Wiederholung: JavaDoc, JML Objekterzeugung

Programmierung und Softwareentwicklung
Hörsaalübung

JavaDoc und JML

Wiederholung und Vertiefung

JavaDoc – Aufbau

Aufbau eines JavaDoc Kommentars

- Kurzzusammenfassung
  - Beschreibt in einem kurzen Satz, was die Operation macht
- Langbeschreibung
  - Beschreibt im Detail in einem größeren Block, was die Operation macht
- Pre- und Postconditions

JavaDoc – Aufbau

Aufbau eines JavaDoc Kommentars

- Parameter
  - Mit `@param` Tag Beschreibung des Namens und der Funktion des Parameters
  - 0 bis * viele
- Return
  - Mit `@return` Tag Beschreibung der Rückgabe
- Throws
  - Mit `@throws` Tag Beschreibung der jeweiligen Exceptions und Grund
  - 0 bis * viele

Aufgabe 1

Erinnern Sie sich an das PSE Übungsblatt 4, Aufgabe 1.

Sie mussten Objekte für ein Fußballspiel identifizieren und anschließend die Schnittstelle der Klasse `FootballPlayer` definieren.

Beachten Sie nun die Abfrage `Boolean hasBall()` und das Kommando `void pass(Player other)` und beschreiben Sie diese möglichst genau mit einem JavaDoc und einem JML Kommentar.

Achten Sie darauf die Schnittstelle möglichst vollständig zu dokumentieren.

Aufgabe 1 – Lösung (I)

```java
/*@
 @ ensures \result == (this.ball != null);
 */
/**
 * Checks if player has the ball.
 *
 * Takes a look at the ball object
 * and checks if the ball is null or set.
 * If the player has the ball, the return value
 * will be true, else false.
 *
 * @return true if player has ball, else false
 */
public /*@ pure @*/ Boolean hasBall() {
    // ...
}
```

Aufgabe 1 – Lösung (II)

```
/*@
 @ requires hasBall();
 @ ensures !hasBall() && other.hasBall();
 */
/**
 * Passes the ball to another player.
 *
 * Passes the ball to a given other player,
 * only if the player has the ball and
 * the other players location is near to the player.
 * The player has no ball afterwards.
 *
 * @param other - the other player object to pass the ball
 * @throws IllegalArgumentException - other Player is not near to player
 */
public void pass(/*@ non_null @*/ Player other) {
    // ...
}
```

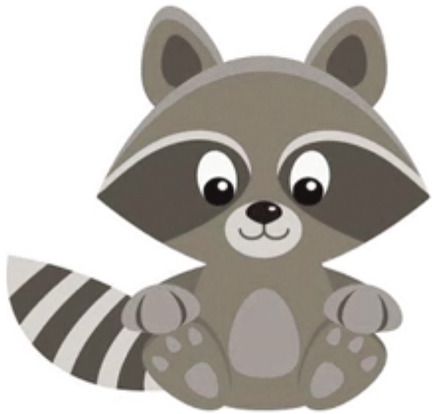
Objekterzeugung

Objekterzeugung

Objekte können mit dem Schlüsselwort `new` erzeugt werden

- `Klassenname objektname = new Konstruktoraufruf;`

```
// Creation of a new Raccoon object
Raccoon sweetRaccoon = new Raccoon();
```



Konstruktor

Spezielle Operation, die bei Objekterzeugung aufgerufen wird

- Steht bei Objekterzeugung nach `new` Schlüsselwort
- Muss nicht explizit definiert werden

```
// Creation of a new Raccoon object
Raccoon sweetRaccoon = new Raccoon();
```

Konstruktoren und Destruktoren

Was geschieht bei dem Aufruf

```
Student s = new Student() ?
```

Jede Klasse besitzt einen Konstruktor:

- Aufbau wie Operation
- Kein Rückgabetyp wird angegeben
- Konstruktorname entspricht Klassennamen
- Konstruktoren können nicht direkt aufgerufen werden
- Compiler erzeugt Defaultkonstruktor, wenn für die Klasse kein Konstruktor existiert

Konstruktor

Spezielle Operation, die bei Objekterzeugung aufgerufen wird

- Steht bei Objekterzeugung nach `new` Schlüsselwort
- Muss nicht explizit definiert werden
- Kann folgendermaßen definiert werden

```
public Raccoon() {
    // ...
}
```

Objekterzeugung

Waschbär
Fellfarbe Lieblingsessen Größe
laufen essen schlafen

Festlegen der Attributswerte

- Möglich nach Objekterzeugung über Setter

```
// Creation of a new Raccoon object
Raccoon sweetRaccoon = new Raccoon();
// Set attribute values of sweetRaccoon
sweetRaccoon.setColor("Grey");
sweetRaccoon.setFavoriteFood("Jiaozi");
sweetRaccoon.setHeight(60.0);
```

- Welche Werte haben die Attribute vor Aufruf des Setters?
  - Standardwerte: Für die String Attribute sind die Werte `null`, für die Größe 0.0
  - Ist das sinnvoll?
  - Wie kann das verhindert werden?

Konstruktor

Operation, die bei Objekterzeugung aufgerufen wird

- Kann Werte für Attribute direkt bei Objekterzeugung festlegen
  - Objekteattribute können keine ungewollten Standardwerte besitzen
- Kann folgendermaßen definiert werden

```
private String color;
private String favoriteFood;
private double height;
public Raccoon(String color, String favoriteFood, double height) {
    this.color = color;
    this.favoriteFood = favoriteFood;
    this.height = height;
}
```

Objekterzeugung und Konstruktor

```
private String color;
private String favoriteFood;
private double height;
public Raccoon(String color, String favoriteFood, double height) {
    this.color = color;
    this.favoriteFood = favoriteFood;
    this.height = height;
}
```

Konstruktor benötigt dann bei Objekterzeugung die erforderlichen Argumente

```
// Creates a new Raccoon object with given attribute values
Raccoon sweetRaccoon = new Raccoon("Grey", "Jiaozi", 60.0);
```

Beispiel zu Konstruktoren

```
public class Student {
    private String name;
    private int matrikelnummer;
}
```

```
Student s = new Student();
```

Aufruf von Defaultkonstruktor,
Name und Matrikelnummer sind nicht initialisiert!

Beispiel zu Konstruktoren

```
public class Student {
    private String name;
    private int matrikelnummer;
    public Student() {
        name = new String();
        matrikelnummer = -1;
    }
}
```

```
Student s = new Student();
```

Name und Matrikelnummer werden mit Defaultwerten initialisiert!

Beispiel zu Konstruktoren

```
public class Student {
    private String name;
    private int matrikelnummer;
    public Student(String name) {
        this.name = name;
        matrikelnummer = -1;
    }
}
```

```
Student s = new Student();
```

Konstruktor existiert nicht, zur Erzeugung einer Instanz der Klasse Student wird ein Name benötigt.

Konstruktoren und Destruktoren

Wie wird ein erzeugtes Objekt wieder zerstört?

- Java besitzt keinen Destruktor!

Wie stellt Java dann sicher, dass der Speicher nicht durch unbenötigte Objekte überläuft?

- In Java gibt es einen automatischen Garbage Collector:
  - GC ist ein Hintergrundprozess
  - GC kann nicht vom Nutzer gesteuert werden
  - GC eliminiert alle Objekte, auf die keine Referenz mehr existiert

Sichtbarkeit von Konstruktoren

Was bedeutet es, wenn ein Konstruktor als private deklariert wird?

- private-Konstruktoren sind außerhalb der Klasse nicht sichtbar!

Überlegen Sie, wie man einen solchen Konstruktor nutzen kann!

- Instanzen dieser Klassen müssen innerhalb der Klasse gebildet werden
  - Siehe statische (Fabrik-)Operationen

Vorteile von diesen sogenannten Fabrikoperationen:

- Volle Kontrolle über Objekterzeugung einer Klasse
- Dynamische Auswahl der konkreten Klasse zur Laufzeit (Frameworks)

Pakete

Vergleiche HU1 Folien

Pakete (I)

Was sind Pakete?

- Pakete helfen ein Projekt zu strukturieren
- Sammlungen von Klassen unter einem bestimmten Namen
- Styleguide: Üblicherweise als umgedrehte Domänennamen
  - rss.uni-stuttgart.de → de.unistuttgart.rss
- Stehen an erster Stelle in einer Java Datei

```java
package de.unistuttgart.pse.hu;

public class Raccoon {
    // Attributes

    // Operations
}
```

Pakete (II)

Import von Paketen

- Datentypen müssen bekannt sein, um genutzt werden zu können
- Durch Imports von Paketen werden Datentypen in die Java Datei „eingebunden“
- Imports stehen zwischen Paketnamen und Deklaration der Java Klasse

```
import de.unistuttgart.pse.hu.Raccoon;
// Other imports

public class Zoo {
    // Some implementation...
    // Inside an operation:
        Raccoon raccoon = new Raccoon();
    //...
}
```