

# FOP Recap #6



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

## Abstrakte Klassen





Hinweis

Abstrakte Klassen

Statische Methoden und Attribute

Scopes, Bedingungsoperator

String, Casting

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Hinweis

Raumänderung  
Vererbung

Abstrakte Klassen

Statische Methoden und Attribute

Scopes, Bedingungsoperator

String, Casting



- **Nächste Woche in S402/101!**

- Sehr wichtiger Klausurinhalt
- Fokus auf Interfaces und abstrakte Klassen

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Hinweis

Abstrakte Klassen

Syntax

Wichtige Eigenschaften

Zusammenhang

Statische Methoden und Attribute

Scopes, Bedingungsoperator

String, Casting



```
1 public abstract class MyClassName {  
2     ....  
3 }
```

### Syntax abstrakte Klasse:

*Zugriffsmodifikatoren* abstract class *Klassen-Name*

extends *Basis-Klassen-Name*

implements *Interface-Namen*



- Abstrakte Klassen können abstrakte Methoden deklarieren
- Es kann kein Objekt von einer abstrakten Klasse erstellt werden
- Abstrakte Klassen müssen nicht alle abstrakten Methoden ihrer Basis-Klassen implementieren
  - ▣ Weder die abstrakten Methoden ihrer abstrakten Basisklassen
  - ▣ Noch die Methoden der Interfaces, von denen sie direkt oder indirekt erben





---

# Intermezzo: Live-Coding

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Hinweis

Abstrakte Klassen

Statische Methoden und Attribute

- Allgemein

- Klassenmethoden

- Klassenattribute

Scopes, Bedingungsoperator

String, Casting



- Methoden und Attribute gehören immer zu einer Klasse
- `static` Methoden und Attribute sind jedoch unabhängig von Objekten dieser Klasse
- Man kann auch ohne Objekte auf `static` Methoden und Attribute zugreifen
- Objektmethoden lassen sich jedoch weiterhin nur mit einem Objekt aufrufen

```
1 public class MyClassName {  
2     public static void test() {  
3         ....  
4     }  
5     public void omnom() {  
6         test();  
7     }  
8 }
```

```
1 MyClassName.test();
```



- Haben programm-weit denselben Wert
- Unabhängig von jeglichen Objekten



```
1 public class MyClassName {  
2     public static int myValueName = 5;  
3     public void omnom() {  
4         myValueName = -1;  
5     }  
6 }
```

```
1 MyClassName.myValueName = -24;
```

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Hinweis

Abstrakte Klassen

Statische Methoden und Attribute

Scopes, Bedingungsoperator

Klassen/Interfaces/Enums

Attribute/Methoden

Parameter/Lokale Variablen

Shadowing

Überblick

Unterschied



- Definieren in welchem Bereich Identifier sichtbar sind
- Möglicher Zugriff lässt sich dann (teils) durch Access Modifiers weiter einschränken
- Identifier sind zum Beispiel:
  - Klassen/Interfaces/Enums
  - Attribute
  - Variablen
  - Methoden
  - ...



# Scopes

## Klassen/Interfaces/Enums



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Sind bei uns normalerweise alle `public`
- Können dann überall importiert und genutzt werden



- Sind bei uns normalerweise alle `public`
- Können dann überall importiert und genutzt werden
  
- Ergänzung:
  - ▣ In jeder Datei gibt es genau eine Top-Level Klasse, die den Dateinamen trägt
  - ▣ Kann jedoch beliebig viele nicht `public` Klassen geben

- Können je nach Access Modifier genutzt werden

# Scopes

## Parameter/Lokale Variablen



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Können nur in ihrer Methode genutzt werden
- Sind nur innerhalb ihres „geschweiften Klammerpaares“ zulässig

```
1 public String global = "global";
2 public void foo() {
3     String local = "local";
4     System.out.println(global); // -> "global"
5     System.out.println(local); // -> "local"
6 }
7 public void bar() {
8     System.out.println(global); // -> "global"
9     System.out.println(local); // -> Compiler-Error
10 }
```



- Falls zwei Variable im Scope mit demselben Namen vorliegen
- Lässt sich dann mit `this` und `super` lösen

```
1 public class Auto {  
2     public double maxSpeed;  
3  
4     public Auto(double maxSpeed) {  
5         this.maxSpeed = maxSpeed;  
6     }  
7 }
```



- Kurzform für if-else
- Nicht ganz indentisch, wegen Auswertungsreihenfolge

```
1  int number = -25;
```

```
1  int absValue;  
2  if(number < 0) {  
3      absValue = -number;  
4  }  
5  else {  
6      absValue = number;  
7  }
```

```
1  int absValue = number < 0 ? -number : number;
```

# Bedingungsoperator

## Unterschied



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 int[] array = new int[] {1, 2, 3};  
2 int index = 1;
```

```
1 array[index++] = index == 1 ? 2 : -5;  
2 // -> array = {1, 2, -5}
```

```
1 if(index == 1) {  
2     array[index++] = 2;  
3 }  
4 else {  
5     array[index++] = -5;  
6 }  
7 // -> array = {1, 2, 2}
```

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Hinweis

Abstrakte Klassen

Statische Methoden und Attribute

Scopes, Bedingungsoperator

String, Casting

Interner Aufbau

char

Beispiele

char-Arithmetik

Schon besser



- Im Prinzip nur ein `char [ ]`
- Jeder `String` is unveränderbar



- Ist wie `int` ein primitiver Datentyp
- Belegt 2 Bytes Speicher statt wie ein `int` 4 Bytes
- Normale Verwendung:
  - ▣ Repräsentiert (im Normalfall) genau einen Buchstaben/Zeichen
  - ▣ Jedes Zeichen hat nach Unicode einen festen Zahlenwert zugeordnet



```
1 String s = "ABC";  
2 char c0 = s.charAt(0); // == 'A'  
3 char c1 = s.charAt(1); // == 'B'  
4 char c2 = s.charAt(2); // == 'C'
```

# String

## Beispiele



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 String s = "ABC";  
2 char[] arr = s.toCharArray();  
3 char c0 = arr[0]; // == 'A'  
4 char c1 = arr[1]; // == 'B'  
5 char c2 = arr[2]; // == 'C'
```

# String

## Beispiele



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 String s = "ABC";  
2 String result = "";  
3 result += s.charAt(2);  
4 result += s.charAt(1);  
5 result += s.charAt(0);
```



```
1 String s = "ABC";  
2 String result = "";  
3 for(int i = s.length() - 1; i >= 0; i--) {  
4     result += s.charAt(i);  
5 }  
6 System.out.println(result);
```



```
1 String s = "ABC";  
2 String result = "";  
3 for(int i = s.length() - 1; i >= 0; i--) {  
4     result += s.charAt(i);  
5 }  
6 System.out.println(result);
```

\$ CBA



- In meinen Augen: Absolut schrecklich

```
1 char c = 'E';  
2 char k = (char)(c + 1); // == 'F'
```



## ■ Für Groß und Kleinschreibung!

```
1 char c = 'a';  
2 char c2 = Character.toUpperCase(c); // == 'A'  
3 boolean check = Character.isUpperCase(c2); // == true  
4 char c3 = Character.toLowerCase(c2);
```



```
1 String message = "hihi";
2 String upperOne = message.toUpperCase();
3 System.out.println(upperOne);
4
5 final Locale turkish = new Locale("tr");
6 String upperTwo = message.toUpperCase(turkish);
7 System.out.println(upperTwo);
```



```
1 String message = "hihi";  
2 String upperOne = message.toUpperCase();  
3 System.out.println(upperOne);  
4  
5 final Locale turkish = new Locale("tr");  
6 String upperTwo = message.toUpperCase(turkish);  
7 System.out.println(upperTwo);
```

\$ HIHI

\$ HİHİ



- Bei primitiven Datentypen
  - ▣ um einen Zahlenwert in einem anderen Typen zu speichern
  - ▣ passiert implizit oder explizit
- Bei Objekttypen
  - ▣ um den statischen Typen zu ändern
  - ▣ kann jedoch je nach dynamischen Typen fehlschlagen

# Casting

## Bei primitiven Datentypen



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Jeder Zahltyp hat einen Bereich an Zahlen, den er repräsentieren kann
- `long`, `int`, `char`, `short`, `byte` für Ganzzahlen
- `double`, `float` für Komma-Zahlen
- Hierbei erkennt man folgende Rangordnung der Bereiche:
- `double > float > long > int > char > short > byte`

# Casting

## Bei primitiven Datentypen – Widening Casting



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Passiert implizit, automatisch

```
1 int a = 5;  
2 int b = 27;  
3 long c = a - b;
```

# Casting

## Bei primitiven Datentypen – Widening Casting



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Passiert implizit, automatisch

```
1 char a = 5;  
2 int b = 27;  
3 long c = a - b;
```



- Muss explizit angegeben werden, da Verlust von Präzision stattfinden kann!

```
1  int a = 288;  
2  char c = a; // ERROR!  
3  char c2 = (char) a; // OK!  
4  
5  char c3 = a + 5; // ERROR!  
6  char c4 = ((char) a) + 5; // ERROR!  
7  char c5 = (char) (a + 5); // OK!
```





- Funktioniert nur ohne Fehler bei passenden dynamischen Typen!

```
1 class A { .... }
2 class B extends A { .... }
3 class C extends B { .... }
4
5 A a = ....;
6 C castedC = (C) a; // Nur möglich, wenn a dynamischen Typen von C
   ↳ oder Subtypen hat
```



---

# Live-Coding!