1 Variablen

Variablen werden verwendet um Informationen zu speichern.
Sie können global oder lokal deklariert werden. Bsp folgt.

1.1 Typen

Variablen haben Typen, diese bestimmen die Art von Information, welche in der Variable gespeichert werden kann.

Typ	Information	Bsp
int	ganze Zahlen	25; -312
double	gebrochene zahlen	2.44; 0.333
bool	Wahrheitswert	True; False
char	Zeichen	'A'; '?'
string	Zeichenkette	"Hallo Welt!"

1.2 Deklaration

Bevor Variablen verwendet werden können müssen sie deklariert werden. Gleichzeitig können sie initialisiert werden, d.h. der Variablen einen Anfangswert zuweisen.

```
int a; //Deklaration einer Variable vom Typ int
double x,y,z; //Mehrere Variablen von Typ double gleichzeitig deklarieren
string satz = "Hallo Welt!" //Deklaration und Initialisierung einer Variablen vom Typ string
```

1.3 Zuweisung

Variablen können nachdem sie deklariert wurden Werte zugewiesen werden:

```
int a,b,c;
a = 25;
b = -12;
c = a + b;
```

Hierbei spielt die Richtung eine Rolle: (Variable) a (Zuweisung) = (Wert) 25.

1.4 Umwandlung

Um Werte in variablen zu speichern muss der Typ des Wertes mit dem Typ der variablen übereinstimmen.

```
string zahl = "25";
int a=12, b;
b = a * Convert.ToInt32(zahl);
//zahl wird in int umgewandelt um die Multiplikation ausführen zu können.
```

Beispiele für Umwandlungen:

Umwandlung	Ergebnistyp
`Convert.ToInt32()`	int
`Convert.ToDouble()`	double
`Convert.ToString()`	string

Insbesondere bei der Umwandlung zwischen char und int Variablen ist zu beachten, dass diese Umwandlung nach der ASCII-Tabelle stattfindet.

Abbildung 1: ASCII Tabelle

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

```
int b;
char d;
d = Convert.ToChar(43);
b = Convert.ToInt32('33');
// d beinhalted dann das Zeichen '+'
// b beinhalted dann die Zahl 70
```

2 Ein und Ausgabe in der Konsole

Werte können aus der Konsole gelesen und in die Konsole geschrieben werden.

2.1 Eingabe

Einlesen von Informationen `Console.ReadLine()` liest die nächste Eingabe bis zum drücken der Enter-Taste als Typ string ein.

```
int a;
string s;
s = Console.ReadLine();//Eingabe einlesen
a = Convert.ToInt32(s);//eingelesene Eingabe in Typ int umwandeln
```

2.2 Ausgabe

Ausgeben von Informationen `Console.WriteLine(x);` gibt den Inhalt von x in der Konsole aus.

```
int a = 32;
double u = 2.5;
//Variante 1
Console.WriteLine("ganze Zahl " + a + " und gebrochene Zahl " + u );
//Variante 2
Console.WriteLine("ganze Zahl {0} und gebrochene Zahl {1}", a, u );
```

Beide Varianten liefern die selbe Ausgabe:

```
ganze Zahl 32 und gebrochene Zahl 2.5
```

3 Konditional

Um Bedingungen zu überprüfen kann `if - else` verwendet werden.

```
if(Bedingung1)
{
      Code;//wird ausgeführt wenn Bedingung1 wahr ist
}
else if(Bedingung2)
{
      Code;//wird ausgeführt wenn Bedingung2 wahr ist
}
else
{
      Code;//wird ausgeführt wenn keine Bedingung wahr ist
}
```

Ähnliche Funktionalität bietet die switch-case Umgebung:

```
switch (Bedingung)
{
case bedingung1:
        Code Code;//wird ausgeführt wenn Bedingung==bedingung1 wahr ist
        break;
case bedingung2:
    Code;//wird ausgeführt wenn Bedingung==bedingung2 wahr ist
        break;
default:
    Code;//wird ausgeführt wenn keine Bedingung übereinstimmt
        break;
}
```

3.1 Vergleichsoperatoren

Vergleichsoperator	Bedeutung	Beschreibung
>=	größer gleich	linke Seite größer gleich der rechten Seite
<=	kleiner gleich	linke Seite kleiner gleich der rechten Seite
>	größer	linke Seite größer als die rechten Seite
<	kleiner	linke Seite kleiner als die rechten Seite
!=	ungleich	linke Seite ungleich rechte Seite
==	gleich	linke Seite gleich rechte Seite

3.2 Verknüpfungen

Operator	Bedeutung	Beschreibung
&&	und	beide Bedingungen müssen wahr sein
||	oder	eine Bedingung muss wahr sein

Bsp:

```
int a = 5;
int b = 3;
int c = 4;
if ((a + b + c > 10) || (a == b))
{
    Console.WriteLine("Das Ergebnis ist größer als 10");
    Console.WriteLine("Oder die erste Zahl ist gleich der Zweiten");
}
else
{
    Console.WriteLine("Das Ergebnis ist nicht größer als 10");
    Console.WriteLine("Und die erste Zahl ist nicht gleich der Zweiten");
}
```

4 Schleifen

Um sich wiederholende Aufgaben zu erledigen verwendet man Schleifen.

4.1 while-Schleife

Die Befehle in der Schleife werden ausgeführt solange die Bedingung wahr ist.

```
int zähler = 0;
while (zähler < 10)
```

```
{
  Console.WriteLine("Hallo Welt! Der Zähler ist " + zähler");
  zähler++;
}
```

4.2 do-while-Schleife

Die Befehle in der Schleife werden ausgeführt solange die Bedingung wahr ist.

```
int zähler = 0;
do
{
  Console.WriteLine("Hallo Welt! Der Zähler ist " + zähler");
  zähler++;
}while (zähler < 10);
```

4.3 for-Schleife

Bei der for-Schleife wird intern die Zahl der Wiederholungen festgelegt.

```
for(int zähler = 0; zähler < 10; zähler++)
{
  Console.WriteLine("Hallo Welt! Der Zähler ist " + zähler);
}
```

4.4 label-goto

Um Innerhalb eines Programms zu verschiedenen Stelen im Code zu springen kann man labels verwenden und mit hilfe des goto Befehls zur entsprechenden stelle springen:

```
label:
        Code1;
        if (bedingung==true) goto label;//Programm springt zurück zu label
        else Code2;//Programm läuft weiter
```

5 Arrays und strings

5.1 Arrays

Ein array ist eine Variablensammlung (ein Feld von Variablen) des selben Typs.

```
int[] a = new int[10]; //Deklaration eines array der Länge 10
for (int i = 0; i < a.Length; i++)
{
    a[i] = i * i;
}
for (int i = 0; i < a.Length; i++)
{
    Console.WriteLine("a[{0}] = {1}", i, a[i]);
}
```

Die Deklaration kann mit der Initialisierung kombiniert werden:

```
//Variante 1
int[] a = new int[] {1, 2, 3};
//Variante 2
int[] a = {1, 2, 3};
```

```
//Variante 3
int[] t = new int[3];
t[0] = 1;
t[1] = 2;
t[2] = 3;
int[] a = t;
```

Alle drei Varianten tun das gleiche.

Arrays haben eine Länge (d.h. Anzahl der Elemente) `a.Length` und jedes Element kann einzeln abgefragt werden `a[2]`.

5.2 strings

Strings sind Zeichenketten und gleichzeitig arrays vom Typ char, d.h. jedes Element eines strings (also jedes Zeichen der Zeichenkette) kann einzeln als char abgefragt werden.

```
string wort = "Hallo"
char zeichen0;
char zeichen1;
zeichen0 = wort[0];// Zeichen0 ist dann 'H'
zeichen0 = wort[1];// Zeichen1 ist dann 'a'
```

Soll z.B. die letzte Ziffer aus der Variablen `string a = "456"` in einen int umgewandelt werden, ist es nötig zweimal zu konvertieren: `int x = Convert.ToInt32(Convert.ToString(a[2]))`.

String Variablen haben weitere nützliche Methoden:
Gegeben sei `string s`

Methode	Bedeutung
`s.Length`	gibt die Länge des string an
`s.substring(x,y)`	erzeugt eine Teilzeichenkette von s beginnend bei Zeichen an Stelle x und mit der Länge y.

5.3 characters

Auch der Datentyp char bietet nützliche funktionalitäten:
`Char.IsLetter()` beispielsweise überprüft ob der angegebene char ein Buchstabe ist. `Char.Is...` bietet viele weitere nützliche Methoden an, diese geben in der Regel true oder false zurück!

6 Methoden

Methoden sind einzelne Codeabschnitte die im Code aufgerufen werden können zB `Math.Cos()`. Diese können auch selbst geschrieben werden, dafür wird ein Name eine Typ Argumente und Rückgabewerte benötigt. Der Typ void benötigt keinen Rückgabewert. Beispiel, die Summe aus zwei integer Zahlen berechnen und als double zurück geben:

```
private double Summe(int a, int b)
{
 double x;
 x = Convert.ToDouble(a + b);
 return x;
}
//Aufgerufen wir die Methode dann zB so
double y = Summe(2,3);
```

7 Zufallszahlen

Werden Zufallszahlen benötigt kann der interne Zufallszahlengenerator verwendet werden um beliebige ganze Zahlen aus einem Intervall oder Gleitkommazahlen zwischen 0.0 und 1.0 zu erzeugen:

```
Random rng = new Random();
int a;
double b;

a = rng.Next(5,26) // erzeugt eine Zahl von 5 bis 25
b = rng.NextDouble() // erzeugt eine Zahl von 0.0 bis 1.0
```

8 Sonstiges

Befehl	Bedeutung
`a += 7`	`a = a + 7`
`i++`	`i = i + 1`
`a % 2`	Division mit Rest

9 WPF

Windos Presentation Foundation (WPF) bezeichnet ein Framework zur Erstellung grafischer Benutzeroberflächen. Programme werden nun aufgeteilt in Code behind (C#) und View (xaml) Codedateien.
Die Oberfläche kann mit Hilfe der Toolbox und dem Designer, dierekt als xaml Code oder auch über den Code behind gestaltet werden.

9.1 Container

Ein Programm besitzt ein Fenster das MainWindow, darin kann genau ein Container platziert werden, zB Grid, Canvas, Panel, etc. Jeder Container kann wiederum weitere Container oder Elemente enthalten.

9.1.1 Grid

Ein Grid wird genutzt um Steuerelemente (Controls) zu platzieren, dies kann frei im dahinterliegenden Koordinatensystem geschehen oder innerhalb von erstellten Zeilen und Spalten innerhalb des Grid:

```
<Window x:Class="Beispiel.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Beispiel"
        mc:Ignorable="d"
        Title="MainWindow" Height="500" Width="900">
 <Grid>
  <Grid.ColumnDefinitions>
   <ColumnDefinition Width="2*"/>
   <ColumnDefinition Width="3*"/>
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
```

```
    <RowDefinition Height="1*"/>
    <RowDefinition Height="2*"/>
    <RowDefinition Height="3*"/>
   </Grid.RowDefinitions>
    <Button x:Name="Btn_R" Grid.Row="1" Grid.Column="0" Content="R" Click="Btn_R_Click"/>
    <Button x:Name="Btn_C" Grid.Row="2" Grid.Column="0" Content="C" Click="Btn_C_Click"/>
   </Grid>
</Window>
```

Hier wird ein Fenster mit einem Grid erzeugt, das in drei Zeilen (je ein Sechstel, ein Drittel und ein Halb der Gesamthöhe) und zwei Spalten (je zwei und drei Fünftel der Gesamtbreite) aufgeteilt ist. Außerdem werden zwei Buttons platziert entsprechend in Zeilen und Spalten mit den Angaben `Grid.Row="1" Grid.Column="0"` zB in Zeile 1 und Spalte 0.
Das gleiche kann im Code behind wiefolgt mit entsprechendem xaml als Basis erzeugt werden:

```
<Window x:Class="Beispiel.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Beispiel"
        mc:Ignorable="d"
        Title="MainWindow" Height="500" Width="900">
 <Grid x:Name="Gitter"/>
</Window>
```

```
public partial class MainWindow : Window
{
 public MainWindow()
 {
  InitializeComponent();
  Erstellen();
 }

 private void Erstellen()
 {
  for (int i = 0; i < 2; i++)
  {
   ColumnDefinition cd = new ColumnDefinition();
   cd.Width = new GridLength(i+2, GridUnitType.Star);
   Gitter.ColumnDefinitions.Add(cd);
  }
  for (int i = 0; i < 3; i++)
  {
   RowDefinition rd = new RowDefinition();
   rd.Height = new GridLength(i+1, GridUnitType.Star);
   Gitter.RowDefinitions.Add(rd);
  }
  Button Btn_R = new Button();
  Btn_R.Content = "R";
  Btn_R.Click += new RoutedEventHandler(Btn_R_Click);
  Grid.SetRow(Btn_R,1);
  Grid.SetColumn(Btn_R,0);
  Gitter.Children.Add(Btn_R);

  Button Btn_C = new Button();
  Btn_C.Content = "R";
  Btn_C.Click += new RoutedEventHandler(Btn_C_Click);
  Grid.SetRow(Btn_C,2);
  Grid.SetColumn(Btn_C,0);
  Gitter.Children.Add(Btn_C);
 }
}
```

Zusätzlich müssen im Code behind bei beiden Varianten jeweils die Methoden für den entsprechenden Button_Click erstellt werden!

```
private void Btn_R_Click(object sender, RoutedEventArgs e)
{

}

private void Btn_C_Click(object sender, RoutedEventArgs e)
{

}
```

9.2 Beispiele für Controls

9.2.1 TabControl

```
<Window x:Class="Beispiel.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Beispiel"
        mc:Ignorable="d"
        Title="MainWindow" Height="500" Width="500">
    <Grid>
        <TabControl>
            <TabItem Header="Allgemein">
                <Label Content="Inhalt hierhin..." />
            </TabItem>
            <TabItem Header="Aufgabe 1" />
            <TabItem Header="Aufgabe 2" />
        </TabControl>
    </Grid>
</Window>
```

9.2.2 Combobox

```
<Window x:Class="Beispiel.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Beispiel"
        mc:Ignorable="d"
        Title="MainWindow" Height="500" Width="500">
    <Grid>
        <x:Name="CB_n" SelectionChanged="CB_n_SelectionChanged">
            <ComboBoxItem>ComboBox Item #1</ComboBoxItem>
            <ComboBoxItem IsSelected="True">ComboBox Item #2</ComboBoxItem>
            <ComboBoxItem>ComboBox Item #3</ComboBoxItem>
        </ComboBox>
    </Grid>
</Window>
```

Die Items der Combobox können dann Beispielsweise wie folgt verwendet werden:

```
private void CB_n_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
 int n = CB_n.SelectedIndex;
 switch (index)
 {
  case 0:
  //code
  break;
  case 1:
  //code
  break;
  case 2:
```

```
  //code
  break;
 }
}
```

9.3 MessageBox

Hier ein Beispiel zur Verwendung von MessageBox:

```
//Messagebox konfigurieren
string messageBoxText = "Gewonnen! Nochmal?";
string caption = "Ende";
MessageBoxButton button = MessageBoxButton.YesNo;
MessageBoxImage icon = MessageBoxImage.Warning;

//Messagebox anzeigen
MessageBoxResult result = MessageBox.Show(messageBoxText, caption, button, icon);

//Auswahl der aus der Messagebox auswerten
switch (result)
{
 case MessageBoxResult.Yes://Neustart
  System.Diagnostics.Process.Start(Application.ResourceAssembly.Location);
  System.Windows.Application.Current.Shutdown();
  break;
 case MessageBoxResult.No://Beenden
  Application.Current.Shutdown();
  break;
}
```

10 Objekorientiertes Programmieren

10.1 Klassen, Methoden und Eigenschaften

Klassen werden wie folgt definiert, dazu gehört mindestens eine Methode, der Konstruktor, eine Klasse kann weitere Methoden, sowie Eigenschaften enthalten. Zu unterscheiden sind Zugriffmodifikatoren, public (voller Zugriff), protected (Zugriff durch abgeleitete Klassen) und private (kein Zugriff von außen). Des weiteren kann man statische Eigenschaften bzw. Methoden definieren, diese sind dann für alle Objektinstanzen die *selben*.

```
class Dreieck //Klasse Dreieck wird definiert
 {
  //Eigenschaften
  double x, y, vx, vy;
  Polygon umriss;//Deklaration eines Polygons
  //Konstruktor
  public Dreieck(Canvas Zeichenflaeche, double x, double y)
  {
   this.x = x;
   this.y = y;
   vy = 0;
   vx = 50;
   //Definition des Polygons
   umriss = new Polygon();
```

```
  umriss.Points.Add(new Point(0,20));
  umriss.Points.Add(new Point(5,7));
  umriss.Points.Add(new Point(-5,7));
  umriss.Fill = Brushes.Gray;
 }
 //weitere Methoden
 public void Zeichne(Canvas Zeichenflaeche)
 {
  //Polygon in Bewegungsrichtung ausrichten
  double winkelInGrad = Math.Atan2(vy, vx) * 180 / Math.PI -90;
  umriss.RenderTransform = new RotateTransform(winkelInGrad);

  Canvas.SetTop(umriss, y);
  Canvas.SetLeft(umriss, x);
  Zeichenflaeche.Children.Add(umriss);
 }
 public void Bewegen(TimeSpan intervall, Canvas Zeichenflaeche)
  {
   x += intervall.TotalSeconds * vx;
   y += intervall.TotalSeconds * vy;

   if (x < 0) x = Zeichenflaeche.ActualWidth;
   if (x > Zeichenflaeche.ActualWidth) x = 0;
   if (y < 0) y = Zeichenflaeche.ActualHeight;
   if (y > Zeichenflaeche.ActualHeight) y = 0;
  }

  public void RichtungAendern(int Richtung)
  {
   switch (Richtung)
   {
    case 0:
     vx = -50;
     vy = 0;
     break;
    case 1:
     vx = 50;
     vy = 0;
     break;
    case 2:
     vx = 0;
     vy = -50;
     break;
    case 3:
     vx = 0;
     vy = 50;
     break;
   }
  }
}
```

10.2 Timer, Listen, foreach-Schleife

Timer können beispielsweise zur Animation verwendet werden. Listen finden Verwendung, wenn viele Objektinstanzen der gleichen Klasse benötigt werden. Um durch eine Liste zu iterieren verwendet man üblicherweise die foreach-Schleife.

Alles dargestellt im nachfolgenden Bsp.:

```
public partial class MainWindow : Window
{
 DispatcherTimer timer = new DispatcherTimer();//Definition des timers
 List<Dreieck> Dreiecke = new List<Dreieck>();

 public MainWindow()
 {
  InitializeComponent();
  timer.Interval = TimeSpan.FromMilliseconds(17);//timer intervall setzten
  timer.Tick += Animiere;//Methode zum Aufruf bei jedem timer tick hinzufügen
 }
 //Methode zum Aufruf bei jedem timer tick definieren
 private void Animiere(object sender, EventArgs e)
 {
  Zeichenflaeche.Children.Clear();
  //durch alle Objekte vom Typ Dreieck in der Liste Dreiecke iterieren
  foreach (Dreieck item in Dreiecke)
  {
   //Methodenaufruf der öffentlichen Methode Zeichne des Objekts Dreieck
   item.Zeichne(Zeichenflaeche);
   item.Bewegen(timer.Interval, Zeichenflaeche)
  }
 }

 private void BTN_Start_Click(object sender, RoutedEventArgs e)
 {
  timer.Start();//timer starten
  BTN_Start.IsEnabled = false;
  for (int i = 0; i < 20; i++)
  {
   double x, y;
   x = Zeichenflaeche.ActualWidth / 20 * i;
   y = Zeichenflaeche.ActualHeight / 20 * i;
   Dreiecke.Add(new Dreieck(Zeichenflaeche, x, y));
  }
 }
}
```

10.3 Tasten zum Methodenaufruf verwenden

```
private void Window_KeyDown(object sender, KeyEventArgs e)
{
 if (timer.IsEnabled)
 {
  switch (e.Key)
  {
   case Key.Left:
    Dreiecke.ForEach(x => x.RichtungAendern(0));
```

```
    break;
   case Key.Right:
    Dreiecke.ForEach(x => x.RichtungAendern(1));
    break;
   case Key.Up:
    Dreiecke.ForEach(x => x.RichtungAendern(2));
    break;
   case Key.Down:
    Dreiecke.ForEach(x => x.RichtungAendern(3));
    break;
  }
 }
}
```

10.4 Abgeleitete Klassen, abstract, virtual, override, base

Abstrakte Klassen dienen ausschließlich als Mutterklasse von der keine Objektinstanzen erzeugt werden können. Abstrakte Klassen können abstrakte Methoden enthalten, die also in der Mutterklasse nur deklariert und erst von den Abgeleiteten Klassen definiert werden müssen. Virtuelle Methoden können von innerhalb der abgeleiteten Klassen überschrieben werden. Das base Schlüsselwort kann verwendet werden um überschriebene Methoden oder den abgeleiteten Konstruktor zu definieren.

```
abstract class Appointment
{
 private int minutes { get; set; }
 private int hours { get; set; }
 private string title { get; set; }

 private static int n;

 public Appointment(string title, int hours, int minutes)
 {
  this.title = title;
  this.hours = hours;
  this.minutes = minutes;
  n++;
 }

 public override string ToString()//ToString ist bereits eine virtuelle Methode
 {
  return hours.ToString("D2") + ":" + minutes.ToString("00") + " " +title;
 }

 public abstract void Alarm();


 public bool IsActive()
 {
  DateTime d = DateTime.Now;
  if (d.Hour == hours && d.Minute == minutes) return true;
  else return false;
 }
```

```
}

class AppointmentSound : Appointment
{
 public AppointmentSound(string titel, int hours, int minutes ) : base(titel, hours, minutes)
 {
 }

 public override string ToString()
 {
  return "K "+base.ToString();
 }

 public override void Alarm()
 {
  System.Media.SystemSounds.Beep.Play();
 }
}
```