

---

# Introduction

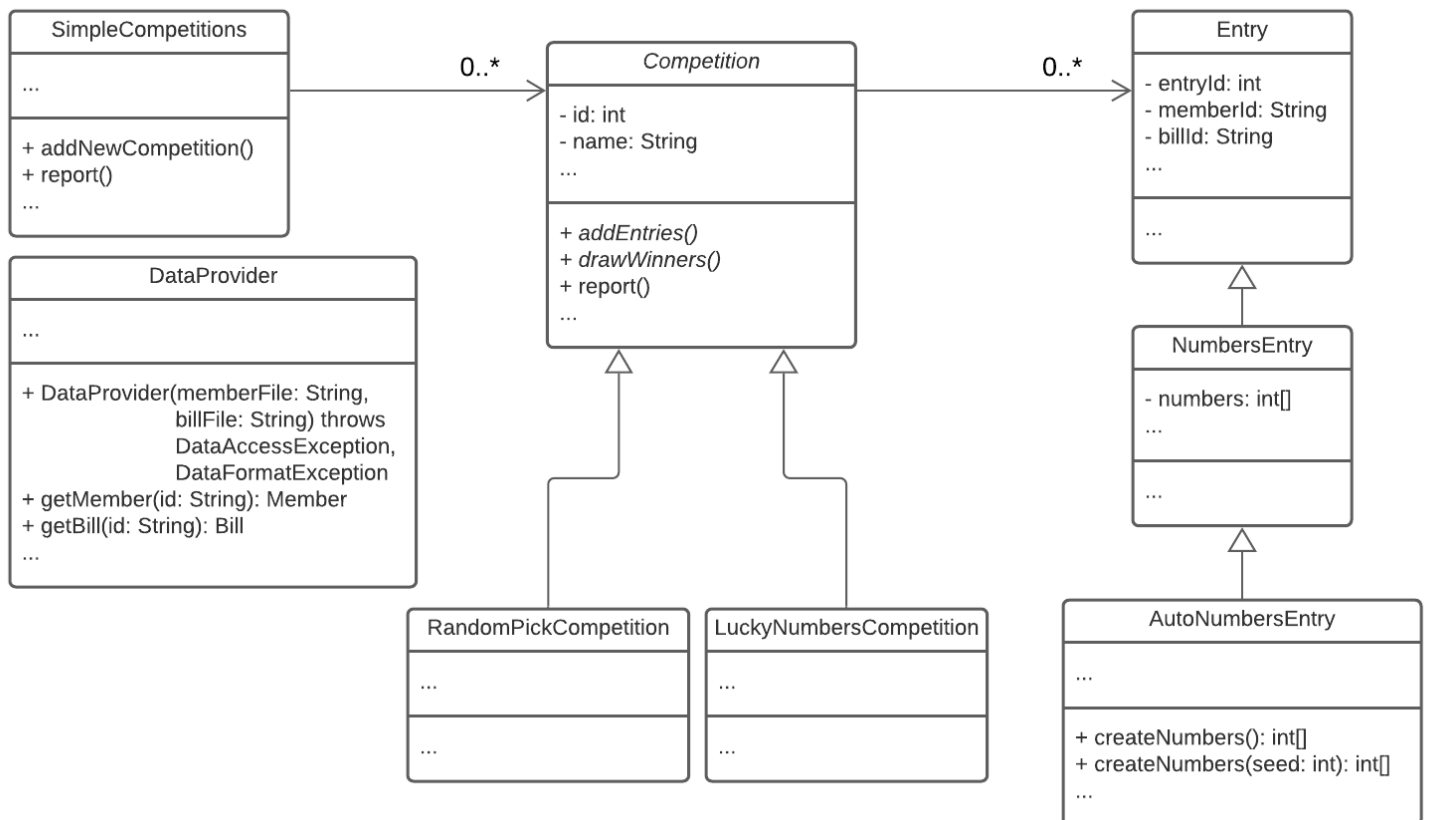
In this final project, you will develop a program called **SimpleCompetitions**. This program helps companies (e.g., retailers) create competitions to boost their sales. For instance, during the Easter Sale, a retailer can use your software to create a lucky draw game. In the game, a customer is given entries equivalent to the size of its purchase after their single-purchase final balance, after discounts, reach a specific amount, say \$50. Once the competition time ends, the retailer draws winners and the winning customers will get points added to their membership account; customers can use these points for future purchases if they wish. Please read the competition policy and application walk-through sections for more information. This final project is designed in such a way that we can evaluate your knowledge on all important topics you have learnt in this subject: 1) class design and implementation, 2) control flow structures (e.g., if-then-else & switch-case statements, loops), 3) basic I/O (e.g., Scanner class, formatted printing), 4) arrays, 5) inheritance & polymorphism, 6) exception handling, and 7) file I/O.

# Your Task

You will be provided a project scaffold containing 12 classes as listed below. All classes come with some initial implementations that follow the UML design shown in Figure 1.

**SimpleCompetitions.java** will be the main application file containing your `main()` method.

1. SimpleCompetitions.java
2. Competition.java
3. LuckyNumbersCompetition.java
4. RandomPickCompetition.java
5. Entry.java
6. NumbersEntry.java
7. AutoNumbersEntry.java
8. DataProvider.java
9. Member.java
10. Bill.java
11. DataAccessException.java
12. DataFormatException.java



**Figure 1. UML Class Diagram of SimpleCompetitions**

You are asked to add constructors, methods, and instance variables to complete the program. See the competition policy and application walk-through sections to understand the required features of the program. **Note that in the provided UML class diagram and the project scaffold, we only suggest methods' names; you can make any modifications to the methods' return types and their argument lists if necessary to complete the task.** You may add your own class(es) if you think it is necessary, but your program must contain the classes mentioned above. In this project, you are also asked to write Javadoc comments for classes and most important methods so that a documentation can be automatically generated. Please read the Documentation section to learn more about specific requirements for writing Javadoc comments.

**In this final projec, there is *NO ASSUMPTION* that all data (e.g., user inputs, file content) will be of correct data types. So error control for incorrectly entered data types ARE REQUIRED unless stated otherwise!**

# Competition Policy

In this project, your program must support two competition types (i.e., LuckyNumbers competition and RandomPick competition). Following is the competition policy that covers both of them.

- Only customers who have valid membership accounts can enter a competition. Your program should check for this condition, see the program walk-through section for more details.
- There is no limit on the number of competitions that a customer can enter. However, for this project, there is only one active competition (whose result is not decided) at a specific point in time i.e. the system does not support concurrent competitions.
- In a specific competition, one customer can have an unlimited number of entries based on their paid bills.
- Customers get one entry for each \$50 in a single bill. For example, if a total bill is \$245, it is eligible for getting four entries.
- In terms of entry creation, if a competition is of LuckyNumbers type, for each entry, the customer is asked to select 7 numbers in the range from 1 to 35. You can imagine that they are provided a card on which they can manually select numbers for some entries and/or ask the system to randomly generate entries. Completed cards must be returned to a staff member after purchases so that the information can be inputted into the system.
- In case of the RandomPick competition type, customers do not need to provide any numbers. Entries are automatically generated and each entry only has basic information like the Entry ID.
- Once the competition time ends, the organizer can draw winners. Note that if a customer has several winning entries, only the first one with the highest prize will be awarded. By saying first winning entry, we mean the entry that has smallest entry identifier (i.e., Entry ID).
- To find winners, in a LuckyNumbers competition, the system automatically generates a lucky entry that contains 7 lucky numbers. If an entry shares at least two numbers in common with the lucky entry, the owner of that entry would get some prize. The detailed prizes are listed on Table 1.

Division	Winning number required	Prize
1	7 numbers	50000 points
2	6 numbers	5000 points
3	5 numbers	1000 points
4	4 numbers	500 points
5	3 numbers	100 points
6	2 numbers	50 points

**Table 1. Prize table for LuckyNumbers competitions**

- In a RandomPick competition, the system automatically picks 3 winning entries from the entry list of the competition. The prizes for the first, second, and third winning entries are 50,000 points, 5,000 points, and 1,000 points, respectively.

# Application Walk-through

For this application, you are asked to support two different modes: 1) the **normal mode** (i.e. release mode), and 2) the **testing mode**. The testing mode is designed in such a way that you can test your program in a deterministic manner. The markers will also use that mode for running automated tests. The mode is required because this application is supposed to generate random numbers and it is hard to test its correctness without controlling the randomness. If a competition is of LuckyNumbers type, random numbers are used for both customers' entries and the lucky entries. If a competition is of RandomPick type, random numbers are used to pick the winning entries. More details about how to implement the testing mode are explained below.

**Note that all the user options are case insensitive unless stated otherwise. The program outputs (e.g., menus and output messages) look the same in both the normal and testing mode. The two modes are different only in the ways entries are generated and/or winning entries are chosen.**

1. Your program will start by displaying a welcome message and then ask if the user wants to load existing competitions from a file. When the user runs the program for the first time, they should say "No" by typing down 'N' or 'n'. Afterwards, they will need to choose one mode (normal or testing mode) to start the program. And then the program will ask for names of the files containing membership and bill information. Once the file names are given, your program should create an object of the **DataProvider** class and use it to retrieve data from those files. If there is something wrong (e.g., the file cannot be found/opened or the file data is not in the correct format), your program should print an error message and gracefully terminate.

```
----WELCOME TO SIMPLE COMPETITIONS APP----
Load competitions from file? (Y/N)?
N
Which mode would you like to run? (Type T for Testing, and N for Normal mode):
T
Member file:
members.csv
Bill file:
bills.csv
```


If the user chooses to load existing competitions from a binary file, they need to provide the file name. The program will try to load that file and if there is an exception, it should display an error message and terminate. If the competitions are successfully loaded, the user does not need to choose the running mode (Testing or Normal) because the mode should be the same as the existing competitions. The remaining logic is similar. That is, file names for membership and bill data are still required.

```
----WELCOME TO SIMPLE COMPETITIONS APP----
Load competitions from file? (Y/N)?
```

```
Y
File name:
demo.dat
Member file:
members.csv
Bill file:
bills.csv
```

Following are the content of the text files **members.csv** and **bills.csv** that are included in the project scaffold.

Each line in the file members.csv contains a member id, a member name, and an address.

 Member ids, names, and email addresses are required data fields for each line and these are string. Note that in this final project, you don't need to check if the email addresses follow the `xxx@yyy.zzz` format.

```
111111,John,john@abc.com
222222,Marry,marry@abc.com
333333,Atish,atish@abc.com
444444,Long,long@abc.com
555555,Liang,liang@abc.com
666666,Mia,mia@abc.com
777777,Louis,luis@abc.com
888888,Jenny,jenny@abc.com
```

Each line in the file bills.csv contains a bill id, a member id of the customer who paid for the bill, a bill total amount, and a boolean value showing whether that bill has been used in a competition or not. Some bills (e.g., 100013, and 100014) do not have member id; these represent the situations in which the customers did not have valid membership accounts or they did not give the information on their purchases. Based on the competition policy, those bills are ineligible to enter the competition.

```
100000,111111,300.5,false
100001,222222,340.8,false
100002,333333,125.9,false
100003,444444,256.8,false
100004,111111,145.6,false
100005,222222,300.0,false
100006,555555,200.5,false
100007,666666,500.5,false
100008,777777,234.5,false
100009,888888,490.6,false
100010,666666,201.6,false
100011,111111,400.5,false
100012,555555,123.9,false
100013,,400.5,false
100014,,345.8,false
100015,333333,100.5,false
100016,444444,236.7,false
100017,888888,123.4,false
100018,777777,500.8,false
100019,444444,348.9,false
```

2. If everything works, the program will display the main menu. The main menu has five options for 1) creating a new competition, 2) adding new entries for the current active competition, 3) drawing to find winners, 4) viewing a summary report of all competitions, and 5) exiting the program. Note that the user ***should not*** be able to 1) create a new competition if there is already an active one, or 2) add new entries or draw winners if there is no active competition. Error messages should be displayed if users try to do so. Following are some sample error messages. See all error messages in the visible test cases.

```
Please select an option. Type 5 to exit.  
1. Create a new competition  
2. Add new entries  
3. Draw winners  
4. Get a summary report  
5. Exit  
2  
There is no active competition. Please create one!  
Please select an option. Type 5 to exit.  
1. Create a new competition  
2. Add new entries  
3. Draw winners  
4. Get a summary report  
5. Exit  
3  
There is no active competition. Please create one!  
Please select an option. Type 5 to exit.  
1. Create a new competition  
2. Add new entries  
3. Draw winners  
4. Get a summary report  
5. Exit
```

3. If the first option ("Create a new competition") is selected, by typing "1" and then pressing Enter, the program should call the **addNewCompetition** method of the **SimpleCompetitions** class to add a new competition using the given competition type and competition name.

This is an example when the user chooses to create a new **LuckyNumbers** competition.

```
Please select an option. Type 5 to exit.  
1. Create a new competition  
2. Add new entries  
3. Draw winners  
4. Get a summary report  
5. Exit  
1  
Type of competition (L: LuckyNumbers, R: RandomPick)?:  
L  
Competition name:  
Easter Holidays  
A new competition has been created!  
Competition ID: 1, Competition Name: Easter Holidays, Type: LuckyNumbersCompetition
```

And this is another example when a new **RandomPick** competition is chosen.

```
Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit
1
Type of competition (L: LuckyNumbers, R: RandomPick)?:
R
Competition name:
Easter Holidays
A new competition has been created!
Competition ID: 1, Competition Name: Easter Holidays, Type: RandomPickCompetition
```

After creating a new competition, the program should go back to the main menu. The newly created competition becomes active and the user cannot create another competition until the current one finishes. If they try to do so, an error message should be displayed.

```
Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit
1
There is an active competition. SimpleCompetitions does not support concurrent competitions!
```

4. Now the user can choose the second option ("Add new entries") to add entries. The program should first ask for the bill ID (i.e. receipt number). The member ID and the total amount of the bill can be retrieved from the given bill file, which is **bills.csv** in this example. The bill ID needs to follow a specific format -- it must be a 6-digit number and the number can start with zero(es). For example, both 001234 and 123456 are valid identifiers but 123abc is not. Moreover, according to the competition policy, only customers who have valid membership accounts are eligible to enter a competition. Your program should also check if a bill has already been used for a competition.

```
Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit
2
Bill ID:
123abc
Invalid bill id! It must be a 6-digit number. Please try again.
Bill ID:
100013
This bill has no member id. Please try again.
Bill ID:
```



```
100000
```

This bill (\$300.5) is eligible for 6 entries. How many manual entries did the customer fill up?:

5. Once the bill ID passes all the required checks, your program will generate entries for the bill. The total bill amount should be used to calculate how many entries the customer can get for this bill. ***The two competition types generate entries in different ways.*** In case of LuckyNumbers competition, your program should ask for the number of manual entries for this bill. In the above example, the customer is eligible to get six entries for their bill. Suppose that they manually filled up one entry, by selecting 7 numbers in the range from 1 to 35, and asked for others to be automatically generated. The program output should look like the following. **Note that manual entries should be added first and entries' indices start from 1 for each new competition. Moreover, all entry numbers should be sorted in an ascending order.**

```
This bill ($300.5) is eligible for 6 entries. How many manual entries did the customer fill up?:
1
Please enter 7 different numbers (from the range 1 to 35) separated by whitespace.
1 2 3 4
Invalid input! Fewer than 7 numbers are provided. Please try again!
Please enter 7 different numbers (from the range 1 to 35) separated by whitespace.
1 2 3 4 9 6 7 10
Invalid input! More than 7 numbers are provided. Please try again!
Please enter 7 different numbers (from the range 1 to 35) separated by whitespace.
1 2 3 4 9 9 5
Invalid input! All numbers must be different!
Please enter 7 different numbers (from the range 1 to 35) separated by whitespace.
1 2 3 4 9 5 45
Invalid input! All numbers must be in the range from 1 to 35!
Please enter 7 different numbers (from the range 1 to 35) separated by whitespace.
1 2 3 4 10 7 5
The following entries have been added:
Entry ID: 1      Numbers: 1  2  3  4  5  7 10
Entry ID: 2      Numbers: 3  4 17 18 24 29 30 [Auto]
Entry ID: 3      Numbers: 2  7 14 18 22 25 35 [Auto]
Entry ID: 4      Numbers: 1  5  6  8 31 32 35 [Auto]
Entry ID: 5      Numbers: 2  5 11 12 23 24 34 [Auto]
Entry ID: 6      Numbers: 1  2 13 24 25 31 34 [Auto]
Add more entries (Y/N)?
```

6. The auto entries can be generated in different ways. In this project we leverage the shuffle method of the Collections class (<https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html>). We provide the implementation for the **createNumbers** method in the **AutoNumbersEntry** class. **The randomness is controlled by the seed argument. In the testing mode, you should use the competition identifier as seed for generating the lucky entry and the number of entries in the currently active competition to generate automated customers' entries.**

```
private final int NUMBER_COUNT = 7;
private final int MAX_NUMBER = 35;

public int[] createNumbers (int seed) {
    ArrayList<Integer> validList = new ArrayList<Integer>();
```

```

    int[] tempNumbers = new int[NUMBER_COUNT];
    for (int i = 1; i <= MAX_NUMBER; i++) {
        validList.add(i);
    }
    Collections.shuffle(validList, new Random(seed));
    for (int i = 0; i < NUMBER_COUNT; i++) {
        tempNumbers[i] = validList.get(i);
    }
    Arrays.sort(tempNumbers);
    return tempNumbers;
}

```

7. If the competition is of RandomPick type, your program will use a simpler algorithm to generate the entries. Specifically, it just generate basic entries (objects of the base Entry class) -- no numbers are required.

```

Bill ID:
100000
This bill ($300.5) is eligible for 6 entries.
The following entries have been automatically generated:
Entry ID: 1
Entry ID: 2
Entry ID: 3
Entry ID: 4
Entry ID: 5
Entry ID: 6
Add more entries (Y/N)?

```

8. The user can then add more entries into the current competition. Your program must check if the given bill ID has already been used. If so, an error message should be displayed.

```

Add more entries (Y/N)?
Y
Bill ID:
100000
This bill has already been used for a competition. Please try again.
Bill ID:
100001
This bill ($340.8) is eligible for 6 entries. How many manual entries did the customer fill up?:
0
The following entries have been added:
Entry ID: 7      Numbers:  2  3 15 18 26 27 31 [Auto]
Entry ID: 8      Numbers:  4  6 14 16 17 18 20 [Auto]
Entry ID: 9      Numbers:  2 13 26 28 29 32 34 [Auto]
Entry ID: 10     Numbers:  1  6 11 13 14 18 24 [Auto]
Entry ID: 11     Numbers:  5  7 16 21 23 26 28 [Auto]
Entry ID: 12     Numbers:  2 10 13 21 23 24 27 [Auto]
Add more entries (Y/N)?

```

The user can choose 'N' to go back to the main menu. At the main menu, they might want to choose the 2nd option to add more entries or choose the 3rd option to draw and find winners.

9. If the 3rd option from the main menu is selected and the current active competition is of LuckyNumbers type, the program will automatically generate a lucky entry using the same algorithm it used to generate automated user entries. **In the testing mode, your program should use the competition identifier as seed for generating the lucky entry.** The program then display the information of the lucky numbers and all the winning entries with their prizes. The entries are sorted by their identifiers. **As stated in the competition policy, if a customer has several winning entries, only the first one with the highest prize will be awarded. By saying first winning entry, we mean the entry that has smallest entry identifier.**

```
Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit
3
Competition ID: 1, Competition Name: Easter Holidays, Type: LuckyNumbersCompetition
Lucky Numbers: 3 4 17 18 24 29 30 [Auto]
Winning entries:
Member ID: 111111, Member Name: John, Prize: 50000
--> Entry ID: 2, Numbers: 3 4 17 18 24 29 30 [Auto]
Member ID: 222222, Member Name: Marry, Prize: 100
--> Entry ID: 8, Numbers: 4 6 14 16 17 18 20 [Auto]
Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit
```

10. If the current competition is of RandomPick type, your program should randomly pick three winning entries from the entry list. Please use the following piece of code to implement the **drawWinners** method of the RandomPickCompetition class. **Note that this piece of code does not include the code to ensure that one customer gets at most one winning entry.** You may observe the competition ID is used as seed for the random generator in the testing mode.

```
private final int FIRST_PRIZE = 50000;
private final int SECOND_PRIZE = 5000;
private final int THIRD_PRIZE = 1000;
private final int[] prizes = {FIRST_PRIZE, SECOND_PRIZE, THIRD_PRIZE};

private final int MAX_WINNING_ENTRIES = 3;

public void drawWinners() {
    Random randomGenerator = null;
    if (this.getIsTestingMode()) {
        randomGenerator = new Random(this.getId());
    } else {
        randomGenerator = new Random();
    }
}
```

```

int winningEntryCount = 0;
while (winningEntryCount < MAX_WINNING_ENTRIES) {
    int winningEntryIndex = randomGenerator.nextInt(entries.size());

    Entry winningEntry = entries.get(winningEntryIndex);

    /*
        * Ensure that once an entry has been selected,
        * it will not be selected again.
        */
    if (winningEntry.getPrize() == 0) {
        int currentPrize = prizes[winningEntryCount];
        winningEntry.setPrize(currentPrize);
        winningEntryCount++;
    }
}

/*
    * Note that the above piece of code does not ensure that
    * one customer gets at most one winning entry. Add your code
    * to complete the logic.
    */
}

```

And following is a sample output after running the drawWinners method of the RandomPickCompetition class.

```

Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit
3
Competition ID: 1, Competition Name: Easter Holidays, Type: RandomPickCompetition
Winning entries:
Member ID: 111111, Member Name: John, Entry ID: 5, Prize: 5000
Member ID: 222222, Member Name: Marry, Entry ID: 10, Prize: 50000
Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit

```

11. Now the user can follow similar steps, as explained above, to create other competitions, or they can use the 4th option to display a summary report, or the 5th option to terminate the program.

```

Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report

```

```

5. Exit
1
Type of competition (L: LuckyNumbers, R: RandomPick)? :
L
Competition name:
Anzac Day
A new competition has been created!
Competition ID: 2, Competition Name: Anzac Day, Type: LuckyNumbersCompetition
Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit
2
Bill ID:
100003
This bill ($256.8) is eligible for 5 entries. How many manual entries did the customer fill up?:
0
The following entries have been added:
Entry ID: 1      Numbers:  8 19 22 23 28 30 32 [Auto]
Entry ID: 2      Numbers:  3  4 17 18 24 29 30 [Auto]
Entry ID: 3      Numbers:  2  7 14 18 22 25 35 [Auto]
Entry ID: 4      Numbers:  1  5  6  8 31 32 35 [Auto]
Entry ID: 5      Numbers:  2  5 11 12 23 24 34 [Auto]
Add more entries (Y/N)?
N
Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit
4
----SUMMARY REPORT----
+Number of completed competitions: 1
+Number of active competitions: 1

Competition ID: 1, name: Easter Holidays, active: no
Number of entries: 12
Number of winning entries: 2
Total awarded prizes: 55000

Competition ID: 2, name: Anzac Day, active: yes
Number of entries: 5
Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit

```

12. If the user selects the fifth option from the main menu, your program will ask if they want to save

competitions to a binary file. If the answer is "No", your program will say "Goodbye!" before exiting.

```
Please select an option. Type 5 to exit.  
1. Create a new competition  
2. Add new entries  
3. Draw winners  
4. Get a summary report  
5. Exit  
5  
Save competitions to file? (Y/N)?  
N  
Goodbye!
```

Otherwise, the user needs to provide the file name, say demo.dat. All competitions should be saved to the file in a binary format (using the writeObject method of the ObjectOutputStream class). The bill file (bills.csv in this example) should also be updated/overwritten as well to make the whole system consistent. **The first three columns of the bill file (bill IDs, member IDs, and bill total) should not be modified. Only the last column, which records if a bill has been used in a competition, should be updated.**

```
Please select an option. Type 5 to exit.  
1. Create a new competition  
2. Add new entries  
3. Draw winners  
4. Get a summary report  
5. Exit  
5  
Save competitions to file? (Y/N)?  
Y  
File name:  
demo.dat  
Competitions have been saved to file.  
The bill file has also been automatically updated.  
Goodbye!
```

13. Since there is a binary file (demo.dat) keeping the existing competitions, now the user can start the SimpleCompetitions program again with that file and the up-to-date bill file. Once the program is initialised, the user can continue working on their previous work (i.e. the competition for Anzac Day in this example). As shown in the following program summary report, all the existing competitions have been successfully loaded.

```
----WELCOME TO SIMPLE COMPETITIONS APP----  
Load competitions from file? (Y/N)?  
Y  
File name:  
demo.dat  
Member file:  
members.csv  
Bill file:  
bills.csv  
Please select an option. Type 5 to exit.
```

1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit

4

----SUMMARY REPORT----

+Number of completed competitions: 1

+Number of active competitions: 1

Competition ID: 1, name: Easter Holidays, active: no

Number of entries: 12

Number of winning entries: 2

Total awarded prizes: 55000

Competition ID: 2, name: Anzac Day, active: yes

Number of entries: 5

---

## Example Executions

In this section, we show two example executions in the testing mode with and without existing competitions in file. Note that the program logic for testing and normal mode is almost the same and the outputs look similar. The two modes are only different in the ways random numbers are generated. Specifically, for the testing mode, since the randomness is controlled by using seeds, if you provide the same input, the program should produce the same output. **Note that in the example executions, we show both the input and output of the program. In the given test cases, the input and output are stored in different files.**

1. Example execution without existing competitions. The user starts running the program from scratch.

```
----WELCOME TO SIMPLE COMPETITIONS APP----
Load competitions from file? (Y/N)?
A
Unsupported option. Please try again!
Load competitions from file? (Y/N)?
N
Which mode would you like to run? (Type T for Testing, and N for Normal mode):
A
Invalid mode! Please choose again.
Which mode would you like to run? (Type T for Testing, and N for Normal mode):
T
Member file:
members.csv
Bill file:
bills.csv
Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit
A
A number is expected. Please try again.
Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit
6
Unsupported option. Please try again!
Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
```



4. Get a summary report

5. Exit

2

There is no active competition. Please create one!

Please select an option. Type 5 to exit.

1. Create a new competition

2. Add new entries

3. Draw winners

4. Get a summary report

5. Exit

3

There is no active competition. Please create one!

Please select an option. Type 5 to exit.

1. Create a new competition

2. Add new entries

3. Draw winners

4. Get a summary report

5. Exit

4

No competition has been created yet!

Please select an option. Type 5 to exit.

1. Create a new competition

2. Add new entries

3. Draw winners

4. Get a summary report

5. Exit

1

Type of competition (L: LuckyNumbers, R: RandomPick)?:

A

Invalid competition type! Please choose again.

Type of competition (L: LuckyNumbers, R: RandomPick)?:

R

Competition name:

Easter Holidays

A new competition has been created!

Competition ID: 1, Competition Name: Easter Holidays, Type: RandomPickCompetition

Please select an option. Type 5 to exit.

1. Create a new competition

2. Add new entries

3. Draw winners

4. Get a summary report

5. Exit

2

Bill ID:

123abc

Invalid bill id! It must be a 6-digit number. Please try again.

Bill ID:

111111

This bill does not exist. Please try again.

Bill ID:

100013

This bill has no member id. Please try again.

Bill ID:

100000

This bill (\$300.5) is eligible for 6 entries.

The following entries have been automatically generated:

Entry ID: 1

Entry ID: 2

Entry ID: 3

Entry ID: 4

Entry ID: 5

Entry ID: 6

Add more entries (Y/N)?

Y

Bill ID:

100000

This bill has already been used for a competition. Please try again.

Bill ID:

100001

This bill (\$340.8) is eligible for 6 entries.

The following entries have been automatically generated:

Entry ID: 7

Entry ID: 8

Entry ID: 9

Entry ID: 10

Entry ID: 11

Entry ID: 12

Add more entries (Y/N)?

N

Please select an option. Type 5 to exit.

1. Create a new competition

2. Add new entries

3. Draw winners

4. Get a summary report

5. Exit

1

There is an active competition. SimpleCompetitions does not support concurrent competitions!

Please select an option. Type 5 to exit.

1. Create a new competition

2. Add new entries

3. Draw winners

4. Get a summary report

5. Exit

3

Competition ID: 1, Competition Name: Easter Holidays, Type: RandomPickCompetition

Winning entries:

Member ID: 111111, Member Name: John, Entry ID: 5, Prize: 5000

Member ID: 222222, Member Name: Marry, Entry ID: 10, Prize: 50000

Please select an option. Type 5 to exit.

1. Create a new competition

2. Add new entries

3. Draw winners

4. Get a summary report

5. Exit

1

Type of competition (L: LuckyNumbers, R: RandomPick)?:

L

Competition name:

Anzac Day

A new competition has been created!

Competition ID: 2, Competition Name: Anzac Day, Type: LuckyNumbersCompetition

Please select an option. Type 5 to exit.

1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit

3

The current competition has no entries yet!

Please select an option. Type 5 to exit.

1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit

4

----SUMMARY REPORT----

+Number of completed competitions: 1

+Number of active competitions: 1

Competition ID: 1, name: Easter Holidays, active: no

Number of entries: 12

Number of winning entries: 2

Total awarded prizes: 55000

Competition ID: 2, name: Anzac Day, active: yes

Number of entries: 0

Please select an option. Type 5 to exit.

1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit

2

Bill ID:

100001

This bill has already been used for a competition. Please try again.

Bill ID:

100003

This bill (\$256.8) is eligible for 5 entries. How many manual entries did the customer fill up?:

1

Please enter 7 different numbers (from the range 1 to 35) separated by whitespace.

1 2 3 4 10 7 5

The following entries have been added:

Entry ID: 1        Numbers: 1 2 3 4 5 7 10

Entry ID: 2        Numbers: 3 4 17 18 24 29 30 [Auto]

Entry ID: 3        Numbers: 2 7 14 18 22 25 35 [Auto]

Entry ID: 4        Numbers: 1 5 6 8 31 32 35 [Auto]

Entry ID: 5        Numbers: 2 5 11 12 23 24 34 [Auto]

Add more entries (Y/N)?

N

Please select an option. Type 5 to exit.

```
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit
5
Save competitions to file? (Y/N)?
Y
File name:
demo.dat
Competitions have been saved to file.
The bill file has also been automatically updated.
Goodbye!
```

2. Example execution with existing competitions saved in a binary file named demo.dat. This is the second run of the SimpleCompetitions program, after the above execution.

```
----WELCOME TO SIMPLE COMPETITIONS APP----
Load competitions from file? (Y/N)?
Y
File name:
demo.dat
Member file:
members.csv
Bill file:
bills.csv
Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit
4
----SUMMARY REPORT----
+Number of completed competitions: 1
+Number of active competitions: 1

Competition ID: 1, name: Easter Holidays, active: no
Number of entries: 12
Number of winning entries: 2
Total awarded prizes: 55000

Competition ID: 2, name: Anzac Day, active: yes
Number of entries: 5
Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit
1
There is an active competition. SimpleCompetitions does not support concurrent competitions!
Please select an option. Type 5 to exit.
```

```
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit
2
Bill ID:
100004
This bill ($145.6) is eligible for 2 entries. How many manual entries did the customer fill up?:
0
The following entries have been added:
Entry ID: 6      Numbers:  1  2 13 24 25 31 34 [Auto]
Entry ID: 7      Numbers:  2  3 15 18 26 27 31 [Auto]
Add more entries (Y/N)?
Y
Bill ID:
100005
This bill ($300.0) is eligible for 6 entries. How many manual entries did the customer fill up?:
0
The following entries have been added:
Entry ID: 8      Numbers:  4  6 14 16 17 18 20 [Auto]
Entry ID: 9      Numbers:  2 13 26 28 29 32 34 [Auto]
Entry ID: 10     Numbers:  1  6 11 13 14 18 24 [Auto]
Entry ID: 11     Numbers:  5  7 16 21 23 26 28 [Auto]
Entry ID: 12     Numbers:  2 10 13 21 23 24 27 [Auto]
Entry ID: 13     Numbers:  1  4 14 16 26 28 32 [Auto]
Add more entries (Y/N)?
N
Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit
3
Competition ID: 2, Competition Name: Anzac Day, Type: LuckyNumbersCompetition
Lucky Numbers:  2  7 14 18 22 25 35 [Auto]
Winning entries:
Member ID: 444444, Member Name: Long, Prize: 50000
--> Entry ID: 3, Numbers:  2  7 14 18 22 25 35 [Auto]
Member ID: 111111, Member Name: John, Prize: 50
--> Entry ID: 6, Numbers:  1  2 13 24 25 31 34 [Auto]
Member ID: 222222, Member Name: Marry, Prize: 50
--> Entry ID: 8, Numbers:  4  6 14 16 17 18 20 [Auto]
Please select an option. Type 5 to exit.
1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit
4
-----SUMMARY REPORT-----
+Number of completed competitions: 2
+Number of active competitions: 0
```

Competition ID: 1, name: Easter Holidays, active: no  
Number of entries: 12  
Number of winning entries: 2  
Total awarded prizes: 55000

Competition ID: 2, name: Anzac Day, active: no  
Number of entries: 13  
Number of winning entries: 3  
Total awarded prizes: 50100

Please select an option. Type 5 to exit.

1. Create a new competition
2. Add new entries
3. Draw winners
4. Get a summary report
5. Exit

5

Save competitions to file? (Y/N)?

Y

File name:

demo.dat

Competitions have been saved to file.

The bill file has also been automatically updated.

Goodbye!

---

# SimpleCompetitions

In this final project, you will develop a program called **SimpleCompetitions**. This program helps companies (e.g., retailers) create competitions to boost their sales. For instance, during the Easter Sale, a retailer can use your software to create a lucky draw game. In the game, a customer is given entries equivalent to the size of its purchase after their single-purchase final balance, after discounts, reach a specific amount, say \$50. Once the competition time ends, the retailer draws winners and the winning customers will get points added to their membership account; customers can use these points for future purchases if they wish. Please read the competition policy and application walk-through sections for more information. This final project is designed in such a way that we can evaluate your knowledge on all important topics you have learnt in this subject: 1) class design and implementation, 2) control flow structures (e.g., if-then-else & switch-case statements, loops), 3) basic I/O (e.g., Scanner class, formatted printing), 4) arrays, 5) inheritance & polymorphism, 6) exception handling, and 7) file I/O.

---

# Documentation

Always make sure to document your code in general.

## Javadoc

Make sure all your classes indicate their author and general description. For each constructor and method specified in the final project description, you must provide at least the following tags:

- @param tags
- @returns tag
- @throws



If your method is a void one, you don't need to use the @returns tag. Similarly, you don't need to write @throws tags if your method does not throw an exception.

You can leave out minor helper functions that you may have added and setters/getters. Make sure to test the correct generation of your documentation using [javadoc](#).

For the submission, you do not need to generate and add the javadoc (i.e., HTML files generated by running the javadoc command) to edstem. For marking, we will create the javadoc programmatically from your code.



---

# Assessment

**The deadline for this project is 5.30pm (Melbourne time), June 20, 2022. This is the HARD DEADLINE, late submissions are not allowed!!**

The final project is worth 60% of the total marks for the subject, including the oral exam (40/20). If you fail to submit any code or it becomes obvious that no serious programming attempt has been made, no oral exam will be conducted and the overall project marked 0. Your Java program will be assessed based on the correctness of the output as well as the quality of code implementation.

We will use automated tests to compile, run, and compare your outputs for several test cases with generated expected outputs. The automatic test will deem your output wrong if your output does not match the expected output, even if the difference is just having extra space or missing a colon. Therefore, it is crucial that your output follows exactly the same format shown in the provided examples.



Passing the tests we provide here on edstem does not mean you will score full marks. Each submission will further be subject to hidden tests and, more importantly, manual inspection by human markers.

But, more importantly, we expect you to apply good coding practices and make use of appropriate concepts as discussed throughout the lecture. A general marking rubric will be released here on edstem.



Your submission is only valid if your code compiles and runs on edstem! Arguments, such as *"It runs on my local machine"*, can and will not be accepted.

## What will be graded?

We will only mark the **last** version of your program committed before the submission deadline. So make sure you submit code to edstem early and regularly.

Be aware of timezone differences! Submissions via email or Canvas will not be accepted.

BUT we believe in you! And are looking forward to your unique and ingenious solutions to this fascinating problem!

# Marking Scheme □



**Warning! Please make sure your code runs on edstem. If your code does not compile on edstem, 0 marks will be given.**

## COMP90041 Final Project: Marking Scheme

Student: ***[Student ID goes here]***

### 1. Program Presentation

Including: layout and style, readability, adherence to coding expectations and conventions, general care and appearance. The full marks for this section of marking is **6**.

#### Gaining marks

Some subset of the following lines will be selected by the marker.

Gain **0.5** marks for any of the achievements listed below.

- all choices of variable names were meaningful;
- all choices of method names were meaningful;
- variable and method names follow Java convention (camelCase);
- constant names follow Java convention (UPPER\_SNAKE\_CASE);
- class names follow Java convention (UpperCamelCase);
- comments were sufficient and meaningful;
- consistent bracket placement;
- indentation was consistent;
- whitespace was used to separate logical blocks of code;
- authorship statement (name, university email, student number) provided;
- all magic numbers (essentially numbers other than 0 or 1) were assigned to constants;
- overall care and presentation, such as particularly good comments, up to **+1.0** mark;

#### Deductions

- stylistic issue, if major **-1.0** mark per issue; if minor, **-0.5** mark per issue

Total marks for this section: **/ 6**

Other comments from marker: ***[Comments go here]***

## 2. Javadoc

If Javadoc cannot be created (using the command "javadoc -d doc \*.java"), you will lose all marks in this section.

Gain 2 marks if:

- Javadoc can be created without errors.
- Following Javadoc conventions.
- All major classes and methods are annotated.

### Deductions

- missing javadoc comments of major classes/methods or do not follow javadoc conventions, if major **-1.0** mark per issue; if minor, **-0.5** mark per issue

Total marks for this section: / 2

Other comments from marker: **[Comments go here]**

## 3. Structure and Approach

Including: decomposition into methods, declaration of instance variables at the appropriate locations, choice of parameters to methods.

The full marks for this section of marking is **14**.

### Gaining marks

Some subset of the following lines will be selected by the marker.

Gain **1** mark for any achievement listed below.

- Follow the given design as modelled in the UML diagram;
- Good use of methods;
- No methods were too long or too complex;
- code was not duplicated (tasks to be done in more than one place are in method);
- simple algorithmic approach to solving the problem;
- clearly structured code;
- no more than 3 `static` methods (including main) were used -- most should be bound to objects;
- no more than 4 `static` variables were used -- most should be non-static ("`final static`" constants are not considered variables);

- appropriate use of encapsulation and polymorphism; limiting scope of variables to `private` or `protected` except where reasonable justification is given.
- constants were defined as `final static`;



If constants are not used externally, we also accept definitions using "private final" without the "static" keyword.

- exceptions were appropriately thrown when required, and try-catch blocks were appropriately used (this usually means the catch is in a different method from the throw, though not always);
- custom exceptions `DataAccessException` and `DataFormatException` were implemented with appropriate constructors;
- classes were implemented using inheritance relationships where appropriate;
- all required classes (as modelled in the given UML class diagram) were used in the implementation of the system.

## Deductions

- structural & design & implementation issues, if major **-2.0** marks per issue, otherwise **-1.0** per issue.

Total marks for this section: / **14**

Other comments from marker: **[Comments go here]**

## 4. Program Execution

Including: compilation, execution on test data, output presentation and readability.

Programs that do not compile in the test environment will lose all marks in this section. Be sure to verify your submission and **check the output** before you say "finished" to yourself.

The full marks for this section of marking is **18**.

### Marks awarded

Gain **1** mark per test case passed.

## Deductions

Gain **0.5** marks for visible tests with **slightly** different output (e.g., small changes in whitespace; if the marker says the difference is not slight, that is final. This should not occur if the available test cases are all checked.);



Note that Visible Test **#2** and Test **#4** are used to test your program in the normal mode. These tests are considered fully passed and you get full mark (1 mark for each test) if your program produces the same output as the sample one except 1) the content of the automatically generated entries (for the `LuckyNumbersCompetition`), and 2) the winners & winning entries (for both types of competitions).



Assessment Tests (hidden tests) are used to evaluate how well your program handles corner cases that could lead to unexpected behaviours e.g., unhandled exceptions, infinite loops. A hidden test case is passed if given a test input file, the program under test does not introduce infinite loops and it can capture the correct exception types. By the "correct exception types", we mean that you must capture specific exceptions (e.g., `InputMismatchException` or `FileNotFoundException`) before the generic `Exception` class. Note that for these test cases, we do not check for exact same output (i.e., error messages outputted by the your program can be different from messages generated by the sample solution). However, we do check for the logic of the error-handling code. For instance, it does not make sense to terminate the program just because the user provides a non-numeric string when a number is expected.

Gain **0** for other failed tests

Visible tests passed: / 8

Assessment tests passed: / 10

Total tests passed: / 18

Total marks for this section: / **18**

**Total Marks for the Assignment: / 40**

Overall comments from marker: **[Comments go here]**

Assignment Marker: **[Assignment marker name goes here]**

If you have any questions regarding your mark, please contact Dr. Thuan Pham ([thuan.pham@unimelb.edu.au](mailto:thuan.pham@unimelb.edu.au)).