# Einführung in die Programmierung

Georg Ringwelski



## Wiederholung

1. Wie wird in Java eine Prozedur definiert?
2. Wie kann eine Prozedur ein Ergebnis liefern?
3. Was ist Rekursion?



## Einführung in die Programmierung

1. Die Programmiersprache Java
2. Imperativen Programmierung

Exkurs: Software testen

3. Einführung Objektorientierte Programmierung



## Was ist Testen?

- Dynamisches Testen: Programme mit Testprogrammen ausführen und Ergebnisse automatisch analysieren

```
void testFoo(){
     int y = foo(5);
     annahmeGleich(y,42);
}
void annahmeGleich(int ist, int soll){
     if(ist == soll) print(„test ok");
     else print("Fehler: erwartet:"
                +soll+" bekommen: "+ist)
}
```



## Ziel des Testes

- Möglichst viele „Fehlerwirkungen“ finden
- Diese protokollieren (nachvollziehbar, wiederholbar, verständlich)
- Qualität der Software objektiv einschätzen

*Keine Ziele sind:*

- *Fehler beheben*
- *Verbesserungsvorschläge geben*
- *möglichst viele erfolgreiche Tests*



## Grundregeln beim Testen

Dynamische Tests sollen…

- automatisch wiederholbar
- determiniert
- kontextunabhängig

… sein

## Übung

Arbeit in Kleingruppe (3 Minuten)

- Schreiben Sie eine Prozedur, die testet, ob eine Prozedur `int add(int a, int b)` ganze Zahlen richtig addiert



## Austesten?

- Ein einfaches Programm soll getestet werden, das drei ganzzahlige Eingabewerte hat. Übrige Randbedingungen haben keinen Einfluss auf das Testobjekt.
- Jeder Eingabewert kann bei 16 Bit Integerzahlen $2^{16}$ unterschiedliche Werte annehmen.
- Bei drei unabhängigen Eingabewerten ergeben sich $2^{16} * 2^{16} * 2^{16} = 2^{48}$ Kombinationen.
- Jede dieser Kombinationen ist zu testen.
- Wie lange dauert es bei 100.000 Tests pro Sekunde?
- Es sind 281.474.976.710.656 Testfälle
  Dauer: ca. 90 Jahre




## Testfälle

- Ein Paar von Eingabewerten und erwarteter Ausgabe wird als *Testfall* bezeichnet.
- In der Regel sind mehrere Testfälle für eine Prozedur notwendig, um alle qualitativ unterschiedlichen Eingaben zu überprüfen
- zB bei Zahlen bedeutet „qualitativ unterschiedlich“:
  - Größer null
  - Kleiner null
  - Gleich null
  - Sehr groß
  - Sehr klein



## Klassen von Testfällen

- Um die Anzahl der positiven Testfälle zu verringern werden Klassen von Testfällen gebildet, und nur ein Fall daraus wirklich als Test realisiert (equivalence partitioning)
- Z.B.
  - Klasse der positiven Zahlen→ Testfall 5
  - Klasse der negativen Zahlen→ Testfall -1
  - 0 → Testfall 0
  - Sehr große Zahlen → Testfall Integer.MAX_VALUE
  - Sehr kleine Zahlen → Testfall Integer.MIN_VALUE



## Übung

Einzeln (2 min)

- Notieren Sie einen Testfall für eine Prozedur `int div(int a, int b)`, die ganzzahlig dividiert auf eine Karteikarte
- Hinweis: Ein Testfall ist ein Paar aus Eingabewerte und Ausgabewert, als hier z.B.
  - Eingabe: a=10, b=3
  - Erwartete Ausgabe: 3



## Übung

Im Plenum

1.) Nennen Sie ihren Testfall

2.) Ordnen Sie ihn unter denen an der Tafel ein, ODER sagen Sie, dass er sich qualitativ von allen bisher genannten Fällen unterscheidet

- Ich habe …, das ist ähnlich wie…

ODER:

- Ich habe …, das ist wirklich anders als die, die bereits an der Tafel zu sehen sind

## Positive und negative Tests

- Postive Tests überprüfen das Verhalten des Programms bei gültigen Eingaben
- *Negative Tests* überprüfen, ob fehlerhafte Eingaben wie gewünscht verarbeitet werden
- Beispiel:

```
void test(){
     int b = dividiere(2,0);
     // was erwarte ich für b ???
}
```



## Testfälle in negativenTests

- Eingabe: ungültige Werte
- Erwartete Ausgabe (Varianten):
  - Fehlercode wie in der Dokumentation
    - Bsp: Bei der Division durch 0 wird Integer.MAX_VALUE zurückgegeben
  - Expliziter Fehlerzustand des Objekts (dokumentiert)
    - Bsp: Nach der Division durch 0 liefert die Methode boolen gueltig() den Wert false
  - Exception → im 2. Sem.



## Testbarkeit

- Programme haben nicht „von Natur aus“ die Struktur, dass man alle nötigen positiven und negativen Tests durchführen kann
- Es ist die Aufgabe eines Entwicklers, testbaren Code zu schreiben
  - Ein- und Ausgabe jeder Prozedur spezifizieren !!!
  - Funktionalitäten in Prozeduren kapseln



## Übung 7

- Eine neue Klasse schreiben: TestBruch.java im *selben Verzeichnis* wie Bruch.java
- In main wird je Prozedur vom Bruchrechner eine Testmethode aufgerufen
- Darin werden jeweils positive (und negative) Tests der Prozedur realisiert
- Ggf. müssen Sie Ihren Code nochmal ändern, um ihn testbar zu machen

Hinweise:

- Aufruf der Methoden aus der andere Klasse im selben Verzeichnis mit Klassennahmen, z.B.:

```
int[] erg = Bruch.add(1,2,3,4);
erwarteGleichheit(erg[0], 5);
erwarteGleichheit(erg[1], 4);
```

- Nicht alle tests müssen erfolgreich sein!!!



## More to come

- Testen ist Thema in Software Engineering 2, Grundlagen des Software Testens und mehr



## Einführung OOP

1. Objekt, Instanz und Klasse
2. Methoden
3. Statisch vs. dynamisch
4. Vererbung
5. packages
6. Sichtbarkeit und Modifikatoren
7. Aufzählungs-Typen
8. Code-Dokumentation

## 4.1 Objekt, Instanz und Klasse

Objektorientierte Programmierung (OOP)

- Stand der Technik der Programmierung
- Ersetzt das Konzept von Programmen und Prozeduren durch interagierende Objekte
- Idee: Bessere Wiederverwendbarkeit durch Modellierung realer Objekte (nicht nur physische) als SW-Objekte
- Annahmen:
  - Die Welt besteht aus Objekten
  - Objekte sind klassifizierbar, haben Zustand und Verhalten



## 4.1 Objekt, Instanz und Klasse

Was ist ein Objekt?

- Ein Ding mit **Zustand** und **Verhalten**
- Eigenschaften definieren den **Zustand** des Objekts
- Fertigkeiten können
  - diesen Zustand ändern
  - Interaktionen mit anderen Objekten auslösen
  - Interaktionen anderer Objekte verarbeiten
- Durch seine Fertigkeiten wird das **Verhalten** des Objekts definiert



## 4.1 Objekt, Instanz und Klasse

Objekte sind Dinge der realen Welt

zum Beispiel:

- Der Hund Waldi
- Mein blaues Fahrrad
- Mein Warenkorb im Internet-Laden

Was haben diese Objekte für Zustände, welches Verhalten haben sie?



## 4.1 Objekt, Instanz und Klasse

Übung: Brainstorming

- Nennen Sie ein Objekt das im Hörsaal existiert
- Geben Sie dazu an:
  - Verhalten
  - Zustand



## 4.1 Objekt, Instanz und Klasse

Software Objekte sind Modelle von Dingen der realen Welt

- zum Beispiel: Java Bytecode der „den Hund Waldi“ repräsentiert
- Das Modell ist i.d.R abstrakter als das eigentliche Objekt (ist die Farbe von Waldis Augen für die Software wichtig?)



## 4.1 Objekt, Instanz und Klasse

Was ist ein Software-Objekt genau?

- Ein Softwaremodul mit **Zustand** und **Verhalten**
- Variablen definieren den **Zustand** des Objekts
- Methoden können diesen Zustand ändern oder Methoden anderer Objekte aufrufen
- Durch Methoden wird das **Verhalten** des Objekts definiert

## 4.1 Objekt, Instanz und Klasse

Was ist eine Klasse?

… eine Schablone für Objekte gleicher Art

zum Beispiel

- Hund
- Fahrrad
- Warenkorb



## 4.1 Objekt, Instanz und Klasse

Klassen definieren Eigenschaften und Verhalten

Zum Beispiel: zwei Hunde

| Klasse Hund | Objekt 1 | Objekt 2 |
|---|---|---|
| Attribut „Name“ | Waldi | Bello |
| Attribut „Farbe“ | braun | weiß |
| Methoden | Bellen, fressen, jagen | |



## 4.1 Objekt, Instanz und Klasse

Was ist eine SW-Klasse?

- SW-Klassen sind Schablonen für SW-Objekte
- Jedes SW-Objekt hat eine SW-Klasse
- Objekte der selben Klasse können sich durch ihren Zustand unterscheiden
- Klassen definieren das Verhalten all ihrer Objekte



## 4.1 Objekt, Instanz und Klasse

Klassen und Objekte in der OOP

- implementiert werden Klassen
- Klassen sind Typen für Objekte
- Jede Klasse hat mindestens eine Konstruktor-Methode zur Erzeugung neuer Instanzen (Objekte)
- Objekte kommunizieren indem sie Methoden anderer Objekte aufrufen



## 4.1 Objekt, Instanz und Klasse

Bsp: Klasse Fahrrad in Java

```
class Bicycle {
      int speed = 0;
      int gear = 1;
      void changeGear(int newValue) {
            gear = newValue;
      }
      void speedUp(int increment) {
            speed = speed + increment;
      }
}
```

Nicht jede Klasse ist eine Applikation!



## 4.1 Objekt, Instanz und Klasse

Klassen in Java

- Jedes Java-Programm besteht mindestens aus einer Klasse
- Syntax zur Deklaration von Klassen:

```
class <Klassenname> {
        <Implementierung>
}
```

## 4.1 Objekt, Instanz und Klasse

Implementierung von Klassen in Java

- Die Implementierung einer Klasse besteht aus
  - Deklarationen von Attributen
  - Definitionen von Methoden
  - Definitionen von Konstruktoren
- Attribute sind globale Variablen (Deklaration wie gewohnt)
- Methoden sind Prozeduren, die nicht statisch sind
- Konstruktoren sind…



## 4.1 Objekt, Instanz und Klasse

Übung (Plenum, Tafel)

- Definieren Sie eine java-Klasse für Objekte, die Modelle von Hunden sind.



## Wiederholung

- Was ist der Unterschied zwischen einem Objekt und einer Klasse?
- Wie wird der Zustand von Objekten in Java modelliert?
- Wie wird das Verhalten von Objekten in Java modelliert?

Übung im Plenum:

1. Schreiben Sie ein Programm, in dem zwei Rechtecke erzeugt (mit r = new Rechteck(2,3)), ihre Fläche berechnet und ausgegeben werden.
2. Definieren Sie eine dazu passende Klasse für Rechtecke in Java



## 4.1 Objekt, Instanz und Klasse

Konstruktoren in Java

- Konstruktoren sind Methoden mit dem selben Namen wie ihre Klasse
- Sie erzeugen Objekte ihrer Klasse
- Haben keinen Rückgabe-Typ
- Durch Deklaration der Klasse ist ein *„Standard-Konstruktor“* (ohne Parameter) bereits implementiert
- Dieser und weitere Konstruktoren können in der Klasse implementiert werden



## 4.1 Objekt, Instanz und Klasse

Bsp: Klasse Fahrrad mit 2 Konstruktoren

```
class Bicycle {
      int speed;
      int gear;
      Bicycle(){
            speed = 0; gear = 1;
      }
      Bicycle(int s, int g){
            speed = s; gear = g;
      }
      …
}
```



## 4.1 Objekt, Instanz und Klasse

Übung

- Implementieren Sie zwei Konstruktoren für Ihre Klasse für Rechtecke.

## 4.1 Objekt, Instanz und Klasse

Klassen als Typen in Java

- Jede Klasse ist ein Typ
- man kann Variablen zu diesem Typ deklarieren: `MyClass var;`
- Defintion dieser Variablen durch Objekte : `var = objVonMyClass;`
- Erzeugen von Objekten mit
  `new <Konstruktor>`
  Bsp: `var = new MyClass();`



## 4.1 Objekt, Instanz und Klasse

Objekte in Java

- Sind Instanzen einer Klasse
- Entstehen durch Konstruktoren,
  z.B. `Hund h = new Hund();`
- haben einen eindeutigen Namen
- der Name ist eine Referenz auf das Objekt



## 4.1 Objekt, Instanz und Klasse

Beispiel: Ein Programm, das zwei Bicycle-Objekte erzeugt

```
class Application{
     public static void main(String[] a){
          Bicycle b1 = new Bicycle();
          Bycicle b2 = new Bicycle(10,24);
     }
}
```



## 4.1 Objekt, Instanz und Klasse

Übung

- Implementieren Sie ein ausführbares Objektorientiertes Programm, dass zwei Objekte x uns y der Klasse „Bruch“ erzeugt, die die Werte 1/3 und 2/5 modellieren.



## 4.1 Objekt, Instanz und Klasse

Übung 8 (bis 13): Minesweeper

- Erster Schritt: „Programmlogik“ ohne Nutzer-Interaktion
- Zwei Klassen (also zwei Dateien): „Feld“ und „Spielfeld“
- Alle nötigen Attribute sind in der Aufgabe spezifiziert
- Konstruktor für Klasse Spielfeld:
  - N x M Feld-Objekte erzeugen und in Matrix abspeichern
  - Mienen zufällig verteilen und Zahlen in Nachbarfelder eintragen
- Hier gibt es keine main(), Sie können aber eine zum Ausprobieren implementieren
- Ausführen (mit main) mit Test: Übung 9



## 4.1 Objekt, Instanz und Klasse

Schriftkonventionen in Java

- Klassennamen beginnen mit Großbuchstaben
- Objektnamen beginnen mit Kleinbuchstaben
- Attributsnamen und Methodennamen beginnen mit Kleinbuchstaben
- Zusammengesetzte Wörter durch Großbuchstaben (sog. CamelCase) : z.B.
  - meineMethode():
  - MeineKlasse
  - meinObjekt

# 4. Grundlagen OOP

1. Objekt, Instanz und Klasse
2. Methoden
3. Statisch vs. dynamisch
4. Vererbung
5. packages
6. Sichtbarkeit und Modifikatoren
7. Aufzählungs-Typen
8. Code-Dokumentation



# 4.2 Methoden

- Methoden sind Prozeduren von Objekten (und werden in den Klassen implementiert)
- Sie können den Zustand des Objektes verändern
- Die Methoden einer Klasse definieren das Verhalten ihrer Objekte
- Ihre Semantik ist i.d.R. abhängig vom Zustand des Objekts



# 4.2 Methoden

Aufruf von Methoden durch Objekte

Wenn in der Klasse von `<Objekt>` eine Methode `<Methode>` definiert ist, so kann diese aufgerufen werden durch:

```
<Objekt>.<Methode>(<Parameter>);
```



# 4.2 Methoden

Übung im Plenum

Erstellen Sie die Hunde Wuffi und Bello als Objekte der Klasse Hund. Sie wiegen beide anfangs 10 kg.

Füttern Sie dann die Hunde mit je 1 kg Futter und lassen sie anschließend bellen.



# 4.2 Methoden

Verschachtelter Aufruf

- Der Aufruf einer Methode steht für ihren Rückgabewert (falls nicht `void`)

  Bsp:

```
obj.gibAnderesObj().machWas();
if (x > obj.getZahl())…
```



# 4.2 Methoden

Übung, Murmelgruppe

Implementieren Sie die Methode void add(Bruch b) in einer Klasse, die Brüche modelliert. Schreiben Sie dann ein Programm, das die Summe von 1/2 und 1/3 berechnet und an der Konsole ausgibt.

## 4.2 Methoden

Aufruf von Methoden innerhalb einer Klasse

- Jedes Objekt hat Referenz auf sich selbst mit Schlüsselwort `this`
- Aufruf von Methoden des selben Objekts mit `this.methode();`
- Das „`this`" wird als default angenommen und kann weggelassen werden.



## 4.2 Methoden

Übung (Einzeln)

Implementieren Sie Ihre Methode add(…) für Brüche so, dass nach der Addition immer gekürzt wird.

Sie können dabei eine Implementierung von void kuerzen() in der Klasse Bruch voraussetzen



## Wiederholung

- Wie werden in Java Objekte erzeugt?
- Wie wird die Semantik der Methoden eines Objekts in Java definiert?
- In welchem Fall kann sich die Semantik einer Methode der selben Klasse unterscheiden?
- Wie werden Methoden eines Objekts aufgerufen?



## Übung mit Mixgruppen

1. Bilden Sie Gruppen a 4 Personen
2. Formulieren Sie schriftlich als Gruppe Anforderungen an eine beliebige Software-Klasse (10 min):
   - Was modelliert die Klasse?
   - Welche Methoden und Konstruktoren stellt sie zur Verfügung?
3. Entsenden Sie ein Gruppenmitglied mit den Anforderungen in eine andere Gruppe, es erklärt dort die Anforderungen
4. Implementieren Sie eine Klasse, zu den Anforderungen, die ihnen erklärt wurden (10 min)
5. Entsenden Sie ein anderes Gruppenmitglied in eine andere Gruppe, das dort Anforderungen und Implementierung erklärt
6. Implementieren Sie eine Testklasse zu der Implementierung, die ihnen erklärt wurde (10 min)



## 4.1 Objekt, Instanz und Klasse

Übung 9 Tests für Minesweeper

- Definieren Sie eine Testklasse (mit main), die Ihre Klassen Spielfeld (und ggf. Feld) testet
- Für jede Methode eine Testmethode, in denen ein neues Testobjekt erzeugt wird.
- Unterschiedliche Testfälle
- Insbesondere:
  - Ist die Anzahl der Minen korrekt
  - Werden die Zahlen richtig berechnet (hier müssen Sie Spielfelder ohne Zufall mit Minen bestücken.)



## 4.2 Methoden

Parameterübergabe in der Programmierung

- Call by Value: es wird ein Wert übergeben
  - Eine **Kopie** wird übergeben, möglicherweise verändert und dann **verworfen**
- Call by Reference: es wird eine Referenz übergeben
  - Die Adresse der Speicherstelle wird übergeben
  - In der Methode kann der Wert, der sich hinter der Adresse verbirgt **nachhaltig verändert** werden

## 4.2 Methoden

Parameterübergabe in Java

- immer Call by Value, aber es gibt einen Unterschied
  - bei Grundtypen wird einfach der Wert kopiert
  - bei Referenz-Typen (Objekten) wird die Referenz kopiert, sie zeigt auf das selbe Objekt. Effekt: wie Call by Reference



## 4.2 Methoden

Experiment: Parameter.java

Was gibt dieses Programm aus?



## Übung (zur Selbstreflexion)

```
class Hello{
        int attr = 7;
        public static void main(String[] args){
                int x = 43; int[] y = {1,2,3};
                Hello o = new Hello(); String s ="Hallo";

                m(x,y,o,s);
                // welche Werte haben  hier x, y, o.attr und s ?
        }
        static void m(int x, int[] y, Hello o, String s){
                x = 5;
                y[0] = 5;
                o.attr = 3;
                s = "X";
        }
}
```



## 4.2 Methoden

Parameterübergabe in Java

- Verhinderung des „Call-by-Reference-Effekts“ durch explizites Kopieren des Objekts
- Standard-Methode zum Kopieren: `clone()`
- Alternativ Kopierkonstruktor: `K(K x){…}`
- Bspe: „echtes“ Call-by-value:

```
callByValueMethod(obj.clone());
callByValueMethod(new K(obj));
```



## 4.2 Methoden

Übung im Plenum

- Verhindern Sie den Call-by-Reference Effekt im Programm Parameter.java



## Wiederholung

- Was ist bei der Übergabe von Objekten als Parameter in Java zu beachten?
- Wie kann man den „Call-by-value-Effekt“ von verhindern?

## 4.2 Methoden

Zugriff auf Attribute

- Auf Attribute kann von außen wie auf Methoden zugegriffen werden

Bsp:
```
obj.attribut = 5; //schreiben
x = obj.attribut; // lesen
```

- Besser ist aber das nicht zu tun, sondern setter und getter zu verwenden:

Bsp:
```
obj.setAttribut(5)
x = obj.getAttribut();
```



## 4.2 Methoden

Bsp Implementierung setter/getter

```
class X{
   int attr;
   void setAttr(int a){attr = a;}
   int getAttr(){ return attr;}
   …
}
```



## 4.2 Methoden

Übung Einzeln

Implementieren Sie getter und setter für ein Attribut Ihrer Klasse Hund.



## 4. Grundlagen OOP

1. Objekt, Instanz und
2. Methoden
3. Statisch vs. dynamisch
4. Vererbung
5. packages
6. Sichtbarkeit und Modifikatoren
7. Aufzählungs-Typen
8. Code-Dokumentation



## 4.3 Statisch vs. Dynamisch

Klassen sind statisch

- Klassen werden implementiert und bestehen fortan
- Methoden und Attribute können auch Klassen zugeordnet werden
- Solche Klassenvariablen und Klassenmethoden sind statisch
- Markierung in Java mit „`static`“
- von statischen Methoden ist kein Zugriff auf dynamische Methoden/Attribute möglich



## 4.3 Statisch vs. Dynamisch

Klassenmethoden

- Deklaration mit

```
static <Typ> <Methode>(<Par>){…}
```

- Benutzung mit Klassenname (oder selbe Klasse)

```
<Klasse>.<Methode>(par);
```

z.B. `x = Integer.parseInt(y);`

- Können aus statischem oder dynamischen Kontext aufgerufen werden
- sinnvoll für „klassische Bibliotheken“

## 4.3 Statisch vs. Dynamisch

Klassenvariablen

- Deklaration mit

```
static <Typ> <Bezeichner>;
static <Typ> <b1>,<b2>,…,<bn>;
```

- Definition von außen mit setter (empfohlen) oder …
- …mit Klassenname

```
Klasse.<Bezeicher> = 5; //schreiben
x = Klasse.<Bezeichner>; //lesen
```

- **Achtung: alle Objekte einer Klasse nutzen so die selbe Variable**



## 4.3 Statisch vs. Dynamisch

Übung, gemeinsam

Erzeugen Sie zwei Objekte der Klasse Hund mit unterschiedlichen Gewichten.

Deklarieren Sie das Attribut „gewicht“ in der Klasse Hund als statisch.

- Was passiert?



## 4.3 Statisch vs. Dynamisch

Kommunikation zwischen Objekten mittels Klassenvariablen

- Da alle Objekte Zugriff auf die selbe Variable haben kann darüber Information ausgetauscht werden
- Beispiel: Fortlaufende, eindeutige Nummern vergeben



## 4.3 Statisch vs. Dynamisch

Objekte sind dynamisch

- Instanzvariablen und Instanzmethoden beziehen sich auf Objekte
- Ihre Benutzung ist immer mit einem Objekt verbunden
- Objekte haben eine kürzere Lebensdauer als die Dauer der Ausführung des Programms
- daher nennt man diese Methoden und Attribute dynamisch
- dynamisch ist der Normalfall und braucht keinen Modifikator



## 4.3 Statisch vs. Dynamisch

Jedes Objekt hat seine eigenen…

- Attribute: Jede globale Variable einer Klasse (die nicht statisch ist) hat jedes Objekt dieser Klasse für sich selbst einmal
- Methoden sind ohnehin für alle Objekte gleich, aber das Verhalten kann von Attributen abhängig sein



## 4.3 Statisch vs. Dynamisch

Dynamisch ist der Normalfall

- Statische Methoden/Attribute dienen den genannten speziellen Zwecken.
- Sonst sind sie nicht angebracht

- Normalerweise:
  - Erzeugung eines Objektes als einziger Befehl in main(…)
  - Dann gibt es keinen weiteren statischen Kontext

## 4.3 Statisch vs. Dynamisch

Konstanten

- Sind „Variablen“, die nach ihrer Initialisierung nicht mehr veränderbar sind
- Deklaration:
  `final <Typ> <Bezeichner>`
- Einmalige Definition und Benutzung wie üblich
- Schriftkonvention: Konstanten werden mit Großbuchstaben bezeichnet



## 4.3 Statisch vs. Dynamisch

Konstanten (forts.)

- Konstanten können statisch oder dynamisch sein

Bsp:
```
final static int MAXVAL = 1000;
```

- Spätestens im Konstruktor müssen sie definiert werden

Bsp:
```
final int MAXVAL;// nicht statisch
K(int x ){ MAXVAL = x;}
```

- Achtung: Konstanten die Objekte bezeichnen haben eine unveränderliche Referenz (das Objekt kann sich ändern)



## Übungsaufgabe 10

Implementieren Sie eine Methode linksKlick(int y, int y) in der Klasse Spielfeld

- Zustand des gewählten Feldes ändert sich: offen= true
- GGf wird Methode rummms() aufgerufen, wenn man eine Miene getroffen hat
- Wenn nachbarMienen==0, dann alle 3-8 Nachbarn „durch Programm klicken“ (Rekursion)
- Test implementieren auf (nicht-zufälligem) Spielfeld:
  - Werden alle Nachbarn geöffnet?
  - Werden größere Bereiche mit Nullen freigelegt?



## Wiederholung

- Was unterscheidet statische Methoden von „normalen“ Methoden?
- Was unterscheidet statische Attribute von „normalen“ Attributen?



## 4. Grundlagen OOP

1. Objekt, Instanz und
2. Methoden
3. Statisch vs. Dynamisch
4. **Exkurs: GUI Implementierung mit javaFX**
5. Vererbung
6. packages
7. Sichtbarkeit und Modifikatoren
8. Aufzählungs-Typen
9. Code-Dokumentation



## 4. Graphische Benutzerschnittstellen

1. Grundideen JavaFX
2. Container und Controls, Layout
3. Ereignisse

## 4.1 Grundidee JavaFX

Entwicklung

- 2008 erste Impl. als Alternative zu zu HTML/JavaScript → kaum genutzt
- 2011 Realisierung in Java (keine Scripts) als Alternative zu javax.swing
- 2014: JavaFX 8: Standard UI-Technologie für Desktop-Anwendungen **ab Java 8**



## 4.1 Grundidee JavaFX

Realisierung: Klicki-Bunti, XML oder Java?

- *Java*: Implementierung auf Basis der Programmierschnittstelle (API) mit klassischem Java
- *FXML*: Konfiguration der GUI in XML
- *Klicki-Bunti*: Diverse graphische Editoren erhältlich, zB JavaFX Scene Builder



## 4.1 Grundidee JavaFX

Unsere Lernziele (Kompetenzen):

- Umgang mit Objekten, Klassen, Methoden
- Benutzung von Programmierschnittstellen (APIs)
- Grundfertigkeiten zur UI-Erstellung



## 4.1 Grundidee JavaFX

Fallstudie

Wir Implementieren ein interaktives Sudoku mit Java FX

…und lernen dabei, was alles zu tun ist.

Sie lernen dann durch „Selbermachen“ in den Übungen und im Selbststudium



## Wiederholung

1. Was unterschiedet statische von nicht-statischen Attributen in Objekten?
2. Welchen Vorteil bieten statische Methoden?
3. Was ist in statischen Methoden nicht möglich?
4. Was sollte in der OOP normalerweise nur statisch sein und wie erreicht man, dass nichts anderes statisch sein muss?
5. Wie heißt die derzeit aktuelle Technologie für GUIs in Java



## 4. Graphische Benutzerschnittstellen

1. Grundidee
2. Container und Controls, Layout
3. Ereignisse

## 4.1 Grundidee JavaFX

Fallstudie Sudoku

1. Mache eine Skizze auf Papier - wirklich!
2. Definiere einen top-level Container
3. Definiere weitere Container nach Bedarf
4. Definiere alle Controls (Buttons etc)
5. Definiere Event-Listener für Nutzer-Interaktion



## 4.1 Grundlagen

Fallstudie Sudoku

1. Mache eine Skizze auf Papier - wirklich!
2. Definiere einen top-level Container
3. Definiere weitere Container nach Bedarf
4. Definiere alle Controls(Buttons etc)
5. Definiere Event-Listener für Nutzer-Interaktion



public static void main(String[] args) { launch(args); }

## 4.2 Container und Controls, Layout

`Application` und `Stage`

- Die Schablone für interaktive Anwendungen, Bsp:

```
import javafx.application.Application;
import javafx.stage.Stage;

public class MyAppl extends Application{
      public void start(Stage stage){
            stage.setTitle("MyAppl");
            // Hier kommt das Programm
            stage.show();
      }
      public static void main(String[] a){
            launch(a);
      }
}
```



## 4.2 Container und Controls, Layout

Anwendung im Plenum: wir implementieren das Soduku GUI



## 4.2 Container und Controls, Layout

Stage, „Scene“ und verschiedene „Pane“

- „Stage“ bezeichnet das Fenster an sich
- „Scene“ ist der Fensterinhalt
- „…Pane“ ist ein Container für Controls oder andere Container
- Verschiedene Arten „…Pane“ ordnen die „Controls“ unterschiedlich an

```
stage.setTitle("MyAppl");
BorderPane root = new BorderPane();
// TODO
Scene scene = new Scene(root);
stage.setScene(scene);
stage.show();
```



## 4.2 Container und Controls, Layout

Verschiedene Layouts

- Controls (Buttons etc) werden vom Layout-manager angeordnet
- Layout-Manager sind in „…Pane“ Klassen implementiert
- In javafx.scene.layout gibt es BorderPane, FlowPane, GridPane, StackPane etc
- Beispiele auf docs.oracle.com/javase/8/javafx/layout-tutorial/builtin_layouts.htm

## 4.2 Container und Controls, Layout

Anwendung im Plenum: wir implementieren das Layout für das Soduku GUI



## 4.1 Grundlagen

Fallstudie Sudoku

1. Mache eine Skizze auf Papier - wirklich!
2. Definiere einen top-level Container
3. Definiere weitere Container/Menüs nach Bedarf
4. Definiere alle Controls (Buttons etc)
5. Definiere Event-Listener für Nutzer-Interaktion



## 4.2 Container und Controls, Layout

Menu-Leisten sind auch Container, Bsp

```
BorderPane root = new BorderPane();
MenuBar menuBar = new MenuBar();
Menu menu = new Menu("Datei");
MenuItem item = new MenuItem("Beenden");

menu.getItems().addAll(item);
menuBar.getMenus().addAll(menu);
root.setTop(menuBar);
```



## 4.2 Container und Controls, Layout

Anwendung im Plenum:

wir implementieren ein Menü für das Soduku GUI und zeigen es am oberen Rand der GUI an



## 4.1 Grundlagen

Fallstudie Sudoku

1. Mache eine Skizze auf Papier - wirklich!
2. Definiere einen top-level Container
3. Definiere weitere Container nach Bedarf
4. Definiere alle Controls (Buttons etc)
5. Definiere Event-Listener für Nutzer-Interaktion



## 4.2 Container und Controls, Layout

GUI Komponenten: Controls

- Es gibt eine große Auswahl an fertigen Controls: Buttons, Slider etc
- Quelle: docs…/ui_controls.htm
- Man kann jedem …Pane Controls hinzufügen:

```
gridPane.setCenter(new Label("Hello"));
hbox.getChildren().addAll(new Button(„x"));
```

## 4.2 Container und Controls, Layout

Anwendung im Plenum: wir implementieren die Textfelder in einem GridPane für das Soduku GUI



## Quellen

- docs.oracle.com/javafx → Das Original
- Viele andere Tutorials in vielen anderen Sprachen im Internet, beachten Sie:
  - JavaFX **8** (oder höher, nicht 2.0 oder so)
  - Kein CSS
  - Ausprogrammieren, kein FXML oder Abhängigkeit von einer speziellen IDE
  - Erlaubt ist was gefällt, Ziel: selber GUIs erstellen können



## Übungsaufgabe

*Bis KW 2*

*Implementieren Sie eine graphische Oberfläche mit javafx mit einem Button für jedes Feld. Die Größe des Spielfeldes und die Anzahl der Minen soll dabei in einem Menü definierbar sein.*

- Neue Klasse mit start realisieren
- Darin wird ein GUI-Objekt Stage und 3 Panes erzeugt und angezeigt
  - Hbox für Menü
  - GridPane für Spielfeld
  - BorderPane für beides zusammen (analog Sudoku)
- Implementieren Sie nur die GUI, noch keine Interaktivität!!!



## Wiederholung

1. Wie werden Kontrollelemente in GUIs angeordnet, so dass sie auf unterschiedlichen Bildschirmen gut lesbar sind?
2. Wie kann man herausfinden, welche Kontrollelemente in JavaFX zur Verfügung stehen und wie man sie einsetzt?



## 4. Graphische Benutzerschnittstellen

1. Die swing-API
2. Container und Controls, Layout
3. Ereignisse



## 4.3 Ereignisse

Was ist ein Ereignis?

- Ereignisse lösen Aktionen bzw. Zustands-Veränderungen „von außen“ aus
- Z.B. Benutzereingaben
  - Maus-Klick, Tastatur-Eingabe, Geräte-Anschluss
- Z.B. System-Ereignisse
  - Zeitpunkt eingetreten, Fehler, Datenveränderung, Sensor-Eingabe

## 4.3 Ereignisse

Ereignisse in GUIs

- Viele GUI-Komponenten „warten“ auf Ereignisse (Benutzereingaben)
  - Menüs, Buttons, Scroll-Bars, Listen …
- Es existiert keine vorhersehbare Reihenfolge der Ereignisse (Nebenläufigkeit)
- In javaFX sind Ereignisse Objekte (z.B. der Klassen ActionEvent, KeyEvent, MouseEvent …)



## 4.3 Ereignisse

Ereignis-Handler registrieren

- Der Eintritt eines Ereignisses der Klasse **`X`** wird von der JVM registriert und meldet das an alle *registrierten* Objekte der Klasse **`EventHandler<X>`**
- Der EventHandler hatte sich zuvor als Interessent für die Events registriert:

**`node.addEventHandler`**(<Event>,<Handler>**`)`**

**`node.setOnAction`**(<Handler>)

- dann wird die Methode **`handle(X x)`** in diesen EventHandler-Objekten aufgerufen (callback)



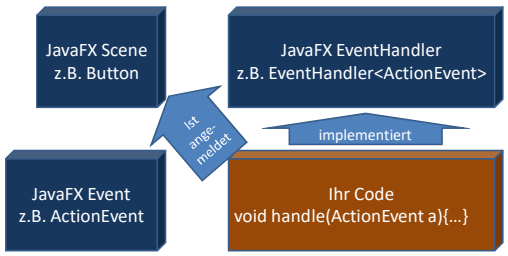
## 4.3 Ereignisse

Callback Methoden

- Werden vom Entwickler geschrieben
- Werden aber nicht durch seinen code aufgerufen
- Nach dem Hollywood-Prinzip: „Don‘t call us, we‘ll call you.“

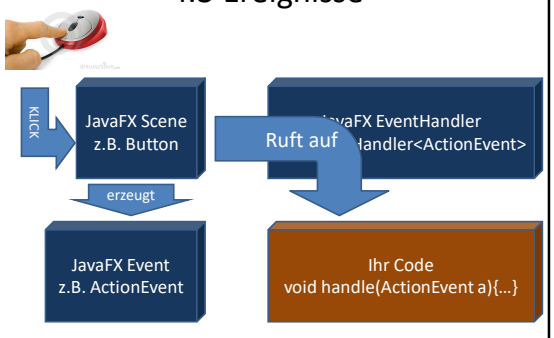


## 4.3 Ereignisse






## 4.3 Ereignisse






## 4.3 Ereignisse

Bsp: EventHandler (als „anonyme innere Klasse“)

```
import javafx.event.*;
…
…
Button button = new Button();
button.setOnAction(new EventHandler<ActionEvent>(){
    public void handle(ActionEvent e){
      // Ihr Code: Wir ausgeführt bei Button-Klick
    }
});
…
```

## 4.3 Ereignisse

Fallstudie

Ich habe Hausaufgaben gemacht:

- Menu eingebaut
- Logik implementiert
- Besser strukturiert

Jetzt machen wir gemeinsam

- Implementieren Sie im Sudoku Programm EventHandler für
  - die Menu-Einträge
  - Erster Versuch: die Textfelder im Spielfeld



## 4.3 Ereignisse

Pitfall: in „inneren Klassen“ final-Referenzen

- Die Methode handle(…) wird dem Interaktionsobjekt (zB Button) zugeordnet *lange bevor* sie aufgerufen wird
- In der Zwischenzeit könnten sich Variablenwerte, die sie verwendet, ändern
- Fehlermeldung des compilers: „… referenced from inner class must be final…“
- Lösung: Variablen **`final`** deklarieren



## 4.3 Ereignisse

Fallstudie

Implementieren Sie nun EventHandler für die TextFelder im Sudoku



## 4.3 Ereignisse

Verschiedene Ereignisse

- EventHandler bzw. addEventHandler( (<Event>,<Handler>**) {**…**}** gehen immer
- Aber alle javafx-Controls haben vereinfachte Handler, z.B.
  - setOnMouseClicked(MouseEvent e) → bei klick
  - setOnMouseEntered(MouseEvent e) → Berührung durch Mauszeiger
  - setOnKeyPressed(KeyEvent e) → Taste gedrückt
  - U.v.m setOn“…“(…Event e) → …



## 4.3 Ereignisse

Fallstudie

- Implementieren Sie die Eingabe von Zahlen im Sudoku so, dass
  - man nicht auf „return“ drücken muss



## 4.3 Ereignisse

Immer nur ein Ereignis in Java!

- In Java kann immer nur ein Ereignis auf einmal verarbeitet werden
- Wenn dort ein (synchroner) Methodenaufruf stattfindet kann das nächste Ereignis erst verarbeitet werden, wenn die Methode fertig ist.

## 4.3 Ereignisse

Fallstudie

- Wie verhält sich das Programm, die Verarbeitung eines Event nicht terminiert?



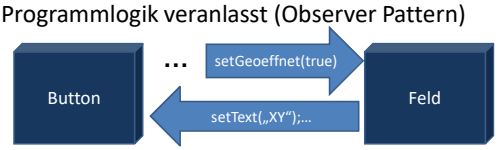
## Übungsaufgabe 12

1. Verbinden von GUI und Logik: in GUI.start() wird ein Spielfeld-Objekt erzeugt
2. Implementierung von EventHandler für Buttons
   - Mit „inneren anonymen Klassen“ wie im Sudoku Beispiel
   - Darin Aufruf der Methode linksKlick(i,j) einer Spielfeld Instanz
3. Registrierung der Handler in Logik für Ausgabe
   - Machen wir jetzt gleich: Observer
4. Implementierung der Handler für MenuItems



## Ausgabe in GUIs

- JavaFX-Scenes können ihr Aussehen ändern (z.B. button.setText(…))
- Dies soll als Reaktion auf Eingaben geschehen
- Konzept: Diese Änderungen werden aus der Programmlogik veranlasst (Observer Pattern)






## 4.3 Ereignisse

Fallstudie

- Implementieren Sie die EventHandler im Sudoku Programm so, dass bei Konflikten (zB zweimal die selbe Zahl in einer Zeile) beide Werte rot eingefärbt werden.

Selbststudium:

- Verbessern Sie die Implementierung, so dass man Sudoku spielen kann



## 4. Grundlagen OOP

1. Objekt, Instanz und
2. Methoden
3. Statisch vs. Dynamisch
4. Exkurs: GUI Implementierung mit javaFX
5. Vererbung
6. packages
7. Sichtbarkeit und Modifikatoren
8. Aufzählungs-Typen
9. Code-Dokumentation



## Übungsaufgabe 12 (bis nächste Wo.)

1. Verbinden von GUI und Logik: in GUI.start() wird ein Spielfeld-Objekt erzeugt
2. Implementierung von EventHandler für Buttons
   - Mit „inneren anonymen Klassen“ wie im Sudoku Beispiel
   - Darin Aufruf der Methode linksKlick(i,j) einer Spielfeld Instanz
3. Registrierung der Handler in Logik für Ausgabe
4. Implementierung der Handler für MenuItems

## Prüfungsaufgaben

1. Vermischtes: 40 P. (das ist die schwierigste Aufgabe)
   - 5 Fragen zu Code-Beispielen: Was ist die Ausgabe?
   - 10 Fragen zu Themen der VL auch mit Code-Beispielen
2. Imperative Programmierung: 30 P.
   - Eine statische Methode, die eine Berechnung durchführt. Kompetenzen: Variablen, if, for, while, „algorithmische Tricks“
3. Testen: 20 P.
   - Ein Test für eine Klasse ist gegeben: Sie müssen die Klasse und ihre Methoden implementieren. Kompetenz: Verhalten von Objekten verstehen und implementieren
4. OOP: 30 P.
   - Zwei Klassen implementieren, die gegebene Anforderungen erfüllen. Kompetenzen: OOP-Vokabular, Umgang mit Objekten



## Klausurvorbereitung

1. Organisatorisches, Ablauf
2. Aufgaben - Typ und Punkte
3. Übung
   1. Ihre Fragen, die alle interessieren (sonst: Übung oder nach Vereinbarung)
   2. Lösungen zur Probeklausur (und Ihre Fragen dazu)



## Abschluss Inhalte ???

Jetzt besser OOP üben, keine weitere Inhalte für das Semestetr?



## 4.4 Vererbung

Was ist Vererbung?

- Attribute und Methoden einer Klasse werden auf eine andere übertragen (vererbt)
- Implementiert das Konzept der **Generalisierung/Spezialisierung** von Objekten in ihren Klassen
- Die Generalisierung heißt *Superklasse*, die Spezialisierung *Subklasse*
- Vererbung erzeugt „**ist-ein**“-Hierarchien



## 4.4 Vererbung

Übung im Plenum: Klausuraufgabe

Es soll ein System zur Bearbeitung geometrischer Formen implementiert werden.

*(6 Punkte) geben Sie die Vererbungshierarchie folgender geometrischer Formen an: Viereck, Quadrat, Strecke, Mehreck, Punkt, Dreieck, Form*



## 4.4 Vererbung

Vererbung in Java

- Erstellung einer Subklasse in ihrer Deklaration:

```
class <Spez.> extends <Gen.> {
  <Implementierung>
}
```

- Damit stehen alle Methoden und Attribute der Superklasse in this zur Verfügung
- Bsp. aus JavaFX

```
class MyGui extends Application{…}
```

# 4.4 Vererbung

Fallstudie im Plenum: Klausuraufgabe

*(22 Punkte) Implementieren Sie Java-Klassen für **alle** oben genannten Formen, so dass*

- *in einem Punkt die x und y-Koordinaten des Punktes gespeichert sind,*
- *in einer Strecke Anfangs- und Endpunkt gespeichert sind,*
- *in Mehrecken alle Seiten als Strecken gespeichert sind,*
- *alle Streckenlängen durch eine Methode berechnet werden können bei allen Mehrecken der Umfang berechnet werden kann, ($\sqrt{(x1-x2)^2+(y1-y2)^2}$)*
- *…*



# 4.4 Vererbung

Übung in Kleingruppen

Erweitern Sie die Implementierung der geometrischen Formen so, dass auch Flächeninhalte für alle Formen berechnet werden können.

(Hinweis: der Flächeninhalt eines Dreiecks kann mit dem Satz von Heron aus den Seitenlängen berechnet werden)



# 4.4 Vererbung

Nutzung der Vererbung

Eine Spezialisierung hat alle Methoden und Attribute all seiner Superklassen.

z.B.

```
class Quadrat extends Rechteck{}
class Rechteck{ double umfang(){…}}
…
Quadrat q = new Quadrat();
double u = q.umfang();
```



# 4.4 Vererbung

Übung in Kleingruppen

Implementieren Sie eine Anwendung, die ein Quadrat und ein Dreieck erzeugt und deren Umfang und Flächeninhalt ausgibt.



# 4.4 Vererbung

Vererbung und Konstruktoren

- Konstruktoren werden nicht vererbt
- impliziter Aufruf des Standard-Konstruktors der Superklasse vor dem Aufruf des Konstruktors der Subklasse
- Oder: expliziter Aufruf des Konstruktors der Superklasse mit `super(<par>)` möglich
- `super(<par>)` muss immer die erste Anweisung eines Konstruktors sein



# 4.4 Vererbung

Programmierdemo

- Implementieren Sie Konstruktoren für alle Klassen zu den geometrischen Formen ohne duplizierten Code
- Realisieren Sie in jedem Konstruktor eine Ausgabe an der Konsole
- Erzeugen Sie die Objekte und analysieren Sie, welche Konstruktoren ausgeführt werden

## 4.4 Vererbung

Überschreiben von Methoden

- Instanzmethoden (also nicht static) können von Subklassen überschrieben werden
- Eine Methode die die gleiche Deklaration (Modifikatoren, Typ und Name) hat wie die einer Superklasse überschreibt die Methode der Superklasse
- Die Unterscheidung findet nur an Name und Parametern statt. Nicht am Rückgabetyp.
- Zugriff auf Methoden der Superklasse mit der Referenz `super`

Bsp: `super.methode(par);`



## 4.4 Vererbung

Programmierdemo

- Erweitern Sie die Klassen für geometrische Formen so, dass alle Objekte dieser Klassen Methoden zur Berechnung von Umfang und Fläche haben
- Überschreiben Sie dazu ggf. die Methoden in Subklassen



## 4.4 Vererbung

Überschreiben von Attributen

- Ist genauso möglich wie das Überschreiben von Methoden
- Ist aber im Ggs. dazu nicht zu empfehlen



## 4.4 Vererbung

`super` und `this`

- Sind beides Referenzen auf die aktuelle Instanz
- `super` beschreibt den geerbten Teil, `this` die Instanz als Ganzes
- Beide Referenzen können i.d.R. weggelassen werden, Suchreihenfolge: erst `this`, dann `super`



## 4.4 Vererbung

```
TODO: Folie zu Konstruktor
this() und Methoden aufrufen
der Methode der Superklasse
```



## 4.4 Vererbung

Überschreiben vs. Verstecken

- Instanzmethoden überschreiben Methoden der Superklasse, die den selben Namen und die selben Parametertypen haben
- Instanzvariablen überschreiben Attribute der Superklasse, die den selben Namen und den selben Typ haben
- Klassenmethoden oder Klassenattribute mit selben Modifikator/Name/Typ verstecken die geerbten Anteile
- Versteckete Anteile können mit dem Klassennamen der Superklasse erreicht werden

# 4.4 Vererbung

Überblick: Überschreiben und Verstecken

Defining a Method with the Same Signature as a Superclass's Method

| | Superclass Instance Method | Superclass Static Method |
|---|---|---|
| Subclass Instance Method | Overrides | Generates a compile-time error |
| Subclass Static Method | Generates a compile-time error | Hides |

[Oracle, java Tutorial]



# 4.4 Vererbung

Covariant Return Types

- Seit Java 7: Spezialisierte Klassen können Methoden überschreiben, wenn der Rückgabetyp die selbe Spezialisierung ist
- Beispiel, die Methode getElement wird mit Covariant Return Type überschrieben:

```
class A{   A getElement(){…}}
class B extends A{ B getElement(){…}}
```



# Organisatorisches

- Am 25.1.: Klausurvorbereitung, nehmen Sie Teil!
- Prüfungsvorleistung: 11 von 13 Aufgaben im Termin präsentiert…
- Probeklausur auf LV-website
- Am Fr., 2.2. 8.00 Uhr, GI/1.01 Klausur



# Wiederholung

„Ein Pinguin ist ein Vogel“

1. Was ist hier Superklasse und Subklasse?
2. Wie kann man die Klassen in Java implementieren?
3. In welche Klasse gehört das Attribut „alter“?
4. In welche Klasse gehöret die Methode „schwimmen“?
5. Wie werden hier Konstruktoren definiert, mit denen man das Alter des Vogels festlegen kann?
6. In welche Klasse gehört die Methode „fliegen“



# 4. Grundlagen OOP

1. Objekt, Instanz und
2. Methoden
3. Statisch vs. Dynamisch
4. Exkurs: GUI Implementierung mit javaFX
5. Vererbung
6. packages
7. Sichtbarkeit und Modifikatoren
8. Aufzählungs-Typen
9. Code-Dokumentation



# 4.4 Vererbung

Dynamische Bindung

- In Programmen soll manchmal bei der Implementierung nicht entschieden werden, welche Spezialisierung zur Laufzeit verwendet wird
- Man deklariert dann eine Variable der Superklasse definiert sie mit einem Objekt der Subklasse
- Die Implementierung aus der Subklasse wird dann zur Laufzeit „dynamisch gebunden“

→ Dynamische Bindung ist m.E. der größte Vorteil der OOP

## 4.4 Vererbung

Dynamische Bindung, Beispiel 1

```
class GeometrieUI{
 static void printFlaeche(Mehreck f){
      System.out.println("Flaeche: "+f.flaeche());
 }
}
…
Quadrat q = new Quadrat(a,b,c,d);
Dreieck d = new Dreieck(e,f,g);
GeometrieUI.printFlaeche(q);
GeometrieUI.printFlaeche(d);
```



## 4.4 Vererbung

Dynamische Bindung, Beispiel 2

```
class Schach{…
  Spieler[] spieler = new Spieler[2];
  if(zweiSpielerModus){
      spieler[0] = new SpielerGUI();
      spieler[1] = new SpielerGUI(); }
  if(einSpielerModus){
      spieler[0] = new SpielerGUI();
      spieler[1] = new ComputerSpieler();
  }…
  while(…) spieler[i].zugMachen();
```



## 4.4 Vererbung

Übung im Plenum

Wir erweitern das Sudoku mittels dynamischer Bindung so,

1. dass zwei verschiedene Sudokus durch zwei Klassen realisiert werden
2. es zur Laufzeit möglich ist, das Spiel zu wechseln.

Was ist zu tun?



## 4.4 Vererbung

Abstrakte Klassen

- Es gibt keine Instanzen/Objekte abstrakter Klassen
- Sie dienen allein der Vererbung
- Deklaration

```
abstract class <Bezeichner>{
   <Implementierung>
}
```

- Man vergibt Namen für die dynamische Bindung



## 4.4 Vererbung

Abstrakte Methoden und Klassen

- Abstrakte Methoden deklarieren Methoden für die Vererbung
- Bsp `abstract class X{ abstract int f(int a);}`
- Jede Subklasse solcher abstrakter Klassen muss die abstrakte Methode implementieren oder selbst abstrakt sein:
- `class Y extends X{ int f(int a){…}}`
- Wenn eine Klasse eine abstrakte Methode hat muss sie auch abstrakt sein



## 4.4 Vererbung

Im Plenum: Fallstudie

Implementieren Sie die Klasse SudokuLogic als abstrakte Klasse, so dass alle Spezialisierungen eine Methode zur Definition der Spielinstanz (vorgegebene Zahlen) zur Verfügung stellen müssen.

## 4.4 Vererbung

Mehrfachvererbung

- Unter Mehrfachvererbung versteht man wenn eine Klasse mehr als eine Superklasse erweitert

*Nennen Sie Beispiele, bei denen Objekte zwei Generalisierungen haben*

- Mehrfachvererbung ist in Java (aus gutem Grund) verboten



## 4.4 Vererbung

Interfaces

- Ein interface (Schnittstelle) besteht ausschließlich aus abstrakten Methoden Deklaration:

```
interface <Bezeichner>{
   <Implementierung>
}
```

- Die Methoden werden *nicht* `abstract` deklariert
- (Variablendeklarationen sind auch erlaubt, sind dort aber Konstanten → lassen Sie es lieber)
- Bsp: siehe Java API: List, Comparable, Serializable…



## 4.4 Vererbung

Interfaces instantiieren

- Die Implementierung eines Interface ist eine Klasse, die all ihre Methoden implementiert
- Deklaration

```
class <Klasse> implements <Interface>
{…}
```

- Klassen dürfen mehrere Interfaces implementieren (Mehrfach-Spezialisierung)



## 4.4 Vererbung

TODO: <ab hier in 2.Sem

Fallstudie

Implementieren Sie ein Interface für die Ausgabe von Informationen im Sudoku.

1. Instantiieren Sie diese mit dem GUI
2. implementieren eine Ausgabe an der Konsole



## 4.4 Vererbung

Murmelgruppe: Klausuraufgabe (5 min)

Schreiben Sie eine Prozedur **int[] map(int[] a, Function f)** in Java, die die Funktion **f** auf alle Elemente von **a** anwendet und das so entstandene Array zurückgibt.

```
interface Function{
  // Eine Funktion f. Wenn execute(x) den Wert y
  // liefert, bedeutet das f(x)=y
        int execute(int x);
}
```

Schreiben Sie einen Testfall für map, in dem zu jedem Element des Arrays 5 addiert wird.



## 4.4 Vererbung

Die Mutter aller Klassen:

`java.lang.Object`

- Ist Superklasse jeder Java-Klasse
- kein explizites `extends Object` nötig
- Legt einige Methoden fest, die jedes Objekt haben sollte
- Stellt Standard-Implementierungen zur Verfügung
- Diese können/sollten überschrieben werden

## 4.4 Vererbung

Methoden von `java.lang.Object`

`public boolean equals()`

- Implementiert Gleichheit von Objekten
- Operator == funktioniert nur für Grundtypen bzw. die Referenzen der Objekte



## 4.4 Vererbung

Methoden von `java.lang.Object`

`protected void finalize()`

- Wird vom Garbage Collector aufgerufen um nicht mehr benötigten Speicher freizugeben
- Der GC erspart dem Programmierer das Löschen von Hand und verhindert memory-leaks



## 4.4 Vererbung

Methoden von `java.lang.Object`

`public String toString();`

- Darstellung des Objekts als Zeichenkette
- Wird z.B. aufgerufen bei print nach stdout
- Sollte für jedes Objekt individuell implementiert werden



## 4.4 Vererbung

Einzelübung

- Implementieren Sie eine Java-Klasse für Personen mit den Methoden
  - toString
  - equals



## 4.4 Vererbung

AutoBoxing und unboxing

- Die einfachen Datentypen int, char, float… haben eine Entsprechung als Klasse: Integer, Character, Float…
- Autoboxing: die automatische Umwandlung von Werten einfacher Typen in Objekte ihrer Klasse

zB: int x = 5; Integer y = new Integer(3); y = x;

- Unboxing: umgekehrt

zB: Integer y = new Integer(3); int x = y % 2;



## 4.4 Vererbung

Übung

- Implementieren Sie ein Interface für mathematische Verknüpfungen (wie +, -, * oder /) in Java
- Legen Sie sich dabei nicht auf den Typ der Funktionsparameter fest sondern verwenden die Klasse „Number“
- Implementieren Sie eine Klasse für die Addition, die diese Schnittstelle implementiert
- Probieren Sie aus: Auf Zahlen welchen Typs kann man diese Addition anwenden?

## TODO: in Üb: feedback

Bitte öffnen Sie die Zielscheibe

http://zielscheibe.hszg.de/aims/???

Ich interessiere mich für Ihre (subjektive) Einschätzung Ihres Lernerfolgs.

Bitte denken Sie mal kurz darüber nach.



## 4. Grundlagen OOP

1. Objekt, Instanz und
2. Methoden
3. Statisch vs. dynamisch
4. Vererbung
5. packages
6. Sichtbarkeit und Modifikatoren
7. Aufzählungs-Typen
8. Code-Dokumentation