Fjölbreytni

Í Java getum við notað skil (interface) til að útfæra fjölbreytni og endurnýta kóða

Tvær aðferðir í viðbót

- Erfðir – inheritance (subtyping)
- Sniðmát - generics

Dæmi: sort fallið fyrir hluti

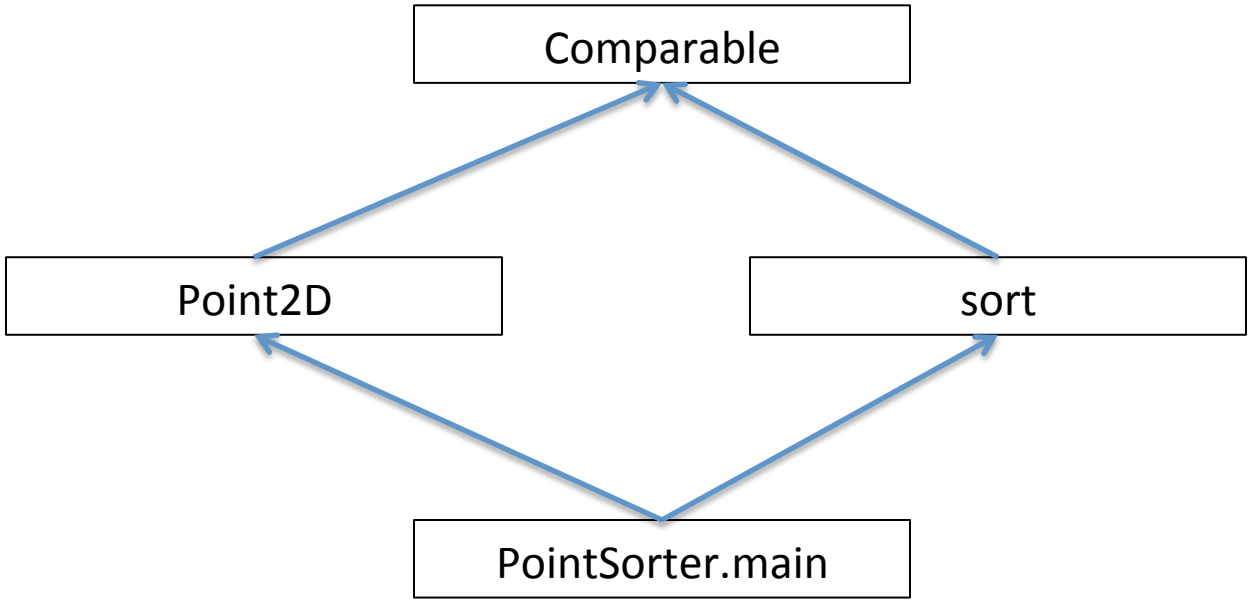
1. Skilgreinum skilin Comparable með aðferðina compareTo í skránni Comparable.java

```
public interface Comparable {
  public int compareTo(Object o);
}
```

a.compareTo(b) Skilar heiltölu < 0 ef a < b, > 0 ef a > b og 0 annars

Hlutbundin forritun og erfðir

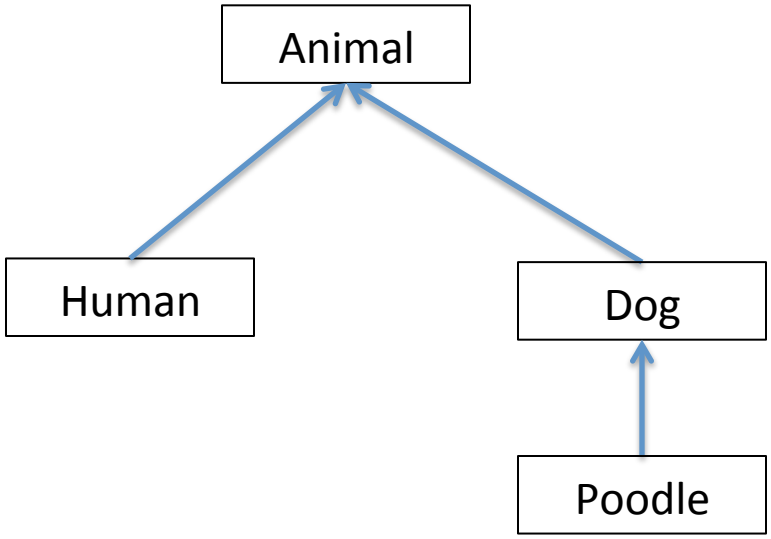
Síðast sáum við sort með skilum (interface)



Comparable skilin tiltaka að klasinn útfærir compareTo fall.

Hlutbundin forritun og erfðir

Erfðir leyfa okkur að taka fram samband milli hluta



Human er Animal, Dog er Animal og Poodle er Dog (og líka Animal)

Erfðir (inheritance) eru stór hluti af Java forritunarmálinu

Erfðir í Java

Notum extends til að erfa frá klasa

```
public class A{
   ...
}

public class B extends A {
   …
}
```

getum bara erft
frá einum klasa

- B hefur allar public aðferðir í A og allar public tilviksbreytur
- B hefur líka allar private tilviksbreytur, en B hefur ekki beinan aðgang að þeim verður að fara í gegnum A

Erfðir í Java

Með erfðum endurnýtum við kóða, ef við breytum kóða í A þá endurspeglast það í B

Við getum notað hluti af taginu B hvar sem er hægt að nota hluti af taginu A

- Ef f(A a) er fall sem tekur hlut af taginu A getum við kallað á f með öllum hlutum af taginu B (f sér ekki muninn!)
- Ef x er tilvísun á hlut af taginu A þá má x vísa á hlut af taginu B

Dæmi

```
Animal a = new Animal(…);
Human b = new Human(…);
Animal c = b; // b er tilvísun á Human og Human er Animal

Human d = b; // b er Human
Human e = c; // þýðingarvilla, c er tilvísun á Animal,
             // jafnvel þóttc vísi í raun og veru á’
              // Human hlut

Human f = (Human) c; // kast á tilvísun, í lagi
Human g = (Human) a; // keyrsluvilla, kóðinn þýðist en a er
                     // ekki af taginu Human
```

Hvenær viljum við nota erfðir?

- Sjaldan
- Aðeins þegar við getum sagt
  “B er í raun og veru A”
- Á sjaldnar við en við höldum
  - oft einfaldara að geyma tilvísun á hlut A
  - interface tekur fram hvað hlutur getur gert,
    ekki hvernig hann lítur út
  - erfðir taka fram hvernig hlutur lítur út
    og hvað hann getur gert

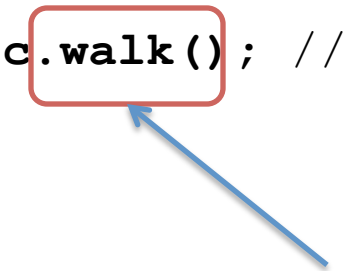
Dæmi

```
public class Animal {
   private String name;
   public Animal(String nm) {
        this.name = nm;
   }

   public void walk() {
        System.out.println(this.name+" walking");
   }
}

public class Human extends Animal {
   public Human(String nm) { super(nm);}
   public void think() {
        System.out.println("thinking…");
   }
}
```

Menn geta hugsað, öll dýr geta labbað þar af leiðir að allir menn geta labbað

Dæmi

```
Animal a = new Animal("Mouse");
Human b = new Human("John");
Animal c = b;

a.walk(); // prentar "Mouse walking"
b.walk(); // prentar "John walking"
c.walk(); // prentar "John walking" !
```



Hluturinn sem c bendir á fær boð um að kalla á walk() aðferðina, þessi hlutur er í raun af taginu Human og því er kallað á walk() fyrir Human hlutinn

```
b.think(); // printar "thinking …"
c.think(); // þýðingarvilla, c er ekki tilvísun á Human
((Human) c).think(); // kast í lagi, prentar "thinking"
((Human) a).think(); // villa á keyrslutíma
```

Dæmi

```
public class Dog extends Animal {
    private String sound;
    public Dog(String s) {
        super("Dog");
        this.sound = s;
    }
    public void bark() {
        System.out.println(
                this.sound);
    }
}
```

```
public class Poodle extends Dog {
    public Poodle() {
        super("bjeff!")
    }

    public void bark() {
        super.bark();
        System.out.println("grr");
        super.bark();
    }

    public void sitInLap() {
        …
    }
}
```

super("Dog") kallar á smiðinn fyrir yfirklasann af Dog, þ.e. Animal smiðinn sem upphafsstillir name breytuna

super.bark() kallar á bark aðferðina sem er skilgreind í Dog, this.bark() eða bark() vísar til aðferðarinnar í Poodle

Super

Ef B er undirklasi fyrir A (`class B extends A`)

- super er tilvísun á hlutinn (þ.e. `this`) sem tilvik af klasanum A
- super.f() kallar á aðferðina f í klasanum A
  - ef f er ekki yfirskrifuð (override) í B þá getum við alveg eins notað f() eða this.f()
  - ef f er yfirskrifuð (override) í B þá getum við bara kallað á upprunalegu f með super.f()
- super() eða super(…) kallar á smiðinn fyrir A
  - Fyrsta línan í smiðnum á B verður að kalla á smiðinn fyrir A
  - Nema ef A hefur sjálfgefinn smið, A(), sem tekur engin viðföng, þá kallar Java á A() fyrir okkur ef við gerum það ekki
- Það er víst ekki hægt að kalla á super.super :(

Dæmi

```
Dog d1 = new Dog("voff");
Poodle d2 = new Poodle();
Dog d3 = d2;

d1.bark(); // prentar "voff"
d2.bark(); // prentar "bjeff grr bjeff"
d2.sitInLap(); // í lagi
d3.sitInLap(); // þýðingarvilla

d3.bark(); // prentar "bjeff grr bjeff"
```

Aðferðin sem er valin fer eftir taginu á **hlutnum** sem d3 bendir á ekki taginu á tilvísuninni

Abstract

Í raun ætti ekki að vera hægt að búa til Animal hlut

- Human, Dog og Poodle eru Animal, en við viljum ekki að það sé hægt að búa til hlut af taginu Animal
- Þetta er gert í Java með **`abstract class`**

```
public abstract class Animal {
  private String name;
  public Animal(String nm) {
     this.name = nm;
  }

  public abstract void walk();
}
```

Engin {} útfærsla, bara ;
Undirklasar verða að útfæra walk() ekki hægt að gera
Animal a = new Animal("..");

Þá þarf Human að útfæra walk() aðferð

```
public class Human extends Animal {
  public void think() {
      System.out.println("thinking…");
  }

  public void walk() {
      System.out.println("walking…");
  }
}
```

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.

Dæmi

3. (7 stig) Í klösunum A og B er aðferðin f yfirskrifuð og x, y eru jákvæðar heiltölur. Hvaða skilyrði verða að gilda um x og y til þess að rökfræðilegt samband klasanna A og B sé rétt?
The method f is overwritten in classes A and B, and x, y are positive integers. What relationship must hold between x and y in order for the logical relationship between A and B is correct?

```
public class A {
    ...
    // Notkun: t =  a.f(n)
    // Fyrir:  n er margfeldi af 24
    // Eftir:  t er margfeldi af 120
    // Use:  t = a.f(n)
    // Pre:  n is a multiple of 24
    // Post: t is a multiple of 120
    public int f(int n) { ... }
}
```

```
public class B extends A {
    ...
    // Notkun: t =  b.f(n)
    // Fyrir:  n er margfeldi af x
    // Eftir:  t er margfeldi af y
    // Use:  t = a.f(n)
    // Pre:  n is a multiple of x
    // Post: t is a multiple of y
    public int f(int n) { ... }
}
```

Lausn: $F_A \Rightarrow F_B$ leiðir til þess að x gengur upp í 24
$E_B \Rightarrow E_A$ leiðir til þess að y er margfeldi af 120

Verkefni

Spurningin um Square klasann sem erfir frá Rectangle klasanum er af þessu tagi

Eru allir ferningar rétthyrningar?

Getum við notað Square þar sem við tökum við Rectangle?

Ath. Fastayrðing gagna verður alltaf að vera sönn!