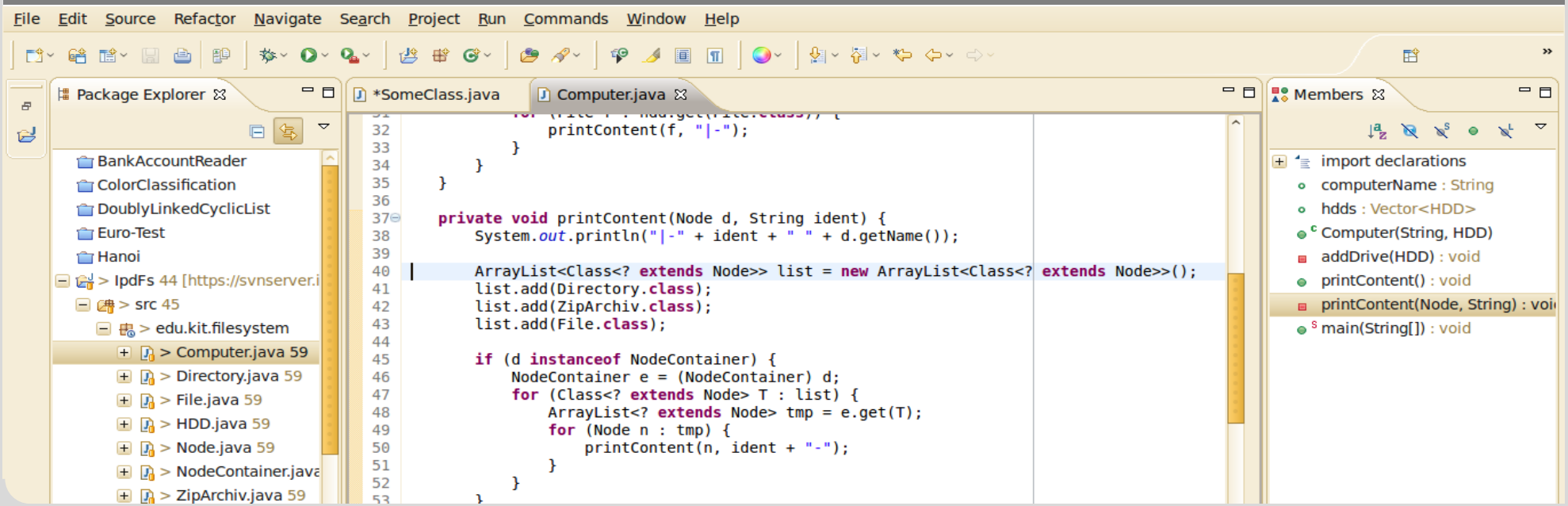
Karlsruher Institut für Technologie

Programmieren-Tutorium Nr. 10 bei Martin Thoma

Late Binding, Generics, Libraries

Martin Thoma | 16. Dezember 2012

FAKULTÄT FÜR INFORMATIK



Inhaltsverzeichnis

Quiz: Vererbung II

Animal.java

```
public class Animal {
    private String sound;

    public void setSound(String sound) {
        this.sound = sound;
    }

    public String getSound() {
        return this.sound;
    }
}
```

Cat.java

```
public class Cat extends Animal {
    public String sound;

    public Cat() {
        sound = "Maunz";
    }

    public String getCatSound() {
        return sound;
    }
}
```

Jungle.java

```
public class Jungle {
    public static void main(String[] args) {
        Animal felix = new Cat();
        System.out.println(felix.getCatSound());
        System.out.println(felix.getSound());
    }
}
```

- Gibt es einen Compiler-Fehler?
- Gibt es einen Laufzeit-Fehler?
- Gibt es eine Ausgabe? Welche?

Quiz: Antwort

Karlsruher Institut für Technologie

Animal.java

```java
public class Animal {
    private String sound;

    public void setSound(String sound) {
        this.sound = sound;
    }

    public String getSound() {
        return this.sound;
    }
}
```

Cat.java

```java
public class Cat extends Animal {
    public String sound;

    public Cat() {
        sound = "Maunz";
    }

    public String getCatSound() {
        return sound;
    }
}
```

Jungle.java

```java
public class Jungle {
    public static void main(String[] args) {
        Animal felix = new Cat();
        System.out.println(felix.getCatSound());
        System.out.println(felix.getSound());
    }
}
```

- Compiler-Fehler
- The method getCatSound() is undefined for the type Animal
- Thema: „Late binding“

Allgemeines

Motivation

- Eine Unterklasse kann alles, was die Oberklasse kann

$\rightarrow$ sie hat die gleichen Methoden, wenn auch eventuell überschrieben

- Eventuell ist uns die konkrete Klasse egal
- oder wir benötigen einen Kontainter für viele verschiedene Objekte mit einer gemeinsamen Oberklasse

Was ist „Late binding'"?

- `Animal felix = new Cat();`
- `Animal[] zoo = new Animal[10];`
  `zoo[0] = felix;`

Beispiel

```
1 import java.util.LinkedList;
2 import java.util.List;
3
4 public class Jungle {
5     public static void main(String[] args) {
6         List<Bike> bikeStock = new LinkedList<Bike>();
7     }
8 }
```

Wann wird Late binding verwendet?

- Man Benötigt eigentlich nur bestimmte Methoden (ein Interface)
  - Wenn ihr später aus einer LinkedList eine ArrayList macht, müsst ihr nur eine Zeile ändern!
- Man weiß zur Compile-Zeit die exakten Klassen noch nicht
- Bei den Abschlussaufgaben

Wann wird Late binding verwendet?

- Man Benötigt eigentlich nur bestimmte Methoden (ein Interface)
  - Wenn ihr später aus einer LinkedList eine ArrayList macht, müsst ihr nur eine Zeile ändern!
- Man weiß zur Compile-Zeit die exakten Klassen noch nicht
- Bei den Abschlussaufgaben

Wann wird Late binding verwendet?

- Man Benötigt eigentlich nur bestimmte Methoden (ein Interface)
  - Wenn ihr später aus einer LinkedList eine ArrayList macht, müsst ihr nur eine Zeile ändern!
- Man weiß zur Compile-Zeit die exakten Klassen noch nicht
- Bei den Abschlussaufgaben

Wann wird Late binding verwendet?

- Man Benötigt eigentlich nur bestimmte Methoden (ein Interface)
  - Wenn ihr später aus einer LinkedList eine ArrayList macht, müsst ihr nur eine Zeile ändern!
- Man weiß zur Compile-Zeit die exakten Klassen noch nicht
- Bei den Abschlussaufgaben

Allgemeines

- Problem: Man schreibt genau eine verkettete Listen-Klasse für genau eine Klasse (z.B. „Bike“), obwohl die Logik (hinzufügen, entfernen, suchen) von der Klasse unabhängig ist
- Lösung: Generics

Verwendung

```
1 public class SinglyLinkedList<E> {
2     private Node<E> head;
3
4     public void add(E object) {
5         Node<E> newNode = new Node<E>(object);
6         newNode.setNext(head);
7         head = newNode;
8     }
9 }
```

Hinweis

Ihr könnt den „Generic-Parameter“ wie eine Variable für die Bezeichnung einer Klasse verwenden.

Praxis

Arbeitsauftrag

Programmiert die Einfach-verkettete Liste um, sodass sie nirgends mehr „Bike“ sondern nur noch Generics verwendet.

Arbeitsauftrag

Hinweis

- URL für die Bike-Version: http://goo.gl/RoiBC
- Wer nur Zettel und Papier hat, implementiert die „Node.java“ komplett neu und „SinglyLinkedList.java“ mindestens mit „add()“

Praxis: Node

```java
public class Node<E> {
    private final E element;
    private Node<E> next;

    /**
     * Constructor.
     *
     * @param element the element you want to add
     */
    public Node(E element) {
        this.element = element;
    }

    /**
     * Getter for the content of this node.
     *
     * @return the element
     */
    public E getElement() {
        return element;
    }

    /**
     * @return the next
     */
    public Node<E> getNext() {
        return next;
    }

    /**
     * @param next the next to set
```

Praxis: SinglyLinkedList

```
1  public class SinglyLinkedList<E> {
2      private Node<E> head;
3
4      /**
5       * Add an object to the list.
6       *
7       * @param object the element you want to add
8       */
9      public void add(E object) {
10         Node<E> newNode = new Node<E>(object);
11         newNode.setNext(head);
12         head = newNode;
13     }
```

Man kanns auch übertreiben

Siehe Blog-Artikel „Java Generics“

Besonderheit: ?

```
List<? extends HasWord> wordList = toke.tokenize();
```

Bedeutet:

- Die Liste erwartet Objekte, deren Klasse `HasWord` erweitern

→ `HasWord` oder Kinder

- `?` wird „bounded wildcard“ genannt
- SO 1 und SO 2

Besonderheit: ?

```
List<? extends HasWord> wordList = toke.tokenize();
```

Bedeutet:

- Die Liste erwartet Objekte, deren Klasse `HasWord` erweitern

→ `HasWord` oder Kinder

- `?` wird „bounded wildcard“ genannt
- SO 1 und SO 2

Besonderheit: ?

```
List<? extends HasWord> wordList = toke.tokenize();
```

Bedeutet:

- Die Liste erwartet Objekte, deren Klasse `HasWord` erweitern

→ `HasWord` oder Kinder

- `?` wird „bounded wildcard“ genannt
- SO 1 und SO 2

Besonderheit: ?

```
List<? extends HasWord> wordList = toke.tokenize();
```

Bedeutet:

- Die Liste erwartet Objekte, deren Klasse `HasWord` erweitern

→ `HasWord` oder Kinder

- `?` wird „bounded wildcard“ genannt
- SO 1 und SO 2

Quiz

Hinweis

Das folgende Quiz stammt von www.grayman.de. Danke für die Erlaubnis, es in meine Folien einbinden zu dürfen!

Quiz

Frage

Mit Generics hat der Compiler mehr Typ-Informationen. Explizite Casts müssen als nicht so of benutzt werden.
Welche Bedeutung für die Laufzeit haben Generics?

- Der Compiler kann mit Generics den Code besser für Typen optimieren. Das, und das Wegfallen der Casts sind Gründe warum der kompilierte Code mit Generics **schneller** läuft als ohne
- Generics haben **keinen Einfluss** auf die Laufzeit
- Die erhöhte Flexibilität und Typsicherheit bedeutet, dass der Compiler für jeden konkreten Typen Code aus dem generischen Code erstellen mus. Das bedeutet, die Programme sind **etwas langsamer**

Quiz

Frage

Mit Generics hat der Compiler mehr Typ-Informationen. Explizite Casts müssen als nicht so of benutzt werden.
Welche Bedeutung für die Laufzeit haben Generics?

Die Java Virtual Machine und der kompilierte Byte-Code sind unabhängig von Generics. Der kompilierte Byte-Code mit Generics unterscheidet sich nicht von Byte-Code ohne Generics. Generics haben also **keinen Einfluss** auf die Laufzeit von Java-Programmen.

Quiz

Frage

```
public class Basket<E> {
    private E element;

    public void setElement(E x) {
        element = x;
    }

    public E getElement() {
        return element;
    }
}
```

```
class Fruit {
}

class Apple extends Fruit {
}

class Orange extends Fruit {
}
```

In main

```
1 Basket<Fruit> basket = new Basket<Fruit>();
2 basket.setElement(new Apple());
3 Apple apple = basket.getElement();
```

- Der source code ist OK. Es gibt weder Compiler-, noch Laufzeitfehler
- Compiler-Fehler in Zeile 2
- Compiler-Fehler in Zeile 3

Quiz

Frage

```java
public class Basket<E> {
    private E element;

    public void setElement(E x) {
        element = x;
    }

    public E getElement() {
        return element;
    }
}
```

```java
class Fruit {
}

class Apple extends Fruit {
}

class Orange extends Fruit {
}
```

In main

```
1 Basket<Fruit> basket = new Basket<Fruit>();
2 basket.setElement(new Apple());
3 Apple apple = basket.getElement();
```

- Zeile 2 ist ok
- Zeile 3 verursacht einen Laufzeitfehler
- Der Rückgabewert der Methode `getElement` in `Basket<Fruit>` ist `Fruit`. Man kann eine `Fruit`-Variable keiner `Apple`-Variable ohne cast zuweisen.

Quiz

Frage

In main

```
1 Basket<Fruit> basket = new Basket<Fruit>();
2 basket.setElement(new Apple());
3 Orange orange = (Orange) basket.getElement();
```

- Es gibt weder Laufzeit-, noch Compiler-Fehler
- Compiler-Fehler in Zeile 2
- Compiler-Fehler in Zeile 3
- Eine ClassCastException tritt in Zeile 3 auf

Quiz

Frage

In main

```
1 Basket<Fruit> basket = new Basket<Fruit>();
2 basket.setElement(new Apple());
3 Orange orange = (Orange) basket.getElement();
```

- Sowohl `Apple` als auch `Orange` sind `Fruit` und können in `Basket<Fruit>` landen

→ Der Cast in Zeile 3 ist nötig

- Die JVN überprüft während der Laufzeit den Cast in Zeile 3

→ Ein `ClassCastException` wird geworfen, da `Apple` keine `Orange` ist

Quiz

Frage

Welche der folgenden Zeilen kann ohne Compiler-Fehler kompilieren?

```
1 Basket b = new Basket();
2 Basket b1 = new Basket<Fruit>();
3 Basket<Fruit> b2 = new Basket<Fruit>();
4 Basket<Apple> b3 = new Basket<Fruit>();
5 Basket<Fruit> b4 = new Basket<Apple>();
6 Basket<?> b5 = new Basket<Apple>();
7 Basket<Apple> b6 = new Basket<?>();
```

Quiz

Frage

Welche der folgenden Zeilen kann ohne Compiler-Fehler kompilieren?

```
1 Basket b = new Basket();
2 Basket b1 = new Basket<Fruit>();
3 Basket<Fruit> b2 = new Basket<Fruit>();
4 Basket<Apple> b3 = new Basket<Fruit>();
5 Basket<Fruit> b4 = new Basket<Apple>();
6 Basket<?> b5 = new Basket<Apple>();
7 Basket<Apple> b6 = new Basket<?>();
```

- Generische Klassen können ohne spezifizierten Typ genutzt werden. Allerdings sollte man das nicht machen und Eclipse warnt auch davor.
- Korrekt sind: 1, 2, 3, 6
- Grundsätzlich gilt: Rechts vom `=` darf man genauer sein als links

Quiz

Frage

SourceA.java

```
1 Basket<?> b5 = new Basket<Apple>();
2 b5.setElement(new Apple());
3 Apple apple = (Apple) b5.getElement();
```

SourceB.java

```
1 Basket b = new Basket();
2 b.setElement(new Apple());
3 Apple apple = (Apple) b.getElement();
```

SourceC.java

```
1 Basket b1 = new Basket<Orange>();
2 b1.setElement(new Apple());
3 Apple apple = (Apple) b1.getElement();
```

Which of the following statements are true?

(a) SourceA kompiliert nicht

(b) SourceB kompiliert mit warning(s). Es gibt keine Laufzeit-Fehler

(c) SourceC kompiliert mit warning(s). Es gibt eine ClassCastException zur Laufzeit

Quiz

Frage

SourceA.java

```
1 Basket<?> b5 = new Basket<Apple>();
2 b5.setElement(new Apple());
3 Apple apple = (Apple) b5.getElement();
```

SourceB.java

```
1 Basket b = new Basket();
2 b.setElement(new Apple());
3 Apple apple = (Apple) b.getElement();
```

SourceC.java

```
1 Basket b1 = new Basket<Orange>();
2 b1.setElement(new Apple());
3 Apple apple = (Apple) b1.getElement();
```

Which of the following statements are true?

- (a) und (b) sind richtig.
- Der Compiler kennt nicht den Typ von Elementen in b5

→ Er kann nicht garantieren, dass ein Apfel in b5 eingefügt werden kann

- Da `b5.setElement(..)` nicht erlaubt ist, kompiliert es nicht
- SourceB wird so behandelt, als ob es pre-Java 1.5 Code wäre

Frage

Sollen wir die restlichen 9 Fragen auf grayman.de machen?

Einschub: Libaries

Warum heißen Programmbibliotheken „Bibliotheken“?
→ Video ‘"RRZE1973-MPEG-1.mpg’" der Uni Erlangen

Kommende Tutorien

- 24.12.2012: Heiligabend - Kein Tutorium
- 31.12.2012: Silvester - Kein Tutorium

4. 07.01.2013
3. 14.01.2013
2. 21.01.2013
1. 28.01.2013: Abschlussprüfunsvorbereitung
0. 04.02.2013: Abschlussprüfunsvorbereitung

- 10.02.2013: Ende der Vorlesungszeit des WS 2012/2013 (Quelle)

Frohe Weihnachten!