252-0027

Einführung in die Programmierung

Übungen

Reference Passing & Rekursion

Henrik Pätzold
Departement Informatik
ETH Zürich

Heutiger Plan

- Theorie
  - Reference-Passing
  - Vergleiche von Objekten
  - Basics zu Rekursion
- Praxis
  - Coding mit Rekursion

Slides mit Kommentaren zu Code-Beispielen ab nun auch immer online :)

Was ist eine Objektreferenz

Objekte 101 – (Deep Dive in einer späteren Übung)

- Klassen sind „Baupläne“, die definieren, wie Daten gespeichert werden
- Objekte sind konkrete Realisierungen der Klasse
- Wie speichern wir Objekte? (***DEMO***)

```java
public class Main {
    public static void main(String[] args){
        Coordinate coord = new Coordinate(2,3);
        System.out.println(coord);
    }
}
public class Coordinate {
    int x;
    int y;
    public Coordinate(int xcoord, int ycoord){
        x = xcoord;
        y = ycoord;
    }
}
```

Primitive D.t.

- speichern tatsächlichen Wert
- sind von der Größe begrenzt
- Standardwerte sind festgelegt
- Können niemals ***NULL*** sein

Objekte

- speichern Referenz auf Speicherort
- Größe ist variabel
- Standardwert ist ***NULL*** (keine Referenz in den Speicher)

Weitergabe von Referenzen

- ***numbers*** speichert die Referenz, wo die tatsächlichen Werte liegen
- gib Referenz an ***modifyArray*** weiter
- ***modifyArray*** sucht Werte im Speicher, verdoppelt sie, terminiert
- ***main*** sucht mit derselben im Speicher und findet verdoppelte Werte **ohne**, dass wir sie zurückgeben mussten

```java
public class ReferenceArrayDemo {
    public static void main(String[] args) {
        // Erstelle ein Array
        int[] numbers = {1, 2, 3, 4, 5};
        System.out.println("Vor der Änderung:");
        System.out.println(Arrays.toString(numbers));
        modifyArray(numbers); // Übergabe des Arrays
        System.out.println("Nach der Änderung:");
        System.out.println(Arrays.toString(numbers));
    }
    public static void modifyArray(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            arr[i] *= 2; // Verdopple jeden Wert im Array
        }
    }
```

Wir wollen das verstehen, um simple Fehler während der Prüfung zu vermeiden.
(Handout – WarmUp.java)

Zusammenfassung Demo 1

- ***temp = KontoStandProTag*** gibt ***temp*** den selben Pointer wie ***KontoStandProTag***
- damit wird jede Veränderung in ***temp*** auch in ***KontoStandProTag*** durchgeführt
- Wir verändern also durch das Reference-Passing hier das globale Array, was offensichtlich nicht gewollt ist.

Ausnahmen - Unveränderlichkeit von Strings

- Java erzwingt ein „Pass-By-Value“-Verhalten für einzelne Ausnahmen
- Strings sind Objekte
- diese Art von Fehlern kann mit Strings trotzdem nicht passieren
- bei Weitergabe wird Kopie mit selbem Wert erstellt (mit neuer Referenz)

```java
public class StringDemo {
    public static void main(String[] args) {
        String a = "Hello";
        System.out.println(a); // Hello
        modifyString(a);
        System.out.println(a); // Hello
    }
    public static void modifyString(String a) {
        a = a+" World!";
        System.out.println(a); // Hello World!
    }
}
```

Ausnahmen - Unveränderlichkeit von Wrappern

- Wrapper-Klassen sind nützliche Klassen, die primitive Datentypen in Form von Objekten repräsentieren
- ***boolean*** -> ***Boolean***, ***int*** -> ***Integer***, ***double*** -> ***Double*** etc.
- stellen zusätzliche Methoden zur Manipulation dieser Werte bereitstellen.
- bei Weitergabe wird Kopie mit selbem Wert erstellt (mit neuer Referenz)

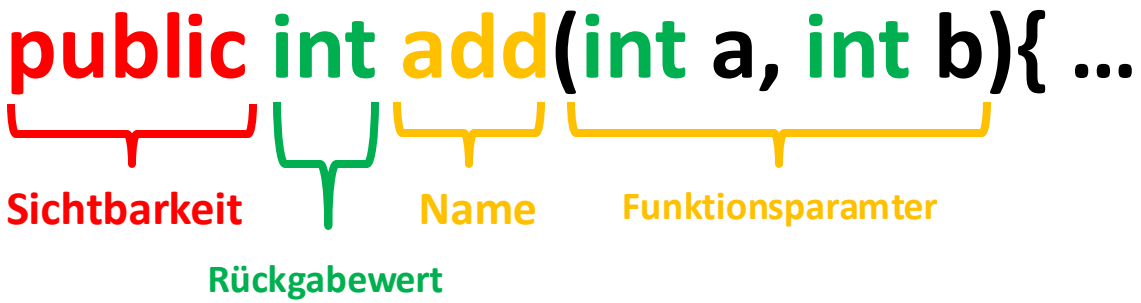
```
1  public class WrapperDemo {
2      public static void main(String[] args) {
3          Integer a = 12;
4          System.out.println(Integer.toHexString(a)); // c
5          modifyInteger(a);
6          System.out.println(Integer.toHexString(a)); // c
7      }
8      public static void modifyInteger(int a) {
9          a+=3;
10         System.out.println(Integer.toHexString(a)); // f
11     }
```

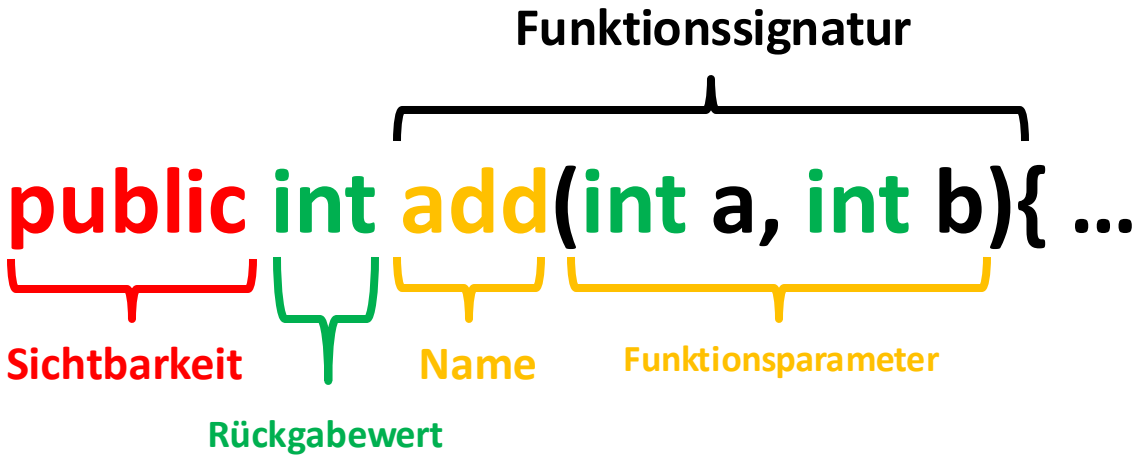
Vergleichen von Objekten (Demo)

Zusammenfassung Demo 2

- **==** - Operator vergleicht bei Objekten die Objektreferenzen
- Es ist **immer** sicherer bei Objekten die Vergleichs-Methoden, wie **.equals()** zu verwenden, sofern vorhanden (Strings bspw.)
- sonst die gespeicherten Werte einzeln vergleichen
- oder die **.equals()** Methode selbst implementieren (später Übung in der Übung zu **OO-Programmierung**)

```
public int add(int a, int b){ …
```


public int add(int a, int b){ …
Sichtbarkeit
Rückgabewert
Name
Funktionsparamter


Funktionssignatur
public int add(int a, int b){ …
Sichtbarkeit
Rückgabewert
Name
Funktionsparameter

Method-Overloading

- Eine Funktion kann überladen werden, in dem die Funktionsparameter verändert werden
- trotz gleichem Namen hat sie dann eine andere Signatur
- **add** kann durch Überladung mit unterschiedlichen Parametern aufgerufen werden, hat nun unterschiedliche Rückgabetypen

```java
public class Main {
    // Erste Version: Addiert zwei int-Werte
    public static int add(int a, int b) {
        return a + b;
    }
    // Überladene Version: Addiert drei int-Werte
    public static int add(int a, int b, int c) {
        return a + b + c;
    }
    // Überladene Version: Addiert zwei double-Werte
    public static double add(double a, double b) {
        return a + b;
    }
}
```

Rekursion

Rekursion (Anfang)

- Wir reden von Rekursion, wenn wir die gleiche Funktion kontrolliert in sich selbst einsetzen
- Wir reduzieren das Problem damit auf Unterprobleme
- sobald diese (Sub)probleme gelöst sind, können wir mit (sub)-Lösungen die Hauptlösung konstruieren
- Meist intuitiver im Ansatz, manchmal schwieriger in der Praxis, **extrem lohnenswert zu können. Daher Zeit zum üben nutzen!**

How to Rekursion

- Wie wende ich das Problem auf eine Teilmenge der Eingabe an.
- Was sind die Basisfälle (Länge = 0, Länge=1, Referenz=Null, etc)
- In welchen Fällen terminieren wir für die Unterprobleme
- **Ratschlag:** Zuerst Abbruchbedingungen in die Funktion schreiben, dann rekursiv weiterrechnen (macht es wirklich sehr viel leichter. :))
- **Ratschlag 2:** Auf dem Papier ein bisschen zu Kritzeln hilft **wirklich** stark, einen Lösungsansatz zu entwickeln.

racecar

Demo

racecar

Aufgerufen
Nicht überprüft
Valide

racecar

Aufgerufen
Nicht überprüft
Valide

racecar

Aufgerufen
Nicht überprüft
Valide

racecar

Aufgerufen
Nicht überprüft
Valide

racecar

Aufgerufen
Nicht überprüft
Valide

racecar

Aufgerufen
Nicht überprüft
Valide

racecar

Aufgerufen
Nicht überprüft
Valide

racecar

Aufgerufen
Nicht überprüft
Valide

Rekursive Intuition - Palindrom

- **Subproblem:** das innere Wort – **racecar** -> **aceca**
- **Basisfälle:** Länge < 2: **true**; Länge == 2: **true,** wenn beide Buchstaben gleich
- **Rekursion:** äußersten Buchstaben gleich (in-place) & inneres Wort ein Palindrom? (rekursiv)

Rekursive Intuition – String-Umkehr

- **Subproblem:** das innere Wort – **Hello** -> **ell**
- **Basisfälle:** Länge < 2: **bleibt gleich**;
- **Rekursion:** vertausche die äußersten Buchstaben (in-place) & rufe Funktion auf das innere Wort auf (rekursiv)

Vorbesprechung – Übung 3