CIS 162 Project 4
Mega Millions Lottery Simulation

Due Dates

- Phase 1 due in lab Wednesday, March 19th. Show all unit tests passing (20 pts). No printouts are turned in during this phase. The `LotteryMachine` additional challenges are not included in the unit tests.
- Phase 2 due in lab Wednesday, March 26th. Turn in printouts for `LotteryTicket`, `LotteryMachine` and `GUI`. Be prepared for a full demonstration.

Before Starting the Project

- Read chapter 6
- Read this entire project description before starting

Learning Objectives

After completing this project you should be able to:

- *use* ArrayLists to maintain and process collections of objects
- *use* for-each loops
- *use* while loops
- *read data* from external text files
- *use* static methods available in the Java library

Mega Millions Lottery

Players pick five numbers (1-75) with no duplicates and a special number called the Mega Ball (1-15). Numbers are randomly generated twice per week to determine the winning amount on each ticket. The vast majority of tickets do not win anything! Here is the payout based on how many numbers match.

# of matches	Mega Ball match	Prize
5	1	$5,000,000
5	0	$1,000,000
4	1	$5,000
4	0	$500
3	1	$50
3	0	$5
2	1	$5
1	1	$2
0	1	$1

Step 1: Create a New BlueJ Project

Step 2: Download Files

- Download the "TicketInfo.txt" file and save it in the folder that BlueJ created for this new project. There are 50,000 entries! You will not see it from within BlueJ but you can see it from within the Windows Explorer.
- Download the provided "GUI.java" and save it in the project folder. You should see this class within BlueJ. Do not make any changes until later.

Step 3: Create a class called LotteryTicket (5 pts)

Implement a class to maintain information about the ticket including: first name, last name, birth date (day, month, year), city, ST, zip code, b1, b2, b3, b4, b5, megaBall and a prize amount (double).

- Provide appropriate names and data types for each of the instance variables.
- `public LotteryTicket (String info)` - a constructor that initializes all of the instance variables to appropriate values given a single String. See information below about parsing Strings into smaller pieces.

```
Amy, Zu, Phoenix, AZ, 78234, 4/20/1960, 4, 12, 15, 36, 67, 12
```

- `public String getFirst()`
- `public String getLast()`
- `public String getCity()`
- `public String getState()`
- `public int getZipcode()`
- `public int getDay()`
- `public int getMonth()`
- `public int getYear()`
- `public double getPrize()`
- `public void setPrize(double amount)` – set the prize amount
- `public boolean hasBall(int val)` – return true if `val` is one of the five numbers (but not the mega ball).
- `public boolean hasMegaBall(int val)` – return true if `val` matches the mega ball.
- `public String toString( )` - return a formatted ticket summary (a single String with embedded newlines) . Use `NumberFormat` for the currency. For example,

```
Jane Smith
San Francisco, CA 49401
1 2 3 4 5   6
Prize: $1.00
```

- `public static void main(String args[])` – thoroughly test each of the methods in this class.

Step 4: Create a class called LotteryMachine (60 pts)

Instance Variables and Constructors
A class should contain several instance variables (section 4.2) that represent the state of the object. Constructors (section 4.5) provide initial values to these variables.

- Define an instance variable that holds an `ArrayList` of `LotteryTicket`, and six integers for the selected numbers.
- `public LotteryMachine ( )` - a constructor that instantiates an empty `ArrayList` in one line of code. No tickets are inserted yet. Initialize additional variables as appropriate.

Accessor Methods
It is good practice to provide *accessor* methods for most instance members (section 4.3). These methods are informally called *getter* methods and allow access to the state of the object.

- `public void addTicket (LotteryTicket t)` – add the ticket to the `ArrayList`. This method contains one line of code.
- `public int countTickets( )` - return the number of tickets. This method only needs to contain one line of code!

Support Methods (10 pts)
It is good practice to include *support* methods to help the class compete various tasks (section 4.3). These private methods can only be invoked internally and are not available to the public. They are invoked by other methods to keep them short and tidy.

- `private void pickNumbers()` – Select five random numbers (1-75) without duplicates. Select the random mega ball (1-15). Feel free to design your own solution or read below for a suggestion.
- `private int countMatches(LotteryTicket t)` - return the number of matches between the selected five numbers (not the mega ball) and the provided ticket.
- `private void makePayouts()` – step through all tickets and assign the appropriate prize based on the selected six numbers. See award table. Invoke `countMatches()` to assist.
- `private String formatNumbers()` - return a String that summarizes the six selected numbers.

```
Selected Numbers: 7 23 29 41 68   5
```

Advanced Methods (40 pts)
The following methods search and analyze the ArrayList of tickets. Use the elegant for-each loop to process the ArrayList (section 6.4).

- `public void drawTicket( )` – pick random numbers and assign awards to all tickets. This method only needs to contain two lines of code.
- `public void drawTicket(int b1, int b2, int b3, int b4, int b5, int m )` – this method is used for testing. Assign the provided values to the six numbers before assigning awards to all tickets.

- `public void readTickets(String filename)` - open the provided `filename` and read all the data. There are 50,000 entries in "TicketInfo.txt". Refer to the information about reading text files later in this document.
- `public String createReport( )` – create a short report for all tickets that includes the six selected numbers, number of tickets, average prize amount and the biggest winner. See below for sample output.
- `public String createReport(String st)` – create a report for all tickets sold in the state with the two-letter abbreviation `st`. The report includes the six selected numbers, number of tickets, average prize amount and the biggest winner. See below for sample output. Beware if no tickets were sold in the state.
- `public LotteryTicket getOldestPlayer ( )` - return the oldest person in the database (to the day). Give careful thought to this one! If more than one person happens to share the title of oldest it does not matter which one you return.
- `public LotteryTicket getBiggestWinner ( )` - return the ticket with the largest prize. If more than one ticket happens to share the title then it does not matter which one you return.
- `public ArrayList <LotteryTicket> getMajorWinners(double amount)` - return an `ArrayList` of all tickets with a prize of at least `amount`. If necessary, it is OK to return a list with no tickets.

Additional Challenges (10 pts)
The following methods search and analyze the tickets. Use the elegant **for-each loop** to process the `ArrayList` (section 6.4).

- `public String multipleGames ( )` – use a loop to repeatedly draw tickets until a player wins the Mega Millions payout with six matches. After each drawing without a jackpot winner, increase the prize by $1,500,000. After a player eventually wins, return a String that reports the number of games played and the jackpot winner (see sample output). Consider adding instance variables for the jackpot amount and a boolean variable if a player wins.
- Lottery numbers are generally reported in sorted order. Enhance the `pickNumbers()` method to assign the five numbers to the instance variables in sorted order. For example, num1 would have the lowest selected numbers and num5 would have the highest. Consider placing the five numbers into an array (section 8.1 – 8.3) and then using a provide method to sort them.

```
int nums[] = new int[5];
// fill the array with the selected numbers
Arrays.sort(nums);
// retrieve the numbers from the array
```

Parsing Strings

Ticket information is contained in a long string with values separated by commas. You must write code that splits the string into individual pieces and converts them to integers and Strings as necessary. Blank spaces at the start and end of String must be removed with the `trim()` method.

Limited sample of data

```
Amy, Zu, Phoenix, AZ, 78234, 4/20/1960, 4, 12, 15, 36, 67, 12
```

Here is sample code you will need to adapt.

```
public LotterTicket(String info){

      // Split the info string into multiple pieces separated by ',' or '/'
      String [] tokens = info.split(",|/");
      first = tokens[0].trim();
      last = tokens[1].trim();
      month = Integer.parseInt(tokens[5].trim());
      megaBall = Integer.parseInt(tokens[13].trim());
}
```

Selecting Lottery Numbers

Devise a solution to select five random numbers (1-75) with no duplicates. Consider creating an ArrayList of Integer that holds the numbers 1-75. Choose a random number between 0 and the size of the ArrayList. REMOVE that number from the ArrayList and repeat four more times.

Ticket Info Text File

This file contains fictional player and ticket information. A sample line of the file is shown below. Note that items are separated by commas and the birthday is separated by slashes. The `readTickets()` method reads one line at a time and passes the text to the LotteryTicket class constructor.

```
Amy, Zu, AnyCity, NC, 27834, 4/20/1960, 5, 12, 37, 39, 68, 11
```

Sample Method to Read a Text File

The following method reads from a text data file one line at a time and prints to the screen. The solution is a bit more complex than Listing 5.10 because the book is hiding some necessary details. You will have to modify this code to create LotteryTickets and insert into the machine.

Sample Code

```
public void readTickets(String filename){

        // add a line of code to instantiate a new ArrayList

        // Attempt to read the complete set of data from file.
        try{
            File f = new File(filename);
            Scanner sc = new Scanner(f);
            String info;

            while(sc.hasNext()) {
                info = sc.nextLine();

                // remove this print statement after method works
                System.out.println(info);

                // add two lines of code to instantiate a new Lottery Ticket
                // and add to ArrayList
            }
            sc.close();
        }
        catch(IOException e) {
            System.out.println("Failed to read the data file: " +
                               filename);
        }

    }
```

Step 5: Software Testing (10 pts)

Software developers must plan from the beginning that their solution is correct. BlueJ allows you to instantiate objects and invoke individual methods. You can carefully check each method and compare actual results with expected results. However, this gets tedious. Another approach is to write a `main` method that calls all the other methods.

Create 3-4 tickets and add them to the mchine. Your tests will be more thorough than the following incomplete example.

```
public static void main(String args[]){
   LotteryMachine m = new LotteryMachine();
   String info;

   info = "Jane, Smith, San Francisco, CA, 49401, 2/14/1983, 1,2,3,4,5,6";
   LotteryTicket tix = new LotteryTicket(info);

   m.addTicket(tix);
   m.drawTicket();
   LotteryTicket old = m.getOldestPlayer();
   System.out.println(old);
}
```

Possible Output (prize depends on random drawing)

```
Jane Smith
San Francisco, CA 49401
1 2 3 4 5   6
Prize: $1.00
```

JUnit Testing

As an alternative, JUnit is a Java library that helps to automate software testing. A JUnit test file `LotteryTest.java` has been provided on BlackBoard. Refer to Project 2 instructions for additional information about JUnit Testing. Remember, your class names and method names must match exactly with these specifications. A compile error for `LotteryTest.java` suggests a mismatch of some identifier or method name.

Sample Results

Results from the GUI with the text file properly read should yield the following results.

Result from Report for All Tickets

```
Report for All Sales
Selected Numbers: 4 8 31 42 68   7
Tickets sold: 50,000
Average prize: $0.10

Biggest Winner
Stephanie Goza
Camden, NJ 8102
8 17 31 38 68   7
Prize: $50.00
```

Result for Biggest Winner

```
Biggest Winner

Stephanie Goza
Camden, NJ 8102
8 17 31 38 68   7
Prize: $50.00
```

Result for Multiple Games Until Jackpot

```
Games until Jackpot: 1,822

Barbara Patterson
Louisville, KY 40219
5 29 44 51 61   6
Prize: $2,732,500,000.00
```

Result for Major Winners ($50 or higher)

```
Major Winners ($50 or higher)

Lawanda Hughes
Harveyville, KS 66431
37 39 43 54 59   5
Prize: $50.00

Melody Crider
Ridley Park, PA 19078
37 39 45 54 68   5
Prize: $50.00

Brandon Fowler
Rosemount, MN 55068
37 39 54 71 74   5
Prize: $50.00

Mary Lehn
Rocky Mount, NC 27804
32 37 39 54 59   7
Prize: $500.00
4 tickets listed
```

Step 6: Create a GUI class (15 pts)

Now that you have the basic application working within it's own class it is time to create a more interesting graphical user interface for the user. Read sections 4.7, 4.8 and 5.7. Read about BorderLayout and BoxLayout in section 7.11

- Modify the provided `GUI` class.
- Define instance variables for a `LotteryMachine` object, and `JLabels`, `JTextFields` and `JButtons` as needed to match the sample screenshot. The central `JTextArea` is provided for you.

GUI Constructor

- Instantiate a `LotteryMachine` and invoke the `readTickets`() method.

```
lm = new LotteryMachine();
lm.readTickets("TicketInfo.txt");
```

- Several menu items are created for you.
- The `JTextArea` is created for you and inserted in the CENTER region.
- Create a `JPanel` for the three `TextFields` and corresponding labels. Insert the panel in the SOUTH region.
- Add `JButtons` and `JLabels` to the provided `eastPanel`. Insert the panel in the EAST region.

```
eastPanel.add(new JLabel("Draw Ticket"));
```

- Instantiate an `ActionListener` object and register with all buttons. Your code will look a bit different but here is an example:

```
ButtonListener listener = new ButtonListener();
oldestButton.addActionListener(listener);
```

Inner Class for the Action Listener

The inner class `ActionListener` has already been created for you (see the example in Listing 5.13). You will add several if statements for each of the buttons and menu items.

```
if (e.getSource() == quitItem){
      System.exit(1);
}

if(e.getSource() == oldest){
      results.setText("Oldest Player");
      t = lm.getOldestPlayer();
      results.append(t.toString());
}
```

Additional Methods

Provide the following methods to support the GUI class.

- `public void displayTickets (ArrayList <LotteryTicket> list)` – this method displays the information for each ticket in the provided `list`. Note that this method is invoked from a variety of places within the GUI class. The following example is incomplete but demonstrates the basic idea. This assumes the `LotteryTicket toString()` method is working properly.

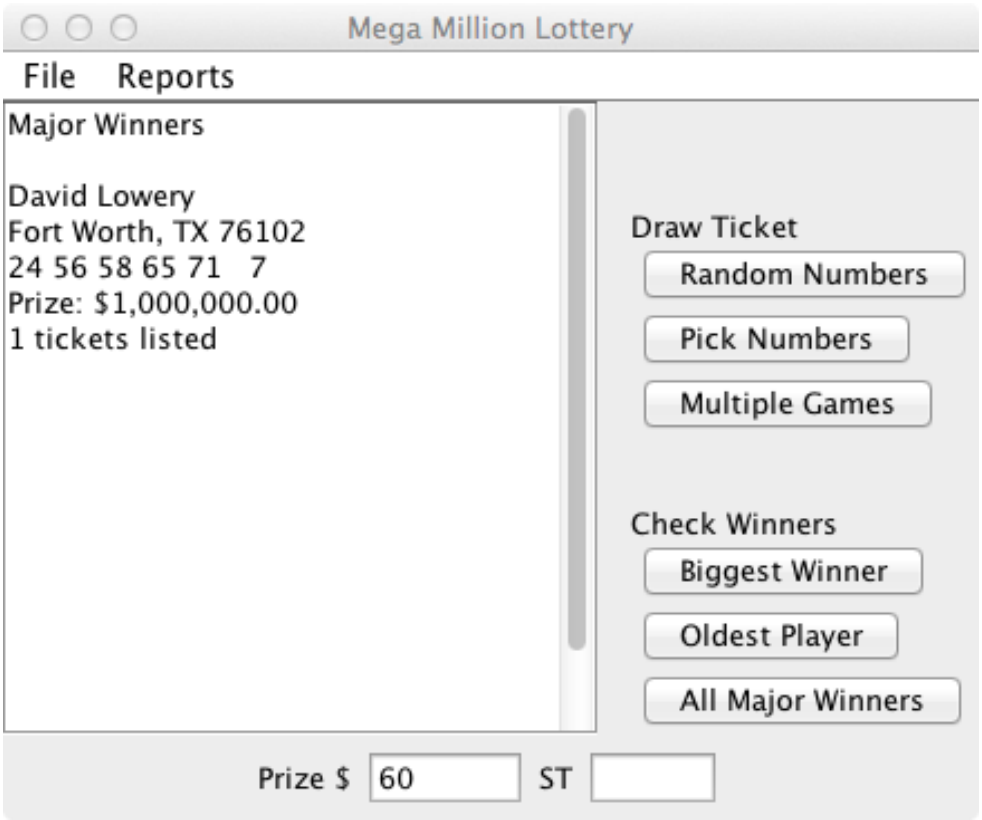
```
for (LotteryTicket t: list){
    results.append(t + "\n");
}
```

- `private void pickNumbers()` – use `JOptionPane` dialog boxes to prompt the user for each of the six numbers. Afterwards, invoke the drawTicket() method.
```
String str = JOptionPane.showInputDialog("Enter Ball #1:");
int b1 = Integer.parseInt(str);
lm.drawTicket(b1,b2,b3,b4,b5,mega);
```
- provide code in `actionPerformed()` to provide a popup `JOptionPane` warning about empty JTextFields when invoking the Report by State or All Major Winners. Otherwise, the user will receive runtime exception.

Extra Challenge (for fun)

It is OK for the GUI to automatically load the data file by invoking `readTickets()` within the constructor. However, a better solution is to allow the user to specify a data file. We provide a method `openFile()` that you can adapt. The user will select the File – Open menu item and a dialog box will pop up to select the data file. This file is then processed by invoking `readTickets()`.

GUI Screenshot



Grading Criteria

There is a 50% penalty on programming projects if your solution does not compile.

- Stapled cover page with your name and signed pledge. (-5 pts if missing)
- Project requirements as specified above. (90 pts)
- Elegant source code that follows the GVSU Java Style Guide. (10 pts)

Late Policy

Projects are due at the START of the class period. However, you are encouraged to complete a project even if you must turn it in late.

- The first 24 hours (-20 pts)
- Each subsequent weekday is an additional -10 pts
- Weekends are free days and the maximum late penalty is 50 pts.

Turn In

A professional document **is stapled** with an attractive cover page. Do not expect the lab to have a working stapler!

- Cover page - Provide a cover page that includes your name, a title, and an appropriate picture or clip art for the project
- Signed Pledge – The cover page must include the following signed pledge: "I pledge that this work is entirely mine, and mine alone (except for any code provided by my instructor). " Also, provide the names of people you worked with and explain the level of cooperation. You are responsible for understanding and adhering to the School of CIS Guidelines for Academic Honesty.
- Time Card – The cover page must also include a brief statement of how much time you spent on the project. For example, "I spent 7 hours on this project from September 22-27 reading the book, designing a solution, writing code, fixing errors and putting together the printed document."
- Source code - a printout of your elegant source code for the `LotteryTicket`, `LotteryMachine` and `GUI` classes.