

Data-Hiding
Objekterzeugung
Datensatz





Außen- und Innensicht

Außensicht = abstrakter Datentyp (ADT) aus Anwendersicht
nur sichtbar, was für die Verwendung bekannt sein muss

Innensicht = Implementierung des abstrakten Datentyps
alle Implementierungsdetails sichtbar

- unterschiedliche Schwerpunktsetzung (Vorrang für Außensicht)
- unterschiedliche Sichtbarkeit von Methoden und Variablen (**Data-Hiding**)



Wie und warum Data-Hiding

vereinfacht gilt: **public** gehört zur Außen- und Innensicht
private gehört nur zur Innensicht

Innensicht geändert und $\left[\begin{array}{ll} \text{Außensicht unverändert} & \rightarrow \text{Anwendungen nicht betroffen} \\ \text{Außensicht geändert} & \rightarrow \text{Anwendungen finden und ändern} \end{array} \right.$

→ zwecks besserer Wartbarkeit möglichst viel **private**
(obwohl dies mit wenig Erfahrung oft als Nachteil erlebt wird)



Sichtbarkeit auf Klassenebene

Außen- und Innensicht beziehen sich auf einzelne Objekte,
aber `public` und `private` beziehen sich auf Klasse:

public Methoden und Variablen überall zugreifbar,

private Methoden und Variablen innerhalb der Klasse zugreifbar

```
public class A {  
    private int x;  
    public int add(A a) {  
        return x + a.x;  
    }  
}
```

da Parametertyp von eigener Klasse A
darf auch auf private Teile von a zugegriffen werden
(Außen- und Innensicht nicht klar getrennt)

wahrscheinlich unterschiedliche x
(von unterschiedlichen Objekten)

Aufgabe: Warum Sichtbarkeit auf Klassenebene?

In Gruppen zu 2 bis 3 Personen:

Finden Sie Gründe für und gegen Sichtbarkeit auf Klassenebene (statt auf Objektebene).

Zeit: 2 Minuten

Modifizier `public` auf Klassen

Wenn `public` vor `class` steht:

Klasse allgemein verwendbar,
genau eine solche Klasse pro Datei,
Klassenname ist Dateiname (bis auf Endung)

Normalfall

Wenn kein `public` vor `class` steht:

Hilfsklasse, die nur von anderen Klassen im selben Ordner verwendet wird Ausnahme



Was beim Erzeugen eines Objekts passiert

bei Ausführung von `new A()` wird

Speicherbereich passender Größe reserviert (Objektvariablen und **Identität**),
Speicherbereich mit Null-Werten gefüllt (**Vorinitialisierung** der Objektvariablen),
ein **Konstruktor** für `A` ausgeführt um das Objekt zu initialisieren,
eine Referenz auf den Speicherbereich (jetzt Objekt) zurückgegeben

Identität: wenn `x == y` wahr,
dann referenzieren `x` und `y` gleichen Speicherbereich (selbes Objekt)

Konstruktor: ähnlich den Methoden programmierbar,
Default-Konstruktor wenn Konstruktor nicht selbst definiert

Konstruktor

```
public class Point {  
    private int x, y;  
  
    public Point(int initX, int initY) {  
        x = initX;  
        y = initY;  
    }  
    ...  
}
```

gleicher Name, kein Ergebnistyp

Parameterwerte für Initialisierung

Objekterzeugung: `new Point(3, 5)`

Aufgabe: Wozu Konstruktoren?

In Gruppen zu 2 bis 3 Personen:

Beschreiben Sie, was Konstruktoren von Methoden unterscheidet und wozu wir Konstruktoren verwenden können.

Zeit: 2 Minuten

Überladene Konstruktoren und Default-Konstruktor

```
public class Point {  
    private int x, y;  
  
    public Point(int initX, int initY) {  
        x = initX; new Point(3, 5)  
        y = initY;  
    }  
  
    public Point() {} Default-Konstruktor schaut genau so aus;  
wird automatisch in Klasse eingeführt,  
wenn kein anderer Konstruktor vorhanden  
        new Point()  
    ...  
        new Point() entspricht new Point(0, 0)  
}
```

Überladene Konstruktoren und this(...);

```
public class Point {  
    private int x, y;  
  
    public Point(int initX, int initY) {        new Point(3, 5)  
        x = initX;  
        y = initY;  
    }  
  
    public Point() {        new Point()  
        this(1, 1);  
    }  
  
    public Point(Point p) {  
        this(p.x, p.y);  
    }  
    ...  
}
```

Konstruktor-Aufruf darf nur als erste Anweisung
in einem Konstruktor vorkommen

Selbstreferenz this

```
public class Point {  
    private int x, y;  
  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public Point(Point p) {  
        this(p.x, p.y);  
    }  
    public Point copy() {  
        return new Point(this);  
    }  
    ...  
}
```

this referenziert Objekt, in dem wir uns befinden
Pseudovariablen – lesbar, aber nicht schreibbar

das ist Konstruktor-Aufruf, keine Selbstreferenz

Aufgabe: Übersicht

Rekapitulieren Sie:

Was sind und wozu verwenden wir abstrakte Datentypen?

Was haben Objekte mit abstrakten Datentypen zu tun?

Was hat Sichtbarkeit mit abstrakten Datentypen zu tun?



Datenstruktur \neq abstrakter Datentyp

Datenstruktur:

wie hängen Daten zusammen, wie sind sie auffindbar,
wie greifen Operationen auf die Daten zu,
offen: Programmiersprache, Typen, Größenbeschränkungen, ...

abstrakter Datentyp:

wie sind Objekte verwendbar (Außensicht),
offen: Implementierungsdetails, Algorithmen, Datenstrukturen, ...

Implementierung eines abstrakten Datentyps:

implementiert auch Algorithmen und Datenstrukturen,
klärt offene Punkte (aus Sicht von Datenstrukturen und abstrakten Datentypen)

Übergänge fließend

Datensatz ist sehr einfache Datenstruktur:

relativ uninteressant

Menge zusammengehörender Variablen,
die bei Bedarf gelesen oder geschrieben werden

Student:

regNumber
name
mail



Datensatz als abstrakter Datentyp

abstrakter Datentyp **abstrahiert** über Variablen in Datensatz

Art der Abstraktion entscheidend

da wird es interessant

Auswahl an Fragestellungen:

Wie sind die Werte der Variablen eingeschränkt?

Welche Variablen sind wann lesbar, welche wann schreibbar?

Bleiben Variablen hinter Abstraktion sichtbar?

Welche Abstraktion verspricht einfache Verwendbarkeit?

Aufgabe: Datenstruktur versus abstrakter Datentyp

In Gruppen zu 2 bis 3 Personen:

Überlegen Sie, wie durch Datenstrukturen und abstrakte Datentypen abstrahiert wird.
Warum unterscheiden sich diese Abstraktionen grundlegend voneinander?

Zeit: 2 Minuten



```
public class Student {  
    private final int regNumber;  
    private String name;  
  
    public Student(int regNumber, String name) {  
        this.regNumber = regNumber;  
        setName(name);  
    }  
  
    public int regNumber() { return regNumber; }  
    public String getName() { return name; }  
    public void setName(String name) { this.name = name; }  
}
```

wesentliche Funktionalität außerhalb der Klasse

Einschränkungen durch Typen auch in Außensicht

nach Initialisierung nur lesbar (auch ohne final)

Getter und Setter lassen Variablen durchscheinen

Umgang mit zusammenhängenden Daten

```
private static Student find(Student[] studs, int reg) {  
    for (Student stud : studs) {  
        if (stud.regNumber() == reg) {  
            return stud;  
        }  
    }  
    return new Student(reg, "Max Mustermann");  
}
```

ein Array, mehrere Variablen pro Arrayeintrag

indirekter Variablenzugriff für Vergleich

Rückgabe eines Datensatzes mit allen Variablen

Aufgabe: Zusammenhängende Daten

In Gruppen zu 2 bis 3 Personen:


Überlegen Sie, wie Sie die Suche nach Studierendendaten (zusammenhängende Daten) ohne Verwendung eines Datensatzes gestalten könnten.

Zeit: 2 Minuten

```
public class Student {  
    private final int regNumber;  
    private String name;  
    private String mail;  
  
    public Student(int regNumber, String name) {  
        this.regNumber = regNumber;  
        this.name = name;  
        mail = "e"+regNumber+"@student.tuwien.ac.at";  
    }  
  
    public void showPersonalData() { ... }  
    public void editPersonalData() { ... }  
    public void mail(String head, String text) { ... }  
}
```

wesentliche Funktionalität in die Klasse verschoben

keine Getter und Setter nötig
wenn alle zugreifenden Methoden innerhalb der Klasse



Idee hinter objektorientierter Programmierung

Schwerpunkt liegt nicht auf Datensatz, sondern auf Funktionalität
Datensatz bleibt hinter der Funktionalität gänzlich abstrakt

Entwurfsprinzip: Software-Objekt simuliert „reales Objekt“, wobei

- „reale Objekte“ auch immateriell sein können,
- nur in der Software nötige Eigenschaften simuliert werden

derart modellierte Objekte sind meist mit Funktionalität angereicherte Datensätze