hhu.

Kapitel 6: Vererbung & Fehlerbehandlung

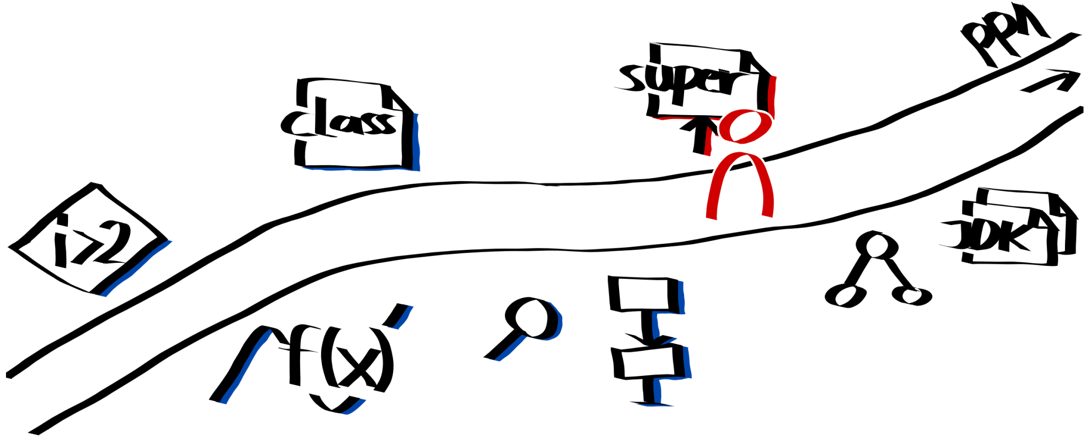
VL 21: Finale Klassen, Überladen, Hiding

Wintersemester 2023/24

Stand 19. Dezember 2023, 09:27 Uhr

Wo stehen wir gerade?

Heinrich Heine Universität Düsseldorf



Vorlesungsfeedback

equals

- Standardverhalten von `equals` und `toString` meist nutzlos ⇒ im Regelfall überschreiben
  - ! Parameter von `equals` vom Typ `Object`
  - später: automatische Generierung des Codes

```java
public class Person {
  private String name;

  public Person(String name) {
    this.name = name;
  }

  @Override
  public boolean equals(Object other) {
    if(this == other)
      return true;
    if(other == null || getClass() != other.getClass())
      return false;
    // Objektinhalte vergleichen
    return ((Person) other).name.equals(this.name);
  }

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

Sieht das sinnvoll aus?


```java
Ball b1 = new Ball(10);
boolean reflexiv = b1.equals(b1); // false
```

Sieht das sinnvoll aus?

Heinrich Heine
Universität Düsseldorf

```java
Ball b1 = new Ball(10);
boolean reflexiv = b1.equals(b1); // false
```

- `equals` soll Äquivalenzrelation bilden:
  - `a.equals(a)` (reflexiv)
  - `a.equals(b)` $\Leftrightarrow$ `b.equals(a)` (symmetrisch)
  - `a.equals(b)` $\land$ `b.equals(c)` $\Rightarrow$ `a.equals(c)` (transitiv)

Überladen durch Vererbung

Heinrich Heine Universität Düsseldorf

```java
public class A {
  public void print(int i) {
    System.out.println("int " + i);
  }
}
```

```java
public class B extends A {
  public void print(String s) {
    System.out.println("String " + s);
  }
}
```

```java
B object = new B();
object.print(5);
```

Überladen durch Vererbung

hhu
Heinrich Heine
Universität Düsseldorf

```java
public class A {
  public void print(int i) {
    System.out.println("int " + i);
  }
}
```

```java
public class B extends A {
  public void print(String s) {
    System.out.println("String " + s);
  }
}
```

```java
B object = new B();
object.print(5);
```

```
int 5
```

Was passiert hier? I

Heinrich Heine Universität Düsseldorf

```java
public class A {
}
```

```java
public class B extends A {
}
```

```java
public class C {

}
```

```java
public class D extends C {
  public void print(A object) {
    System.out.println("print(A)");
  }
}
```

```java
D d = new D();
B object1 = new B();
A object2 = new B();
d.print(object1);
d.print(object2);
```

Was passiert hier? I

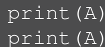
```java
public class A {
}
```

```java
public class B extends A {
}
```

```java
public class C {

}
```

```java
public class D extends C {
  public void print(A object) {
    System.out.println("print(A)");
  }
}
```

```java
D d = new D();
B object1 = new B();
A object2 = new B();
d.print(object1);
d.print(object2);
```



Was passiert hier? II

```java
public class A {
}
```

```java
public class B extends A {
}
```

```java
public class C {
  public void print(A object) {
    System.out.println("A: print(A)");
  }
}
```

```java
public class D extends C {
  @Override
  public void print(A object) {
    System.out.println("D: print(A)");
  }
}
```

```java
D d = new D();
B object1 = new B();
A object2 = new B();
d.print(object1);
d.print(object2);
```

Was passiert hier? II

```java
public class A {
}
```

```java
public class B extends A {
}
```

```java
public class C {
  public void print(A object) {
    System.out.println("A: print(A)");
  }
}
```

```java
public class D extends C {
  @Override
  public void print(A object) {
    System.out.println("D: print(A)");
  }
}
```

```java
D d = new D();
B object1 = new B();
A object2 = new B();
d.print(object1);
d.print(object2);
```

Was passiert hier? III

Heinrich Heine Universität Düsseldorf

```java
public class A {
}
```

```java
public class B extends A {
}
```

```java
public class C {
  public void print(A object) {
    System.out.println("A: print(A)");
  }
}
```

```java
public class D extends C {
  @Override
  public void print(A object) {
    System.out.println("D: print(A)");
  }

  public void print(B object) {
    System.out.println("D: print(B)");
  }
}
```

```java
D d = new D();
B object1 = new B();
A object2 = new B();
d.print(object1);
d.print(object2);
```

Was passiert hier? III

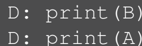
```java
public class A {
}
```

```java
public class B extends A {
}
```

```java
public class C {
  public void print(A object) {
    System.out.println("A: print(A)");
  }
}
```

```java
public class D extends C {
  @Override
  public void print(A object) {
    System.out.println("D: print(A)");
  }

  public void print(B object) {
    System.out.println("D: print(B)");
  }
}
```

```java
D d = new D();
B object1 = new B();
A object2 = new B();
d.print(object1);
d.print(object2);
```



```
D: print(B)
D: print(A)
```

Was passiert hier? IV

Heinrich Heine Universität Düsseldorf

```java
public class A {
}
```

```java
public class B extends A {
}
```

```java
public class C {
  public void print(B object) {
    System.out.println("print(B)");
  }
}
```

```java
public class D extends C {
  public void print(A object) {
    System.out.println("print(A)");
  }
}
```

```java
D d = new D();
B object1 = new B();
A object2 = new B();
d.print(object1);
d.print(object2);
```

Was passiert hier? IV

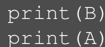
```java
public class A {
}
```

```java
public class B extends A {
}
```

```java
public class C {
  public void print(B object) {
    System.out.println("print(B)");
  }
}
```

```java
public class D extends C {
  public void print(A object) {
    System.out.println("print(A)");
  }
}
```

```java
D d = new D();
B object1 = new B();
A object2 = new B();
d.print(object1);
d.print(object2);
```



Lösung


```
print(B)
print(A)
```

- hier wird nichts überschreiben, da Signaturen nicht identisch
- passendste, überladene Methode basierend auf **deklariertem** Variablentyp gewählt
- verwirrend, kommt in der Praxis aber nicht regelmäßig vor

Verhaltensdefinition

Java Language Specification §8.4.9 Overloading[1]

"When a method is invoked (§15.12), the number of actual arguments [. . .] and the compile-time types of the arguments are used, at compile time, to determine the signature of the method that will be invoked (§15.12.2). If the method that is to be invoked is an instance method, the actual method to be invoked will be determined at run time, using dynamic method lookup (§15.12.4)."

- ! Laufzeitpolymorphie von Instanzmethoden ist in Java der Sonderfall, nicht der Standard für alles
- ! Instanzmethoden: tatsächlicher Typ (*actual type*) zur Laufzeit entscheidet, der Code welcher Klasse ausgeführt wird
- ! Überladen wird zur Compilezeit anhand der deklarierten Typen (*declared type*) entschieden

[1] https://docs.oracle.com/javase/specs/jls/se21/html/jls-8.html#jls-8.4.9

Was passiert hier?

Heinrich Heine Universität Düsseldorf

```java
public class A {
    public final int value = 1;

    public void print() {
        System.out.println("A");
    }
}
```

```java
public class B extends A {
    public final int value = 2;

    @Override
    public void print() {
        System.out.println("B");
    }
}
```

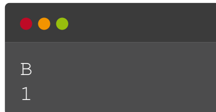
```java
A object = new B();
object.print();
System.out.println(object.value);
```

Was passiert hier?

```java
public class A {
    public final int value = 1;

    public void print() {
        System.out.println("A");
    }
}
```

```java
public class B extends A {
    public final int value = 2;

    @Override
    public void print() {
        System.out.println("B");
    }
}
```

```java
A object = new B();
object.print();
System.out.println(object.value);
```



Hiding

```
B
1
```

⇒ Nur Instanz**methoden** können überschrieben werden.

- Effekt oben: Verdeckung[2]
- Verwirrend, ergibt sich mit privaten Instanzvariablen aber erst gar nicht

[2] Hiding

Was passiert hier?

hhu
Heinrich Heine
Universität Düsseldorf

```java
public class A {
    public static final int value = 1;

    public static void print() {
        System.out.println("A");
    }
}
```

```java
public class B extends A {
    public static final int value = 2;

    public static void print() {
        System.out.println("B");
    }
}
```

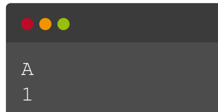
```java
A object = new B();
object.print();
System.out.println(object.value);
```

Was passiert hier?

```java
public class A {
    public static final int value = 1;

    public static void print() {
        System.out.println("A");
    }
}
```

```java
public class B extends A {
    public static final int value = 2;

    public static void print() {
        System.out.println("B");
    }
}
```

```java
A object = new B();
object.print();
System.out.println(object.value);
```



Hiding II

```
A
1
```

⇒ Nur **Instanz**methoden können überschrieben werden.

Hiding II


```
A
1
```

⇒ Nur **Instanz**methoden können überschrieben werden.

- Verwirrend, ergibt sich erst gar nicht, wenn auf statische Methoden/Variablen immer per Klasse zugegriffen:

```java
A.print();
System.out.println(A.value);
```

```
A
1
```

Begriffsübersicht

Heinrich Heine Universität Düsseldorf

- **Überschreiben** (Overriding): Ändern des Verhaltens einer Instanzmethode der Oberklasse (gleiche Signatur in Ober- und Unterklasse)
  - ! klappt nur mit Instanzmethoden
- **Verdeckung** (Hiding): Definition einer Instanzvariable, Klassen-Methode oder Klassen-Variable, die es genau so in der Oberklasse gibt
  - ! Hier findet (in Java) kein Überschreiben statt! (keine Laufzeit-Polymorphie)
  - ⇒ verwirrend, nicht benutzen
- **Überladen** (Overloading): Methode mit gleichem Namen wie andere Methode derselben Klasse, aber unterschiedlichen Parametern
  - funktioniert auch mit vererbten Methoden, kann daher verwirrend werden
- **Shadowing**: z. B. Parameter, der wie eine Instanzvariable heißt
  - oft in Konstruktoren benutzt, in diesen Fällen mit `this.` umgehbar
- **Obscuring**: z. B. Variable mit Namen `System` ⇒ „normales“ `System` nicht mehr verfügbar
  - nicht machen (bitte)

finale Instanzvariablen

Heinrich Heine Universität Düsseldorf

```java
public class Kasse {

    private final int[] einzelkosten;

    public Kasse(int[] einzelkosten) {
        this.einzelkosten = einzelkosten;
    }

    public int summe() {
        int ergebnis = 0;
        for(int betrag: einzelkosten) {
            ergebnis += betrag;
        }
        return ergebnis;
    }
```

```java
int[] kosten = {999, 129, 349};
Kasse kasse = new Kasse(kosten);
System.out.println(kasse.summe());
```

finale Instanzvariablen

```java
public class Kasse {

    private final int[] einzelkosten;

    public Kasse(int[] einzelkosten) {
        this.einzelkosten = einzelkosten;
    }

    public int summe() {
        int ergebnis = 0;
        for(int betrag: einzelkosten) {
            ergebnis += betrag;
        }
        return ergebnis;
    }
```

```java
int[] kosten = {999, 129, 349};
Kasse kasse = new Kasse(kosten);
System.out.println(kasse.summe());
```

```
1477
```

Klappt das?

Heinrich Heine Universität Düsseldorf

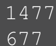
```java
public class Kasse {

    private final int[] einzelkosten;

    public Kasse(int[] einzelkosten) {
        this.einzelkosten = einzelkosten;
    }

    public int summe() {
        int ergebnis = 0;
        for(int betrag: einzelkosten) {
            ergebnis += betrag;
        }
        return ergebnis;
    }

    public void kostenKorrigieren(int index,
    ↪   int betrag) {
        this.einzelkosten[index] = betrag;
    }
```

```java
int[] kosten = {999, 129, 349};
Kasse kasse = new Kasse(kosten);
System.out.println(kasse.summe());

kasse.kostenKorrigieren(0, 199);
System.out.println(kasse.summe());
```

Klappt das?

```java
public class Kasse {

    private final int[] einzelkosten;

    public Kasse(int[] einzelkosten) {
        this.einzelkosten = einzelkosten;
    }

    public int summe() {
        int ergebnis = 0;
        for(int betrag: einzelkosten) {
            ergebnis += betrag;
        }
        return ergebnis;
    }

    public void kostenKorrigieren(int index,
    ↪   int betrag) {
        this.einzelkosten[index] = betrag;
    }
```

```java
int[] kosten = {999, 129, 349};
Kasse kasse = new Kasse(kosten);
System.out.println(kasse.summe());

kasse.kostenKorrigieren(0, 199);
System.out.println(kasse.summe());
```



```
1477
677
```

Können Methoden final sein?

Heinrich Heine
Universität Düsseldorf

Können Methoden final sein?

hhu
Heinrich Heine
Universität Düsseldorf

- finale Methoden können nicht überschrieben werden

```
 9  public final int summe() {
10      int summe = 0;
11      for(int betrag: einzelkosten) {
12          summe += betrag;
13      }
14      return summe;
15  }
```

Können Klassen final sein?

Heinrich Heine
Universität Düsseldorf

Können Klassen final sein?

Heinrich Heine Universität Düsseldorf

- von finalen Klassen kann nicht geerbt werden
  - Beispiel: `String`
  - Durchsetzen von Immutability
  - Durchsetzen der Regel „bevorzuge Komposition gegenüber Vererbung“ →Propra+Zusatzmaterial

```
1 public final class Kasse {
```

Lernziele

Sie können am Ende der Woche . . .

- **erklären**, warum eigene Klassen `equals` überschreiben sollten.
- den Unterschied zwischen „Überladen durch Vererbung“ und „Überschreiben“ **erklären**.
- den Unterschied zwischen „Überschreiben“ und „Hiding“ **erklären**.
- **erklären**, warum der Zugriff auf statische Variablen/Methoden über Instanzen verwirrend sein kann.

Wichtige Begriffe

Hiding