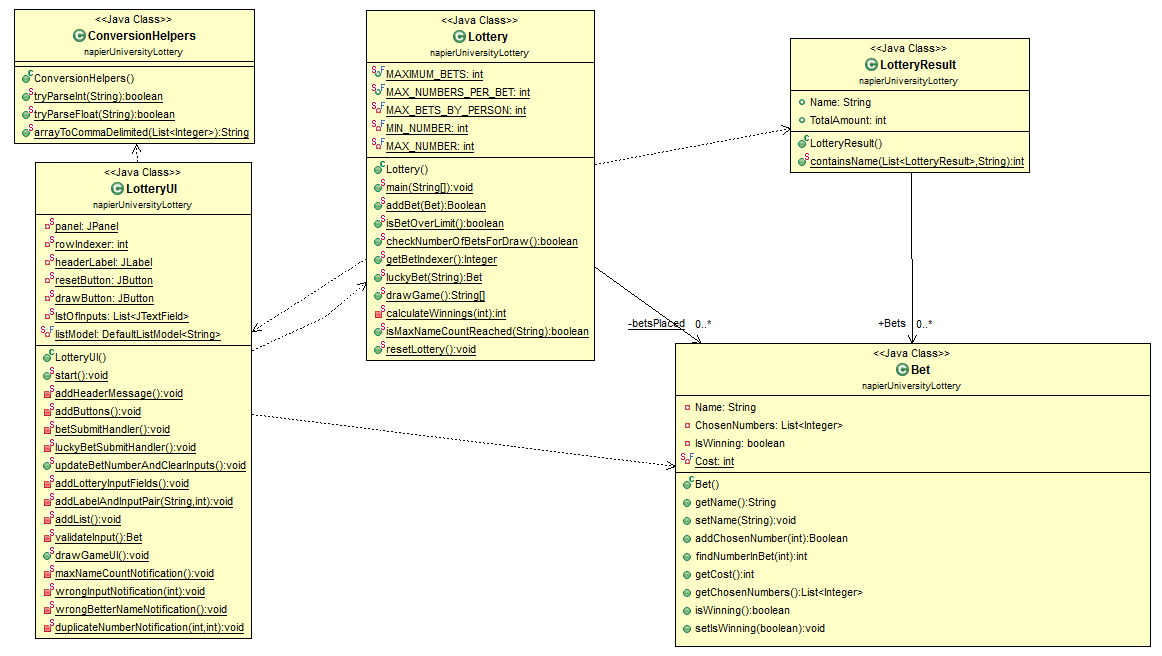
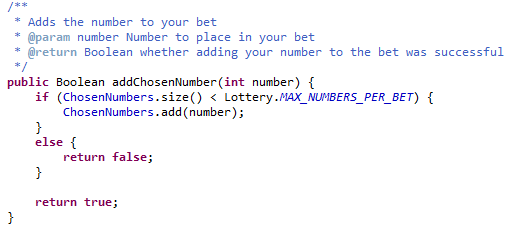
1. **Class Diagram**
2. **Description of classes:**

|  |  |  |
| --- | --- | --- |
| Bet class | Fields | Functions |
|  | Name (String) | getName |
|  | ChosenNumbers (List<Integer>) | setName |
|  | IsWinning (Boolean) | addChosenNumber |
|  | Cost (static final Integer) | findNumberInBet |
|  |  | getCost |
|  |  | getChosenNumbers |
|  |  | isWinning |
|  |  | setIsWinning |

A bet is always associated with a person so the **Name** field (String object) will contain the user placing the bet. All numbers will be added to the **ChosenNumbers** field (List of integers) with the help of ‘setter’ functions that will expose adding capability publicly to other classes**,** namely **addChosenNumber(int** number) will add a parsed number from the input if the number of betting numbers haven’t exceeded the maximum amount allowed. **findNumberInBet** function will return the index of the element’s location in the collection or if it doesn’t exist it will return -1.

**getName** and **setName** are just getter and setter functions for the private property called Name, which stores the name of the person entering the bet.

**IsWinning** and **setIsWinning** are getters and setters to change or retrieve the value of the property called IsWinning. This property defaults to a value of **‘false’** and is only changed when the lottery is drawn and the current bet has any numbers matching the winning numbers drawn.

|  |  |  |
| --- | --- | --- |
| Lottery class (Main) | Fields | Functions |
|  | betsPlaced (List<Bet>) | main() |
|  | MAXIMUM\_BETS (final int) | addBet(Bet b) |
|  | MAX\_NUMBERS\_PER\_BET (final int) | isBetOverLimit() |
|  | MAX\_BETS\_BY\_PERSON (final int) | checkNumberOfBetsForDraw() |
|  | MIN\_NUMBER (final int) | getBetIndexer() |
|  | MAX\_NUMBER (final int) | luckyBet(String name) |
|  |  | drawGame() |
|  |  | calculateWinnings(int numberOfMatchingNumbers) |
|  |  | isMaxNameCountReached(String betterName) |
|  |  | resetLottery() |

The **Lottery** class is responsible for the running of the application and it also keeps track of the bets placed.

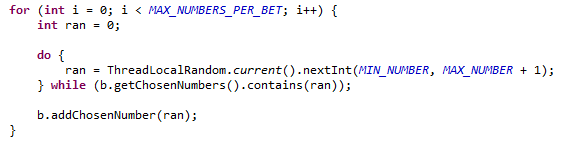
It contains the fields and functions required to define the constraints of the game as well as for allowing a bet to be added to the lottery, checking the number of bets in the lottery, checking the bets placed by the same person, placing a ‘Lucky Bet’ and of course drawing the winning numbers and calculating the winnings/loss for winning bets in the lottery.

*The image shows an extract of the Lottery class.*

|  |  |  |
| --- | --- | --- |
| LotteryResult class | Fields | Functions |
|  | Name (String) | containsName(List<LotteryResult>, String) |
|  | Bets (List<Bet>) |  |
|  | TotalAmount (int) |  |

The **LotteryResult** class is used for constructing and calculating the results of a draw. It is a data structure to hold the bets grouped by their owners and the TotalAmount of winnigs/losses per person. This was required to solve the task for the advanced features where the requirements were to group bets by their buyer’s.

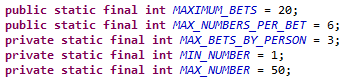
The **LotteryUI** is the class that displays the whole application and shows the relevant labels and input fields required to retrieve the numbers from the user. It contains the input validation functions and notification functions that will be triggered when the user enters something that’s either not allowed or not the required input.

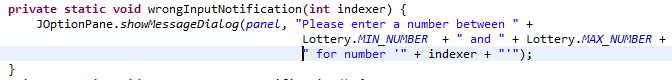
1. **No duplication**

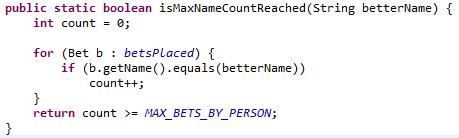
When drawing the lottery the application keeps generating a random number until the number it generated isn’t a duplicate one. This way, having duplicate numbers drawn can be avoided.

1. **Extensibility and advanced features**

The application can be easily changed to allow users to place more numbers per bets or more bets for a single draw.

The global constants define the constraints of the application. For example if you were to change the value of **MAX\_NUMBERS\_PER\_BET** the number of input fields on the UI will match that value.

If you were planning to change to lottery to allow a different range of numbers, just change the global constants and the application will adjust. For example if you enter a number that’s outside that range, the notification will always show you the range defined by the constants.

The number bets by person are limited by having a function check previous values each time the application tries to add a new Bet to the **betsPlaced** variable as seen on the screenshot to the right.

The winnings are displayed by person using an extra class (**LotteryResult)** that allows the results to be grouped by names. The **drawGame()** function in **Lottery** class contains the logic for grouping the names together and calculating total winnings/loss per person.