IT I: Heute

- Nachbetrachtung Wissensüberprüfung

- Konstanten
- *this*
- Java Klassen verwenden
- Wrapper Klassen
- Unit Test

Nachbetrachtung Wissensüberprüfung

- **Konstruktoren:**

  erschaffen Objekt, indem Attribute gesetzt werden
- ***Typischwerweise:***

  Hat der Konstruktor Parameter, so werden entsprechende Attribute auf Parameter gesetzt:

  this.blablabla = blablabla;

Nachbetrachtung Wissensüberprüfung

- **Konstruktor von ArrayList:**

  Aufruf von

  new ArrayList<Typ>()

  erschafft leere Liste:

  {} (nicht *null*!)

- Aufruf des Konstruktors ist nötig, bevor mit Methode *add* Elemente hinzugefügt werden können.

Nachbetrachtung Wissensüberprüfung

- Konstruktor mit ArrayList-Parameter

```
public Blabla(ArrayList<Typ> liste)
{
    this.liste = liste;
}
```

- Das funktioniert nur, wenn es entsprechendes Attribut *liste* gibt!

Nachbetrachtung Wissensüberprüfung

- Was passiert hier?

```
public Blabla(ArrayList<Typ> liste)
{
    this.liste = newArrayList<Typ>();
    this.liste = liste;
}
```

Nachbetrachtung Wissensüberprüfung

- Initialisierung der Liste ist unnötig:

```
public Blabla(ArrayList<Typ> liste)
{
    this.liste = new ArrayList<Typ>();
    this.liste = liste;
}
```

- Attribut wird ohnehin gesetzt.

Nachbetrachtung Wissensüberprüfung

- Was passiert hier?

```
public Blabla(ArrayList<Typ> liste)
{
    liste = new ArrayList<Typ>();
    this.liste = liste;
}
```

Nachbetrachtung Wissensüberprüfung

- Noch schlechter:

```
public Blabla(ArrayList<Typ> liste)
{
   liste = new ArrayList<Typ>();
   this.liste = liste;
}
```

- Überschreibt Parameter, bevor Attribut auf Parameterwert gesetzt.

Nachbetrachtung Wissensüberprüfung

- Was passiert hier?

```
public Blabla(ArrayList<Typ> liste)
{
    this.liste =
                new ArrayList<Typ>();
}
```

Nachbetrachtung Wissensüberprüfung

- Noch schlechter:

```
public Blabla(ArrayList<Typ> liste)
{
    this.liste =
                new ArrayList<Typ>();
}
```

- Initialisiert Attribut, Parameter nicht verwendet.

Nachbetrachtung Wissensüberprüfung

- Was passiert hier?

```
public Blabla(ArrayList<Typ> liste)
{
    ArrayList<Typ> liste =
                    new ArrayList<Typ>();
}
```

Nachbetrachtung Wissensüberprüfung

- Noch schlechter:

```
public Blabla(ArrayList<Typ> liste)
{
    ArrayList<Typ> liste =
                    new ArrayList<Typ>();
}
```

- Kompiliert nicht, weil Variable *liste* bereits existiert.

Klammern sparen

- Geschwungene Klammern können weggelassen werden, wenn sie nur ein Statement enthalten:

```
if(i==j)
    a++;
else
    a--;
```

oder

```
for(int i=0; i<10; i++)
  for(int j=10; j>0; j--)
    System.out.println(i+j);
```

Strings in Java

- Strings sind Objekte der Klasse *String* im package *java.lang*.
- Die Klassen von *java.lang* sind die einzigen Klassen, die nicht explizit importiert werden müssen.
- Vergleich von Strings mit Methode *equals*
- Besondere Syntax für Strings:
  - Automatisches Erzeugen eines String-Objektes durch Angabe eine Zeichenkette, z.B. "abc".
  - Zusammenfügen von Strings durch den Operator +, z.B. *string1 + "abc".*
  - Datentypen werden automatisch in Strings umgewandelt, wenn das nötig ist, z.B. "" + 17.

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.

Die Klasse *Math*

- Die Klasse *Math* in *java.lang* stellt ausschließlich statische Methoden zur Verfügung, die mathematische Funktionen berechnen.

Statische Attribute
Klassenattribute

- Auch Attribute können statisch (*static*) sein.
- Solche Attribute sind nicht einem Objekt sondern der Klasse zugeordnet.
- Statische Attribute können auch *public* oder *private* sein.
- Auf statische Attribute wird über den Klassennamen zugegriffen, z.B.

  System.out.
- Für die eigene Programmierung haben statische Attribute nur als Konstanten eine wichtige Bedeutung.

System.out

- In der Klasse *System* ist *out* ein statisches Attribut vom Datentyp *PrintStream* (aus *java.util*), das *public* ist.

```
public static PrintStream out;
```

- *System* ist eine Klasse in *java.lang*.

- Mit *System.out* stehen über Objektmethoden der Klasse *PrintStream* einfache Ausgabemöglichkeiten zur Verfügung.

Konstanten (1)

- Konstanten werden verwendet, um Werten einen Namen zu geben.
- Dadurch werden Programme leichter lesbar und leichter veränderbar.
- **Die Verwendung von Konstanten ist sehr empfehlenswert!**
- Beispiel: Konstante für die Kapazität des LKW.
- Konstanten sind Attribute, die mit der Bezeichnung *final* versehen werden.
- Die Bezeichnung *final* bedeutet, dass der Wert des Attributs nicht verändert werden kann, also konstant ist.
- Weiters sind Konstanten meist statisch, da sich ihr Wert nicht auf ein einzelnes Objekt bezieht.

Konstanten (2)

- Beispiel:

  ```
  private static final int LKW_KAPAZITAET = 100;
  ```

- Eine Konstante hat immer einen Datentyp und kann auch *public* oder *private* sein.
- Java-Konvention:
  - Konstantennamen werden mit Blockbuchstaben geschrieben, wobei Wort zur besseren Lesbarkeit durch "_" getrennt werden.
- In der Klasse *Math* ist z.B. die Konstante

  ```
  public static final double PI = 3.14…
  ```

  definiert, auf die mit Math.PI zugegriffen werden kann.

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.

Vordefinierte Java-Klassen

- Klassen-Bibliotheken
  - Sammlungen von nützlichen Klassen, die nicht selbst programmiert werden müssen.
  - In Java sind diese in *packages* organisiert.
  - Für die Java-Programmierung muss man einige wichtige dieser Klassen kennen.
  - Man muss die Klassen-Dokumentation (*Class API*) lesen können, um sich über weiter Klassen informieren zu können.
  - API: Applications Programmers Interface

Lesen der Klassen-Dokumentation

- Die Dokumentation beschreibt in der Regel nur die Verwendung der Klassen, nicht wie die Klassen implementiert sind.
- Die Dokumentation liegt in HTML-Format vor.
- Eine Klassen-Dokumentation enthält:
  - Name der Klasse
  - Allgemeine Beschreibung der Klasse
  - Liste der Konstruktoren und Methoden mit Parametern und Rückgabewerten
  - Eine Beschreibung für jeden Konstruktor und jede Methode:
    - Was bewirkt der Konstruktor oder die Methode?

Information Hiding

- Die Klassen-Dokumentation enthält keine Implementierungsdetails:
  - Attribute und Methoden, die *private* sind.
  - Die Implementierung der Methoden.

Verwendung von Bibliotheksklassen

- Klassen aus einem Java *package* müssen importiert werden, und können dann wie eine Klasse des eigenen Projektes/Paketes verwendet werden:

  ```
  import <Paketname>.<Klassenname>;
  ```

- Z.B.:

  ```
  import java.util.Random;
  ```

  Dabei ist *java.util* das Paket, das die Klasse *Random* enthält. (Diese Klasse ist für aktuelle Übungsaufgaben nützlich!)

Die Klasse *Random*

- Die Klasse *Random* stellt verschiedene Methoden zur Erzeugung von Zufallszahlen zur Verfügung.
- Mit Konstruktor

  new Random()

  wird ein Zufallszahlengenerator erstellt.
- Mit Methoden

  double nextDouble()

  int nextInt()

  ...

  werden dann Zufallszahlen erzeugt.

Die Klasse *Random*

Drei Möglichkeiten, wo/wie Random-Objekt angelegt wird:

- lokal
- als nichtstatisches Attribut
- als statisches Attribut

Die Klasse *Random*

Drei Möglichkeiten, wo/wie Random-Objekt angelegt wird:

- lokal
- als nichtstatisches Attribut
- als statisches Attribut

Letzteres ist beste Variante!

BlueJ Class Template

Das BlueJ Class Template File findet man unter:

BlueJ/lib/german/templates/newclass/stdclass.tmpl

Wrapper classes

- Primitive types (int, long, double, …) are not objects.
- They must be wrapped into an object to be put into an ArrayList.
- Wrapper classes exist for all primitive types:

primitive type	*wrapper class*
int	Integer
long	Long
double	Double
…	…

Autoboxing und Unboxing

- Die Konvertierung von primitiven Datentypen und Objekten der Wrapper-Klassen erfolgt in Java (fast immer) automatisch.

```
// Integer n = new Integer(42);
Integer n = 42;


// int i = n.intValue();
int i = n;
```

Autoboxing and unboxing

```
private ArrayList<Integer> markList;

…

public void storeMark(int mark)

{

    markList.add(mark);

}
```

autoboxing

```
int firstMark = markList.remove(0);
```

unboxing

Spezialfall zum Autoboxing & Unboxing

```
private ArrayList<Integer> list;

    void storeIntegers(){

        list.add(42);

        list.add(43);

    }

    void removeAndPrint(){

        list.remove(1);

        for(Integer i : list){

            System.out.println(i);

    }

}
```

Weitere Eigenschaften von Wrapper-Klassen

- Wrapper-Klassen stellen weitere Funktionalitäten bereit:
  - Enthalten Konstanten
  - static Methoden zur Konvertierung, ...

```
Integer.MAX_VALUE =  2^31-1
Integer.MIN_VALUE = -2^31
Double.MAX_VALUE  = (2-2^-52)·2^1023
Double.MIN_VALUE  =  2^-1074
Double.POSITIVE_INFINITY
Double.NEGATIVE_INFINITY
```

Schleifen und ArrayLists

- Objekte in ArrayList können innerhalb eines Schleifendurchlaufs geändert werden.

- ***Problemlos:*** aktuelle Elemente ändern oder ersetzen
- ***Problematisch:*** aktuelle Elemente löschen oder Elemente hinzufügen

Schleifen und ArrayList

Löschen/Hinzufügen von Elementen während Durchlaufs einer ArrayList:

- ***for-each:*** geht nicht (Exception!)
- ***for****-Schleife:* geht, aber muss Indizes entsprechend ändern (oder Liste von hinten nach vorne durchgehen).

Unit Test

- Neben den bisher vorgestellten Möglichkeiten zum Testen von Programmen, gibt es in Java eigene *Unit Test* Klassen.
- Diese erlauben das automatische Anlegen von Objekten.
- In *BlueJ* können die in der *Unit Test* Klasse angelegten Objekte in die Objektleiste kopiert werden bzw. umgekehrt.
- Testfälle können durch Objektinteraktion festgelegt werden.