TÖL101G - Tölvunarfræði 1

Vikublað 10

Lausn

Æfingar

3.2.7

Hér er fastayrðing gagna að ræða talan $\frac{a}{b}$ þarf að vera þ.a. a og b eru ósamþátta og $b > 0$, þ.e. formerkið er bara geymt í a breytunni. Þetta þýðir að meiri vinna er framkvæmd í smiðnum, t.d. er deilt út stærsta sameiginlega þætti og formerkið fundið og sett eingöngu í a. Þar sem breyturnar a, b eru merktar sem `final` verður að reikna út gildið sem á að fara í þær og svo gefa þeim gildi, því það er bara hægt að gera einu sinni í smiðnum. Þar sem þessi vinna fer fram í smiðnum þurfum við ekki að hafa áhyggjur af þessum útreikningum í öðrum föllum, t.d. í `plus` fallinu er búin til nýr `Rational` hlutur með teljara og nefnara sem verða fullstyttir á endanum.

```
public class Rational {
  private final int a,b;
  // a og b eru ósamþátta og b > 0

  // Notkun: x = new Rational(n,m)
  // Fyrir:  m != 0
  // Eftir:  x er ræða talan n/m
  public Rational(int n, int m) {
    if (m == 0) {
      throw new IllegalArgumentException("Ekki deila með 0!");
    }

    int g = gcd(n,m);

    if (m < 0) {
      m = -m;
      n = -n;
    }
    this.a = n/g;
    this.b = m/g;
  }

  // Notkun: x = a.plus(b)
  // Fyrir:  ekkert
  // Eftir:  x er ræða talan a+b
  public Rational plus(Rational o) {
    return new Rational(this.a*o.b + o.a*this.b, this.b*o.b);
  }

  // Notkun: x = a.minus(b)
  // Fyrir:  ekkert
```

```
  // Eftir:  x er ræða talan a-b
  public Rational minus(Rational o) {
    return new Rational(this.a*o.b - o.a*this.b, this.b*o.b);
  }

  // Notkun: x = a.times(b)
  // Fyrir:  ekkert
  // Eftir:  x er ræða talan a*b
  public Rational times(Rational o) {
    return new Rational(this.a*o.a, this.b*o.b);
  }

  // Notkun: x = a.over(b)
  // Fyrir:  b er ekki ræða talan 0
  // Eftir:  x er ræða talan a/b
  public Rational over(Rational o) {
    return new Rational(this.a*o.b, this.b*o.a);
  }

  public String toString() {
    return this.a+"/"+this.b;
  }

  // Notkun: g = gcd(p,q)
  // Fyrir:  (p,q) er ekki (0,0)
  // Eftir:  g er minnsta jákvæða heiltala sem
  //         gengur bæði upp í p og q
  private int gcd(int p, int q) {
    if (q == 0) return Math.abs(p);
    else return gcd(q,p%q);
  }

  // do not modify this code
  public static void main(String[] args) {
    Rational a = new Rational(Integer.parseInt(args[0]), Integer.parseInt(args[1]));
    Rational b = new Rational(Integer.parseInt(args[2]), Integer.parseInt(args[3]));

    System.out.println(a.plus(b) + " " + a.minus(b) + " " + a.times(b) + " " + a.over
  }
}
```

3.2.6 og 3.3.2

Í þessari lausn útfærum við líka `random` fallið, jafnvel þó að þess þyrfti ekki á Cloudcoder.

```
public class Location {
  private final double lat,lon;
  // -90 <= lat <= 90, -180 < lon <= 180
```

```
// Notkun: x = new Location(lat,lon)
// Fyrir:  -90 <= lat <= 90 og -180 < lon < 180
// Eftir:  x er Location hlutur með hnitin lat,lon
public Location(double lat, double lon) {
  this.lat = lat;
  this.lon = lon;
}

// Notkun: d = a.distanceTo(b)
// Fyrir:  ekkert
// Eftir:  d er fjarlægð milli a og b eftir jörðinni
public double distanceTo(Location b) {
  double lat_a = Math.toRadians(this.lat);
  double lat_b = Math.toRadians(b.lat);
  double lon_a = Math.toRadians(this.lon);
  double lon_b = Math.toRadians(b.lon);

  double angle = Math.acos(Math.sin(lat_a)*Math.sin(lat_b)
                           + Math.cos(lat_a)*Math.cos(lat_b)
                           * Math.cos(lon_a-lon_b));
  return 60 * Math.toDegrees(angle);
}

public String toString() {
  return lat+", "+lon;
}

// Notkun: x = Location.random()
// Fyrir:  ekkert
// Eftir:  x er Location hlutur jafndreifður yfir jörðina
public static Location random() {
  double u = Math.random();
  double v = Math.random();

  // sjá http://mathworld.wolfram.com/SpherePointPicking.html
  double lat = 180.0-360.0*u;
  double lon = 90.0-(180.0/Math.PI)*Math.acos(2*v-1);
  return new Location(lat,lon);
}

// Notkun: x = Location.parseLocation(loc)
// Fyrir:  loc er á forminu "lat [N|S], lon [E|W]"
// Eftir:  x er staðsetning með hnitin lat, long með réttu formerki
public static Location parseLocation(String loc) {
  double lat,lon;
  String[] v = loc.split(",");
```

```
    String a = v[0];
    lat = Double.parseDouble(a.substring(0,a.length()-1));
    if (a.charAt(a.length()-1) == 'S') {
      lat = -lat;
    }

    String b = v[1];
    lon = Double.parseDouble(b.substring(0,b.length()-1));
    if (b.charAt(b.length()-1) == 'W') {
      lon = -lon;
    }

    return new Location(lat,lon);

  }


  // do not modify this code
  public static void main(String[] args) {
    Location reykjavik = new Location(64.1333, -21.9333);
    String s = args[0];
    for (int i = 1; i < args.length; i++) {s += " " + args[i];}
    Location b = Location.parseLocation(s);
    System.out.println(reykjavik.distanceTo(b));
  }

}
```

3.3.8

Hér þarf að bæta við `implements Comparable<Vector>` við lýsinguna á `Vector` til að hægt sé að nota `Vector` klasann með röðunarföllunum.

```
public class Vector implements Comparable<Vector> {
    //...

    // Notkun: c = a.compareTo(b)
    // Fyrir:  ekkert
    // Eftir:  c > 0 ef a hefur stærri evklíðska lengd en b
    //         c < 0 ef b er stærri en a, annars er c = 0
    public int compareTo(Vector b) {
        double ma = this.magnitude();
        double mb = b.magnitude();

        if (ma > mb) {
            return 1;
        } else if (ma < mb) {
            return -1;
        } else {
```

```
            return 0;
        }
    }

}
```

Einfaldari lausn hefði verið að nota `Math.signum()`fallið, sem skilar $-1.0, 0.0$ eða 1.0 eftir formerkinu. t.d.

```
public int compareTo(Vector b) {
  return (int) Math.signum(this.magnitude() - b.magnitude());
}
```

Ef notað notað er `implements Comparable` sem er eldri ritháttur þá þarf `compareTo` fallið að vera

```
public int compareTo(Object o) {
  Vector b = (Vector) o; // Let's hope this is really a vector
  ...
}
```

Verkefni

```
public class Rectangle extends Shape {
    private double x,y,h,w;
    // x,y eru hnitin fyrir miðjuna
    // hæðin er h > 0,  breiddin er w > 0

    // Notkun: r = new Rectangle(x,y,w,h)
    // Fyrir:  w > 0, h > 0
    // Eftir:  r er ferhyrningur með miðju í x,y, hæð h og breidd w
    public Rectangle(double x, double y, double w, double h) {
        this.x = x;
        this.y = y;
        this.w = w;
        this.h = h;
    }

   // Notkun: p = s.getCenter()
    // Fyrir:  ekkert
    // Eftir:  p er miðjan á s
    public  Point2D getCenter() {
        return new Point2D(x,y);
    }

    // Notkun: r = s.getBoundingBox()
    // Fyrir:  ekkert
    // Eftir:  r er minnsti ferhyrningur sem passar utan um s
    public  Rectangle getBoundingBox() {
```

```
    // við viljum ekki skila tilvísun á this því að það er
    // hægt að breyta hlutnum
    return new Rectangle(x,y,w,h);
}

// Notkun: c = s.intersects(o)
// Fyrir:  ekkert
// Eftir:  c er true of s og o skarast
public  boolean intersects(Shape o) {
    Rectangle box = o.getBoundingBox();
    return this.rectangleIntersects(box);
}

// Notkun: c = s.intersects(o)
// Fyrir:  ekkert
// Eftir:  c er true of s og o skarast
private boolean rectangleIntersects(Rectangle o) {
    return (Math.abs(this.x-o.x) <= (this.w+o.w)/2)
        && (Math.abs(this.y-o.y) <= (this.h+o.h)/2);
}

// Notkun: s.scale(f)
// Fyrir:  f > 0
// Eftir:  s er f-sinnum stærra og miðjan er óbreytt
public  void scale(double f) {
    this.w *= f;
    this.h *= f;
}

// Notkun: w = a.getWidth()
// Fyrir:  ekkert
// Eftir:  w er breiddin á a
public double getWidth() {
    return this.w;
}

// Notkun: h = a.getHeight()
// Fyrir:  ekkert
// Eftir:  h er hæðin á a
public double getHeight() {
    return this.h;
}

// Notkun: a.setWidth(w)
// Fyrir:  w > 0
// Eftir:  a hefur breiddina w
public void setWidth(double w) {
    this.w = w;
```

```
    }

    // Notkun: a.setHeight(h)
    // Fyrir:  h > 0
    // Eftir:  a hefur hæðina h
    public void setHeight(double h) {
        this.h = h;
    }
}

public class Circle extends Shape {
    private double x,y,r;

    // Notkun: x = new Circle(x,y,r)
    // Fyrir:  r > 0
    // Eftir:  x er hringur með miðju í x,y og radíus r
    public Circle(double x, double y, double r) {
        this.x = x;
        this.y = y;
        this.r = r;
    }

    // Notkun: p = s.getCenter()
    // Fyrir:  ekkert
    // Eftir:  p er miðjan á s
    public  Point2D getCenter() {
        return new Point2D(x,y);
    }

    // Notkun: r = s.getBoundingBox()
    // Fyrir:  ekkert
    // Eftir:  r er minnsti ferhyrningur sem passar utan um s
    public  Rectangle getBoundingBox() {
        return new Rectangle(x,y,2*r,2*r);
    }

    // Notkun: c = s.intersects(o)
    // Fyrir:  ekkert
    // Eftir:  c er true of s og o skarast
    public  boolean intersects(Shape o) {
        Rectangle box = this.getBoundingBox();
        // látum Rectangle sjá um þetta fyrir okkur
        return box.intersects(o);
    }

    // Notkun: s.scale(f)
    // Fyrir:  f > 0
    // Eftir:  s er f-sinnum stærra og miðjan er óbreytt
    public  void scale(double f) {
```

```
        this.r *= f;
    }

    // Notkun: r = a.getRadius()
    // Fyrir:  ekkert
    // Eftir:  r er radíusinn á a
    public double getRadius() {
        return this.r;
    }

    // Notkun: a.setRadius(r)
    // Fyrir:  r > 0
    // Eftir:  a hefur radíusinn r
    public void setRadius(double r) {
        this.r = r;
    }
}
```

Í þessari lausn forðumst við að bera saman hring og rétthyrning þar sem það er frekar erfitt ná rúmfræðinni réttri. Einfaldari lausn er að nota getBoundingBox() og bera saman rétthyrningana. Ef við viljum bera saman hringi og rétthyrninga þyrfti að breyta kóðanum í Rectangle í

```
public boolean intersects(Shape o) {
    if (o instanceof Rectangle) {
        return this.rectangleIntersects((Rectangle) o);
    } else if (o instanceof Circle) {
        Circle c = (Circle) o;
        // berum saman this og c...
    }
}
```

Þarna notum við `instanceof` virkjann í Java sem segir til um hvort hluturinn o sé af taginu `Circle` eða `Rectangle`.

Ef `Square` klasinn ætti að erfa frá Rectangle þá þyrfti fastayrðing gagna að tryggja að rétthyrningurinn væri með jafnlangar hliðar, þ.e. `w == h` í Rectangle. Þetta væri hægt að gera með smiðnum t.d.

```
// Notkun: r = new Square(x,y,s)
// Fyrir:  s > 0
// Eftir:  r er ferningur með miðju í x,y og hliðarlengd s
public Square(double x, double  y, double s) {
    super(x,y,s,s);
}
```

Aðferðin `scale` myndi varðveita þetta samband milli `w` og `h`, en föllin `setWidth` og `setHeight` myndu annað hvort brjóta fastayrðingu gagna eða reglur um rökstudda forritun með erfðum. Í fyrra tilfellinu væri hægt að nota `setWidth` frá `Rectangle` sem breytir þá `w` tilviksbreytunni en ekki `h`, þá væri fastayrðing gagna fyrir `Square` brotin. Í seinna tilvikinu myndum við yfirskrifa `setWidth` í `Square` með

```
// Notkun: a.setWidth(w)
// Fyrir:  w >= 0
// Eftir:  a hefur breiddina w og hæðina w
public void setWidth(double w) {
    this.w = w;
    this.h = w;
}
```

Þá myndi fallið varðveita fastayrðingu gagna en brjóta regluna um að eftirskilyrðin í yfirskrifaðri aðferð þurfa að vera þrengri en eftirskilyrðin í yfirklasa, þ.e. $E_B \Rightarrow E_A$. Ef `Rectangle` og `Rectangle` væru óbreytanlegir þá væri hægt að láta erfðirnar ganga upp.