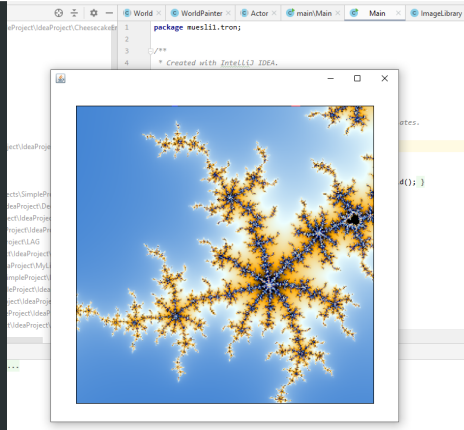


# FOP Recap #2

## Objekte, Attribute, Methoden





---

# Hey, los gehts!

# Heute aufn Menü



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Organisation, Hinweise

break, continue, if, else

if und else

Import-Anweisungen

Objekte und Typen

Attribute vs lokale Variable

Enumeration

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Organisation, Hinweise

Was ist das Recap?

Wann und wo findet das Recap statt?

`break, continue, if, else`

`if` und `else`

Import-Anweisungen

Objekte und Typen

Attribute vs lokale Variable

Enumeration

# Organisation

## Was ist das Recap?



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Das Recap ist ein zusätzliches und freiwilliges Angebot
- Keine Anwesenheitspflicht
- Keine Beantwortung von Fragen zu Hausübungen
- Stattdessen:
  - Wiederholung des Stoffes
  - Erklärung anhand von Beispielen
  - Rückfragen und Diskussion **erwünscht**, auch über Discord!

# Organisation

## Wann und wo findet das Recap statt?



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Jeden Mittwoch um 15:30 Uhr bis ca. 17:00 Uhr
- Verfügbar:
  - ▣ In Präsenz (regulär in S1|03 226)
  - ▣ per Live-Stream auf YouTube
  - ▣ Aufzeichnung nachträglich als YouTube-Video verfügbar

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Organisation, Hinweise

**break, continue, if, else**  
break und continue

if und else

Import-Anweisungen

Objekte und Typen

Attribute vs lokale Variable

Enumeration

# Ergänzung Schleifen

## break und continue



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1  for(int i = 0; i < 3; i++) {  
2      System.out.println("i = " + i + "!");  
3  }
```



# Ergänzung Schleifen

## break und continue



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1  for(int i = 0; i < 3; i++) {  
2      System.out.println("i = " + i + "!");  
3  }
```

```
$ i = 0!  
$ i = 1!  
$ i = 2!
```

# Ergänzung Schleifen

## break und continue



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1  for(int i = 0; i < 3; i++) {  
2      if(i == 1) {  
3          continue;  
4      }  
5      System.out.println("i = " + i + "!");  
6  }
```

# Ergänzung Schleifen

## break und continue



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1  for(int i = 0; i < 3; i++) {  
2      if(i == 1) {  
3          continue;  
4      }  
5      System.out.println("i = " + i + "!");  
6  }
```

```
$ i = 0!
```

```
$ i = 2!
```

# Ergänzung Schleifen

## break und continue



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1  for(int i = 0; i < 3; i++) {  
2      if(i == 1) {  
3          break;  
4      }  
5      System.out.println("i = " + i + "!");  
6  }
```

# Ergänzung Schleifen

## break und continue



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1  for(int i = 0; i < 3; i++) {  
2      if(i == 1) {  
3          break;  
4      }  
5      System.out.println("i = " + i + "!");  
6  }
```

```
$ i = 0!
```

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Organisation, Hinweise

break, continue, if, else

if und else

Mit && sowie | |

Import-Anweisungen

Objekte und Typen

Attribute vs lokale Variable

Enumeration

# if und else



```
1  if(condition1) {  
2      //...  
3  }  
4  else if(condition2) {  
5      //...  
6  }  
7  else {  
8      //...  
9  }
```

```
1  if(condition1) {  
2      //...  
3  }  
4  else {  
5      if(condition2) {  
6          //...  
7      }  
8      else {  
9          //...  
10     }  
11 }
```

# if und else



```
1  boolean isMoving = true;
2  boolean lowOnGas = false;
3
4  if(isMoving == false) {
5      car.startMotor();
6  }
7  else if(lowOnGas == false) {
8      car.drive();
9  }
10 else {
11     car.stopMotor();
12 }
```

```
1  boolean isMoving = true;
2  boolean lowOnGas = false;
3
4  if(isMoving == false) {
5      car.startMotor();
6  }
7  else {
8      if(lowOnGas == false) {
9          car.drive();
10     }
11     else {
12         car.stopMotor();
13     }
14 }
```



# if und else

Mit && sowie ||



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1  int x = 25;
2  int y = 7;
3  int c = 12;
4  Robot r = new Robot(4, 0, DOWN, 12);
5  if((r.getX() == x && r.getY() == y) ||
6      r.getNumberOfCoins() == c) {
7      r.move();
8  }
9  else {
10     r.turnOff();
11 }
```

# if und else

Mit && sowie ||



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1  int x = 25;
2  int y = 7;
3  int c = 12;
4  Robot r = new Robot(4, 0, DOWN, 12);
5  if(r.getX() >= x || r.hasAnyCoins() || -c == x) {
6      r.move();
7  }
8  else {
9      r.turnOff();
10 }
```

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Organisation, Hinweise

break, continue, if, else

if und else

**Import-Anweisungen**

Objekte und Typen

Attribute vs lokale Variable

Enumeration

## ■ Beispiel package-Struktur:

### ■ fopbot

- Robot
- Direction
- World

### ■ other

- MyClass



MyClass.java



```
1  package other;
2
3  import fopbot.Robot;
4
5  public class MyClass {
6      public Robot myRobot;
7
8      public void testCall() {
9          myRobot.move();
10     }
11 }
```

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Organisation, Hinweise

break, continue, if, else

if und else

Import-Anweisungen

Objekte und Typen

- Typen

- Objekte

- Konstruktoren

- Methodenaufruf

- Spezialwert null

- Pass-by-value, Pass-by-reference

Attribute vs lokale Variable

Enumeration



```
1 public class CoolClassName {  
2     public int importantNumber;  
3  
4     public void coolMethodName() {  
5         importantNumber += 1;  
6         boolean b = false;  
7         LemonTree tree = new LemonTree();  
8     }  
9 }
```



```
1 public int importantNumber;  
2 ....  
3 boolean b = ....;  
4 LemonTree tree = ....;
```

### ■ Unterteilung:

- ❑ Primitive Datentypen: `int`, `bool`, usw.
- ❑ Objekt-Datentypen: `LemonTree`, `Robot`, usw.



### Objekt:

Ein **Objekt** ist eine Instanz einer **Klasse**. Es hat seinen eigenen Speicher und die gespeicherten Werte in seinen Attributen können unabhängig von anderen Instanzen geändert werden.

Auf Objekten können **Methoden** aufgerufen werden. Dies geht auf primitiven Datentypen nicht. Welche **Methoden** verfügbar sind, hängt von der **Klasse** ab.

Um ein neues Objekt zu erstellen, muss der **new**-Operator verwendet werden. Hierbei wird der jeweilige **Konstruktor** auf dem neu erstellten Objekt implizit aufgerufen.

Nicht mehr genutzte Objekte werden von Java (irgendwann) automatisch gelöscht [Stichwort Garbage-Collection]



# Objekte und Typen

## Konstruktoren



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



LemonTree.java



```
1  package mypackage;
2
3  public class LemonTree {
4      public int numberOfLemons;
5      public boolean fullyGrown;
6      public String name = "NoName";
7
8      public LemonTree(int lemons) {
9          numberOfLemons = lemons;
10         fullyGrown = false;
11     }
12 }
```



```
1 public LemonTree(int lemons) {  
2     numberOfLemons = lemons;  
3     fullyGrown = false;  
4 }
```

### Syntax Konstruktor:

*Zugriffsmodifikatoren Klassen-Name ( Parameter1, Parameter2, ... )*

- Konstruktor legt fest, mit welchen Parametern ein neues Objekt von der Klasse erstellt werden kann und wie diese das neue Objekt beeinflussen

# Objekte und Typen

## Konstruktoren



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public LemonTree(int lemons) {  
2     numberOfLemons = lemons;  
3     fullyGrown = false;  
4 }
```

```
1 LemonTree myTree = new LemonTree(25);  
2 System.out.println(myTree.numberOfLemons);  
3 System.out.println(myTree.fullyGrown);
```

# Objekte und Typen

## Konstruktoren



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public LemonTree(int lemons) {  
2     numberOfLemons = lemons;  
3     fullyGrown = false;  
4 }
```

```
1 LemonTree myTree = new LemonTree(25);  
2 System.out.println(myTree.numberOfLemons);  
3 System.out.println(myTree.fullyGrown);
```

```
$ 25  
$ false
```

# Objekte und Typen

## Konstrukturen – Standard-Konstruktor



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



LemonTree.java



```
1 public class LemonTree {  
2     public int numberOfLemons;  
3     public boolean fullyGrown;  
4     public String name = "NoName";  
5 }
```

# Objekte und Typen

## Konstruktoren – Standard-Konstruktor



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



LemonTree.java



```
1 public class LemonTree {  
2     public int numberOfLemons;  
3     public boolean fullyGrown;  
4     public String name = "NoName";  
5 }
```

```
1 LemondTree myTree = new LemonTree(????);
```

# Objekte und Typen

## Konstrukturen – Standard-Konstruktor



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



LemonTree.java



```
1  public class LemonTree {  
2      .....  
3  
4      public LemonTree() {  
5          // Automatisch generiert.  
6      }  
7  }
```

# Objekte und Typen

## Konstruktoren – Standard-Konstruktor



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



LemonTree.java



```
1  public class LemonTree {  
2      .....  
3  
4      public LemonTree() {  
5          // Automatisch generiert.  
6      }  
7  }
```

```
1  LemondTree myTree = new LemonTree();
```



# Objekte und Typen

## Konstruktoren – Attribute



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



LemonTree.java



```
1  package mypackage;  
2  
3  public class LemonTree {  
4      public int numberOfLemons;  
5      public boolean fullyGrown;  
6      public String name = "NoName";  
7  
8      public LemonTree(int lemons) {  
9          numberOfLemons = lemons;  
10         fullyGrown = false;  
11     }  
12 }
```

# Objekte und Typen

## Konstruktoren – Attribute



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 LemonTree a = new LemonTree(5);  
2 a.numberOfLemons = -2;  
3 LemonTree b = new LemonTree(6);  
4 b.fullyGrown = true;  
5  
6 System.out.println(a.fullyGrown);  
7 System.out.println(b.numberOfLemons);
```

# Objekte und Typen

## Konstruktoren – Attribute



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 LemonTree a = new LemonTree(5);  
2 a.numberOfLemons = -2;  
3 LemonTree b = new LemonTree(6);  
4 b.fullyGrown = true;  
5  
6 System.out.println(a.fullyGrown);  
7 System.out.println(b.numberOfLemons);
```

```
$ false
```

```
$ 6
```

# Objekte und Typen

## Methodenaufruf



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



LemonTree.java



```
1 public void grow() {  
2     fullyGrown = true;  
3 }  
4 public void water() {  
5     grow(); // Ohne Objekt??  
6     numberOfLemons += 1;  
7 }
```

```
1 LemonTree tree = new LemonTree(0);  
2 tree.grow();  
3 tree.water();
```



```
1 LemonTree tree = new LemonTree(0);
2 LemonTree other = new LemonTree(5);
3 LemonTree third = null;
4
5 third.water(); // NullPointerException
6 third = new LemonTree(25);
7 third.grow();
```

# Objekte und Typen

Pass-by-value, Pass-by-reference



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public void doMorningRoutine(LemonTree tree) {  
2     tree.water();  
3 }
```

```
1 LemonTree a = new LemonTree(0);  
2 doMorningRoutine(a);  
3 System.out.println(a.numberOfLemons);
```

# Objekte und Typen

Pass-by-value, Pass-by-reference



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public void doMorningRoutine(LemonTree tree) {  
2     tree.water();  
3 }
```

```
1 LemonTree a = new LemonTree(0);  
2 doMorningRoutine(a);  
3 System.out.println(a.numberOfLemons);
```

\$ 1

# Objekte und Typen

Pass-by-value, Pass-by-reference



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public void doMorningRoutine(int num) {  
2     num += 1;  
3 }
```

```
1 int a = 0;  
2 doMorningRoutine(a);  
3 System.out.println(a);
```



# Objekte und Typen

Pass-by-value, Pass-by-reference



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public void doMorningRoutine(int num) {  
2     num += 1;  
3 }
```

```
1 int a = 0;  
2 doMorningRoutine(a);  
3 System.out.println(a);
```

\$ 0

# Objekte und Typen

Pass-by-value, Pass-by-reference



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Ein Objekt belegt einen bestimmten Speicherplatz
- Mehrere Variablen können auf dasselbe Objekt verweisen
- Objekte werden per Referenz übergeben! Es wird keine Kopie erstellt.

# Objekte und Typen

Pass-by-value, Pass-by-reference



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 LemonTree a = new LemonTree(0);  
2 LemonTree b = a;  
3 LemonTree c = b;  
4  
5 a.grow();  
6 System.out.println(a.fullyGrown);  
7 System.out.println(b.fullyGrown);  
8 System.out.println(c.fullyGrown);
```

# Objekte und Typen

Pass-by-value, Pass-by-reference



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 LemonTree a = new LemonTree(0);
2 LemonTree b = a;
3 LemonTree c = b;
4
5 a.grow();
6 System.out.println(a.fullyGrown);
7 System.out.println(b.fullyGrown);
8 System.out.println(c.fullyGrown);
```

```
$ true
$ true
$ true
```

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Organisation, Hinweise

break, continue, if, else

if und else

Import-Anweisungen

Objekte und Typen

Attribute vs lokale Variable

Initialisierung

Enumeration

# Attribute vs lokale Variable

## Initialisierung



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public class DataContainer {  
2     public int data; // Attribut  
3  
4     public void printData() {  
5         System.out.println(data);  
6     }  
7 }
```

```
1 public void abc() {  
2     int data; // Lokale Variable  
3     System.out.println(data);  
4 }
```

# Attribute vs lokale Variable

## Initialisierung — Standardwerte



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Name	Typ	Attribut Standardwert
boolean	Wahr/Falsch	false
int	Ganze Zahl	0
double	Gleitkommazahl	0.0
String	Zeichenkette	null
LemonTree	Eigene Klasse	null

# Attribute vs lokale Variable

## Initialisierung – Uninitialized local variable

```
1  int data; // Lokale Variable
2
3  if(....) {
4      data = 5;
5  }
6
7  System.out.println(data); // ERROR!
```



# Attribute vs lokale Variable

## Initialisierung – Uninitialized local variable



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1  int data; // Lokale Variable
2
3  while(...) {
4      data = 5;
5  }
6
7  System.out.println(data); // ERROR!
```

# Attribute vs lokale Variable

## Initialisierung – Uninitialized local variable



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1  int data = 0; // Lokale Variable
2
3  while(...) {
4      data = 5;
5  }
6
7  System.out.println(data); // OK!
```

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Organisation, Hinweise

break, continue, if, else

if und else

Import-Anweisungen

Objekte und Typen

Attribute vs lokale Variable

Enumeration

Beispiel aus der Welt der FOPBot-Roboter

Notation

Methoden für Enumerationen

Rückgabe

# Enumeration

Beispiel aus der Welt der FOPBot-Roboter



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

## Enumeration Direction

- Direction.UP
- Direction.RIGHT
- Direction.DOWN
- Direction.LEFT

```
1  public enum Direction {  
2      UP,  
3      RIGHT,  
4      DOWN,  
5      LEFT  
6  }
```

# Enumeration

Beispiel aus der Welt der FOPBot-Roboter



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Aufzählung von Konstanten
- können wie andere Konstanten verwendet werden  
z.B. `true` oder `1` – ohne jeweilige Operationen
- Bedeutung der Konstanten wird von Entwickler festgelegt

## Operationen



==

```
1 Robot robot = new Robot(0, 0, UP, 42);
2 Direction direction = robot.getDirection();
3 ...
4 if (direction == UP) {
5     ...
6 }
7 ...
8 if (direction.equals(RIGHT)) {
9     ...
10 }
```



Anstatt z.B. `Direction.UP` kann auch nur `UP` geschrieben werden!

- einzelne Konstante  
`import static fopbot.Direction.UP;`
- alle Konstanten einer Enumeration  
`import static fopbot.Direction.*;`



### Syntax Enumeration:

*Zugriffsmodifikatoren Enumeration-Name { Konstante1, Konstante2, ... }*

### Best Case

- Namen für Enumerationen → Pascal Case – wie bei Namen für Klassen  
z.B. MyDirection
- Namen für Konstanten → Upper Case  
z.B. MY\_UP, MY\_RIGHT, MY\_DOWN und MY\_LEFT





`ordinal()` für Konstante

- liefert Position der Konstante in Enumeration

### Beispiel für `Direction`

- `Direction.UP.ordinal() → 0`
- `Direction.RIGHT.ordinal() → 1`
- `Direction.DOWN.ordinal() → 2`
- `Direction.LEFT.ordinal() → 3`

# Enumeration

## Methoden für Enumerationen



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

`values()`

- liefert Array mit Konstanten der Enumeration

### Beispiel für `Direction`

- `Direction.values()` → `[UP, RIGHT, DOWN, LEFT]`
- `Direction.values()[0]` → `UP`

# Enumeration

## Methoden für Enumerationen



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

`valueOf(String)`

- liefert Konstante mit gegebenem Namen

## Beispiel für `Direction`

- `Direction.valueOf("UP")` → UP (Konstante)



```
1 public boolean checkNumber(int a) {  
2     return a > 5;  
3 }
```

```
1 boolean resultA = checkNumber(4);  
2 boolean resultB = checkNumber(6);  
3 System.out.println(resultA + " & " + resultB);
```



```
1 public boolean checkNumber(int a) {  
2     return a > 5;  
3 }
```

```
1 boolean resultA = checkNumber(4);  
2 boolean resultB = checkNumber(6);  
3 System.out.println(resultA + " & " + resultB);
```

\$ false & true



```
1 public LemonTree createATree(int a) {  
2     return new LemonTree(a + 42);  
3 }
```

```
1 LemonTree resultA = createATree(4);  
2 LemonTree resultB = createATree(6);
```



```
1 public LemonTree createATree(int a) {  
2     LemonTree localVar = new LemonTree(a + 42);  
3     localVar.grow();  
4     localVar.fullyGrown = false;  
5  
6     return localVar;  
7 }
```



```
1 public int runMe(int a) {  
2     if(a > 5) {  
3         return 3;  
4     }  
5     else {  
6         a = 2;  
7     }  
8     // ERROR! Return benötigt!  
9 }
```





```
1 public void growIt(LemonTree tree) {  
2     if(tree == null) {  
3         return; // Beendet Methode direkt  
4     }  
5     tree.grow();  
6     // Kein return benötigt! Nur optional  
7 }
```



---

# Live-Coding!