Organisatorisches

- Übungen:
  - Diese Woche erste Wissensüberprüfungen in den Übungen.
  - Besprechung von häufigen Fehlern in der VO.

IT I: Heute

Wiederholung CuP ctd:

- *this*
- *ArrayList*
- Schleifen:
  - *for*
  - *for-each*
- Projekt *Kaffeeautomat*

Konzepte der OOP (5)

- ***Datenkapselung:***
  - Die Attribute von Objekten sollen von außen nicht direkt zugreifbar sein.
  - Attribute werden daher mit dem Vermerk **private** versehen.
  - Lesende (get-) und verändernde (set-) Methoden werden verwendet, um den Zugriff zu gewährleisten.
  - Dadurch kann die Integrität der Attributwerte sichergestellt werden. (Unerwünschte/unbeabsichtigte Veränderungen sind nicht so leicht möglich.)

Beispiel zur Sichtbarkeit

- An automobile consists of several parts and pieces and is capable of doing many useful things.
  - Awareness of the accelerator pedal, the brake pedal, and the steering wheel is important to the driver.
  - Awareness of the fuel injectors, the automatic braking control system, and the power steering pump is not important to the driver.

Pakete in Java

- Java-Projekte können durch Pakete (packages) strukturiert werden.
- Zusammengehörige Klassen können zusammengefasst werden.
- Ihre Sichtbarkeit nach außen kann definiert werden.

Sichtbarkeiten in Java

- **private**
  - Attribute und Methoden sind außerhalb der Klasse, in der sie definiert sind, nicht zugreifbar/sichtbar.
- ohne *visibility modifier*
  - Methoden sind von allen Klassen desselben Paketes zugreifbar.
- **public**
  - Methoden sind auch aus Klassen in anderen Paketen zugreifbar.

BlueJ Debugger

- Mit Hilfe des Debuggers kann der Programmablauf und die Veränderung der Attribut- und Variablenwerte verfolgt werden.
- Der Debugger wird durch das Setzen von *Breakpoints* in Methoden (und Konstruktoren) aktiviert.

Java-Konventionen für die Übungen

- Klassennamen beginnen mit einem Großbuchstaben.
- Attribut-, Methoden-, Parameternamen und lokale Variablen beginnen mit einem Kleinbuchstaben.
- Einrückungsstufen (bei Klassendefinitionen, Konstruktoren, Methoden) sind genau vier Leerzeichen.
- Eine Einrückungsstufe enden mit einer schließenden geschwungenen Klammer „}“, die nicht mehr eingerückt wird.
- Alle Attribute sind **private**.

Dokumentation

- Für das Verständnis von (fremden) Programmen ist besonders die verbale Beschreibung von Klassen und Methoden wichtig.
- In Java stehen dafür die Javadoc-Kommentare zur Verfügung.
- Aus diesen kann automatisch eine Dokumentation des Programms generiert werden.

Syntax von Kommentaren

- Javadoc:

```
/**
 *  Kommentar
 */
```

- Weitere Kommentare:

```
/*
 *  Kommentar
 */
```

- oder einzeilig:

```
/* Kommentar */
// Kommentar
```

Die Objekt-Referenz *this*

- Auf das Objekt, dessen Methode gerade ausgeführt wird, kann mit ***this*** zugegriffen werden.
- In einem Konstruktor bezeichnet ***this*** das Objekt, das gerade erzeugt wird.
- Auf Attribute kann eindeutig mit ***this***.*attributname* zugegriffen werden.

Dynamische Arrays: Die Klasse *ArrayList*

- Die Klasse *ArrayList* aus *java.util* kann im wesentlichen wie ein Array verwendet werden (allerdings mit einer anderen Syntax).
- Im Gegensatz zu Arrays ist die Größe einer ArrayList aber nicht im vorhinein festgelegt, sondern es können beliebig Elemente in die Liste eingefügt und aus der Liste entfernt werden. Die Größe der Liste verändert sich dabei automatisch.
- Weiters stellt die Klasse *ArrayList* zusätzliche Methoden zur Verfügung.

Verwendung einer ArrayList (1)

- Definition eines ArrayList-Datentyps:

```
ArrayList<Auto> fuhrpark;
```

- Eine ArrayList kann für jeden **nicht-primitiven** Datentyp definiert werden.
- Erzeugen einer ArrayList:

```
new ArrayList<Auto>();
```

- Eine neu erzeugte ArrayList ist leer, sie enthält keine Elemente.

Verwendung einer ArrayList (2)

- Das Elemente an der Stelle *i* in einer ArrayList erhält man von der Methode *get(i):*

  fuhrpark.get(i);
- Auch in einer ArrayList sind die Elemente mit 0 beginnend nummeriert, also 0,1,2,…
- Die Größe einer ArrayList erhält man durch die Methode *size()*:

  fuhrpark.size()
- Mit der Methode *set(i,element)* kann ein Element an die Stelle *i* geschrieben werden:

  fuhrpark.set(i,auto1);
- Die Stelle *i* muss es in der ArrayList geben, d.h.

  i < *size()*.

Verwendung einer ArrayList (3)

- Zusätzliche Elemente können mit der Methode *add(element)* am Ende der ArrayList angefügt werden:

  ```
  fuhrpark.add(auto2);
  ```

- Mit der Methode *add(i,element)* kann ein zusätzliches Element an der Stelle i in die ArrayList eingefügt werden:

  ```
  fuhrpark.add(i,auto3);
  ```

  Dabei werden die ursprünglichen Elemente ab der Stelle *i* um eine Stelle nach hinten verschoben, damit Platz für das zusätzliche Element geschaffen wird.

- Mit der Methode *remove(i)* kann das Element an der Stelle *i* aus der ArrayList entfernt werden:

  ```
  fuhrpark.remove(i);
  ```

  Die Elemente ab Stelle *i+1* werden um eine Stelle nach vorne verschoben, damit die Lücke geschlossen wird.

Iteration und Schleifen

- Schleifen erlauben die kontrollierte Wiederholung von Anweisungen/Aktionen.
- Diese schnelle Wiederholung ist eine der wesentlichen Stärken von Computern.
- Arrays und Schleifen
  - Mit Hilfe einer Schleife können dieselben Aktionen für alle (oder ausgewählte) Elemente eines Arrays durchgeführt werden.

Die *for-each* Schleife (1)

- Als sehr einfache Form der Schleife gibt es in Java die *for-each* Schleife.
- Damit können Aktionen für alle Elemente eines Arrays/einer ArrayList ausgeführt werden.

Die *for-each* Schleife (2)

- Für ein Array

  ```
  Datentyp[] liste;
  ```

  oder eine ArrayList

  ```
  ArrayList<Datentyp> liste;
  ```

  führt die for-each Schleife

  ```
  for(Datentyp el : liste) {
      Schleifenrumpf
  }
  ```

  den *Schleifenrumpf* für jedes Element des Arrays aus. Auf das jeweilige Element kann über die lokale Variable *el* zugegriffen werden.
- Der Name der Laufvariablen (hier *el*) kann frei gewählt werden.
- Die Laufvariable nimmt der Reihe nach alle im Array enthaltenen Werte an.

Die *for* Schleife

- Eine *for*-Schleife

```
for(Initialisierung;Fortsetzungsbedingung; Aktualisierung)
{
    Schleifenrumpf
}
```

wird ausgeführt, solange die *Fortsetzungsbedingung* **true** gibt.

- Eine *for*-Schleife wird meist verwendet, wenn vor der Schleife klar ist, wie oft diese durchlaufen werden soll, z.B.

```
for(int i=0; i<liste.size(); i++)
{
    ... liste.get(i) ...
}

for(int i=0; i<array.length; i++)
{
    ... array[i] ...
}
```

Statische Methoden

- Statische Methoden werden mit dem Wort *static* gekennzeichnet, z.B.

```
public static double sqrt(double a)
```

- Der Aufruf einer statischen Methode ist an kein Objekt gebunden und erfolgt über den Klassennamen, z.B.

```
Math.sqrt(x*x+y*y);
```

- Da eine statische Methode nicht an ein Objekt gebunden ist, kann eine solche Methode auch nicht unmittelbar auf Objekt-Attribute zugreifen.

BlueJ Debugger

- Mit Hilfe des Debuggers kann der Programmablauf und die Veränderung der Attribut- und Variablenwerte verfolgt werden.
- Der Debugger wird durch das Setzen von *Breakpoints* in Methoden (und Konstruktoren) aktiviert.

Code duplication

- Code duplication
  - is an indicator of bad design,
  - makes maintenance harder,
  - can lead to introduction of errors during maintenance.