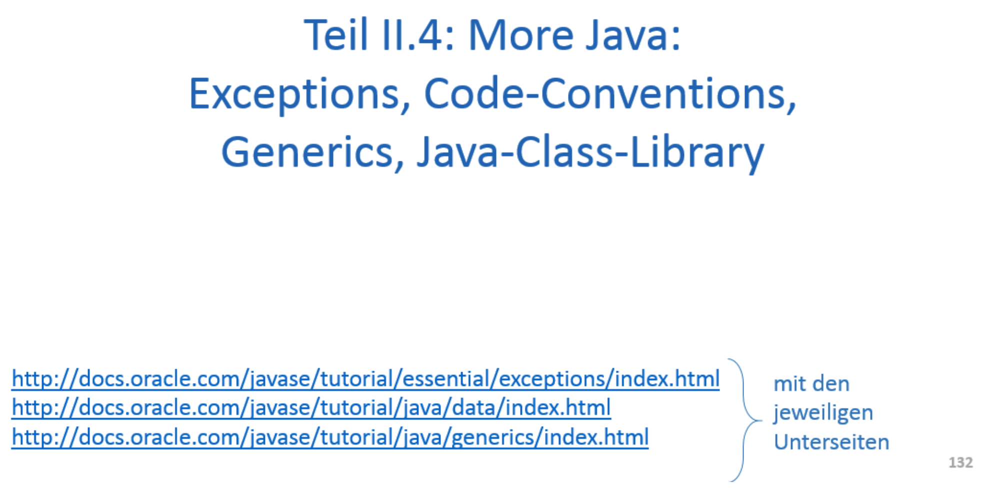
Script generated by TTT

Title: groh: profile1 (03.07.2015)

Date: Fri Jul 03 15:24:20 CEST 2015

Duration: 22:55 min

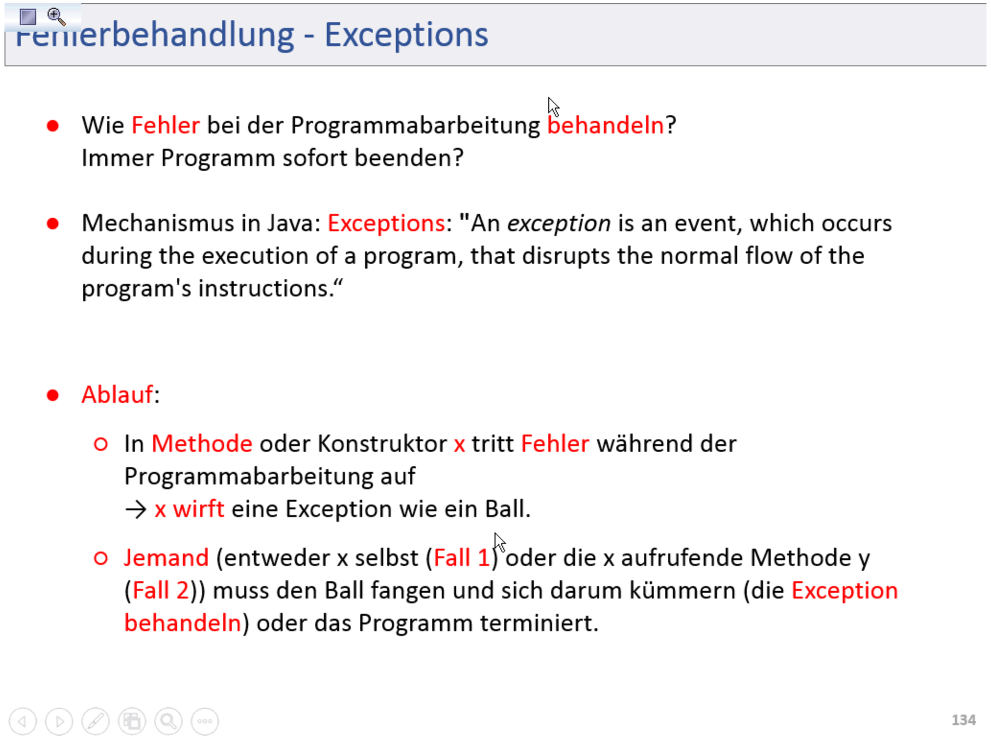
Pages: 31



Teil II.4: More Java: Exceptions, Code-Conventions, Generics, Java-Class-Library

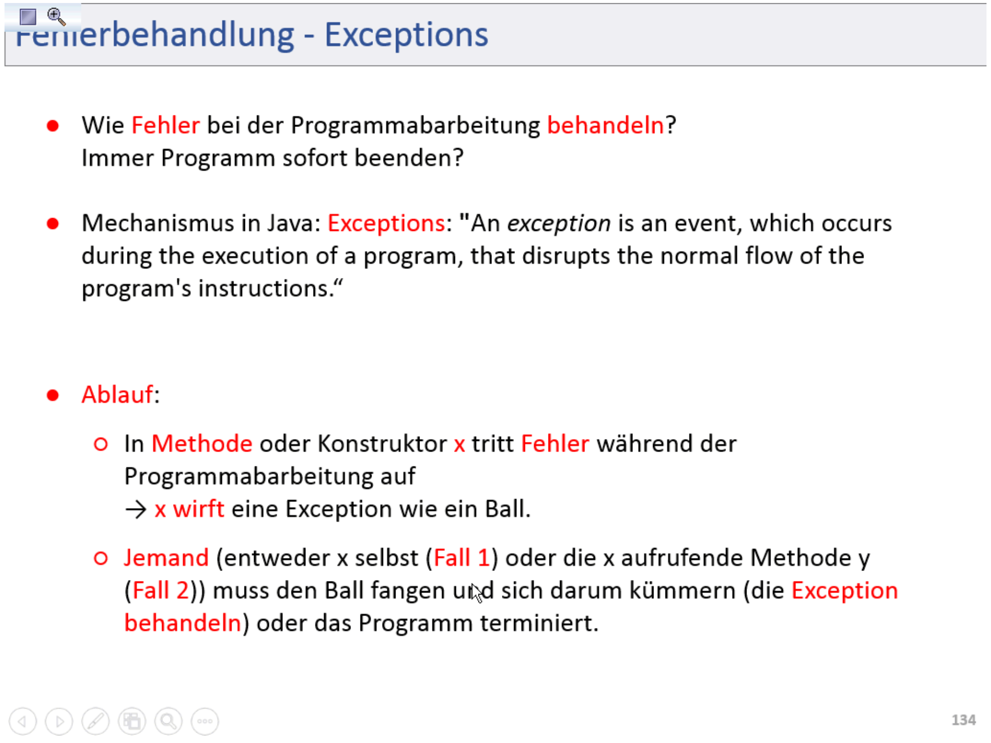
http://docs.oracle.com/javase/tutorial/essential/exceptions/index.html
http://docs.oracle.com/javase/tutorial/java/data/index.html
http://docs.oracle.com/javase/tutorial/java/generics/index.html

mit den jeweiligen Unterseiten



Fehlerbehandlung - Exceptions

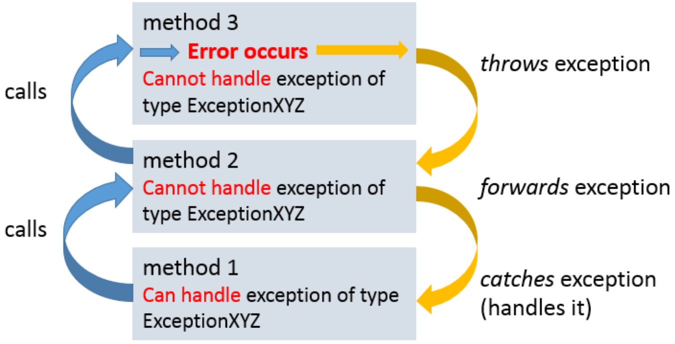
- Wie Fehler bei der Programmabarbeitung behandeln?
  Immer Programm sofort beenden?
- Mechanismus in Java: Exceptions: "An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.“
- Ablauf:
  - In Methode oder Konstruktor x tritt Fehler während der Programmabarbeitung auf
    → x wirft eine Exception wie ein Ball.
  - Jemand (entweder x selbst (Fall 1) oder die x aufrufende Methode y (Fall 2)) muss den Ball fangen und sich darum kümmern (die Exception behandeln) oder das Programm terminiert.



Fehlerbehandlung - Exceptions

- Wie Fehler bei der Programmabarbeitung behandeln?
  Immer Programm sofort beenden?
- Mechanismus in Java: Exceptions: "An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.“
- Ablauf:
  - In Methode oder Konstruktor x tritt Fehler während der Programmabarbeitung auf
    → x wirft eine Exception wie ein Ball.
  - Jemand (entweder x selbst (Fall 1) oder die x aufrufende Methode y (Fall 2)) muss den Ball fangen und sich darum kümmern (die Exception behandeln) oder das Programm terminiert.

Fehlerbehandlung - Exceptions



calls

calls

method 3
Error occurs
Cannot handle exception of type ExceptionXYZ

method 2
Cannot handle exception of type ExceptionXYZ

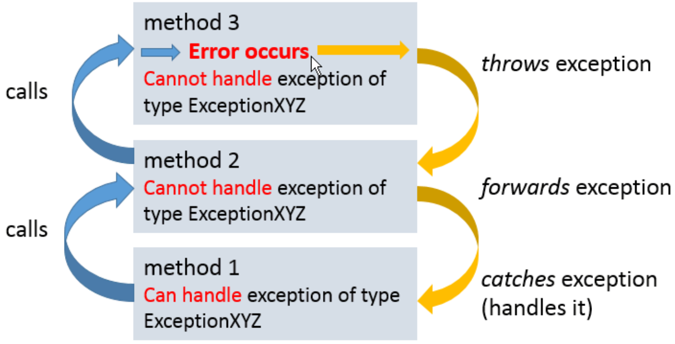
method 1
Can handle exception of type ExceptionXYZ

throws exception

forwards exception

catches exception (handles it)

Fehlerbehandlung - Exceptions



calls

calls

method 3
Error occurs
Cannot handle exception of type ExceptionXYZ

method 2
Cannot handle exception of type ExceptionXYZ

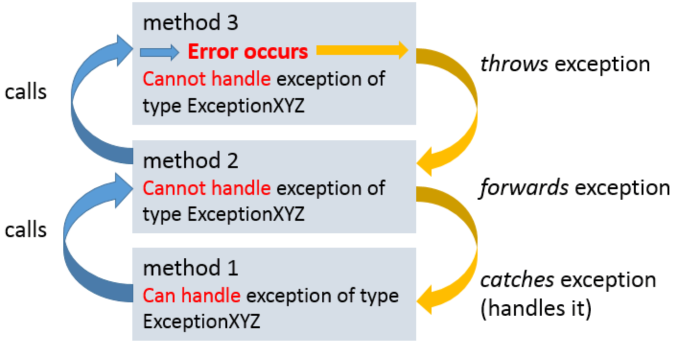
method 1
Can handle exception of type ExceptionXYZ

throws exception

forwards exception

catches exception (handles it)

Fehlerbehandlung - Exceptions



calls

calls

method 3
Error occurs
Cannot handle exception of type ExceptionXYZ

method 2
Cannot handle exception of type ExceptionXYZ

method 1
Can handle exception of type ExceptionXYZ

throws exception

forwards exception

catches exception (handles it)

Exceptions: `try` `catch`

- Fall 1: x behandelt Exception selbst:
  Probiere gefährlichen Code aus und fange ggf. auftretende Exceptions selbst.

```
try {
    // ...
    FileWriter fileWriter = new FileWriter("someFileName.txt");
    fileWriter.write('a');
    // ...

} catch (IOException e) {
    // Exception handling code goes here
    e.printStackTrace();

} catch (SomeOtherException e) {
    // Exception handling code goes here
    e.printStackTrace();

} catch (Exception e) {
 // Exception handling code goes here
    e.printStackTrace();
}
```

Exceptions: `try` `catch`

- **Fall 1**: x behandelt Exception selbst:
  Probiere gefährlichen Code aus und fange ggf. auftretende Exceptions selbst.

```
try {
    // ...
    FileWriter fileWriter = new FileWriter("someFileName.txt");
    fileWriter.write('a');
    // ...

} catch (IOException e) {
    // Exception handling code goes here
    e.printStackTrace();

} catch (SomeOtherException e) {
    // Exception handling code goes here
    e.printStackTrace();

} catch (Exception e) {
 // Exception handling code goes here
    e.printStackTrace();
}
```

Exceptions: try catch

Fall 1: x behandelt Exception selbst:
Probiere gefährlichen Code aus und fange ggf. auftretende Exceptions selbst.

try {
    // ...
    FileWriter fileWriter = new FileWriter("someFileName.txt");
    fileWriter.write('a');
    // ...

} catch (IOException e) {
    // Exception handling code goes here
    e.printStackTrace();

} catch (SomeOtherException e) {
    // Exception handling code goes here
    e.printStackTrace();

} catch (Exception e) {
 // Exception handling code goes here
    e.printStackTrace();
}

Exceptions: try catch

Fall 1: x behandelt Exception selbst:
Probiere gefährlichen Code aus und fange ggf. auftretende Exceptions selbst.

try {
    // ...
    FileWriter fileWriter = new FileWriter("someFileName.txt");
    fileWriter.write('a');
    // ...

} catch (IOException e) {
    // Exception handling code goes here
    e.printStackTrace();

} catch (SomeOtherException e) {
    // Exception handling code goes here
    e.printStackTrace();

} catch (Exception e) {
 // Exception handling code goes here
    e.printStackTrace();
}

Exceptions: try catch

Fall 1: x behandelt Exception selbst:
Probiere gefährlichen Code aus und fange ggf. auftretende Exceptions selbst.

try {
    // ...
    FileWriter fileWriter = new FileWriter("someFileName.txt");
    fileWriter.write('a');
    // ...

} catch (IOException e) {
    // Exception handling code goes here
    e.printStackTrace();

} catch (SomeOtherException e) {
    // Exception handling code goes here
    e.printStackTrace();

} catch (Exception e) {
 // Exception handling code goes here
    e.printStackTrace();
}

Exceptions: `finally`

- Fall 1: x behandelt Exception selbst:
  Probiere gefährlichen Code aus und fange ggf. auftretende Exceptions selbst.
  - `finally` – Block wird auf jeden Fall ausgeführt, egal ob Exception auftrat.

```
try {
    fileWriter = new FileWriter("someFileName.txt");
    fileWriter.write('a');
    // ...
} catch (Exception e) {
    // Exception handling code goes here
} finally {
    // Make sure that the file is closed, no matter whether
    // an exception occured or not.
    if (fileWriter != null) {
        fileWriter.close();
    }
}
```

Exceptions: `finally`

- Fall 1: x behandelt Exception selbst:
  Probiere gefährlichen Code aus und fange ggf. auftretende Exceptions selbst.
  - `finally` – Block wird auf jeden Fall ausgeführt, egal ob Exception auftrat.

```
try {
    fileWriter = new FileWriter("someFileName.txt");
    fileWriter.write('a');
    // ...
} catch (Exception e) {
    // Exception handling code goes here
} finally {
    // Make sure that the file is closed, no matter whether
    // an exception occured or not.
    if (fileWriter != null) {
        fileWriter.close();
    }
}
```

Exceptions: `throws`

- Fall 2: Jemand anders (aufrufende Methode y) soll Exception behandeln:
  → füge `throws` clause zu Methodendefinition hinzu:

```
class Bicycle {
    int gear;

    Bicycle(int initialGear) throws InvalidGearException {
        if (initialGear > 0) {
            gear = initialGear;
        } else {
            throw new InvalidGearException();
        }
    }

    void inflateTires() throws TireExplodedException {
        // ...
    }
    …
}
```

```
class InvalidGearException extends Exception {…}
```

```
class TireExplodedException extends Exception {…}
```

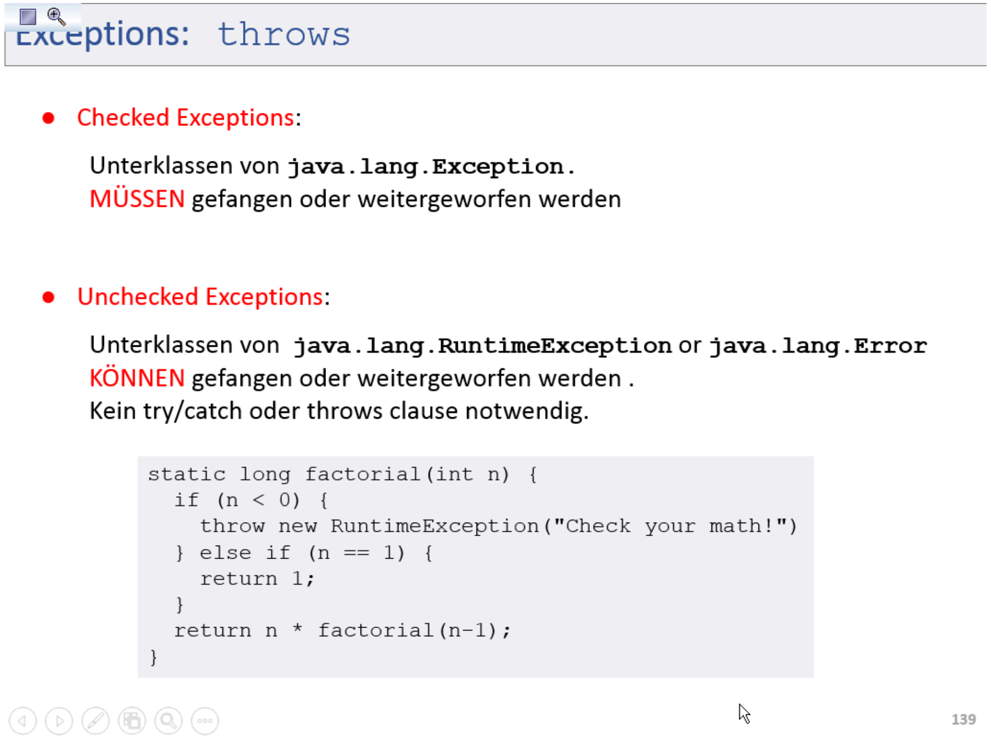
Exceptions: `throws`

- Fall 2: Jemand anders (aufrufende Methode y) soll Exception behandeln:
  → füge `throws` clause zu Methodendefinition hinzu:

```
class Bicycle {
    int gear;

    Bicycle(int initialGear) throws InvalidGearException {
        if (initialGear > 0) {
            gear = initialGear;
        } else {
            throw new InvalidGearException();
        }
    }

    void inflateTires() throws TireExplodedException {
        // ...
    }
    …
}
```

```
class InvalidGearException extends Exception {…}
```

```
class TireExplodedException extends Exception {…}
```



Exceptions: `throws`

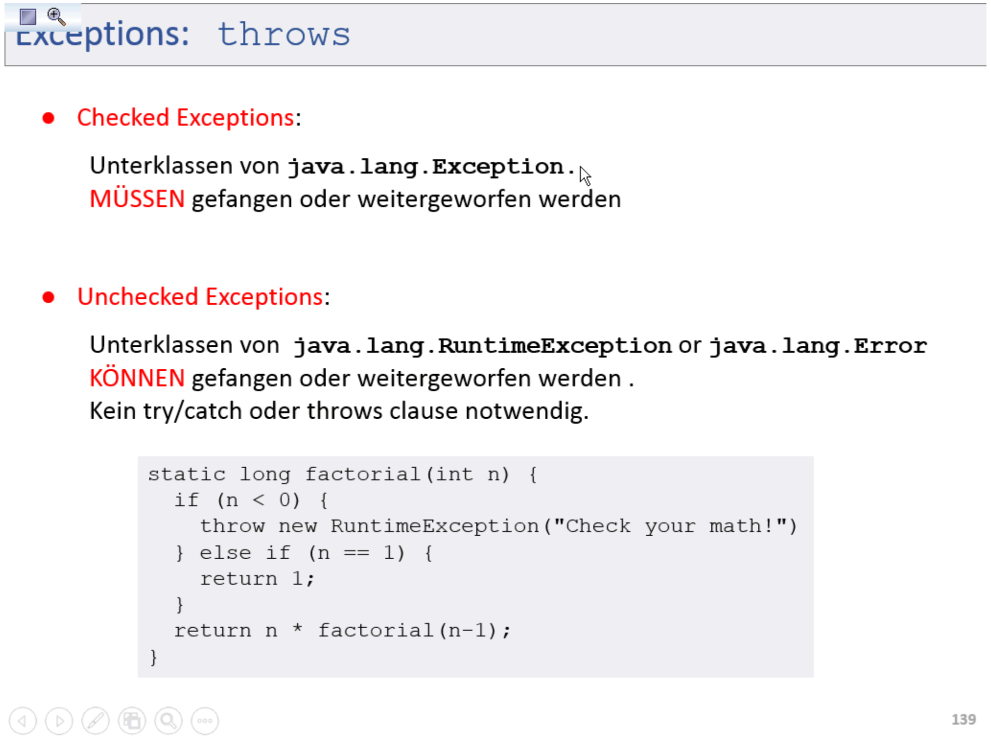
- Checked Exceptions:

  Unterklassen von **`java.lang.Exception`**.
  MÜSSEN gefangen oder weitergeworfen werden

- Unchecked Exceptions:

  Unterklassen von **`java.lang.RuntimeException`** or **`java.lang.Error`**
  KÖNNEN gefangen oder weitergeworfen werden .
  Kein try/catch oder throws clause notwendig.

```
static long factorial(int n) {
  if (n < 0) {
    throw new RuntimeException("Check your math!")
  } else if (n == 1) {
    return 1;
  }
  return n * factorial(n-1);
}
```



Exceptions: `throws`

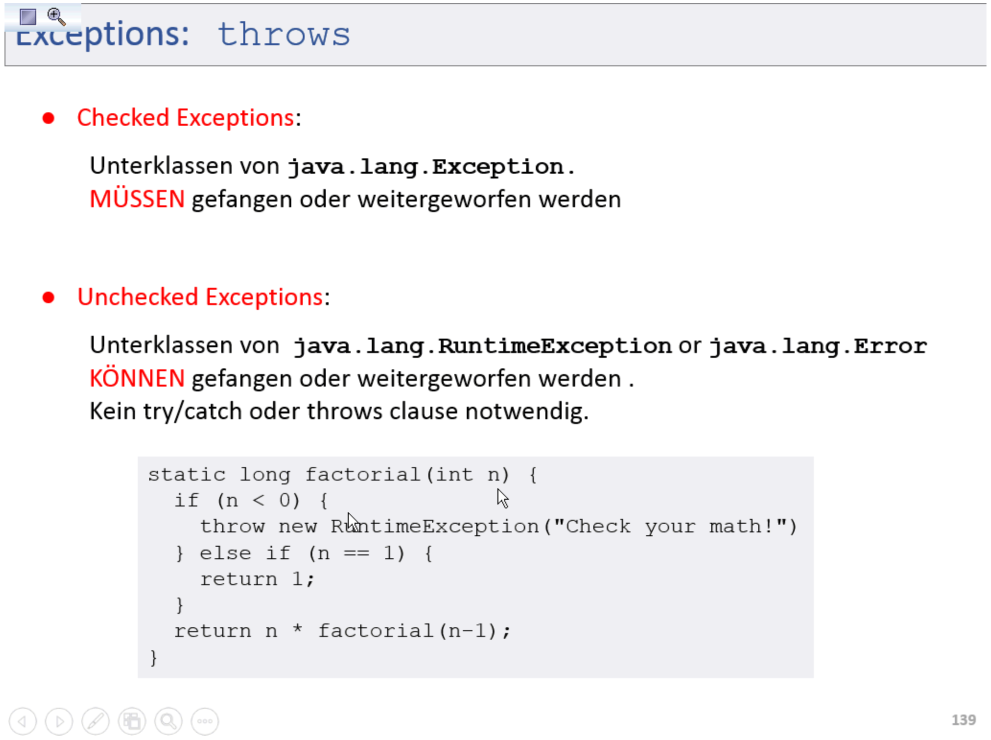
- Checked Exceptions:

  Unterklassen von **`java.lang.Exception`**.
  MÜSSEN gefangen oder weitergeworfen werden

- Unchecked Exceptions:

  Unterklassen von **`java.lang.RuntimeException`** or **`java.lang.Error`**
  KÖNNEN gefangen oder weitergeworfen werden .
  Kein try/catch oder throws clause notwendig.

```
static long factorial(int n) {
  if (n < 0) {
    throw new RuntimeException("Check your math!")
  } else if (n == 1) {
    return 1;
  }
  return n * factorial(n-1);
}
```



Exceptions: `throws`

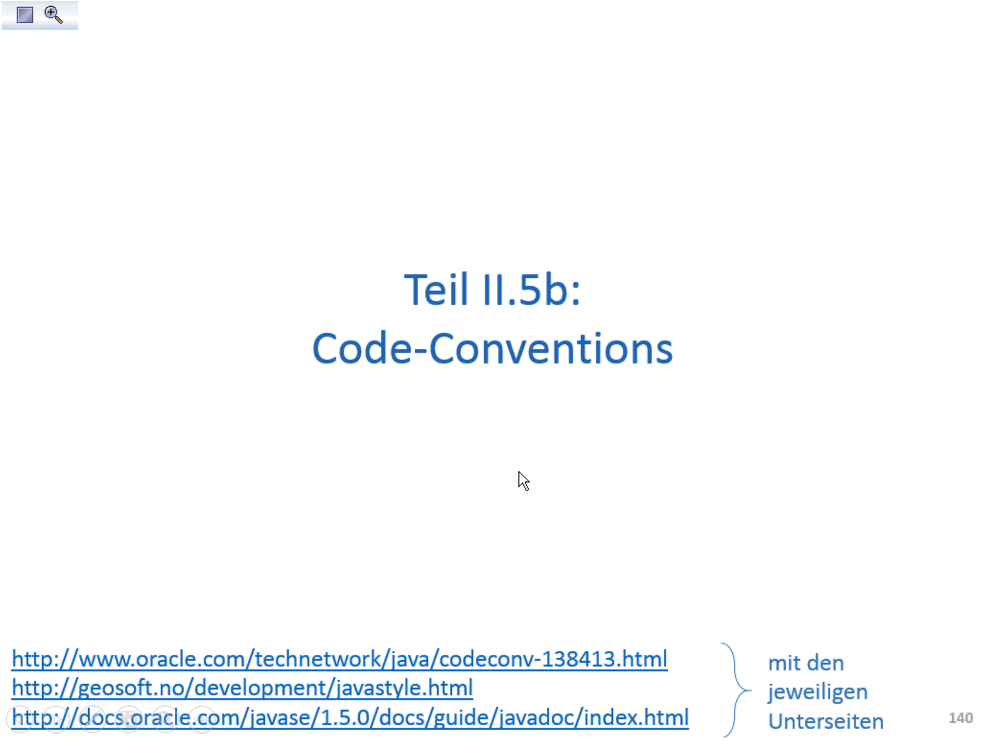
- Checked Exceptions:

  Unterklassen von **`java.lang.Exception`**.
  MÜSSEN gefangen oder weitergeworfen werden

- Unchecked Exceptions:

  Unterklassen von **`java.lang.RuntimeException`** or **`java.lang.Error`**
  KÖNNEN gefangen oder weitergeworfen werden .
  Kein try/catch oder throws clause notwendig.

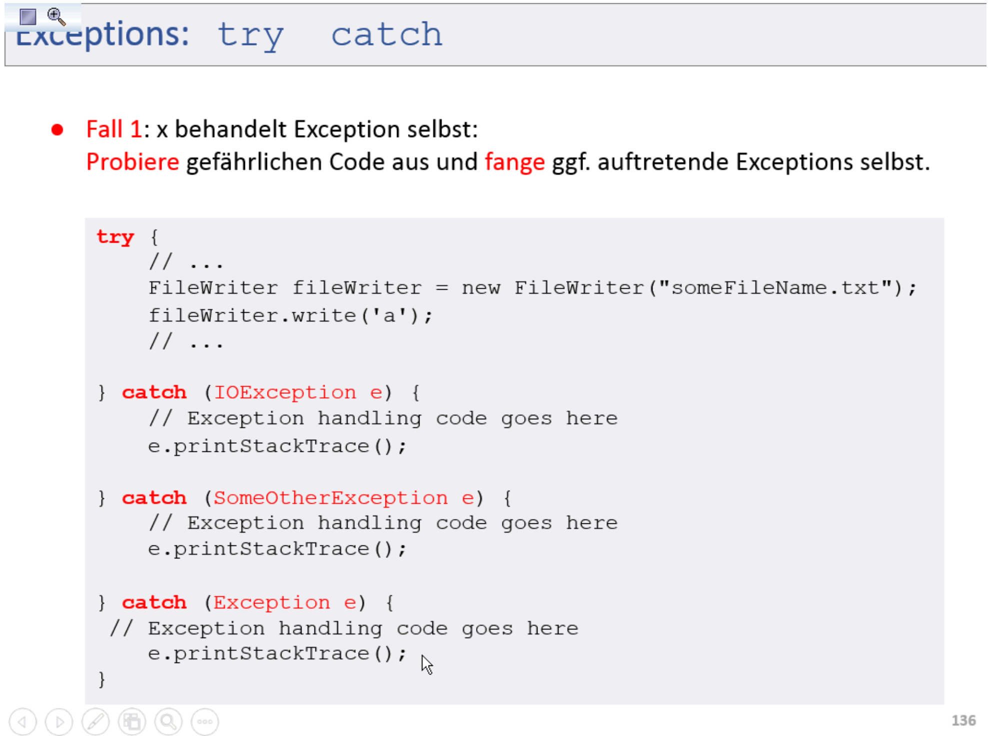
```
static long factorial(int n) {
  if (n < 0) {
    throw new RuntimeException("Check your math!")
  } else if (n == 1) {
    return 1;
  }
  return n * factorial(n-1);
}
```



Teil II.5b:
Code-Conventions

http://www.oracle.com/technetwork/java/codeconv-138413.html
http://geosoft.no/development/javastyle.html
http://docs.oracle.com/javase/1.5.0/docs/guide/javadoc/index.html

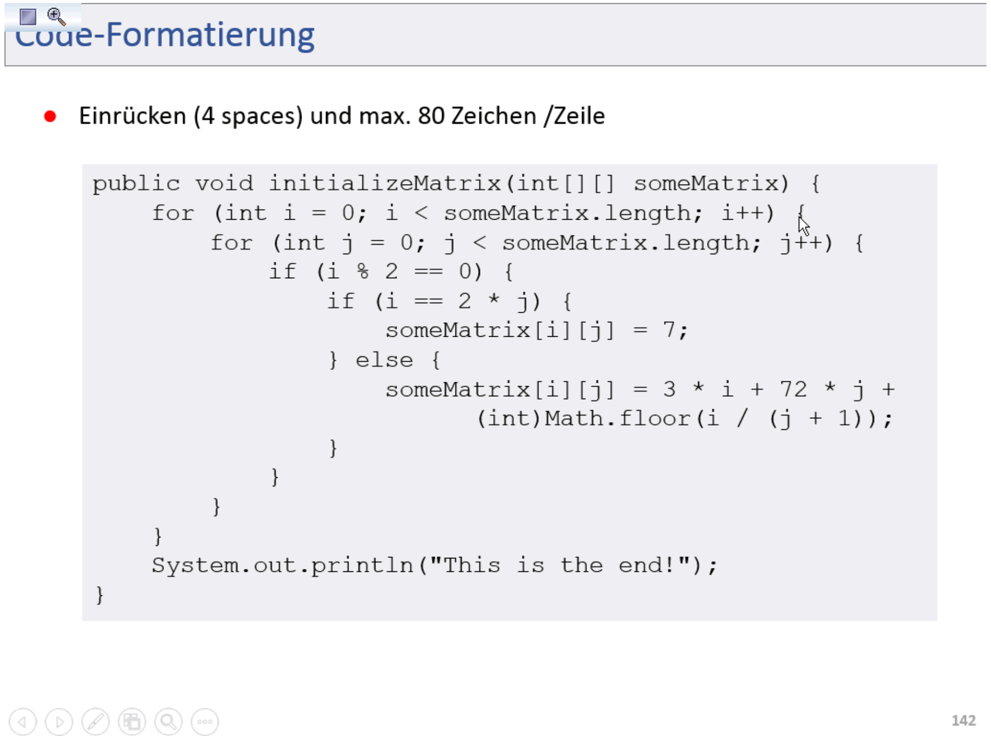
} mit den jeweiligen Unterseiten



Exceptions: `try` `catch`

- Fall 1: x behandelt Exception selbst:
  Probiere gefährlichen Code aus und fange ggf. auftretende Exceptions selbst.

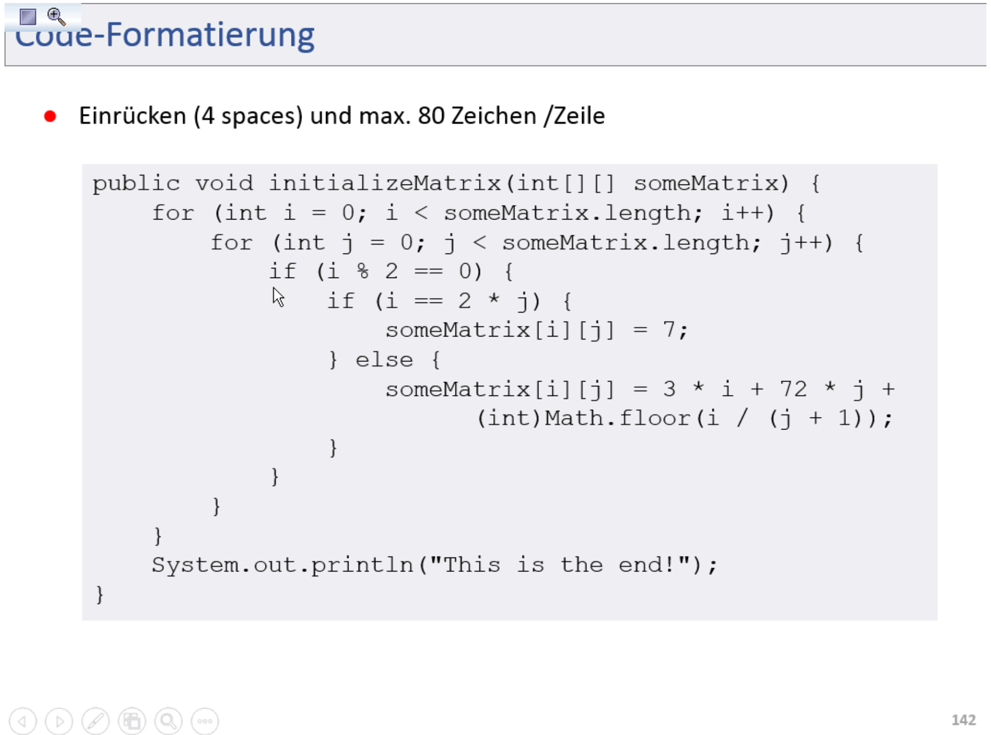
```
try {
    // ...
    FileWriter fileWriter = new FileWriter("someFileName.txt");
    fileWriter.write('a');
    // ...

} catch (IOException e) {
    // Exception handling code goes here
    e.printStackTrace();

} catch (SomeOtherException e) {
    // Exception handling code goes here
    e.printStackTrace();

} catch (Exception e) {
 // Exception handling code goes here
    e.printStackTrace();
}
```



Code-Formatierung

- Einrücken (4 spaces) und max. 80 Zeichen /Zeile

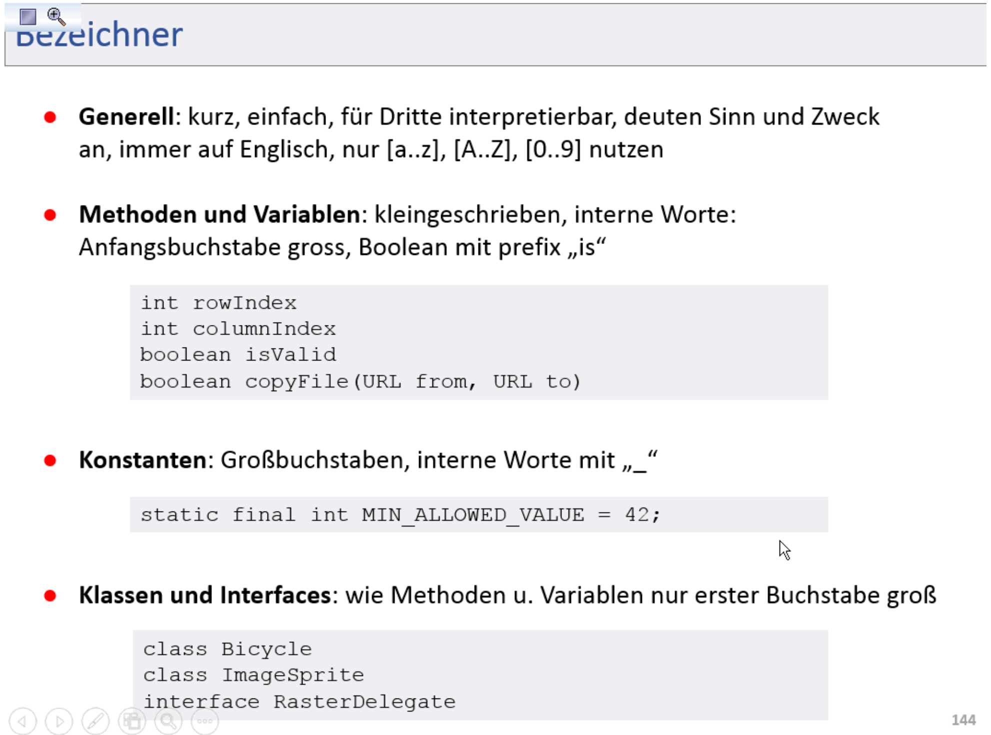
```
public void initializeMatrix(int[][] someMatrix) {
    for (int i = 0; i < someMatrix.length; i++) {
        for (int j = 0; j < someMatrix.length; j++) {
            if (i % 2 == 0) {
                if (i == 2 * j) {
                    someMatrix[i][j] = 7;
                } else {
                    someMatrix[i][j] = 3 * i + 72 * j +
                          (int)Math.floor(i / (j + 1));
                }
            }
        }
    }
    System.out.println("This is the end!");
}
```



Code-Formatierung

- Einrücken (4 spaces) und max. 80 Zeichen /Zeile

```
public void initializeMatrix(int[][] someMatrix) {
    for (int i = 0; i < someMatrix.length; i++) {
        for (int j = 0; j < someMatrix.length; j++) {
            if (i % 2 == 0) {
                if (i == 2 * j) {
                    someMatrix[i][j] = 7;
                } else {
                    someMatrix[i][j] = 3 * i + 72 * j +
                          (int)Math.floor(i / (j + 1));
                }
            }
        }
    }
    System.out.println("This is the end!");
}
```



Bezeichner

- **Generell**: kurz, einfach, für Dritte interpretierbar, deuten Sinn und Zweck an, immer auf Englisch, nur [a..z], [A..Z], [0..9] nutzen
- **Methoden und Variablen**: kleingeschrieben, interne Worte: Anfangsbuchstabe gross, Boolean mit prefix „is“

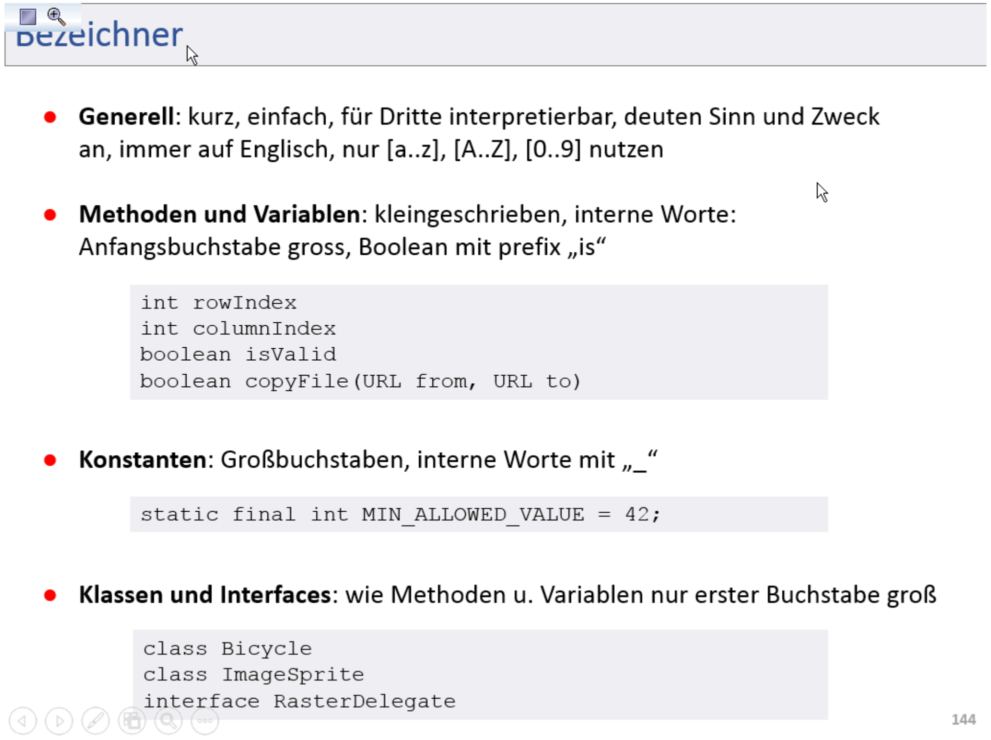
```
int rowIndex
int columnIndex
boolean isValid
boolean copyFile(URL from, URL to)
```

- **Konstanten**: Großbuchstaben, interne Worte mit „_“

```
static final int MIN_ALLOWED_VALUE = 42;
```

- **Klassen und Interfaces**: wie Methoden u. Variablen nur erster Buchstabe groß

```
class Bicycle
class ImageSprite
interface RasterDelegate
```



Bezeichner

- **Generell**: kurz, einfach, für Dritte interpretierbar, deuten Sinn und Zweck an, immer auf Englisch, nur [a..z], [A..Z], [0..9] nutzen
- **Methoden und Variablen**: kleingeschrieben, interne Worte: Anfangsbuchstabe gross, Boolean mit prefix „is"

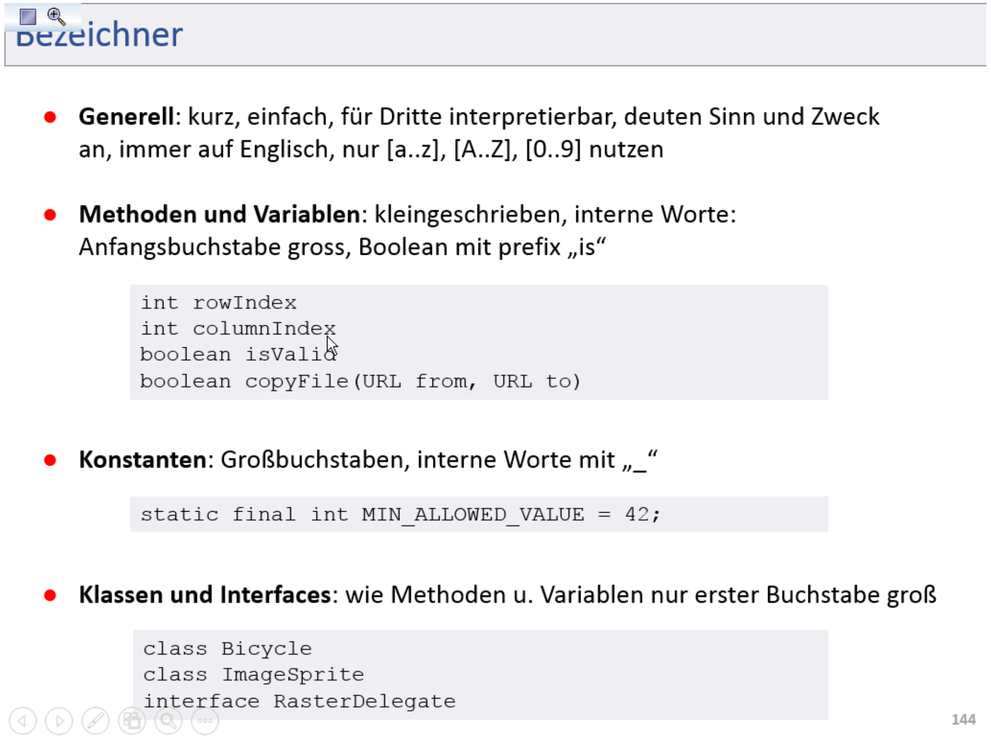
```
int rowIndex
int columnIndex
boolean isValid
boolean copyFile(URL from, URL to)
```

- **Konstanten**: Großbuchstaben, interne Worte mit „_"

```
static final int MIN_ALLOWED_VALUE = 42;
```

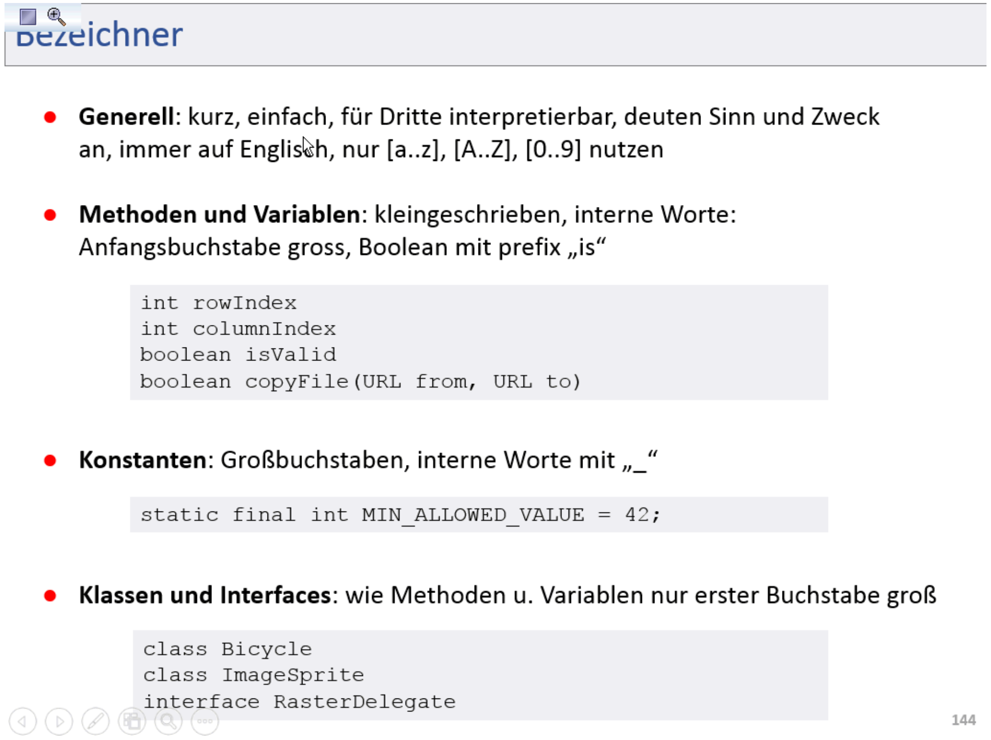
- **Klassen und Interfaces**: wie Methoden u. Variablen nur erster Buchstabe groß

```
class Bicycle
class ImageSprite
interface RasterDelegate
```



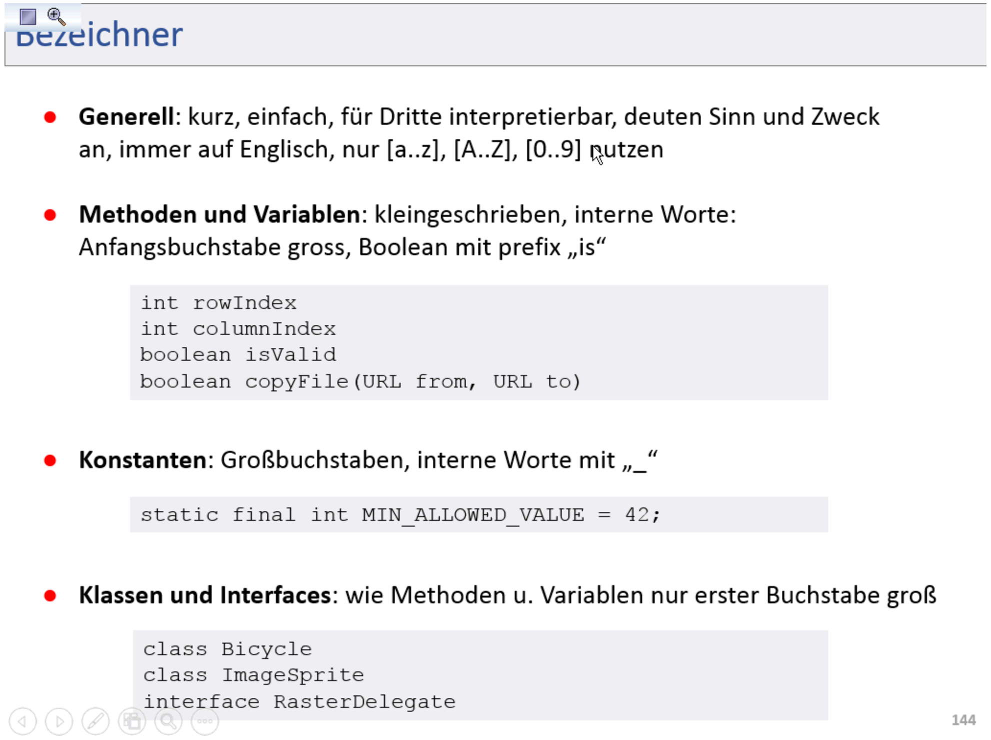
Bezeichner

Generell: kurz, einfach, für Dritte interpretierbar, deuten Sinn und Zweck an, immer auf Englisch, nur [a..z], [A..Z], [0..9] nutzen
Methoden und Variablen: kleingeschrieben, interne Worte: Anfangsbuchstabe gross, Boolean mit prefix „is"
int rowIndex
int columnIndex
boolean isValid
boolean copyFile(URL from, URL to)
Konstanten: Großbuchstaben, interne Worte mit „_"
static final int MIN_ALLOWED_VALUE = 42;
Klassen und Interfaces: wie Methoden u. Variablen nur erster Buchstabe groß
class Bicycle
class ImageSprite
interface RasterDelegate
144



Bezeichner

Generell: kurz, einfach, für Dritte interpretierbar, deuten Sinn und Zweck an, immer auf Englisch, nur [a..z], [A..Z], [0..9] nutzen
Methoden und Variablen: kleingeschrieben, interne Worte: Anfangsbuchstabe gross, Boolean mit prefix „is"
int rowIndex
int columnIndex
boolean isValid
boolean copyFile(URL from, URL to)
Konstanten: Großbuchstaben, interne Worte mit „_"
static final int MIN_ALLOWED_VALUE = 42;
Klassen und Interfaces: wie Methoden u. Variablen nur erster Buchstabe groß
class Bicycle
class ImageSprite
interface RasterDelegate
144



Bezeichner

Generell: kurz, einfach, für Dritte interpretierbar, deuten Sinn und Zweck an, immer auf Englisch, nur [a..z], [A..Z], [0..9] nutzen
Methoden und Variablen: kleingeschrieben, interne Worte: Anfangsbuchstabe gross, Boolean mit prefix „is"
int rowIndex
int columnIndex
boolean isValid
boolean copyFile(URL from, URL to)
Konstanten: Großbuchstaben, interne Worte mit „_"
static final int MIN_ALLOWED_VALUE = 42;
Klassen und Interfaces: wie Methoden u. Variablen nur erster Buchstabe groß
class Bicycle
class ImageSprite
interface RasterDelegate
144

Kommentare und Dokumentation

- Häufig und sorgfältig Kommentare und JavaDoc benutzen!

```
//Recursive version of faculty

/**
 * Computes faculty f(n)=n!.
 * @param n: Argument of faculty
 * @return n!
 */
  public long faculty(int n)
  {
    if (n<0)
        return -1;  //-1 as error code; fix me!
    else
       if (n==0)
            return 1; //0!==1;
       else
            return n*faculty(n-1); //recursive call
  }
```

Kommentare und Dokumentation

- Häufig und sorgfältig Kommentare und JavaDoc benutzen!

```
//Recursive version of faculty

/**
 * Computes faculty f(n)=n!.
 * @param n: Argument of faculty
 * @return n!
 */
  public long faculty(int n)
  {
    if (n<0)
        return -1;  //-1 as error code; fix me!
    else
       if (n==0)
            return 1; //0!==1;
       else
            return n*faculty(n-1); //recursive call
  }
```

Kommentare und Dokumentation

- Häufig und sorgfältig Kommentare und JavaDoc benutzen!

```
//Recursive version of faculty

/**
 * Computes faculty f(n)=n!.
 * @param n: Argument of faculty
 * @return n!
 */
  public long faculty(int n)
  {
    if (n<0)
        return -1;  //-1 as error code; fix me!
    else
       if (n==0)
            return 1; //0!==1;
       else
            return n*faculty(n-1); //recursive call
  }
```

Teil II.5c:
Java Class Library, Generics

http://docs.oracle.com/javase/tutorial/java/data/index.html
http://docs.oracle.com/javase/tutorial/java/generics/index.html

} mit den jeweiligen Unterseiten