Informatik II - Übung 6

Pascal Schärli

WelcomeToMyCrib@pascscha.ch

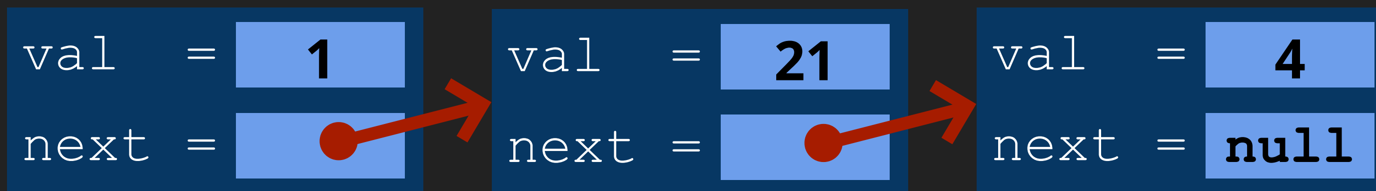
30.10.2020

ETH zürich

Nachbesprechung

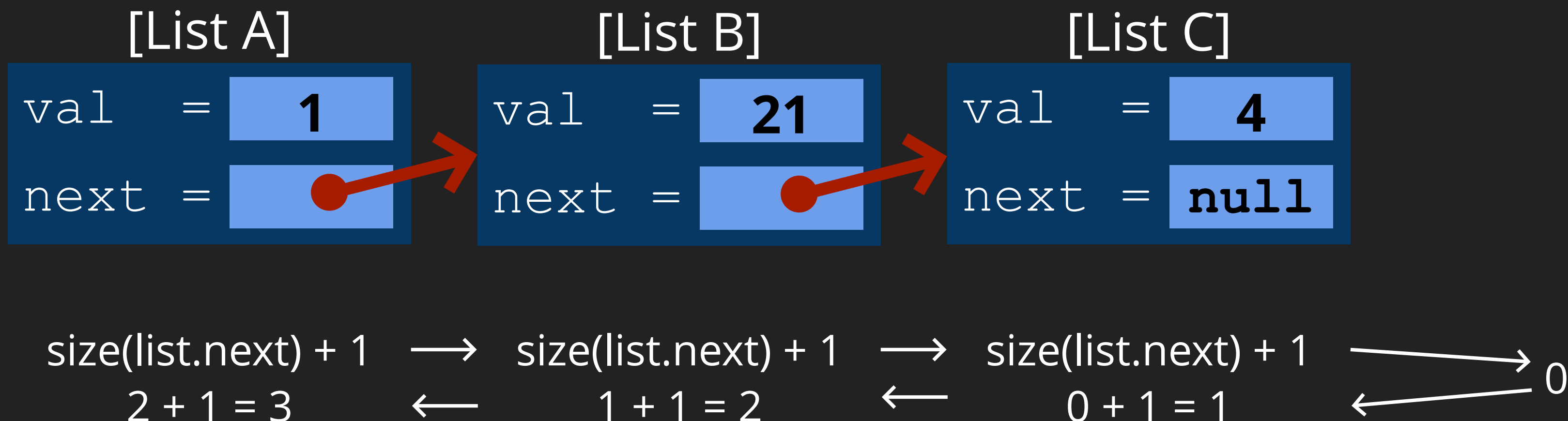
Einfach verkettete Liste

```java
public static List add(List list, int value) {
    return new List(value, list);
}
```



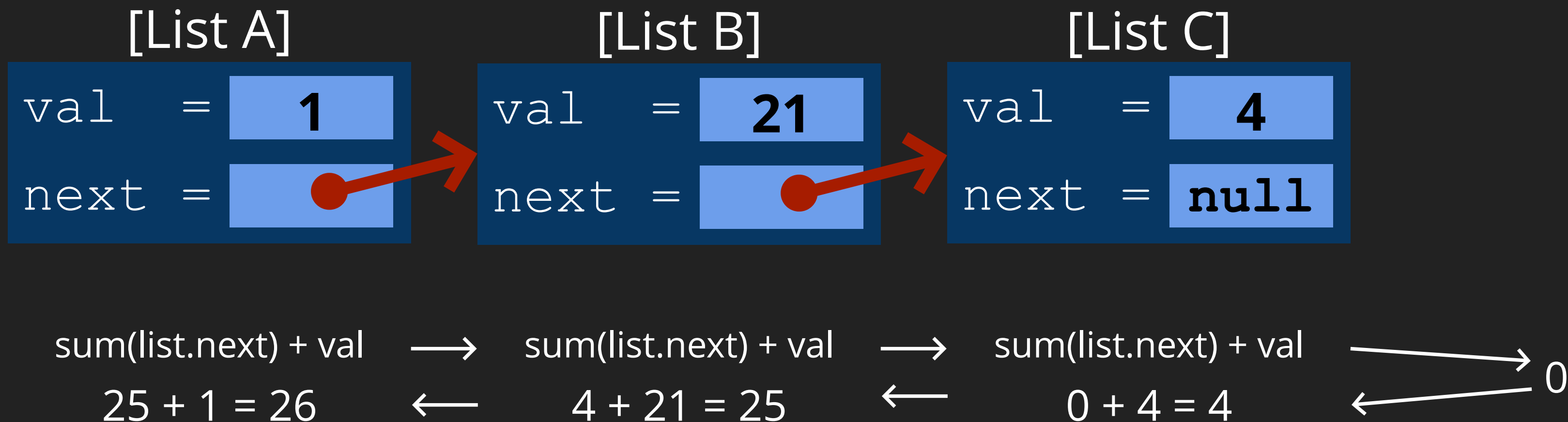
Einfach verkettete Liste

```java
1 public static int size(List list) {
2     if (list == null) return 0;
3     return size(list.next) + 1;
4 }
```



Einfach verkettete Liste

```java
1 public static int sum(List list) {
2     if (list == null) return 0;
3     return list.value + sum(list.next);
4 }
```



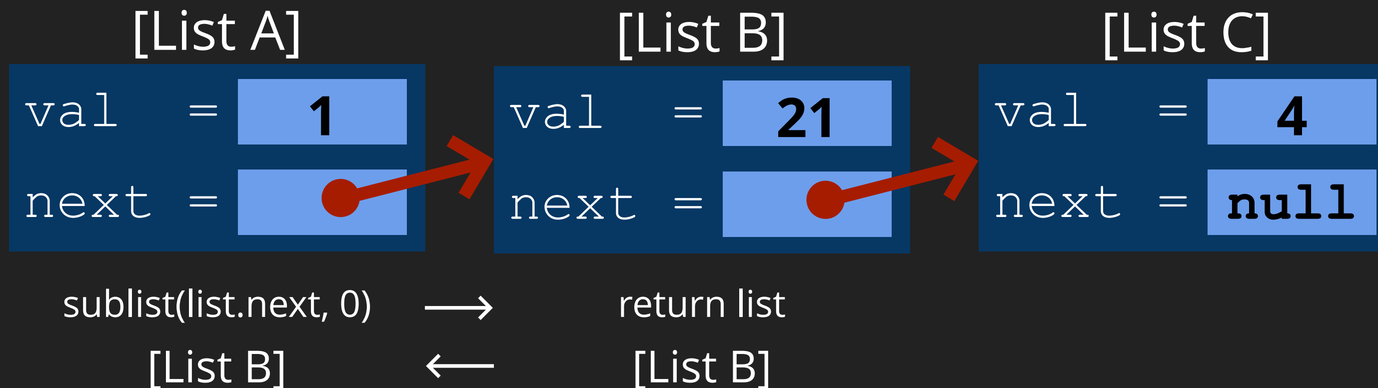
Einfach verkettete Liste

```java
1 public static List last(List list) {
2     if (list == null) return null;
3     if (list.next == null) return list;
4     return last(list.next);
5 }
```

Einfach verkettete Liste

```java
public static List sublist(List list, int index) throws IndexOutOfBoundsException {
    if (list == null || index < 0) throw new IndexOutOfBoundsException();
    if (index == 0) return list;
    return sublist(list.next, index - 1);
}
```

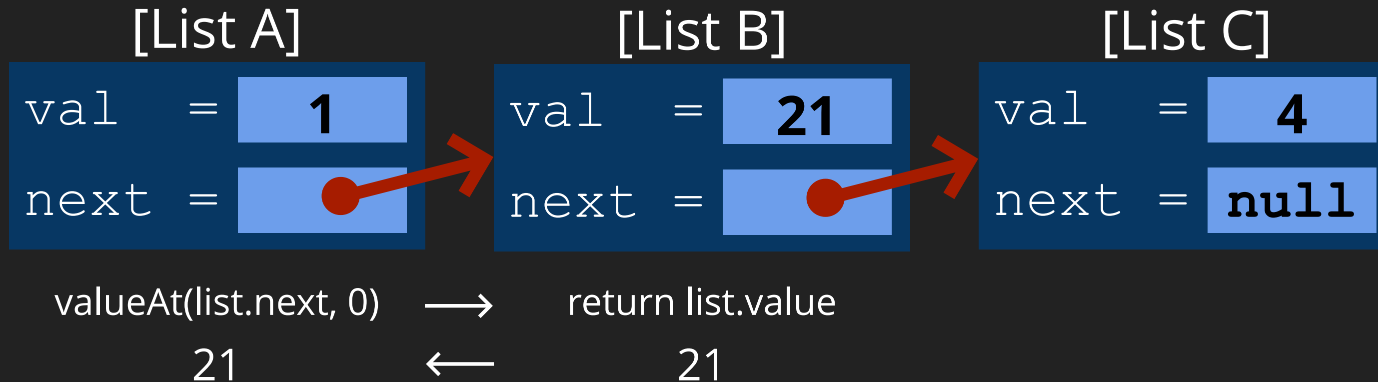
sublist(list,1)



Einfach verkettete Liste

```java
public static int valueAt(List list, int index) throws IndexOutOfBoundsException {
    if (list == null || index < 0) throw new IndexOutOfBoundsException();
    if (index == 0) return list.value;
    return valueAt(list.next, index - 1);
}
```

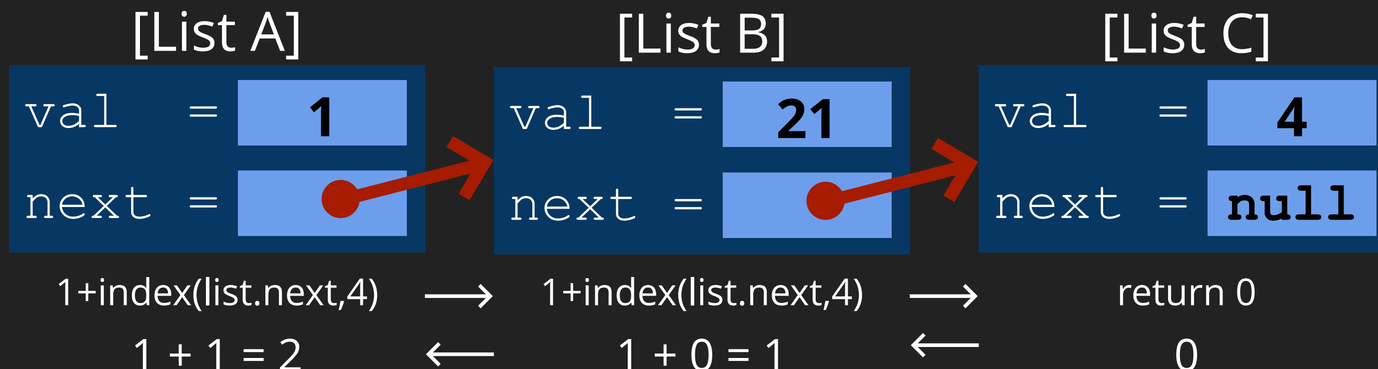
valueAt(list,1)



Einfach verkettete Liste

```
1 public static int index(List list, int value) throws NoSuchElementException {
2     if (list == null) throw new NoSuchElementException();
3     if (list.value == value) return 0;
4     return 1 + index(list.next, value);
5 }
```

index(list,4)



Modifizierung von Listen

```
1 public static void append(List list, int value) throws IllegalArgumentException {
2     if (list == null) throw new IllegalArgumentException();
3     Lists.last(list).next = new List(value, null);
4 }
```

Modifizierung von Listen

```java
public static void concat(List head, List tail) throws IllegalArgumentException {
    if (head == null) throw new IllegalArgumentException();
    Lists.last(head).next = tail;
}
```

Modifizierung von Listen

```java
1 public static void insertAt(List list, int index, int value) throws IndexOutOfBoundsException {
2     if (list == null || index < 0) throw new IndexOutOfBoundsException();
3     if (index == 0) {
4        list.next = new List(value, list.next);
5     } else {
6        insertAt(list.next, index - 1, value);
7    }
8 }
```

Modifizierung von Listen

```
public static void insertAt(List list, int index, List newList) throws IndexOutOfBoundsException {
    if (newList == null) return;
    if (list == null || index < 0) throw new IndexOutOfBoundsException();
    if (index == 0) {
       Lists.last(newList).next = list.next;
       list.next = newList;
    } else {
        insertAt(list.next, index - 1, newList);
    }
}
```

Modifizierung von Listen

```
public static List remove(List list, int index) throws IndexOutOfBoundsException {
    if (list == null || index < 0) throw new IndexOutOfBoundsException();
    if (index == 0) return list.next;
    list.next = remove(list.next, index - 1);
    return list;
}
```

Sortieren von Listen

```
1  public static List insertSorted(List list, int value) {
2      if (list == null){
3          return new List(value, null);
4      }
5      else if (value < list.value){
6          return new List(value, list);
7      }
8      else {
9          list.next = insertSorted(list.next, value);
10         return list;
11     }
12 }
```

```
public static List sort(List list) {
    if (list == null){
        return null;
    }
    List nextSorted = sort(list.next);
    return insertSorted(nextSorted, list.value);
}
```

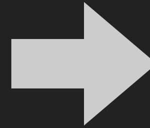
Best of Vorlesung

Klassenhierarchie

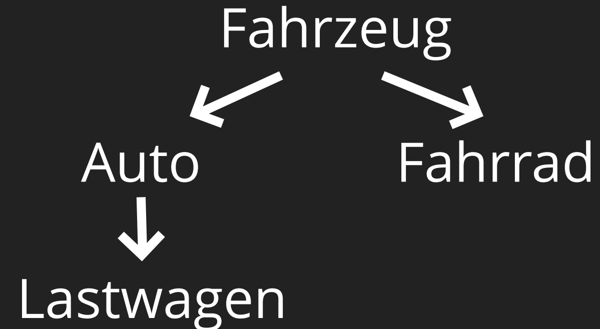
- In Java ist es möglich, dass eine Klasse von einer anderen Klasse "erbt".
- Die erbende Klasse erbt somit alle Funktionen und Attribute und kann diese noch erweitern.
- Dies ermöglicht es gewisse Typen und Strukturen besser zu modellieren.

Klassenhierarchie

```java
public class Fahrzeug{
    int radzahl;

    public Fahrzeug(int radzahl){
        this.radzahl = radzahl;
    }
}
```



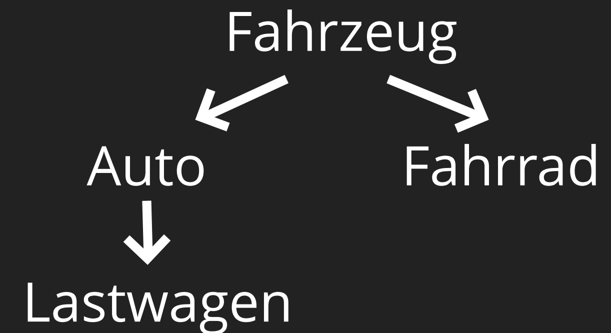
```java
public class Fahrrad extends Fahrzeug{

    public Fahrrad(){
        super(2);
    }
}
```




```java
public class Auto extends Fahrzeug{
    protected float hubraum;

    public Auto(float hubraum){
        super(4);
        this.hubraum = hubraum;
    }

    public float getHubraum(){
        return hubraum;
    }

    public void setHubraum(float hubraum){
        if(hubraum>0) this.hubraum = hubraum;
    }

}
```

```java
public class Lastwagen extends Auto{
    float capacity;

    public Lastwagen(float hubraum,
                     float capacity){
        super(hubraum);
        this.capacity = capacity;
    }
}
```

Klassenhierarchie




```
1 Fahrzeug myFahrrad = new Fahrrad();
2
3 System.out.println(myFahrrad.radzahl);
4
5 myFahrrad.setHubraum(300);
```

Output: 2

Compile Error:
The method setHubraum(int) is undefined for the type Fahrzeug

```
1 Fahrzeug myLastwagen = new Lastwagen(80000,50);
2
3 System.out.println(myLastwagen.radzahl);
4
5 myLastwagen.setHubraum(300);
```

Output: 4

Compile Error:
The method setHubraum(int) is undefined for the type Fahrzeug

Klassenhierarchie

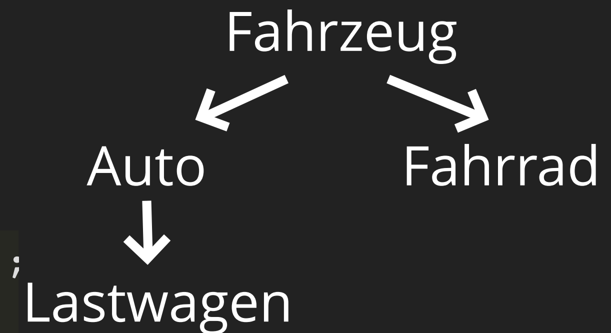
- Warum können wir den Hubraum von myLastwagen nicht bestimmen, obwohl es ein Objekt vom Typ Lastwagen ist?
- Beim Erstellen von myLastwagen haben wir gesagt, dass es vom Typ Fahrzeug ist:

```
Fahrzeug myLastwagen = new Lastwagen(80000,50);
```

- myLastwagen hat also alle Attribute von einem Lastwagen, man kann sie einfach nicht abrufen.
- ⇒ Type Casts

```
((Lastwagen)myLastwagen).setHubraum(300);
```

Type Casts



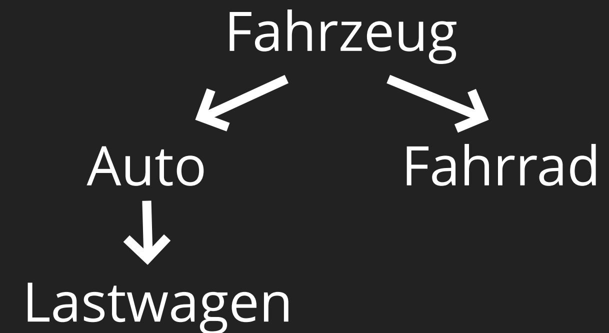

```
1 Fahrzeug myLastwagen = new Lastwagen(80000,50);
2
3 System.out.println(myLastwagen.radzahl);
4
5 ((Lastwagen)myLastwagen).setHubraum(300);
```

```
1 Fahrzeug myFahrrad = new Fahrrad();
2
3 System.out.println(myFahrrad.radzahl);
4
5 ((Lastwagen)myFahrrad).setHubraum(300);
```

Runtime Error:

Exception in thread "main" java.lang.ClassCastException: Fahrzeug cannot be cast to Lastwagen

Zusammenfassung



1. Falls ihr auf ein Attribut zugreifen möchtet, welches nicht existiert oder nicht abrufbar ist, gibt es einen **Compile Error**

```
Fahrzeug myLastwagen = new Lastwagen(80000, 50);
myLastwagen.setHubraum(300);
```

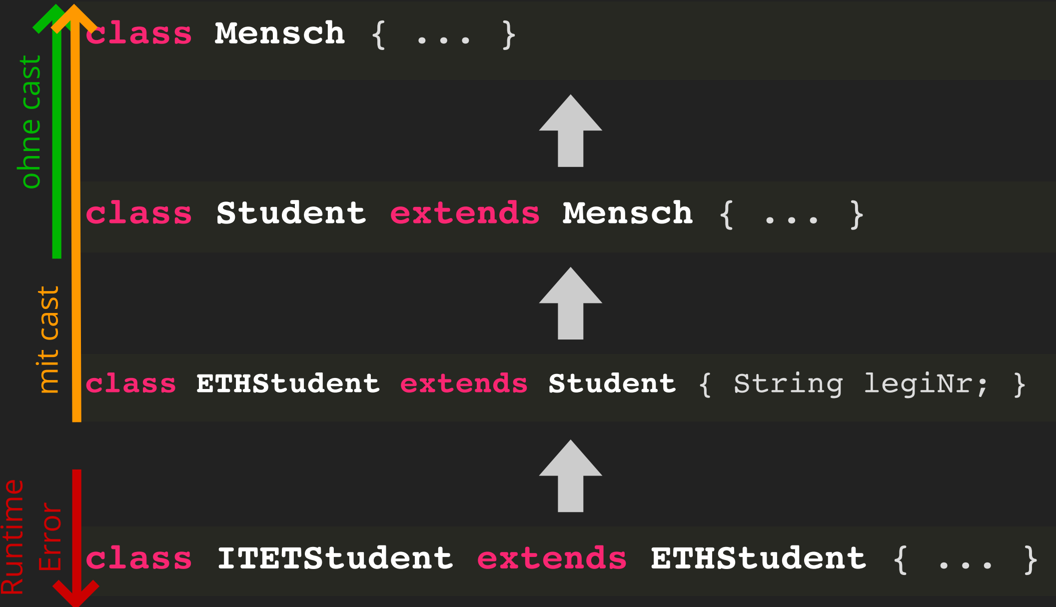
2. Falls ihr einen Cast machen wollt, welcher nicht funktioniert, gibt es einen **Runtimer Error**

```
Fahrzeug myFahrrad = new Fahrrad();
((Lastwagen)myFahrrad).setHubraum(300);
```

Beispiele

Java denkt es ist ein Student

Wir wissen, es ist ein ETHStudent

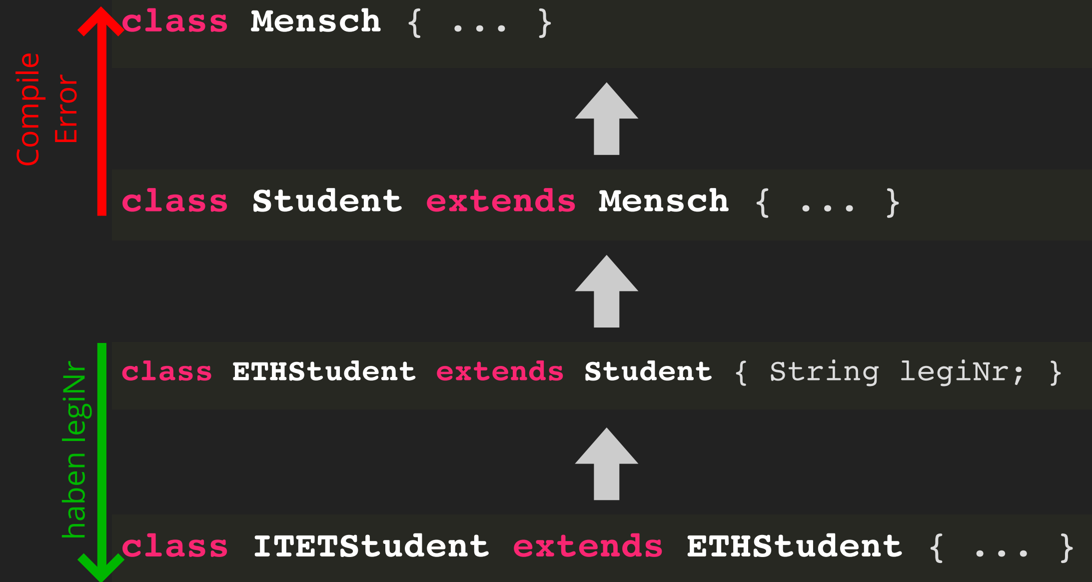



```
Student tim = new ETHStudent();
```

```
1 Objekt obj               = tim;
2 Mensch mensch            = tim;
3 Student student          = tim;
4 ETHStudent ethStud       = (ETHStudent) tim;
5 ITETStudent itetStud = (ITETStudent) tim;
```

jede Java Klasse erbt von Object
zuweisung möglich ohne Cast
zuweisung möglich ohne Cast
zuweisung möglich mit Cast
Runtime Error

Beispiele




```
ITETStudent tim = new ITETStudent();
```

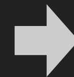
```
1 Mensch mensch         = tim;
2 Student student       = tim;
3 ETHStudent ethStud    = tim;
4 ITETStudent itetStud = tim;
5
6 System.out.println(mensch.legiNr);
7 System.out.println(student.legiNr);
8 System.out.println(ethStud.legiNr);
9 System.out.println(itetStud.legiNr);
```

Compile Error
Compile Error
Kein Compile Error
Kein Compile Error

instanceof

- Falsche Type-Casts können zu Runtime-Error führen.
- Wie können wir das verhindern?
- instanceof Operator gibt zurück, ob ein Objekt gecasted werden kann.

```java
Student tim = new ETHStudent();

System.out.println(tim instanceof Mensch);
System.out.println(tim instanceof Student);
System.out.println(tim instanceof ETHStudent);
System.out.println(tim instanceof ITETStudent);
```



Output:

```
true
true
true
false
```

Abstrakte Klassen

- Manchmal macht es Sinn für bestimmte Klassen bloss zu definieren *was* sie können muss und noch nicht *wie* sie implementiert wird.

- Beispiel: Stacks
  - Jeder Stack hat bestimmte Funktionen (pop / peek / push)
  - Es gibt aber verschiedene möglichkeiten einen Stack zu implementieren (LinkedList, Array, ...)
- Zwei Arten:
  - *abstract*
  - *interface*

Abstrakte Klassen - *abstract*

- In einer Abstrakten Klasse kann man Methoden implementieren, aber auch nur deklarieren.
- Man kann keine Abstrakten Objekte instanzieren (weil man sonst unimplementierte Methoden aufrufen könnte)

```java
public abstract class AbstractStack {
    // declare abstract functions
    public abstract int push();
    public abstract int pop();
    public abstract int peek();
    public abstract int size();

    // already implement some functions
    public boolean isEmpty(){
        return size() == 0;
    }
}
```

```java
public class Stack extends AbstractStack {
    // declare abstract functions
    public int push(){/*TODO*/}
    public int pop(){/*TODO*/}
    public int peek(){/*TODO*/}
    public int size(){/*TODO*/}
}
```

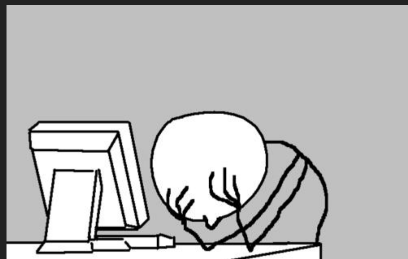
Abstrakte Klassen - *interface*

- Interfaces funktionieren gleich wie abstrakte Klassen
- Ein Interface ist eine Schnittstelle zwischen Benutzung un
- Man darf keine Funktionen mehr implementieren, nur noch deklarieren
- Vorteil: Eine Klasse kann mehrere Interfaces implementieren, aber nur von einer Klassen erben (*extends*)

```
public class C implements I1,I1 extends A{
        // [...]
}
```

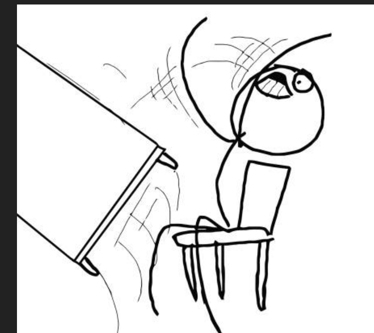
Factories

- Julia und Carina arbeiten mit Bäumen.
- Um gleichzeitig daran zu arbeiten implementiert Carina den Baum und Julia schreibt das Programm.
- Sie entscheiden sich: Sie verwenden einen *ArrayTree*
- Nach der Hälfte der Zeit merkt Carina, dass ein ListTree besser geeignet wäre.
- Carina sagt Julia, sie solle doch schnell alle Verwendungen von ArrayTree zu ListTree ändern.


Carina

Benutze *ArrayTree*

Meinung geändert, benutze *ListTree*


Julia

Factories

- Um solche *Katastrophen* zu verhindern benutzen wir Interfaces und Factories
- Julia benutzt eine "TreeFactory" um Bäume zu erstellen, welche dieses Interface implementieren.

Tree Interface

```
interface Tree{
    public Tree leftChild();
    public Tree rightChild();
    public int getValue();
    public int setValue();
}
```

Julias Code Vorher

```
1 public class FancyCode {
2         public Tree important(){
3         Tree myRoot = new ArrayTree();
4         myRoot.setValue(4);
5         return myRoot;
6     }
7 }
```

Tree Factory

```
public class TreeFactory {
    public static Tree makeTree(){
        return new ListTree();
    }
}
```

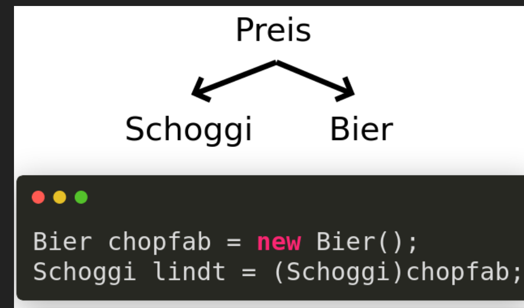
Julias Code Jetzt

```
1 public class FancyCode {
2         public Tree important(){
3         Tree myRoot = TreeFactory.makeTree();
4         myRoot.setValue(4);
5         return myRoot;
6     }
7 }
```

K!

Fehler bei meiner Kahoot Lösung

Ist die Zuweisung **Schoggi lindt = (Schoggi)chopfab;** möglich?

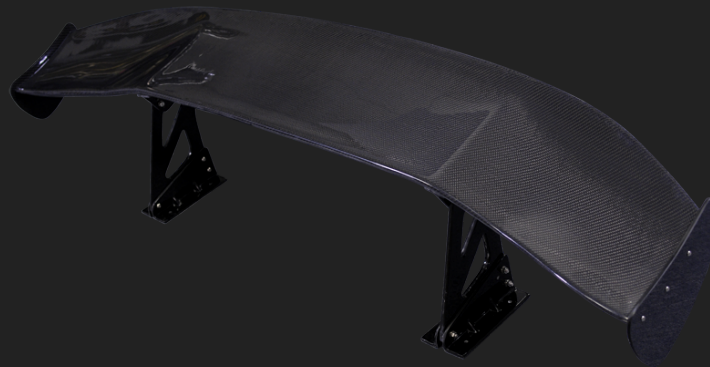


Die Antwort ist wie auch schon während der Übungsstunde gesagt ***nein***, es gibt aber einen ***Compile***-Error und nicht einen *Runtime*-Error. Java weiss, dass **chopfab** ein **Bier** ist und somit auf einem anderen "Ast" als **Schoggi**, somit kann es bereits beim kompilieren wissen, dass es niemals möglich wäre das zu einer **Schoggi** zu casten und es gibt daher schon beim Kompilieren einen Fehler.

Fehler bei meiner Kahoot Lösung

```java
Bier b1 = new Bier();
Preis b2 = new Bier();
Preis p = new Preis();
Preis s1 = new Schoggi();
Schoggi s2 = new Schoggi();

Schoggi mySchoggi1 = (Schoggi)b1; // Compile error
Schoggi mySchoggi2 = (Schoggi)b2; // Runtime error, da Java nicht weiss, dass es ein Bier ist
Schoggi mySchoggi3 = (Schoggi)p;  // Runtime error
Schoggi mySchoggi4 = (Schoggi)s1; // Funktioniert
Schoggi mySchoggi5 = (Schoggi)s;  // Funktioniert, cast wäre nicht nötig
```

Vorbesprechung



Klassen, Schnittstellen und Typumwandlungen

Ihr habt die folgende Klassenhierarchie gegeben:

```
interface A { }
abstract class B implements A { }
interface C extends A { }
class D extends B implements C { }
class E extends B { }
class F implements C { }
```

1. Zeichnet die Abhängigkeiten in einem Graph auf
2. Welche Typen kann man mit new instanzieren?
3. Ganz viele Cast-/Instanzierungsbeispiele, analog zu hier:

Schnittstellen und Implementierungen

1. Vervollständigt die Fabrikmethode, gebt einen ListStack zurück:

```
public class StackFactory {
    public static IStack create() {
        // TODO
        return null;
    }
}
```

2. Ergänzt das Interface IStack, so dass es noch eine empty() Funktion hat **(Inklusive Java-Doc)**

3. Implementiert diese Funktion in der ListStack Klasse

4. Schreibt einen Public Test um die Funktion zu testen

Polymorphie

- Erstellt das FIle ListUtils.java und implementiert dort die Klasse ListUtils, welche die Funktionen vom Interface IListUtils implementiert. (Musterlösung U5A3)
- Erstellt die Fabrikmethode in *ListUtilsFactory.java*
- Implementiert die Funktion area in den Files *Rectangle.java* sowie *Triangle.java*
- Implementiert die Funktion *smallerThan* im File *GeometricObject.java*, welches prüft ob die eigene Fläche kleiner ist als die vom Übergabeargument
- Implementiert die Funktion sort im File *ListUtils.java* (Musterlösung U5A3)

Stacks und Optimierungen

- Für "fortgeschrittene" - Ihr seid alle genug fortgeschritten ;)
- Kreiert die Klasse ChunkedStack im File "ChunkedStack.java".
- Stack, welcher als Mischung vom ListStack und ArrayStack ist:
  Array $\rightarrow$ Array $\rightarrow$ Array $\rightarrow$ null
- Was ist schneller – ChunkedStack oder ListStack?
  Wie wächst der Aufwand für size in Abhängigkeit der Listengrösse?
- Kann man size auch besser implementieren?

Viel Spass!