CIS 2168 Data Structures **Fall 2014**

Lecture 1: Sep. 26,28

Lecturer: Anwar Mamat

1.1 Syllabus Overview

- There will be 10 programming assignments.
- midterm exam
- final exam

1.2 Java and NetBeans Setup

Install the latest version of Java and NetBeans. We use NetBeans in class because all lab computers have NetBeans installed. You may use an IDE of your choice. If you do so, when you submit the assignment, give instructions to compile your code from command line.

1.3 Java Review

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

1.3.1 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

```java
/**
 * Fraction class implements non-negative fractions
 * @author anwar
 */
public class Fraction {
    protected int numerator;
    protected int denominator;
    /** Constructs a Fraction n/d.
    *  @param n is the numerator, assumed non-negative.
    *  @param d is the denominator, assumed positive.
    */
    Fraction(int n, int d) {
        int g = gcd(d,n);
        /** reduce the fraction */
        numerator = n/g;
        denominator = d/g;
    }

    /** Constructs a Fraction n/1.
    *  @param n is the numerator, assumed non-negative.
    */
    public Fraction(int n) {
        this(n,1);
    }

    /** Constructs a Fraction 0/1.
    */
    public Fraction() {
        numerator = 0;
        denominator = 1;
    }

    public String toString()   {
        return (numerator + "/" + denominator);
    }

    /** Calculates and returns the double floating point value of a fraction.
    *  @return a double floating point value for this Fraction.
    */
    public double evaluate(){
        double n = numerator;   // convert to double
        double d = denominator;
        return (n / d);
    }

    /** Add f2 to this fraction and return the result.
    * @param f2 is the fraction to be added.
    * @return the result of adding f2 to this Fraction.
    */
    public Fraction add (Fraction f2) {
        Fraction r = new Fraction((numerator * f2.denominator) +
                              (f2.numerator * denominator),
                              (denominator * f2.denominator));
        return r;
    }

    /** subtract f2 from this fraction and return the result.
    * @param f2 is the fraction to be added.
    * @return the result of adding f2 to this Fraction.
    */
    public Fraction sub (Fraction f2) {
        Fraction r = new Fraction((numerator * f2.denominator) -
                                  (f2.numerator * denominator),
                                  (denominator * f2.denominator));
        return r;
    }

    /** multiple f2 to this fraction and return the result.
    * @param f2 is the fraction to be added.
    * @return the result of adding f2 to this Fraction.
    */
    public Fraction mul (Fraction f2) {
        return (
                new Fraction(numerator * f2.numerator,
                denominator * f2.denominator)
                );
    }

    /** divide f2 to this fraction and return the result.
    * @param f2 is the fraction to be added.
    * @return the result of adding f2 to this Fraction.
    */
    public Fraction div (Fraction f2) {
        return (
                new Fraction(numerator * f2.denominator,
                denominator * f2.numerator)
                );
    }

    /** Computes the greatest common divisor (gcd) of the two inputs.
    * @param a is assumed positive
    * @param b is assumed non-negative
    * @return the gcd of a and b
    */
    static private int gcd (int a, int b) {
        int t;
        // a must be greater than or equal to b
        if( a < b){
            t = a;
            a = b;
            b = t;
        }
        if(b == 0){
            return a;
        }else{
            return gcd(b,a%b);
        }
  }

    public static void main(String[] argv) {
    /* Test all three contructors and toString. */
    Fraction f0 = new Fraction();
    Fraction f1 = new Fraction(3);
    Fraction f2 = new Fraction(12, 20);

    System.out.println("\nTesting constructors (and toString):");
    System.out.println("The fraction f0 is " + f0.toString());
    System.out.println("The fraction f1 is " + f1); // toString is implicit
    System.out.println("The fraction f2 is " + f2);

    /* Test methods on Fraction: add and evaluate. */
    System.out.println("\nTesting add and evaluate:");
    System.out.println("The floating point value of " + f1 + " is " +
                       f1.evaluate());
    System.out.println("The floating point value of " + f2 + " is " +
                       f2.evaluate());

    Fraction sumOfTwo = f1.add(f2);
    Fraction sumOfThree = f0.add(f1.add(f2));

    System.out.println("The sum of " + f1 + " and " + f2 + " is " + sumOfTwo);
    System.out.println("The sum of " + f0 + ", " + f1 + " and " + f2 + " is "
                       + sumOfThree);
    /**
     * test sub,div,mul here
     */
    /* Test gcd function (static method). */
    System.out.println("\nTesting gcd:");
    System.out.println("The gcd of 2 and 10 is: " + gcd(2, 10));
    System.out.println("The gcd of 15 and 5 is: " + gcd(15, 5));
    System.out.println("The gcd of 24 and 18 is: " + gcd(24, 18));
    System.out.println("The gcd of 10 and 10 is: " + gcd(10, 10));
    System.out.println("The gcd of 21 and 400 is: " + gcd(21, 400));
  }
}
```

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

```
/**
 * This class implements mixed fraction
 * @author anwar mamat
 */
public class MixedFraction extends Fraction{
    /** Constructs a Fraction m n/d.
    *  @param m is the integer part.
    *  @param n is the numerator, assumed non-negative.
    *  @param d is the denominator, assumed positive.
    */
    public MixedFraction(int m, int n, int d){
        super(m*d+n,d);//convert mixed fraction into proper fraction.
    }
```

```java

    /** Constructs a Fraction m n/d.
    *  @param f is a fraction
    */
    public MixedFraction(Fraction f) {
        super(f.numerator, f.denominator);
    }

    public String toString() {
        int m = numerator / denominator;
        int n = numerator % denominator;
        return (m + " " + n + "/" + denominator);
    }
}
```

Listing 3: Fraction Test

```java
public class FractionTest {
    public static void main(String[] args) {
        Fraction f1 = new Fraction(2,10);
        Fraction f2 = new MixedFraction(1,2,10);
        System.out.println("f1=" + f1);
        System.out.println("f2 = " + f2);
        MixedFraction f3 = new MixedFraction(f2.add(f1));
        System.out.println(f1 + "+" + f2 + "=" + f3);
        Fraction f4 = f2.sub(f1);
        System.out.println(f2 + "-" + f1 + "=" + f4);
        Fraction f5 = f1.mul(f2);
        System.out.println(f1 + "*" + f2 + "=" + f5);
        Fraction f6 = f1.div(f2);
        System.out.println(f1 + " / " + f2 + "=" + f6);
    }
}
```

1.3.2 Code Examples: Shape Class

In this section, we will introduce abstract classes, abstract methods, and inheritance. We start with an abstract class Shape. Abstract classes cannot be instantiated. They can only be inherited by child classes.

Listing 4: Shape Class

```java
abstract class Shape {
   private int x;
   private int y;
   // constructor
   Shape(int newx, int newy) {
      moveTo(newx, newy);
   }
   // accessors for x & y
   int getX() { return x; }
```

```
    int getY() { return y; }
    void setX(int newx) { x = newx; }
    void setY(int newy) { y = newy; }

    // move the x & y position
    void moveTo(int newx, int newy) {
       setX(newx);
       setY(newy);
    }
    void rMoveTo(int deltax, int deltay) {
       moveTo(getX() + deltax, getY() + deltay);
    }
    // virtual draw method
    abstract void draw();
    abstract double getArea();
    abstract double getPrimeter();
}
```

Retangle class inherits Shape. Because Shape is an abstract class, Rectangle class has to implement all abstract methods of Shape.

Listing 5: Rectangle Class

```
class Rectangle extends Shape {
    private int width;
    private int height;

    // constructor
    Rectangle(int newx, int newy, int newwidth, int newheight) {
       super(newx, newy);
       setWidth(newwidth);
       setHeight(newheight);
    }

    // accessors for the width & height
    int getWidth() { return width; }
    int getHeight() { return height; }
    void setWidth(int newwidth) { width = newwidth; }
    void setHeight(int newheight) { height = newheight; }

    // draw the rectangle
    void draw() {
       System.out.println("Drawing a Rectangle at:(" + getX() + ", " + getY() +
          "), width " + getWidth() + ", height " + getHeight());
    }

    double getArea(){
        return width * height;
    }

    double getPrimeter(){
        return 2 * (width + height);
    }
}
```

Circle class inherits Shape. Because Shape is an abstract class, Circle class has to implement all abstract methods of Shape.

Listing 6: Circle Class

```
class Circle extends Shape {
   private int radius;

   // constructor
   Circle(int newx, int newy, int newradius) {
      super(newx, newy);
      setRadius(newradius);
   }

   // accessors for the radius
   int getRadius() { return radius; }
   void setRadius(int newradius) { radius = newradius; }

   // draw the circle
   void draw() {
      System.out.println("Drawing a Circle at:(" + getX() + ", " + getY() +
         "), radius " + getRadius());
   }

   double getArea(){
       return radius * radius * Math.PI;
   }
   double getPrimeter(){
       return 2 * Math.PI * radius;
   }
}
```

Square is a kind of Rectangle. Therefore, Square Class inherits Rectangle class. because Rectangle implemts the abstract methods of Shape, square does not have to implement them.

Listing 7: Square Class

```
class Square extends Rectangle{
    public Square(int top, int left , int side){
        super(top,left ,side ,side);
    }
}
```

Now, you add Triangle and other geometric shapes. Here is the code that test the shape class.

Listing 8: Test Class

```
public class ShapeTest {
    public static void main(String argv[]) {

        // create some shape instances
        Shape shapes[] = new Shape[3];
        shapes[0] = new Rectangle(10, 20, 5, 6);
        shapes[1] = new Circle(15, 25, 8);
        shapes[2] = new Square(30, 30, 10);

        // iterate through the list and handle shapes polymorphically
        for (Shape s: shapes){
            s.draw();
            s.rMoveTo(100, 100);
            s.draw();
            double a = s.getArea();
            double p = s.getPrimeter();
            System.out.println("area=" + a +  " \tPrimeter=" + p);
        }

        // call a rectangle specific function
        Rectangle arect = new Rectangle(0, 0, 15, 15);
        arect.setWidth(30);
        arect.draw();
        System.out.println("area=" + arect.getArea() +
        " \tPrimeter=" + arect.getPrimeter());
    }
}
```