# IT-Basics 2 Handout 9.9.2008 u. 16.9.2008

Für das Erstellen von objektorientierten Programmen ist es neben dem Verständnis der Objektorientierung auch nötig, einige grundlegende Konzepte der Programmierung zu verstehen. Diese grundlegenden Konzepte sind Thema der zweiten Lehrveranstaltung und teilweise auch der dritten Lehrveranstaltung.

Einen guten und ausführlichen Überblick zu den Themen gibt das Kapitel 2 in http://openbook.galileocomputing.de/visual_csharp/

## Variablen

Variablen bieten die Möglichkeit einzelne Werte zu speichern. Sie müssen deklariert und initialisiert werden. Bei der Dekleration wird der Speicherbereich für die Variable reserviert. Es gibt Syntaxkonventionen wie Namen von Variablen aufgebaut sein müssen. Sie müssen alphanumerisch sein, dürfen keine Sonderzeichen (#, §, $) beinhalten; das Zeichen _ ist allerdings erlaubt. Sie müssen mit Buchstaben oder _ anfangen. Nur _ ist nicht erlaubt. Sie müssen eindeutig sein (es darf also keine zwei gleichen Variablennamen geben) und sie dürfen nicht wie ein Schlüsselwort benannt werden.

## Datentypen

Variablen können Zahlen, Zeichenketten oder logische Ausdrücke beinhalten. Datentypen haben einen bestimmten Wertebereich. Wird dieser überschritten (durch z.B. eine mathematische Operation) kommt es zu einem Fehler und das Programm stürzt u.U. ab. Die Einhaltung der Wertebereiche muss sichergestellt werden. Die wichtigsten Datentypen mit Ihrem Wertebereich sind:

- byte :0-255
- int :$-2^{31}$ … $2^{31}-1$
- float: $1{,}4 * 10^{-45}$ bis $3{,}4 * 10^{38}$
- doule: $5{,}0 * 10^{-324}$ bis $1{,}7 * 10^{308}$
- string: Zeichenkette
- boolean: true/false

Datentypen können unter Einhaltung des Wertebereiches implizit (automatisch) konvertiert werden weil der Wertebereich dabei nicht verletzt werden kann: byte -> int -> float -> double. In eine andere Richtung kann man nur dann zuweisen wenn man dies explizit angibt. Das kann man durch Angabe des Typs in (...) .

```
int a = 10;
double c = a;
a = (int)c
```

Man kann auch Convert.<Methode> benutzen. Damit ist es möglich z.B. eine Zeichenkette in einen int zu wandeln.

```
string a = "10";
int b = 0
b = Convert.ToInt32(a)
```

## Felder

Will man mehrere Werte verwalten so bieten sich vorerst Felder (Arrays) an. Ein Feld hat eine fixe Größe, die bei der Initialisierung angegeben wird. Bei der Dekleration gibt man [] an um zu markieren, dass dies ein Feld ist.

```
int[] feld;
feld = new feld[3];
```

Nun kann man auf die einzelnen Werte im Feld über einen fortlaufenden Index zugreifen der mit 0 beginnt.

```
feld[1] = 3;
```

Man kann die Dekleration und die Initialisierung auch in einer Zeile schreiben wobei Werte die in {} geschrieben werden direkt zugewiesen werden. Man kann entweder die Größe des Arrays angeben oder durch die Anzahl der Werte in {} bestimmen lassen.

```
int[] feld1 = new int[3]{1,2,3};
int[] feld2 = new int[]{1,2,3};
```

Felder können für alle Datentype benutzt werden.

## Bedingungen

Bedingte Verzweigungen dienen dazu, den Ablauf in einem Programm zu beeinflussen. Zentral dabei sind boolsche Ausdrücke über die entschieden wird, welcher Zweig durchlaufen wird.

Das if-Konstrukt bietet die Möglichkeit anhand eines boolschen Ausdruckes zu entscheiden welcher Codeabschnitt benutzt wird.

```
if ( boolscher-Ausdruck )
{..}
else
{..}
```

Muss man zwischen mehreren Werten unterscheiden kann man das switch-Konstrukt benutzen.

```
switch (variable)
{
  case Wert1:
   ...
   break;

  case Wert2:
   ...
   break;

  default:
   ...
  break;
}
```

Jede, case Abschnitt indem ein Programmcode steht muss mit einem break beendet werden. Es wird von oben beginnend die Werte ausgewertet. Sollte kein Wert übereinstimmen, so wird der default Zweig ausgewählt.

## Schleifen

In manchen Fällen ist es notwendig nicht nur verschiedene Zweige der Verarbeitung auszuwählen sondern gewisse Abschnitte im Quellcode zu wiederholen. Dazu gibt es drei verschiedenen Arten von Schleifen.

Die for-Schleife benutzt man dann, wenn man von vornherein weiß wie oft man einen Teil wiederholen möchte.

```
for (Start;Bedingung;Increment)
{...}
```

Start dient dazu die Schleife zu initialisieren und wird nur am Anfang ausgeführt. Die Schleife wird solange wiederholt, solange die angegebene Bedingung (boolscher Ausdruck) erfüllt ist. Das wird vor jedem Schleifendurchlauf geprüft. Increment wird oft dazu verwendet den Schleifenzähler zu erhöhen. Das wir nach jedem Durchlauf ausgeführt.

```
for(int index=0;index<10;index++)
{
  Console.WriteLine(index);
}
```

Möchte man z.B. Feld (später werden wir noch andere Datenstrukturen kennen lernen) durchlaufen so bietet sich die foreach-Schleife an. Dabei werden alle Werte in dem Feld durchlaufen und stehen in der Schleife als Variable zu Verfügung.

```
foreach (type var1 in var2)
{...}
```

Type definiert den benutzten Datentyp, var1 die Variable die den einzelnen Wert im Feld beinhaltet und var2 ist die Variable die das Feld beinhaltet.

```
int[] feld2 = new int[]{1,2,3};
for(int wert in feld2)
{
   Console.WriteLine(wert);
}
```

Die letzte der für uns wichtigen Schleifen ist die while Schleife. Diese wird benutzt wenn man die Wiederholung des Abschnittes an eine allgemeine Bedingung binden möchte und nicht genau weiss wie oft die Schleife wiederholt werden soll.

```
while (Bedingung)
{...}
```

Solange die Bedingung erfüllt ist wird die Schleife wiederholt.

## Übungsaufgaben

1. Erstellen Sie eine Anwendung die den arithmetischen Mittelwert eines Arrays bestehend aus int Werten berechnet und auf der Console ausgibt.
2. Erstellen Sie eine Anwendung die größte und die Kleinste Zahl in einem Array bestehend aus int Werten sucht und auf der Console ausgibt.
3. Erstellen Sie eine Anwendung die überprüft ob in einem Array bestehend aus int Werten mindesten ein Werte mehr als einmal vorkommt. Sie soll ausgeben ob und welcher Wert mehr als einmal vorkommt.
4. Schreiben Sie eine Anwendung die den größten Teiler einer Zahl ausgibt. Benutzen Sie dabei z.B. den % Operator.

## Klassen und Objekte

Im Laufe der Vorlesung werden wir uns noch ausführlich mit den Begriffen Klasse und Objekt beschäftigen. Die erste Auseinandersetzung mit den Begriffen ist an real existierende Dinge angelehnt, da diese bekannt und deswegen auch leichter analysierbar sind. Wir gehen am Anfang auch noch nicht auf alle Details ein sondern zuerst beschäftigen wir uns mit dem Wesen von Klassen und Objekten.

Eine Klasse kann wie ein Bauplan, eine Vorlage verstanden werden. Sie beschreibt welche Eigenschaften etwas hat und welche Aktionen/Vorgänge etwas durchführen kann. Die Eigenschaften werden in Variablen der Klasse beschrieben die Aktionen in Methoden. Ein Auto hat z.B eine Farbe, eine aktuelle Geschwindigkeit usw. Das wären die Eigenschaften. Es kann beschleunigen, abbiegen, bremsen usw. Das wären die Aktionen. Diese haben immer eine Auswirkung auf die Eigenschaften. Beim Beschleunigen verändert sich z.B. die aktuelle Geschwindigkeit.

Aus einer Klasse kann man nun ein konkretes Objekt machen. Das Objekt hat nun konkrete Werte für z.B. die Geschwindigkeit oder die Farbe. Hat man z.B. ein Auto modelliert dann kann man ein Objekt erstellen das ein rotes Auto mit der aktuellen Geschwindigkeit 0 beschreibt.

Manchmal sollen die Eigenschaften einer Klasse nicht veränderbar sein aber abgefragt werden können z.B. die Matrikelnummer eines Studierenden. In diesem Fall muss man sicher stellen, dass beim Erstellen des Objektes diese Werte gesetzt werden und von außen nur lesbar sind.

## Klassen in C#

All diese Aspekte von Klassen lassen sich in C# realisieren. Die Eigenschaften kann man mit Variablen die innerhalb der Klasse deklariert werden umsetzten wie im Beispiel die Variablen breite und hoehe.

Die Aktionen sind in Methoden Umgesetzt wie im Beispiel die Methode berechneFlaeche() und das Initialisieren von Variablen beim erstellen der Klasse ist über den Konstruktor gelöst. Der Konstruktor ist eine spezielle Methode die beim Erstellen eines Objektes aufgerufen wird. Er heißt immer gleich wie die Klasse.

```
class Rechteck
{
  private double breite;
  private double hoehe;

  public Rechteck(double h, double b)
  {
    breite = b;
    hoehe = h;
   }

  public double berechneFlaeche()
  {
   return breite* hoehe;
  }

  public void skalieren(double f)
  {
    breite = breite * f;
    hoehe = hoehe * f;
  }
}
```

Gehen wir nun kurz auf die einzelnen Teile noch einmal genauer ein. Die Variablen breite und hoehe sind als privat markiert, das heißt von außen sind diese nicht sichtbar. Sie können aber in Methoden der Klasse (wie z.B. berechneFlaeche) benutzt werden. Neben private gibt es auch public. Da diese Schlüsselworte den Zugriff steuern, gehören sie in die Gruppe der Access-Modifier.

Methoden bilden die Aktionen einer Klasse ab und ihre Definition beginnt auch mit einem Access-Modifier. Danach muss angegeben werden ob die Methode ein Ergebnis zurück liefert (Datentyp) oder nicht (void). Wenn ein Wert zurückgegeben wird so benutzt man das Schlüsselwort return. Die Methode muss weiters einen Namen bekommen (z.B. berechneFlaeche). In runden Klammern werden Parameter definiert die an die Methode übergeben werden können. Das muss aber nicht sein. Es ist oft der Fall, dass keine Parameter übergeben werden. Will man aber Werte übergeben (siehe skaliere(double f)) muss man zuerst den Datentyp angeben und dann den Namen der Variablen die innerhalb der Methode den Wert beinhalten soll.

## Objekte und Referenzen in C#

Um nun aus einer Klasse ein konkretes Objekt zu erzeugen benötigt man zuerst eine Variable die von Typ der Klasse ist und danach muss man ein neues Objekt mit dem Schlüsselwort new erzeugen. Dabei wird der Konstruktor aufgerufen und die Werte für die Initialisierung des Objektes übergeben.

```
Rechteck r1; // Dekleration
r1= new Rechteck(10.0, 20.0);
Rechteck r2:
r2=r1;
```

r1 und r2 werden als Referenzen bezeichnet. Es kann sein, dass zwei Referenzvariablen auf ein und das selbe Objekt zeigen. Im Beispiel wird z.B. nur einmal ein Objekt mit new angelegt aber r1 und r2 zeigen auf das selbe Objekt. Verändert man dieses z.B. durch den Aufruf von skaliere, so können die Auswirkungen sowohl über r1 als auch über r2 beobachtet werden.

Referenzen können natürlich auch wie andere Variablen in Feldern und auch in foreach Schleifen benutzt werden.

```
Rechteck[] rechtecke = new Rechteck[] { new Rechteck(2,3),
                                        new Rechteck(3,4)};

 foreach (Rechteck r in rechtecke)
 {
   Console.WriteLine(r.berechneFlaeche());
 }
```

## Übungsbeispiele

1. Implementieren Sie eine Klase Student. Die Matrikelnummer soll beim Instanzieren gesetzt werden, Vor- und Nachname über eigene Methoden. Schreiben Sie auch eine Methode, die Informationen zum Studenten als String zurück gibt (Matrikelnummer: Vorname Nachname).
2. Implementieren Sie eine Klasse Bestellzeile die einen Produktnamen , einen Produktpreis und eine Anzahl verwalten kann. Sie soll auch den Summenpreis und Informationen zur Bestellzeile (Name, Anzahl, Preis, Summe) ausgeben können. Erstellen Sie ein Feld von solchen Bestellzeilen und berechnen Sie die Gesamtsumme aller Bestellzeilen.