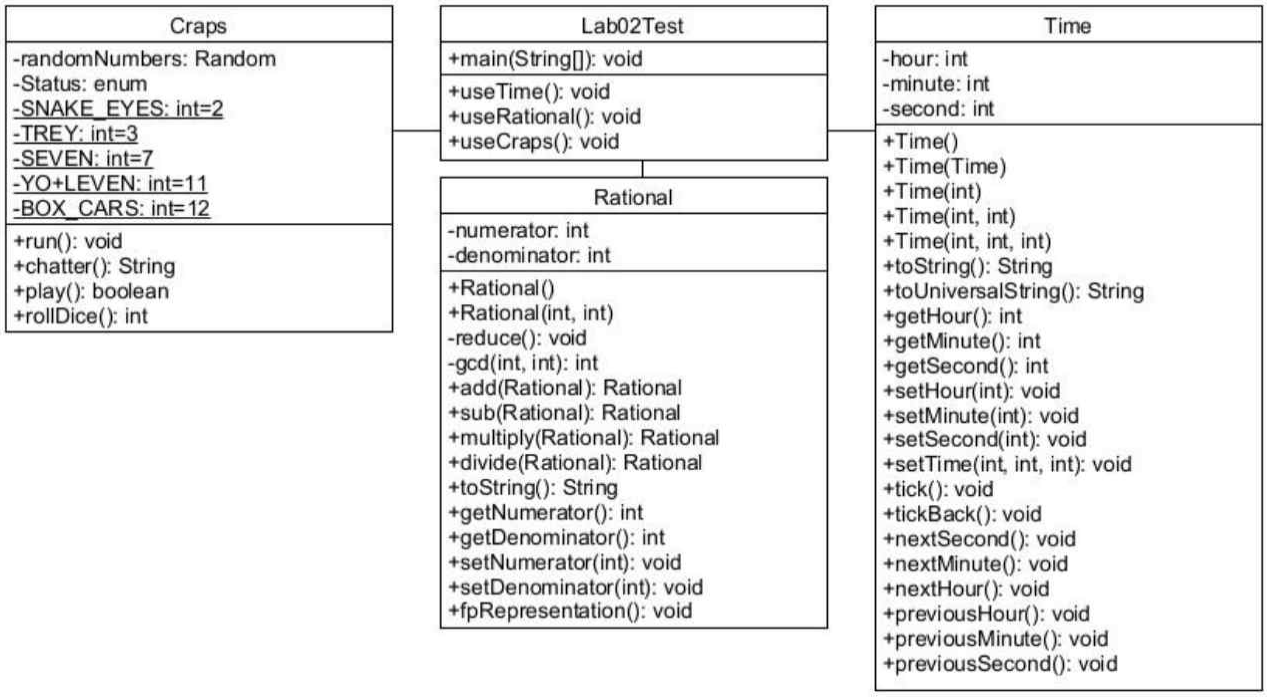
Java Programming (CSE220)

Lab 02

ID: 2020136129 Name: 최수연

Class diagram



Task-1 ***(Rational Numbers)***

Create a class called Rational for performing arithmetic with fractions. Use integer variables to represent the private instance variables of the class—the numerator and the denominator.

1. Provide a constructor that enables an object of this class to be initialized when it's declared. The constructor should store the fraction in reduced form. The fraction 2/4 is equivalent to 1/2 and would be stored in the object as 1 in the numerator and 2 in the denominator.
2. Provide a no-argument constructor with default values in case no initializers are provided.
3. Provide public methods that perform each of the following operations:
    a) Add two Rational numbers: The result of the addition should be stored in reduced form.
    b) Subtract two Rational numbers: The result of the subtraction should be stored in reduced form.
    c) Multiply two Rational numbers: The result of the multiplication should be stored in reduced form.
    d) Divide two Rational numbers: The result of the division should be stored in reduced form.
    e) Return a String representation of a Rational number in the form a/b, where a is the numerator and b is the denominator.
    f) Return a String representation of a Rational number in floating-point format.

Code

Rational Class

```
public class Rational {
	// Generate numerator and denominator variables
	private int numerator;
	private int denominator;

	// Generate Getters and Setters for each variable
	public int getNumerator() {
		return numerator;
	}

	public void setNumerator(int numerator) {
		this.numerator = numerator;
	}

	public int getDenominator() {
		return denominator;
	}

	public void setDenominator(int denominator) {
		this.denominator = denominator;
	}

	// Convert to floating-point format
	public void fpRepresentation() {
		System.out.printf("%.2f\n", (double) numerator / denominator);
	}

	// Convert to a string representation of a Rational number in the form a / b
	@Override
	public String toString() {
		// System.out.printf("Rational number (%d, %d) = ", numerator, denominator);
		if (denominator == 0)
			return String.format("It cannot be divided by zero! So it doesn't exist.");
		else
			return String.format("%d / %d", numerator, denominator);
	}

	// Provide a no-argument constructor with default values in case no initializers
	// are provided
	public Rational() {
		numerator = 0;
		denominator = 1;
	}

	// Put into numerator, denominator variables in order of received arguments
	public Rational(int a, int b) {
		numerator = a;
		denominator = b;
		if (denominator != 0)
			reduce();
```

```
	}

	// Add two Rational numbers
	public Rational add(Rational other) {
		System.out.printf("(%d/%d) + (%d/%d) = ", this.numerator, this.denominator, other.numerator,
other.denominator);
		int newNumerator = this.numerator * other.denominator + other.numerator * this.denominator;
		int newDenominator = this.denominator * other.denominator;
		return new Rational(newNumerator, newDenominator);
	}

	// Subtract two Rational numbers
	public Rational sub(Rational other) {
		System.out.printf("(%d/%d) - (%d/%d) = ", this.numerator, this.denominator, other.numerator,
other.denominator);
		int newNumerator = this.numerator * other.denominator - other.numerator * this.denominator;
		int newDenominator = this.denominator * other.denominator;
		return new Rational(newNumerator, newDenominator);
	}

	// Multiply two Rational numbers
	public Rational multiply(Rational other) {
		System.out.printf("(%d/%d) * (%d/%d) = ", this.numerator, this.denominator, other.numerator,
other.denominator);
		int newNumerator = this.numerator * other.numerator;
		int newDenominator = this.denominator * other.denominator;
		return new Rational(newNumerator, newDenominator);
	}

	// Divide two Rational numbers
	public Rational divide(Rational other) {
		System.out.printf("(%d/%d) / (%d/%d) = ", this.numerator, this.denominator, other.numerator,
other.denominator);
		int newNumerator = this.numerator * other.denominator;
		int newDenominator = this.denominator * other.numerator;
		return new Rational(newNumerator, newDenominator);
	}

	// Convert fractions to reduced form
	private void reduce() {
		int gcd = gcd(Math.abs(numerator), Math.abs(denominator));
		numerator /= gcd;
		denominator /= gcd;
	}

	// Greatest Common Divisor
	private int gcd(int a, int b) {
		if (b != 0) return gcd(b, a % b);
		else return a;
	}
}
```

Results/Output

```java
public static void useRational() {
    Rational r1 = new Rational(3, 6);
    Rational r2 = new Rational(6, 5);
    Rational r3 = new Rational(44, 55);
    Rational r4 = new Rational(12, 6);
    Rational r5 = new Rational();
```

```
[Rational number in the form a / b]
1 / 2
6 / 5
4 / 5
2 / 1
0 / 1

[Rational number in floating-point format]
0.50
1.20
0.80
2.00
0.00

[Add two Rational numbers]
(1/2) + (6/5) = 17 / 10 = 1.70
(6/5) + (4/5) = 2 / 1 = 2.00
(4/5) + (2/1) = 14 / 5 = 2.80
(2/1) + (0/1) = 2 / 1 = 2.00

[Subtract two Rational numbers]
(1/2) - (6/5) = -7 / 10 = -0.70
(6/5) - (4/5) = 2 / 5 = 0.40
(4/5) - (2/1) = -6 / 5 = -1.20
(2/1) - (0/1) = 2 / 1 = 2.00

[Multiply two Rational numbers]
(1/2) * (6/5) = 3 / 5 = 0.60
(6/5) * (4/5) = 24 / 25 = 0.96
(4/5) * (2/1) = 8 / 5 = 1.60
(2/1) * (0/1) = 0 / 1 = 0.00

[Divide two Rational numbers]
(1/2) / (6/5) = 5 / 12 = 0.42
(6/5) / (4/5) = 3 / 2 = 1.50
(4/5) / (2/1) = 2 / 5 = 0.40
(2/1) / (0/1) = It cannot be divided by zero! So it doesn't exist. = Infinity
```

Task-2: Create a Time class and test it in the Lab02Tes class. Time class should have the following data and function members

Data Members:

- hour: between 0 to 23.
- minute: between 0 to 59.
- second: between 0 to 59.

Function members:

- Constructors
- Setters: setHour(int hour), setMinute(int minute), setSecond(int second)
- Getters: getHour(), getMinute(), getSecond()
- setTime(int hour, int minute, int second)
- toString(): returns String for standard-time format (H:MM:SS AM or PM)
- toUniversalString(): String in universal-time format (HH:MM:SS)
- nextSecond(), nextMinute(), nextHour()
- previousSecond(), previousMinute(), previousHour()
- tick(), tickBack()

Code

Time Class

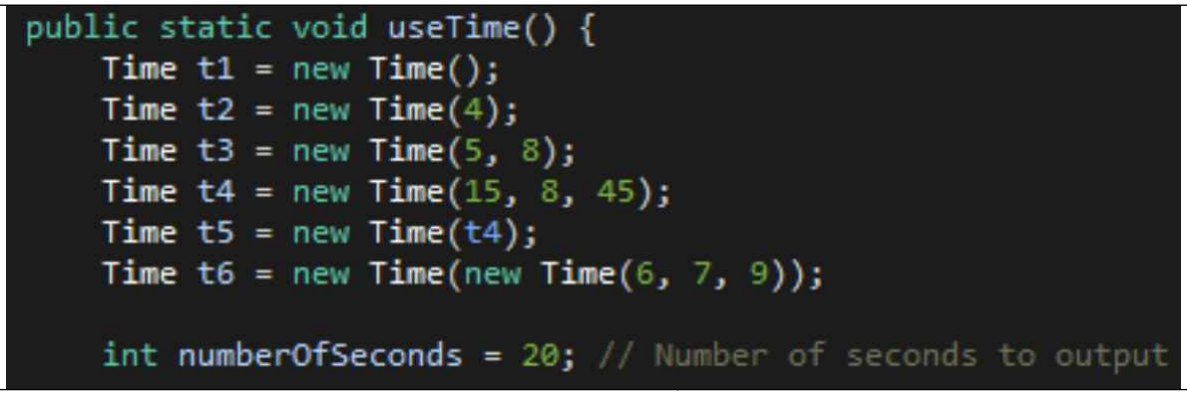
```
public class Time {

	private int hour;
	private int minute;
	private int second;

	// Output as string
	public String toUniversalString() {
		return String.format("%2d : %2d : %2d", getHour(), getMinute(), getSecond());
	}

	// Output as string using AM and PM
	public String toString() {
		return String.format("%2d : %2d : %2d %s", (hour == 0 || hour == 12) ? 12 : hour % 12, minute,
second,
					(hour < 12) ? "AM" : "PM");
	}

	/*
	 * Increase by 1 second
	 * and reset to 0 if it exceeds 60
	 * and call nextMinute()
	 */
	public void nextSecond() {
		second++;
		if (second >= 60) {
			second = 0;
```

```
                        nextMinute();
                }
        }

        /*
         * Increase by 1 minute
         * and reset to 0 if it exceeds 60
         * and call nextHour()
         */
        public void nextMinute() {
                minute++;
                if (minute >= 60) {
                        minute = 0;
                        nextHour();
                }
        }

        /*
         * Increase by 1 hour
         * and reset to 0 if it exceeds 24
         */
        public void nextHour() {
                hour++;
                if (hour >= 24) {
                        hour = 0;
                }
        }

        /*
         * Decrease by 1 second
         * and reset to 59 if it falls below 0
         * and call previousMinute()
         */
        public void previousSecond() {
                second--;
                if (second <= 0) {
                        second = 59;
                        previousMinute();
                }
        }

        /*
         * Decrease by 1 minute
         * and reset to 59 if it falls below 0
         * and call previousHour()
         */
        public void previousMinute() {
                minute--;
                if (minute <= 0) {
                        minute = 59;
                        previousHour();
                }
        }
```

```
/*
 * Decrease by 1 hour
 * and reset to 23 if it falls below 0
 */
public void previousHour() {
	hour--;
	if (hour <= 0) {
		hour = 23;
	}
}

// Increase by 1 second
public void tick() {
	nextSecond();
}

// Decrease by 1 second
public void tickBack() {
	previousSecond();
}

// Initialize constructors by parameter
public Time() {
	// this.hour = 0;
	// this.minute = 0;
	// this.second = 0;
	this(0, 0, 0);
}

public Time(Time t) {
	this.hour = t.hour;
	this.minute = t.minute;
	this.second = t.second;
}

public Time(int h) {
	// this.hour = h;
	// this.minute = 0;
	// this.second = 0;
	this(h, 0, 0);
}

public Time(int h, int m) {
	// this.hour = h;
	// this.minute = m;
	// this.second = 0;
	this(h, m, 0);
}

public Time(int h, int m, int s) {
	// this.hour = h;
	// this.minute = m;
```

```
                // this.second = s;
                setTime(h, m, s);
        }

        // Generate Getters and Setters for each variable
        public void setTime(int hour, int minute, int second) {
                this.hour = hour;
                this.minute = minute;
                this.second = second;
        }

        public int getHour() {
                return hour;
        }

        public void setHour(int hour) {
                this.hour = hour;
        }

        public int getMinute() {
                return minute;
        }

        public void setMinute(int minute) {
                this.minute = minute;
        }

        public int getSecond() {
                return second;
        }

        public void setSecond(int second) {
                this.second = second;
        }

}
```

Results/Output



Set up constructors and Number of seconds to output

```java
for (int i = 0; i < numberOfSeconds; i++) {
    if (i == 0) System.out.println("[Decrease t1 Time]");
    System.out.println(t1);
    t1.tickBack(); // Decrease by 1 second
    try {
        Thread.sleep(1000); // Waiting for 1 second for reality
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

```
[Decrease t1 Time]
12 :  0 :  0 AM
11 : 59 : 59 PM
11 : 59 : 58 PM
11 : 59 : 57 PM
11 : 59 : 56 PM
11 : 59 : 55 PM
11 : 59 : 54 PM
11 : 59 : 53 PM
11 : 59 : 52 PM
11 : 59 : 51 PM
11 : 59 : 50 PM
11 : 59 : 49 PM
11 : 59 : 48 PM
11 : 59 : 47 PM
11 : 59 : 46 PM
11 : 59 : 45 PM
11 : 59 : 44 PM
11 : 59 : 43 PM
11 : 59 : 42 PM
11 : 59 : 41 PM
```

```java
for (int i = 0; i < numberOfSeconds; i++) {
    if (i == 0) System.out.println("[Increase t2 Time]");
    System.out.println(t2);
    t2.tick(); // Increase by 1 second
    try {
        Thread.sleep(1000); // Waiting for 1 second for reality
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

```
[Increase t2 Time]
 4 :  0 :  0 AM
 4 :  0 :  1 AM
 4 :  0 :  2 AM
 4 :  0 :  3 AM
 4 :  0 :  4 AM
 4 :  0 :  5 AM
 4 :  0 :  6 AM
 4 :  0 :  7 AM
 4 :  0 :  8 AM
 4 :  0 :  9 AM
 4 :  0 : 10 AM
 4 :  0 : 11 AM
 4 :  0 : 12 AM
 4 :  0 : 13 AM
 4 :  0 : 14 AM
 4 :  0 : 15 AM
 4 :  0 : 16 AM
 4 :  0 : 17 AM
 4 :  0 : 18 AM
 4 :  0 : 19 AM
```

```java
for (int i = 0; i < numberOfSeconds; i++) {
    if (i == 0) System.out.println("[Decrease t3 Time]");
    System.out.println(t3);
    t3.tickBack(); // Decrease by 1 second
    try {
        Thread.sleep(1000); // Waiting for 1 second for reality
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

```
[Decrease t3 Time]
 5 :  8 :  0 AM
 5 :  7 : 59 AM
 5 :  7 : 58 AM
 5 :  7 : 57 AM
 5 :  7 : 56 AM
 5 :  7 : 55 AM
 5 :  7 : 54 AM
 5 :  7 : 53 AM
 5 :  7 : 52 AM
 5 :  7 : 51 AM
 5 :  7 : 50 AM
 5 :  7 : 49 AM
 5 :  7 : 48 AM
 5 :  7 : 47 AM
 5 :  7 : 46 AM
 5 :  7 : 45 AM
 5 :  7 : 44 AM
 5 :  7 : 43 AM
 5 :  7 : 42 AM
 5 :  7 : 41 AM
```

```java
for (int i = 0; i < numberOfSeconds; i++) {
    if (i == 0) System.out.println("[Increase t4 Time]");
    System.out.println(t4);
    t4.tick(); // Increase by 1 second
    try {
        Thread.sleep(1000); // Waiting for 1 second for reality
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

```
[Increase t4 Time]
 3 :  8 : 45 PM
 3 :  8 : 46 PM
 3 :  8 : 47 PM
 3 :  8 : 48 PM
 3 :  8 : 49 PM
 3 :  8 : 50 PM
 3 :  8 : 51 PM
 3 :  8 : 52 PM
 3 :  8 : 53 PM
 3 :  8 : 54 PM
 3 :  8 : 55 PM
 3 :  8 : 56 PM
 3 :  8 : 57 PM
 3 :  8 : 58 PM
 3 :  8 : 59 PM
 3 :  9 :  0 PM
 3 :  9 :  1 PM
 3 :  9 :  2 PM
 3 :  9 :  3 PM
 3 :  9 :  4 PM
```

```java
for (int i = 0; i < numberOfSeconds; i++) {
    if (i == 0) System.out.println("[Decrease t5 Time]");
    System.out.println(t5);
    t5.tickBack(); // Decrease by 1 second
    try {
        Thread.sleep(1000); // Waiting for 1 second for reality
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

```
[Decrease t5 Time]
 3 :  8 : 45 PM
 3 :  8 : 44 PM
 3 :  8 : 43 PM
 3 :  8 : 42 PM
 3 :  8 : 41 PM
 3 :  8 : 40 PM
 3 :  8 : 39 PM
 3 :  8 : 38 PM
 3 :  8 : 37 PM
 3 :  8 : 36 PM
 3 :  8 : 35 PM
 3 :  8 : 34 PM
 3 :  8 : 33 PM
 3 :  8 : 32 PM
 3 :  8 : 31 PM
 3 :  8 : 30 PM
 3 :  8 : 29 PM
 3 :  8 : 28 PM
 3 :  8 : 27 PM
 3 :  8 : 26 PM
```

```java
for (int i = 0; i < numberOfSeconds; i++) {
    if (i == 0) System.out.println("[Increase t6 Time]");
    System.out.println(t6);
    t6.tick(); // Increase by 1 second
    try {
        Thread.sleep(1000); // Waiting for 1 second for reality
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

```
[Increase t6 Time]
 6 :  7 :  9 AM
 6 :  7 : 10 AM
 6 :  7 : 11 AM
 6 :  7 : 12 AM
 6 :  7 : 13 AM
 6 :  7 : 14 AM
 6 :  7 : 15 AM
 6 :  7 : 16 AM
 6 :  7 : 17 AM
 6 :  7 : 18 AM
 6 :  7 : 19 AM
 6 :  7 : 20 AM
 6 :  7 : 21 AM
 6 :  7 : 22 AM
 6 :  7 : 23 AM
 6 :  7 : 24 AM
 6 :  7 : 25 AM
 6 :  7 : 26 AM
 6 :  7 : 27 AM
 6 :  7 : 28 AM
```

Task-3: Case Study: A **Game of Chance**; Create a class Craps and test it in the Lab02Test class.

Data Members

See the code

Function Members

play()

1) You roll two dice. (call rolldice())
2) Each dice has six faces, which contain one, two, three, four, five, and six spots, respectively.
3) After the dice have come to rest, the sum of the spots on the two upward faces is calculated. If the sum is 7 or 11 on the first throw, you win. If the sum is 2, 3 or 12 on the first throw (called “craps”), you lose (i.e., the “house” wins).
4) If the sum is 4, 5, 6, 8, 9, or 10 on the first throw, that sum becomes your “point.”
5) To win, you must continue rolling the dice until you “make your point” (i.e., roll that same point value). You lose by rolling a 7 before making your point.

run()

Initialize variable **bankBalance** to 1000 dollars. Prompt the player to enter a **wager**. Check that wager is less than or equal to bankBalance, and if it’s not, have the user reenter wager until a valid wager is entered. Then, run one game of craps. If the player wins, increase bankBalance by wager and display the new bankBalance. If the player loses, decrease bankBalance by wager, display the new bankBalance, check whether bankBalance has become zero and, if so, display the message "Sorry. You busted!" As the game progresses, display various messages to create some **“chatter,”** such as "Oh, you're going for broke, huh?" or "Aw c'mon, take a chance!" or "You're up big. Now's the time to cash in your chips!". Implement the “chatter” as a separate method that randomly chooses the string to display

rollDice()

chatter()

Code

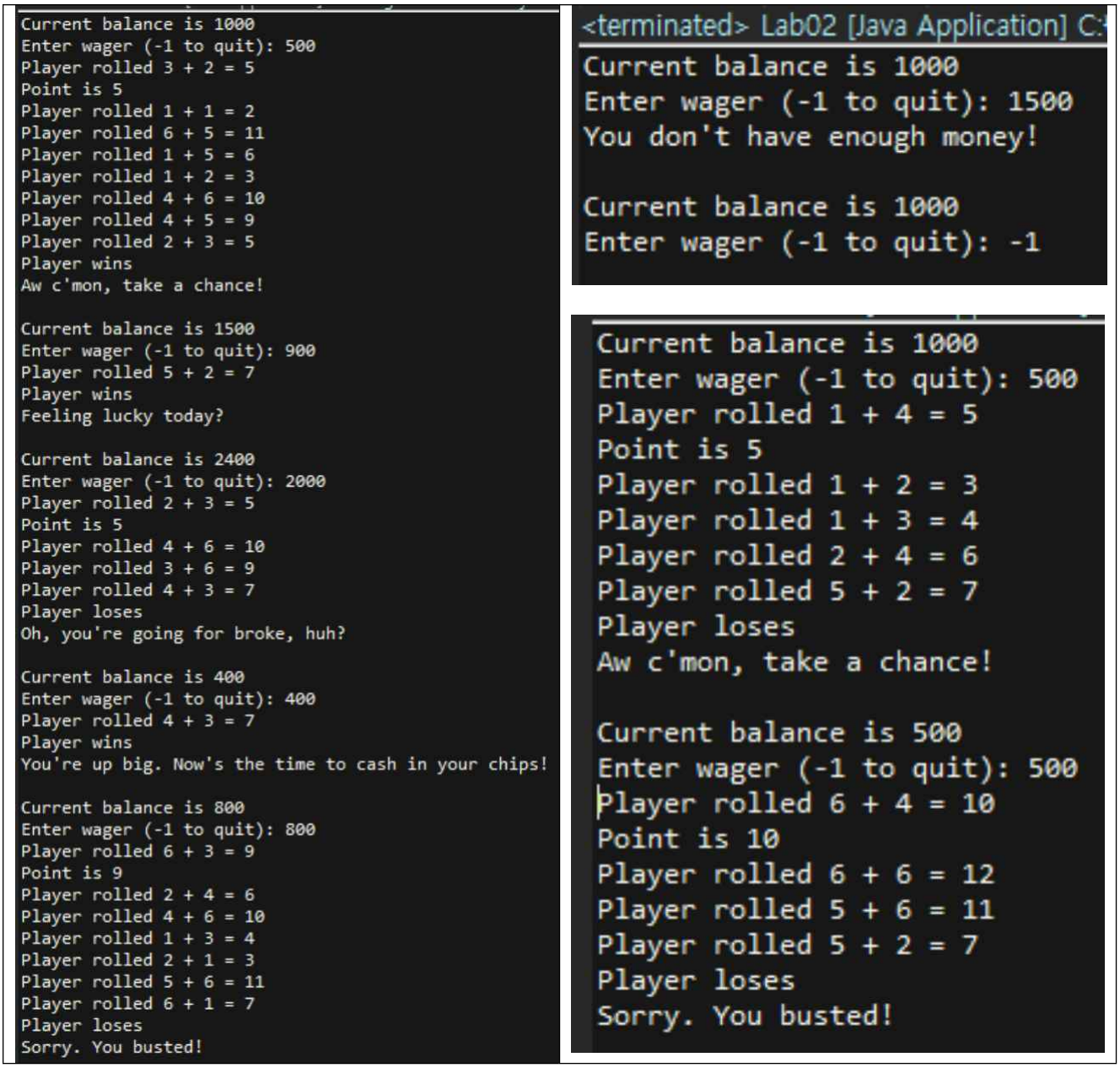
Craps Class

```
import java.util.Random;
import java.util.Scanner;

public class Craps {
	// create secure random number generator for use in method rollDice
	private Random randomNumbers = new Random();

	// enum type with constants that represent the game status
	private enum Status {
		CONTINUE, WON, LOST
	};

	// constants that represent common rolls of the dice
	private static final int SNAKE_EYES = 2;
	private static final int TREY = 3;
	private static final int SEVEN = 7;
	private static final int YO_LEVEN = 11;
	private static final int BOX_CARS = 12;

	public void run() {
		int bankBalance = 1000;
		int wager;
		Scanner input = new Scanner(System.in);

		do {
			// prompt the user for a wager
			System.out.printf("Current balance is %d\n", bankBalance);
			System.out.print("Enter wager (-1 to quit): ");
			wager = input.nextInt();
			if (wager >= 0) {
				if (wager > bankBalance)
					System.out.println("You don't have enough money!");
				else {
					boolean won = play(); // play a game

					// If you win, add a wager
					if(won) bankBalance += wager;
					// If you lose, subtract a wager
					else bankBalance -= wager;

					if (bankBalance <= 0)
						System.out.println("Sorry. You busted!");
					else
						System.out.println(chatter());
				} // end else
					// reset the wager
				wager = 0;
				System.out.println();
			} // end if
				// terminate if the user quits or runs out of money
```

```
            } while ((wager != -1) && (bankBalance > 0));
            input.close();
      } // end method bet

      public String chatter() {
            switch (randomNumbers.nextInt(5)) {
            case 0:
                  return "Aw c'mon, take a chance!";
            case 1:
                  return "Oh, you're going for broke, huh?";
            case 2:
                  return "You're up big. Now's the time to cash in your chips!";
            case 3:
                  return "Feeling lucky today?";
            default:
                  return "Let's see those dice fly!";
            }
      }

      // plays one game of craps
      public boolean play() {
            int myPoint = 0; // point if no win or loss on first roll
            Status gameStatus; // can contain CONTINUE, WON or LOST

            int sumOfDice = rollDice(); // first roll of the dice

            // determine game status and point based on first roll
            switch (sumOfDice) {
            case SEVEN: // win with 7 on first roll
            case YO_LEVEN: // win with 11 on first roll
                  gameStatus = Status.WON;
                  break;
            case SNAKE_EYES: // lose with 2 on first roll
            case TREY: // lose with 3 on first roll
            case BOX_CARS: // lose with 12 on first roll
                  gameStatus = Status.LOST;
                  break;
            default: // did not win or lose, so remember point
                  gameStatus = Status.CONTINUE; // game is not over
                  myPoint = sumOfDice; // remember the point
                  System.out.printf("Point is %d%n", myPoint);
                  break;
            }

            // while game is not complete
            while (gameStatus == Status.CONTINUE) // not WON or LOST
            {
                  sumOfDice = rollDice(); // roll dice again

                  // determine game status
                  if (sumOfDice == myPoint) // win by making point
                        gameStatus = Status.WON;
                  else if (sumOfDice == SEVEN) // lose by rolling 7 before point
```

```
                                        gameStatus = Status.LOST;
                }

                // display won or lost message
                if (gameStatus == Status.WON)
                        System.out.println("Player wins");
                else
                        System.out.println("Player loses");
                return gameStatus == Status.WON;
        }

        // roll dice, calculate sum and display results
        public int rollDice() {
                // pick random die values
                int die1 = 1 + randomNumbers.nextInt(6); // first die roll
                int die2 = 1 + randomNumbers.nextInt(6); // second die roll

                int sum = die1 + die2; // sum of die values

                // display results of this roll
                System.out.printf("Player rolled %d + %d = %d%n", die1, die2, sum);

                return sum;
        }
}
```

Results/Output



```
Current balance is 1000
Enter wager (-1 to quit): 500
Player rolled 3 + 2 = 5
Point is 5
Player rolled 1 + 1 = 2
Player rolled 6 + 5 = 11
Player rolled 1 + 5 = 6
Player rolled 1 + 2 = 3
Player rolled 4 + 6 = 10
Player rolled 4 + 5 = 9
Player rolled 2 + 3 = 5
Player wins
Aw c'mon, take a chance!

Current balance is 1500
Enter wager (-1 to quit): 900
Player rolled 5 + 2 = 7
Player wins
Feeling lucky today?

Current balance is 2400
Enter wager (-1 to quit): 2000
Player rolled 2 + 3 = 5
Point is 5
Player rolled 4 + 6 = 10
Player rolled 3 + 6 = 9
Player rolled 4 + 3 = 7
Player loses
Oh, you're going for broke, huh?

Current balance is 400
Enter wager (-1 to quit): 400
Player rolled 4 + 3 = 7
Player wins
You're up big. Now's the time to cash in your chips!

Current balance is 800
Enter wager (-1 to quit): 800
Player rolled 6 + 3 = 9
Point is 9
Player rolled 2 + 4 = 6
Player rolled 4 + 6 = 10
Player rolled 1 + 3 = 4
Player rolled 2 + 1 = 3
Player rolled 5 + 6 = 11
Player rolled 6 + 1 = 7
Player loses
Sorry. You busted!
```

```
<terminated> Lab02 [Java Application] C:
Current balance is 1000
Enter wager (-1 to quit): 1500
You don't have enough money!

Current balance is 1000
Enter wager (-1 to quit): -1
```

```
Current balance is 1000
Enter wager (-1 to quit): 500
Player rolled 1 + 4 = 5
Point is 5
Player rolled 1 + 2 = 3
Player rolled 1 + 3 = 4
Player rolled 2 + 4 = 6
Player rolled 5 + 2 = 7
Player loses
Aw c'mon, take a chance!

Current balance is 500
Enter wager (-1 to quit): 500
Player rolled 6 + 4 = 10
Point is 10
Player rolled 6 + 6 = 12
Player rolled 5 + 6 = 11
Player rolled 5 + 2 = 7
Player loses
Sorry. You busted!
```

Code

Lab02Test Class

```
public class Lab02Test {
	public static void main(String[] args) {
		// Select the desired function
		useRational();
		useTime();
		useCraps();
	}

	public static void useRational() {
		Rational r1 = new Rational(3, 6);
		Rational r2 = new Rational(6, 5);
		Rational r3 = new Rational(44, 55);
		Rational r4 = new Rational(12, 6);
		Rational r5 = new Rational();

		System.out.println("[Rational number in the form a / b]");
		System.out.println(r1);
		System.out.println(r2);
		System.out.println(r3);
		System.out.println(r4);
		System.out.println(r5);
		System.out.println();

		System.out.println("[Rational number in floating-point format]");
		r1.fpRepresentation();
		r2.fpRepresentation();
		r3.fpRepresentation();
		r4.fpRepresentation();
		r5.fpRepresentation();
		System.out.println();

		System.out.println("[Add two Rational numbers]");
		Rational add1 = r1.add(r2);
		System.out.print(add1 + " = ");
		add1.fpRepresentation();
		Rational add2 = r2.add(r3);
		System.out.print(add2 + " = ");
		add2.fpRepresentation();
		Rational add3 = r3.add(r4);
		System.out.print(add3 + " = ");
		add3.fpRepresentation();
		Rational add4 = r4.add(r5);
		System.out.print(add4 + " = ");
		add4.fpRepresentation();
		System.out.println();

		System.out.println("[Subtract two Rational numbers]");
		Rational sub1 = r1.sub(r2);
		System.out.print(sub1 + " = ");
```

```
		sub1.fpRepresentation();
		Rational sub2 = r2.sub(r3);
		System.out.print(sub2 + " = ");
		sub2.fpRepresentation();
		Rational sub3 = r3.sub(r4);
		System.out.print(sub3 + " = ");
		sub3.fpRepresentation();
		Rational sub4 = r4.sub(r5);
		System.out.print(sub4 + " = ");
		sub4.fpRepresentation();
		System.out.println();

		System.out.println("[Multiply two Rational numbers]");
		Rational multiply1 = r1.multiply(r2);
		System.out.print(multiply1 + " = ");
		multiply1.fpRepresentation();
		Rational multiply2 = r2.multiply(r3);
		System.out.print(multiply2 + " = ");
		multiply2.fpRepresentation();
		Rational multiply3 = r3.multiply(r4);
		System.out.print(multiply3 + " = ");
		multiply3.fpRepresentation();
		Rational multiply4 = r4.multiply(r5);
		System.out.print(multiply4 + " = ");
		multiply4.fpRepresentation();
		System.out.println();

		System.out.println("[Divide two Rational numbers]");
		Rational divide1 = r1.divide(r2);
		System.out.print(divide1 + " = ");
		divide1.fpRepresentation();
		Rational divide2 = r2.divide(r3);
		System.out.print(divide2 + " = ");
		divide2.fpRepresentation();
		Rational divide3 = r3.divide(r4);
		System.out.print(divide3 + " = ");
		divide3.fpRepresentation();
		Rational divide4 = r4.divide(r5);
		System.out.print(divide4 + " = ");
		divide4.fpRepresentation();
		System.out.println();
	}

	public static void useTime() {
		Time t1 = new Time();
		Time t2 = new Time(4);
		Time t3 = new Time(5, 8);
		Time t4 = new Time(15, 8, 45);
		Time t5 = new Time(t4);
		Time t6 = new Time(new Time(6, 7, 9));

		int numberOfSeconds = 20; // Number of seconds to output
```

```
		for (int i = 0; i < numberOfSeconds; i++) {
			if (i == 0) System.out.println("[Decrease t1 Time]");
			System.out.println(t1);
			t1.tickBack(); // Decrease by 1 second
			try {
				Thread.sleep(1000); // Waiting for 1 second for reality
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
		for (int i = 0; i < numberOfSeconds; i++) {
			if (i == 0) System.out.println("[Increase t2 Time]");
			System.out.println(t2);
			t2.tick(); // Increase by 1 second
			try {
				Thread.sleep(1000); // Waiting for 1 second for reality
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
		for (int i = 0; i < numberOfSeconds; i++) {
			if (i == 0) System.out.println("[Decrease t3 Time]");
			System.out.println(t3);
			t3.tickBack(); // Decrease by 1 second
			try {
				Thread.sleep(1000); // Waiting for 1 second for reality
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
		for (int i = 0; i < numberOfSeconds; i++) {
			if (i == 0) System.out.println("[Increase t4 Time]");
			System.out.println(t4);
			t4.tick(); // Increase by 1 second
			try {
				Thread.sleep(1000); // Waiting for 1 second for reality
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
		for (int i = 0; i < numberOfSeconds; i++) {
			if (i == 0) System.out.println("[Decrease t5 Time]");
			System.out.println(t5);
			t5.tickBack(); // Decrease by 1 second
			try {
				Thread.sleep(1000); // Waiting for 1 second for reality
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
		for (int i = 0; i < numberOfSeconds; i++) {
			if (i == 0) System.out.println("[Increase t6 Time]");
			System.out.println(t6);
```

```
                        t6.tick(); // Increase by 1 second
                        try {
                                Thread.sleep(1000); // Waiting for 1 second for reality
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }

        public static void useCraps() {
                Craps c1 = new Craps();
                c1.run(); // with a bet
                //c1.play(); // without a bet
        }
}
```

Conclusion

문제의 조건이 많았지만 하나씩 차근차근 하다보니 생각보다 괜찮았다. 이번 과제를 통해 여러 매개변수를 받는 생성자를 선언하는 방법에 대해서 알 수 있어서 좋았다. 그리고 마지막 과제는 게임을 만드는 과제여서 좀 더 즐겁게 과제를 할 수 있어서 좋았다.

There were many conditions in question, but it was better than I thought because I did it one by one. It was good to know how to declare a constructor that receives multiple parameters through this task. And the last task was to make a game, so it was good to be able to do the task more pleasantly.