Rökstudd forritun

Þegar við erfum frá klasa eða útfærum skil þarf ákveðið samband að gilda milli fyrir- og eftirskilyrða. Hvar sem A er notað á að vera hægt að setja B í staðin.

```
public class A {
   // Fyrir: F_A
   // Eftir: E_A
   public void f() { .. }
}

public class B extends A {
   // Fyrir: F_B
   // Eftir: E_B
   public void f() { .. } // yfirskrifar A.f()
}
```

Rökstudd forritun

```
public class A {
    // Fyrir: F_A
    // Eftir: E_A
    public void f() { .. }
}
```

```
public class B extends A {
    // Fyrir: F_B
    // Eftir: E_B
    public void f() { .. }
}
```

```
A a = new B();
// x uppfyllir F_A
a.f(x); // í raun kallað á B.f(x)
        // x þarf að uppfylla F_B
```

Þurfum: $F_A \Rightarrow F_B$. Fyrir eftirskilyrði er áttin öfug, $E_B \Rightarrow E_A$.

Þetta heitir contravariance, auðvelt að gera villur með því að brjóta þessi skilyrði.

Rökstudd forritun

```
public class Alien {
    private int gender;
    // Fyrir:
    //  gender != b.gender
    // Eftir: new Alien
    public Alien mate(Alien b)
    { .. }
}
```

```
public class WeirdAlien extends
Alien {
    // Fyrir: ekkert
    // Eftir: new WeirdAlien
    public WeirdAlien mate(Alien b)
    { .. }
}
```

Er þetta í lagi?

Fyrir venjulegar geimverur gilda “venjulegar” reglur um kyn.
Fyrir furðulegar geimverur gilda engar reglur um kyn.

- Fyrirskilyrðin eru útvíkkuð í undirklasa
- Ef a,b eru af taginu Alien (eða undirklasa) og a.gender != b.gender þá getum við alltaf kallað á a.mate(b) (líka ef önnur hvor er WeirdAlien)

Skilatagið á mate() er í raun hluti af eftirskilyrðinu.

- Eftirskilyrðin eru þrengd í undirklasa
- WeirdAlien.mate() skilar WeirdAlien hlut, sem er líka Alien hlutur.

vektorar

Fylki hafa alltaf fasta stærð.

Ef við lesum inn gögn af stdin inn í fylki, þá þurfum við að vita fyrirfram stærðina.

Ekki smekkleg lausn. Viljum eitthvað betra.

Vektorar

Það sem við vildum helst

- Láta fylkið vaxa og vera jafnstórt og það þarf að vera
- Geta bætt við staki aftast
- Vísa í öll möguleg stök í fylkinu
- Fengið að vita hversu stórt það er

Einfalt trix, þegar a er of lítið búum við til nýtt fylki og látum a vísa á það.

Vektorar

```
int n=0
int[] a = new int[10];

while(!StdIn.isEmpty()) {
  int x = StdIn.readInt();
  if (n >= a.length) {
     int[] tmp = new int[2*a.length];
     for (int i = 0; i < a.length; i++) {
        tmp[i] = a[i];
     }
     a = tmp;
  }
  a[n] = x;
  n++;
}
```

Vektorar

Sniðug hugmynd, en hún “lekur” upplýsingum

- Við þurfum að passa að vísa aldrei í a[n],…,a [a.length-1]
- Erfitt að endurnýta, þurfum að skrifa of mikinn kóða aftur og aftur

Betra að hafa klasa sem heldur utan um þetta fyrir okkur

```
public class Vector:
        Vector()       // býr til tóman vektor
void    add(Object o) // setur o aftast
int     size()         // skilar stærð
Object  get(int i)    // stak í sæti i
```

Vektorar

Object er sérstakur abstract klasi sem allir klasar erfa frá

Tilvísun á Object hlut getur bent á hvað sem er

```
Object a = new Integer(1);
```

Hefur nokkrar aðferðir

- equals()
- hashCode()
- toString()
- …

Vektorar

Sniðug hugmynd, en hún “lekur” upplýsingum

- Við þurfum að passa að vísa aldrei í a[n],…,a [a.length-1]
- Erfitt að endurnýta, þurfum að skrifa of mikinn kóða aftur og aftur

Betra að hafa klasa sem heldur utan um þetta fyrir okkur

```
public class Vector:
        Vector()        // býr til tóman vektor
void    add(Object o)   // setur o aftast
int     size()          // skilar stærð
Object  get(int i)      // stak í sæti i
```

Sama dæmi og áðan

vektor getur bara geymt hluti, int er ekki hlutur, Integer er hlutur sem geymir int

```
Vector v = new Vector();
while(!StdIn.isEmpty()) {
  v.add(new Integer(StdIn.readInt()));
}
…
int max = Integer.MIN_VALUE;
for (int i = 0; i < v.size(); i++) {
  Integer x = (Integer) v.get(i);
  max = Math.max(max, x.intValue());
}
```

Sama dæmi og áðan

```
Vector v = new Vector();
while(!StdIn.isEmpty()) {
   v.add(new Integer(StdIn.readInt()));
}
…
int max = Integer.MIN_VALUE;
for (int i = 0; i < v.size(); i++) {
   Integer x = (Integer) v.get(i);
   max = Math.max(max, x.intValue());
}
```

get() skilar Object, við þurfum að kasta yfir í Integer

Sama dæmi og áðan

```
Vector v = new Vector();
while(!StdIn.isEmpty()) {
   v.add(new Integer(StdIn.readInt()));
}
…
int max = Integer.MIN_VALUE;
for (int i = 0; i < v.size(); i++) {
   Integer x = (Integer) v.get(i);
   max = Math.max(max, x.intValue());
}
```

x.intValue() skilar int gildinu sem Integer geymir

Vektor útfærsla

Fastayrðing gagna, gríðarlega mikilvæg!

```
public class Vector {
   private Object[] a;
   private int size;
   // a[0],…,a[size-1] eru gildin í vektornum
   // size >= 0, a.length >= size

   public Vector() {
       this.size = 0;
       this.a = new Object[10];
   }

   public int size() {
       return this.size;
   }
}
```

Vektor útfærsla

```
public class Vector {
   private Object[] a;
   private int size;
   // a[0],…,a[size-1] eru gildin í vektornum
   // size >= 0, a.length >= size
   ... // framhald
   public void add(Object o) {
       if (size >= a.length) {
          Object[] tmp = new Object[2*a.length];
          … // afrita frá a yfir í tmp
          a = tmp;
       }
       a[size] = o;
       size++;
   }
}
```

Vektor útfærsla

```
public class Vector {
   private Object[] a;
   private int size;
   // a[0],…,a[size-1] eru gildin í vektornum
   // size >= 0, a.length >= size
   ... // framhald
   public Object get(int i) {
       if (i < 0 || i >= size) {
          throw new ArrayIndexOutOfBoundsException(i);
       }
       return a[i];
   }
}
```

Krassar forritinu með
“hjálplegum” villuskilaboðum

Fylki af hlutum eru upphafsstillt með **`null`**

```
String s = null;
```

Þýðir að s er tilvísun á ekkert (af taginu String).

```
s.substring(0,1);
```

gefur Null Pointer Exception