)
      * Filetest.java (example)
* LastNameFirstName\_Assignment6\_Moss.zip, a compressed file containing one or more Java files:
  + File1.java (example)
  + File2.java (example)
  + Filetest.java (example)

This folder should contain Java source files only