## Erste Schritte mit C# und Sharp Develop

Allgemeines zu C# und .NET

C# (lies: „c sharp“) ist eine recht junge Programmiersprache. Sie wurde von Microsoft im Rahmen seiner .NET (lies: „dot net“) Initiative vor einigen Jahren entwickelt. C# ähnelt einigen anderen Programmiersprachen, wie z.B. C, C++ und Java, und hat viele Elemente von ihnen übernommen. Deshalb kann man mit C# Kenntnissen leicht auf diese Sprachen umsteigen. Dasselbe gilt natürlich auch für die umgekehrte Richtung.

C# gibt es nicht nur für Windows. Da Microsoft die Sprache standardisieren ließ, gibt es C# auch für andere Betriebssysteme, u.a. im Mono-Projekt für Linux und MacOS.

Der Begriff .NET begegnet dem Besucher auf vielen Webseiten. Für uns im Informatikunterricht bedeutet .NET eine Plattform zum Programmieren, mit der man leicht Windows- und Web-Programme erstellen kann, ohne dass wir uns mit historisch gewachsenen Problemen quälen. .NET ist objektorientiert und besteht aus vielen Klassen, die dem Programmierer die Arbeit erleichtern. Man spricht auch vom „.NET Framework“. Aktuell ist Version 4.6.2. Für unsere Zwecke reicht eigentlich schon die Version 2.0 völlig aus.

Um .NET Programme ausführen zu können, muss man das .NET Framework insbesondere auf älteren Computern installieren. Dies ist in der Regel recht unproblematisch. Zusätzlich wird für Programmierer noch das sogenannte .NET Framework SDK (Software Development Kit) angeboten, das weitere Tools, die Dokumentation der Klassen und Beispiele enthält. Wir können im Kurs auf das .NET Framework SDK verzichten, da viele Informationen im SDK zu weit führen würden.

Wir werden für die Entwicklung von Programmen die kostenlose Entwicklungsumgebung Sharp Develop (#D) verwenden, die recht einfach zu bedienen ist.

Hinweise zur Installation und Einrichtung der Programmierumgebung gibt es auf einer eigenen Seite auf der Homepage des Kurses.

Grundlagen

Um ein C#-Programm zu erstellen, muss man zunächst seinen **Quelltext** schreiben. Das ist eine Textdatei mit der Dateiendung .cs, speziellen Befehlen und dem folgenden prinzipiellen Aufbau:

```
public class <<Name der Klasse>>{

  << Allgemeine (Globale) Definitionen >>

  public static void Main(string[] args){

     << Hier beginnt das eigentliche Programm >>
  }
}
```

C#-Quelltexte sind normale Textdateien und können damit mit jedem beliebigen Texteditor erstellt werden. Als sinnvoll haben sich aber Editoren erwiesen, die die C#-Befehle hervorheben, da man dann besser im Quelltext arbeiten kann.

Um einen C#-Quelltext in ein Programm umzuwandeln, muss man ihn **kompilieren**. Dies geschieht mit dem Programm „**csc.exe“**. Es erzeugt aus der Textdatei eine Binärdatei mit der Dateiendung .exe, die das eigentliche Programm darstellt.

Obwohl die Datei die Endung .exe hat, ist sie kein normales ausführbares Programm. Damit das Programm funktionieren kann, ist das .NET Framework notwendig.

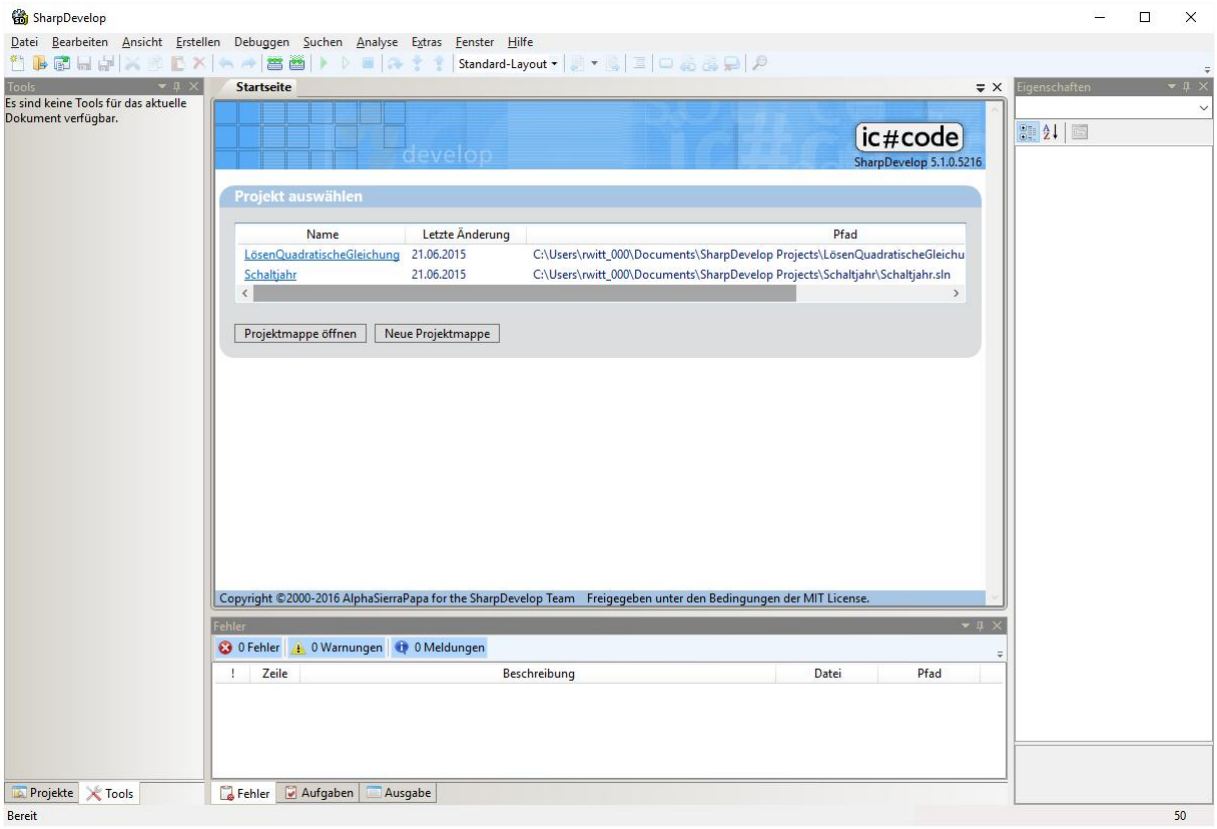
Wir brauchen uns um den genauen Ablauf und die verwendeten Programme nicht zu kümmern, da uns SharpDevelop die Arbeit abnimmt und Menüpunkte dafür zur Verfügung stellt.

Man muss aber folgende Reihenfolge beachten:

1. Quelltext schreiben (und speichern)
2. Quelltext kompilieren -> Binärdatei: Programm
3. Programm starten/ausführen

## Das erste Programm mit Sharp Develop

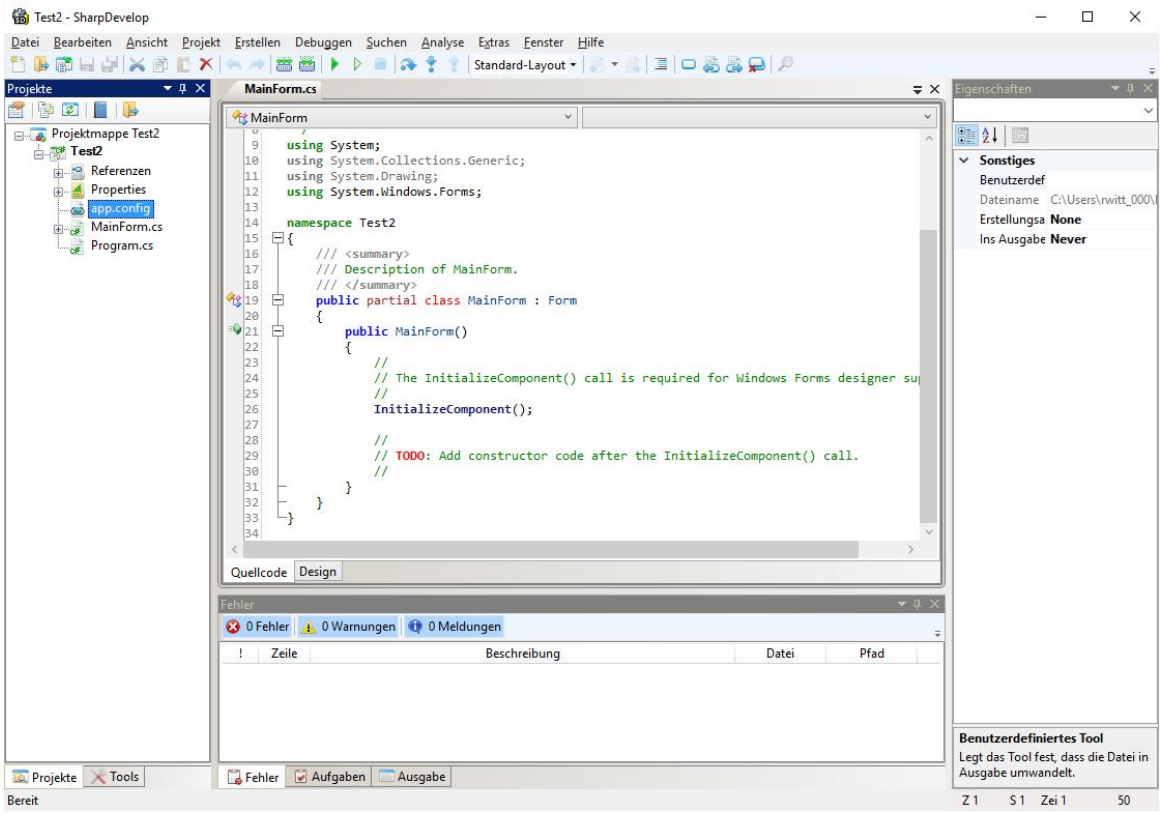
Wenn man Sharp Develop startet, erhält man einen Bildschirm, der so ähnlich wie der folgende aussieht:




Auf den ersten Blick sieht der Bildschirm verwirrend aus, aber wenn man sich daran gewöhnt hat, ist es nicht mehr ganz so schlimm.

In der Mitte befindet sich eine Liste der Projekte, die zuletzt bearbeitet wurden. Ein neues Programm erstellt man, indem man auf die Schaltfläche „Neue Projektmappe“ klickt. Den Quelltext eines schon vorhandenen Programms öffnet man, indem man auf „Projektmappe öffnen“ klickt oder das Programm aus der Liste über den Schaltflächen auswählt.

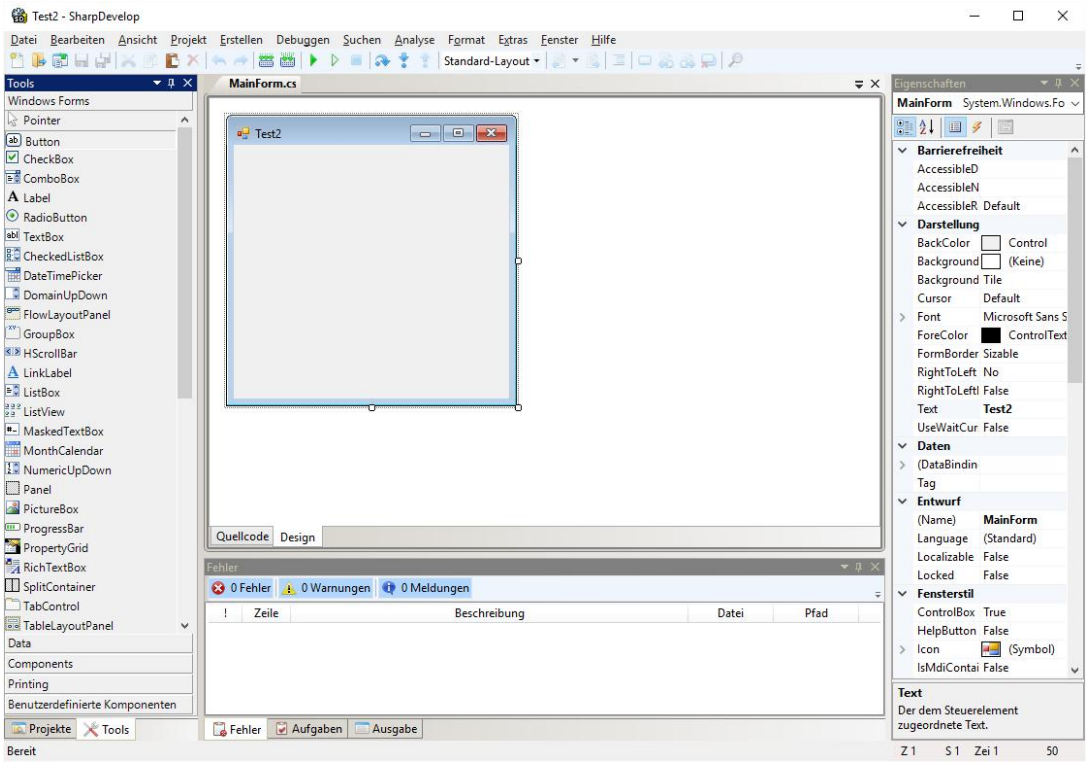
Wenn man auf „Neue Projektmappe“ klickt, gelangt man zu einem Dialog, indem man links bei den Kategorien C#, dann Windowsanwendungen wählt und dann auf der rechten Seite bei Schablonen „Windows-Anwendung“ auswählt. Weiter unten den gibt man den Namen des Programms (und der Projektmappe) an. **Im Namen dürfen nur Buchstaben, Zahlen, Unterstriche („_“), Leerzeichen und Punkte vorkommen.** Wenn man alles eingegeben hat, klickt man schließlich auf „Erstellen“ und gelangt zum Hauptbildschirm von Sharp Develop:




Hier ist wiederum der Bereich in der Mitte wichtig, indem man den Quelltext des Programms bearbeitet.

Mit Hilfe der Reiter „Quellcode“ und „Design“ kann man zwischen dem Quelltext und dem grafischen Designer für sogenannte Forms (Dialoge, Formulare) umschalten.

Um zu den Komponenten (spezielle Objekte) zu gelangen, die man auf das Formular positionieren kann, muss man sich im Design-Modus befinden und damit in der Mitte ein Formular sehen. Aus den Registerkarten, die sich ganz unten links befinden, muss man „Tools“ ausgewählt haben. Rechts unten sollte man die Eigenschaften-Registerkarte aktiviert haben.



Im Bereich „Tools“ links kann man auf „Windows Forms“ klicken. Dort erhält man eine Liste, aus der man die Komponenten auf das Formular ziehen kann. Auf der rechten Seite sieht man jeweils die Eigenschaften der gerade ausgewählten Komponente.

Um das erste Programm zu erstellen, ziehen wir einen Button auf das Formular. Rechts suchen wir die Eigenschaft „Text“ und ändern den Wert der Eigenschaft in „drück mich!“
Nun kompilieren bzw. erstellen wir das Programm, indem wir oben auf der Symbolleiste auf  klicken. Im Bereich „Ausgabe“ (in der Mitte unten auf dem Bildschirm) erscheinen Meldungen, die man, wenn es Probleme gibt, näher untersuchen muss. Anschließend starten wir das Programm mittels  . Nach einigen Momenten taucht das Programm auf. Wenn wir aber auf den Button klicken, passiert nichts. Wenn man auf den Button drückt, wird ein sogenanntes **Ereignis (Event)** ausgelöst. Wir müssen auf dieses Ereignis reagieren, indem wir eine **Ereignisbehandlung (Event Handler)** erstellen. Dazu beenden wir das Programm wieder und klicken **in Sharp Develop** doppelt auf den Button. Wir gelangen wieder in den Quelltexteditor und sehen:

```
void Button1Click(object sender, EventArgs e)
{

}
```

Die geschweiften Klammern begrenzen Blöcke in C#. In diesem Fall begrenzen sie die Ereignisbehandlung. Zwischen den Klammern gibt man `MessageBox.Show("Hallo Welt");` ein. Wenn man den Punkt nach `MessageBox` eingibt, taucht eine Liste mit den Eigenschaften und Methoden des angesprochenen Objektes bzw. der angesprochenen Klasse auf. Diese Liste hilft, die passende Methode oder Eigenschaft zu finden.

Ganz wichtig ist auch, dass man den Text so eingibt, wie oben angegeben, da C# Groß- und Kleinschreibung unterscheidet.

Folgendes sollte jetzt als Ereignisbehandlung zu sehen sein:

```
void Button1Click(object sender, EventArgs e)
{
  MessageBox.Show("Hallo Welt");
}
```

Wenn man jetzt den Quelltext kompiliert und das Programm ausführt, taucht ein neues Fenster mit der Meldung „Hallo Welt“ auf, wenn der Button gedrückt wird.

Aufgabe 1:
Ändere den Text zwischen den Anführungszeichen und teste das neue Programm.

Schon an diesem kleinen Beispiel erkennt man einige wichtige Sachverhalte:

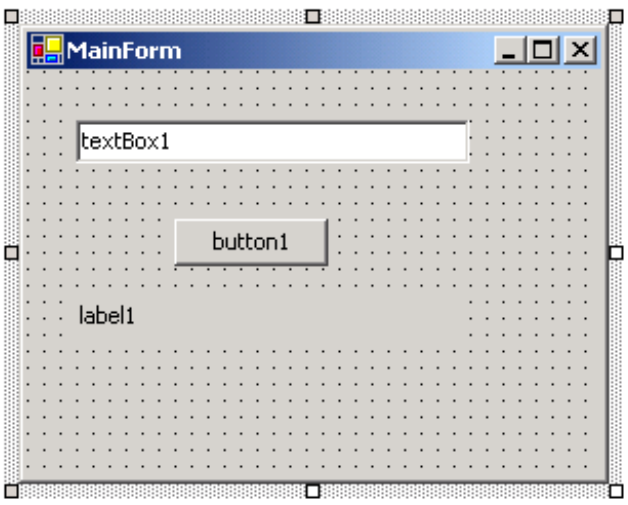
- Jeder Befehl muss in C# mit einem Strichpunkt `;` enden.
- Auf Methoden und Eigenschaften eines Objektes/einer Klasse greift man mit Hilfe des Punktes zu.
- Meldungsfenster kann man Hilfe von „MessageBox.Show(...)“ anzeigen. Den Text, den man ausgeben will, schreibt man innerhalb der Klammern in Anführungszeichen. Dieser Text wird unverändert ausgegeben.
- C# erwartet nicht einen Befehl pro Zeile. Trotzdem sollte man seinen Quelltext sinnvoll strukturieren. Eine solche Möglichkeit ist im Beispiel dargestellt. Wie so oft gilt also auch hier: *Weniger* (Ausdrücke pro Zeile) *ist mehr* (Übersichtlichkeit).
  Nach einer geschweiften Klammer { sollte um 4 Stellen nach rechts eingerückt werden, was mit der Tabulator-Taste gut funktioniert nicht.

## *Zuweisungen*

Beispiel: Ausgabe eines Textes in einem Label

Erstelle ein neues Programm. Wähle den grafischen Formulardesigner aus. Ziehe einen Button, ein Label und eine TextBox auf das Formular und ordne sie sinnvoll an, z. B.:




Ändere den Text von button1 in „zeige“ und erzeuge wieder eine Ereignisbehandlung. Trage folgendes ein:

```
label1.Text = textBox1.Text;
```

Damit wird der Eigenschaft Text von label1 der Wert der Eigenschaft Text des Objektes textbox1 zugewiesen.

Das Gleichheitszeichen in der Programmierung mit C# hat nicht die aus dem Alltag (bzw. aus der Mathematik) gewohnte Bedeutung. Das Gleichheitszeichen hat hier eine ähnliche Bedeutung wie bei Zahlenfolgen. Die Eigenschaft auf der linken Seite erhält den Wert des Ausdrucks auf der rechten Seite.

Man spricht in der Informatik von einer **Zuweisung**. Bedeutend bei einer Zuweisung ist, dass immer zunächst der Ausdruck auf der rechten Seite berechnet wird und erst dann das Ergebnis auf die linke Seite übertragen wird.

Teste das Programm.

*Hinweis:* Normalerweise sollte man vor einer TextBox ein Label setzen, in dem steht, was man in der TextBox eingeben soll. Die TextBox kann einen sinnvollen Anfangswert enthalten.
Zum Beispiel steht dann im Label (bzw. deren Eigenschaft Text) „auszugebender Text“ und die TextBox bleibt leer. Bitte nicht den Hinweis, was eingegeben werden soll, in die TextBox schreiben, auch wenn das manche anderen Anwendungen tun.

Aufgabe 2:

Wie kann ich dafür sorgen, dass im Label der Text nur in Großbuchstaben erscheint?
*Tipp:* Tippe hinter `textBox1.Text` einen Punkt und untersuche die angezeigten Methoden und Eigenschaften.

## *Variablen*

Variablen sind Behälter, in denen man verschiedene Werte speichern kann. Im Gegensatz zur Mathematik muss man in C# sofort angeben, welche Werte man überhaupt in der Variable ablegen will.

Wenn man ganze Zahlen in einer Variablen speichern will, hat die Variable den **Datentyp** `int`. Dezimalzahlen, genauer gesagt Fließkommazahlen, werden in `double` Variablen gespeichert. Zeichenketten werden im Datentyp `String` gespeichert.

Jede Variable hat außerdem einen Namen, der aus folgenden Elementen bestehen kann:

- den *Buchstaben* des Alphabets (d.h. a,b,c,...). C# unterscheidet hierbei zwischen Groß- und Kleinschreibung.
- dem *Unterstrich* `"_"`
- den *Ziffern* `0,1,2,...,9`

Variablen dürfen außerdem nicht so heißen, wie Befehle in C# und sie dürfen nicht mit einer Ziffer beginnen.

Beispiel zum Arbeiten mit Variablen:

Erstelle ein neues Programm. Ziehe einen Button und zwei Labels auf das Formular. Erzeuge eine Ereignisbehandlung für den Button und fülle sie mit folgendem Quelltext:

```
int x;
x=3+4;
label1.Text = x.ToString();
label2.Text = "x = "+x.ToString();
```

Aufgabe 3:

Teste das Programm. Was wird angezeigt?

Das Programm enthält viele neue Elemente:

- Durch `int x;` wird eine Variable mit dem Namen x deklariert und damit Speicher reserviert.
- `x=3+4;` ist eine Zuweisung. In der Variablen x wird der Wert des Ausdrucks 3+4 (d. h. 7) gespeichert.
- Jede Eigenschaft hat einen Datentyp. Die Eigenschaft Text ist eine Zeichenkette (ein String). Deshalb muss man die Zahl in eine Zeichenkette umwandeln, um sie im Label ausgeben. Dies geschieht mit Hilfe der Methode ToString() von x.
- `"x = "+x.ToString();` demonstriert, wie man Text und eine Variable gemeinsam ausgeben kann. Der Text zwischen den Anführungszeichen wird wieder unverändert ausgegeben.

Aufgabe 4:

Ändere das Programm, so dass die Variable x andere Werte enthält. Verändere auch die Ausgabe.

Aufgabe 5:

Welche der folgenden Bezeichner sind in C# erlaubt? Warum sind die anderen nicht erlaubt?

```
Hallo_Welt, 10kleineSchokoküsse, _H_A_L_L_O_ , Das_war's,
hallo123, hallo_123, Hallo Welt, class
```

Aufgabe 6:

Schreibe ein C#-Programm, das die Zahlen 12 und 9 addiert, subtrahiert, multipliziert und dividiert. Dabei sollen die beiden Zahlen und das jeweilige Ergebnis in einer passenden Variablen gespeichert und diese jeweils mit Hilfe von Labels ausgegeben werden.