GFN

Gemeinsam weiterbilden

Objektorientierte Programmierung

1.0 Das Konzept der Objektorientierten Programmierung

Einleitung

- Standardmethode der Softwareentwicklung
- Strukturierte Programmierung als Vorläufer der OOP
- Wesentliche Bestandteile sind Klassen und Objekte
- Basisaspekte der objektorientierten Programmierung:
  - Datenkapselung
  - Polymorphie
  - Vererbung
- Objektorientierte Programmierung folgt verschiedene Prinzipien

1.1 Single Responsibility Principle

Prinzip einer einzigen Verantwortung

- Eine Klasse bzw. ein Objekt hat genau eine Verantwortung
- Eine Verantwortung wird genau auf eine Klasse übertragen

- Beispiel: Der Trainer bringt Ihnen die objektorientierte Programmierung näher, seine Aufgabe ist es nicht Ihnen auch das Kochen beizubringen. In Ihrem Kurs ist nur ein Trainer mit dem Erreichen der Kursziele betraut.

1.2 Separation of Concerns

Trennung der Anliegen

- Ein Anliegen ist eine zusammenhängende, in sich geschlossene Teilaufgabe des Programms.
- Die Lösung dieser Teilaufgabe soll nicht über mehrere Klassen verteilt sein.
- In der Regel werden Funktionalitäten separiert.

- Beispiel: Eine Klasse kümmert sich um das Auslesen aus der Datenbank, eine weitere Klasse sorgt für die Darstellung der Information.

1.3 Don‘t repeat yourself

Wiederholungen vermeiden

- Redundanzen im Quelltext
  - blähen diesen unnötig auf
  - machen ihn schlechter wartbar
- Teile des Quelltextes sollten wiederverwendbar sein

- Beispiel: Eine komplexe Operation wird mehrmals im Quelltext durchgeführt. Durch Mechanismen, die Sie noch kennenlernen werden, extrahieren wir diese Operation und müssen in Folge nur noch die Extraktion ansprechen, nicht die Operation selbst. Änderungen können so an einem Ort durchgeführt werden und wirken sich auf alle Aufrufe aus.

1.4 Open-Closed-Principle

Offen für Erweiterungen, geschlossen für Änderungen

- Muss die Funktionalität einer Klasse angepasst werden, soll man sie erweitern können.
- An der Klasse selbst soll nichts geändert werden.
- Nur in der Erweiterung wird die Abweichung der Funktionalität formuliert.

- Beispiel: Möchte man auf Grund des Erfolges eines Kinofilmes die Kindheit des Protagonisten näher beleuchten, wird man einen weiteren Kinofilm drehen, nicht aber den alten Kinofilm, ergänzt durch Kindheitsszenen, neu drehen.

1.5 Program to Interfaces

Halte Inhalte von Klassen allgemein

- Soll eine Klasse ein bestimmte Funktionalität mit verschiedenen Ausprägungen haben, halte diese Funktionalität allgemein

- Beispiel: Ein Fahrzeug kann vorwärts und rückwärts fahren. Realisiere die Fahrtrichtung nicht in der Klasse, sondern halte es allgemein mit fahren. Ob es nun vorwärts oder rückwärts fährt wird an einer anderen Stelle (Bsp.: Interface) geregelt. Kann das Fahrzeug irgendwann auch seitwärts fahren muss man nichts an der Klasse ändern, sondern nur an eben dieser anderen Stelle.

- Steigende Komplexität in der Programmierung, sinkende Komplexität in der Klasse
- Auflösung von Abhängigkeiten

1.6 Dependency Inversion Principle

Fokus auf Gemeinsamkeiten

- Ähnlich dem vorangegangenen Prinzip
- Konkrete Abhängigkeiten vermeiden, auf Gemeinsamkeiten fokussieren

- Beispiel: Eine Klasse Telefonverzeichnis soll auf eine Datenbank zugreifen. Verschiedene Datenbankmanagementsysteme haben unterschiedliche Zugriffsvarianten. Die Zugriffsmöglichkeiten bleiben allerdings gleich (lesen, schreiben, ändern). Nutze in der Klasse die, in einem Interface formulierten, Gemeinsamkeiten statt alle Varianten in der Klasse zu definieren. Am Ende hat man die Klasse Telefonverzeichnis, ein Interface der Gemeinsamkeiten und die Klassen ZugriffDBMS1 und Zugriff DBMS2 die abhängig von dem Interface sind. Die Klasse Telefonverzeichnis nutzt nur die Gemeinsamkeiten und ist so nicht abhängig von DBMS1 oder DBMS2.

2.0 Objekte

Der zentrale Akteur

- Objekte sind Dinge des realen Lebens (Bildschirm, Tasse, Person usw.).
- Objekte sind *Instanzen* einer Klasse
- Objekte werden mit dem Schlüsselwort *new {Klassenname}* erstellt (Bsp.: new Person())
- Objekte fassen sowohl Zustände, als auch Funktionalitäten zusammen.
- Insbesondere die Zustände, aber mitunter auch die Funktionalitäten werden vor direktem Zugriff von Außen geschützt. Dies bezeichnet man als *Datenkapselung*.
- Beispiel: Eine Person hat einen Namen und eine Haarfarbe. Zudem kann unsere Person schlafen. Die Person schätzt auch besonders ihre Privatsphäre, und möchte auf keinen Fall, dass ein Fremder die Haarfarbe ändern kann oder ihr vorschreibt wie sie zu schlafen hat.

2.1 Klassen

Die Vorlage

- Klassen sind allgemeingültige Vorlagen eines oder mehrerer Objekttypen
- Gemeinsamkeiten werden in Klassen zusammengefasst.
- In Klassen werden *Eigenschaften/Attribute* eines Objektes formuliert, nicht jedoch der Zustand dieser Eigenschaft.
- Funktionalitäten in Klassen und Objekten nennt man *Methoden.*
- Mit dem Prinzip der *Klassifizierung* müssen Eigenschaften und Methoden nicht für jedes Objekt erneut formuliert werden. Redundanzen werden so vermieden.

- Beispiel: Durch unsere Klasse wissen wir zwar, dass eine Person einen Namen hat, aber wie genau der nun lautet erfahren wir erst mit dem Objekt.

2.2 Die Bestandteile einer Klasse

Klassenname, Attribute und Methoden

- Eine Klasse besteht aus 3 Teilen:
  - Klassenname
  - Attribute
  - Methoden

- Der Klassenname dient der eindeutigen Benennung
- Attribute stellen die abstrakten Eigenschaften eines Objektes dar.
- Methoden sind die Fähigkeiten eines Objektes

1.5 Java

Einleitung

- Erscheinungsjahr: 1995
- Entwickelt von Sun Microsystems, später Oracle
- Objektorientierte Programmiersprache
- Starke, statische Typisierung
- Plattformunabhängig
  Der Quelltext wird nicht direkt in Maschinencode kompiliert, sondern in Bytecode umgewandelt, der dann von eine Laufzeitumgebung interpretiert werden kann
- Umfangreiche Einsatzmöglichkeiten für Desktop-, Web- oder Mobil-Anwendungen

1.5 Java

Installation und Bereitstellung

- Download und Installation JDK

  https://www.oracle.com/java/technologies/downloads/

- Quelltexteditor (Bsp.: Eclipse)

  https://www.eclipse.org/downloads/

1.5 Java

Klassen in Java

- Quelltext-Dateien in Java tragen den Namen ihrer beinhalteten Klasse (mit Ausnahmen)
- Schlüsselwort für das Erstellen einer Klasse ist *class*
- Weitere Modifizierungen sind möglich (Bsp.: public)

MyClass.java

```
1
2 public class MyClass {
3
4
5
6 }
7
```

1.5 Java

Die Main-Methode

- Einstiegspunkt in eine Java-Anwendung
- Notwendig um eine Java-Applikation zu starten
- Die gesamte Anwendung entwickelt sich aus der Main-Methode

MyClass.java

```
1
2  public class MyClass {
3
4      public static void main(String[] args) {
5
6          // Programmablauf wird hier initialisiert
7
8      }
9
10 }
11
```

```
D:\Java\OOP2\bin>java MyClass
Fehler: Hauptmethode in Klasse MyClass nicht gefunden. Definieren Sie die Hauptmethode als:
   public static void main(String[] args):
```

1.5 Java

Kompilieren und Aufrufen von Java-Anwendungen

- Über Eclipse:
  - Run-Button 
  - Kompilierung automatisch
- Über Konsole:
  - Kompilieren mit *javac* <Name> im Verzeichnis der Quelldatei
  - Ausführen mit *java* <Name> im Verzeichnis der *class*-Datei

MyClass.java

```
1
2  public class MyClass {
3
4      public static void main(String[] args) {
5
6          System.out.println("Hello World");
7
8      }
9
10 }
11
```

```
D:\Java\OOP2\src>javac MyClass.java

D:\Java\OOP2\src>java MyClass
Hello World
```

1.5 Java

Objekte in Java

- Erstellung mit dem Schlüsselwort *new*
- In der Variablen *p* liegt eine Instanz (Objekt) der Klasse *Person*
- Von *p* aus kann nun auf *name*, *vorname* oder *essen()* zugegriffen werden.

```
1
2 public class MyClass {
3
4     public static void main(String[] args) {
5
6         Person p = new Person();
7
8         p.essen();
9
10    }
11
12 }
13
```

```
1
2 public class Person {
3
4     public String name;
5     public String vorname;
6
7     public void essen() {
8         System.out.println("Person isst");
9     }
10
11 }
12
```

1.5 Java

Bestandteils von Klassen in Java

- Klassenname
  - Eindeutig
  - Beginnt mit Großbuchstaben
  - Bsp.: Person
- Attribute
  - Bsp.: name, vorname
  - Kleinbuchstaben oder lowerCamelCase
- Methoden
  - Bsp.: essen()

```
1
2  public class Person {
3
4      public String name;
5      public String vorname;
6
7      public void essen() {
8          System.out.println("Person isst");
9      }
10
11 }
12
```

1.5 Java

Zugriff auf Attribute und Methoden

- Auf Attribute und Methoden kann zugegriffen werden, indem man diese mit einem Punkt an die *Referenz* hängt
- Rudimentäre, einfachste Form des Zugriffs

```java
public class Person {

    public String name;
    public String vorname;

    public void essen() {
        System.out.println("Person isst");
    }

}
```

```java
public class MyClass {

    public static void main(String[] args) {

        Person p = new Person();

        p.name = "Mustermann";
        p.vorname = "Annette";

        p.essen();

    }

}
```

1.1 Selbstreferenzierung

Hey ?! Sie da!

- Nehmen wir an, wir haben 2 Objekte der Klasse Person.
  Beide Objekte greifen auf die Methode schlafen() zu.
- schlafen() erzeugt die Ausgabe: „{Name der Person} schläft.“
- Beispiel: „Peter schläft.“
  „Annette schläft.“
- Problem: Zum Zeitpunkt der Erstellung der Klasse bzw. der Methode schlafen() kennen wir die Namen der Personen noch gar nicht.
- Lösung: Objektorientierte Sprachen bieten ein Schlüsselwort mit dem in der Klasse auf zukünftige Objekte verwiesen werden kann.
- Beispiel: In Java ist dieses Schlüsselwort *this*, in Python *self*.

1.5 Java

Selbstreferenzierung in Java

- Um in der Klasse auf das jeweils verwendete Objekt zu referenzieren wird das Schlüsselwort *this* verwendet.

```
Annette isst.
Peter isst.
```

```java
public class Person {

    public String name;
    public String vorname;

    public void essen() {
        System.out.println(this.vorname + " isst.");
    }

}
```

```java
public class MyClass {

    public static void main(String[] args) {

        Person p1 = new Person();
        p1.vorname = "Annette";
        p1.essen();

        Person p2 = new Person();
        p2.vorname = "Peter";
        p2.essen();

    }

}
```

1.1 Konstruktoren

Automatisierung bei Objekterstellung

- Methode die bei Erstellung einer Instanz automatisch aufgerufen wird
- Eine Klasse kann mehrere Konstruktoren besitzen
- Erst durch Konstruktoren bekommen Instanziierungen Parameter
- Konstruktoren werden in der Klasse definiert
- In Konstruktoren können Vorgänge die stets ausgeführt werden müssen automatisiert werden.

- Beispiel: Jede Person hat einen Namen. Dieser Vorgang soll automatisiert werden. new Person(„Klaus“) -> im Konstruktor wird formuliert, dass die Person den Namen „Klaus“ hat.

1.5 Java

Konstruktoren in Java

- Konstruktoren tragen in Java den gleichen Namen wie die Klasse.
- Es ist möglich mehrere Konstruktoren bereitzustellen. Sie müssen sich allerdings in der Parameterliste unterscheiden.
- Konstruktoren können verkettet werden.

```
1
2  public class Person {
3
4      public String name;
5      public String vorname;
6
7      public Person() {
8          this.name = "unbekannt";
9          this.vorname = "unbekannt";
10     }
11
12     public void essen() {
13         System.out.println(this.vorname + " isst.");
14     }
15
16 }
17
```

```
1
2  public class MyClass {
3
4      public static void main(String[] args) {
5
6          Person p3 = new Person();
7          p3.essen();
8
9      }
10
11 }
12
unbekannt isst.
```

1.5 Java

Mehrere Konstruktoren in Java

```java

public class Person {

    public String name;
    public String vorname;

    public Person() {
        this.name = "unbekannt";
        this.vorname = "unbekannt";
    }

    public Person(String name) {
        this.name = name;
    }

    public Person(String name, String vorname) {
        this.name = name;
        this.vorname = vorname;
    }

    public void essen() {
        System.out.println(this.vorname + " isst.");
    }

}

```

```java

public class MyClass {

    public static void main(String[] args) {

        Person p4 = new Person();
        p4.essen();

        Person p5 = new Person("Hansen");
        p5.essen();

        Person p6 = new Person("Doe", "Jane");
        p6.essen();

    }

}

```

```
unbekannt isst.
null isst.
Jane isst.
```

1.5 Java

Konstruktoren in Java verketten

```java

public class Person {

    public String name;
    public String vorname;

    public Person() {
        this("unbekannt", "unbekannt");
    }

    public Person(String name) {
        this(name, "unbekannt");
    }

    public Person(String name, String vorname) {
        this.name = name;
        this.vorname = vorname;
    }

    public void essen() {
        System.out.println(this.vorname + " isst.");
    }

}

```

```java

public class MyClass {

    public static void main(String[] args) {

        Person p4 = new Person();
        p4.essen();

        Person p5 = new Person("Hansen");
        p5.essen();

        Person p6 = new Person("Doe", "Jane");
        p6.essen();

    }

}

```

```
unbekannt isst.
null isst.
Jane isst.
```

1.1 Typisierung

Kategorisierung von Daten

- Unterscheidung *statische* und *dynamische* Typisierung
  - statisch: Der Datentyp eines Wertes wird im Quelltext festgelegt, und beim Kompilieren überprüft
  - dynamisch: Der Datentyp wird erst zur Laufzeit der Anwendung überprüft
- Unterscheidung *starke* und *schwache* Typisierung
  - stark: Eine Variable muss dem Typ der Umwandlung eines Wertes entsprechen
  - schwach: Eine Variable muss nicht dem Typ der Umwandlung eines Wertes entsprechen
  - Beispiel: Der Zahlenwert 1 soll in eine Variable vom Typ Wahrheitswert gespeichert werden. Stark typisiert ist das nicht möglich (Ganzzahl/Wahrheitswert). Schwach typisiert wird die 1 als Wahr angenommen und in die Variable gespeichert.
- Jede Information wird einem Datentyp zugeordnet (Bsp.: „Hallo Welt“ -> Zeichenkette, 42 -> Ganzzahl)
- Klassen stellen den Datentyp des Objekts dar (Bsp.: Objekt „obj“ der Klasse Person ist vom Typ Person)

1.5 Java

Typisierung in Java

- Für alle Werte muss vorab ein Datentyp deklariert werden.
- Datentypen in Java sind zum Beispiel String für Zeichenketten und int für Ganzzahlen
- In Person p8 entsteht ein Fehler, da versucht wird eine Ganzzahl in eine Variable vom Typ Zeichenkette zu schreiben

```
Type mismatch: cannot convert from int to String
```

```java

public class Person {

    public String name;
    public String vorname;

    // ...
```

```java

public class MyClass {

    public static void main(String[] args) {

        Person p7 = new Person();
        p7.name = "Horst";

        Person p8 = new Person();
        p8.name = 42;

    }

}

```

1.1 Sichtbarkeit von Daten und Methoden

Sichtbarkeitsstufen: public, private, protected & package

- In der Modellierung von Datenstrukturen werden 4 Sichtbarkeitsstufen unterschieden:
  - *public*: Alle *public* deklarierten Elemente sind vollumfänglich ansprechbar
  - *private*: Private Elemente sind nur in ihrer eigenen Klasse ansprechbar. Zugriff von Außen ist nicht möglich.
  - *protected*: Nur die eigene Klasse und mögliche Kindklassen können zugreifen.
  - *package*: Zugriff wird nur im aktuellen Namensraum gewährt.
- Sichtbarkeitsstufen sind ein wesentliches Werkzeug der *Datenkapselung.*
- Die Existenz von Eigenschaften und Methoden kann verborgen werden.
- In der Regel sind alle Klasseneigenschaften *private* gehalten.

1.5 Java

Sichtbarkeit in Java

```
1
2 public class Person {
3
4     public String name;
5     private String vorname;
6
7     // ...
8
```

```
1
2 public class MyClass {
3
4     public static void main(String[] args) {
5
6         Person p9 = new Person();
7         p9.name = "Horst";
8         p9.vorname = "Annette";
9
10    }
11
12 }
13
```

```
The field Person.vorname is not visible
```

1.1 Kapselung mit Gettern und Settern

Zugriff auf Umwegen

- Getter und Setter sind spezielle Methoden einer Klasse
- Getter und Setter stehen in direktem Zusammenhang mit einer Eigenschaft einer Klasse
- Mit diesen Methoden können Werte einer Eigenschaft ausgelesen (get) oder belegt (set) werden
- Um auf *private* deklarierte Eigenschaften von außerhalb der Klasse zugreifen zu können benötigt man Getter und Setter
- Der Zugriff auf Eigenschaften kann über Getter und Setter gesteuert und kontrolliert werden.

- Beispiel: Die Eigenschaft „Länge“ einer Klassen muss größer 0 sein. Diese Bedingung kann ich im *Setter* kontrollieren. Die Ausgabe der „Länge“ soll mit Maßeinheit erfolgen, dies kann im *Getter* gesteuert werden.

1.5 Java

Datenkapselung in Java

```java

public class Person {

    public String name;
    private String vorname;

    public void setVorname(String vorname) {
        this.vorname = vorname;
    }

    public String getVorname() {
        return "Vorname: " + this.vorname;
    }

    // ...
```

```java

public class MyClass {

    public static void main(String[] args) {

        Person p10 = new Person();
        p10.setVorname("Bärbel");

        System.out.println(p10.getVorname());

    }

}

```

```
Vorname: Bärbel
```