Lab 04 Solutions for Practice problems

Topic Java Tutorial - User Data Types

Additional information:
https://introcs.cs.princeton.edu/java/30oop/

Task 1: Write a function to check the validity of a DNA sequence.

```
/*************************************************************************
 *  Compilation:  javac PotentialGene.java
 *  Execution:    java PotentialGene < input.txt
 *
 *  Determines whether a a DNA string corresponds to a potential gene
 *      - length is a multiple of 3
 *      - starts with the start codon (ATG)
 *      - ends with a stop codon (TAA or TAG or TGA)
 *      - has no intervening stop codons
 *
 *  % java PotentialGene ATGCGCCTGCGTCTGTACTAG
 *  true
 *
 *  % java PotentialGene ATGCGCTGCGTCTGTACTAG
 *  false
 *
 *************************************************************************/

public class PotentialGene {

    public static boolean isPotentialGene(String dna) {

        // Length is a multiple of 3.
        if (dna.length() % 3 != 0) return false;

        // Starts with start codon.
        if (!dna.startsWith("ATG")) return false;

        // No intervening stop codons.
        for (int i = 3; i < dna.length() - 3; i++) {
            if (i % 3 == 0) {
                String codon = dna.substring(i, i+3);
                if (codon.equals("TAA")) return false;
                if (codon.equals("TAG")) return false;
                if (codon.equals("TGA")) return false;
            }
        }

        // Ends with a stop codon.
        if (dna.endsWith("TAA")) return true;
        if (dna.endsWith("TAG")) return true;
        if (dna.endsWith("TGA")) return true;

        return false;
    }
```

```
    public static void main(String[] args) {
        String dna = args[0];
        StdOut.println(isPotentialGene(dna));
    }
}
```

Task 2: Implement the Stopwatch API

```
/******************************************************************************
 *  Compilation:  javac Stopwatch.java
 *  Execution:    java Stopwatch n
 *  Dependencies: none
 *
 *  A utility class to measure the running time (wall clock) of a program.
 *
 *  % java8 Stopwatch 100000000
 *  6.666667e+11  0.5820 seconds
 *  6.666667e+11  8.4530 seconds
 *
 ******************************************************************************/

/**
 *  The {@code Stopwatch} data type is for measuring
 *  the time that elapses between the start and end of a
 *  programming task (wall-clock time).
 *
 *  See {@link StopwatchCPU} for a version that measures CPU time.
 *
 *  @author Robert Sedgewick
 *  @author Kevin Wayne
 */


public class Stopwatch {

    private final long start;

    /**
     * Initializes a new stopwatch.
     */
    public Stopwatch() {
        start = System.currentTimeMillis();
    }


    /**
     * Returns the elapsed CPU time (in seconds) since the stopwatch was created.
     *
     * @return elapsed CPU time (in seconds) since the stopwatch was created
     */
    public double elapsedTime() {
        long now = System.currentTimeMillis();
        return (now - start) / 1000.0;
    }


    /**
```

```
     * Unit tests the {@code Stopwatch} data type.
     * Takes a command-line argument {@code n} and computes the
     * sum of the square roots of the first {@code n} positive integers,
     * first using {@code Math.sqrt()}, then using {@code Math.pow()}.
     * It prints to standard output the sum and the amount of time to
     * compute the sum. Note that the discrete sum can be approximated by
     * an integral - the sum should be approximately 2/3 * (n^(3/2) - 1).
     *
     * @param args the command-line arguments
     */
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);

        // sum of square roots of integers from 1 to n using Math.sqrt(x).
        Stopwatch timer1 = new Stopwatch();
        double sum1 = 0.0;
        for (int i = 1; i <= n; i++) {
            sum1 += Math.sqrt(i);
        }
        double time1 = timer1.elapsedTime();
        StdOut.printf("%e (%.2f seconds)\n", sum1, time1);

        // sum of square roots of integers from 1 to n using Math.pow(x, 0.5).
        Stopwatch timer2 = new Stopwatch();
        double sum2 = 0.0;
        for (int i = 1; i <= n; i++) {
            sum2 += Math.pow(i, 0.5);
        }
        double time2 = timer2.elapsedTime();
        StdOut.printf("%e (%.2f seconds)\n", sum2, time2);
    }
}
```

Task 3: Create an immutable Point data type.

```
/******************************************************************************
 *  Compilation:  javac Point.java
 *  Execution:    java Point
 *
 *  Immutable data type for 2D points.
 *
 ******************************************************************************/

public class Point {
    private double x;   // Cartesian
    private double y;   // coordinates

    // create and initialize a point with given (x, y)
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    // return Euclidean distance between invoking point p and q
    public double distanceTo(Point that) {
        double dx = this.x - that.x;
        double dy = this.y - that.y;
        return Math.sqrt(dx*dx + dy*dy);
```

```
    }

    // draw point using standard draw
    public void draw() {
        StdDraw.point(x, y);
    }

    // draw the line from the invoking point p to q using standard draw
    public void drawTo(Point that) {
        StdDraw.line(this.x, this.y, that.x, that.y);
    }

    // return string representation of this point
    public String toString() {
        return "(" + x + ", " + y + ")";
    }



    // test client
    public static void main(String[] args) {
        Point p = new Point(0.6, 0.2);
        StdOut.println("p  = " + p);
        Point q = new Point(0.5, 0.5);
        StdOut.println("q  = " + q);
        StdOut.println("dist(p, q) = " + p.distanceTo(q));
    }
}
```

4. Practice Problems

1. Write a function `reverse()` that takes a string as an argument and returns a string that contains the same sequence of characters as the argument string but in reverse order.

2. Write a static method `isValidDNA()` that takes a string as its argument and returns `true` if and only if it is composed entirely of the characters `A`, `T`, `C`, and `G`.

3. Implement a data type Rational.java numbers that supports addition, subtraction, multiplication, and division.

public class Rational		
	Rational(int numerator, int denominator)	
Rational	plus(Rational b)	*sum of this number and b*
Rational	minus(Rational b)	*difference of this number and b*
Rational	times(Rational b)	*product of this number and b*
Rational	divides(Rational b)	*quotient of this number and b*
String	toString()	*string representation*

```
/*************************************************************************
 *  Compilation:  javac Rational.java
 *  Execution:    java Rational
 *
```

```
 *  ADT for nonnegative Rational numbers. Bare-bones implementation.
 *  Cancel common factors, but does not stave off overflow. Does not
 *  support negative fractions.
 *
 *  Invariant: all Rational objects are in reduced form (except
 *  possibly while modifying).
 *
 *  Remarks
 *  --------
 *    - See http://www.cs.princeton.edu/introcs/92symbolic/BigRational.java.html
 *      for a version that supports negative fractions and arbitrary
 *      precision numerators and denominators.
 *
 *  % java Rational
 *  5/6
 *  1
 *  28/51
 *  17/899
 *  0
 *
 ******************************************************************************/

public class Rational {
    private int num;   // the numerator
    private int den;   // the denominator

    // create and initialize a new Rational object
    public Rational(int numerator, int denominator) {
        if (denominator == 0) {
            throw new RuntimeException("Denominator is zero");
        }
        int g = gcd(numerator, denominator);
        num = numerator   / g;
        den = denominator / g;

    }

    // return string representation of (this)
    public String toString() {
        if (den == 1) return num + "";
        else          return num + "/" + den;
    }

    // return (this * b)
    public Rational times(Rational b) {
        return new Rational(this.num * b.num, this.den * b.den);
    }


    // return (this + b)
    public Rational plus(Rational b) {
        int numerator   = (this.num * b.den) + (this.den * b.num);
        int denominator = this.den * b.den;
        return new Rational(numerator, denominator);
    }

    // return (1 / this)
    public Rational reciprocal() { return new Rational(den, num);  }

    // return (this / b)
```

```
   public Rational divides(Rational b) {
      return this.times(b.reciprocal());
   }


  /**********************************************************************
   *  Helper functions
   **********************************************************************/

   // return gcd(m, n)
   private static int gcd(int m, int n) {
      if (0 == n) return m;
      else return gcd(n, m % n);
   }


  /**********************************************************************
   *  Test client
   **********************************************************************/

   public static void main(String[] args) {
      Rational x, y, z;

      // 1/2 + 1/3 = 5/6
      x = new Rational(1, 2);
      y = new Rational(1, 3);
      z = x.plus(y);
      StdOut.println(z);

      // 8/9 + 1/9 = 1
      x = new Rational(8, 9);
      y = new Rational(1, 9);
      z = x.plus(y);
      StdOut.println(z);

      //  4/17 * 7/3 = 28/51
      x = new Rational(4, 17);
      y = new Rational(7,  3);
      z = x.times(y);
      StdOut.println(z);

      // 203/16957 * 9299/5887 = 17/899
      x = new Rational(203, 16957);
      y = new Rational(9299, 5887);
      z = x.times(y);
      StdOut.println(z);

      // 0/6 = 0
      x = new Rational(0, 6);
      StdOut.println(x);

   }

  }
```

4. Create a `Rectangle` ADT that represents a rectangle. Represent a rectangle by two points. Include a constructor, a `toString` method, and a method for computing the area.

```
/*************************************************************************
 *  Compilation:  javac Rectangle.java
 *  Execution:    none
 *
 *  Immutable data type for axis-aligned rectangle.
 *
 *************************************************************************/

/**
 *  The {@code Rectangle} class is an immutable data type to encapsulate a
 *  two-dimensional axis-aligned rectagle with real-value coordinates.
 *  The rectangle is <em>closed</em>—it includes the points on the boundary.
 *
 *  @author Robert Sedgewick
 *  @author Kevin Wayne
 */

public final class Rectangle {
    private final double xmin, ymin;   // minimum x- and y-coordinates
    private final double xmax, ymax;   // maximum x- and y-coordinates

    /**
     * Initializes a new rectangle [<em>xmin</em>, <em>xmax</em>]
     * x [<em>ymin</em>, <em>ymax</em>].
     *
     * @param  xmin the <em>x</em>-coordinate of the lower-left endpoint
     * @param  xmax the <em>x</em>-coordinate of the upper-right endpoint
     * @param  ymin the <em>y</em>-coordinate of the lower-left endpoint
     * @param  ymax the <em>y</em>-coordinate of the upper-right endpoint
     * @throws IllegalArgumentException if any of {@code xmin},
     *         {@code xmax}, {@code ymin}, or {@code ymax}
     *         is {@code Double.NaN}.
     * @throws IllegalArgumentException if {@code xmax < xmin} or {@code ymax <
ymin}.
     */
    public Rectangle(double xmin, double ymin, double xmax, double ymax) {
        if (Double.isNaN(xmin) || Double.isNaN(xmax))
            throw new IllegalArgumentException("x-coordinate cannot be NaN");
        if (Double.isNaN(ymin) || Double.isNaN(ymax))
            throw new IllegalArgumentException("y-coordinates cannot be NaN");
        if (xmax < xmin || ymax < ymin) {
            throw new IllegalArgumentException("Invalid rectangle");
        }
        this.xmin = xmin;
        this.ymin = ymin;
        this.xmax = xmax;
        this.ymax = ymax;
    }

    /**
     * Returns the minimum <em>x</em>-coordinate of any point in this rectangle.
     *
     * @return the minimum <em>x</em>-coordinate of any point in this rectangle
     */
    public double xmin() {
        return xmin;
    }

    /**
     * Returns the maximum <em>x</em>-coordinate of any point in this rectangle.
```

```
     *
     * @return the maximum <em>x</em>-coordinate of any point in this rectangle
     */
    public double xmax() {
        return xmax;
    }

    /**
     * Returns the minimum <em>y</em>-coordinate of any point in this rectangle.
     *
     * @return the minimum <em>y</em>-coordinate of any point in this rectangle
     */
    public double ymin() {
        return ymin;
    }

    /**
     * Returns the maximum <em>y</em>-coordinate of any point in this rectangle.
     *
     * @return the maximum <em>y</em>-coordinate of any point in this rectangle
     */
    public double ymax() {
        return ymax;
    }

    /**
     * Returns the width of this rectangle.
     *
     * @return the width of this rectangle {@code xmax - xmin}
     */
    public double width() {
        return xmax - xmin;
    }

    /**
     * Returns the height of this rectangle.
     *
     * @return the height of this rectangle {@code ymax - ymin}
     */
    public double height() {
        return ymax - ymin;
    }

    /**
     * Returns true if the two rectangles intersect.
     *
     * @param  that the other rectangle
     * @return {@code true} if this rectangle intersect the argument
               rectagnle at one or more points, including on the boundary
     */
    public boolean intersects(Rectangle that) {
        return this.xmax >= that.xmin && this.ymax >= that.ymin
            && that.xmax >= this.xmin && that.ymax >= this.ymin;
    }

    /**
     * Returns true if this rectangle contain the rectangle.
     * @param  rect the rectangle
     * @return {@code true} if this rectangle contain the rectangle {@code
rect},
```

```
           possibly at the boundary; {@code false} otherwise
     */
    public boolean contains(Rectangle rect) {
        return (rect.xmin >= xmin) && (rect.xmax <= xmax)
            && (rect.ymin >= ymin) && (rect.ymax <= ymax);
    }

    /**
     * Compares this rectangle to the specified rectangle.
     *
     * @param  other the other rectangle
     * @return {@code true} if this rectangle equals {@code other};
     *         {@code false} otherwise
     */
    @Override
    public boolean equals(Object other) {
        if (other == this) return true;
        if (other == null) return false;
        if (other.getClass() != this.getClass()) return false;
        Rectangle that = (Rectangle) other;
        if (this.xmin != that.xmin) return false;
        if (this.ymin != that.ymin) return false;
        if (this.xmax != that.xmax) return false;
        if (this.ymax != that.ymax) return false;
        return true;
    }

    /**
     * Returns an integer hash code for this rectangle.
     * @return an integer hash code for this rectangle
     */
    @Override
    public int hashCode() {
        int hash1 = ((Double) xmin).hashCode();
        int hash2 = ((Double) ymin).hashCode();
        int hash3 = ((Double) xmax).hashCode();
        int hash4 = ((Double) ymax).hashCode();
        return 31*(31*(31*hash1 + hash2) + hash3) + hash4;
    }

    /**
     * Returns a string representation of this rectangle.
     *
     * @return a string representation of this rectangle, using the format
     *         {@code [xmin, xmax] x [ymin, ymax]}
     */
    @Override
    public String toString() {
        return "[" + xmin + ", " + xmax + "] x [" + ymin + ", " + ymax + "]";
    }

    /**
     * Draws this rectangle to standard draw.
     */
    public void draw() {
        StdDraw.line(xmin, ymin, xmax, ymin);
        StdDraw.line(xmax, ymin, xmax, ymax);
        StdDraw.line(xmax, ymax, xmin, ymax);
        StdDraw.line(xmin, ymax, xmin, ymin);
    }
```

```
}
```

5. **Encapsulation.** Is the following class immutable?

```
import java.util.Date

public class Appointment {
    private Date date;
    private String contact;

    public Appointment(Date date) {
        this.date = date;
        this.contact = contact;
    }

    public Date getDate() {
        return date;
    }
```

Solution: No, because Java's java.util.Date is mutable. To correct, make a defensive copy of the date in the constructor and make a defensive copy of the date before returning to the client.