Arten von Klassen-Beziehungen

Untertypbeziehung: Ersetzbarkeit

Vererbung von Code aus Oberklasse irrelevant

Vererbungsbeziehung: Klasse entsteht durch Abänderung anderer Klassen

Ersetzbarkeit irrelevant

Reale-Welt-Beziehung: Beziehung zw. Einheiten im Entwurf

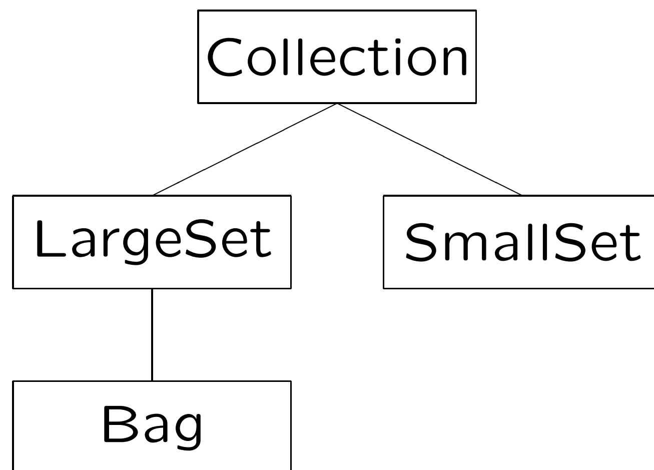
intuitiv klar, ohne Details zu kennen

oft zu Untertypbeziehung weiterentwickelbar

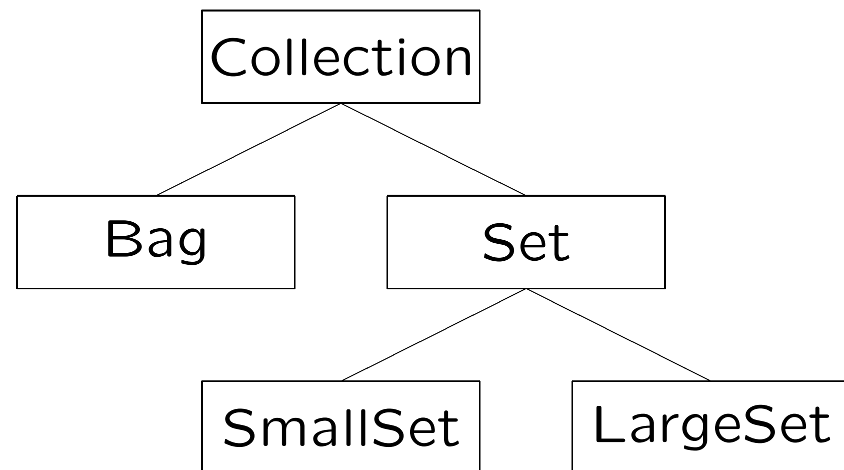
Untertypen versus Vererbung in Java

- Untertypbeziehungen setzen Vererbung voraus
- Vererbung setzt Untertypbeziehung (soweit diese statisch prüfbar ist) voraus
- daher Untertyp- und Vererbungsbeziehungen nur durch (nicht statisch prüfbare) Zusicherungen unterscheidbar
- trotzdem oft einfach erkennbar, was angestrebt wird:
  - Vererbung: Ähnlichkeiten im Code ausgenützt
  - Untertypen: ersetzbares Verhalten beschrieben

Beispiel: Untertypen versus Vererbung



reine Vererbungsbeziehung



Untertypbeziehung

Untertypen versus Vererbung (Tip)

- Vererbung = direkte Codewiederverwendung (sichtbar)
- Untertypbeziehung = indirekte Codewiederverwendung (manchmal nicht gleich sichtbar)
- Untertypen bedeuten oft weniger direkte Codewiederverwendung als Vererbung, da Zusicherungen berücksichtigt
- indirekte Codewiederverwendung langfristig viel wichtiger als direkte (lokale Programmänderungen möglich)
- Tip: immer auf indirekte Wiederverwendung achten, direkte Wiederverwendung nur ohne großen Aufwand

  $\Rightarrow$ jede Vererbungsbeziehung ist Untertypbeziehung

Direkte Codewiederverwendung

- direkte Codewiederverwendung durch Vererbung ist auch wichtig (solange Untertypbeziehung nicht verletzt):
  - Code nur einmal geschrieben
  - Änderungen nur an einer Stelle
- durch Möglichkeit des Überschreibens keine Nachteile

Vererbung, 1. Beispiel

```
class A {
    public void foo() { ... }
}

class B extends A {
    private boolean b;
    public void foo() {
        if (b) { ... }
        else { super.foo(); }
    }
    ...
}
```

Vererbung, 2. Beispiel

```
class A {
    public void foo() {
        if (...) { ... }
        else { ...; x = 1; ... }
    }
}

class B extends A {
    public void foo() {
        if (...) { ... }
        else { ...; x = 2; ... }
    }
}
```

Vererbung, 3. Beispiel

```
class A {
    public void foo() {
        if (...) { ... }
        else { fooX(); }
    }
    protected void fooX() { ...; x = 1; ... }
}

class B extends A {
    protected void fooX() { ...; x = 2; ... }
}
```

Vererbung, 4. Beispiel

```
class A {
    public void foo() { fooY(1); }
    protected void fooY (int y) {
        if (...) { ... }
        else { ...; x = y; ... }
    }
}

class B extends A {
    public void foo() { fooY(2); }
}
```

Variablen in Java

- Instanzvariablen (über Instanz zugreifbar):

```
int x = 2, y = 1, z;
int x[] = new int[32], y[], z;
int[] x = new int[32], y = null;
```

- Klassenvariablen (über Klasse zugreifbar):

```
static int maxRadius = 1023;
```

- Konstanten (nicht schreibbar):

```
static final int MAX_SIZE = 1024;
final int sizeOfThis = MAX_SIZE - n;
```

Statische Methoden in Java

- Instanzmethoden (über Instanz zugreifbar):

```
void foo (String[] args) { ... }
```

  dynamisches Binden

- statische Methoden (über Klasse zugreifbar):

```
static void main (String[] args) { ... }
```

  kein dynamisches Binden

- static initializer (zur Initialisierung von Klassenvariablen):

```
static { ... }
```

  Code nur einmal ausgeführt

Statische geschachtelte Klassen

- gehören zu umschließender Klasse selbst

```
class EnclosingClass {
    ...
    static class StaticNestedClass { ... }
    ...
}
```

- nur Klassenvariablen und statische Methoden umschließender Klasse zugreifbar

- Erzeugung: `new EnclosingClass.StaticNestedClass()`

Innere Klassen

- gehören zu Instanzen umschließender Klasse

```
class EnclosingClass {
    ...
    class InnerClass { ... }
    ...
}
```

- nur Instanzvariablen und Instanzmethoden umschließender Klasse direkt zugreifbar

- Erzeugung: `a.new InnerClass()`
  (wobei `a` eine Instanz von `EnclosingClass` ist)