Programmierkurs

Vorlesung 07

M.Sc. Laslo Hunhold

Department Mathematik/Informatik
Abteilung Informatik
Universität zu Köln

21. Dezember 2020

Letzte Vorlesung

- Methoden (Deklaration, Aufruf, `return`, `static`)
- Überladung
- Konstruktoren (Default-Konstruktor, Deklaration)

Schlüsselwort `this`

```
 1  class Test {
 2          int wert;
 3
 4          Test(int wert) {
 5                  this.wert = wert; /* wert = wert nicht eindeutig */
 6          }
 7
 8          Test() {
 9                  this(0); /* Test(0) geht wiederum nicht */
10          }
11  }
```

- `this` ist die Instanz, auf der man momentan operiert (also nur in Instanzmethoden und Konstruktoren sinnvoll und nutzbar)
- Erlaubt Unterscheidung zwischen lokalen Variablen und Variablen und Methoden der Instanz
- Erlaubt Aufruf eines Konstruktors in einem Konstruktor (nur in der ersten Zeile des Konstruktors!)

Zugriffsmodifikatoren

Motivation

```
class Bruch {
        int zaehler;
        int nenner;

        Bruch() {
                setze(0, 1);
        }

        void setze(int z, int n) {
                if (nenner == 0) {
                        /* Fehler */
                        return;
                }
                this.zaehler = z;
                this.nenner = n;
        }

        int gebe_zaehler() {
                return zaehler;
        }

        int gebe_nenner() {
                return nenner;
        }
}
```

```
Bruch b = new Bruch();
b.zaehler = 0;
b.nenner = 0;
```

- Direktzugriff auf `zaehler` und `nenner` soll nicht möglich sein, sondern nur mit den `setze()` und `gebe()`-Methoden
- Wo soll die Klasse `Bruch` sichtbar sein (Package, Programm, ...)?
- Standard: Sichtbarkeit im Package; Methoden und Variablen beliebig überschreibbar

Veränderungsbeschränkung `final`

Anwendbarkeit und Effekte auf eine ...

- Klasse: Anwendbar; verhindert Bildung von „Subklassen“ (später)
- Methode: Anwendbar; verhindert „Überschreibung“ in „Subklassen“ (später)
- Klassenvariable: Anwendbar; Wert muß sofort gesetzt werden, darf danach nicht mehr geändert werden („Konstante“)
- Instanzvariable: Anwendbar; Wert muß sofort oder im Konstruktor gesetzt werden, darf danach nicht mehr geändert werden
- Lokale Variable: Anwendbar; Wert muß sofort gesetzt werden oder später (aufgeschobene Initialisierung), darf danach nicht mehr geändert werden

```
final class Kreis {
        final static double PI = 3.141592653589793;
        final double radius;

        Kreis(double radius) {
                this.radius = radius;
        }
}
```

Zugriffsbeschränkung `private`

Setzt Zugriffsbereich auf die Klasse (Kein Zugriff von außen).

Anwendbarkeit und Effekte auf eine ...

- Klasse: Nicht anwendbar; sinnlos, da Klasse dann nicht mehr erreichbar
- Methode: Anwendbar
- Klassenvariable: Anwendbar
- Instanzvariable: Anwendbar
- Lokale Variable: Nicht anwendbar; sinnlos, da Gültigkeitsbereich sowieso nur in der deklarierenden Methode

Wie greift man auf `private` Instanz- und Klassenvariablen zu?

- Setze und lese Werte mit „Getter“- und „Setter“-Methoden
- Vorteil: Erlaubt Prüfung und Reinigung der Parameter
- Automatische Generierung in Eclipse („Source“ → „Generate Getters and Setters...“)

Zugriffserweiterung `public`

Setzt Zugriffsbereich auf das gesamte Programm (Zugriff von überall).

Anwendbarkeit und Effekte auf eine ...

- Klasse: Anwendbar
- Methode: Anwendbar
- Klassenvariable: Anwendbar
- Instanzvariable: Anwendbar
- Lokale Variable: Nicht anwendbar; sinnlos, da Gültigkeitsbereich sowieso nur in der deklarierenden Methode

```
public class Test {
        public int wert;

        public void ausgabe() {
                System.out.println(wert);
        }
}
```

Zugriffserweiterung `protected`

Setzt Zugriffsbereich auf das Package (wie Standard) und „Subklassen“ der Klasse in *anderen* Packages (später).

Anwendbarkeit und Effekte auf eine ...

- Klasse: Nicht anwendbar; sinnlos, da „Subklassen“ die Klasse sowieso benötigen
- Methode: Anwendbar
- Klassenvariable: Anwendbar
- Instanzvariable: Anwendbar
- Lokale Variable: Nicht anwendbar; sinnlos, da Gültigkeitsbereich sowieso nur in der deklarierenden Methode

Beispiel

```
class Bruch {
        private int zaehler;
        private int nenner;

        public Bruch() {
                setze(0, 1);
        }

        public void setze(int zaehler,
     int nenner) {
                if (nenner == 0) {
                        /*
     Fehlermeldung */
                        return;
                }
                this.zaehler = zaehler;
                this.nenner = nenner;
        }
        public int gebe_zaehler() {
                return zaehler;
        }
        public int gebe_nenner() {
                return nenner;
        }
}
```

```
Bruch b = new Bruch();
b.zaehler = b.nenner = 1; /*
     Fehler */
b.setze(1, 1);
```

Vererbung

Motivation

- Online-Shop für Bücher und Schallplatten, Produkte als Instanzen der Klassen

```
class Buch {
        int artnr;
        int bestand;

        String name;
        String autor;
        int seiten;

        void info() { ... }
}
```

```
class Schallplatte {
        int artnr;
        int bestand;

        String titel;
        String interpret;

        void info() { ... }
}
```

- Jedes Produkt hat Gemeinsamkeiten, aber auch Unterschiede
- Können wir die Klassen `Buch` und `Schallplatte` von einer anderen Klasse (z.B. `Produkt`) ableiten?

Schlüsselwort extends

```
1  class Produkt {
2          int artnr;
3          int bestand;
4
5          void info() { ... }
6  }
7
8  class Buch extends Produkt {
9          String name;
10         String autor;
11         int seiten;
12 }
13
14 class Schallplatte extends Produkt {
15         String titel;
16         String interpret;
17 }
```

- Die Klassen `Buch` und `Schallplatte` „erben“ alle Variablen und Methoden von `Produkt`, so als hätten wir sie ihnen explizit hinzugefügt
- `Buch` und `Schallplatte` fügen aber noch weitere Variablen und Methoden hinzu
- `Produkt` ist die „Superklasse“/„Oberklasse“, `Buch` und `Schallplatte` sind „Subklassen“/„Unterklassen“

Bemerkungen

- Methoden können „überschrieben“ werden (sofern nicht `final` in der Superklasse)
- Die überschriebene Methode wird in der Subklasse ausgeführt; die Superklasse bleibt unberührt
- Variablen/Methoden der Superklasse sind bei Subklassen in anderen Packages nur sichtbar, wenn sie `protected` sind

Schlüsselwort super

- super ist die Superklasse der aktuellen Klasse
- Erlaubt Zugriff auf überschriebene Methoden und Konstruktoraufruf der Superklasse

```
1  class Rechteck {
2          double a;
3          double b;
4
5          Rechteck(double a, double b) {
6                  this.a = a;
7                  this.b = b;
8          }
9
10         void info() {
11                 System.out.println("Rechteck " + a + "x" + b + "cm");
12         }
13 }
14
15 class Quadrat extends Rechteck {
16         Quadrat(double a) {
17                 super(a, a);
18         }
19
20         void info() {
21                 super.info();
22                 System.out.println("Quadrat mit " + a + "cm-Kanten");
23         }
24 }
```

Vorteile der Vererbung

- Verhindert doppelten Quelltext
- Verringert Fehleranfälligkeit und erhöht Flexibilität
- Instanzen von `Buch` und `Schallplatte` sind auch vom Typ `Produkt` (Prinzip der „Kompatibilität“, später)

Kompatibilität/Polymorphismus

Statischer und Dynamischer Typ

- Beobachtung: Java erlaubt Folgendes

```
Produkt a = new Buch();
```

a ist nur ein Zeiger; funktioniert, da die Klasse `Buch` als Erweiterung alle Felder von `Produkt` hat.

```
Buch b = new Produkt();
```

geht nicht, da die `Produkt`-Instanz z.B. nicht die `Buch`-Variable `autor` hat; `b.autor` wäre ungültig

- a hat den „statischen Typ“ `Produkt` (verändert sich nicht); der statische Typ ist Basis für Variablen- und Methodenzugriff
- a hat den „dynamischen Typ“ `Buch` (kann sich ändern; hängt vom Objekt ab, auf das gezeigt wird)
- a ist „polymorph“, weil es auf Objekte verschiedenen Typs zeigen kann
- Wie prüfen wir, welchen dynamischen Typ a hat, wenn wir es nicht wissen?
- Wie greifen wir auf die `Buch`-Variablen und -Methoden in a zu?

`instanceof` und explizite Typumwandlung

- Operator `instanceof`
- Zwei Operanden: Ein Zeiger und eine Klasse
- Gibt `true` zurück, falls der dynamische Typ des Zeigers der Klasse entspricht, sonst `false`
- Bedingte Typumwandlung für vollständigen Variablenzugriff (der statische Typ wird quasi dem dynamischen angepaßt)

```
1  Produkt a = new Buch();
2
3  if (a instanceof Buch) {
4          Buch b = (Buch)a;
5          ...
6  } else if (a instanceof Schallplatte) {
7          Schallplatte s = (Schallplatte)a;
8          ...
9  } else {
10         ...
11 }
```

Abstrakte Klassen und Methoden

Motivation

Manche Methoden sind *nur* in Subklassen, nicht in der Superklasse, sinnvoll

```
1  class Produkt {
2          int artnr;
3          int bestand;
4
5          void info() { /* hier kann nichts Sinnvolles hinein */ }
6  }
7
8  class Buch extends Produkt {
9          String name;
10         String autor;
11         int seiten;
12
13         void info() {
14                 System.out.println("Buch " + name + "von " + autor);
15         }
16 }
17
18 class Schallplatte extends Produkt {
19         String titel;
20         String interpret;
21
22         void info() {
23                 System.out.println("Schallplatte " + titel + "von " + interpret);
24         }
25 }
```

Deklaration - Schlüsselwort `abstract`

- Klassendeklaration bekommt ein `abstract`, genau wie jede „abstrakte“ Methode, die nur als Methodenkopf mit Semikolon gegeben wird
- Superklasse ist damit also nur ein Muster
- `abstract` „erzwingt“ die Deklaration der abstrakten Methoden

```
1  abstract class Produkt {
2          int artnr;
3          int bestand;
4
5          abstract void info();
6  }
7
8  class Buch extends Produkt {
9          String name;
10         String autor;
11         int seiten;
12
13         void info() {
14                 System.out.println("Buch " + name + "von " + autor);
15         }
16 }
17
18 ...
```

Bemerkungen

- Man darf (polymorphe) Zeiger einer abstrakten Klasse benutzen
- Instanzerzeugung einer abstrakten Klasse geht nicht (da die Klasse unvollständig ist)
- Erst die Subklassen sind vollständige Klassen

Organisatorisches

Veranstaltungsübergabe

- Elternzeitvertretung für Frau Dr. Vera Weil endet zum 31. Dezember
- Vorstellungsvideo auf der Veranstaltungswebseite
- Vorlesungen und Übungen weiterhin wie gewohnt von mir
- Inhaltliche Rückfragen ab dem 1. Januar bitte an Frau Dr. Weil an die Adresse weil@cs.uni-koeln.de (Webseite `https://weil.cs.uni-koeln.de/`, Kontakthinweise beachten)
- Komplette Übergabe auf ihre Veranstaltungswebseite (`https://weil.cs.uni-koeln.de/lehre/programmierkurs/pk_start.php`) zum Vorlesungsende im Februar, bis dahin Verbleib auf der bisherigen Webseite
- Klausur wird von Frau Dr. Weil gestellt (inhaltlich eng abgestimmt, Sonderübung im Februar)

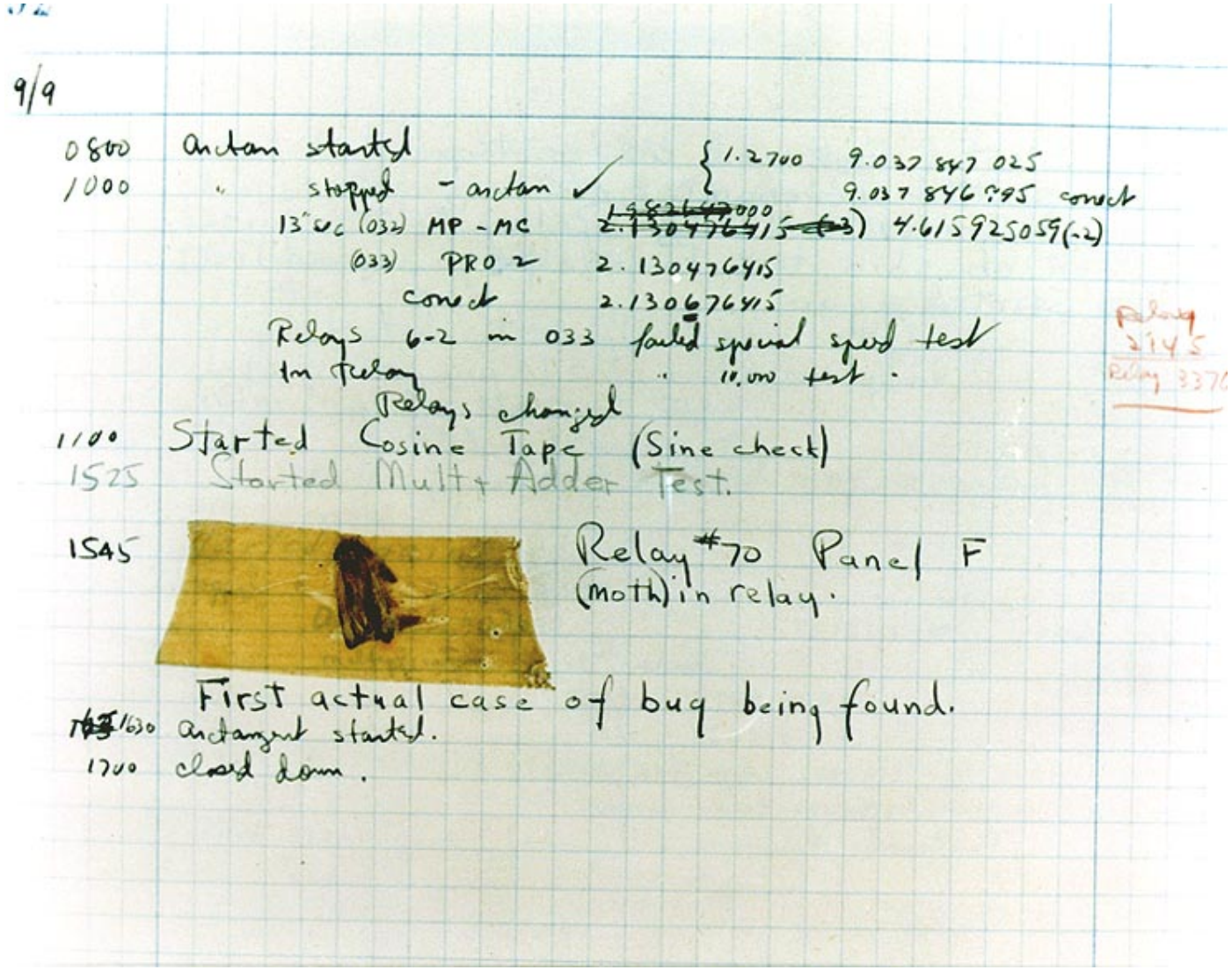

Abbildung: Erster Computer-„Bug“ (Logbuch des „Mark II Aiken Relay Calculator“, Harvard-Universität, 9. September 1947) (Quelle: U.S. Naval Historical Center Online Library, NH 96566-KN)