# Motivation

1. Design, and develop scalable, high-performance, highly reliable and secure Java EE applications.
2. Perform development using TDD and agile methodologies including code reviews, retrospectives and CI.
3. Be active in always striving to improve my skill set.

# User Requirements

Tommy would like to ask you to write **a console application** to replace his old ball machine and tickets.

1. **Algorithm:**  The old ball machine that contains 50 balls numbered from 1 to 50. After mixing the balls, 3 balls are drawn at random. The first ball drawn wins 75%, the second wins 15%, and the third wins 10% of the available prize money. This available prize money corresponds to 50% of the total money in the pot at draw time.
2. **Processing steps:**

**Step1:** When client buys a ticket, need to provide a first name. At purchase time, the number of the ball is displayed on the screen.

**Step2:** Tommy starts to draw.

**Step3:** Display the winning tickets as follows:

1st ball 2nd ball 3rd ball

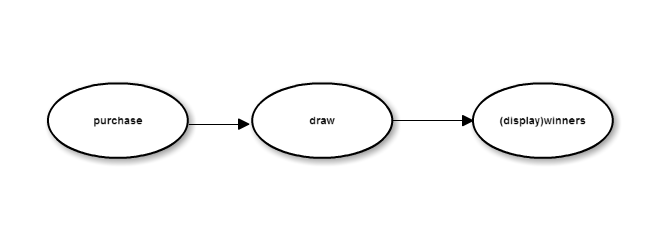
Dave: 75$ Remy: 15$ Greg: 10$

1. Tommy only executed with the commands “**purchase**”, “**draw**” and “**winners**”.
2. A ticket anytime at the price of 10$. Upon each draw, Tommy uses a new series of 50 tickets numbered the same way as the balls in the ball machine.
3. Upon launching the program, you can initialize the pot with 200$.
4. You can round or truncate amounts to the nearest dollar value.
5. The program does not need to store information on disk nor in a database.

# Problem Statement

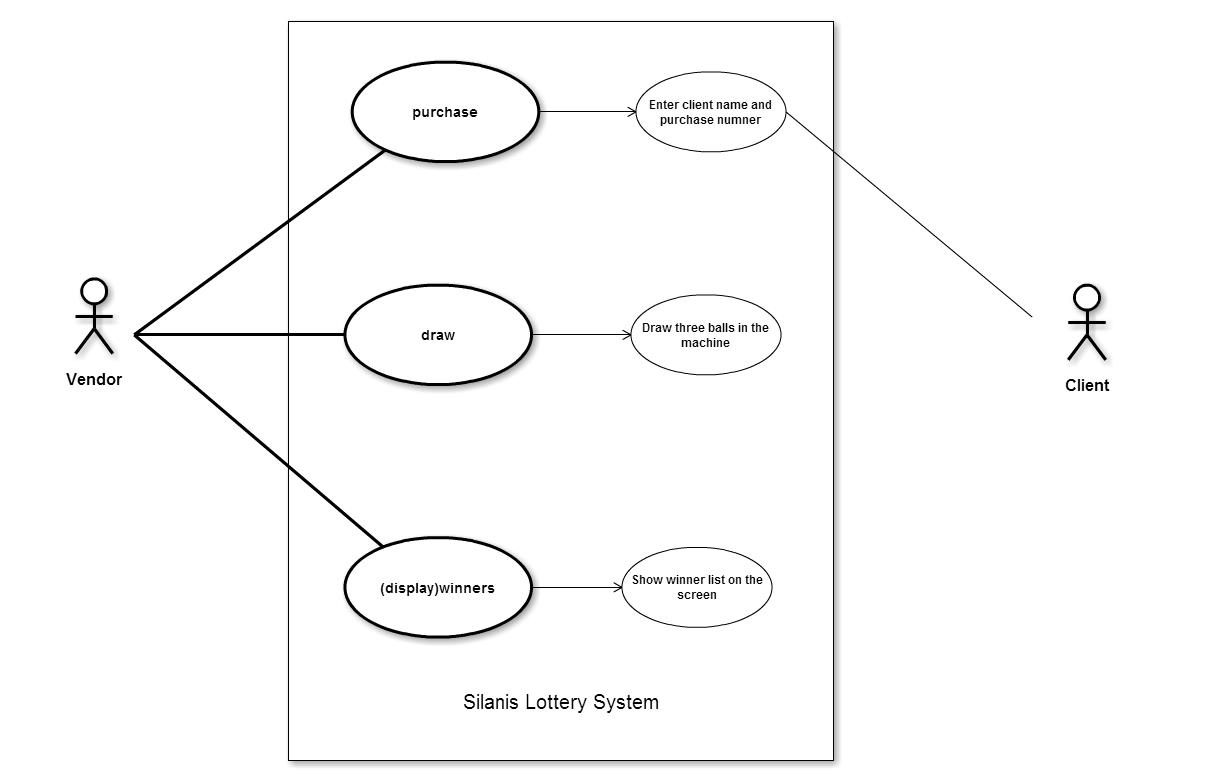
1. When we implement the algorithm, it is necessary to consider the variability and expansibility of the algorithm.
2. We need to avoid order confusion to cause a system error
3. We need to sure a correct display format , also need to consider the case of no one winning
4. When Tommy Inputs commands “purchase”, “draw” and “winners”, we need to avoid other input characters
5. When after each draw, the machine need to reset 50 tickets numbered.
6. Round the nearest dollar value.
7. How to store the data?

# Purchase a lottery overview

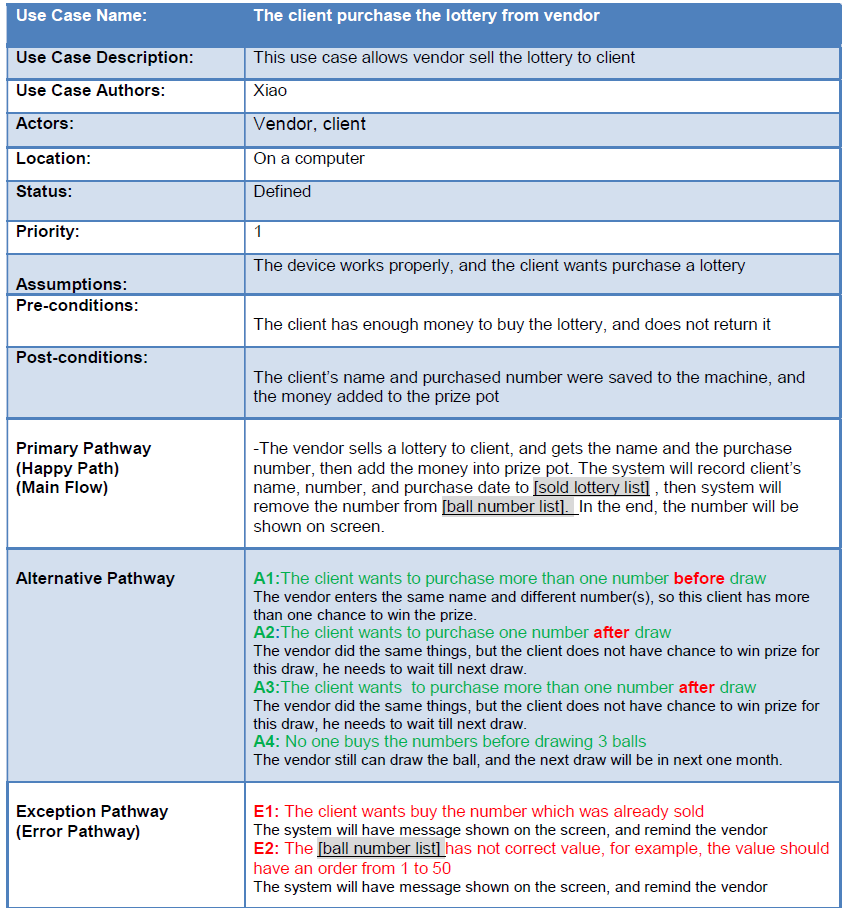


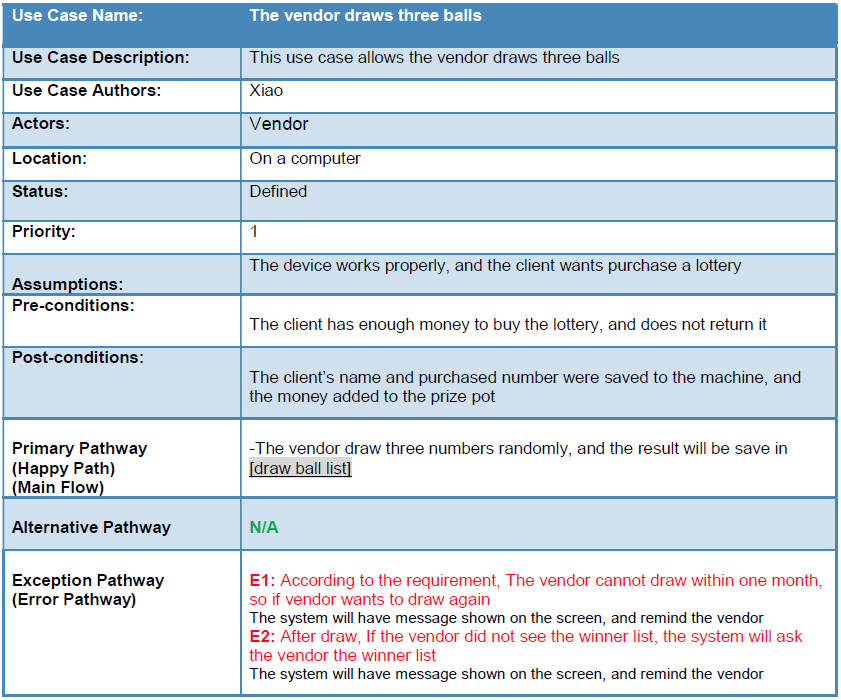
1. The client can purchase several times with different numbers **before** drawing the balls.
2. The client can purchase same number or more than one number **after** drawing the balls, but the client cannot win this time.
3. The vendor draws three numbers randomly, and compares them with purchase record list to make sure which client wins the prize, the prize pot will be reduced this prize.
4. The vendor input “Winners”, then the system will show the winner list on the screen

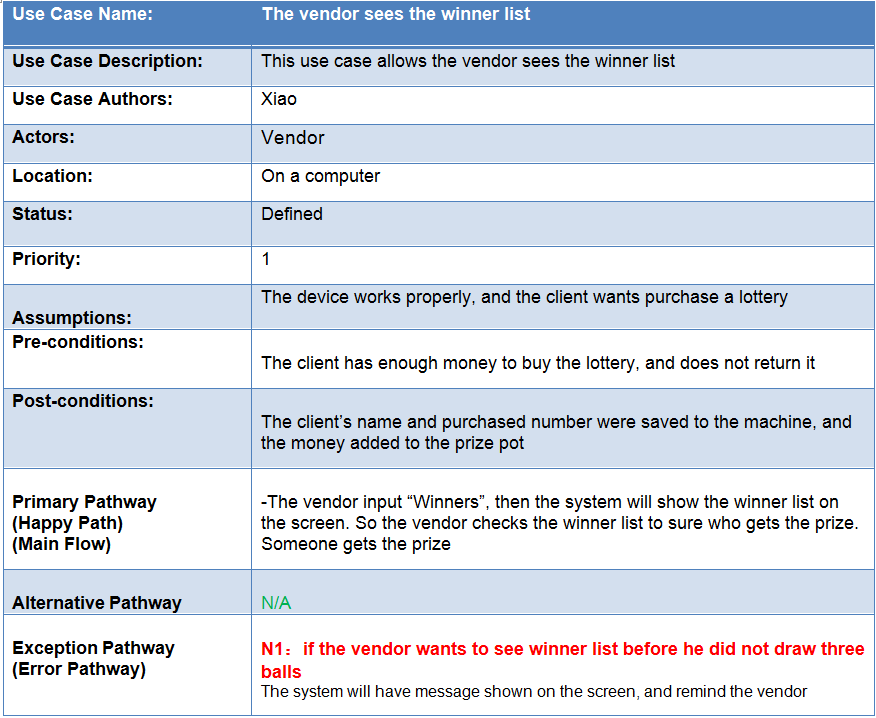
# Use case diagram



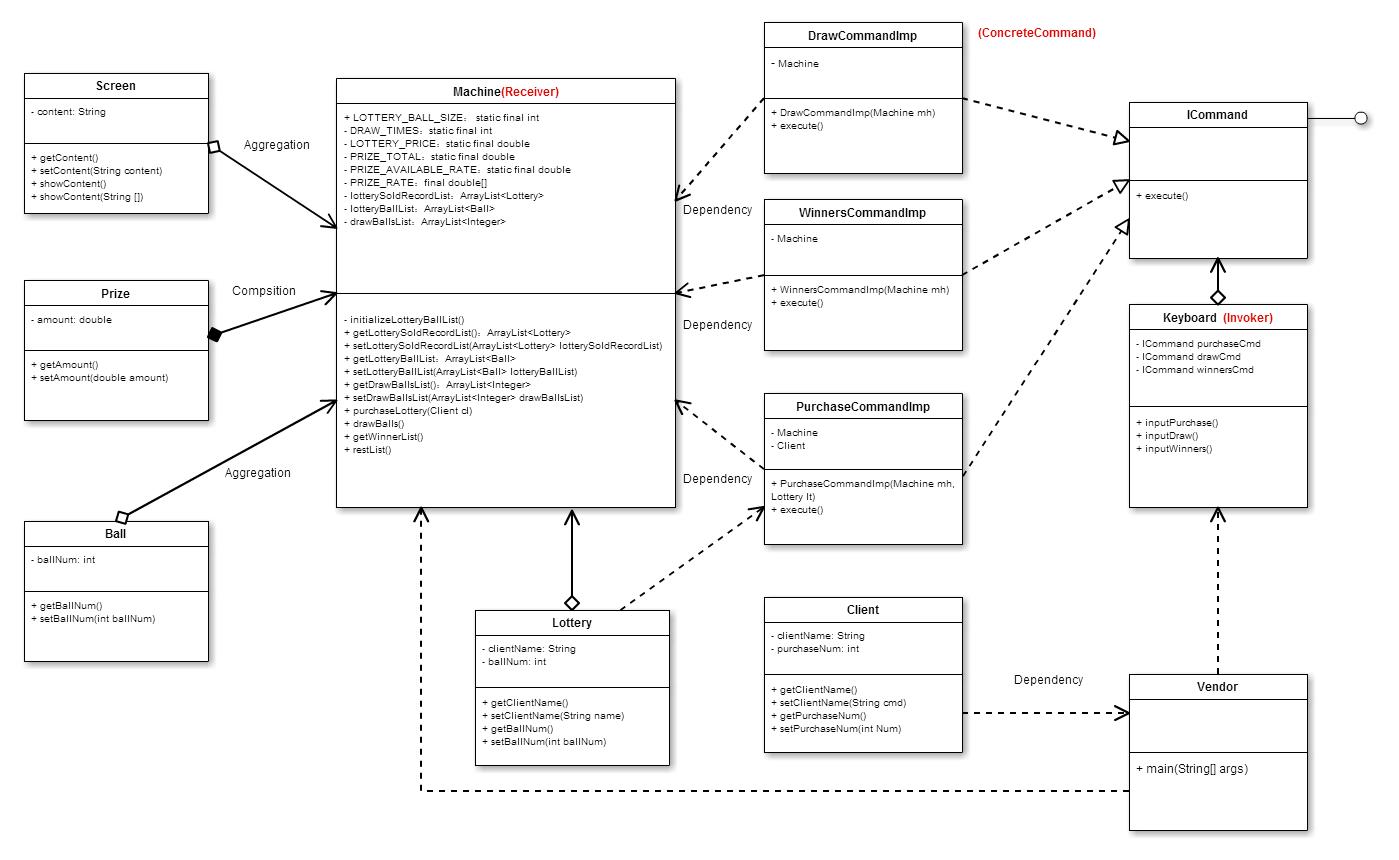
# Use case template







# Class diagram



# Conclusion

I have used “**Command design pattern**” to implement the design, because in my understanding, it is useful when creating a structure, particularly when the creating of a request and executing are not dependent on each other. And this pattern helps in terms of extensibility as we can add a new command without changing the existing code. In the end, it allows me to create a sequence of commands named **macro**. To run the **macro**, create a list of Command instances and call the execute method of all commands.

I have known system design is composed of the needed requirements to meet the specified goals. This focuses on how to make the system more practical, efficient, less costly, flexibility and more secure. Furthermore, using the right process to approach the questions is also very important. Lastly, **Continuous Integration** doesn’t get rid of bugs, but it does make them dramatically easier to find and remove, so I can solve problems quickly in my project.