Objektorientierte Programmierung

UML und Interfaces

Prof. Dr. Ulrike Hammerschall
Fakultät für Informatik und Mathematik

Interfaces in Java - Wiederholung

- In Java werden Interfaces wie Klassen erstellt. Schlüsselwort ist hier: **interface**.

Ein Interface kann enthalten:	Ein Interface enthält nicht
○ Signaturen von öffentlichen Methoden. ○ In manchen Fällen öffentliche Konstanten. ○ In manchen Fällen statische Methoden. ○ In manchen Fällen Default-Methoden.	○ Signaturen oder Rümpfe von nicht-öffentlichen-Methoden. ○ Konstruktoren oder Objektvariablen.

- Interfaces werden wie Klassen in Dateien definiert:
  - Eine Interface-Definition pro Quelltextdatei: <Name>.java
  - Eine Bytecode Datei pro Quelltextdatei: <Name>.class
- Es können keine Objekte von Interfaces erstellt werden.
- Ein Interface kann im Extremfall auch komplett leer sein (Tagging Interface).

Interfaces als Datentypen

- Interfaces definieren Referenztypen. Bei der Initialisierung der Variable wird ein Objekt einer der Klassen zugewiesen, die das Interface implementieren.
- Beispiel: Definition einer Variable vom Typ Printable

```
Printable printable;
```

- Beispiel: Initialisierung mit einem Objekt vom Typ Book

```
Printable printable = new Book();
```

- Beispiel: Initialisierung mit einem Objekt vom Typ Poster

```
Printable printable = new Poster();
```


Methodenauswahl

- Methodenaufruf auf der Variable:
- Beispiel Book:

```
Printable printable = new Book();
printable.print();  // -> "Druck auf Laserdrucker"
```

- Beispiel Poster:

```
Printable printable = new Poster();
printable.print();  // -> "Druck auf Plotter"
```

Gleichrangige Implementierungen

- Die Klassen *Book* und *Poster* implementieren beide unabhängig voneinander das Interface *Printable*.
- Das Interface garantiert, dass diese Klassen die Methode *print()* implementieren. Die Prüfung erfolgt durch den Compiler.
- Klassen, die ein Interface implementieren, müssen für ALLE Methoden des Interfaces eine Implementierung liefern.
- Die Klassen können beliebig viele weitere Methoden, Konstruktoren und Objektvariablen definieren.

Methodenauswahl

- Die Implementierung der print-Methode kann bei Bedarf erst zur Laufzeit ausgewählt werden.

```
boolean condition = …;      // Berechnung der Bedingung
Printable printable;
if(condition) {
        printable = new Book();
} else {
        printable = new Poster();
}
printable.print()    //Book oder Poster ?
```

Polymorphie (Vielgestaltigkeit) und dynamisches Binden

- Kennzeichnende Konzepte objektorientierter Sprachen.
- Polymorphie:
  - Es existieren unterschiedliche Implementierungen einer Methodensignatur gleichberechtigt nebeneinander.
- Dynamisches Binden:
  - Erst zur Laufzeit wird anhand des aktuellen Objekts die konkrete Implementierung einer Methode ausgewählt.

Statischer und dynamischer Typ

- Als statischer Typ wird der Typ einer Variablen gemäß Definition bezeichnet.
- Als dynamischer Typ wird der Typ des tatsächlich an eine Variable zugewiesenen Objekts bezeichnet.

```
Printable printable = new Book();
```

- Gilt auch für Basisdatentypen. Für den dynamischen Typ müssen die die Regeln der impliziten Typkonversion erfüllt sein.

```
int intValue = 'a'; // intValue erhält den Wert 97 (ASCII Code von a)

double doubleValue = intValue; // doubleValue erhält den Wert 97.0
```

Prüfung der Typsicherheit

- Der Compiler prüft, ob die aufgerufene Methoden im Interface definiert sind.
- Der Compiler prüft, ob alle in Frage kommenden Klassen das Interface vollständig implementieren.
- Der Compiler kann für einen Methodenaufruf somit
  - sicherstellen, dass irgendeine passende Methode existiert,
  - nicht entscheiden, welche konkrete Methode das sein wird.
  - kann nicht vor dem Wert null schützen: in diesem Fall Programmabbruch mangels Objekt .
- Der Interpreter entscheidet, welches Objekt die Methode zur Laufzeit tatsächlich ausführt.

Interfaces als Datentyp für Parameter

- Interfaces können wie normale Klassen als Datentypen für Parameter in den Parameterlisten von Methoden verwendet werden.

```
public void printMedium(final Printable printable) {
	printable.print();
}
```

- Übergeben werden konkrete Objekte, die das Interface implementieren.

```
public void doSomething() {
	Printable printable = new Book(...);
	printMedium(printable);
}
```

Interfaces als Ergebnistyp von Methoden

- Interfaces können als Ergebnistypen von Methoden verwendet werden.

```
public Printable getPrintable() {
       Printable printable = ....
       return printable;
}
```

- Kompatible Ergebnistypen sind ebenfalls zulässig.

```
public Printable getPrintable() {
       Book book = new Book();
       return book;
}
```


Einsatz von Interfaces

- Ziel: Einfache Erweiterbarkeit eines Programms:
- Im Beispiel:
  - Weitere Arten von druckbaren Medien (z.B. Journal, EBook, etc) möglich.
  - Können ohne Änderungen an bestehenden Klassen oder Interfaces in das Programm integriert werden.
- Alle vorhandenen Klassen im Programm können über das Interface sofort mit den neuen Klassen arbeiten.

Zusammenfassung

- Interfaces stellen den Vertrag zwischen dem Anwender einer Klasse und der Implementierung der Klasse dar.
- Sie definieren Methodensignaturen, die von implementierenden Klassen garantiert über deren öffentliche Schnittstelle angeboten werden müssen.
- Interfaces können nicht selbst initialisiert werden, sie können jedoch als Variablentypen bzw. Parametertypen verwendet werden (statische Typen). Werte sind Objekte kompatibler Klassen (dynamischer Typ).
- Sie erlauben damit die einfache Erweiterung von Programmen durch Integration neuer Klassen.
- Polymorphie und dynamischen Binden sind zentrale Konzepte der Objektorientierung.
  - Es existieren parallel unterschiedliche Methodenimplementierungen.
  - Die Auswahl der tatsächlich verwendeten Methode findet zur Laufzeit statt.
- Seit Java 8 Erweiterung der Interfaces um Default- und statische Methoden.