Programovanie (4)

Cvičenie 03 - Code review

Alexander Šimko

- Code review je subjektívna záležitosť
- Ak by ju robil niekto iný, mal by na kód iný názor

Hodina geometrie po 1.

Čo vieme zlepšiť?

```
public class Kruh extends DvaD {
   public Bod2D stred;
   public double polomer;

   public Kruh(Bod2D stred, double polomer) {
      super();
      this.stred = stred;
      this.polomer = polomer;
   }

   public boolean jeV(Bod2D b) {
      return Math.pow(stred.getX()-b.getX(), 2) +
             Math.pow(stred.getY()-b.getY(), 2) <= Math.pow(polomer, 2);
   }

}
```

Čo vieme zlepšiť?

```
public class Kruh extends DvaD {
   public Bod2D stred;
   public double polomer;

   public Kruh(Bod2D stred, double polomer) {
      super();
      this.stred = stred;
      this.polomer = polomer;
   }

   public boolean jeV(Bod2D b) {
      return Math.pow(stred.getX()-b.getX(), 2) +
             Math.pow(stred.getY()-b.getY(), 2) <= Math.pow(polomer, 2);
   }

}
```

Vzdialenosť bodov sa týka triedy Bod2D, nie triedy Kruh

Čo vieme zlepš

```
public class Kruh exter
   public Bod2D stred;
   public double polome

   public Kruh(Bod2D st
      super();
      this.stred = stre
      this.polomer = po
   }

   public boolean jeV(Bod2D b) {
      return Math.pow(stred.getX()-b.getX(), 2) +
             Math.pow(stred.getY()-b.getY(), 2) <= Math.pow(polomer, 2);
   }

}
```

```
public class Bod2D {
   private double x, y;

   public double distanceToSquared(Bod2D b) {
      double dx = x - b.x;
      double dy = y - b.y;
      return Math.pow(dx, 2) + Math.pow(dy, 2);
   }
}
```

nie triedy Kruh

```
public boolean jeV(Bod2D b) {
   return stred.distanceToSquared(b) <= Math.pow(polomer, 2);
}
```

Čo vieme zlepšiť?

```
public class Kruh extends DvaD {
    public Bod2D stred;
    public double polomer;

    public Kruh(Bod2D stred, double polomer) {
        super();
        this.stred = stred;
        this.polomer = polomer;
    }

    public void posun(Bod2D b) {
        stred.setX(stred.getX()+b.getX());
        stred.setY(stred.getY()+b.getY());
    }
}

public class Obdlznik extends DvaD {
    public Bod2D lavyDolny;
    public double dx, dy;

    public Obdlznik(Bod2D lavyDolny, double dx, double dy) {
        super();
        this.lavyDolny = lavyDolny;
        this.dx = dx;
        this.dy = dy;
    }

    public void posun(Bod2D b) {
        lavyDolny.setX(lavyDolny.getX()+b.getX());
        lavyDolny.setY(lavyDolny.getY()+b.getY());
    }
}
```

Čo vieme zlepšiť?

```
public class Kruh extends DvaD {
    public Bod2D stred;
    public double polomer;

    public Kruh(Bod2D stred, double polomer) {
        super();
        this.stred = stred;
        this.polomer = polomer;
    }

    public void posun(Bod2D b) {
        stred.setX(stred.getX()+b.getX());
        stred.setY(stred.getY()+b.getY());
    }
}

public class Obdlznik extends DvaD {
    public Bod2D lavyDolny;
    public double dx, dy;

    public Obdlznik(Bod2D lavyDolny, double dx, double dy) {
        super();
        this.lavyDolny = lavyDolny;
        this.dx = dx;
        this.dy = dy;
    }

    public void posun(Bod2D b) {
        lavyDolny.setX(lavyDolny.getX()+b.getX());
        lavyDolny.setY(lavyDolny.getY()+b.getY());
    }
}
```

Duplicita

Čo vieme zlepšiť?

```
public class Kruh extends DvaD {
    public Bod2D stred;
    public double polomer;

    public Kruh(Bod2D stred, double
        super();
        this.stred = stred;
        this.polomer = polomer;
    }

    public void posun(Bod2D b) {
        stred.setX(stred.getX()+b.getX());
        stred.setY(stred.getY()+b.getY());
    }
}
```

```
public class Bod2D {
    private double x, y;

    public void moveBy(Bod2D vector) {
        x += vector.getX();
        y += vector.getY();
    }
}
```

Kód dajme do triedy, ktorej sa týka

```
public class Obdlznik extends DvaD {
    public Bod2D lavyDolny;
    public double dx, dy;

    public Obdlznik(Bod2D lavyDolny, double dx, double dy) {
        super();
        this.lavyDolny = lavyDolny;
        this.dx = dx;
        this.dy = dy;
    }

    public void posun(Bod2D b) {
        lavyDolny.setX(lavyDolny.getX()+b.getX());
        lavyDolny.setY(lavyDolny.getY()+b.getY());
    }
}
```

Duplicita

Čo vieme zlepšiť?

```
public class Kruh extends DvaD {
    public Bod2D stred;
    public double polomer;

    public Kruh(Bod2D stred, double
        super();
        this.stred = stred;
        this.polomer = polomer;
    }

    public void posun(Bod2D b) {
        stred.setX(stred.getX()+b.getX());
        stred.setY(stred.getY()+b.getY());
    }
}

public class Obdlznik extends DvaD
    public Bod2D lavyDolny;
    public double dx, dy;

    public Obdlznik(Bod2D lavyDolny, double dx, double dy) {
        super();
        this.lavyDolny = lavyDolny;
        this.dx = dx;
        this.dy = dy;
    }

    public void posun(Bod2D b) {
        lavyDolny.setX(lavyDolny.
        lavyDolny.setY(lavyDolny.
    }
}
```

```
public class Bod2D {
   private double x, y;

   public void moveBy(Bod2D vector) {
      x += vector.getX();
      y += vector.getY();
   }
}
```

Kód dajme do triedy, ktorej sa týka

```
public void posun(Bod2D b) {
   stred.moveBy(b);
}
```

Duplicita

```
public void posun(Bod2D b) {
   lavyDolny.moveBy(b);
}
```

Čo vieme zlepšiť?

```
public class Obdlznik extends DvaD {
   public Bod2D lavyDolny;
   public double dx, dy;

   public Obdlznik(Bod2D lavyDolny, double dx, double dy) {
      super();
      this.lavyDolny = lavyDolny;
      this.dx = dx;
      this.dy = dy;
   }

   public boolean jeV(Bod2D b) {
      double x1 = lavyDolny.getX();
      double x2 = b.getX();
      double y1 = lavyDolny.getY();
      double y2 = b.getY();

      return (x2 >= x1 && x2 <= x1+dx) &&
            (y2 >= y1 && y2 <= y1+dy);
   }
}
```

Čo vieme zlepšiť?

```
public class Obdlznik extends DvaD {
   public Bod2D lavyDolny;
   public double dx, dy;

   public Obdlznik(Bod2D lavyDolny, double dx, double dy) {
      super();
      this.lavyDolny = lavyDolny;
      this.dx = dx;
      this.dy = dy;
   }

   public boolean jeV(Bod2D b) {
      double x1 = lavyDolny.getX();
      double x2 = b.getX();
      double y1 = lavyDolny.getY();
      double y2 = b.getY();

      return (x2 >= x1 && x2 <= x1+dx) &&
             (y2 >= y1 && y2 <= y1+dy);
   }
}
```

x1, x2 evokuje, že x1 bude ľavý okraj, x2 pravý okraj

Čo vieme zlepšiť?

```
public class Obdlznik extends DvaD {
   public Bod2D lavyDolny;
   public double dx, dy;

   public Obdlznik(Bod2D lavyDolny, double dx, double dy) {
      super();
      this.lavyDolny = lavyDolny;
      this.dx = dx;
      this.dy = dy;
   }

   public boolean jeV(Bod2D b) {
      double x1 = lavyDolny.getX();
      double x2 = b.getX();
      double y1 = lavyDolny.getY();
      double y2 = b.getY();

      return (x2 >= x1 && x2 <= x1+dx) &&
             (y2 >= y1 && y2 <= y1+dy);
   }
}
```

x1, x2 evokuje, že x1 bude ľavý okraj, x2 pravý okraj

ale

x1 je ľavý okraj

a

x2 je súradnica kontrolovaného bodu

Čo vieme zlepšiť?

```
public class Obdlznik extends DvaD {
   public Bod2D lavyDolny;
   public double dx, dy;

   public Obdlznik(Bod2D lavyDolny, double dx, double dy) {
      super();
      this.lavyDolny = lavyDolny;
      this.dx = dx;
      this.dy = dy;
   }

   public boolean jeV(Bod
      double x1 = lavyDol
      double x2 = b.getX(
      double y1 = lavyDol
      double y2 = b.getY(

      return (x2 >= x1 &&
            (y2 >= y1 && y
   }
}
```

```
public boolean jeV(Bod2D b) {
   double x1 = lavyDolny.getX();
   double x2 = x1 + dx;
   double y1 = lavyDolny.getY();
   double y2 = y1 + dy;

   double x = b.getX() ;
   double y = b.getY();

   return  x1 <= x && x <= x2 &&
         y1 <= y && y <= y2;
}
```

Polynómy po 1.

Čo príde, by sa s testami neskompilovalo kvôli predpísanému rozhraniu tried. V produkčnom kóde je to ale dobré urobiť.

Čo vieme zlepšiť?

```
public class Premenna extends Polynom {

    private final String name;
    public Premenna(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    @Override
    Double valueAt(String[] vars, double[] values) {
        for (int i = 0; i < vars.length; i++) {
            if (vars[i].equals(name)) {
                return values[i];
            }
        }
        return null;
    };
    @Override
    Polynom derive(String var) {
        if (var.equals(name)) {
            return new Konstanta(1);
        }
        return new Konstanta(0);
    };
}
```

Čo vieme zlepšiť?

```
public class Premenna extends Polynom {

    private final String name;
    public Premenna(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    @Override
    Double valueAt(String[] vars, double[] values) {
        for (int i = 0; i < vars.length; i++) {
            if (vars[i].equals(name)) {
                return values[i];
            }
        }
        return null;
    };
    @Override
    Polynom derive(String var) {
        if (var.equals(name)) {
            return new Konstanta(1);
        }
        return new Konstanta(0);
    };
}
```

Čím sa líšia dve inštancie pre 1?

Čím sa líšia dve inštancie pre 0?

Čo vieme zlepšiť?

```
public class Konstanta extends Polynom {

    private final double value;
    public Konstanta(double value) {
        this.value = value;
    }

    @Override
    public String toString() {
        return Double.toString(value);
    }

    @Override
    Double valueAt(String[] vars, double[] values) {
        return value;
    };
    @Override
    Polynom derive(String var) {
        return new Konstanta(0);
    };
}
```

Vytvorená inštakcia triedy Konstanta sa nemení

Čo vieme zlepšiť?

p

```
public class Konstanta extends Polynom {

    private static final Konstanta ZERO = new Konstanta(0);
    private static final Konstanta ONE = new Konstanta(1);

    public static Konstanta zero() { return ZERO; }
    public static Konstanta one() { return ONE; }

    public static Konstanta of(double value) {
        if (value == 0) return zero();
        if (value == 1) return one();
        return new Konstanta(value);
    }

    private final double value;

    private Konstanta(double value) {
        this.value = value;
    }

    …
    Polynom derive(String var) { return zero(); }
}
```

Čo vieme zlepšiť?

```
public class Konstanta extends Polynom {

    private static final Konstanta ZERO = new Konstanta(0);
    private static final Konstanta ONE = new Konstanta(1);

    public static Konstanta zero() { return ZERO; }
    public static Konstanta one() { return ONE; }

    public static Konstanta of(double value) {
        if (value == 0) return zero();
        if (value == 1) return one();
        return new Konstanta(value);
    }

    private final double value;

    private Konstanta(double value) {
        this.value = value;
    }

    …
    Polynom derive(String var) { return zero(); }
}
```

Kľúčové

Čo vieme zlepšiť?

```
public class Konstanta extends Polynom {

    private static final Konstanta ZERO = new Konstanta(0);
    private static final Konstanta ONE = new Konstanta(1);

    public static Konstanta zero() { return ZERO; }
    public static Konstanta one() { return ONE; }

    public static Konstanta of(double value) {
        if (value == 0) return zero();
        if (value == 1) return one();
        return new Konstanta(value);
    }

    private final double value;

    private Konstanta(double value) {
        this.value = value;
    }

    …
    Polynom derive(String var) { return zero(); }
}
```

Inštancie získavame cez tieto factory metódy

Kľúčové

p

Čo vieme zlepšiť?

```
public class Premenna extends Polynom {

    private final String name;
    public Premenna(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    @Override
    Double valueAt(String[] vars, double[] values) {
        for (int i = 0; i < vars.length; i++) {
            if (vars[i].equals(name)) {
                return values[i];
            }
        }
        return null;
    }
    @Override
    Polynom derive(String var) {
        if (var.equals(name)) {
            return new Konstanta(1);
        }
        return new Konstanta(0);
    }
}
```

Čo vieme zlepšiť?

```
public class Premenna extends Polynom {

    private final String name;
    public Premenna(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    @Override
    Double valueAt(String[] vars, double[] values) {
        for (int i = 0; i < vars.length; i++) {
            if (vars[i].equals(name)) {
                return values[i];
            }
        }
        i
    }
    @Ove
    Polyn
        i
        i
        i
    }
}
```

```
@Override
Polynom derive(String var) {
    if (var.equals(name)) {
        return Konstanta.one();
    }
    return Konstanta.zero();
}
```

Čo vieme zlepšiť?

```
public class Sucet extends Polynom {

    private final Polynom a;
    private final Polynom b;
    public Sucet(Polynom a, Polynom b) {
        this.a = a;
        this.b = b;
    }

    @Override
    Polynom derive(String var) {
        Polynom derA = a.derive(var);
        Polynom derB = b.derive(var);

        return new Sucet(derA, derB);
    }
}
```

Čo vieme zlepšiť?

```
public class Sucet extends Polynom {

    private final Polynom a;
    private final Polynom b;
    public Sucet(Polynom a, Polynom b) {
        this.a = a;
        this.b = b;
    }

    @Override
    Polynom derive(String var) {
        Polynom derA = a.derive(var);
        Polynom derB = b.derive(var);

        return new Sucet(derA, derB);
    }
}
```

Ak aspoň jedno je 0, je zbytočné vytvárať inštanciu súčtu

Čo vieme zlepšiť?

```
pu public class Sucet extends Polynom {

    public static Polynom of(Polynom a, Polynom b) {
        if (a == Konstanta.zero()) return b;
        if (b == Konstanta.zero()) return a;
        return new Sucet(a, b);
    }

    private final Polynom a;
    private final Polynom b;
    public Sucet(Polynom a, Polynom b) {
        this.a = a;
        this.b = b;
    }

    @Override
}   Polynom derive(String var) {
        Polynom derA = a.derive(var);
        Polynom derB = b.derive(var);

        return Sucet.of(derA, derB);
    }
}
```

Čo vieme zlepšiť?

```
public class Sucin extends Polynom  {
    private final Polynom a;
    private final Polynom b;
    public Sucin(Polynom a, Polynom b) {
        this.a = a;
        this.b = b;
    }

    Polynom derive(String var) {
        Polynom derA = a.derive(var);
        Polynom derB = b.derive(var);

        return new Sucet(
                new Sucin(derA, b),
                new Sucin(derB, a)
                );
    }
}
```

Čo vieme zlepšiť?

```
public class Sucin extends Polynom  {
    private final Polynom a;
    private final Polynom b;
    public Sucin(Polynom a, Polynom b) {
        this.a = a;
        this.b = b;
    }

    Polynom derive(String var) {
        Polynom derA = a.derive(var);
        Polynom derB = b.derive(var);

        return new Sucet(
                new Sucin(derA, b),
                new Sucin(derB, a)
                );
    }
}
```

Čo vieme zlepšiť?

```
public class Sucin extends Polynom  {
    private final Polynom a;
    private final Polynom b;
    public Sucin(Polynom a, Polynom b) {
        this.a = a;
        this.b = b;
    }

    Polynom derive(String var) {
        Polynom derA = a.derive(var);
        Polynom derB = b.derive(var);

        return Sucet.of(
                new Sucin(derA, b),
                new Sucin(derB, a)
                );
    }
}
```

Čo vieme zlepšiť?

```
public class Sucin extends Polynom  {
    private final Polynom a;
    private final Polynom b;
    public Sucin(Polynom a, Polynom b) {
        this.a = a;
        this.b = b;
    }

    Polynom derive(String var) {
        Polynom derA = a.derive(var);
        Polynom derB = b.derive(var);

        return Sucet.of(
                new Sucin(derA, b),
                new Sucin(derB, a)
                );
    }
}
```

Ak aspoň jedno je 1, je zbytočné vytvárať inštanciu súčinu

Čo vieme zlepšiť?

```
pu public class Sucin extends Polynom  {

    public static Polynom of(Polynom a, Polynom b) {
        if (a == Konstanta.one()) return b;
        if (b == Konstanta.one()) return a;
        return new Sucin(a, b);
    }

    private final Polynom a;
    private final Polynom b;
    public Sucin(Polynom a, Polynom b) {
        this.a = a;
        this.b = b;
    }
    Polynom derive(String var) {
        Polynom derA = a.derive(var);
        Polynom derB = b.derive(var);
}

        return Sucet.of(
                Sucin.of(derA, b),
                Sucin.of(derB, a)
                );
    };
}
```

Čo vieme zlepšiť?

```
pu public class Sucin extends Polynom  {

    public static Polynom of(Polynom a, Polynom b) {
        if (a == Konstanta.one()) return b;
        if (b == Konstanta.one()) return a;
        return new Sucin(a, b);
    }

    private final Polynom a;
    private final Polynom b;
    public Sucin(Polynom a, Polynom b) {
        this.a = a;
        this.b = b;
    }
    Polynom derive(String var) {
        Polynom derA = a.derive(var);
        Polynom derB = b.derive(var);
}
        return Sucet.of(
                Sucin.of(derA, b),
                Sucin.of(derB, a)
                );
    };
}
```

Čo ak jedno z toho je 0?

Čo vieme zlepšiť?

```
Double.POSITIVE_INFINITY * 0.0d    ->    NaN
Double.NEGATIVE_INFINITY * 0.0d    ->    NaN
Double.NaN * 0.0d                  ->    NaN
```

Čo vieme zlepšiť?

```
Double.POSITIVE_INFINITY * 0.0d    ->    NaN
Double.NEGATIVE_INFINITY * 0.0d    ->    NaN
Double.NaN * 0.0d                  ->    NaN
```

Takže a * 0 nemusí byť 0

Čo vieme zlepšiť?

```
Double.POSITIVE_INFINITY * 0.0d    ->    NaN
Double.NEGATIVE_INFINITY * 0.0d    ->    NaN
Double.NaN * 0.0d                  ->    NaN
```

Takže a * 0 nemusí byť 0

Aby sme vedeli, či násobenie nulou môžeme optimalizovať, musíme si ujasniť, ako sa má vyhodnotiť polynóm pre vstupy: POSITIVE_INFINITY, NEGATIVE_INFINITY a NaN

Čo vieme zlepšiť?

```
Double.POSITIVE_INFINITY * 0.0d    ->    NaN
Double.NEGATIVE_INFINITY * 0.0d    ->    NaN
Double.NaN * 0.0d                  ->    NaN
```

Takže a * 0 nemusí byť 0

Aby sme vedeli, či násobenie nulou môžeme optimalizovať, musíme si ujasniť, ako sa má vyhodnotiť polynóm pre vstupy: POSITIVE_INFINITY, NEGATIVE_INFINITY a NaN

Zadanie tieto hodnoty nespomína

Čo vieme zlepšiť?

```
pu
}
```

```
public class Sucin extends Polynom  {

    public static Polynom of(Polynom a, Polynom b) {
        if (a == Konstanta.one()) return b;
        if (b == Konstanta.one()) return a;
        if (a == Konstanta.zero() || b == Konstanta.zero())
          return Konstanta.zero();
        return new Sucin(a, b);
    }

    private final Polynom a;
    private final Polynom b;
    public Sucin(Polynom a, Polynom b) {
        this.a = a;
        this.b = b;
    }
    Polynom derive(String var) {
        Polynom derA = a.derive(var);
        Polynom derB = b.derive(var);

        return Sucet.of(
                Sucin.of(derA, b),
                Sucin.of(derB, a)
                );
    };
```

Čo sme docielili?

Vstup: (((x * y) + 2.0) * (((x * y) + 2.0) * ((x * y) + 2.0)))

Pôvodný derive podľa x: ((((((1.0 * y) + (0.0 * x)) + 0.0) * (((x * y) + 2.0) * ((x * y) + 2.0))) + ((((((1.0 * y) + (0.0 * x)) + 0.0) * ((x * y) + 2.0)) + ((((1.0 * y) + (0.0 * x)) + 0.0) * ((x * y) + 2.0))) * ((x * y) + 2.0)))

Nový derive podľa x: ((y * (((x * y) + 2.0) * ((x * y) + 2.0))) + (((y * ((x * y) + 2.0)) + (y * ((x * y) + 2.0))) * ((x * y) + 2.0)))

Čo sme docielili?

Vstup: (((x * y) + 2.0) * (((x * y) + 2.0) * ((x * y) + 2.0)))

Pôvodný derive podľa x: ((((((1.0 * y) + (0.0 * x)) + 0.0) * (((x * y) + 2.0) * ((x * y) + 2.0))) + ((((((1.0 * y) + (0.0 * x)) + 0.0) * ((x * y) + 2.0)) + ((((1.0 * y) + (0.0 * x)) + 0.0) * ((x * y) + 2.0))) * ((x * y) + 2.0)))

Nový derive podľa x: ((y * (((x * y) + 2.0) * ((x * y) + 2.0))) + (((y * ((x * y) + 2.0)) + (y * ((x * y) + 2.0))) * ((x * y) + 2.0)))

Neriešili sme všeobecný problem zjednodušovania výrazu, iba optimalizáciu niektorých alokácií:

Čo sme docielili?

Vstup: (((x * y) + 2.0) * (((x * y) + 2.0) * ((x * y) + 2.0)))

Pôvodný derive podľa x: ((((((1.0 * y) + (0.0 * x)) + 0.0) * (((x * y) + 2.0) * ((x * y) + 2.0))) + ((((((1.0 * y) + (0.0 * x)) + 0.0) * ((x * y) + 2.0)) + ((((1.0 * y) + (0.0 * x)) + 0.0) * ((x * y) + 2.0))) * ((x * y) + 2.0)))

Nový derive podľa x: ((y * (((x * y) + 2.0) * ((x * y) + 2.0))) + (((y * ((x * y) + 2.0)) + (y * ((x * y) + 2.0))) * ((x * y) + 2.0)))

Neriešili sme všeobecný problem zjednodušovania výrazu, iba optimalizáciu niektorých alokácií:

Malá zmena priniesla zlepšenie:

- zaberáme menej pamäte
- výrazy sú jednoduchsie
- práca s menšími výrazmi bude rýchlejšia

Čo vieme zlepšiť?

```
public class Premenna extends Polynom {

    private final String name;
    public Premenna(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    @Override
    Double valueAt(String[] vars, double[] values) {
        for (int i = 0; i < vars.length; i++) {
            if (vars[i].equals(name)) {
                return values[i];
            }
        }
        return null;
    }
}
```

Čo vieme zlepšiť?

```
public class Premenna extends Polynom {

    private final String name;
    public Premenna(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    @Override
    Double valueAt(String[] vars, double[] values) {
        for (int i = 0; i < vars.length; i++) {
            if (vars[i].equals(name)) {
                return values[i];
            }
        }
        return null;
    }
}
```

Čo vieme zlepšiť?

```
public class Sucin extends Polynom  {

    private final Polynom a;
    private final Polynom b;
    public Sucin(Polynom a, Polynom b) {
        this.a = a;
        this.b = b;
    }


    @Override
    Double valueAt(String[] vars, double[] values) {
        Double valA = a.valueAt(vars, values);
        Double valB = b.valueAt(vars, values);

        if (valA == null || valB == null) {
            return null;
        }

        return valA * valB;
    }
}
```

Čo vieme zlepšiť?

```
public class Sucin extends Polynom  {

    private final Polynom a;
    private final Polynom b;
    public Sucin(Polynom a, Polynom b) {
        this.a = a;
        this.b = b;
    }


    @Override
    Double valueAt(String[] vars, double[] values) {
        Double valA = a.valueAt(vars, values);
        Double valB = b.valueAt(vars, values);

        if (valA == null || valB == null) {
            return null;
        }

        return valA * valB;
    }
}
```

S null musíme použiť if

Čo vieme zlepšiť?

```
public class Premenna extends Polynom {

    private final String name;
    public Premenna(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    @Ov
    Dou
        double valueAt(String[] vars, double[] values) {
            for (int i = 0; i < vars.length; i++) {
                if (vars[i].equals(name)) {
                    return values[i];
                }
            }
            return Double.NaN;
        }
    }
}
```

Čo vieme zlepšiť?

```
public class Premenna extends Polynom {

    private final String name;
    public Premenna(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    @Ov double valueAt(String[] vars, double[] values) {
    Dou     for (int i = 0; i < vars.length; i++) {
                if (vars[i].equals(name)) {
                    return values[i];
                }
            }
            return Double.NaN;
        }
    }
}
```

Na tento účel existuje hodnota NaN

Čo vieme zlepšiť?

```
public class Sucin extends Polynom  {

    private final Polynom a;
    private final Polynom b;
    public Sucin(Polynom a, Polynom b) {
        this.a = a;
        this.b = b;
    }


    @Override
    Double valueAt(String[] vars, double[] values) {
        Double valA = a.valueAt(vars, values);
        Double valB = b.valueAt(vars, values);

        return valA * valB;
    }
}
```

```
double valueAt(String[] vars, double[] values) {
    double valA = a.valueAt(vars, values);
    double valB = b.valueAt(vars, values);

    return valA * valB;
}
```

S NaN neifujeme

Čo vieme zlepšiť?

```
Double.NaN * 5.0f                                   -> NaN
Double.NaN * Double.POSITIVE_INFINITY;              -> NaN
Double.NaN * Double.NEGATIVE_INFINITY;              -> NaN
Double.NaN * Double.NaN;                            -> NaN
Double.NaN * 0.0d;                                  -> NaN
```

Odporúčané čítanie

Joshua Bloch: Effective Java

Happy coding …