Programmierkurs Java

UE 22 - Dokumentation

Dr.-Ing. Dietrich Boles

Gliederung

- Programmdokumentation
- Dokumentation der Implementierung
- Java-Bezeichnerkonventionen
- Dokumentation des Protokolls / javadoc
- Beispiel ADT-Klasse Cardinal
- Zusammenfassung

Programmdokumentation

- 2 Typen von Sourcecode-Dokumentation:
  (1) Dokumentation des Protokolls
  (2) Dokumentation der Implementierung

- Motivation für (1):
  - Beschreibung der Funktionalität einer Klasse / Methode für andere Programmierer
- Motivation für (2):
  - bessere Verständlichkeit der gewählten Algorithmen
  - einfachere Fehlersuche und Fehlerbeseitigung
  - einfachere Wartung, Änderbarkeit und Erweiterbarkeit des Codes

Dokumentation der Implementierung (1)

- Dokumentation der Implementierung in Java:
  - aussagekräftige Bezeichner (Klassen / Methoden / Variablen) verwenden
  - den Code gut strukturieren
  - an "kritischen" Stellen Zeilenkommentare einbauen

  → Java-Code-Konventionen!

  Schlechtes Beispiel:

```
public class M {
public static int s(int x) {
int r = 0;
for(int i=1; i<=x; i++) { r = r + i; }
return r; } }
```

Gutes Beispiel:

```
public class Mathematik {
    public static int summe(int bis) {
        int ergebnis = 0;

        // Zahlen von 1 bis "bis" addieren
        for(int i = 1; i <= bis; i++) {
            ergebnis = ergebnis + i;
        }
        return ergebnis;
    }

    public static int fakultaet(int n) {
        ...
    }
}
```

Java-Bezeichnerkonventionen

- Klassen:
  - Anfangsbuchstaben jedes Wortteils groß
    - **`DibosSchachProgramm`**
    - **`ArrayIndexOutOfBoundsException`**
- Attribute / Variablen:
  - Beginn mit Kleinbuchstaben; Anfangsbuchstaben weiterer Wortteile groß
    - **`anzahlAnZeichen`**
    - **`maximaleGroesse`**
- Methoden:
  - Beginn mit Kleinbuchstaben; Anfangsbuchstaben weiterer Wortteile groß
    - **`getNumber`**
    - **`liefereNaechstenSpielzug`**
- Konstanten:
  - ausschließlich Großbuchstaben; Trennung von Wortteilen durch Unterstrich (_)
    - **`MAX_VALUE`**
    - **`INVALID_PARAM`**

Dokumentation des Protokolls / javadoc

- **`javadoc`**: Werkzeug des JDK zur Generierung von HTML-Code zur Dokumentation des Protokolls einer oder mehrerer Klassen:
- Aufruf: **`javadoc <.java-Datei(en)>`**
- Voraussetzung: Verwendung von **javadoc-Kommentaren** und **-Schlüsselwörtern**
- javadoc-Kommentare: **`/** <zeichen> */`**
- javadoc-Schlüsselwörter:
  - Beschreibung von Klassen:

```
Beschreibung der Funktionalität
@version <text>
@author <text>
```

  - Beschreibung von Methoden:

```
Beschreibung der Funktionalität
Vorbedingungen, Nachbedingungen
@param <name> <beschreibung>
@return <beschreibung>
@throws <klassenname> <beschreibung>
```

Beispiel ADT-Klasse Cardinal (1)

```
package number;


/**
 * ADT-Klasse zur Repraesentation Natuerlicher Zahlen als Objekte.
 *
 * @author Dietrich Boles
 * @version 3.0, 30.04.2020
 */
public class Cardinal {

   /**
    * speichert den eigentlichen Wert
    */
   protected int wert;
```

Beispiel ADT-Klasse Cardinal (2)

```
/**
 * Default-Konstruktor; initialisiert mit dem Wert 0
 */
public Cardinal() {
     this.wert = 0;
}

/**
 * Konstruktor; initialisiert mit dem uebergebenen Wert
 *
 * @param w Initial-Wert
 * @throws NegativerWertException wird geworfen, falls w &lt; 0
 */
public Cardinal(int w) throws NegativerWertException {
     if (w < 0)
             throw new NegativerWertException(w);
     this.wert = w;
}
```

Beispiel ADT-Klasse Cardinal (3)

```
/**
 * Copy-Konstruktor; initialisiert mit einem existierenden Objekt
 *
 * @param obj existierendes Cardinal-Objekt (!= null)
 */
public Cardinal(Cardinal obj) {
     this.wert = obj.wert;
}

/**
 * Klonieren eines Cardinal-Objektes
 *
 * @return kloniertes Cardinal-Objekt
 */
public Cardinal clone() {
     return new Cardinal(this);
}
```

Beispiel ADT-Klasse Cardinal (4)

```
 /**
   * liefert Hash-Code
   * @return einen generierten Hash-Code
   */
  @Override
  public int hashCode() {  return 31 + this.wert;  }

  /**
   * Vergleich zweier Cardinal-Objekte
   *
   * @param obj  das zu vergleichende Objekt
   * @return liefert genau dann true, wenn die beiden Objekte dieselbe
   *         Natuerliche Zahl repraesentieren
   */
  @Override
  public boolean equals(Object obj) {
    if (this == obj) { return true;  }
    if (obj == null) { return false; }
    if (this.getClass() != obj.getClass()) { return false; }
    return this.wert == ((Cardinal)obj).wert;
}
```

Beispiel ADT-Klasse Cardinal (5)

```
/**
 * String-Konvertierung
 *
 * @return Wert des Cardinal-Objektes als String-Repraesentation
 */
public String toString() {
     return "" + this.wert;
}

/**
 * Aufaddieren eines anderen Cardinal-Objektes
 *
 * @param obj anderes Cardinal-Objekt (!= null)
 */
public void add(Cardinal obj) {
     this.wert = Cardinal.add(this, obj).wert;
}
```

Beispiel ADT-Klasse Cardinal (6)

```
/**
 * Addieren zweier Cardinal-Objekte
 *
 * @param obj1 erstes Cardinal-Objekt (!= null)
 * @param obj2 zweites Cardinal-Objekt (!= null)
 * @return Summe der beiden Parameter-Objekte
 */
public static Cardinal add(Cardinal obj1, Cardinal obj2) {
     try {
             if (obj1.wert + obj2.wert >= 0) {
                     return new Cardinal(obj1.wert + obj2.wert);
             } else {  // Summe > Integer.MAX_VALUE
                     return new Cardinal(obj1.wert + obj2.wert -
                                         Integer.MIN_VALUE);
             }
     } catch (NegativerWertException exc) {
             // kann nicht sein
             return null;
     }
}
```

Beispiel ADT-Klasse Cardinal (7)

```
/**
 * Dividieren zweier Cardinal-Objekte
 *
 * @param divident Divident als Cardinal-Objekt (!= null)
 * @param divisor Divisor als Cardinal-Objekt (!= null)
 * @return Quotient der beiden Parameter-Objekte
 * @throws DivNullException wird geworfen, falls divisor den
 *                          Wert 0 repraesentiert
 */
public static Cardinal div(Cardinal divident, Cardinal divisor)
            throws DivNullException {
    if (divisor.wert == 0)
            throw new DivNullException();
    try {
            return new Cardinal(divident.wert / divisor.wert);
    } catch (NegativerWertException exc) {
            // kann nicht sein
            return null;
    }
}
```

Beispiel ADT-Klasse Cardinal (8)

```
    // Testprogramm
    public static void main(String[] args) {
        try {
                int eingabe1 = IO.readInt("Natuerliche Zahl: ");
                int eingabe2 = IO.readInt("Natuerliche Zahl: ");
                Cardinal nat1 = new Cardinal(eingabe1);
                Cardinal nat2 = new Cardinal(eingabe2);
                Cardinal nat3 = Cardinal.div(nat1, nat2);
                IO.println("Divisor: " + nat2);
                IO.println("Divident: " + nat1);
                IO.println("Quotient = " + nat3);
        } catch (NegativerWertException exc) {
                IO.println("negativer Wert: " + exc.getWert());
        } catch (DivNullException exc) {
                IO.println("Division durch Null!");
        }
    }
}
```

Beispiel ADT-Klasse Cardinal (9)

```
package number;

/**
 * Fehlerklasse fuer die Division durch 0
 *
 * @version 3.0, 30.04.2020
 * @author Dietrich Boles
 */
public class DivNullException extends Exception {
}
```

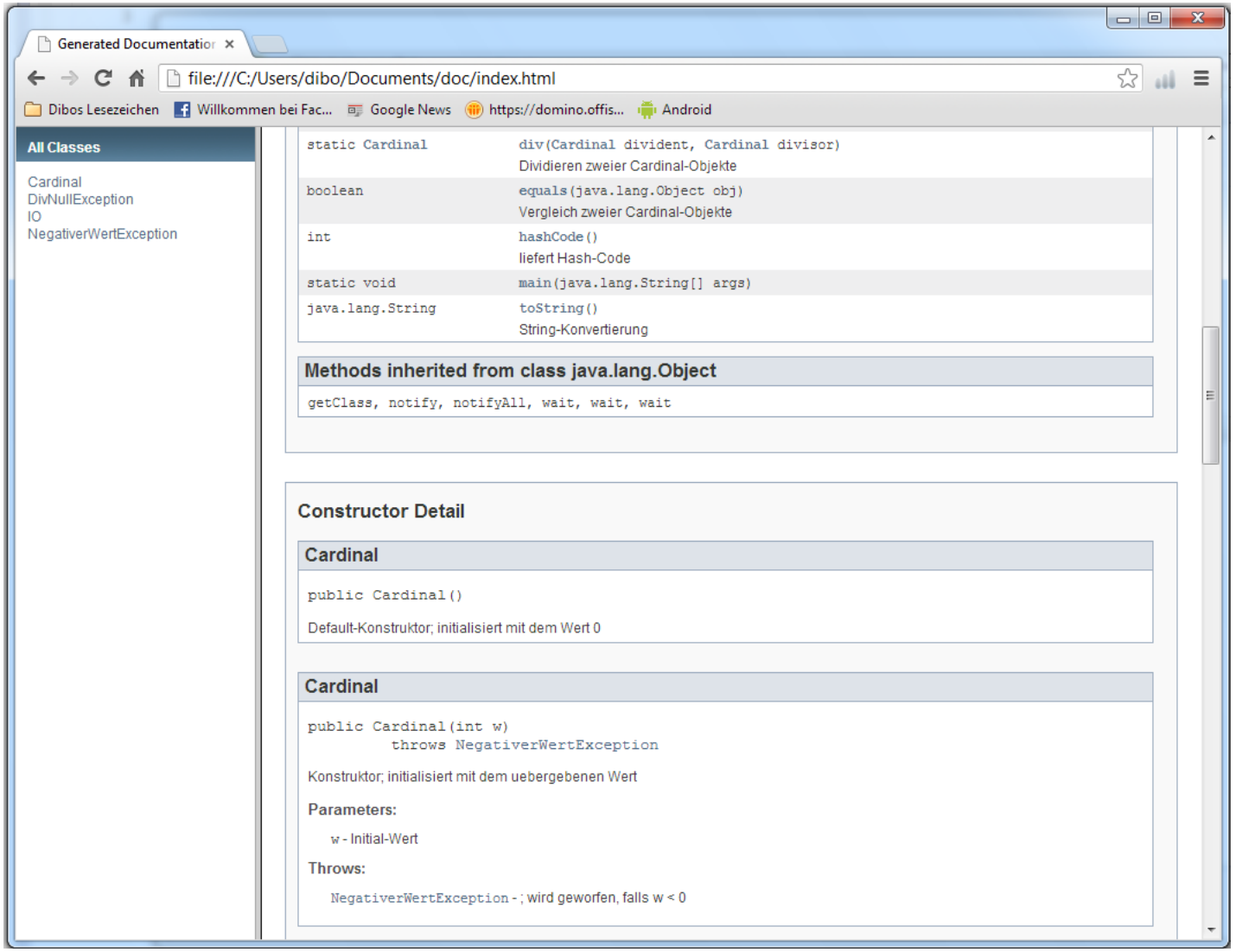
Beispiel ADT-Klasse Cardinal (10)

```
package number;

/**
 * Fehlerklasse fuer negative Zahlenwerte
 *
 * @version 3.0, 30.04.2020
 * @author Dietrich Boles
 */
public class NegativerWertException extends Exception {

   /**
    * der fehlerhafte Wert
    */
   protected int wert;
```

Beispiel ADT-Klasse Cardinal (11)

```
    /**
     * Konstruktor; initialisiert mit uebergebenem Wert
     *
     * @param wert
     *             Initial-Wert
     */
    public NegativerWertException(int wert) {
         this.wert = wert;
    }

    /**
     * get-Methode zum Liefern des (negativen) Wertes
     *
     * @return aktueller Wert des Attributes "wert"
     */
    public int getWert() {
         return this.wert;
    }
}
```

Beispiel ADT-Klasse Cardinal (12)



Zusammenfassung

- Typen von Sourcecode-Dokumentation:
  - Dokumentation des Protokolls (→ javadoc)
  - Dokumentation der Implementierung

- Sinn und Zweck:
  - Nutzung von Klassen
  - einfachere Fehlersuche und Fehlerbeseitigung
  - einfachere Wartung, Änderbarkeit und Erweiterbarkeit des Codes