Course 2DCis: 2D-Computer Graphics with C#
Chapter C1: The Intro Project

Ein leeres Fenster

Anleitung für **Visual Studio 2008**:
`0)` Main Menu nach dem Start von VS 2008: `Tools → Options →` check lower left checkbox: `Show all Settings → Projects and Solutions → Visual Studio projects location: → C:\temp`

`1)` Main Menu nach dem Start von VS 2008: `File → New Project... →`
`Visual Studio installed templates: Windows Forms Application`
`Name: intro1 → Location: C:\temp → Create directory for solution:` ausschalten `→ OK`
Es meldet sich `Form1.cs[Design]`.

`2)` Sie müssen zwei überflüssige Files löschen: `Form1.Designer.cs` und `Program.cs`.
Sie erreichen diese Files über das `Solution Explorer - intro1`-Window:
Klicken Sie das Pluszeichen vor `intro1` und dann das Pluszeichen vor `Form1.cs`.
Klicken Sie mit der **rechten** Maustaste auf den Ast `Program.cs`. Es öffnet sich ein Kontextmenu.
Sie Klicken auf `Delete`. Eine Message Box erscheint: `'Program.cs' will be deleted permanently.`
Sie quittieren mit `OK`.
Klicken Sie mit der **rechten** Maustaste auf den Ast `Form1.Designer.cs` und löschen auch dieses File.

`3)` Klicken Sie mit der **rechten** Maustaste auf das graue Fenster `Form1`.
Es öffnet sich ein kleines Kontextmenü. Klicken Sie auf `View Code`.
Sie sehen jetzt den vorprogrammierten Code von `Form1.cs`. Löschen Sie den gesamten Code vollständig.

`4)` Schreiben Sie in das vollständig leere File `Form1.cs` folgende 3 Zeilen:

```
public class Form1 : System.Windows.Forms.Form
{ static void Main() { System.Windows.Forms.Application.Run( new Form1() ); }
}
```

`5)` Klicken Sie `Debug` im Main Menu oberhalb des Hauptfensters.
Es öffnet sich ein Untermenü. Klicken Sie auf `Start Without Debugging Ctrl F5`.

Das Miniprogramm wird jetzt automatisch übersetzt, gelinkt und gestartet.
Beachten Sie das `Error List`-Fenster am unteren Rand von Visual Studio.
Unser Programm startet automatisch als eigenes Fenster, das aus drei Teilen besteht:
1. Hauptfenster = MainFrame mit blauer Titelzeile
2. drei funktionierende Buttons für `Minimize, Maximize, Close`
3. ein schmaler Rahmen, dessen 4 Seiten beweglich sind und dessen 4 Ecken beweglich sind.
Verkleinern/Vergrößern Sie den Rahmen.
Diese gesamte Funktionalität ist in der vorprogrammierten Klasse `System.Windows.Forms.Form` enthalten und steht jetzt zu Verfügung als Basis für Ihr erstes C#-Programm.

Minimieren Sie Visual Studio, damit Sie ein Gefühl dafür bekommen, dass `intro1.exe` ein Windows-Programm ist. Starten Sie den `Explorer`. Gehen sie nach `C:\temp\intro1\bin\Release.`
Doppelklicken Sie auf `intro1.exe`. Sie können so beliebig viele Instanzen von `intro1.exe` starten (Diese müssen aber alle einzeln wieder beendet werden, bevor das Programm geändert wird.). Minimieren Sie den `Explorer`.

Wichtig: Immer zuerst alle Instanzen von `intro1` beenden, bevor neuer Code eingegeben und übersetzt wird !
Im Zweifel Task Manager mit `Ctrl+Alt+Del` starten und kontrollieren, ob noch ein `intro1.exe`-Prozess läuft und diesen töten.

DrawString: Hallo Welt

Falls Sie den Code nicht mehr sehen, klicken Sie in Visual Studio auf den Karteireiter **`Form1.cs`**,
das bringt den Code zum Vorschein.
Lesen Sie zunächst den **Wichtigen Hinweis 2** unten in diesem Absatz, sonst wird Sie der Code-Editor durch eigenmächtige Formatierungen in den Wahnsinn treiben.
Löschen Sie alles von der ersten Zeile bis zur letzten (inklusive der letzten geschweiften Klammer).
Es bleibt nichts stehen.
Schreiben Sie in das jetzt leere Codefenster `Form1.cs` folgende Zeilen:

```
using System;
using System.Drawing;
using System.Windows.Forms;

public class Form1 : Form
{ [STAThread] static void Main() { Application.Run( new Form1() ); }
  Font arial18 = new Font( "Arial", 18 );
  Brush blackbrush = SystemBrushes.ControlText;
  protected override void OnPaint( PaintEventArgs e )
  { Graphics g = e.Graphics;
    g.DrawString( "Hello world !", arial18, blackbrush, 0, 0 );
  }
}
```

Klicken Sie `Debug` in der Menü-Zeile oberhalb des Hauptfensters
Es öffnet sich ein Untermenü. Klicken Sie auf `Start Without Debugging Ctrl F5`.
Das Programm wird wieder automatisch übersetzt, gelinkt und gestartet.

Beenden Sie diese Primitiv-Version und programmieren Sie einen Konstruktor `public Form1()`:

```
using System;
using System.Drawing;
using System.Windows.Forms;

public class Form1 : Form
{ [STAThread] static void Main() { Application.Run( new Form1() ); }
  Font arial18 = new Font( "Arial", 18 );
  Brush blackbrush = SystemBrushes.ControlText;
  public Form1()
  { BackColor  = Color.White;
    Text       = "intro1";
    SetStyle( ControlStyles.ResizeRedraw, true );
    StartPosition = FormStartPosition.Manual;
    Top    =  50;
    Left   =  50;
    Width  = 800;
    Height = 600;
  }
  protected override void OnPaint( PaintEventArgs e )
  { Graphics g = e.Graphics;
    String s0 = "Hello world, here is intro1 !";
    g.DrawString( s0, arial18, blackbrush, 0, 0 );
  }
}
```

Klicken Sie `Debug` in der Menü-Zeile oberhalb des Hauptfensters
Es öffnet sich ein Untermenü. Klicken Sie auf `Start Without Debugging Ctrl F5`.
Das Programm wird wieder automatisch übersetzt, gelinkt und gestartet.
Probieren Sie andere Startpositionen und Abmessungen des Fensters, andere Strings, andere Fontgrößen, andere Fonts (z.B. `Courier New`), andere Farben (z.B. `Color.Blue, Color.Green`) und andere Textpositionen.

Wichtiger Hinweis: Wenn Sie sich beim Programmieren vertippt haben, dann übersetzt der Compiler nichts, sondern meldet sich mit einer Message Box: `There were build errors. ...`. Sie quittieren mit **`No`**.
Am unteren Rand von Visual Studio erscheint ein `Error List`-Fenster mit Warnungen und Fehlermeldungen. Scrollen Sie innerhalb dieser Fehlerliste nach oben zur ersten Fehlermeldung (Ignorieren Sie eventuelle Warnungen). Doppelklicken Sie die erste Fehlermeldung. Der Cursor springt nun automatisch in Ihren Code auf die Zeile, wo der Fehler entdeckt wurde. Suchen Sie dort den Fehler (manchmal liegt er gar nicht in dieser Zeile, sondern in vorherigen Zeilen, wenn dort ein Semikolon oder eine Klammer fehlt) und beseitigen Sie ihn. Ignorieren Sie alle weiteren Fehler (das sind meistens Folgefehler des ersten Fehlers) und übersetzen Sie neu. Diesen Vorgang wiederholen Sie so oft, bis keine Fehlermeldung mehr erscheint und Ihr übersetztes und gelinktes Programm automatisch startet.
Wenn Sie nicht Visual C# Express sondern Visual Studio 2005 Professional benutzen, sollten Sie die Nerven tötende Formatier- und Einrückautomatik des Code-Editors auf folgendem Weg ausschalten:
1. Hauptmenu von Visual Studio 2005 Professional: Klick auf Menupunkt "`Tools`".
2. Es erscheint ein DropDown-Menu. Klick auf "`Options...`".
3. Es erscheint eine `Options` Dialog Box.
4. Klick auf den Ast "`Projects and Solutions`". Klick auf "`General`". Alle drei Pfade auf `C:\temp` stellen.
5. Klick auf den Ast "`Text Editor`", dann auf "`C#`".
6. Es erscheint ein Unterbaum mit den Ästen "`General, Tabs, Advanced, Formatting, IntelliSense`".
7. Klick auf "`Tabs`". Stellen Sie "`Indenting`" auf `None`, "`Tab size`" und "`Indent size`" auf `1` und schalten Sie die Option "`Insert spaces`" ein.
8. Klicken sie im Unterbaum "`C#`" auf das Pluszeichen vor "`Formatting`" und
ändern Sie alle "`Formatting`"-Äste:
"`General`": alle CheckBoxes ausschalten, "`Indentation`": alle CheckBoxes ausschalten, "`New Lines`": alle CheckBoxes ausschalten, "`Spacing`": alle CheckBoxes ausschalten,
"`Wrapping`": beide CheckBoxes **ein**schalten.
9. Verlassen Sie den Dialog mit Button "`OK`".

Fenstergröße anzeigen und farbiger Text

Version2: Beenden Sie Ihr Programm `intro1`.
Löschen Sie im Kopf der Klasse `Form1()` die beiden Zeilen `Font arial18 = new Font("Arial", 18);` und `Brush blackbrush = SystemBrushes.ControlText;` und ersetzen diese durch:

```
Font  arial18    = new Font( "Arial", 18 );
Font  arial16    = new Font( "Arial", 16 );
Font  courier14  = new Font( "Courier New", 14 );
Brush blackbrush = SystemBrushes.ControlText;
Brush redbrush   = new SolidBrush( Color.Red );
Brush whitebrush = new SolidBrush( Color.White );
Pen   blackpen   = new Pen( Color.Black, 4 );
```

Löschen Sie die 5-zeilige Funktion `protected override void OnPaint(...)` und schreiben Sie diese neu:

```
protected override void OnPaint( PaintEventArgs e )
{ //Version 2 ************************************************
  Graphics g = e.Graphics;
  Rectangle cr = ClientRectangle;
  String s0  = "Hello world, here is intro1 !";
  String s1  = "Change the size of your window by dragging a corner !";
  String s2w = "Form  : Width = " +    Width.ToString();
  String s3w = "Client: Width = " + cr.Width.ToString();
  String s2h = "  Height= " +    Height.ToString();
  String s3h = "  Height= " + cr.Height.ToString();
  g.DrawString( s0       , arial18  , blackbrush, 0,  0 );
  g.DrawString( s1       , arial16  , redbrush  , 0, 20 );
  g.DrawString( s2w + s2h, courier14, blackbrush, 0, 40 );
  g.DrawString( s3w + s3h, courier14, blackbrush, 0, 60 );
}
```

Klicken Sie `Debug` in der Menü-Zeile oberhalb des Hauptfenster und dann `Start Without Debugging Ctrl F5`.

links, rechts, oben, unten

Version3: Beenden Sie Ihr Programm `intro1`.
Schreiben Sie sechs weitere Zeilen in die Funktion `protected override void OnPaint(...)` unterhalb die schon vorhandenen Befehle, aber noch vor die geschweifte Klammer, die `...OnPaint(...)` abschließt:

```
    //Version 3 ************************************************
    Point mid = new Point( cr.Width/2, cr.Height/2 );
    g.DrawString( "left"  ,arial16, blackbrush, 0          , mid.Y        );
    g.DrawString( "right" ,arial16, blackbrush, cr.Width-50, mid.Y        );
    g.DrawString( "top"   ,arial16, blackbrush, mid.X      , 0            );
    g.DrawString( "bottom",arial16, blackbrush, mid.X      , cr.Height-30 );
```

Klicken Sie `Debug` und dann `Start Without Debugging Ctrl F5`.

Line, Rectangle, Ellipse

Version4: Beenden Sie Ihr Programm `intro1`.
Schreiben Sie 8 weitere Zeilen in die Funktion `protected override void OnPaint(...)` unterhalb die schon vorhandenen Befehle, aber noch vor die geschweifte Klammer, die `...OnPaint(...)` abschließt:

```
    //Version 4 ************************************************
    g.DrawLine( blackpen, 0, 0, cr.Width, cr.Height );
    g.DrawLine( blackpen, cr.Width, 0, 0, cr.Height );
    Int32 w5 = cr.Width  / 5;
    Int32 h5 = cr.Height / 5;
    g.FillRectangle( whitebrush, w5, h5, 3 * w5, 3 * h5 );
    g.DrawRectangle( blackpen  , w5, h5, 3 * w5, 3 * h5 );
    g.DrawEllipse  ( blackpen  , w5, h5, 3 * w5, 3 * h5 );
```

Klicken Sie `Debug` und dann `Start Without Debugging Ctrl F5`.

Zeichenstifte mit zufälligen Farben zeichnen einen Stern mit zufälligen Strichlängen

Version5: Beenden Sie Ihr Programm `intro1`.
Definieren Sie einen neuen Zeichenstift (= `Pen`) im Kopf von `public class Form1 : Form` direkt unter der Zeile `Pen blackpen = new Pen( Color.Black, 4 );`:

```
  Pen   randompen  = new Pen( Color.Black, 20 );
```

Schreiben Sie folgende weiteren Befehle in die Funktion `protected override void OnPaint(...)` unterhalb der schon vorhandenen Befehle, wie bei Version 3 und 4:

```
    //Version 5 ************************************************
    Int16 i, nn = 120;
    Int32 red, green, blue;
    randompen.EndCap = System.Drawing.Drawing2D.LineCap.DiamondAnchor;
    Point[] splash = new Point[nn];
    Double arcus_1 = 2.0 * Math.PI / nn;
    Double arcus_i, factor, sinus, cosinus;
    Double radius_x = 1.35 * w5;
    Double radius_y = 1.35 * h5;
    Random random = new Random();
    for ( i=0; i < nn; i++ )
    { red   = random.Next( Byte.MaxValue );
      green = random.Next( Byte.MaxValue );
      blue  = random.Next( Byte.MaxValue );
      randompen.Color = Color.FromArgb( red, green, blue );
      factor = Math.Max( 0.25, random.NextDouble() );
      arcus_i = arcus_1 * i;
      cosinus = radius_x * factor * Math.Cos( arcus_i );
      sinus   = radius_y * factor * Math.Sin( arcus_i );
      g.DrawLine( randompen, mid.X, mid.Y, mid.X + (Int32)cosinus, mid.Y + (Int32)sinus );
      splash[i].X = mid.X + (Int32)(cosinus * 0.8);
      splash[i].Y = mid.Y + (Int32)(  sinus * 0.8);
    }
```

Klicken Sie `Debug` und dann `Start Without Debugging Ctrl F5`.
Erproben Sie das Programm durch Ziehen am Fensterrand.
Theorie siehe: **../../Lectures/L02_2DVector/2DVector_deutsch.htm#a12**

Polygon zeichnen

Version6: Beenden Sie Ihr Programm `intro1`.
Schreiben Sie folgende zwei Befehle in die Funktion `protected override void OnPaint(...)` direkt unterhalb der geschweiften Klammer, die die `for`-Schleife beendet:

```
    //Version 6 ************************************************
    g.FillClosedCurve( redbrush, splash );
    g.DrawString( "splash !", arial18, whitebrush, mid.X - 40, mid.Y - 9 );
```

Klicken Sie `Debug` und dann `Start Without Debugging Ctrl F5`.
Erproben Sie das Programm durch Ziehen am Fensterrand.

Animieren

Version7: Beenden Sie Ihr Programm `intro1`.
Schreiben Sie folgende zwei Befehle in die Funktion `protected override void OnPaint(...)` unter die letzte Befehlszeile `g.DrawString( "splash !", arial18, whitebrush, mid.X - 40, mid.Y - 9 );`, aber noch vor den beiden abschließenden Zeilen mit den geschweiften Klammern:

```
    //Version 7 ************************************************
    System.Threading.Thread.Sleep( 100 ); //Wait 100 milliseconds.
    Invalidate(); //Ask the operating system to raise the Paint event again.
```

Klicken Sie `Debug` und dann `Start Without Debugging Ctrl F5`.
Erproben Sie das Ziehen am Fensterrand.

Weitere Aufgaben

Klicken Sie auf `Help` in der Menüleiste von Visual Studio. Klicken Sie auf das Untermenü `Index`.
Gehen Sie in das Feld `Filtered by:` und wählen Sie dort `.NET Framework SDK`. Dann geben Sie im Feld `Look for:` folgende Schlüsselwörter ein: `DrawString, DrawLine, Random, FillClosedCurve, Pen, Brush` etc. Sie bekommen dann eine Auswahl von Stichwörtern, mit dem gleichen Anfang.
Lesen Sie die Hilfetexte. Das Hilfefenster überdeckt Ihren Code. Wenn Sie mit dem lesen fertig sind, entfernen Sie das Hilfefenster mit dessen X-Button in der rechten oberen Fensterecke.
Beenden Sie Visual Studio, starten sie den Explorer, löschen Sie die gesamte Directory `C:\temp\intro1`.
Starten Sie Visual Studio wieder und erzeugen dasselbe Programm so oft, bis Sie das Programm ohne Anleitung und schnell von Null an erstellen, verändern und bedienen können.
Benutzen Sie Drag&Drop beim Schreiben von Code = verschieben und kopieren mit Maus mit/ohne `Strg`-Taste.
Erfinden und erproben Sie neue Varianten des Programms (in Form von neuen Projekten `Intro2, Intro3` usw. nach obigem Muster), z.B. bunte Striche waagrecht parallel, senkrecht parallel, bunte Rechtecke und Ellipsen an zufälligen Stellen etc.

Fragen und Antworten

F: **Was ist eine Form ? Wichtige Eigenschaften ?**
A: Am Bildschirm sichtbare Programme benötigen mindestens eine `Form` = rechteckige Benutzeroberfläche = User Interface. Ein Programm startet mit derjenigen `Form`, die die Methode `Main` enthält = Startklasse.
Wichtige Eigenschaften der `Form`-Klasse: Position auf dem Display, Größe, Farbe, Ränder, Inhalt der Titelleiste, Verhalten.
Details der `Form`-Klasse: **http://msdn2.microsoft.com/de-de/library/system.windows.forms.form(VS.80).aspx**

F: **Was ist ein Rectangle ? Wichtige Eigenschaften ?**
A: Ein achsparalleles Rechteck mit `X/Y` = Position der linken oberen Ecke und `Width/Height` = Breite und Höhe.
Beispiel: `Rectangle r = new Rectangle( 50, 50, 200, 100 );` definiert ein 200x100-Rechteck `r` mit der linken oberen Ecke bei (50/50).
Details der `Rectangle`-Klasse: **http://msdn2.microsoft.com/de-de/library/system.drawing.rectangle_members(vs.80).aspx**

F: **Was ist ein ClientRectangle ? Wie bekommt man Breite und Höhe ?**
A: Ein `ClientRectangle` ist ein `Rectangle` innerhalb einer rechteckigen `Form`.
Nur auf die `ClientRectangle`-Fläche kann man schreiben und zeichnen.
`X/Y` = (0/0) von `ClientRectangle` liegt direkt rechts neben dem linken Rand der Form und direkt unter der Titelleiste. `Width/Height` erstreckt sich nach links bis vor den rechten Rand der Form und nach unten bis vor den unteren Rand.

```
int w = ClientRectangle.Width; //nutzbare Breite der Form
int h = ClientRectangle.Height; //nutzbare Hoehe der Form
```

Details der `ClientRectangle`-Eigenschaft:
http://msdn2.microsoft.com/de-de/library/system.windows.forms.control.clientrectangle(VS.80).aspx

F: **Was ist Color ? Wichtige vordefinierte Colors ?**
A: Datenstruktur, die aus den drei Grundfarben rot, grün und blau eine Mischfarbe definiert.
`Color.Black` ist identisch mit `Color.FromArgb( 0, 0, 0 )`.
`Color.White` ist identisch mit `Color.FromArgb( 255, 255, 255 )`.
`Color.Red` ist identisch mit `Color.FromArgb( 255, 0, 0 )`.
`Color.Green` ist identisch mit `Color.FromArgb( 0, 255, 0 )`.
`Color.Blue` ist identisch mit `Color.FromArgb( 0, 0, 255 )`.
Alle vordefinierten Farben finden Sie hier:
http://msdn2.microsoft.com/de-de/library/system.drawing.color_members(VS.80).aspx

F: **Was ist ein Brush ? Was ist ein SolidBrush ? Was ist ein SytemBrush ? Komplexe Brushes ?**
A: Brush = graphisches Objekt zur Füllung leerer Flächen = Pinsel.
SolidBrush: einfachster Brush besteht nur aus einer Farbe. Beispiel:

```
Brush redbrush = new SolidBrush( Color.Red );
```

SystemBrush: In Windows vorhandener Brush, den man benutzen kann, ohne Speicherplatz zu verbrauchen. Beispiel:

```
Brush blackbrush = SystemBrushes.ControlText;
```

komplexe Brushes: `GradientBrush` = Farbverlauf, `HatchBrush` = Schraffur, `TextureBrush` = Rasterbild
Details der `Brush`-Klasse: **http://msdn2.microsoft.com/de-de/library/system.drawing.brush(VS.80).aspx**

F: **Was ist ein Pen ? Einfache Konstruktoren ? LineCap, StartCap, EndCap ?**
A: Pen = graphisches Objekt zum Zeichnen von Linien und Kurven = Stift.

```
Pen mypen = new Pen( Color ); //dünner farbiger Pen
Pen mypen = new Pen( Color, Width ); //dicker bzw. breiter farbiger Pen
Pen mypen = new Pen( Brush, Width ); //Pen zeichnet mit einem Brush
```

LineCap, StartCap, EndCap: Zeichenstil-Eigenschaft am Linienanfang und Linienende: flach, rund, Pfeile.
`LineCap`-Arten: **http://msdn2.microsoft.com/de-de/library/system.drawing.drawing2d.linecap(VS.80).aspx**

F: **Was ist Byte, Int16, Int32, Single, Double, Convert.ToInt32, Byte.MaxValue ?**
A: `Byte, ... Double` sind Klassen von .NET für diverse Zahlentypen.
`Convert.ToInt32` ist ein Typumwandler von irgendeinem Zahletyp nach `Int32`.
`Byte.MaxValue` ist eine Eigenschaft von `Byte`.
C# hat eigene Bezeichner: byte, short, int, float, double für diese Zahlentypen.
Aber die .NET-Namen sind teilweise aussagekräftiger und gelten in allen .NET-Sprachen.
`Byte = byte` : sehr kurze ganze Zahl zwischen 0 und 255 ohne Vorzeichen. Genügt für Farbabstufungen.
`Int16 = short` : kurze ganze Zahl mit Vorzeichen. Genügt für die meisten Zählvorgänge.
`Int32 = int` : lange ganze Zahl mit Vorzeichen. Am meisten verwendeter Zahlentyp.
`Single = float` : kurze Gleitkomma-Zahl. Genügt für Vektorgraphik.
`Double = double`: lange Gleitkomma-Zahl. Obligatorisch bei vielen Funktionen der `Math`-Bibliothek.
`Convert.ToInt32`: rundet `Single` und `Double` zu `Int32`.
`Byte.MaxValue` : ist der höchstmögliche Zahlenwert des Zahlentyps `Byte` = 255 = hexadezimal FF.

F: **Was ist Random ?**
A: R. ist eine Klasse, die bei Bedarf Zufallszahlen liefert. Beispiele:

```
Random r = new Random();
Int32  zufall = r.Next();          liefert irgendeine positive Integerzahl.
Int32  zufall = r.Next(25);        liefert irgendeine Integerzahl zwischen 0 und 24.
Int32  zufall = r.Next(10, 20); liefert irgendeine Integerzahl zwischen 10 und 19.
Double zufall = r.NextDouble(); liefert irgendeine Gleitkommazahl zwischen 0.0 und 1.0.
```

Siehe: **http://msdn2.microsoft.com/de-de/library/system.random_members(VS.80).aspx**

F: **Was ist ein Event ? Beispiele ?**
A: Event = Ereignis. Ein Ereignis ist eine Meldung, die von einem Objekt gesendet wird, um das Auftreten einer Aktion zu signalisieren. Die Aktion kann durch das Betriebssystem, durch Benutzerinteraktionen wie einen Mausklick oder durch andere Programmlogik ausgelöst worden sein.
Das Objekt, das das Ereignis auslöst, wird als Ereignissender bezeichnet.
Das Objekt, das das Ereignis abfängt und darauf reagiert, wird als Ereignisempfänger bezeichnet.
Der Ereignisempfänger programmiert seine Reaktion in Form einer Funktion, des Ereignishandlers.
Die meisten Windows-Programme bestehen überhaupt nur aus Ereignishandlern.
Beispiele für Events:
`Paint` : Fenster ist veraltet oder zerstört → neu zeichnen oder schreiben.
`MouseDown`: Jemand hat auf eine Maustaste gedrückt.
`MouseMove`: Jemand hat die Maus bewegt.
`KeyDown` : Jemand hat auf die Tastatur gedrückt.

F: **Was ist protected override void OnPaint(PaintEventArgs e) ?**
A: `OnPaint` ist ein Ereignishandler = eine Methode für die Ereignisbehandlung als Reaktion auf das Ereignis `Paint`, das signalisiert, dass der Inhalt von `Form1` veraltet oder verloren ist und dass alles neu gezeichnet werden muss.
`Form1` enthält per default (von Hause aus) einen leeren Ereignishandler `OnPaint`.
Wir überschreiben diesen vorhandenen `OnPaint`-Handler und ersparen uns damit die Anmeldung eines eigenen Handlers und müssen keinen eigenen Namen dafür erfinden.
Auslöser des Ereignisses `Paint` ist:
a) entweder das Betriebssystem (z.B. wenn `Form1` unter anderen Fenstern liegt und zum Vorschein kommen soll) oder
b) der User (z.B. wenn er `Form1` verschiebt) oder
c) der Programmierer, wenn er irgendwo im Programm `Invalidate();` schreibt.

F: **Wie verteilt man `nn` Punkte gleichmäßig auf einem Kreis mit dem Radius `radius` und dem Mittelpunkt `Point mid` ?**
A:
1. Alle Konstanten initialisieren.
2. Einen Winkelschritt in Bogenmaß ausrechnen: arcus_1 = 2*Pi / nn.
3. For-Schleife über i=0 bis i=nn-1 Schritte:
3.1. i-ter Winkelschritt: arcus_i = arcus_1 * i;
3.2. x-Koordinate des i-ten Punktes = mid.X + radius*cosinus(arcus_i); (Rundung!)
3.3. y-Koordinate des i-ten Punktes = mid.Y + radius* sinus(arcus_i); (Rundung!)
3.4. i-ten Punkt als kleinen schwarzen Kreis zeichnen.
Kompletter Code → Miniprogramm zum Training Nr. 4.

Weitere Erklärungen für jede einzelne Zeile von `intro1` finden Sie unter C2DCisIntro_Comment.htm.

Miniprogramme zum Training

1. MiniHello:

```
using System;
using System.Drawing;
using System.Windows.Forms;
public class Form1 : Form
{ [STAThread] static void Main() { Application.Run( new Form1() ); }
  Font arial10 = new Font( "Arial", 10 );
  Brush blackbrush = SystemBrushes.ControlText;
  public Form1()
  { SetStyle( ControlStyles.ResizeRedraw, true );
  }
  protected override void OnPaint( PaintEventArgs e )
  { Graphics g = e.Graphics;
    Rectangle cr = ClientRectangle;
    g.DrawString( "Hello", arial10, blackbrush, cr.Width/2-10, cr.Height/2-5 );
  }
}
```

2. MiniCross:

```
using System;
using System.Drawing;
using System.Windows.Forms;
public class Form1 : Form
{ [STAThread] static void Main() { Application.Run( new Form1() ); }
  Pen blackpen = new Pen( Color.Black, 4);
  public Form1()
  { SetStyle( ControlStyles.ResizeRedraw, true );
  }
  protected override void OnPaint( PaintEventArgs e )
  { Graphics g = e.Graphics;
    Rectangle cr = ClientRectangle;
    g.DrawLine( blackpen, 0, 0, cr.Width, cr.Height );
    g.DrawLine( blackpen, cr.Width, 0, 0, cr.Height );
  }
}
```

3. MiniCircle mit Zufallsfarbe:

```
using System;
using System.Drawing;
using System.Windows.Forms;
public class Form1 : Form
{ [STAThread] static void Main() { Application.Run( new Form1() ); }
  Pen blackpen = new Pen( Color.Black, 4 );
  Random r = new Random();
  public Form1()
  { SetStyle( ControlStyles.ResizeRedraw, true );
  }
  protected override void OnPaint( PaintEventArgs e )
  { Graphics g = e.Graphics;
    Rectangle cr = ClientRectangle;
    Byte red   = (Byte)r.Next( Byte.MaxValue );
    Byte green = (Byte)r.Next( Byte.MaxValue );
    Byte blue  = (Byte)r.Next( Byte.MaxValue );
    Brush rbrush = new SolidBrush( Color.FromArgb( red, green, blue ) );
    g.FillEllipse( rbrush  , cr.Width/5, cr.Height/5, (3*cr.Width)/5, (3*cr.Height)/5 );
    g.DrawEllipse( blackpen, cr.Width/5, cr.Height/5, (3*cr.Width)/5, (3*cr.Height)/5 );
  }
}
```

4. Punkte gleichmäßig auf einem Kreis verteilen:

```
using System;
using System.Drawing;
using System.Windows.Forms;
public class Form1 : Form
{ [STAThread] static void Main() { Application.Run(new Form1()); }
  Point mid = new Point( 200, 200 );
  const Int32 radius = 100, nn = 36;
  Double arcus_1 = 2.0 * Math.PI / nn;
  public Form1()
  { Text       = "Circular Points";
    Width  = 400; Height = 400;
  }
  protected override void OnPaint(PaintEventArgs e)
  { for ( Int32 i=0; i < nn; i++ )
    { Double arcus_i = arcus_1 * i;
      Int32 x = mid.X + Convert.ToInt32( radius * Math.Cos( arcus_i ) );
      Int32 y = mid.Y + Convert.ToInt32( radius * Math.Sin( arcus_i ) );
      e.Graphics.FillEllipse( SystemBrushes.WindowText, x-3, y-3, 7, 7 );
    }
  }
}
```