LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mobile and distributed systems group

Javakurs für Anfänger

Einheit 10: Mehr zu Arrays & Klassen

Lorenz Schauer
Lehrstuhl für Mobile und Verteilte Systeme

Heutige Agenda

1. Teil:

- Arrays und die Verwendung der `ArrayList`

2. Teil: Weitere Aspekte der OO

- Klassen Variablen
  - Das Schlüsselwort `static`
  - Beispiel: Instanz-Zähler
- Klassen Methoden
- Unveränderliche Attribute (Konstanten)

Praxis:

- `ArrayList` verwenden
- Klasse Student anpassen
  - Instanz-Zähler implementieren

Lernziele

- Mit `ArrayLists` umgehen lernen
- Unterschiede zwischen Instanz- und Klasseneigenschaften verstehen
- Einführung der Schlüsselwörter: `static` und `final`

Programmieraufgabe

Aufgabe 1 (Aus der letzten Stunde): Nutzereingaben speichern

Erstellen Sie in Ihrem Eclipse-Projekt „Uebung09“ ein neues Programm „Nutzereingaben“, das folgende Aufgaben erledigt:

- Der Nutzer wird immer wieder aufgefordert eine Zahl einzugeben, solange bis eine 0 eingegeben wird.
- Das Programm speichert die Nutzereingaben (Ohne die zur Terminierung eingegebene 0) in eine passende Datenstruktur.
- Anschließend berechnet das Programm den Durchschnitt der eingegebenen Zahlen und gibt das Ergebnis auf der Konsole aus.
- Beachten Sie dabei bitte folgende Punkte:
  - Sie müssen überprüfen, ob eine eingegebene Zahl noch in Ihrer Datenstruktur gespeichert werden kann, bevor Sie die Zahl speichern!
  - Wenn das nicht mehr der Fall ist, müssen Sie sich überlegen, wie Sie mit der Nutzereingabe umgehen wollen:
    - Entweder stellen Sie mehr Speicher zur Verfügung und speichern die Zahl
    - Oder Sie sagen dem Nutzer, dass seine Zahl nicht mehr gespeichert werden kann und beenden die Aufforderung zur Zahleneingabe und fahren im Programm fort.

Arrays anpassen

Wiederholung:

- Die Größe eines Arrays kann nach dessen Definition nicht mehr verändert werden

Häufig weiß man aber zum Zeitpunkt der Array-Erzeugung nicht, wie groß es sein muss.

- Bsp.: Nutzereingaben werden in Array gespeichert. Aber wie viele Eingaben tätigt ein Nutzer?

Lösungen:

- Man erzeugt ein sehr großes Array und befüllt es dann nur teilweise
  - Speicherplatzverschwendung
  - Reicht die gewählte Größe wirklich aus?
    => Fehlerbehandlung

- **Verwendung einer dynamischen Array-Struktur**
  - `ArrayList` aus dem Paket `java.util`

Dynamische Arraystruktur

Die Array-Liste: `java.util.ArrayList`

- Sehr eng mit Arrays verwandt
  - Elemente sind wie bei Arrays linear angeordnet
  - Können mit Index (0 bis length-1) angesprochen werden
  - Ein Objekt der `ArrayList` wird mit `new` erzeugt
- Aber:
  - Elemente können dynamisch hinzugefügt und/oder entfernt werden
  - Keine Angabe einer vordefinierten Länge nötig
  - Nur komplexe Datentypen (Referenztypen) erlaubt!
    - Um dennoch primitive Datentypen zu nutzen werden Wrapperklassen verwendet
      - `Double`, `Integer`, usw.

Erzeugen von `ArrayLists`

Zunächst muss wieder die Klasse `ArrayList` aus dem Packet `java.util.ArrayList` importiert werden!

Bei der Deklaration einer `ArrayList` muss ein Typ der Elemente mit angegeben werden:

- Dieser steh in spitzen Klammern `<Typ>`
  - Stichwort: „generics“ wird später behandelt
- Allgemeine Syntax zur Erzeugung einer ArrayList:
  - `ArrayList<Datentyp> name = new ArrayList<Datentyp>();`

```
// Beispiele zum Erzeugen von ArrayLists verschiedener Datentypen:

ArrayList<Double> temperaturen = new ArrayList<Double>();

ArrayList<Integer> eingabeZahlen = new ArrayList<Integer>();

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

Arbeiten mit `ArrayList`

Auf einem `ArrayList`-Objekt können verschiedenste Methoden angewendet werden. Nähere Infos hierzu in der Java API Spezifikation

- http://docs.oracle.com/javase/8/docs/api/

Die wichtigsten Methoden sind in der Tabelle jeweils mit Beispiel dargestellt:

- Angenommen es existiert eine `ArrayList<Auto> autoListe`
- Und eine `ArrayList<Double> temperaturen;`

Methode (allgemein)	Funktion	Beispiele
`void clear()`	Löscht alle Elemente aus der `ArrayList`	`autoListe.clear();` `temperaturen.clear();`
`E get(int index)`	Liefert das Element vom Typ `E` an der Position `index` zurück	`Auto a = autoListe.get(2);` `double b = temperaturen.get(5);`
`E set(int index, Object o)`	Ersetzt das Element `E` an der Position `index` durch das übergebene Objekt `o`	`autoListe.set(2,fiat);` `temperaturen.set(3,19.8);`
`int size()`	Liefert die Anzahl der Elemente in der Liste zurück	`int anzahlAutos = autoListe.size();`

Arbeiten mit `ArrayList`

Methode (allgemein)	Funktion	Beispiel
`int indexOf(Object e)`	Liefert den Index des übergebenen Elements zurück. Wenn nicht vorhanden, dann wird -1 zurückgeliefert.	`int tag = temperaturen.indexOf(23.3);` `int positionMercedes = autoListe.indexOf(mercedes);`
`E remove(int index)`	Entfernt das Element E an der Position index aus der ArrayList	`autoListe.remove(2);` `temperaturen.remove(5);`
`void add(int index, Object o)`	Fügt der Liste an der Position index das übergebene Objekt als neues Element hinzu	`autoListe.add(3,mercedes);` `temperaturen.add(7,23.5);`
`boolean add(Object o)`	Fügt der Liste das Objekt am Ende hinzu	`autoListe.add(fiat);` `temperaturen.add(22.7);`

Programmieraufgabe

Aufgabe 2: Verwendung der Klasse `ArrayList`:

Schreiben Sie nun Ihr zuvor erstelltes Programm „Nutzereingaben“ so um, so dass Sie statt fester Arrays nun ein Objekt der Klasse `ArrayList` nutzen, welches in der Lage ist, die Eingaben des Nutzers dynamisch zu speichern.

- Nach Eingabe einer 0 soll das Programm die eingegebenen Werte in der `ArrayList` wieder durchgehen und den Mittelwert berechnen, der anschließend auf der Konsole ausgegeben wird.

Programmieraufgabe

Aufgabe 3: Univerwaltung

Erstellen Sie in einem neuen Eclipse Projekt „Uebung10“ eine Klasse Student, die folgende Attribute und Methoden bereitstellen soll:

- **Attribute**: Name (`String`), Matrikelnummer (`int`) und eine Notenliste
  - Die Notenliste soll beim Anlegen eines Studenten leer sein und dynamisch mit Noten befüllt werden können. Verwenden Sie dazu eine passende Datenstruktur
- **Methoden**:
  - Auf alle Attribute soll direkt von außen nur lesend zugegriffen werden können. Legen Sie also eine Getter-Methode für jedes Attribut an und gestalten Sie Ihren Konstruktor so, dass die Instanzvariablen mit Werte belegt werden können.
  - Die Klasse soll zudem die Möglichkeit bieten, dass einem Studenten Noten in seine Notenliste eingetragen werden. Legen Sie eine entsprechende Methode an!
  - Außerdem soll die Klasse die aktuelle Durchschnittsnote der eingetragenen Noten berechnen und widergeben können. Legen Sie hierfür eine entsprechende Methode an!

Schreiben Sie nun ein Programm „Univerwaltung“ , welches 2 Studenten anlegt. Tragen Sie bei beiden Studenten ein paar Noten ein und lassen Sie sich jeweils die aktuelle Durchschnittsnote herausgeben.

Lassen Sie sich optional anzeigen, wie viele Noten ein Student insgesamt schon hat.

Teil 2: Weitere Aspekte der Objektorientierung

Klassenvariablen

So wie es Instanzvariablen gibt, die zu einer gewissen Instanz (Objekt) gehören und deren Attribute speichern, so gibt es aus **Klassenvariablen**:

- Speichern Werte für eine ganze Klasse
  - Sprich: Eine Änderung wirkt sich auf alle Objekte der Klasse aus!
- Beschreiben keine objektspezifischen Eigenschaften
- Werden auch als statische Attribute bezeichnet
  - Eingeleitet mit dem Schlüsselwort `static`
  - Bsp.: `static int instances;`
- Statische Attribute können, wie Instanzvariablen auch `public`, `private`, oder `protected` sein
- Auf Klassenvariablen kann (je nach Sichtbarkeit) auch zugegriffen werden, selbst wenn kein Objekt existiert
  - Über: Klassenname.klassenvariable;
  - Bsp.: `Auto.instances;`
- Zugriff über ein direktes Objekt ist auch möglich
  - Über: objektname.klassenvariable;
  - Bsp.: `fiat.instances;`
  - Sollte allerdings vermieden werden

Anwendung für Klassenvariablen

Ein bekanntes Beispiel für die Anwendung von Klassenvariablen stellt ein Instanz-Zähler dar:

```
// Beispiel für einen Instanz-Zähler über Klassenvariablen

public class Auto {

   //Klassenvariable
   public static int instances;

   //Instanzvariablen
   private String name;
   private int preis;

   //Konstruktor
   public Auto(String name, int preis){

      this.name = name;
      this.preis = preis;

      // Klassenvariable instances wird im Konstruktor erhöht
      Auto.instances++;

   }

   //Weitere Methoden…
}
```

Anwendung für Klassenvariablen

Zugriff auf die Klassenvariable:

```
public class Autohaus {

   public static void main(String[] args) {

      System.out.println("Willkommen im Autohaus!");

      System.out.println(" Es sind bereits "+Auto.instances+" Autos erzeugt worden!");

      Auto fiatPunto = new Auto("fiatPunto", 9000);

      Auto mercedes = new Auto("Mercedes",30000);

      System.out.println("Es sind bereits "+Auto.instances+" Autos erzeugt worden!");

   }

}
```

Ausgabe:

```
Willkommen im Autohaus!
Es sind bereits 0 Autos erzeugt worden!
Es sind bereits 2 Autos erzeugt worden!
```

Klassenmethoden

Neben den Klassenvariablen gibt es natürlich auch **Klassenmethoden**:

- Gehören zur Klasse und nicht zu den Objekten
  - Können daher **NUR** auf Klassenvariablen arbeiten und **NICHT** auf Instanzvariablen operieren!
  - Beschreiben also kein Objektverhalten und manipulieren daher auch keine Instanzvariablen
  - Können nur Statische Methoden aufrufen
  - Die Verwendung von `this` ist nicht möglich, da kein Objekt zugeordnet ist!

- Können direkt auf Klassen aufgerufen werden
  - auch ohne, dass ein Objekt der Klasse existiert
  - Beispiele:
    - Main-Methode: `public static void main(String[] args)`
      - Wird beim Programmstart aufgerufen, an dem noch keine Objekte existieren
    - `public static int getInstances() {return Auto.instances};`

- Das Aufrufen über Objekte ist auch möglich, aber unerwünscht!

Anwendung für Klassenmethoden

Leichte Veränderung des Beispiels von vorher:

```
// Beispiel für einen Instanzzähler über private Klassenvariablen

public class Auto {

   //Klassenvariable
   private static int instances;

   //Instanzvariablen
   private String name;
   private int preis;

   //Konstruktor
   public Auto(String name, int preis){

      this.name = name;
      this.preis = preis;

      // Klassenvariable instances wird im Konstruktor erhöht
      Auto.instances++;

   }
   //Weitere Methoden…

   // Statischer Getter für statische Variable
   public static int getInstances(){
      return Auto.instances;
   }
}
```

Anwendung für Klassenmethoden

Zugriff auf die Klassenvariable nun über den statischen Getter:

```
public class Autohaus {

   public static void main(String[] args) {

      System.out.println("Willkommen im Autohaus!");

      System.out.println(" Es sind bereits "+Auto.getInstances()+" Autos erzeugt
worden!");

      Auto fiatPunto = new Auto("fiatPunto", 9000);

      Auto mercedes = new Auto("Mercedes",30000);

      System.out.println("Es sind bereits "+Auto.getInstances()+" Autos erzeugt
worden!");

   }

}
```

Ausgabe:

```
Willkommen im Autohaus!
Es sind bereits 0 Autos erzeugt worden!
Es sind bereits 2 Autos erzeugt worden!
```

Unveränderliche Attribute (Konstanten)

Ein Anwendungsfall für statische Attribute sind Konstanten, die ihren Initialwert nicht mehr verändern können.

- Beispiele für bekannte Konstanten:
  - `Math.PI, Math.E`
- Wir können Konstanten, die eine Klasse anbieten soll selbst definieren:
  - Mit dem Schlüsselwort `final`
  - Sollten nach Konvention in GROßBUCHSTABEN deklariert werden
  - Sollten `public` sein, um direkt von Außen darauf zugreifen zu können

```
// Beispiel für eigene Konstanten

public class MeineKlasse{
   public static final int MAX_VALUE = 10;

   public static final double MEIN_WERT = 1.2;

   public static final double PI = 3.14;

   //Weitere Attribute und Methoden
}
```

```
// Konstanten benutzen

int max_wert = MeineKlasse.MAX_VALUE;

for(int i=0;i<MeineKlasse.MAX_VALUE;i++)
{
   // Do something…
}
```

Unveränderliche Attribute

Mit `final` lassen sich auch Instanzvariablen deklarieren, die nur einmal mit einem Wert belegt werden dürfen, der sich danach nicht mehr verändern lässt

- Das kann sehr nützlich sein, wenn sich Initialbelegungen von Objekten nicht mehr ändern dürfen
- Beispiel: Der Vorname eines Menschen ändert sich nicht mehr.

```
public class Mensch{

   private final String vorname;
   private String name;

   // Einmalige Belegung durch Konstruktor erlaubt
   public Mensch(String vorname, String name){
      this.vorname = vorname;
      this.name = name;
   }

   //Aber kein Setter für vorname erlaubt!
}
```

Kleine Programmieraufgabe

Nutzen Sie nun Ihre Klasse Student vom Anfang der Stunde und verändern Sie die Klasse wie folgt:

- Fügen Sie einen Instanz-Zähler ein, welcher es mittels einer Klassenvariablen und einer entsprechenden Getter-Methode erlaubt, die Anzahl der erzeugten Instanzen zu zählen und widerzugeben.
- Ändern Sie nun die Instanzvariable „matrikelnr“, so dass diese nur einmal mit einem Wert belegt werden darf.

Prüfen Sie nun in Ihrem Programm (Main-Methode) den Instanz-Zähler und lassen Sie sich die Anzahl der erzeugten Studenten ganz am Anfang und am Ende Ihres Programms ausgeben.