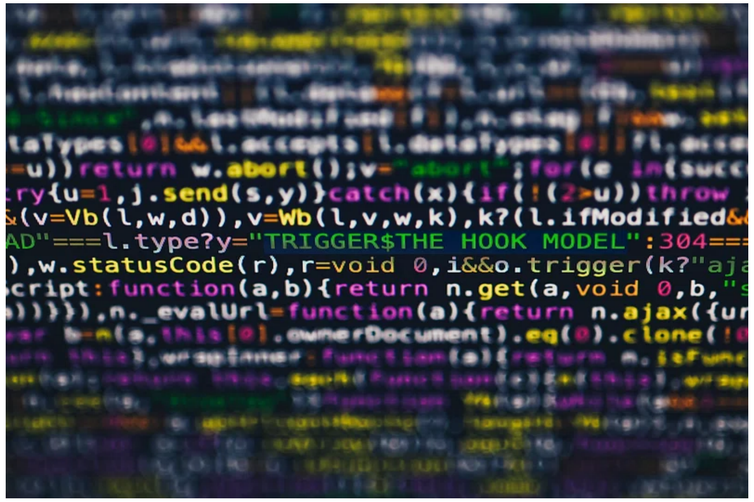
Vorkurs Informatik

Vorlesung 2: Objekt-Orientierung in Java



Prof. Dr. Nils Jansen
Chair of Artificial Intelligence and Formal Methods

RUHR UNIVERSITÄT BOCHUM

RUB

Die heutige Agenda

Recap der Ersten Vorlesung

Methoden

Arrays Continued

Klassen und Objekte

Objektorienterung in Java

Today’s Agenda

Recap der Ersten Vorlesung

Methoden

Arrays Continued

Klassen und Objekte

Objektorienterung in Java

Last Week’s Agenda

Erste Schritte in der JAVA Programmierung

Einfache Datentypen und Berechnungen

Arrays und Kontrollanweisungen

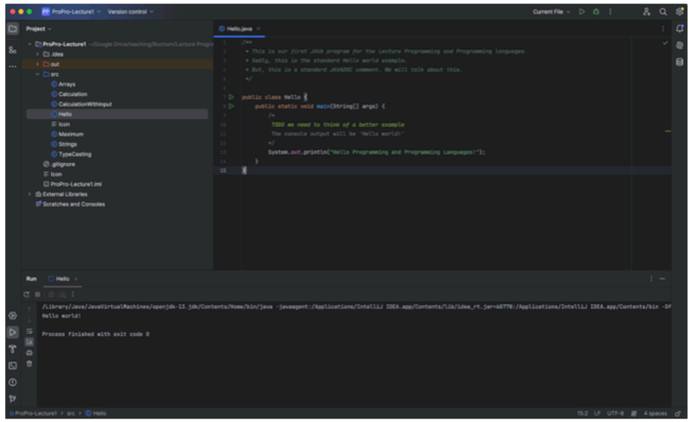
Recall: Hello World?

Filename: Hello.java

```java
/**
 * This is our first JAVA program for the Lecture Programming and Programming languages.
 * Sadly, this is the standard Hello world example.
 * But, this is a standard JAVADOC comment. We will talk about this.
 */

public class Hello {
    public static void main(String[] args) {
        /*
         TODO we need to think of a better example
         The console output will be 'Programming and Programming Languages!'
        */
        System.out.println("Hello Programming and Programming Languages!");
    }
}
```

Integrated Development Environment (IDE)

IntelliJ IDEA



https://www.jetbrains.com/idea/

I hope you all have been able to run a JAVA program so far!

Recap - Hello World?

Filename: Hello.java

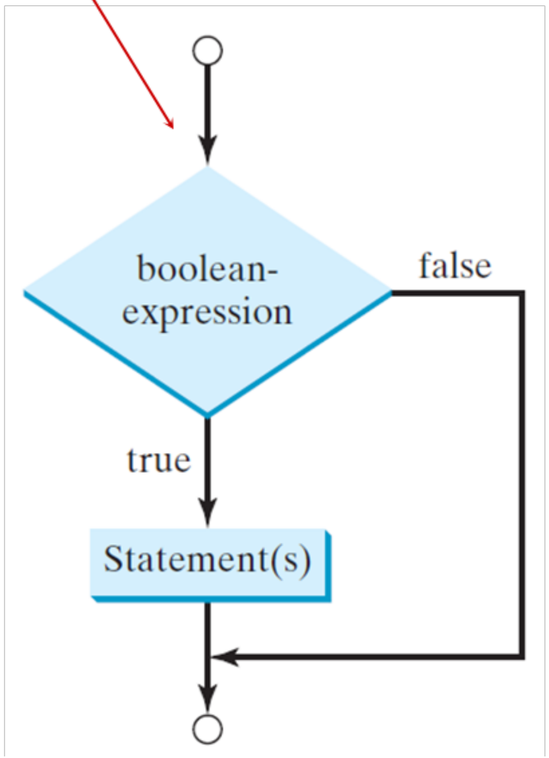
```java
/**
 * This is our first JAVA program.
 * Sadly, this is the standard Hello world example.
*/

public class Hello {
    public static void main(String[] args) {
        /*
        The console output will be ‘Hello Prepcourse CS!'
        */
        System.out.println("Hello Prepcourse CS!”);
    }
}
```

Recap: Einfache Datentypen

- **Integer numbers**: `byte, short, int, long`
- **Floating point numbers:** `float, double`
- **Boolean:** `boolean`
- **Characters:** `char`
- **Strings:** `String`

Recap: `if` Anweisung

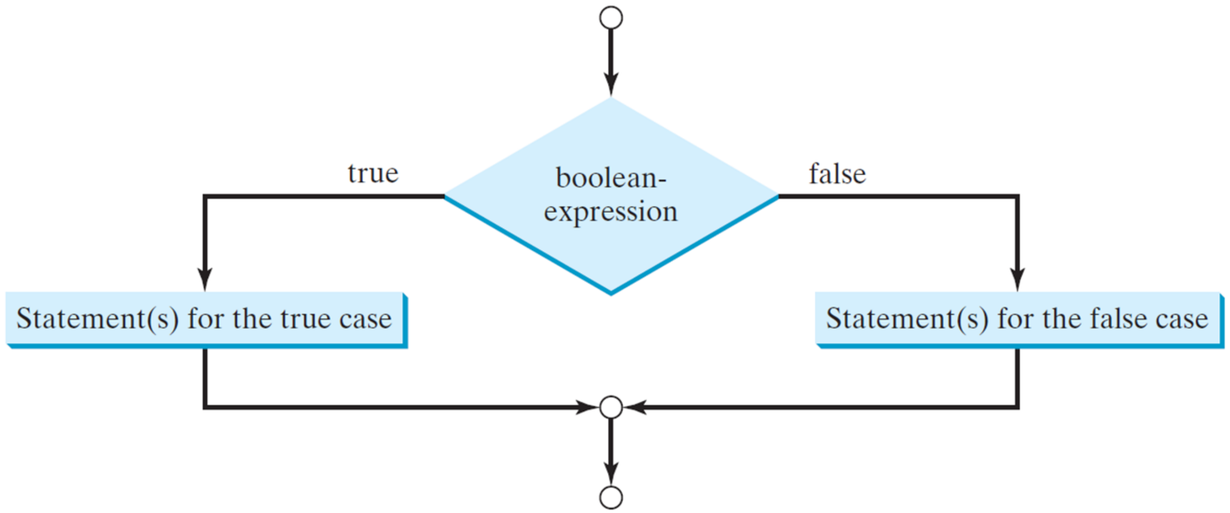
```
if (boolean-expression) {
  statement(s);
}
```




```java
if (i==5) {
    System.out.println("i is 5");
}

if (i!=5) {
    System.out.println("i is not 5");
}

if (i<5) {
    System.out.println("i is smaller than 5");
}
```

Recap: `if` Anweisung

```
if (boolean-expression) {
  statement(s)-for-the-true-case;
}
else {
  statement(s)-for-the-false-case;
}
```



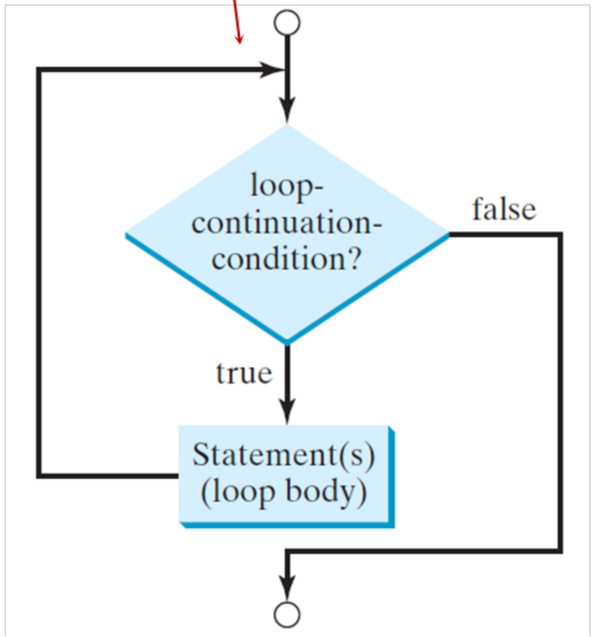

```
if (x==5) {
    System.out.println("x is 5");
}
else {
    System.out.println("x is not 5");
}
```

Recap: `switch` Anweisung

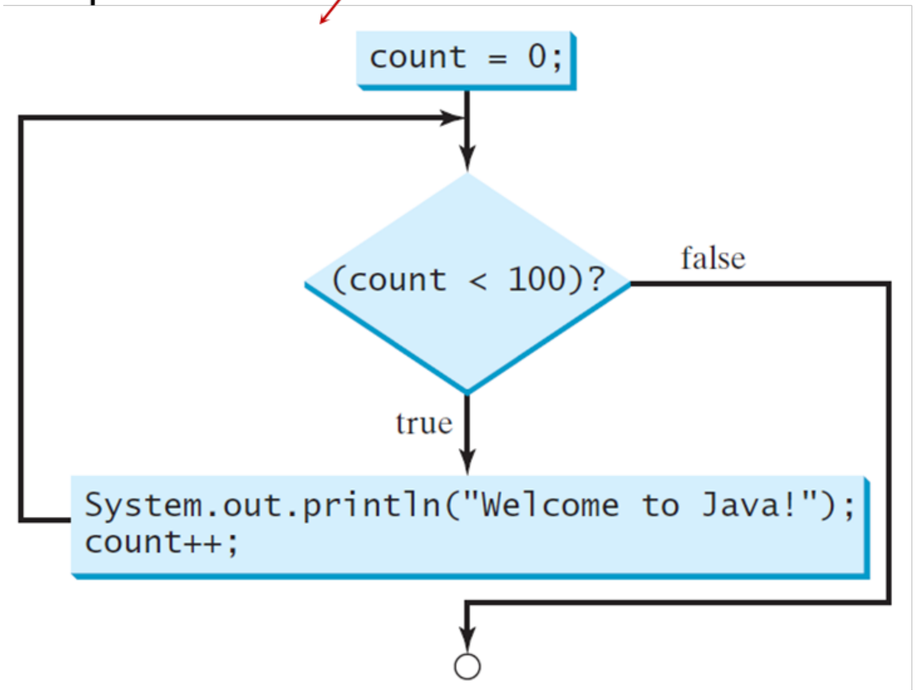
```
switch (x) {

    case 0,1,2,3,4 -> System.out.println("x is smaller than 5”);

    case 5 -> System.out.println("x is 5”);

    default -> System.out.println("x is larger than 5");
}
```

Recap: `while` Schleife

```
while (loop-continuation-condition)
{
  // loop-body;
  Statement(s);
}
```



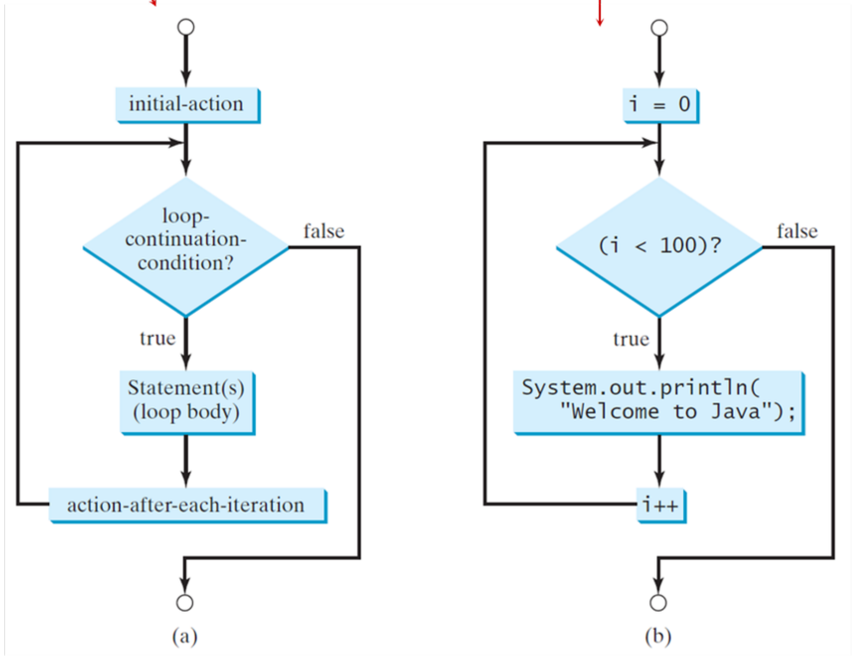

```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java!");
  count++;
}
```



Recap: `for`-Schleife

```
for (initial-action; loop-
    continuation-condition; action-
    after-each-iteration) {
  // loop body;
  Statement(s);
}
```

```
int i;
for (i = 0; i < 100; i++) {
  System.out.println(
    "Welcome to Java!");
}
```



Recap: Array Deklaration, Erzeugung, Initialisierung

```
int[] x;

x = new int[10];

x[0]=1;
x[1]=2;
x[2]=3;
x[3]=4;
x[4]=5;
x[5]=6;
x[6]=7;
x[7]=8;
x[8]=9;
x[9]=10;

System.out.println(Arrays.toString(x));
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
int[] x = new int[10];
```

```
int[] x = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Recap: `while`-Schleife und Array

```
int[] x;

 x = new int[5];

 x[0] = 1;
 x[1] = 2;
 x[2] = 3;
 x[3] = 4;
 x[4] = 5;

 System.out.println(Arrays.toString(x));
```

```
[1, 2, 3, 4, 5]
```

```
 //while-loop to iterate over array
 int count = 0;
 System.out.println("while-loop to iterate over array: ");
 while (count < x.length) {
   System.out.println(x[count]);
   count++;
 }
```

```
[1, 2, 3, 4, 5]
```

Recap: for-Schleife und Array

```java
int[] x;

x = new int[5];

x[0] = 1;
x[1] = 2;
x[2] = 3;
x[3] = 4;
x[4] = 5;

System.out.println(Arrays.toString(x));

//for-loop to iterate over array
System.out.println("for-loop to iterate over array: ");

for (int i = 0; i < 5; i++) {
    System.out.print(x[i]);
}

//enhanced for-loop to iterate over array
System.out.print("enhanced for-loop to iterate over array: ");
for (int element : x) {
    System.out.print(element);
}
```

```
[1, 2, 3, 4, 5]
```

```
[1, 2, 3, 4, 5]
```

```
[1, 2, 3, 4, 5]
```

Take Homes

- Berechnungen und numerische Operatoren
- Typen und Typkonvertierung
- Eingabe/Ausgabe
- Die `Math` class
- booleans und relationale Operatoren
- Arrays
- Kontrollstrukturen

Today’s Agenda

Recap der Ersten Vorlesung

Methoden

Arrays Continued

Klassen und Objekte

Objektorienterung in Java

Methoden können verwendet werden, um redundante Codierung zu reduzieren und die Wiederverwendung von Code zu ermöglichen. Methoden können auch verwendet werden, um Code zu modularisieren und die Qualität des Programms zu verbessern.

Folgendes Problem…

Berechnen Sie die Summe der ganzen Zahlen
von 1 bis 10, von 20 bis 30 bzw. von 35 bis 45.

Summe ganzer Zahlen

Filename: Summation.java

```java
public class summation {

    public static void main(String[] arguments) {
        int sum = 0;
        for (int i = 1; i <= 10; i++)
            sum += i;
        System.out.println("Sum from 1 to 10 is " + sum);

        sum = 0;
        for (int i = 20; i <= 30; i++)
            sum += i;
        System.out.println("Sum from 20 to 30 is " + sum);

        sum = 0;
        for (int i = 35; i <= 45; i++)
            sum += i;
        System.out.println("Sum from 35 to 45 is " + sum);
    }
}
```

```
Sum from 1 to 10 is 55
```

```
Sum from 1 to 10 is 275
```

```
Sum from 1 to 10 is 225
```

Warum ist das dumm?

Summe ganzer Zahlen mittels einer Methode

Filename: SummationMethods.java

```
package methods;

public class SummationMethods {

    public static void main(String[] arguments) {
        System.out.println("Sum from 1 to 10 is " + sumRange(1, 10));
        System.out.println("Sum from 20 to 30 is " + sumRange(20, 30));
        System.out.println("Sum from 35 to 40 is " + sumRange(35, 40));
   }

    //computes the sum of all numbers within the range i1 to i2
    public static int sumRange(int i1, int i2) {
       int sum = 0;
        for (int i = i1; i <= i2; i++) {
            sum += i;
        }
        return sum;
    }
}
```

Methoden in Java

Eine Methode ist eine Sammlung von Anweisungen, die gruppiert werden, um eine bestimmte Operation auszuführen.

```
public static int sumRange(int i1, int i2) {
    int sum = 0;
     for (int i = i1; i <= i2; i++) {
         sum += i;
     }
     return sum;
 }

int sum = sumRange(1,10);
```

Today's Agenda

Recap der Ersten Vorlesung

Methoden

Arrays Continued

Klassen und Objekte

Objektorienterung in Java

Zweidimensionale Arrays
Matritzen

Zweidimensionale Arrays

```
import java.util.Arrays;

public class Matrix {
    public static void main(String[] args) {
        int[][] matrix;
        matrix = new int[3][2];
        matrix [0][0] = 1;
        matrix [0][1] = 2;
        matrix [1][0] = 3;
        matrix [1][1] = 4;
        matrix [2][0] = 5;
        matrix [2][1] = 6;
        System.out.println("Matrix: " + Arrays.deepToString(matrix));
        Arrays.fill(matrix[0], 1);
        Arrays.fill(matrix[1], 2);
        Arrays.fill(matrix[2], 2);
        System.out.println("Matrix: " + Arrays.deepToString(matrix));
    }
}
```

```
Matrix: [[1, 2], [3, 4], [5, 6]]
```

```
Matrix: [[1, 1], [2, 2], [3, 3]]
```

Eine kleine Übung

Entwickeln Sie ein Programm, dass

1. Zwei Matritzen erstellt die mit ganzzahligen Zufallszahlen gefüllt sind
2. Diese Matritzen miteinander multipliziert und das Ergebnis ausgibt

$$A = \begin{bmatrix} 1 & 5 \\ 2 & 3 \\ 1 & 7 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 & 3 & 7 \\ 5 & 2 & 8 & 1 \end{bmatrix} \quad A \cdot B = \begin{bmatrix} 26 & 12 & 43 & 12 \\ 17 & 10 & 30 & 17 \\ 36 & 16 & 59 & 14 \end{bmatrix} = C$$

Matrixmultiplikation

```
public static int[][] multiplyMatrices(int[][] matrixA, int[][] matrixB) {
        //assert that the matrices are compatible for multiplication
        assert matrixA[0].length == matrixB.length;
        //creating another matrix to store the multiplication of two matrices
        //the correct number of rows and columns
        int rowC = matrixA.length;
        int colC = matrixB[0].length;
        int[][] matrixC = new int[rowC][colC];

        //multiplying and printing multiplication of 2 matrices
        int cell = 0;
        for (int row = 0; row < matrixC.length; row++) {
            for (int col = 0; col < matrixC[row].length; col++) {
                cell = 0;
                for (int i = 0; i < matrixB.length; i++) {
                    cell += matrixA[row][i] * matrixB[i][col];
                }
                matrixC[row][col]=cell;
            }
        }
        return matrixC;
    }
```

First Programming Demonstration

1. **Summierung**
2. **Matrixmultiplikation mit 2-Dimensionalen Arrays**

Today's Agenda

Recap der Ersten Vorlesung

Methoden

Arrays Continued

Klassen und Objekte

Objektorienterung in Java

Objekte

Klassen

Felder (Fields)

Verhalten

Objekte

In der echte Welt:

- Haben einen *Zustand*
- Haben ein *Verhalten*

In Software:

- Haben einen Zustand *(Eigenschaften)* (gespeichert in ***fields***)
- Haben ein Verhalten (beschrieben von **Methoden**)

Beispiel Objekt

Door from: https://commons.wikimedia.org/wiki/File:Door.png

Beispiel Objekt

Door from: https://commons.wikimedia.org/wiki/File:Door.png

State:

- Is it closed? `isClosed`
- Is it locked? `isLocked`
- Which material? …

Behavior:

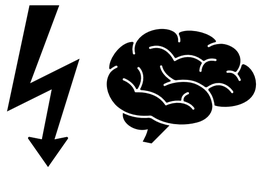
- Close it / Open it
- Lock it / Unlock it (with a key?)
- A door cannot change its material…

Klassen

- Objekte klassifizieren
- Beschreiben die Gemeinsamkeiten von Mengen ähnlicher Objekte

- Beschreiben Sie eine Blaupause (Klasse) als Programmierer
- Lassen Sie Ihr Programm eine beliebige Anzahl von Instanzen "ausarbeiten"

- Nicht alle Türen haben Schlösser?
- Manche Türen sind eine Art Türen?



Doors from: https://www.flickr.com/photos/sackton/7580307812/

Objektorientierte Programme

Objekte sind die Bausteine von Softwaresystemen

- Ein Programm ist eine Sammlung von interagierenden Objekten
- Objekte arbeiten zusammen, um eine Aufgabe zu erledigen
- Um dies zu tun, kommunizieren sie, indem sie die Methoden des anderen aufrufen (oder aufrufen)

Today's Agenda

Recap der Ersten Vorlesung

Methoden

Arrays Continued

Klassen und Objekte

Objektorienterung in Java

Fields

Statische fields

Methoden

Konstruktoren

Klassen

Methoden können die Werte von Feldern ändern.

Konstruktoren (besondere Methoden) werden impliziert aufgerufen (via `new`) um eine Instanz (Objekt) einer Klasse zu erzeugen.

Felder (Instanzvariablen) werden meist vom Konstruktor initialisiert.

Statische Felder (Klassenvariablen) werden von allen Objekten einer Klasse geteilt

Statische Methoden (Funktionen) können auf keine Felder eines Objekts zugreifen.

Beispielobjekt

Door from: https://commons.wikimedia.org/wiki/File:Door.png

State:

- Is it closed? `isClosed`
- Is it locked? `isLocked`
- Which material? …

Behavior:

- Close it / Open it
- Lock it / Unlock it (with a key?)
- A door cannot change its material…

Die Door Klasse: Felder

```java
package sjunges.oolectures.DoorMain;

/**
 * This comment describes the Door class
 */
public class Door {
    /** True iff the door is closed. */
    boolean isClosed;
    /** True iff the door is locked. */
    boolean isLocked;
    /** A textual description of the material */
    final String material;


    …
}
```

Felder erfordern einen Typ und einen Namen. Finale Felder ändern ihren Wert nie, nachdem sie initialisiert wurden.

Beispieltypen: boolean, int, String

Die Door Klasse: Felder & Konstruktor

```java
public class Door {
    boolean isClosed;
    boolean isLocked;
    final String material;

    /**
     * Constructs a closed, unlocked door with the specified material
     * @param materialForDoor Textual description of the material
     */
    public Door(String materialForDoor) {
        isClosed = true;
        isLocked = false;
        material = materialForDoor;
    }
}
```

Objekte werden aus Klassen erstellt, indem der Konstruktor aufgerufen wird:

Eine Methode mit demselben Namen wie die Klasse.

Nutzen der `Door` Klasse in der main Methode (1)

```
public class DoorMain {
    public static void main(String[] args) {
    Door d1 = new Door("wooden");
    Door d2 = new Door("iron");
    System.out.println(d1.material);
    System.out.println(d2.material);
    }
}
```

Mit Konstruktoren ('new') können wir Instanzen einer Klasse (Objekte) erstellen.

Nutzen der `Door` Klasse in der main Methode (1)

```
public class DoorMain {
    public static void main(String[] args) {
    Door d1 = new Door("wooden");
    Door d2 = new Door("iron");
    System.out.println(d1.material);
    System.out.println(d2.material);
    }
}
```

Mit Konstruktoren ('new') können wir Instanzen einer Klasse (Objekte) erstellen.

```
Compile & Run DoorMain:
wooden
iron

Successful
```

Nutzen der `Door` Klasse in der main Methode (2)

```
public class DoorMain {
    public static void main(String[] args) {
    Door d1 = new Door("wooden");
    Door d2 = new Door("iron");
    d1.material = "steel";
    }
}
```

Man darf finale Felder nicht verändern.

Nutzen der `Door` Klasse in der main Methode (2)

```
public class DoorMain {
    public static void main(String[] args) {
    Door d1 = new Door("wooden");
    Door d2 = new Door("iron");
    d1.material = "steel";
    }
}
```

Man darf finale Felder nicht verändern.

```
Compile & Run DoorMain:

Error
cannot assign a value to final variable material
```

Using the Door Class in the main method (2)

```
public class DoorMain {
    public static void main(String[] args) {
    Door d1 = new Door("wooden");
    Door d2 = new Door("iron");
    System.out.println(d1.isClosed);
    System.out.println(d2.isClosed);
    d1.isClosed = false; // this is problematic. Do not do this.
    System.out.println(d1.isClosed);
    System.out.println(d2.isClosed);
    }
}
```

Man kann auf Felder eines Objekts zugreifen via objektName.feldName

Using the Door Class in the main method (2)

```
public class DoorMain {
    public static void main(String[] args) {
    Door d1 = new Door("wooden");
    Door d2 = new Door("iron");
    System.out.println(d1.isClosed);
    System.out.println(d2.isClosed);
    d1.isClosed = false; // this is problematic. Do not do this.
    System.out.println(d1.isClosed);
    System.out.println(d2.isClosed);
    }
}
```

Man kann auf Felder eines Objekts zugreifen via objektName.feldName

```
Compile & Run DoorMain:
true
true
false
true

Successful
```

Datenkapselung (erste Runde)

Man kann den Zustand eines Objekts über sein Verhalten verändern

Die Tür sollte nur geöffnet werden, wenn die Tür nicht verschlossen ist. Dies sollte immer vom Programmierer von Door sichergestellt werden!

Wie können wir das sicherstellen?

Methoden (1)

```
public class Door {
    boolean isClosed;
    boolean isLocked;
    String material;


    …

    /** Open the door. Impossible if the door is locked. */
    void open() {
        if (!isLocked) {
            isClosed = false;
        }
    }

    /** Close the door. Impossible if the door is locked. */
    void close() {
        if (!isLocked) {
            isClosed = true;
        }
    }
}
```

Methoden (2)

```
public class Door {
    /**
     * Lock the the door, but only if you provide the right key.
     * @param key 42 is the right key.
     */
    void lock(int key) {
        // Apparently, all doors have the same key to keep the slides simple.
        if (key == 42) {
            isLocked = true;
        }
    }

    /**
     * Unlock the door, but only if you provide the right key.
     * @param key 42 is the right key.
     */
    void unlock(int key) {
        if (key == 42) {
            isLocked = false;
        }
    }
}
```

Nutzen der `Door` Methoden

```
public static void main(String[] args) {
    Door d1 = new Door("wooden");
    System.out.println(d1.isClosed);
    d1.lock(42);
    d1.open();
    System.out.println(d1.isClosed);




}
```

Nutzen der Door Methoden

```
public static void main(String[] args) {
    Door d1 = new Door("wooden");
    System.out.println(d1.isClosed);
    d1.lock(42);
    d1.open();
    System.out.println(d1.isClosed);




}
```

```
Compile & Run DoorMain:
true
true



Successful
```

Nutzen der Door Methoden

```
public static void main(String[] args) {
    Door d1 = new Door("wooden");
    System.out.println(d1.isClosed);
    d1.lock(42);
    d1.open();
    System.out.println(d1.isClosed);
    d1.unlock(41);
    d1.open();
    System.out.println(d1.isClosed);


}
```

Methoden könne aufgerufen werden durch objektName.methodenName(…).
Methoden ändern (nur) Felder von dem Objekt objectName.

Nutzen der `Door` Methoden

```
public static void main(String[] args) {
    Door d1 = new Door("wooden");
    System.out.println(d1.isClosed);
    d1.lock(42);
    d1.open();
    System.out.println(d1.isClosed);
    d1.unlock(41);
    d1.open();
    System.out.println(d1.isClosed);


}
```

```
Compile & Run DoorMain:
true
true
true


Successful
```

Methoden könne aufgerufen werden durch objektName.methodenName(…).
Methoden ändern (nur) Felder von dem Objekt objectName.

Nutzen der Door Methoden

```
public static void main(String[] args) {
    Door d1 = new Door("wooden");
    System.out.println(d1.isClosed);
    d1.lock(42);
    d1.open();
    System.out.println(d1.isClosed);
    d1.unlock(41);
    d1.open();
    System.out.println(d1.isClosed);
    d1.isClosed = false; // No No No
    System.out.println(d1.isClosed);
}
```

Nutzen der Door Methoden

```
public static void main(String[] args) {
    Door d1 = new Door("wooden");
    System.out.println(d1.isClosed);
    d1.lock(42);
    d1.open();
    System.out.println(d1.isClosed);
    d1.unlock(41);
    d1.open();
    System.out.println(d1.isClosed);
    d1.isClosed = false; // No No No
    System.out.println(d1.isClosed);
}
```

```
Compile & Run DoorMain:
true
true
true
false

Successful
```

Modifier

```
public class Door {
    private boolean isClosed;
    private boolean isLocked;
    private String material;
    …

    /** Open the door. Impossible if the door is locked. */
    public void open() {
        if (!isLocked) {
            isClosed = false;
        }
    }




}
```

Modifier

```
public class Door {
    private boolean isClosed;
    private boolean isLocked;
    private String material;
    …

    /** Open the door. Impossible if the door is locked. */
    public void open() {
        if (!isLocked) {
            isClosed = false;
        }
    }

    /** Close the door. Impossible if the door is locked. */
    public void close() {
        if (!isLocked) {
            isClosed = true;
        }
    }
}
```

Merkregel: Alle Felder private und nur das notwendige Verhalten ist public!

Nutzen der Door Klasse mit Modifiern

```
public class DoorMain {
    public static void main(String[] args) {
      Door d1 = new Door("wooden");
      d1.isClosed = false; // this is problematic and no longer compiles :-)
      System.out.println(d1.isClosed); // neither does this :-(
   }
}
```

Nutzen der Door Klasse mit Modifiern

```java
public class DoorMain {
    public static void main(String[] args) {
      Door d1 = new Door("wooden");
      d1.isClosed = false; // this is problematic and no longer compiles :-)
      System.out.println(d1.isClosed); // neither does this :-(
   }
}
```

```
Compile & Run DoorMain:

Error
isClosed has private access in package.Door
```

Modifier in Java

From:	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
(default)	Y	Y	N	N
private	Y	N	N	N

Nutzen der Door Methoden (getter)

```
public class Door {
    public boolean getIsClosed() {
        return isClosed;
    }

    public boolean getIsLocked() {
        return isLocked;
    }
}
```

Methoden könenn Werte zurückgeben.

```
public static void main(String[] args) {
    Door d1 = new Door("wooden");
    System.out.println(d1.getIsClosed());

    …
}
```

Public getters ermöglichen Lesezugriff zu private Feldern

Nutzen der Door Methoden (getter)

```
public class Door {
    public boolean getIsClosed() {
        return isClosed;
    }

    public boolean getIsLocked() {
        return isLocked;
    }
}
```

Methoden könenn Werte zurückgeben.

```
public static void main(String[] args) {
    Door d1 = new Door("wooden");
    System.out.println(d1.getIsClosed());

    …
}
```

Public getters ermöglichen Lesezugriff zu private Feldern

```
Compile & Run DoorMain:
true


Successful
```

toString

```
public static void main(String[] args) {
    Door d1 = new Door("wooden");
    d1.lock(42);
    System.out.println(d1.toString());
    d1.unlock(42);
    d1.open();
    System.out.println(d1.toString());
    // And with a touch of magic
    // explained in another lecture:
    System.out.println(d1);
}
```

toString

```
public static void main(String[] args) {
    Door d1 = new Door("wooden");
    d1.lock(42);
    System.out.println(d1.toString());
    d1.unlock(42);
    d1.open();
    System.out.println(d1.toString());
    // And with a touch of magic
    // explained in another lecture:
    System.out.println(d1);
}
```

```
Compile & Run DoorMain:
The wooden door is locked and closed.
The wooden door is unlocked and open.
The wooden door is unlocked and open.

Successful
```

Methoden für Hilfsfunktionen

```
public class Door {

    public void lock(int key) {
        if (isKeyCorrect(key)) {
            isLocked = true;
        }
    }


    public void unlock(int key) {
        if (isKeyCorrect(key)) {
            isLocked = false;
        }
    }

    private boolean isKeyCorrect(int key) {
        return (key == 42);
    }
}
```

Private Methoden helfen, den Code zu strukturieren.

Second Programming Demonstration

The Door Class

Java Class Libraries

Java stellt **class libraries** zur Verfügung

- **Java APIs (Application Programming Interfaces)**

Um Java effektiv zu nutzen, braucht man:

- Java Programmiersprache language
- class libraries

Die Verwendung von Java-API-Klassen, anstatt Ihre eigenen Versionen zu schreiben, kann

- Programm **Performance verbessern**, da API Klassen erstellt wurden um effizient zu sein
- Programm **Portabilität verbessern,** weil API Klassen in jeder Java Implementierung verfügbar sind.

Summary

Recap der Ersten Vorlesung

Methoden

Arrays Continued

Klassen und Objekte

Objektorienterung in Java