Programmiermethodik 1
Programmiertechnik

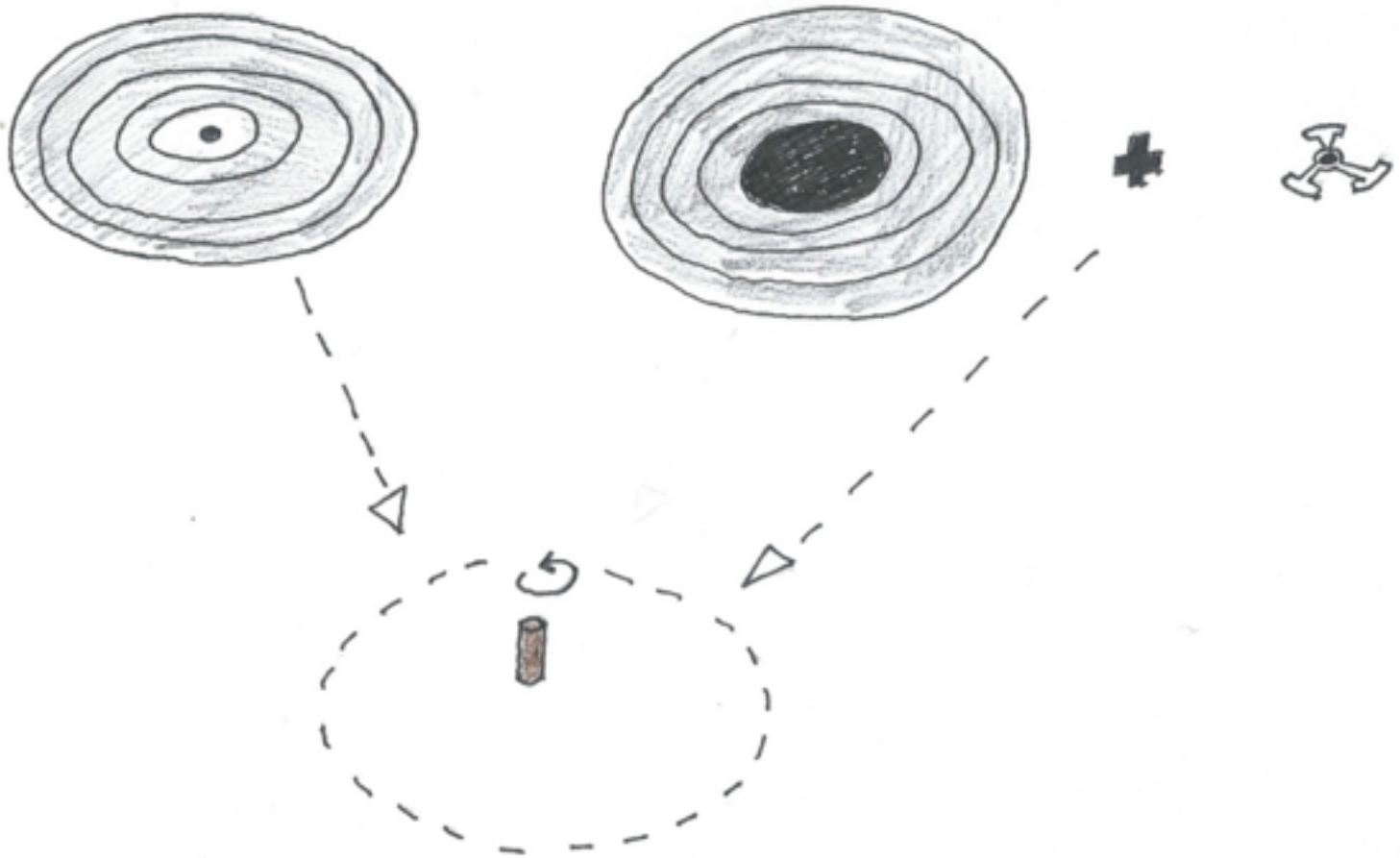
Interfaces und Vererbung

Wiederholung

- Wrapperklassen
- Strings

Ausblick

Worum gehts?



Agenda

- Einführung
- Dynamisches Binden
- Arbeiten mit Interfaces

Einführung

Motivationsbeispiel

- Anwendungsbeispiel:
    - Temperatursensor
    - „Datenbank“, um Sensorwerte abzulegen

- spätere Anforderung:
    - weiterer Sensortyp: Luftfeuchtigkeit

Schnittstellen

- bisher
  - erforderliche Funktionalität direkt implementiert (Klassen)
  - keine Trennung zwischen Schnittstelle und Implementierung
- besser: Trennung von Schnittstelle und Implementierung
  - Reduktion der Abhängigkeiten zwischen Modulen
  - Komplexitätsreduktion durch: "divide et impera!"
  - Module leichter einzeln entwerfen, implementieren, austauschen, erweitern, testen, korrigieren, ...
  - Code besser überblicken und leichter verstehen
  - unerwartete "Seiteneffekte" vermeiden
  - ...

Schnittstellen

- Schnittstelle (engl. interface) einer Klasse:
    - öffentliche Objektvariablen (Konstanten)
    - Signaturen der öffentlichen Methoden (Methodenköpfe)
- vgl. dazu: Implementierung: alles andere
- umgangssprachlich:
    - Schnittstelle beschreibt, was eine Klasse bietet (öffentlich)
    - Implementierung legt fest, wie sie das bewerkstelligt (intern)
- Anwender muss nur die Schnittstelle einer Klasse kennen, um sie zu verwenden

Schnittstellen

- Analogie: Klassen und Geschäftsleben

Java	Geschäftsleben
Klassendefinition	Anbieter, Lieferant
Anwendung einer Klasse	Kunde
Schnittstelle	Vertrag, vereinbarte Leistungen
Implementierung	Betriebsmittel, interne Maßnahmen, Geschäftsgeheimnisse

Idee

- Datenkapselung
  - Schnittstelle beschreibt, was eine Klasse bietet
  - Implementierung legt fest, wie sie das bewerkstelligt
- Interface ist ein Java-Sprachmittel, mit dem eine Schnittstelle ohne Implementierung definiert werden kann!
- Syntax:

```
public interface <Interface-Name> {
   public <Ergebnistyp> <Methodenname1>(<Parameterliste>);
   public <Ergebnistyp> <Methodenname2>(<Parameterliste>);
   …
}
```

Beispiel: Vermögenswerte

- viele, komplett unterschiedliche Dinge (Klassen!) können einen Vermögenswert darstellen:
    - Aktiendepots
    - Grundstücke
    - Girokonto-Geldbestände
    - ...
- Welche Eigenschaften benötige ich, um die Verwendung als Vermögenswert zu ermöglichen?
    - zum Beispiel für die Berechnung des Gesamtvermögens

Beispiel: Vermögenswerte

```
public interface Vermoegenswert {
  public static enum Risiko {
    NIEDRIG, MITTEL, HOCH
  }

  public String getName();

  public double getEuroWert();

  public Risiko getRisko();
}
```

Eigenschaften

- Interface-Definition ähnlich wie Klassendefinition
  - eigene Datei
  - aber mit reserviertem Wort interface statt class
- abstrakte Zusicherung (Vertrag), die von mindestens einer "konkreten" Klasse implementiert werden muss
- kennt keine konkreten Objekte, keine Objektvariablen, sondern nur abstrakte Methoden
- in der Definition fehlen daher …
  - Rümpfe der public-Methoden, statt dessen nur „;“
  - andere als public-Methoden
  - Konstruktoren
  - Objektvariablen

Übung: Interfaces

- Entwerfen Sie ein Interface für Vögel (Vogel). Ein Vogel kann fliegen (flieg) und singen (sing).

Implementierung

- isoliertes Interface ist nutzlos!
- konkrete Klassen implementieren das Interface
    - also: definieren die Methoden des Interface
- Schlüsselwort implements koppelt Klasse und Interface
- Syntax:

```
public class <Klassenname> implements <Interface-Name> { … }
```

- Klassendefinition ansonsten "normal"

Hinweis: Ab Java 8 verschwimmt diese Trennung etwas.
Dazu mehr in PM2.

Beispiel: Klasse Aktiendepot

- muss alle Methoden des Interfaces implementieren
  - beliebige zusätzliche Methoden sind aber erlaubt

```
public class Aktiendepot implements
   Vermoegenswert {
 private String unternehmen;
 private int anzahlAktien;
 private double aktuellerWert;
 public Aktiendepot(String unternehmen,
   int anzahlAktien, double aktuellerWert) {
    this.unternehmen = unternehmen;
    this.anzahlAktien = anzahlAktien;
    this.aktuellerWert = aktuellerWert;

  }
  @Override
  public String getName( ) {
    return unternehmen;
  }
  @Override
  public double getEuroWert( ) {
    return anzahlAktien * aktuellerWert;
  }
  @Override
  public Risiko getRisko( ) {
    return Risiko.HOCH;
  }
}
```

Einschub: @Override ist eine Annotation. Bedeutet: Implementieren/Überschreiben einer Methode

Beispiel: Klasse Grundstück

```
public class Grundstueck implements Vermoegenswert {
 private String adresse;
 private double flaeche;
 private double quadratmeterpreis;
 public Grundstueck(String adresse, double flaeche, double quadratmeterpreis) {
    this.adresse = adresse;
    this.flaeche = flaeche;
    this.quadratmeterpreis = quadratmeterpreis;
  }
 @Override
  public String getName() {
    return adresse;
  }
  @Override
  public double getEuroWert() {
    return flaeche * quadratmeterpreis;
  }
  @Override
  public Risiko getRisko() {
    return Risiko.NIEDRIG;
  }
}
```

Vergleich der Implementierungen

- Die Klassen Aktiendepot und Grundstück ...
  - sind "gleichrangig"
  - implementieren beide alle Methoden des Interfaces Vermoegenswert
  - haben in Bezug auf den Vermögenswert ähnliche Eigenschaften
  - sind aber voneinander komplett unabhängig
  - haben unterschiedliche Objektvariablen, unterschiedliche Methodenrümpfe und ggf. zusätzliche Methoden
    - z.B. eigene Konstruktoren

Übung: Oldtimer

- Auch alte Autos können Vermögenswerte sein. Schreiben Sie eine Klasse Oldtimer, die Vermoegenswert implementiert. Der Wert eines Oldtimers soll sich aus dem Alter und einem Basiswert (beide im Konstruktor gesetzt) errechnen.

Dynamische Bindung

Typen

- Interfaces sind Typen, ebenso wie Klassen
  - jedes Interface ist zulässiger Typ für Variablendeklarationen, Parameterlisten, Ergebnisrückgabe, Arrayelemente, …
- Beispiel
  - Variable vermoegenswert vom Typ Vermoegenswert:
  - Vermoegenswert vermoegenswert;
- Objekte des Interface gibt es nicht
  - Was könnte vermoegenswert überhaupt zugewiesen werden?
  - alle implementierenden Klassen sind kompatibel zum Interface!
- Beispiel: vermoegenswert kann ein Aktiendepot-Objekt zugewiesen werden:

```
Vermoegenswert vermoegenswert =
  new Aktiendepot("Interface AG", 100000, 3.5);
```

Methodenauswahl bei Aufruf

```
Vermoegenswert vermoegenswert = new Aktiendepot(
   "Interface AG", 100000, 3.5);
System.out.println("Value: " + vermoegenswert.getEuroWert());
```

- Methode der Klasse Aktiendepot wird verwendet!

```
String auswahl = ... // z.B. "Aktien";
if (auswahl.equals("Aktien")) {
   vermoegenswert = new Aktiendepot("Interface AG",
      100000, 3.5);
} else {
   vermoegenswert = new Grundstueck("Steindamm 94",
      500, 2000);
}
System.out.println("Wert: " + vermoegenswert.getEuroWert());
```

- der Compiler kann vorab keine Methode auswählen!

Dynamisches Binden von Methoden

- Auswahl einer konkreten Methode fällt erst zur Laufzeit
    - der Compiler trifft keine Entscheidung
- JVM sucht pro Aufruf eine Methode des momentan zugewiesenen Objekts
- Bezeichnung: Dynamisches Binden
    - Zuordnung Methodenaufruf ↔ Methodenrumpf zur Laufzeit

Statischer Typ

- Typ einer Variablen gemäß Deklaration
- Beispiel: Statischer Typ von vermoegenswert ist Vermoegenswert
    - Vermoegenswert vermoegenswert;
- statischer Typ bleibt immer gleich

Statischer Typ

- Für eine Variable vermoegenswert eines Interfacetyps gilt:
  - Der Compiler prüft, ob alle von vermoegenswert aufgerufenen Methoden im Interface definiert sind
  - Zur Laufzeit wird der Variablen ein Objekt einer konkreten, kompatiblen Klasse zugewiesen, die alle Interface-Methoden implementiert
  - eine passende Methode muss existieren!
- Der Compiler kann für einen Methodenaufruf …
  - sicherstellen, dass irgendeine passende Methode existiert
  - nicht entscheiden, welche konkrete Methode das sein wird
- Compiler kann nicht vor dem Wert null schützen
  - Programmabbruch mangels Objekt

Dynamischer Typ

- Typ des tatsächlich an eine Variable zugewiesenen Objekts

```
Vermoegenswert vermoegenswert = new Aktiendepot("Interface AG",
   100000, 3.5);
```

- vermoegenswert hat den …
  - statischen Typ Vermoegenswert (gemäß Variablendeklaration)
  - dynamischen Typ Aktiendepot (das tatsächlich zugewiesene Objekt)
- Der …
  - statische Typ bleibt immer gleich, ist beim Übersetzen entscheidend (Compiler)
  - dynamische Typ kann sich ändern, ist zur Laufzeit entscheidend (JVM)

Risiken

- Erweiterung eines Interfaces betrifft alle implementierenden Klassen
- Beispiel
    - Interface-Erweiterung vergleicht zwei Vermögenswerte bzgl. des Risikos

```
public boolean hatGroesseresRisikoAls(Vermoegenswert anderer);
```

- Implementierung in allen Klassen nötig

```
public boolean hatGroesseresRisikoAls(Vermoegenswert anderer){
   return risiko.compareTo(anderer.getRisiko());
}
```

Dynamisches Binden

```
Vermoegenswert v1 = new Aktiendepot("Interface AG", 100000,
   3.5);
Vermoegenswert v2 = new Grundstueck("Parkallee 345", 500,
   2000);
if (v1. hatGroesseresRisikoAls(v2)) {
   ...
}
```

- Der dynamische Typ von v1 entscheidet darüber, welche Implementierung von hatGroesseresRisikoAls() aufgerufen wird!

Übung: Dynamisches Binden

- Was ist die Konsolenausgabe durch die folgenden Anweisungen?
- Was ist der statische, was der dynamische Typ von ausgabe?

```
ISelbstausgabe ausgabe =
   null;
ausgabe.ausgeben();
ausgabe = new A();
ausgabe.ausgeben();
ausgabe = new B();
ausgabe.ausgeben();
```

```
public interface ISelbstausgabe {
   public void ausgeben();
}

public class A implements
   ISelbstausgabe {
   @Override
   public void ausgeben() {
      System.out.println("Ich bin ein
         A.");
  }
}

public class B implements
   ISelbstausgabe {
   @Override
   public void ausgeben() {
      System.out.println("Ich
         bin ein B.");
   }
}
```

Arbeiten mit Interfaces

Statische Variablen

- öffentliche statische Konstanten sind als Variablen in Interfaces zulässig
    - keine anderen!
- wenn nicht angegeben: Compiler ergänzt automatisch Modifier

```
public static final
```

- Initialisierung ist Pflicht!
- Beispiel: Interface-Erweiterung

```
public interface Vermoegenswert {
public static final double MIN_VALUE = 1000.0;
   …
}
```

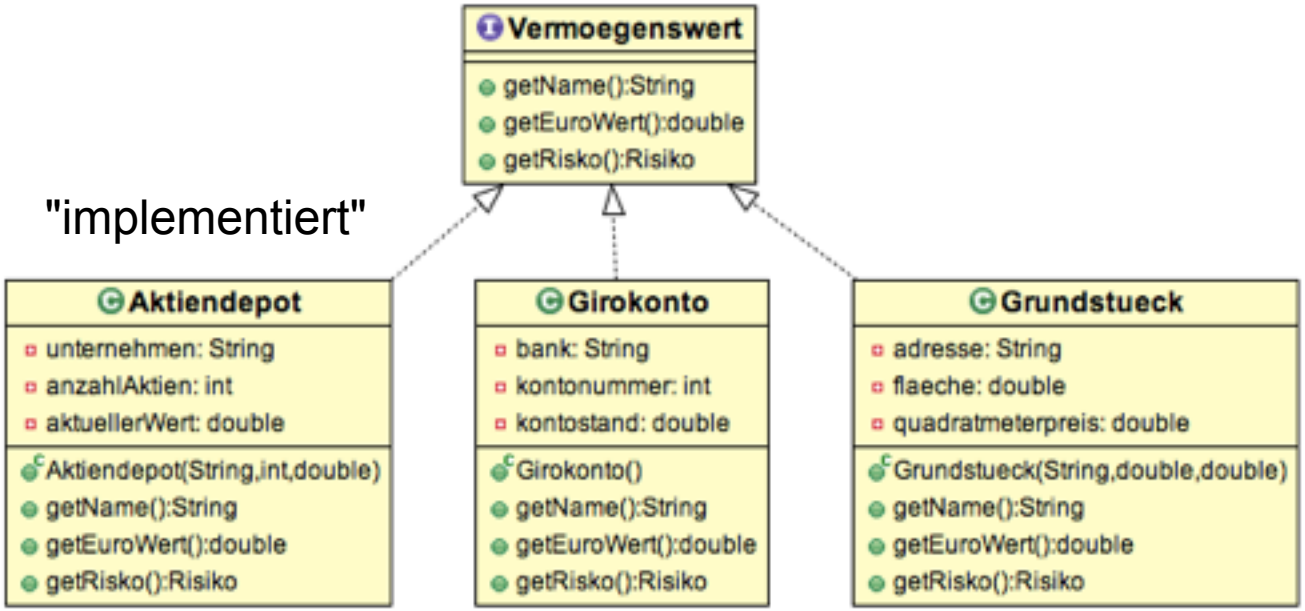
Einsatz von Interfaces

- Entwicklung und Modifikation von Interface-Implementierungen ist ohne Rücksicht auf andere Klassen zum gleichen Interface möglich
- Beispiel: neue Implementierung durch Klasse Girokonto

```
public class Girokonto
    implements Vermoegenswert {
    private String bank;
    private int kontonummer;
    private double kontostand;
    @Override
    public String getName() {
        return bank + "(" +
            kontonummer + ")";
    }
    @Override
    public double getEuroWert() {
        return kontostand;
    }
    @Override
    public Risiko getRisko() {
        return Risiko.NIEDRIG;
    }
}
```

UML: Darstellung der Zusammenhänge

- UML-Darstellung
  - Interface
  - implementierende Klassen



Konvertierung: Widening

- Ausweitung, Up-Cast, Aufwärtsanpassung:
- von der abgeleiteten Klasse zur Basisklasse
- Beispiel

```
Vermögenswert vermoegenswert = new Aktiendepot( ... );
```

- ohne expliziten Type-Cast möglich
    - Erinnerung: impliziter Typ-Cast int → double

Konvertierung: Narrowing

- Verengung, Down-Cast, Abwärtsanpassung
- von der Basisklasse zu einer abgeleiteten Klasse
- Beispiel:

```
Vermoegenswert vermoegenswert = ...
Aktiendepot aktienDepot = (Aktiendepot) vermoegenswert;
```

- vermoegenswert muss vom Typ Aktiendepot sein!
- nur mit explizitem Type-Cast möglich
    - Erinnerung: expliziter Typ-Cast double → int

Datenkapselung und Schnittstellen

- sinnvolle Reihenfolge von Codeabschnitten in Klassendefinitionen

Codeabschnitt	Rolle
Konstanten	Schnittstelle (public|private static final)
Variablen	Implementierung (private)
Default-Konstruktor Weitere Konstruktoren	Schnittstelle (public)
Setter und weitere ändernde Methoden	Schnittstelle (public)
Getter und weitere Auskunftsmethoden	Schnittstelle (public)
Haupt-Methoden	Schnittstelle (public)
Hilfsmethoden	Implementierung (private)

zurück: Motivationsbeispiel

- Anwendungsbeispiel:
    - Temperatursensor
    - „Datenbank“, um Sensorwerte abzulegen

- spätere Anforderung:
    - weiterer Sensortyp: Luftfeuchtigkeit
- Idee: gemeinsames Interface für alle Sensoren

Zusammenfassung

- Einführung
- Dynamisches Binden
- Arbeiten mit Interfaces