Modul 404

Objektbasiert programmieren nach Vorgabe

Daniel Senften

22. Februar 2019

Talent Factory

Inhaltsverzeichnis

Modulidentifikation

Handlungsziele

1. Aufgrund einer Vorgabe den Ablauf darstellen.
2. Eine Benutzerschnittstelle entwerfen und implementieren.
3. Erforderliche Daten bestimmen und Datentypen festlegen.
4. Programmvorgabe unter Nutzung vorhandener Komponenten mit deren Eigenschaften und Methoden, sowie Operatoren und Kontrollstrukturen implementieren.
5. Beim Programmieren vorgegebene Standards und Richtlinien einhalten, das Programm inline dokumentieren und dabei auf Wartbarkeit und Nachvollziehbarkeit achten.
6. Programm auf Einhaltung der Funktionalität testen, Fehler erkennen und beheben.

Bereitsstellen der Software

Installation der Umgebung

Wir unterscheiden im Wesentlichen zwischen den beiden Installationstypen JRE (*Java Runtime Environment*) und JDK (*Java Development Kit*).

Die wichtigsten *Links* zu diesen Umgebungen finden wir hier:

- java.com
- openjdk.java.net

Werkzeuge und Befehle

Die JDK-Tools und ihre Befehle ermöglichen es uns, Entwicklungsaufgaben wie das Kompilieren und Ausführen eines Programms, das Paketieren von Quelldateien und vieles mehr zu erledigen.

Hier nur eine kleine Auswahl:

- javac—Kompilieren von Klassen- und Schnittstellendefinitionen in Bytecode.
- java—Starten/Interpretieren eines Java Programms.
- jar—Erstellen eines Java Archives.
- jshell—Interaktives Auswerten von Deklarationen, Anweisungen und Ausdrücken der Programmiersprache Java.

Beliebte Java-Lernressourcen

- **Finch Robot**—Dieser kleine Roboter von Bird Technology nutzt verschiedene Sensoren, die via Java angesteuert werden können.
- **Oracle Academy**—Die Oracle Academy bietet verschiedene Online-Kurse und Anleitungen
- **Scratch**—Dies ist eine sehr einfache, am MIT entwickelte Programmierumgebung. Typischerweise wird diese im Vorschulalter eingesetzt.
- **BlueJ**—Als sehr professionelle und stark vereinfachte (Entwicklungs-) Umgebung wird BlueJ auch im späteren Alter noch für Schulungszwecke verwendet.

Übung - Installation der Software

Bevor wir mit unseren Übungen starten müssen wir sicherstellen, dass Java und seine *Tools* komplett und korrekt installiert sind.

Bitte installieren Sie das *Java Development Kit* (OpenJDK) gemäss <jdk.java.net/11>.

Anschliessende Kontrolle der Installation mit:

```
# java -version
openjdk version "11.0.1" 2018-10-16
...
```

Entwicklungsumgebungen

Die drei am häufigsten für die Java-Entwicklung gewählten IDEs sind

- IntelliJ IDEA
- Eclipse
- NetBeans

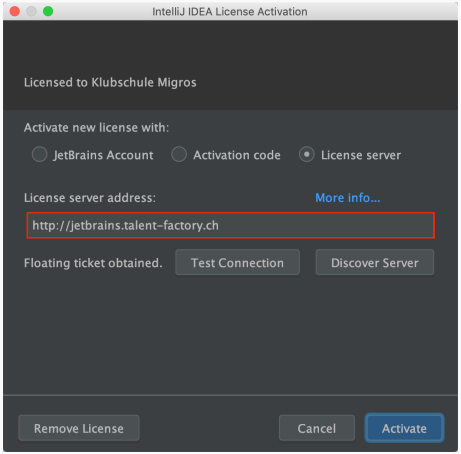
Übung - Installation der Entwicklungsumgebung

Ich habe mich für IntelliJ IDEA Ultimate entschieden. Obwohl es nicht kostenlos wie Eclipse oder NetBeans ist, glaube ich, dass der Produktivitätsgewinn den Preis wert ist.

Für die Ausbildung können wir die Umgebung allerdings kostenfrei nutzen.

Installiert kann diese Software via <www.jetbrains.com/idea/download>

Übung - Eingeben der Lizenz



Während der Dauer dieser Ausbildung dürfen wir die Schulungslizenz kostenfrei nutzen.

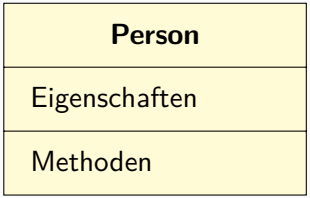
Wir referenzieren an dieser Stelle eine sogenannte *floating license*, die uns von der Talent Factory zur Verfügung gestellt wurde.

Objekte und Klassen

Fundamentale Konzepte

- Klassen
- Objekte
- Eigenschaften | Attribute
- Methoden
- Parameter
- *Daten Typen*

Darstellung einer Klasse




```
public class Person {
// Eigenschaften
// Methoden
}
```

UML Diagramm vs Implementation

Klasse vs. Objekt

Doch was ist eigentlich der Untersdchied zwischen Klassen und Objekten?

Objekte stellen *Dinge* aus der realen Welt oder aus einer Problemdomäne dar (Beispiel: “Das weisse Auto da unten auf dem Parkplatz”).

Klassen repräsentieren Objekte einer bestimmten Art (Beispiel: “Auto”).

Meine erste Java Klasse

```java
package academy;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Sichtbarkeit von Objekten

Es existiern vier unterschiedliche Zugriffsstufen für die Objekte (Daten und Methoden).

Modifier	Class	Package	Subclass	Universe
private	x			
default	x	x		
protected	x	x	x	
public	x	x	x	x

Übung: Ein einfacher Taschenrechner

Für unsere nächste Übung erstellen wir ein neues Projekt aus einem *Repository*. Da es sich um ein öffentliches Projekt handelt sollte der Zugriff auch ohne einen persönlichen Zugang möglich sein.

Das Projekt ist unter folgendem Link zu finden.

https://github.com/dsenften/example-simple-calculator.git

Instanzen erzeugen

```
public class Calculator {

    public static void main(String[] args) {
        String numbers = "1,2,3,4";
        System.out.println(" numbers = " + numbers);

        StringCalculator calculator = new StringCalculator();
        int output = calculator.add(numbers);
        System.out.println(" output = " + output);
    }
}
```

Darstellen von einfachen Grafiken

Für die nächsten Beispiele verwenden wir ein anderes Projekt, welches wir auch unter GitHub finden:

https://github.com/dsenften/example-figures.git

Erzeugen von Objekten

Basierend auf den installierten Klassen wollen wir nun die zur Verfügung gestellten Klassen wie als Objekte erzeugen und darstellen.

Übung
Erzeugen und darstellen eines Kreis.

```
Circle circle = new Circle();
circle.setVisible();
```

Einfache Datentypen in Java

Datentyp	Grösse	Wertebereich
boolean		**true**/**false**
char	16 bit	$0 \ldots 65'535$
byte	8 bit	$-128 \ldots 127$
short	16 bit	$-32'768 \ldots 32'767$
int	32 bit	$-2'147'483'648 \ldots 2'147'483'647$
long	64 bit	$-9'223'372'036'854'775'808 \ldots 9'223'372'036'854'775'807$
float	32 bit	$\pm 1.402'398 \times 10^{-45} \ldots 3.402'823 \times 10^{38}$
double	64 bit	$\pm 4.922'656 \times 10^{-324} \ldots 1.797'693 \times 10^{308}$

Klassendefinitionen

Syntaktischer Aufbau einer Klasse

```
classDeclaration
    : 'class' Identifier typeParameters?
      ('extends' type)?
      ('implements' typeList)?
      classBody
    ;
```

Wichtige Konzepte

- Eigenschaften
- Methoden
- Konstruktoren
- Parameter
- Zuweisungen

Objektinteraktion

Beispiel: Laborkurse

Ein einfaches Beispiel mit Kursen und Studenten. Es zeigt einmal mehr auf, wie wir Objekte, Attribute und Methoden verwenden. In einem späteren Abschnitt werden wir an dieser Stelle die Vererbung integrieren.

https://github.com/dsenften/example-course.git

Objektsammlungen

Einfacher *Cache* ohne *Generics*

```
public class CacheString {
    private String text;

    public void addText(String text) {
        this.text = text;
    }

    public String getText() {
        return this.text;
    }
}
```

Einfacher *Cache* ohne *Generics*

```
public class CacheShirt {
    private Shirt shirt;

    public void addText(Shirt shirt) {
        this.shirt = shirt;
    }

    public Shirt getShirt() {
        return this.shirt;
    }
}
```

Schreiben generischer Klassen

```
public class Cache<T> {
    private T cache;

    public void add(T cache) {
        this.cache = cache;
    }

    public T get() {
        return this.cache;
    }
}
```

Collections

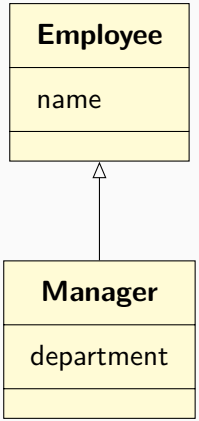
- Eine Sammlung ist eine Objekt, welches eine Reihe anderer Objekte verwaltet
  - Einzelne Objekte einer Sammlung nennen wir Elemente
  - Einfache/Primitive Datentype sind nicht erlaubt
- Es existieren viele vordefinierten Typen
  - *Stack*, *Queue*, *Hash*
- Das *Collection API* basiert stark auf generischen Klassen

Klassendesign durch Vererbung

Vorteile durch Vererbung

- Vermeidung von Code-Duplizierung
- Wiederverwendung von Quelltext
- Einfachere Wartung
- Erweiterbarkeit

Vererbungshierarchie in der Biologie




```
Employee emp = new Employee();
emp.setName("Peter");
```

```
Manager mgr = new Manager("Jack");
mgr.setName("Peter");

// Zugang zu allen Methoden von
// Manager via Casting
((Manager) mgr).getDepartment();
```

Vererbung | Inheritance

```
public class Employee {
    private final String firstName;
    private final String lastName;

    public Employee(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```

Vererbung | Inheritance

```
public class Manager extends Employee {
    private final String department;

    public Manager(String firstName, String lastName,
                   String department) {
        super(firstName, lastName);
        this.department = department;
    }
}
```

Übung: Initialisierung von Subklassen

Versuche mit Hilfe des *Debuggers* die Initialisierung aller Objekte in der Vererbungshierarchie Employee und Manager nachzuvollziehen und zu verstehen.

Was passiert beispielsweise, wenn wir in der Klasse Employee folgende Zeile verwendet hätten:

```
private String firsName = "undefined";
```

Definition einer Schnittstellen | *Interface*

```
public interface ElectronicDevice {
    void turnOn();
    void turnOff();
}
```

Implementation einer S hnittstelle

```
public class Television implements ElectronicDevice {

    @Override
    public void turnOn() { }

    @Override
    public void turnOff() { }

    public void changeChannel(int channel) {}
    public void initializeScreen() {}
}
```