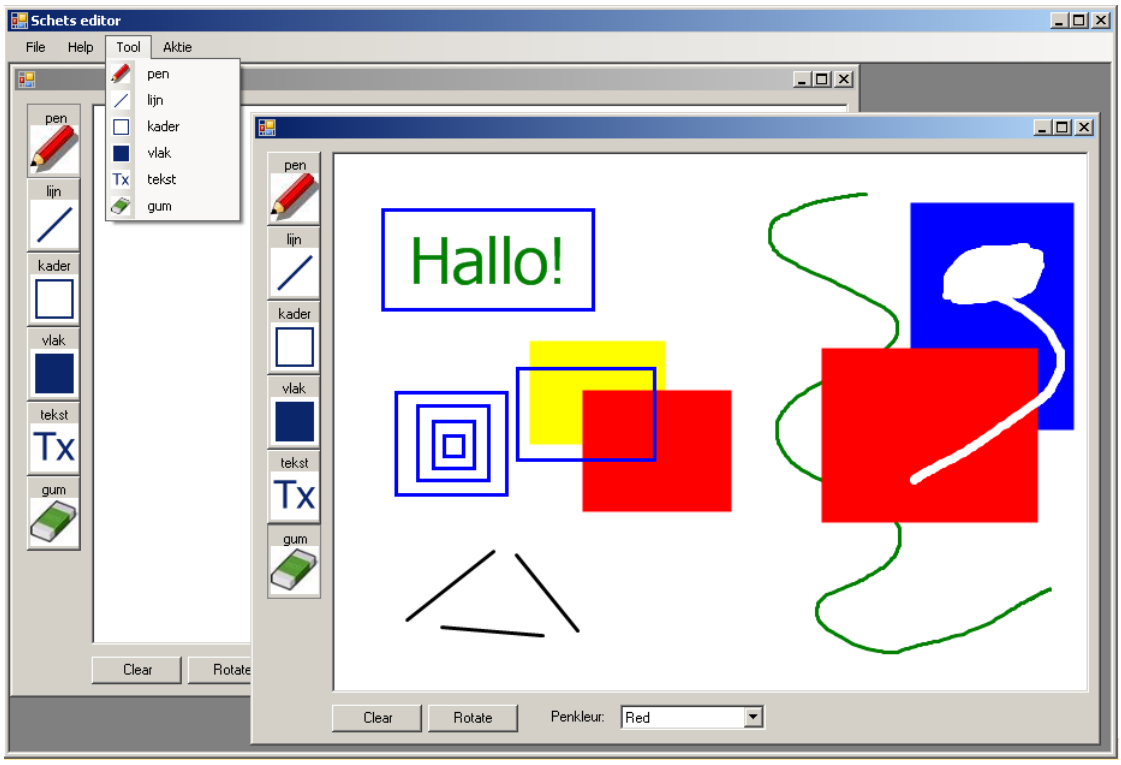
10.5 Het tekenprogramma: “Schets”

Beschrijving van het programma

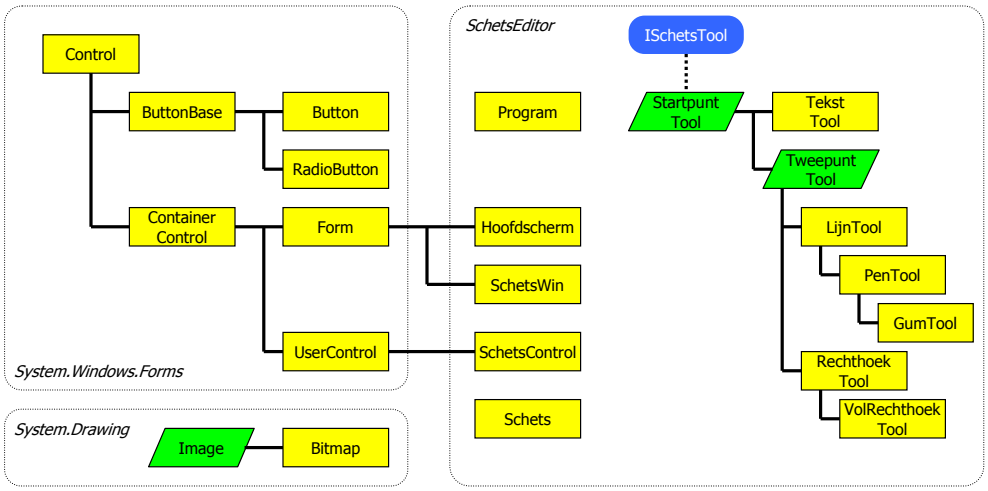
In deze sectie schrijven we een compleet tekenprogramma. In figuur 1 is het programma in werking te zien. Anders dan de bitmap-editor in sectie 10.1 kent dit programma *tools* waarmee je verschillende figuren kunt tekenen: pennetje, lijntje, open en gesloten rechthoekje, en een gummetje om weer te kunnen wissen. Ook is er een tool om tekst toe te voegen aan de tekening.
Behalve de tools zijn er onderin het window nog wat bedieningselementen, die we ter onderscheid maar *akties* zullen noemen: een knop om het plaatje leeg te maken, eentje om hem te draaien, en een combobox om de penkleur uit te kiezen.
Als de gebruiker de rechthoek-tool uitkiest kan hij/zij een blok trekken op het canvas. Het gekozen gebied wordt met een grijze contour aangegeven, en pas bij het loslaten van de muis wordt de rechthoek definitief getekend.
In plaats van met de knoppen langs de linkerrand kunnen de tools ook worden uitgekozen via het uitklapmenu ‘Tool’; de functie van de akties is ook beschikbaar via het menu ‘Aktie’. Zo’n dubbele bediening komt in professionelere programma’s vaak voor, en het is een uitdaging aan de programmeur om dit zonder code-duplicatie voor elkaar te krijgen. In dit geval zijn de methoden die de tool-buttons en de tool-menuitems maken beide geparametriseerd met een collection van de gewenste tools. De methoden die de aktie-button/-menuitems maken zijn geparametriseerd met een array van de gewenste kleuren.
Op deze manier is het gemakkelijk om in een latere versie van het programma nieuwe tools en kleuren toe te voegen, die dan automatisch op twee plaatsen in de userinterface opduiken.



Figuur 1: Het programma Schets in werking

Opzet van het programma

In dit programma zijn we nu eens niet zuinig met klassen. Het programma bestaat uit maar liefst 14 verschillende klassen. In figuur 2 is de subklasse-hiërarchie van deze klassen getekend, en ook hoe ze samenhangen met de bestaande klassen in de library.
Zo'n groot aantal klassen is typerend voor een object-georiënteerd programma. Als je de listings van de aparte klassen leest, zul je zien dat er nergens echt opzienbarende dingen gebeuren. Veel van de klassen zijn ook erg kort, en bevatten maar een paar methoden met een handjevol opdrachten. De magie zit hem in de samenwerking van de klassen.



Figuur 2: De klasse-hiërarchie van het programma Schets

Globale opzet van de klassen

Het programma heeft een MDI-userinterface, en heeft dus dezelfde opbouw als het teksteditor-programma in sectie 10.3.

blz. 17
- De klasse `Program` (listing 11) bevat de methode `Main` waarmee het programma begint. Hier wordt een object gemaakt van het MDI-containerwindow.

blz. 8
- De klasse `Hoofdscherm` (listing 1) modelleert het MDI-containerwindow. De belangrijkste taak hier is het afhandelen van de File-Nieuw menukeuze, waarmee een nieuw MDI-childwindow wordt aangemaakt.

blz. 9
blz. 12
- De klasse `SchetsWin` (listing 2 t/m listing 5) modelleert het MDI-childwindow. Hierin wordt de tekening getoond, maar ook de buttons en menuitems voor de diverse tools en akties.

In het teksteditor-programma werd de eigenlijke tekst getoond in een `TextBox`. Zo gemakkelijk komen we dit keer niet weg, want er is niet een standaard control om een 'schets' te tonen. Daarom maken we een aparte klasse voor dit doel:

blz. 13
- De klasse `SchetsControl` (listing 6) is een subklasse van `Usercontrol`. Het modelleert een control waarmee de gebruiker een 'schets' kan bewerken.

Zoals een `TextBox` gebruikt wordt om een string te bewerken, zo wordt een `SchetsControl` gebruikt om een schets te bewerken. Voor een string bestaat een standaardklasse, maar voor een schets natuurlijk niet. Dus ook hier maken we een eigen klasse:

blz. 14
- De klasse `Schets` (listing 7) modelleert de eigenlijke schets. Dus niet hoe die zichtbaar wordt gemaakt, maar het ding zelf. In de huidige situatie is het een tamelijk kleine klasse, omdat hij dankbaar gebruik maakt van een membervariabele van het type `Bitmap`.

Een hiërarchie van Tool-klassen

Bij het werken met het programma zal de gebruiker een tool uitkiezen, en vervolgens handelingen uitvoeren in de `SchetsControl`. Het indrukken van de muis is belangrijk, daarna kan de muis bewogen worden (met ingedrukte knop), en op een zeker moment wordt de muisknop weer losgelaten. Afhankelijk van de gekozen tool gebeurt er dan iets met de schets.
Wat er precies gebeurt is afhankelijk van de gekozen tool. We maken daarom voor elke tool een

aparte klasse: `LijnTool`, `PenTool`, `RechthoekTool`, `VolRechthoekTool`, `TekstTool` en `GumTool` (listing 8 t/m listing 10). blz. 15
Deze klassen staan in een subklasse-hiërarchie, omdat ze soms functionaliteit van een andere klasse kunnen hergebruiken. Zo is `VolRechthoekTool` een subklasse van `RechthoekTool`, omdat het tekenen van de contour tijdens het *draggen* met de muis hetzelfde is. En `GumTool` is een subklasse van `PenTool` omdat een gum in feite een soort witte pen is. blz. 17
Sommige klassen in de hiërarchie zijn *abstract*, zoals `StartpuntTool` en `TweepuntTool`. Deze klassen hebben een hiërarchisch nut, maar kunnen niet zelfstandig als tool optreden.
Er is een aparte *interface* gedefinieerd waarin we specificeren wat elke tool moet kunnen: reageren op de muis (indrukken, slepen en loslaten) en reageren op het intikken van een letter. De `SchetsControl`-klasse rekent er op dat elke tool voor deze vier dingen methoden beschikbaar heeft. De interface `ISchetsTool` legt vast wat de parameters van die methoden moeten zijn. De klasse `StartpuntTool` komt de belofte van de interface na, al is van sommige methoden de body nog niet ingevuld (die methoden zijn nog abstract). Dieper in de hiërarchie worden echter alle methoden uiteindelijk gedefinieerd, en sommige onderweg ook weer met *override* opnieuw ingevuld.

De tool-klassen
De abstracte klasse `StartpuntTool` werkt een gemenschappelijk kenmerk uit dat voor alle tools van belang is: met de muis klik je een startpunt aan, dat bewaard moet worden om de figuur later definitief te kunnen tekenen. Deze klasse is abstract, want *wat* er dan precies op dat startpunt getekend wordt is in deze klasse nog niet uitgewerkt.
In het programma hebben we twee soorten tools die voortborduren op het idee van de `StartpuntTool`: `TekstTool`, die een tekst toont op het startpunt, en `TweepuntTool` voor de tools die behalve een startpunt ook nog een tweede punt nodig hebben. Die laatste klasse is weer abstract, omdat we nog niet hebben uitgewerkt wat er dan met die twee punten gebeurt.
De klassen `LijnTool` en `RechthoekTool` zijn twee concrete invullingen van het abstracte `TweepuntTool`.
De klasse `VolRechthoekTool` werkt voor de contour hetzelfde als `RechthoekTool`, maar voor het definitief tekenen een beetje anders. Door hem een subklasse van `RechthoekTool` te maken vermijden we duplicatie van de code voor het contour-tekenen.
Zelfs `PenTool` blijkt nog weer wat gemeenschappelijk te hebben met `LijnTool`: een pentekening bestaat immers uit een serie (korte) lijnen. Daarom herdefinieert `PenTool` de methode die correspondeert met muis-slepen: in deze methode simulueert de `PenTool` het loslaten en onmiddellijk weer indrukken van de muistoets, door de overeenkomstige methoden aan te roepen.
Een gum-spoor, tenslotte, is in feite niets anders dan een dikke witte lijn, dus wordt `GumTool` op zijn beurt een subklasse van `PenTool`.

De klasse SchetsWin
De klasse `SchetsWin` is de grootste van het hele project. Dit komt voornamelijk omdat hier de userinterface (tool- en aktie-buttons, menu-items) moet worden opgebouwd.
De constructormethode van deze klasse staat in listing 3. Zoals altijd heeft deze methode als taak om alle controls te maken en te configureren. De belangrijkste control (de `SchetsControl` waarin het plaatje wordt getoond) wordt direct in de constructor gemaakt; voor het aanmaken van de drie menu's en de twee rijen buttons worden hulpmethoden aangeroepen die in de volgende twee listings staan uitgewerkt. blz. 10
Als eerste wordt in de constructormethode echter een array met `ISchetsTool`-objecten aangemaakt. Het type van de elementen van deze array is `ISchetsTool`, dat wil zeggen de interface die alle tools in de hiërarchie implementeren. De elementen van de array zijn objecten van de diverse subklasen uit de hiërarchie: van elke concrete subklasse eentje, om precies te zijn.
Als membervariable in de klasse (regel 14 in listing 2) wordt gedeclareerd: blz. 9

```
ISchetsTool huidigeTool;
```

Het is de bedoeling dat deze variabele steeds de op dat moment uitgekozen tool zal bevatten. De variabele krijgt een andere waarde als de gebruiker een van de knoppen langs de linkerrand indrukt (en ook als de gebruiker een item uit het Tool-menu kiest). Hoe we dat voor elkaar krijgen kun je zien in methode `maakToolMenu` in listing 4 en methode `maakToolButtons` in listing 5. blz. 11
blz. 12

Het Tool-menu in de klasse SchetsWin
De methode `maakToolMenu` (in het midden van listing 4) verwacht als parameter een collection met tools. Hij wordt vanuit de constructor aangeroepen met een *array* van tools als parameter. blz. 11

(Mag dat zomaar? Ja, dat mag: elke array implementeert automatisch de interface `ICollection`). In de body van de methode worden alle 6 de elementen van de array/collection achtereenvolgens verwerkt met een `foreach`-herhaling. Die regel code leest als een gewone Engelse zin: voor elke tool in de collection van tools wordt een menu-item aangemaakt. Belangrijk is hier de regel

```
item.Click += this.klikToolMenu;
```

Daarmee wordt geregeld dat, als het menuitem wordt aangeklikt, de event-handler `klikToolMenu` wordt aangeroepen.

blz. 9 Die methode bevindt zich een stuk eerder in de klasse, afgebeeld in listing 2:

```
private void klikToolMenu(object obj, EventArgs ea)
{
    this.huidigeTool = (ISchetsTool)((ToolStripMenuItem)obj).Tag;
}
```

Hier wordt dus inderdaad een nieuwe waarde aan membervariabele `huidigeTool` toegekend. Maar hoe kunnen we in de event-handler nog achterhalen welke tool correspondeert met het gekozen menuitem? Dat zit subtiel in elkaar. In de event-handler is altijd het `object` dat het event heeft veroorzaakt beschikbaar. In dit geval weten we zeker dat dat een `ToolStripMenuItem` moet zijn, dus kunnen we het object met een *cast* naar dat type converteren: `(ToolStripMenuItem)obj`.
Nu hebben we dus het menuitem te pakken, maar wat is de bijbehorende tool? Kijk nog eens in de methode die het menuitem aanmaakt. Daar staat ook nog:

```
item.Tag = tool;
```

De gewenste tool is dus bewaard in de `Tag`-property van het menuitem. (Wat is dat voor rare property? De property heeft het type `object`, en kan dus gebruikt worden om een willekeurig te kiezen object bij het menuitem te bewaren. Hier komt dat dus goed van pas, want we willen een tool-object bij het menuitem bewaren).
Kijk nu weer naar de event-handler. Daar hebben we met de cast het menuitem te pakken gekregen. Van dat menuitem vragen we de `Tag`-property weer op `((ToolStripMenuItem)obj).Tag`. Let op het extra paar haakjes: de binnenste zijn van de cast, de buitenste omdat we van het *geheel* de `Tag`-property willen pakken. De compiler weet niet beter dan dat die `Tag` van het type `object` is. Maar wij weten dat we daar een `ISchetsTool` in hebeen opgeslagen. Met opnieuw een cast kunnen we de compiler daarvan overtuigen, en al met al leidt dat dus tot de toekenning:

```
this.huidigeTool = (ISchetsTool)((ToolStripMenuItem)obj).Tag;
```

Gebruik van de huidige tool in klasse SchetsWin

In de constructor van `SchetsWin` wordt een membervariabele `schetscontrol` van het type `SchetsControl` geconfigureerd. Hier wordt geregeld dat bij interessante activiteiten van de gebruiker in de `SchetsControl` (bewegen en klikken van de muis, toest indrukken), de momenteel gekozen tool geactiveerd wordt. Dat gebeurt door de aanroep van een van de vier afgesproken methoden, die elke tool beloofd heeft te bezitten: `MuisVast`, `MuisDrag`, `MuisLos` of `Letter`.
Het abonneren op zo'n event ziet er exotisch uit:

```
schetscontrol.MouseDown += (object o, MouseEventArgs mea) =>
                           {   vast=true;
                               huidigeTool.MuisVast(schetscontrol, mea.Location);
                           };
```

We hadden het ook op een andere manier kunnen opschrijven, en dan ziet het er bekender uit:

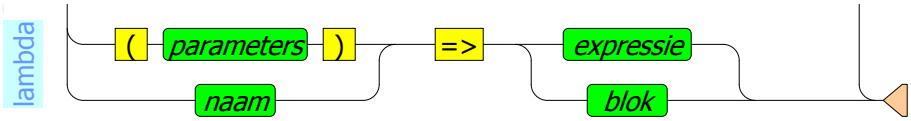
```
schetscontrol.MouseDown += this.muisIngedrukt;
```

Dan had er natuurlijk wel een methode `muisIngedrukt` moeten zijn, met de bekende event-handler parameters:

```
void muisIngedrukt(object o, MouseEventArgs mea)
{   vast=true;
    huidigeTool.MuisVast(schetscontrol, mea.Location);
}
```

Die aanpak hebben we hier echter niet gebruikt, want dan wordt het zo'n zoekplaatje met al die aparte methoden. De gehanteerde aanpak is veel leuker: de hele body van de event-handler staat direct achter het `+=` teken, zonder dat de methode zelfs maar een naam hoeft te krijgen.

Syntax van een lambda-expressie
Zo'n *anonieme methode* staat ook wel bekend als een *lambda-expressie*. Het is, syntactisch gesproken, inderdaad een expressie, en mag daarom op de plek achter `+=` staan. De syntax van een lambda-expressie staat samengevat in figuur 3. Hij is te herkennen aan het speciale symbool `=>` in het midden. Daarachter staat in principe een *blok*, dus een rij opdrachten tussen accolades. Als de enige opdracht in dat blok een `return`-opdracht is, dan mag je de expressie die ge-returnd wordt ook direct achter de `=>` schrijven, dus zonder accolades enzonder `return`. Die notatie hebben we gebruikt in sectie 10.1, waar een anonieme methode als parameter werd meegegeven aan de methode `combineer` (listing 33 op blz. 143).
Voor de `=>` staat in principe de parameterlijst van de methode. Als de methode precies één parameter heeft, mag je de haakjes en het type weglaten. Daarmee kun je de kwadraat-functie super-compact opschrijven als anonieme functie: `x=>x*x`.



Figuur 3: Syntax van een lambda-expressie

Aanroep van tool-methoden in klasse SchetsWin
We gaan verder met de beschouwing van de constructor van `SchetsWin`. De vier events die van belang zijn in het `SchetsControl` (`MouseDown`, `MouseMove`, `MouseUp` en `KeyPress`) worden in feite vertaald in aanroepen van corresponderende tool-methoden (`MuisVast`, `MuisDrag`, `MuiLos` en `Letter`). Maar let op: de `Muis`-methoden zijn *geen* event-handlers; ze hebben immers niet de voor event-handlers karakteristieke twee parameters. Het zijn zelfbedachte methoden, die vanaf vandaag bekend staan als 'schetstool-methoden', want het zijn de methoden die samen de interface `ISchetsTool` vormen. Of deze library een wereldwijde doorbraak gaat beleven valt te bezien, maar de bekendheid is in ieder geval een feit binnen dit project.
De methoden zijn zo ontworpen dat ze ook de `SchetsControl` als parameter krijgen, zodat de tools –als deze methoden te zijner tijd uitgevoerd gaan worden– inderdaad een effect op de tekening kunnen uitoefenen.
Verder is er nog een subtiliteit in de vertaling van `MouseMove`: deze triggert alleen een aanroep van `MuisDrag`, als de muisknop nog ingedrukt is. Dit wordt gecontroleerd door een `bool` variabele `vast`, die bij het afhandelen van `MouseDown` op `true` wordt gezet, en bij `MouseUp` weer op `false`.

Resources
Een laatste aandachtspunt in de klasse `SchetsWin` is nog de manier waarop de icoontjes op de knoppen en menu-items worden gezet. Dat is niet heel moeilijk, want zowel een `Button` als een `MenuItem` hebben een property `Image`, waaraan je een image-object (of een `Bitmap`-object, want dat is een subklasse van `Image`) kunt toekennen. Maar hoe kom je aan een `Image` met een toepasselijk plaatje?
Het eenvoudigste is om de constructor van `Bitmap` te gebruiken, die naar keuze een string met de filenaam of een `Stream` met een reeds geopende file meekrijgt. Dan moeten de files met de plaatjes wel beschikbaar zijn in de directory waarin het programma runt. Dat betekent dat je de plaatjes moet meeleveren als je het programma aflevert. Dat is riskant, want gebruikers zijn gewend dat programma's uitsluitend uit een `.exe`-file bestaan, en raken dit soort extra hulp-bestanden nog wel eens kwijt.
In het programma is daarom een aanpak gehanteerd waarbij de plaatjes worden meeverpakt (*embedded*) in de assembly, en dus uiteindelijk in de `.exe`-file. Zo'n meeverpakt hulpbestand heet een *resource*.
Met Visual Studio kun je eenvoudig interactief resources toevoegen aan een project. Je krijgt dan een bestand `Resources.resx`. Dit is een XML-bestand, waarin de koppeling wordt gemaakt tussen de naam van de resources en de files met de plaatjes. De compiler zoekt de plaatjes op (tijdens het compileren moeten de files van de plaatjes dus wel beschikbaar zijn!) en pakt ze in de assembly in.

Vanuit het programma kun je de resources aanspreken via een `ResourceManager` object. Zo'n ding wordt inderdaad geïnitialiseerd als membervariabele in `SchetsWin`:

```
ResourceManager resourcemanager
    = new ResourceManager("SchetsEditor.Properties.Resources"
                          , Assembly.GetExecutingAssembly()
                          );
```

Daarna kun je de gewenste resources terugvinden via de naam die je er, bij het aanmaken van het `.resx`-bestand, voor gekozen hebt. In dit geval hebben we de 6 resources precies dezelfde naam gegeven als de opschriften van de buttons en menu-items: 'pen', 'lijn', 'kader', 'vlak', 'tekst' en 'gum'. Dit blijkt uit de twee regels bij de opbouw van het menu:

```
item.Text  = tool.ToString();
item.Image = (Image)resourcemanager.GetObject(tool.ToString());
```

Je ziet hier dat de string die als `Text` op het menuitem wordt gezet, tevens de naam is van de resource die via `GetObject` bij de resourcemanager wordt opgevraagd. De cast naar `Image` is nodig omdat `GetObject` ook resources van andere types kan ophalen.
De teksten die corresponderen met de 6 soorten tools zijn afkomstig van een aanroep van `ToString`.
blz. 17 In listing 10 kun je zien dat elke tool-klasse inderdaad de methode `ToString` herdefinieert om z'n eigen naam bekend te maken.
Visual Studio genereert naast het bestand `Resources.resx` ook nog een bestand `Resources.Designer.cs`. Dit is C#-code, waarin voor elke resource een static property wordt aangemaakt met de naam van de resource. In plaats van de opdracht

```
Image plaatje = (Image)resourcemanager.GetObject("pen");
```

is het dan mogelijk om te schrijven:

```
Image plaatje = SchetsEditor.Properties.pen;
```

In het programma laten we die mogelijkheid echter ongebruikt, omdat we in een `for`-opdracht steeds een ander plaatje willen ophalen, waarbij de naam van de resource in een string-variabele beschikbaar is. Dan is de universele methode `GetObject` handiger dan de zes aparte properties.

De klasse SchetsControl

blz. 13 De klasse `SchetsControl` is niet zo groot: hij staat in zijn geheel in listing 6. Het belangrijkste zijn de twee member-variabelen: een `Schets` waarin de eigenlijke schets wordt bewaard, en een `Color` waarin de huidige penkleur wordt bewaard.
De klasse bevat verder een aantal event-handlers. Twee daarvan (`VeranderKleur`) veranderen de penkleur, en vier andere (`Schoon`, `Roteer`, `veranderAfmeting` en `teken`) vertrouwen erop dat het onderliggende `Schets`-object het vuile werk opknapt.
De mysterieuze herdefinitie van `OnPaintBackground` met een lege body maakt dat de achtergrond van de control niet meer uitgewist wordt voorafgaand aan het tekenen. Dat is gewenst, omdat het beeld daardoor stabieler wordt. Dit is alleen verantwoord als het tekenen helemaal 'dekkend' gebeurt, maar dat is hier inderdaad het geval. Het tekenen wordt uitbesteed aan de klasse `Schets`, en die verzorgt het tekenen door een schermvullende bitmap neer te zetten.
Voor het vervolg is nog van belang dat elke `Control` een methode `GetGraphics` heeft, waarmee direct toegang tot het scherm verkregen kan worden. Deze methode wordt geërfd door `UserControl` en dus ook door `SchetsControl`. In hoofdstuk 9 hadden we streng verboden om deze methode te gebruiken, behalve voor het 'tijdelijk' tekenen van figuren. Dat is in dit programma het geval: het grijze kader dat tijdens het 'draggen' met de muis wordt getekend, gebeurt op deze manier.
Dit staat in contrast met het 'definitieve' tekenen. De definitieve tekening moet in de bitmap terecht komen. Voor dit doel schrijven we een methode `MaakBitmapGraphics` die een `Graphics` maakt, die aan de bitmap is gekoppeld. Bij gebruik van die graphics komen de getekende dingen niet op het scherm, maar in de bitmap terecht.
Als dan later het `Paint`-event optreedt, wordt de inhoud van de bitmap alsnog op het scherm gezet.

De klasse Schets

In de klasse `Schets` wordt pas echt de beslissing genomen om de schets als een bitmap te modelleren.
blz. 14 In deze klasse (listing 7) wordt uitgewerkt hoe met behulp van deze bitmap de benodigde methoden `Schoon`, `Roteer`, `Teken` en `VeranderAfmeting` gerealiseerd worden.

De enige methode die wat toelichting behoeft is `VeranderAfmeting`. Als de gewenste afmeting groter is dan de huidige afmeting van de bitmap, dan wordt een grotere bitmap aangemaakt, waarin de huidige gekopieerd wordt. Als de afmeting kleiner wordt, laten we de bitmap echter intact. Dit maakt dat als het window kleiner wordt gemaaakt, het deel van het plaatje 'buiten beeld' toch nog bewaard wordt.

De Tool-hiërarchie

In listing 8 staat om te beginnen de definitie van de interface `ISchetsTool`. Deze zegt in feite dat een tool pas een echte tool is als hij deze vier methoden implementeert. Dit maakt dat we op regel 64, 68, 72 en 75 van listing 3 deze methoden blindelings mogen aanroepen op de `huidigeTool`. blz. 15 blz. 10

Van deze interface kunnen we vervolgens diverse implementaties gaan maken. Het begint met een abstracte klasse `StartpuntTool` die alvast twee methodes invult: `MuisVast` en `MuisLos`. Op het indrukken van de muis reageert zo'n startpunt-tool door dat punt voor later gebruik te bewaren in een speciaal daarvoor gedeclareerde membervariabele, bij `MuisLos` neemt de membervariabele `kwast` de in de `SchetsControl` geldende penkleur aan. De overige twee methoden worden later in de hiërarchie ingevuld, en worden dus voorlopig `abstract` gedefinieerd. Dit is in C# (anders dan in Java) verplicht.

De klasse `TekstTool` is een concrete subklasse van `StartpuntTool`. Bij het slepen van de muis hoeft er bij deze tool niets te gebeuren, maar bij het intikken van een letter moet deze letter worden afgebeeld op de positie van het eerder bewaarde startpunt. Daarna moet de `x`-coordinaat van het startpunt worden verhoogd. Met hoeveel precies hangt af van de eigenschappen van het font, die met `MeasureString` kunnen worden opgevraagd.

In listing 9 gaat het verder met de klasse `TweepuntTool`, een andere uitbreiding van `StartpuntTool`. Deze reageert op het slepen van de muis met het tekenen van een grijze contour. Bij het loslaten van de muis wordt de figuur definitief getekend. Toetsindrukken worden genegeerd. Daarmee zijn alle vier de methoden van de interface `ISchetsTool` ingevuld, maar er zijn twee nieuwe bijgekomen: `Bezig` voor het tekenen van de contour, en `Compleet` voor het tekenen van de definitieve figuur. We kunnen `Compleet` alvast een default-invulling geven, namelijk dat hij ook alleen de contour tekent. Maar hoe je een contour tekent (lijn? rechthoek? cirkel?) kunnen we nog niet vastleggen. Deze methode wordt daarom `abstract` gedeclareerd, en hoeft dus nog geen body te hebben. De prijs is dat ook de klasse daarmee nog `abstract` is. blz. 16

Wel concreet zijn de subklassen `LijnTool` en `RechthoekTool`. Deze geven ieder een eigen invulling aan de ontbrekende methode `Bezig`. De default-invulling dat het definitieve tekenen net zo gebeurt als het voorlopige, is voor deze twee tools in orde.

De klasse `VolRechthoekTool` erft het tekenen van de contour van `RechthoekTool`, maar geeft het definitieve tekenen van een `Compleet` figuur een nieuwe invulling, die het default-gedrag vervangt. De figuur wordt nu ingevuld getekend met behulp van `FillRectangle`.

De implementatie van `PenTool` in listing 10 is subtiel. Deze herdefinieert de methode `MuisDrag`: bij elk verslepen van de muis wordt nu zogenaamd de muis even losgelaten en weer ingedrukt. Die methoden waren al ingevuld met het tekenen van een lijn tot hier, en het beginnen van een nieuw lijnsegment. Het gevolg is dat de pen-tool bij elke muisbeweging (althans met ingedrukte muisknop) een klein lijnstukje tekent en aan een nieuw lijntje gaat beginnen. Dat is precies wat we nodig hebben voor een pen. blz. 17

De klasse `GumTool` tenslotte is weer een verfijning van `PenTool`. Bij het tekenen van de contour (en daarmee ook van de definitieve figuur, want voor lijnen, en dus ook voor pennen, en dus ook voor gummen, is die hetzelfde als de contour) wordt eerst de pendikte 7 uitgekozen, en de tekenkleur wit. Op deze manier doet een gum hetzelfde als een pen, maar dan met een dikke witte lijn in plaats van een dunne gekleurde lijn.

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

namespace SchetsEditor
{
    public class Hoofdscherm : Form
    {
        MenuStrip menuStrip;

        public Hoofdscherm()
        {   this.ClientSize = new Size(800, 600);
            menuStrip = new MenuStrip();
            this.Controls.Add(menuStrip);
            this.maakFileMenu();
            this.maakHelpMenu();
            this.Text = "Schets editor";
            this.IsMdiContainer = true;
            this.MainMenuStrip = menuStrip;
        }
        private void maakFileMenu()
        {   ToolStripDropDownItem menu;
            menu = new ToolStripMenuItem("File");
            menu.DropDownItems.Add("Nieuw", null, this.nieuw);
            menu.DropDownItems.Add("Exit", null, this.afsluiten);
            menuStrip.Items.Add(menu);
        }
        private void maakHelpMenu()
        {   ToolStripDropDownItem menu;
            menu = new ToolStripMenuItem("Help");
            menu.DropDownItems.Add("Over \"Schets\"", null, this.about);
            menuStrip.Items.Add(menu);
        }
        private void about(object o, EventArgs ea)
        {   MessageBox.Show("Schets versie 1.0\n(c) UU Informatica 2010"
                            , "Over \"Schets\""
                            , MessageBoxButtons.OK
                            , MessageBoxIcon.Information
                            );
        }

        private void nieuw(object sender, EventArgs e)
        {   SchetsWin s = new SchetsWin();
            s.MdiParent = this;
            s.Show();
        }
        private void afsluiten(object sender, EventArgs e)
        {   this.Close();
        }
    }
}
```

Listing 1: SchetsEditor/Hoofdscherm.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using System.Reflection;
using System.Resources;

namespace SchetsEditor
{
    public class SchetsWin : Form
    {
        MenuStrip menuStrip;
        SchetsControl schetscontrol;
        ISchetsTool huidigeTool;
        Panel paneel;
        bool vast;
        ResourceManager resourcemanager
            = new ResourceManager("SchetsEditor.Properties.Resources"
                                  , Assembly.GetExecutingAssembly()
                                  );

        private void veranderAfmeting(object o, EventArgs ea)
        {
            schetscontrol.Size = new Size ( this.ClientSize.Width  - 70
                                          , this.ClientSize.Height - 50);
            paneel.Location = new Point(64, this.ClientSize.Height - 30);
        }

        private void klikToolMenu(object obj, EventArgs ea)
        {
            this.huidigeTool = (ISchetsTool)((ToolStripMenuItem)obj).Tag;
        }

        private void klikToolButton(object obj, EventArgs ea)
        {
            this.huidigeTool = (ISchetsTool)((RadioButton)obj).Tag;
        }

        private void afsluiten(object obj, EventArgs ea)
        {
            this.Close();
        }
```

Listing 2: SchetsEditor/SchetsWin.cs, deel 1 van 4

```
        public SchetsWin()
        {
            ISchetsTool[] deTools = { new PenTool()
                                    , new LijnTool()
                                    , new RechthoekTool()
                                    , new VolRechthoekTool()
                                    , new TekstTool()
                                    , new GumTool()
                                    };
            String[] deKleuren = { "Black", "Red", "Green", "Blue"
                                 , "Yellow", "Magenta", "Cyan"
                                 };

            this.ClientSize = new Size(700, 500);
            huidigeTool = deTools[0];

            schetscontrol = new SchetsControl();
            schetscontrol.Location = new Point(64, 10);
            schetscontrol.MouseDown += (object o, MouseEventArgs mea) =>
                                       {   vast=true;
                                           huidigeTool.MuisVast(schetscontrol, mea.Location);
                                       };
            schetscontrol.MouseMove += (object o, MouseEventArgs mea) =>
                                       {   if (vast)
                                           huidigeTool.MuisDrag(schetscontrol, mea.Location);
                                       };
            schetscontrol.MouseUp   += (object o, MouseEventArgs mea) =>
                                       {   if (vast)
                                           huidigeTool.MuisLos (schetscontrol, mea.Location);
                                           vast = false;
                                       };
            schetscontrol.KeyPress +=  (object o, KeyPressEventArgs kpea) =>
                                       {   huidigeTool.Letter  (schetscontrol, kpea.KeyChar);
                                       };
            this.Controls.Add(schetscontrol);

            menuStrip = new MenuStrip();
            menuStrip.Visible = false;
            this.Controls.Add(menuStrip);
            this.maakFileMenu();
            this.maakToolMenu(deTools);
            this.maakAktieMenu(deKleuren);
            this.maakToolButtons(deTools);
            this.maakAktieButtons(deKleuren);
            this.Resize += this.veranderAfmeting;
            this.veranderAfmeting(null, null);
        }
```

Listing 3: SchetsEditor/SchetsWin.cs, deel 2 van 4

```
        private void maakFileMenu()
        {
            ToolStripMenuItem menu = new ToolStripMenuItem("File");
            menu.MergeAction = MergeAction.MatchOnly;
            menu.DropDownItems.Add("Sluiten", null, this.afsluiten);
            menuStrip.Items.Add(menu);
        }

        private void maakToolMenu(ICollection<ISchetsTool> tools)
        {
            ToolStripMenuItem menu = new ToolStripMenuItem("Tool");
            foreach (ISchetsTool tool in tools)
            {   ToolStripItem item = new ToolStripMenuItem();
                item.Tag = tool;
                item.Text = tool.ToString();
                item.Image = (Image)resourcemanager.GetObject(tool.ToString());
                item.Click += this.klikToolMenu;
                menu.DropDownItems.Add(item);
            }
            menuStrip.Items.Add(menu);
        }

        private void maakAktieMenu(String[] kleuren)
        {
            ToolStripMenuItem menu = new ToolStripMenuItem("Aktie");
            menu.DropDownItems.Add("Clear", null, schetscontrol.Schoon );
            menu.DropDownItems.Add("Roteer", null, schetscontrol.Roteer );
            ToolStripMenuItem submenu = new ToolStripMenuItem("Kies kleur");
            foreach (string k in kleuren)
                submenu.DropDownItems.Add(k, null, schetscontrol.VeranderKleurViaMenu);
            menu.DropDownItems.Add(submenu);
            menuStrip.Items.Add(menu);
        }
```

Listing 4: SchetsEditor/SchetsWin.cs, deel 3 van 4

```
125
        private void maakToolButtons(ICollection<ISchetsTool> tools)
        {
            int t = 0;
            foreach (ISchetsTool tool in tools)
130         {
                RadioButton b = new RadioButton();
                b.Appearance = Appearance.Button;
                b.Size = new Size(45, 62);
                b.Location = new Point(10, 10 + t * 62);
135             b.Tag = tool;
                b.Text = tool.ToString();
                b.Image = (Image)resourcemanager.GetObject(tool.ToString());
                b.TextAlign = ContentAlignment.TopCenter;
                b.ImageAlign = ContentAlignment.BottomCenter;
140             b.Click += this.klikToolButton;
                this.Controls.Add(b);
                if (t == 0) b.Select();
                t++;
            }
145     }

        private void maakAktieButtons(String[] kleuren)
        {
            paneel = new Panel();
150         paneel.Size = new Size(600, 24);
            this.Controls.Add(paneel);

            Button b; Label l; ComboBox cbb;
            b = new Button();
155         b.Text = "Clear";
            b.Location = new Point(  0, 0);
            b.Click += schetscontrol.Schoon;
            paneel.Controls.Add(b);

160         b = new Button();
            b.Text = "Rotate";
            b.Location = new Point( 80, 0);
            b.Click += schetscontrol.Roteer;
            paneel.Controls.Add(b);
165
            l = new Label();
            l.Text = "Penkleur:";
            l.Location = new Point(180, 3);
            l.AutoSize = true;
170         paneel.Controls.Add(l);

            cbb = new ComboBox(); cbb.Location = new Point(240, 0);
            cbb.DropDownStyle = ComboBoxStyle.DropDownList;
            cbb.SelectedValueChanged += schetscontrol.VeranderKleur;
175         foreach (string k in kleuren)
                cbb.Items.Add(k);
            cbb.SelectedIndex = 0;
            paneel.Controls.Add(cbb);
        }
180 }
```

Listing 5: SchetsEditor/SchetsWin.cs, deel 4 van 4

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace SchetsEditor
{   public class SchetsControl : UserControl
    {   private Schets schets;
        private Color penkleur;

        public Color PenKleur
        { get { return penkleur; }
        }
        public Schets Schets
        { get { return schets;   }
        }
        public SchetsControl()
        {   this.BorderStyle = BorderStyle.Fixed3D;
            this.schets = new Schets();
            this.Paint += this.teken;
            this.Resize += this.veranderAfmeting;
            this.veranderAfmeting(null, null);
        }
        protected override void OnPaintBackground(PaintEventArgs e)
        {
        }
        private void teken(object o, PaintEventArgs pea)
        {   schets.Teken(pea.Graphics);
        }
        private void veranderAfmeting(object o, EventArgs ea)
        {   schets.VeranderAfmeting(this.ClientSize);
            this.Invalidate();
        }
        public Graphics MaakBitmapGraphics()
        {   Graphics g = schets.BitmapGraphics;
            g.SmoothingMode = SmoothingMode.AntiAlias;
            return g;
        }
        public void Schoon(object o, EventArgs ea)
        {   schets.Schoon();
            this.Invalidate();
        }
        public void Roteer(object o, EventArgs ea)
        {   schets.VeranderAfmeting(new Size(this.ClientSize.Height, this.ClientSize.Width));
            schets.Roteer();
            this.Invalidate();
        }
        public void VeranderKleur(object obj, EventArgs ea)
        {   string kleurNaam = ((ComboBox)obj).Text;
            penkleur = Color.FromName(kleurNaam);
        }
        public void VeranderKleurViaMenu(object obj, EventArgs ea)
        {   string kleurNaam = ((ToolStripMenuItem)obj).Text;
            penkleur = Color.FromName(kleurNaam);
        }
    }
}
```

Listing 6: SchetsEditor/SchetsControl.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;

namespace SchetsEditor
{
    public class Schets
    {
        private Bitmap bitmap;

        public Schets()
        {
            bitmap = new Bitmap(1, 1);
        }
        public Graphics BitmapGraphics
        {
            get { return Graphics.FromImage(bitmap); }
        }
        public void VeranderAfmeting(Size sz)
        {
            if (sz.Width > bitmap.Size.Width || sz.Height > bitmap.Size.Height)
            {
                Bitmap nieuw = new Bitmap( Math.Max(sz.Width,  bitmap.Size.Width)
                                         , Math.Max(sz.Height, bitmap.Size.Height)
                                         );
                Graphics gr = Graphics.FromImage(nieuw);
                gr.FillRectangle(Brushes.White, 0, 0, sz.Width, sz.Height);
                gr.DrawImage(bitmap, 0, 0);
                bitmap = nieuw;
            }
        }
        public void Teken(Graphics gr)
        {
            gr.DrawImage(bitmap, 0, 0);
        }
        public void Schoon()
        {
            Graphics gr = Graphics.FromImage(bitmap);
            gr.FillRectangle(Brushes.White, 0, 0, bitmap.Width, bitmap.Height);
        }
        public void Roteer()
        {
            bitmap.RotateFlip(RotateFlipType.Rotate90FlipNone);
        }
    }
}
```

Listing 7: SchetsEditor/Schets.cs

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;

namespace SchetsEditor
{
    public interface ISchetsTool
    {
        void MuisVast(SchetsControl s, Point p);
        void MuisDrag(SchetsControl s, Point p);
        void MuisLos(SchetsControl s, Point p);
        void Letter(SchetsControl s, char c);
    }

    public abstract class StartpuntTool : ISchetsTool
    {
        protected Point startpunt;
        protected Brush kwast;

        public virtual void MuisVast(SchetsControl s, Point p)
        {   startpunt = p;
        }
        public virtual void MuisLos(SchetsControl s, Point p)
        {   kwast = new SolidBrush(s.PenKleur);
        }
        public abstract void MuisDrag(SchetsControl s, Point p);
        public abstract void Letter(SchetsControl s, char c);
    }

    public class TekstTool : StartpuntTool
    {
        public override string ToString() { return "tekst"; }

        public override void MuisDrag(SchetsControl s, Point p) { }

        public override void Letter(SchetsControl s, char c)
        {
            if (c >= 32)
            {
                Graphics gr = s.MaakBitmapGraphics();
                Font font = new Font("Tahoma", 40);
                string tekst = c.ToString();
                SizeF sz =
                gr.MeasureString(tekst, font, this.startpunt, StringFormat.GenericTypographic);
                gr.DrawString   (tekst, font, kwast,
                                            this.startpunt, StringFormat.GenericTypographic);
                // gr.DrawRectangle(Pens.Black, startpunt.X, startpunt.Y, sz.Width, sz.Height);
                startpunt.X += (int)sz.Width;
                s.Invalidate();
            }
        }
    }
```

Listing 8: SchetsEditor/Tools.cs, deel 1 van 3

```
    public abstract class TweepuntTool : StartpuntTool
55  {
        public static Rectangle Punten2Rechthoek(Point p1, Point p2)
        {   return new Rectangle( new Point(Math.Min(p1.X,p2.X), Math.Min(p1.Y,p2.Y))
                                , new Size (Math.Abs(p1.X-p2.X), Math.Abs(p1.Y-p2.Y))
                                );
60      }
        public static Pen MaakPen(Brush b, int dikte)
        {   Pen pen = new Pen(b, dikte);
            pen.StartCap = LineCap.Round;
            pen.EndCap = LineCap.Round;
65          return pen;
        }
        public override void MuisVast(SchetsControl s, Point p)
        {   base.MuisVast(s, p);
            kwast = Brushes.Gray;
70      }
        public override void MuisDrag(SchetsControl s, Point p)
        {   s.Refresh();
            this.Bezig(s.CreateGraphics(), this.startpunt, p);
        }
75      public override void MuisLos(SchetsControl s, Point p)
        {   base.MuisLos(s, p);
            this.Compleet(s.MaakBitmapGraphics(), this.startpunt, p);
            s.Invalidate();
        }
80      public override void Letter(SchetsControl s, char c)
        {
        }
        public abstract void Bezig(Graphics g, Point p1, Point p2);

85      public virtual void Compleet(Graphics g, Point p1, Point p2)
        {   this.Bezig(g, p1, p2);
        }
    }

90  public class RechthoekTool : TweepuntTool
    {
        public override string ToString() { return "kader"; }

        public override void Bezig(Graphics g, Point p1, Point p2)
95      {   g.DrawRectangle(MaakPen(kwast,3), TweepuntTool.Punten2Rechthoek(p1, p2));
        }
    }

    public class VolRechthoekTool : RechthoekTool
100 {
        public override string ToString() { return "vlak"; }

        public override void Compleet(Graphics g, Point p1, Point p2)
        {   g.FillRectangle(kwast, TweepuntTool.Punten2Rechthoek(p1, p2));
105     }
    }
```

Listing 9: SchetsEditor/Tools.cs, deel 2 van 3

```
    public class LijnTool : TweepuntTool
    {
        public override string ToString() { return "lijn"; }

        public override void Bezig(Graphics g, Point p1, Point p2)
        {   g.DrawLine(MaakPen(this.kwast,3), p1, p2);
        }
    }

    public class PenTool : LijnTool
    {
        public override string ToString() { return "pen"; }

        public override void MuisDrag(SchetsControl s, Point p)
        {   this.MuisLos(s, p);
            this.MuisVast(s, p);
        }
    }

    public class GumTool : PenTool
    {
        public override string ToString() { return "gum"; }

        public override void Bezig(Graphics g, Point p1, Point p2)
        {   g.DrawLine(MaakPen(Brushes.White, 7), p1, p2);
        }
    }
}
```

Listing 10: SchetsEditor/Tools.cs, deel 3 van 3

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

namespace SchetsEditor
{
    static class Program
    {
        [STAThread]
        static void Main()
        {
            Application.Run(new Hoofdscherm());
        }
    }
}
```

Listing 11: SchetsEditor/Program.cs

Bijlage A

Prakticumopgaven

A.3 SchetsPlus

Anders dan de vorige opdrachten gaat het dit keer om de uitbreiding van een bestaand programma. Je hoeft dus niet *from sratch* te beginnen, maar kunt het programma 'Schets' uit hoofdstuk 10.5 als uitgangspunt gebruiken. Dat bestaat uit 6 C#-files, waarin 11 klassen, 2 abstracte klassen en een interface worden gedefinieerd.
Sommige van die klassen kun je onveranderd laten. In sommige klassen zul je methoden moeten toevoegen, of bestaande methoden veranderen (extra opdrachten en/of extra parameters). Ook zul je waarschijnlijk nog enkele klassen moeten toevoegen. Om het voor anderen mogelijk te maken snel een overzicht te krijgen van de uitbreidingen die je hebt gedaan, moet je bij het programma ook een overzicht van de gedane wijzigingen maken. Dat is een tekstfile, waarin je opsomt in welke methoden van welke klassen er iets is veranderd, waar zinvol met een korte motivatie waarom deze verandering noodzakelijk was. De toelichting mag een Word- (.doc of .docx) of PDF-bestand zijn. De file die je inlevert moet een ZIP- of RAR-file zijn, met daarin alle source-bestanden (de ongewijzigde en de gewijzigde en de toegevoegde), resource-bestanden (de ongewijzigde en de gewijzigde en de toegevoegde), en de project-file (VS2008 of VS2010, maar niet allebei). Gegenereerde binaries en executables (de bin- en obj-directories) hoeven niet te worden meeverpakt.

Beoordelingsnormen
Bij de beoordeling van het programma wordt natuurlijk gelet op correcte werking. Maar daarnaast wordt ook gelet op de stijl van het programma. In het bijzonder gaat het ditmaal om het aanhouden van het *once and only once* principe, dus het vermijden van het kopieren en plakken van stukken code. Het gebruik van methoden met geschikte parameters, en het gebruik van overerving in subklassen (inheritance) kunnen daarbij helpen. Ook het zinvol inzetten van klassen uit de C#-bibliotheek (de Collection-hierarchie, File-I/O enz.) kan leiden tot het vermijden van dubbel werk en draagt bij aan de duidelijkheid van het programma; dit wordt dus positief gewaardeerd. Bovendien geldt het als fraai als variabelen waar mogelijk lokaal worden gedeclareerd en/of als parameter doorgegeven aan methoden, zodat de declaraties bovenin de klasse beperkt blijven tot variabelen die inderdaad essentieel zijn voor de toestand van het object. Tenslotte wordt ook de duidelijkheid van de toelichtings-tekst wordt ook de beoordeling betrokken.

Als de onderdelen 1 t/m 4 hieronder goed werken, de code is netjes opgebouwd en een duidelijke beschrijving is toegevoegd, dan kun je het cijfer 8 halen. Voor een hoger cijfer kun je één of meer van de extra's onder punt 5 inbouwen.

1. Cirkels
Voeg twee tools toe voor het tekenen van een open en een gevulde ovaal. De nieuwe tools zijn natuurlijk zowel via de toolbox, als via het Tool-menu beschikbaar. Hint: laat je inspireren door de manier waarop de rechthoeken in het bestaande programma worden aangepakt.

2. Opslaan en teruglezen
Maak nieuwe menu-items om het plaatje op te slaan en weer in te lezen. De file moet worden opgeslagen in een bitmap-formaat, waarbij de gebruiker uit een aantal gangbare filetypen moet kunnen kiezen (bijvoorbeeld PNG, JPG en BMP).
Zorg er bovendien voor dat de gebruiker een waarschuwing krijgt als hij een schets-window afsluit terwijl er wijzigingen zijn gedaan sinds de laatste keer dat het plaatje werd opgeslagen of ingelezen.

3. Het nieuwe gummen
De gum-tool werkt in het basisprogramma door eigenlijk door het tekenen van een dikke witte lijn over de bestaande tekening heen, waardoor het lijkt alsof die verdwijnt. We gaan het gedrag van de gum nu veranderen, zo dat de gebruiker een getekend element (bijvoorbeeld een rechthoek) in n keer kan uitwissen, door het met de 'nieuwe gum' aan te klikken. Niet alleen verdwijnt zo'n element dan, maar ook komen andere elementen die geheel of gedeeltelijk door het weggehaalde element werden bedekt weer tevoorschijn. (Met de 'oude gum' was dat niet mogelijk: als je daarmee een rechthoek wegveegde, bleef er slechts een witte vlek over).
Het is niet nodig om een nieuwe icon te maken; je kunt simpelweg de functionaliteit van de gum veranderen. De oude manier van gummen komt gewoon te vervallen.
Aanpak: Vanwege de mogelijkheid dat bedekte elementen weer tevoorschijn kunnen komen, is het noodzakelijk om ook 'onzichtbare' (want bedekte) elementen nog te onthouden. Met een bitmap, zoals in het basisprogramma, is dat niet mogelijk: weg is weg. In theorie is het mogelijk om bij elk getekend element een kopie van de oude bitmap te maken, althans van het bedekte gedeelte. Dat vreet echter wel erg veel geheugen, en het is dan ook niet de aanpak die hier wordt gevraagd. Een betere aanpak is om de hele tekening, behalve als bitmap, ook nog te bewaren in de vorm van een lijst van getekende elementen. De objecten in die lijst vormen een compacte beschrijving van elk getekend element (soort, beginpunt, eindpunt, kleur, eventuele tekst). Bij elke teken-aktie wordt de lijst uitgebreid. De bitmap uit het basisprogramma mag ook blijven, en voldoet prima voor de meeste teken-akties. De 'nieuwe gum'-aktie werkt echter door het te verwijderen element uit de lijst te halen, en aan de hand van die lijst de hele bitmap te reconstrueren, door alle overgebleven elementen in de lijst opnieuw te tekenen.
Vrijheid: Wat je precies als elementen beschouwt, die met één klik kunnen worden uitgewist, mag je zelf bedenken. Het ligt voor de hand dat een rechthoek en een ovaal in zijn geheel als één element te beschouwen. Maar of je een met de pen-tool gemaakte kromme lijn als één element, of als allemaal losse lijntjes beschouwt, mag je zelf kiezen. Hetzelfde geldt voor een tekst: is dat in zijn geheel één element, of is elke aparte letter een apart element? (Zet de gemaakte keuze in de toelichtings-tekst!)
Hints:

- Het model van de tekening zal moeten worden uitgebreid: naast de al bestaande bitmap komt de elementen-lijst erbij, met een methode die de bitmap uit de elementenlijst kan reconstrueren.
- Het is ook niet raar als de methode die een figuur tekent ingrijpend aangepast moet worden aan het aldus gewijzigde model.
- Wanneer heeft een gebruiker 'raak' geklikt op een element van de tekening? Bij een gevulde rechthoek is dat duidelijk. Voor de overige elementen kun je in eerste instantie het programma uittesten door alles binnen de *bounding box* van zo'n figuur als 'raak' te beschouwen. Maar dat kan natuurlijk ook subtieler. Bij de gevulde cirkel kun je met een Pythagoras-achtige formule zorgen dat daadwerkelijk het gekleurde deel van de cirkel aangeklikt moet worden. En bij een open rechthoek? Het hele binnengebied als 'raak' beschouwen is wel erg royaal. Maar om precies de rand te raken moet de gebruiker wel erg goed mikken. Je zou ervoor kunnen zorgen dat een zone van -zeg- vijf pixels rond de rand als 'raak' telt. Licht hoe dan ook de gemaakte keuze toe in de toelichtings-tekst.
- Wanneer is er raak geklikt op een schuine lijn? Nou, bijvoorbeeld als de afstand van het geklikte punt tot de lijn niet te groot is. Hoe bepaal je de afstand van een punt tot een lijn? Tik 'distance point line' in op Google, en je bent al snel op een pagina die de formule daarvoor geeft. Laat je niet afschrikken door de wiskunde eromheen: de benodigde formule staat er gewoon tussen. Vermeld de gebruikte bron in de toelichting en in commentaar in het programma!
- Het zou handig zijn als er een methode is die bepaalt of een punt 'raak' is voor een bepaald type element. Die kun je in eerste instantie simpel houden, om snel te kunnen testen of de rest van het programma werkt; daarna kun je de 'raak'-methode naar believen subtieler maken.

4. Het nieuwe Opslaan en teruglezen
Als de gebruiker een ingewikkelde tekening heeft gemaakt, wil hij daar misschien later weer mee verder werken, waarbij dan nog steeds de elementen als objecten te verwijderen zijn.

Daartoe is het nodig dat bij het opslaan/inlezen ook gekozen kan worden uit een speciaal 'schets'-formaat. Het plaatje wordt nu dus *niet* als bitmap opgeslagen, maar op een manier (bijvoorbeeld als tekstfile) waaruit de opbouw van het plaatje gereconstrueerd kan worden: 'lijntje hier, rechthoek daar, tekstje in het midden'.
Hint: De file mag een gewone tekst-file zijn, bijvoorbeeld met voor elk element een regel. De vorm van de file mag je helemaal zelf bepalen: enige vereiste is dat je eigen programma een geschreven file weer kan inlezen. Je hoeft je dus niet te conformeren aan bestaande standaarden. (Een voorbeeld kun je vinden in de manier waarop 'Steden' en 'Wegen' in een file worden opgeslagen in het Zoeknetwerk-programma in hoofdstuk 11.3).
Als de onderdelen 1 t/m 4 hierboven goed werken, de code is netjes opgebouwd en een duidelijke beschrijving is toegevoegd, dan kun je het cijfer 8 halen. Voor een hoger cijfer moet je één of meer van onderstaande extra's inbouwen. (Beschrijf de gemaakte keuzes ook in de toelichting).

5. Extra's

1. Voeg een 'Undo'-knop toe aan het control-panel en/of een menu. Daarmee kan de gebruiker het laatst getekende element weer weghalen, en door herhaaldelijk gebruik de hele tekening weer langzaam afbreken. Misschien kun je zelfs ook nog een 'Redo'-knop maken,die de weggehaalde elementen toch weer toevoegt. Hint: In de bitmap-representatie is dit vrijwel onmogelijk, maar met behulp van de elementen-lijst, die voor onderdeel 2 sowieso nodig is, is het tamelijk eenvoudig.

2. Maak naast de kleur-control ook een control waarmee de gebruiker de lijndikte kan instellen. (En als je dan ook nog kleur 'Wit' aan de kleur-control toevoegt, hebben we de mogelijkheid van de oude gum terug...)

3. Zorg dat de gebruiker uit veel meer kleuren dan de 7 standaardkleuren kan kiezen.

4. Maak een extra tool waarmee je een aangeklikt element 'bovenop', of juist 'onderop' de stapel van getekende elementen legt. Een half-bedekte rechthoek kun je daarmee dus weer helemaal zichtbaar maken.

5. Maak een extra tool waarmee je een element kunt aanklikken, waarna je het door te 'draggen' naar een andere plaats kunt slepen. Uiteraard kunnen daarbij achterliggende elementen verschijnen of juist bedekt worden.

6. Als je het window groter of kleiner maakt, reageert het basisprogramma door de bitmap van de tekening groter of kleiner te maken. De getekende elementen blijven echter even groot. Je zou het echter ook zo kunnen maken, dat de getekende elementen meegroeien, respectievelijk krimpen, met de bitmap.

7. Verzin zelf een zinvolle uitbreiding van het programma.