Weitere Techniken zur Abstraktion

Buch, Kapitel 12; Abstrakte Klassen

J. Kleimann, H.-W. Sehring, D. Versick, F. Zimmermann

Studiengang Wirtschaftsinformatik

NORDAKADEMIE
HOCHSCHULE DER WIRTSCHAFT

Vorlesungsinhalt

1. Lerninhalte
2. Recap Netzwerk Anwendung
3. Überschreiben von Methoden
4. Abstrakte Klassen

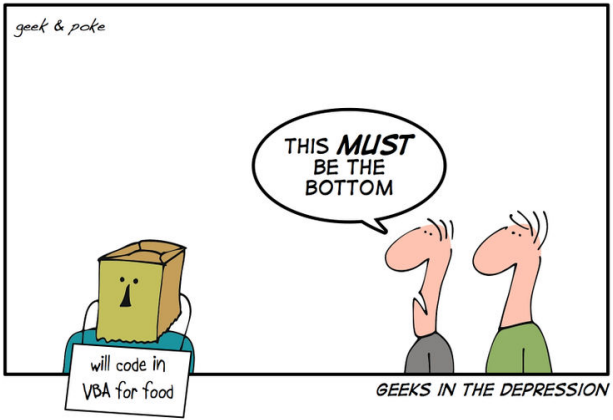


Abbildung: CC-BY-3.0 Oliver Widder

Lernziele

In dieser Lektion werden Sie

- das Überschreiben von Methoden und späte Bindung,
- abstrakte Klassen,
- und Template Methoden

kennenlernen.

Was bisher geschah

1. Durch Einführen eines Interfaces konnten **Code-Duplikationen** in der `NewsFeed`-Klasse (und in unserer Vorstellung in sehr vielen anderen Klassen) vermieden werden.
2. Die `NewsFeed`-Klasse wurde für `IPost`-Klassen **wiederverwendbar**, die noch gar nicht existieren.
3. Das System ist durch die Einführung des Interfaces **erweiterbar** geworden.
4. Durch Einführen einer Klassenhierarchie konnte **identischer Code aus unterschiedlichen Klassen herausgezogen** werden und nur ein einziges Mal hinterlegt werden.

Kritik an der Netzwerk Anwendung

Gehen sie von der Netzwerkanwendung mit einziger `display`-Methode in `Post` aus. `display()` gibt nur Attribute aus `Post` aus.

1. Die `display` -Methode in Post kann **keine Unterschiede** der Klassen `MessagePost` und `PhotoPost` wiedergeben.
2. Die `NewsFeed`-Klasse berücksichtigt deshalb **keine Charakteristiken** der Unterklassen.
3. Die **`Post`-Klasse hat keine Business Bedeutung**. Objekte dieser Klasse sind aus fachlicher Sicht nicht sinnvoll, nur die Unterklassen haben eine fachliche Existenzberechtigung.

Soll-Ist Vergleich

Soll Darstellung

```
L. da Vinci
Forgot about an idea I had last night. Something
about flying.
3 minutes ago - 1 people like this.
No comments.

A. G. Bell
telephone.jpg
New machine which I call telephone.
5 minutes ago
1 comment(s). Click here to view.
```

Ist Darstellung

```
L. da Vinci
3 minutes ago - 1 people like this.
No comments.

A. G. Bell
5 minutes ago
1 comment(s). Click here to view.
```

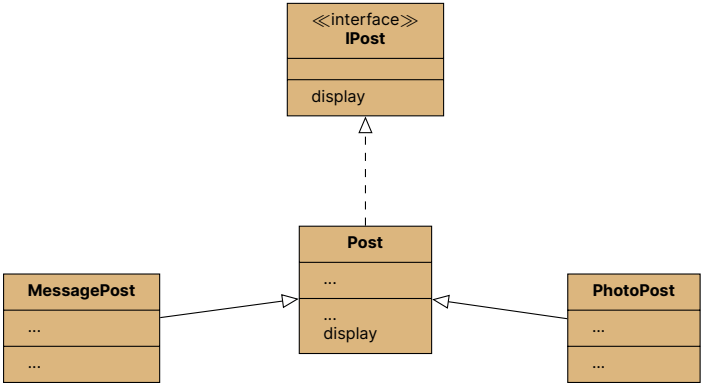
Die speziellen Informationen fehlen. Wir müssen dafür sorgen, dass diese auch ausgegeben werden.

Variante 1

Überschreiben display Methode+super

Die Methode `display` kommt mehrfach in der Typhierarchie vor:

1. Im Interface, als Zusicherung, dass `display`-Aufrufe möglich sind.
2. In der `Post`-Klasse mit der Darstellung der `Post`-Exemplarvariablen.
3. In `Message`- und `PhotoPost` mit der spezifischen Darstellung.

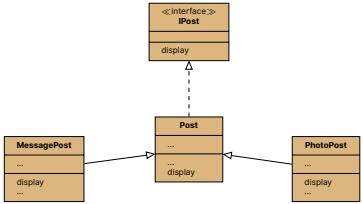


Überschreiben und Implementieren von Methoden

- Die `display`-Methode in `Post` **implementiert** die `display`-Methode aus `IPost`, indem Sie für die Methode Code zur Verfügung stellt.
- Die `display`-Methoden in `Message`- und `PhotoPost` **überschreiben** die Methode aus `Post`.
- Das Überschreiben einer Methode aus einer Oberklasse bzw. das Implementieren einer Methode aus einem Interface setzt dieselbe **Methodensignatur** voraus. Das bedeutet, dass die Methoden

1. denselben Methodennamen,
2. dieselbe Parameteranzahl,
3. dieselben Parametertypen in derselben Reihenfolge
4. und denselben Ergebnistyp (ab Java 5 gibt es den covarianten return-Type, der erlaubt, dass die überschreibende Methode eine Subklasse der überschriebenen Methode zurückgibt.)

haben.



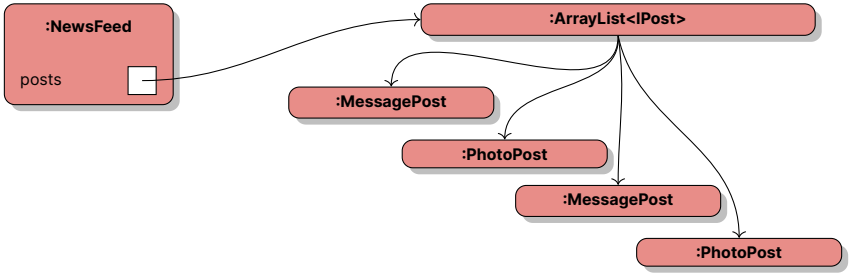
- Überschreibende Methoden werden i.d.R. mit `@Override` annotiert.

Code-Beispiel

Überschreiben einer Methode

```
public class Post{
  private String username; ...
  public void display(){
    System.out.println(username);...
        //Post-Variablen ausgeben
  }
}
public class MessagePost extends Post
    {
  private String text;...
  @Override
  public void display(){
    System.out.println(text);...//
        MessagePost-Var. ausgeben
  }
}
public class PhotoPost extends Post{
  private String filename;...
  @Override
  public void display(){
    System.out.println(filename);...
        //PhotoPost-Var. ausgeben
  }
}
```

Polymorpher Methodenaufruf und späte Bindung



- **Polymorpher Methodenaufruf**: `post.display();` ist die `display`-Methode aus `MessagePost` für ein `MessagePost` Objekt und die Methode aus `PhotoPost` für ein `PhotoPost`-Objekt.
- Welche Methode aufgerufen wird, orientiert sich an der Klasse des Objekts, das in der Sammlung steht. Dieses Konzept nennt man **Späte Bindung.**
- NB: Die `display()` Methode aus `Post` wird nie aufgerufen!

Späte Bindung

```
private TypA variableA;
...
variableA.methodeA();
...
```

- **TypA** steht für eine beliebige Klasse oder ein beliebiges Interface.
- Da Variablen polymorph sind, kann in der Variablen ein Objekt einer **beliebigen Subklasse** (bzw. implementierenden Klasse) von TypA stehen.
- Die `methodeA` wird zunächst in dieser Subklasse gesucht. Ist sie dort definiert, so wird sie aufgerufen. Ist `methodeA` dort nicht vorhanden wird in der Oberklasse dieser Subklasse weitergesucht und danach evtl. in der Oberklasse der Oberklasse usw. bis `methodeA` gefunden wird.
- Die Regeln für den Java Compiler stellen sicher, dass immer eine Implementation der `methodeA` gefunden wird, die aufgerufen werden kann.

Übung 1 siehe Test Foliensatz 11

1. Laden sie das Projekt network-v3 oder arbeiten Sie mit Ihrem Projekt aus der letzten Lektion weiter.
2. Fügen Sie je eine `display()` Methode in `MessagePost` und in `PhotoPost` ein, die die subklassenspezifischen Informationen ausgibt.
3. Überprüfen Sie das die `show`-Methode aus `NewsFeed` die Methoden aus `MessagePost` und aus `PhotoPost` aufruft, aber nicht die aus `Post`.

Der Vollständigkeit halber: Überladen von Methoden

- Java bietet die Möglichkeit, Methoden zu überladen, d.h.,
  - gleicher Methodenname **aber**
  - unterschiedliche Parameteranzahl oder
  - gleiche Parameteranzahl und unterschiedliche Parametertypen
- Rückgabewerte spielen keine Rolle bei der Methodenzuordnung

Auswahl bei überladenen Methoden

- Ist eine Zuordnung der aufrufenden Methode eindeutig, so wird die Methode ausgewählt, welche gleiche Parameteranzahl mit den passenden Parametertypen hat
- ist die Zuordnung eines Methodenaufrufs nicht eindeutig, wird die Methode ausgewählt, die zu allen in Frage kommenden Methoden die speziellste ist
- Eine Methode A ist spezieller als eine Methode B, falls die Parametertypen von A aus den Parametertypen von B abgeleitet werden können

Soll-Ist Vergleich

Soll Darstellung

```
L. da Vinci
Forgot about an idea I had last night. Something
about flying.
3 minutes ago - 1 people like this.
No comments.

A. G. Bell
telephone.jpg
New machine which I call telephone.
5 minutes ago
1 comment(s). Click here to view.
```

Ist Darstellung

```
Forgot about an idea I had last night. Something
about flying.

telephone.jpg
New machine which I call telephone.
```

Die allgemeinen Informationen fehlen jetzt. Wir müssen dafür sorgen, dass diese auch ausgegeben werden.

Aufruf der `display()`-Methode aus `Post`

- Aufgrund der späten Bindung wird die Methode `display()` aus `Post` nicht aufgerufen, so dass die gemeinsamen Informationen nicht ausgegeben werden.

- wollen wir das erreichen, müssen wir das in den Subklassen `display()`-Methoden die `Post` `display`-Methode explizit aufrufen:

Aufruf einer Methode aus einer Oberklasse

```
public class MessagePost extends Post{
  private String text;...
  @Override
  public void display(){
    super.display();  //Gemeinsame Variablen
        ausgeben
    Sytem.out.println(text);...//MessagePost-Var.
        ausgeben
  }
}
```

Aufrufen einer überschriebenen Methode

`super.methode()`

- Der Aufruf `super.methode()` ruft die Methode aus der Oberklasse auf.
- Dieser Aufruf kann an beliebiger Stelle stehen. Das ist anders als bei Aufruf von Konstruktoren, die immer die erste Anweisung in eine Konstruktor sein müssen.
- Für Besserwisser sei bemerkt, dass für diesen Aufruf keine späte Bindung erforderlich ist, da die aufzurufende Methode zur Übersetzungszeit bekannt ist.

Übung 2 siehe Test Foliensatz 11

1. Als Erweiterung der letzten Übung fügen Sie je eine **`super`**`.display();` Anweisung in die subklassenspezifischen `display`-Methoden ein.
2. Überprüfen Sie das Ergebnis.
3. Nicht schlecht, nicht wahr? Aber warum entspricht das Ergebnis nicht 100%ig dem gewünschten Ergebnis? Wir müssen leider noch mehr arbeiten. Dafür benötigen wir abstrakte Klassen mit abstrakten Methoden.

Soll-Ist Vergleich

Soll Darstellung

```
L. da Vinci
Forgot about an idea I had last night. Something
about flying.
3 minutes ago - 1 people like this.
No comments.

A. G. Bell
telephone.jpg
New machine which I call telephone.
5 minutes ago
1 comment(s). Click here to view.
```

Ist Darstellung

```
L. da Vinci
3 minutes ago - 1 people like this.
No comments.
Forgot about an idea I had last night. Something
about flying.

A. G. Bell
5 minutes ago
1 comment(s). Click here to view.
telephone.jpg
New machine which I call telephone.
```

Alle Informationen werden dargestellt. **Aber:** Die speziellen Informationen sollen in die Allgemeinen eingebettet sein.

Abstrakte Klassen

- Eigenschaften
  - Eine **abstrakte Klasse** ist eine Schablone ⇒ unvollständig implementiert
  - von ihnen können keine Exemplare gebildet werden (also kein **`new`** möglich)
  - Konstruktoren möglich, mit **`super`**`(..)` in Unterklasse aufgerufen
- eine Klasse muss mit **`abstract`** deklariert sein
  - wenn mindestens eine Methode mit **`abstract`** deklariert wurde, oder
  - wenn eine **abstrakte Methode** von der Oberklasse geerbt, und nicht überschrieben wird
- Klassen die nicht abstrakt sind heißen **konkrete Klasse**n.

Klassendefinition, Abstrakte Klasse (Syntax)

Verallgemeinert (Klassendefinition, Abstrakte Klasse)

Abstrakte Klassen können eine andere Klasse (auch abstrakte Klasse) erweitern und Interfaces implementieren. Der Einfachheit halber wurde das weggelassen.

Übung 3 siehe Test Foliensatz 11

1. Deklarieren Sie die Klasse `Post` als **`abstract`**.
2. Versuchen Sie ein Objekt der Klasse `Post` mit in der Direkteingabe mit **`new`** `Post ("Lilienthal")` zu erzeugen. Wie lautet die Fehlermeldung?

Abstrakte Methoden

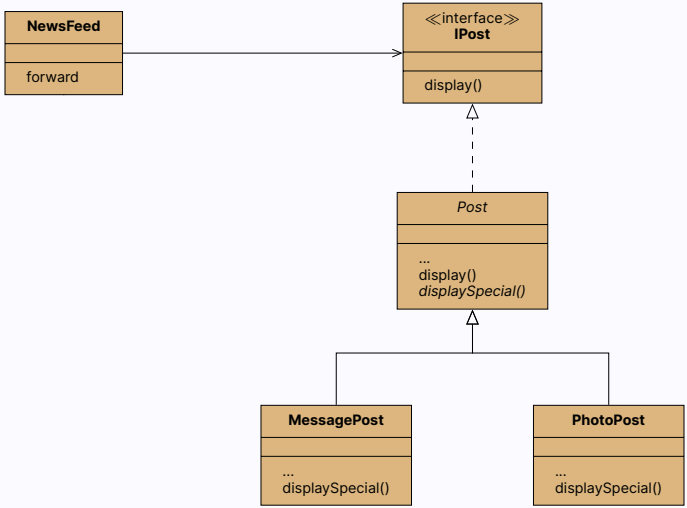
Abstrakte Methoden werden mit dem „Modifier“ **`abstract`** deklariert benötigt keine Implementierung. Eine abstrakte Methode

- hat keine Implementation (keinen Rumpf),
- darf nicht mit **`static`** oder **`final`** deklariert werden und
- muss von den (konkreten) Unterklassen implementiert werden, damit Exemplare erzeugt werden können (**Abstrakte Subklasse**).

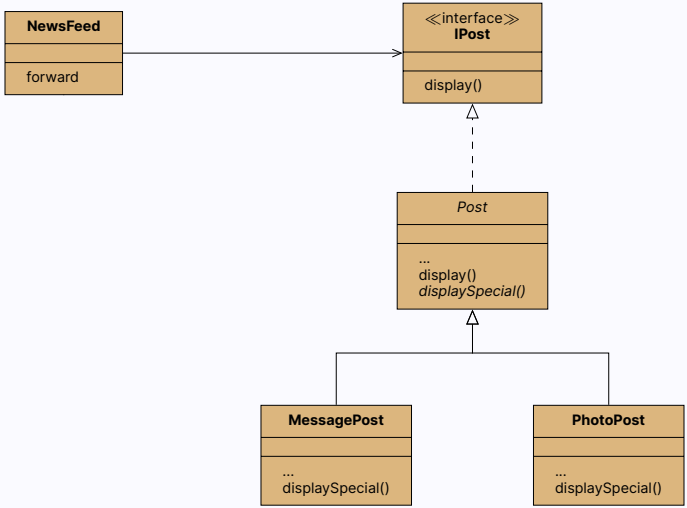
Beispiel

```
public abstract class Post {
  ...
  public abstract void
      displaySpecial();
  ...
}
```

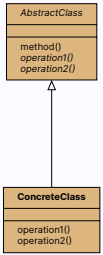
Template Methode display



Template Methode display



Die **Template-Methode** ist eines der bekannten Softwareentwurfsmuster!



Mehr zu Mustern gibt es in PdSE und in Softwaretechnik!

Szenario NewsFeed mit Template Methode

Interface IPost: Schnittstelle für NewsFeed

```
public interface IPost {
  void display(); ...
}
```

Abstrakte Klasse Post: Sichert DRY

```
public abstract class Post implements
     IPost {
  public void display() {...;
      displaySpecial(); ...}
  public abstract void displaySpecial
      ();
}
```

Konkrete Klasse MessagePost: Klassenspezifische Verarbeitung

```
public class MessagePost  extends
    Post {
  @Override
  public void displaySpecial() { /*
      spezifische Informationen*/ }
  ...
}
```

Übung 4 siehe Test Foliensatz 11

1. Fügen Sie eine abstrakte Methode `displaySpecial()` in die Klasse `Post` ein.
2. Machen Sie `display` in `Post` zu einer Templatemethode, indem Sie an der richtigen Stelle `displaySpec` aufrufen.
3. In `MessagePost` und in `PhotoPost` soll die display Methode in `displaySpecial` umbenannt werden.

Soll-Ist Vergleich

Soll Darstellung

```
L. da Vinci
Forgot about an idea I had last night. Something
about flying.
3 minutes ago - 1 people like this.
No comments.

A. G. Bell
telephone.jpg
New machine which I call telephone.
5 minutes ago
1 comment(s). Click here to view.
```

Ist Darstellung

```
L. da Vinci
Forgot about an idea I had last night. Something
about flying.
3 minutes ago - 1 people like this.
No comments.

A. G. Bell
telephone.jpg
New machine which I call telephone.
5 minutes ago
1 comment(s). Click here to view.
```

Alle Informationen werden dargestellt. **Und:** Die speziellen Informationen werden in die Allgemeinen eingebettet.

Einsatz von Interfaces und abstrakten Klassen

1. Ohne Verwendung abstrakter Klassen und Interfaces kann bessere Wiederverwendbarkeit und Wartbarkeit **nicht** erreicht werden.
2. Nutzer einer Klasse verwenden **immer** das Interface und niemals Implementierungen.
   - Wenn die Nutzer des Interfaces nicht von den konkreten Klassen, die das Interface implementieren, abhängen sondern diese nur zur Laufzeit nutzen, spricht man von einer losen Kopplung.
3. Das Interface bietet genau die Methoden an, die der Nutzer wirklich braucht. Nicht mehr und nicht weniger. (Siehe Folie 32)
4. Gemeinsames Verhalten und gemeinsamer Status verschiedener Implementierungen wird in einer abstrakten Klasse hinterlegt.

Kosten für Interface-Modifikationen

- Viele Klassen hängen von den Interfaces ab, deshalb ist der Aufwand für eine Änderung eines Interfaces groß.
- Daher erzeugen Änderungen von Interfaces einen hohen Wartungsaufwand.
- Da Interfaces keine Implementation enthalten, kann eine Veränderung ein Interface nicht betreffen.
- Schnittstellen sind sorgsam zu wählen. Wenn das geschieht ändern sie sich selten.
- Die Änderungswahrscheinlichkeit ist also niedrig.

- Geringe Wahrscheinlichkeit gepaart mit hohen Kosten ist eine tolerierbare Situation.

Kosten für Modifikationen von konkreten Klassen

- Konkrete Klassen enthalten viel Code, der sich in der Regel häufig ändert, da die Anforderungen der Kunden sich häufig ändern.
- Es ist fundamental wichtig, sicherzustellen, das der Aufwand für die notwendigen Änderungen klein bleibt.
- Dies kann man erreichen, wenn die Änderungen lokal bleiben, also andere Klassen nicht betroffen sind.
- Deshalb sollen konkrete Klassen nicht von anderen Klassen genutzt oder auch nur gekannt werden. Dadurch ist der Aufwand für eine Modifikation auf eine einzige Klasse begrenzt. Die Kosten einer Modifikation sind (verhältnismäßig) niedrig.

- Hohe Wahrscheinlichkeit einer Änderung gepaart mit niedrigen Kosten ist eine tolerierbare Situation.

Subclassing und Subtyping

- Unter Implementationsvererbung versteht man das Erben und ggf. Überschreiben von Verhalten von einer Oberklasse.
- Implementierungsvererbung (das Überschreiben von Methoden und ggf. Konstruktoren) wird auch als Subclassing bezeichnet.
- Schnittstellenvererbung wird auch als Subtyping bezeichnet.
- Subclassing und Subtyping bewirken unterschiedliche Arten von Wiederverwendung.
  - Beim Subclassing wird der Code der Superklasse wiederverwendet: Die Member-Variablen und Methoden der Superklasse sind in der Subklasse nicht nochmals zu programmieren.
  - Beim Subtyping dagegen wird der Code wiederverwendet, der die Superklasse benutzt, also seine Klienten.
- Subclassing hat geringen Wiederverwendungswert.
- Subtyping hat hohen Wiederverwendungswert.

Verändern von Interfaces

- Änderungen an einem Interface ziehen große Änderungen nach sich.
  - Nutzer des Interface müssen sich anpassen
  - Implementierer müssen ihre Klassen ändern
- Zum Verändern von Interfaces gibt es zwei Mechanismen:
  - Zum Löschen von Methoden aus einem Interface, kann man über die `@deprecated` Annotation darstellen, dass eine Methode veraltet ist. `@deprecated` sagt: Diese Methode ist veraltet und soll nicht mehr verwendet werden. In der Java Dokumentation wird in der Regel eine Alternative für die Verwendung angegeben.
  - Zum Hinzufügen von Methoden in Interfaces kann man seit Java 7 in Interfaces default-Implementationen angeben. Durch die default-Implementationen ist sichergestellt, dass der bestehende Code weiter compiliert werden kann. Ohne default-Methoden ist das nicht der Fall.

Default-Implementationen in Interfaces

Fortgeschritten

Zur Erinnerung, es sollten Posts nach der Anzahl der Kommentare sortiert werden. Dazu benötigen wir

Beispiel für eine default Implementation

```
public interface IPost extends Comparable<IPost>{
  public int getNumberOfComments();
  public void display();

  public default int compareTo(IPost other){
    return getNumberOfComments() -
                other.getNumberOfComments();
  }
}
```

Sichtbarkeits-Modifier sind in Interfaces immer **`public`** und werden normalerweise weggelassen!

Zusammenfassung

- Abstrakte Klassen erlauben es gemeinsame Aspekte der Implementierung von Klassen in eine gemeinsame Oberklasse auszulagern
- Abstrakte Klassen erhöhen die Wiederverwendbarkeit von Code
- Exemplarmethoden können überschrieben werden, überschriebene Methoden können aufgerufen werden

Fragen?

Für weitere Fragen im Nachgang können Sie mich gerne über Moodle oder via E-Mail kontaktieren!

Konzepte: Zusammenfassung I

- **Methodensignatur** Die (volle) Signatur einer Methode setzt sich aus dem Namen des Return-Typs, dem Namen der Methode und den Namen der Parametertypen (unter Beachtung der Reihenfolge) zusammen.
- **Späte Bindung** Welche Methode bei einem Methodenaufruf `variable.methode()` tatsächlich aufgerufen wird, entscheidet sich zur Laufzeit anhand der Klasse des Objekts, das in der Variablen gespeichert ist, NICHT an der deklarierten Klasse der Variable.
- **abstrakte Klasse** Eine abstrakte Klasse ist eine Klasse, von der keine Instanzen erzeugt werden sollen. Sie dient ausschließlich als Superklasse für andere Klassen. Abstrakte Klassen dürfen abstrakte Methoden anbieten.
- **abstrakte Methode** Die Definition einer abstrakten Methode besteht aus einer Methodensignatur ohne einen Rumpf. Sie wird mit dem Schlüsselwort **`abstract`** markiert.
- **konkrete Klasse** Konkrete Klassen sind Klassen, die nicht abstrakt sind. Sie müssen für alle Methoden aus implementierten Interfaces oder alle abstrakte Methoden aus erweiterten abstrakten Klassen entweder selbst eine Implementation anbieten oder von einer Oberklasse eine Implementation erben.
- **Abstrakte Subklasse** Damit eine Subklasse einer abstrakten Klasse eine konkrete Klasse werden kann, muss sie Implementierungen für alle geerbten abstrakten Methoden anbieten. Sonst ist sie selbst ebenfalls abstrakt.

Methodensignatur

Die (volle) Signatur einer Methode setzt sich aus dem Namen des Return-Typs, dem Namen der Methode und den Namen der Parametertypen (unter Beachtung der Reihenfolge) zusammen.

Späte Bindung

Welche Methode bei einem Methodenaufruf `variable.methode()` tatsächlich aufgerufen wird, entscheidet sich zur Laufzeit anhand der Klasse des Objekts, das in der Variablen gespeichert ist, NICHT an der deklarierten Klasse der Variable.

abstrakte Klasse

Eine abstrakte Klasse ist eine Klasse, von der keine Instanzen erzeugt werden sollen. Sie dient ausschließlich als Superklasse für andere Klassen. Abstrakte Klassen dürfen abstrakte Methoden anbieten.

abstrakte Methode

Die Definition einer abstrakten Methode besteht aus einer Methodensignatur ohne einen Rumpf. Sie wird mit dem Schlüsselwort `abstract` markiert.

konkrete Klasse

Konkrete Klassen sind Klassen, die nicht abstrakt sind. Sie müssen für alle Methoden aus implementierten Interfaces oder alle abstrakte Methoden aus erweiterten abstrakten Klassen entweder selbst eine Implementation anbieten oder von einer Oberklasse eine Implementation erben.

Abstrakte Subklasse

Damit eine Subklasse einer abstrakten Klasse eine konkrete Klasse werden kann, muss sie Implementierungen für alle geerbten abstrakten Methoden anbieten. Sonst ist sie selbst ebenfalls abstrakt.

Syntaxdiagramme I

Verallgemeinert (Klassendefinition, Abstrakte Klasse)

Abstrakte Klassen können eine andere Klasse (auch abstrakte Klasse) erweitern und Interfaces implementieren. Der Einfachheit halber wurde das weggelassen.