Project #1
Computer Science 2334
Fall 2007

User Request:

"Create a Movie Search Engine."

Milestones:

1.	Use command line arguments to read in a file name.	*10 points*
2.	Use simple File I/O to read a file.	*10 points*
3.	Create an ADT to store information about a movie.	*15 points*
4.	Create an ADT that abstracts the use of an array of movies.	*15 points*
5.	Implement a program that allows the user to search the database of movies as described below.	*20 points*
	Develop and use a proper design.	*15 points*
	Use proper documentation and formatting.	*15 points*

Description:

For this project, you will put together several techniques and concepts learned in CS 1323 and some new techniques to make an application that searches a huge database of movies. This project provides the same functionality as some websites that allow you to search their collections of movies in electronic form. The data used in this project is provided by the Netflix Prize competition (http://www.netflixprize.com/).

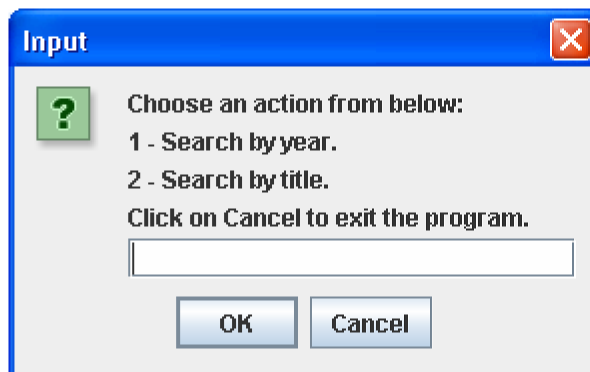
One of the best things about this project is that it uses real data (and lots of it!). There are over 10,000 movies in the database. To the surprise of no one, the best approach to this somewhat large problem is to decompose the problem into separate classes that can be gradually built up.

Operational Issues:

To run the program you will give the following command at the command prompt:

java Project1 <input filename>

Your program will proceed to process the text file specified by <input filename> (see below for information on how to read arguments from the command line). Once the entire list of movies and their corresponding data have been read into your program and stored, in an array, your program will display a menu to the user. You should use a *JOptionPane* to create an input dialog similar to the one shown below.



Note that if the user clicks on cancel, then the program should gracefully exit.

The basic version of the program will allow the user to perform two types of searches; you may extend the program for extra credit as discussed in the *Extra Credit* section below. The first search looks for movies released in a year specified by the user. For example, if the user was to search for the year 1918, then the results shown below are a sample of what should be displayed to the user using a *JOptionPane* message dialog.

The second type of search looks for movies with a specific title. The results of this type of search will also be displayed using a *JOptionPane* message dialog. Note, that searches should be case insensitive (e.g., a search for "chaplin" will return "Chaplin").

Once the user has dismissed the message dialog, that shows the results of the search, the program should display the program menu. The program menu will continue to be displayed, in this fashion, followed by input and output dialogs for the chosen search, until the user clicks on Cancel, at which time the program must gracefully exit.

Movie Data:

The data for the program are stored in a text file that you need to download from the class website. Note that the first line of the data file contains the number of movies in the data file. The movies are listed line-by-line with the following format:

```
Movie ID, Year, Title
```

Implementation Issues:

There are two Java elements which may be new to most students in this project: reading from a file, and command line arguments. These Java features are summarized below.

Reading from a file:

We will discuss File I/O in depth later in the class, this project is just designed to give you a brief introduction to the technique. Reading files is accomplished in Java using a collection of classes in the java.io package. To use the classes you must import the following package:

```
import java.io.*;
```

The first action is to open the file. This associates a variable in the program with the name of the file sitting on the disk.

```
String fileName = "this is the file name";
FileReader fr = new FileReader(fileName);
```

Next the FileReader is wrapped with a BufferedReader. BufferedReaders are more efficient than FileReaders since they save groups of characters during a single operation instead of working with characters individually. Another advantage of using the BufferedReader is that there is a command to read an entire line of the file, instead of a single character at a time. This feature comes in particularly handy for this project.

```
BufferedReader br = new BufferedReader(fr);
```

The BufferedReader can now read in Strings.

```
String nextline;
nextline = br.readLine();
```

Look at the Java API listing for BufferedReader and find out what readLine()returns when it encounters the end of the file (stream). When you are finished with the BufferedReader, the file should be closed. This informs the operating system that you're finished using the file.

```
br.close();
```

Closing the BufferedReader also closes the FileReader.

Any method which performs I/O will have to throw or catch an IOException. If it is not caught, then it must be thrown and caught or thrown in the calling methods. The syntax is given below:

```
public void myMethod(int argument) throws IOException
{
        //method body here
}
```

Command Line Arguments:

Sometimes it is handy to be able to give a program some input when it first starts executing. Command line arguments can fulfill this need. Anyone who has used many MS-DOS or Unix commands has probably come across command line arguments at some point or another. For example, in MS-DOS, to list all of the file names beginning with the letter S in a directory, one enters:

```
dir S*.*
```

In this example, dir is the command name, while the command line argument is S*.*

Command line arguments are handled in Java using a String array that is traditionally called args in Java (the name is actually irrelevant.) This is passed to the main method by the operating system. The program below will print out the command line arguments. This is a sample program designed to demonstrate how command line arguments work and ***should not be directly copied into your project.***

```
public static void main(String[] args)
{
        System.out.println(args.length +
                           " command line arguments:");
        for (int i=0; i< args.length; i++)
                System.out.println("args[" + i + "] = " + args[i]);
}
```

Note that when you execute a Java program, the syntax is as follows:

```
java classname arg0 arg1 arg2 ... argn
```

Neither java nor the classname will show up as command line arguments.

Milestones:

A milestone is a "significant point in development." In other words, milestones serve to guide you in the development of your project. Listed below are a set of milestones for this project along with a brief description of each.

Milestone 1. Use command line arguments to read in a file name.

The name of the file that stores the database of movies will be passed to the program on the command line as discussed above. Type in the sample program given in the section on Command Line Arguments and make sure that you understand how the command line you type in to run your Java program affects the *String[] args* parameter that is passed into the main method of the program. Then, write a main method for your program that reads in the name of the data file from the command line.

Milestone 2. Use simple File I/O to read a file.
Before you can allow the user to search the database of movies, you must first be able to read a text file. Examine the section above on *Reading from a file*. A good start to the program is to be able to read in the name of a file from the command line, read each line from the program, one at a time, and print each line back out to the console using *System.out.println()* as it is read in. *Later, you will want to remove the code that prints out each line read in from the file, since the project requirements do not specify that the file is to be written out to the console as it is read.*

Milestone 3. Create an ADT to store information about a movie.
You must create a class that holds the information related to a single movie in the database before you can store the information that is read in from the input file. Think about what information is associated with each movie and how to most efficiently store the information. Also, think about any methods that may help you to manage and search the data by abstracting operations to be performed on a single entry in the database that will be used by another class. Are there any special constants that will aid the storage and manipulation of data?

Milestone 4. Create an ADT that abstracts the use of an array of movies.
You are to store the object representing each entry in the database into an array of objects. However, it is not necessary for the portions of the program that will display the program's menu of searches and carry out user actions to directly operate on this array. You should create a class that encapsulates this array and allows the addition of movies and their related information and also supports the required search operations on the array. This class will represent the database (or collection of information associated with the program). Think about the operations that this class needs to support and how it will use the ADT created for Milestone 3. At this point, you should be able to read in the input file and create an object for each movie, and its associated data from the file, and store the movie title and year into the array of objects. The movie object must be stored at a location in the array equal to its Movie ID. Note that the data file used for grading may be larger than the data file provided for testing. Also, the movies in the data file might not be ordered according to Movie ID.

Milestone 5. Implement a program that allows the user to search the database of movies as described below.
This is where the entire program starts to take on its final form and come together. Here you will create the input and output dialogs and the menu system. Start by creating the input dialog for the first search option, which searches according to a user-specified year, and the output dialog that outputs the search results. Tie together the input dialog, the ADT from Milestone 4, and the output dialog to make this search functional and test its functionality.

Next, do the same for the second search option, which searches by title.

Finally, you are ready to create the menu using an input dialog and write the main loop of the program that will take a menu option as input and invoke the correct methods that were created to test the two search options. Remember that when the user clicks on cancel in the program menu, the program must gracefully exit. This can be accomplished by using *System.exit(0)*.

How to Complete this Project:

1. Revise your UML design from Lab. Be sure to clearly write your name and "Project 1" on the top of the UML diagram. You must turn this in again on Wednesday, September 12th ***at the beginning of class***. Make sure to keep a copy of this.
2. Create the classes and methods specified in your design, but do not put code in the methods. Add the required documentation to your classes and methods as specified in the Documentation Requirements posted on the class website. This is called "stubbing" your classes and methods.
3. Run your stubbed Java files through Javadoc.
   a) Download the *docs.opt* file from Lab1 on the class website and save it where your Java source code files for the project are located.
   b) Run the following command at the command line. (First open a DOS command window and "cd" to the directory that contains your project files.)

   ```
   javadoc @docs.opt *.java
   ```

   c) This will create a set of HTML files in a directory named javadocs under your project directory.
   d) Open the index.html file found in the javadocs directory in Firefox (or your favorite browser) using "Open..." in the "File" menu (click on the "Browse" button to specify which file to open) and examine the listing for each Java class listed on the left hand side of the page.
4. Submit your "stubbed" Java source code using the submit command on *codd.cs.ou.edu*. See the *Due Dates and Notes* section below for the exact command to use on *codd*.
5. Implement the design you have developed by coding each method you have defined as well as any others you have left out of your design. As you do this, make sure to modify and annotate the changes to your design on your UML and properly document all new code. A good approach to the implementation of your project is to follow the project's Milestones in the order they have been supplied.
6. Test your program and fix any bugs.
7. Once you have completed the project and are ready to submit it for grading, create a new set of Javadoc files using Javadoc and inspect them to make sure your detailed design is properly documented in your source code.
8. Submit your project via the submit command on *codd.cs.ou.edu* by following the instructions found in the Submission Guidelines posted on the class website. See the *Due Dates and Notes* section below for the exact command to use on *codd*.

Extra Credit Features:

You may extend this project with more search features for an extra 5 points of credit. Think of ways to enable a wider range of searches to be used, such as searching based on regular expressions and searching based on how a name sounds rather than how it is spelled.

To receive the full five points of extra credit, your extended search feature must be novel (unique) and it must involve effort in the design of the integration of the feature into the project and the actual coding of the feature. Also, you must indicate on your final UML design, the portions of the design that support the extra feature(s); and you must include a write-up of the feature(s) in your MILESTONES.txt file. The write-up must indicate what the feature is, how it works, how it is unique, and the write-up must cite any outside resources used. (*For example, if you find, and use the information from, a website that explains an algorithm that can be used to perform a particular type of pattern matching, then you must list the website in your MILESTONES.txt file.*)

Note that you are not allowed to use any third-party utilities.

Due Dates and Notes:

1. Your revised design and detailed Javadoc documentation are due ***on Wednesday, September 12th.*** Submit your revised UML design ***on engineering paper*** at the ***beginning of class***. Submit the "stubbed" source code using the submit tool on *codd.cs.ou.edu* by ***9:00pm***. This submission counts as part of the design and documentation portions of the project grade.

   The commands for submitting your "stubbed" Java source code on *codd.cs.ou.edu* are:

   ```
   cd design1
   /opt/cs2334/bin/submit cs2334-010 project1-design <.java filenames>
   ```

where <.java filenames> is a list of the .java files you are submitting.

2. The final version of the project is due on Wednesday***, September 19th.*** Submit your final UML design ***on engineering paper*** at the ***beginning of class***. Submit your source code files and the COMPILATION.txt, EXECUTION.txt, and MILESTONES.txt files using the submit tool on *codd.cs.ou.edu* by ***9:00pm***.
   The commands for submitting your final project on *codd.cs.ou.edu* are:
   ```
   cd project1
   /opt/cs2334/bin/submit cs2334-010 project1-final <.java filenames>
   ```
   where <.java filenames> is a list of the .java files you are submitting.
   Make sure that the COMPILATION.txt, EXECUTION.txt, and MILESTONES.txt files are present in the directory along with all of your .java source files.

3. The *Get Out of Jail Free* due date for this project is Friday***, September 21th***. Submit your final UML design ***on engineering paper*** at the ***beginning of class***. Submit your source code files and the COMPILATION.txt, EXECUTION.txt, and MILESTONES.txt files using the submit tool on *codd.cs.ou.edu* by ***9:00pm***.
   - I advise you to save your *Get Out of Jail Free* until later in the semester.

   The commands for submitting your final project on *codd.cs.ou.edu* using Get Out of Jail Free are:
   ```
   cd project1
   /opt/cs2334/bin/submit cs2334-010 project1-goojf <.java file names>
   ```
   where <.java filenames> is a list of the .java files you are submitting.
   Make sure that the COMPILATION.txt, EXECUTION.txt, and MILESTONES.txt files are present in the directory along with all of your .java source files.

4. You are not allowed to use the StringTokenizer class. Instead ***you must use String.split()*** and a regular expression that specifies the delimiters you wish to use to “tokenize” or split each line of the file.