CMSC 132, Object-Oriented Programming II **Summer 2018**

Lecture 1:

Lecturer: Anwar Mamat

1.1 Course Introduction

Check the course website for the syllabus and course introduction slides.

1.2 Java Review

We start by introducing Object Oriented Programming (OOP) concepts, such as class, inheritance, and encapsulation.

1.2.1 Code Examples: Fraction Class

In this example, we implement a Fraction class, which can represent fractions and support arithmetic operations on the fractions. A common fraction consists of an integer numerator, and non-zero denominator. For example: $\frac{2}{10}$ or $\frac{13}{5}$. The Fraction class has two private members numerator and denominator.

Listing 1: Fraction Class

```
1  /**
2   * Fraction class implements non-negative fractions
3   * @author anwar
4   */
5  public class Fraction {
6      protected int numerator;
7      protected int denominator;
8      /** Constructs a Fraction n/d.
9       *  @param n is the numerator, assumed non-negative.
10      *  @param d is the denominator, assumed positive.
11      */
12     Fraction(int n, int d) {
13         int g = gcd(d,n);
14         /** reduce the fraction */
15         numerator = n/g;
16         denominator = d/g;
17     }
18
19     /** Constructs a Fraction n/1.
20     *  @param n is the numerator, assumed non-negative.
21     */
22     public Fraction(int n) {
23         this(n,1);
24     }
25
26     /** Constructs a Fraction 0/1.
27     */
28     public Fraction() {
```

```
29          numerator = 0;
30          denominator = 1;
31      }
32
33      public String toString()   {
34          return (numerator + "/" + denominator);
35      }
36
37      /** Calculates and returns the double floating point value of a fraction.
38      *  @return a double floating point value for this Fraction.
39      */
40      public double evaluate(){
41          double n = numerator;   // convert to double
42          double d = denominator;
43          return (n / d);
44      }
45
46      /** Add f2 to this fraction and return the result.
47      * @param f2 is the fraction to be added.
48      * @return the result of adding f2 to this Fraction.
49      */
50      public Fraction add (Fraction f2) {
51          Fraction r = new Fraction((numerator * f2.denominator) +
52                                    (f2.numerator * denominator),
53                                    (denominator * f2.denominator));
54          return r;
55      }
56
57      /** subtract f2 from this fraction and return the result.
58      * @param f2 is the fraction to be added.
59      * @return the result of adding f2 to this Fraction.
60      */
61      public Fraction sub (Fraction f2) {
62          Fraction r = new Fraction((numerator * f2.denominator) -
63                                    (f2.numerator * denominator),
64                                    (denominator * f2.denominator));
65          return r;
66      }
67
68      /** multiple f2 to this fraction and return the result.
69      * @param f2 is the fraction to be added.
70      * @return the result of adding f2 to this Fraction.
71      */
72      public Fraction mul (Fraction f2) {
73          return (
74                  new Fraction(numerator * f2.numerator,
75                  denominator * f2.denominator)
76                  );
77      }
78
79      /** divide f2 to this fraction and return the result.
80      * @param f2 is the fraction to be added.
81      * @return the result of adding f2 to this Fraction.
82      */
83      public Fraction div (Fraction f2) {
84          return (
85                  new Fraction(numerator * f2.denominator,
86                  denominator * f2.numerator)
87                  );
88      }
89
90      /** Computes the greatest common divisor (gcd) of the two inputs.
91      * @param a is assumed positive
92      * @param b is assumed non-negative
93      * @return the gcd of a and b
94      */
95      static private int gcd (int a, int b) {
96          if(b == 0) return a;
```

```
97          return gcd(b,a%b);
98      }
99
100     public static void main(String[] argv) {
101     /* Test all three contructors and toString. */
102     Fraction f0 = new Fraction();
103     Fraction f1 = new Fraction(3);
104     Fraction f2 = new Fraction(12, 20);
105
106     System.out.println("\nTesting␣constructors␣(and␣toString):");
107     System.out.println("The␣fraction␣f0␣is␣" + f0.toString());
108     System.out.println("The␣fraction␣f1␣is␣" + f1); // toString is implicit
109     System.out.println("The␣fraction␣f2␣is␣" + f2);
110
111     /* Test methods on Fraction: add and evaluate. */
112     System.out.println("\nTesting␣add␣and␣evaluate:");
113     System.out.println("The␣floating␣point␣value␣of␣" + f1 + "␣is␣" +
114                        f1.evaluate());
115     System.out.println("The␣floating␣point␣value␣of␣" + f2 + "␣is␣" +
116                        f2.evaluate());
117
118     Fraction sumOfTwo = f1.add(f2);
119     Fraction sumOfThree = f0.add(f1.add(f2));
120
121     System.out.println("The␣sum␣of␣" + f1 + "␣and␣" + f2 + "␣is␣" + sumOfTwo);
122     System.out.println("The␣sum␣of␣" + f0 + ",␣" + f1 + "␣and␣" + f2 + "␣is␣"
123                        + sumOfThree);
124     /**
125      * test sub,div,mul here
126     */
127     /* Test gcd function (static method). */
128     System.out.println("\nTesting␣gcd:");
129     System.out.println("The␣gcd␣of␣2␣and␣10␣is:␣" + gcd(2, 10));
130     System.out.println("The␣gcd␣of␣15␣and␣5␣is:␣" + gcd(15, 5));
131     System.out.println("The␣gcd␣of␣24␣and␣18␣is:␣" + gcd(24, 18));
132     System.out.println("The␣gcd␣of␣10␣and␣10␣is:␣" + gcd(10, 10));
133     System.out.println("The␣gcd␣of␣21␣and␣400␣is:␣" + gcd(21, 400));
134   }
135 }
```

MixedFraction inherits Fraction, so that it inherits addition, subtraction etc. Only difference is that mixed fraction is a whole number and a fraction combined. For example: $1\frac{3}{5} = \frac{13}{5}$

Listing 2: MixedFraction Class

```
1  /**
2   * This class implements mixed fraction
3   * @author anwar mamat
4   */
5  public class MixedFraction extends Fraction{
6      /** Constructs a Fraction m n/d.
7      *  @param m is the integer part.
8      *  @param n is the numerator, assumed non-negative.
9      *  @param d is the denominator, assumed positive.
10     */
11     public MixedFraction(int m, int n, int d){
12         super(m*d+n,d);//convert mixed fraction into proper fraction.
13     }
14
15     /** Constructs a Fraction m n/d.
16     *  @param f is a fraction
17     */
18     public MixedFraction(Fraction f) {
19         super(f.numerator, f.denominator);
20     }
21
22     public String toString() {
```

```
23          int m = numerator  /  denominator;
24          int n = numerator  %  denominator;
25          return (m + "␣" + n + "/" + denominator);
26      }
27  }
```

Main is the fraction test driver class. The "main" method of this class creates Fraction objects and prints the result in the console.

Listing 3: Fraction Test Driver

```
1   public class Main{
2       public static void main(String[] args) {
3           Fraction f1 = new Fraction(2,10);
4           Fraction f2 = new MixedFraction(1,2,10);
5           System.out.println("f1=" + f1);
6           System.out.println("f2␣=␣" + f2);
7           MixedFraction f3 = new MixedFraction(f2.add(f1));
8           System.out.println(f1 + "+" +  f2 +  "=" + f3);
9           Fraction f4 = f2.sub(f1);
10          System.out.println(f2 + "-" +  f1 +  "=" + f4);
11          Fraction f5 = f1.mul(f2);
12          System.out.println(f1 + "*" +  f2 +  "=" + f5);
13          Fraction f6 = f1.div(f2);
14          System.out.println(f1 + "␣/␣" +  f2 +  "=" + f6);
15      }
16  }
```

We also create a JUnit test class for fraction.

Listing 4: Fraction JUnit Test

```
1
2   import static org.junit.Assert.*;
3   import org.junit.Test;
4   import org.junit.After;
5   import org.junit.AfterClass;
6   import org.junit.Before;
7   import org.junit.BeforeClass;
8   import org.junit.Test;
9   import static org.junit.Assert.*;
10
11  /**
12   *
13   * @author anwar
14   */
15  public class FractionTest {
16
17      public FractionTest() {
18      }
19
20      @BeforeClass
21      public static void setUpClass() {
22          System.out.println("*␣UtilsJUnit4Test:␣@BeforeClass␣method");
23      }
24
25      @AfterClass
26      public static void tearDownClass() {
27          System.out.println("*␣UtilsJUnit4Test:␣@tearDownClass␣method");
28      }
29
30      @Before
31      public void setUpAgain() {
32          System.out.println("*␣UtilsJUnit4Test:␣@setUp␣method");
33      }
34
```

```
35      @After
36      public void tearDown() {
37          System.out.println("* UtilsJUnit4Test: @tearDown method");
38      }
39      /**
40       * Test of toString method, of class Fraction.
41       */
42      @Test
43      public void testToString() {
44          System.out.println("toString");
45          Fraction instance = new Fraction(2,10);
46          String expResult = "1/5";
47          String result = instance.toString();
48          assertEquals(expResult, result);
49      }
50      /**
51       * Test of evaluate method, of class Fraction.
52       */
53      @Test
54      public void testEvaluate() {
55          System.out.println("evaluate");
56          Fraction instance = new Fraction(5,10);
57          double expResult = 0.5;
58          double result = instance.evaluate();
59          assertEquals(expResult, result, 0.0);
60      }
61
62      /**
63       * Test of add method, of class Fraction.
64       */
65      @Test
66      public void testAdd() {
67          System.out.println("add");
68          Fraction f2 = new Fraction(2,7);
69          Fraction instance = new Fraction(1,5);
70          Fraction expResult = new Fraction(17,35);
71          Fraction result = instance.add(f2);
72          assertEquals(expResult, result);
73      }
74
75      /**
76       * Test of sub method, of class Fraction.
77       */
78      @Test
79      public void testSub() {
80          System.out.println("sub");
81          Fraction f2 = new Fraction(1,5);
82          Fraction instance = new Fraction(4,10);
83          Fraction expResult = new Fraction(1,5);;
84          Fraction result = instance.sub(f2);
85          assertEquals(expResult, result);
86      }
87
88      /**
89       * Test of mul method, of class Fraction.
90       */
91      @Test
92      public void testMul() {
93          System.out.println("mul");
94          Fraction f2 = new Fraction(3,5);
95          Fraction instance = new Fraction(2,3);
96          Fraction expResult = new Fraction(6,15);
97          Fraction result = instance.mul(f2);
98          assertEquals(expResult, result);
99      }
100
101     /**
102      * Test of div method, of class Fraction.
```

```
103       */
104     @Test
105     public void testDiv() {
106         System.out.println("div");
107         Fraction f2 = new Fraction(2,5);
108         Fraction instance = new Fraction(3,7);
109         Fraction expResult = new Fraction(15,14);
110         Fraction result = instance.div(f2);
111         assertEquals(expResult, result);
112     }
113
114 }
```