Uitwerkingen Derde deeltentamen **Imperatief programmeren**
Vrijdag 11 november 2011, 8.30-10.30 uur

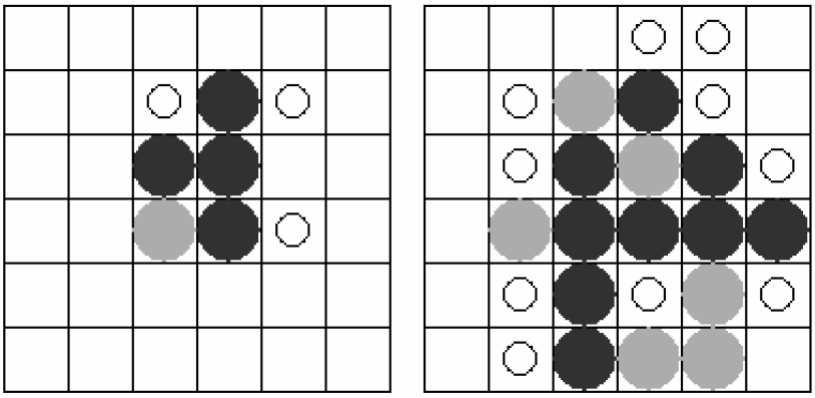
1. *(telt voor 20%)*

   Bij het spel 'Reversi' leggen twee spelers om de beurt een gekleurde steen op een veld van een rechthoekig speelbord.

   Een steen mag alleen maar worden neergelegd op een veld als het veld nog leeg is, en met deze zet een rij van een of meer stenen van de andere kleur wordt ingesloten tussen de nieuwe steen en een al op het bord liggende steen van de eigen kleur.

   De ingesloten stenen kunnen in acht mogelijke richtingen naast de nieuwe steen liggen: horizontaal, verticaal of diagonaal. Stenen insluiten in meerdere richtingen mag ook.

   In de figuur is voor twee voorbeelden met open cirkels aangegeven op welke velden de speler met de lichte stenen mag zetten.



   Als gevolg van een zet veranderen alle ingesloten stenen van kleur. In een programma wordt de situatie opgeslagen in een twee-dimensionale array:

```
int bord[,] = new int[6,6];
```

   Lege velden zijn gecodeerd met 0, gevulde velden met 1 of -1 voor de twee kleuren.

   Gegeven zijn verder volgende methoden (die hoef je dus niet te schrijven):

```
public bool mag  (int kleur, int x, int y)
public bool sluit(int kleur, int x, int y, int dx, int dy)
```

   De methode `sluit` test of speler `kleur`, door te zetten op veld (x, y), n of meer stenen van de tegenstander insluit in de richting (dx, dy), waarbij dx en dy 1, 0 of 1 zijn. De methode `mag` bepaalt of speler kleur mag zetten op veld (x, y).

   De opgave: schrijf nu een methode

```
public void doeZet (int kleur, int x, int y)
```

   die, als dat is toegestaan, de zet voor speler `kleur` op veld (x, y) uitvoert.

2. *(telt voor 45%)*

   Het programma in deze opgave leest een bestand in, waarvan de naam in de constante `filenaam` van de klasse `KaartControl` staat. In dat bestand wordt de ligging van een aantal steden vastgelegd. Het programma toont deze steden op het scherm als zwarte vierkantjes met de naam ernaast.

   De gebruiker daarna overal in het scherm klikken. Op die plekken ontstaat dan een extra stad, met als naam wat er op dat moment in een textbox onderin het scherm is ingevuld. Deze nieuwe steden hebben een *rood* vierkantje.

   In onderdeel (f) passen we het programma aan, zo dat fde gebruiker met de rechter-muisknop ook wegen tussen steden kan toevoegen.

   In de bijlage aan het eind van dit tentamen staat een deel van het programma gegeven. Het bestaat uit vier klassen: `Hoofdscherm`, `Stad`, `Weg`, en `KaartControl`. Een aantal onderdelen moet nog worden

ingevuld. Let op dat sommige variabelen `private` zijn gedeclareerd: die mogen van buiten de klasse niet worden gebruikt. Soms is er ook een corresponderende `public` member, waarvan de naam dan met een hoofdletter begint.

(a) In de constructor van `Hoofdscherm` wordt een property van een `KaartControl`-object gewijzigd. In de methode `Muisklik` van de klasse `KaartControl` wordt juist een property van een `Hoofdscherm`-object opgevraagd. Schrijf deze twee properties.

Antwoord:

```
// in Hoofdscherm:
    public string DeNaam
    {
        get { return naambox.Text; }
    }
// in KaartControl:
    public Hoofdscherm Hoofd
    {
        set { this.hoofd = value; }
    }
```

(b) Schrijf de constructormethode van `KaartControl`. Let op de membervariabelen die nog een waarde moeten krijgen, event-handlers, en het inlezen van de startsituatie.

Antwoord:

```
    public KaartControl()
    {
        this.steden = new List<Stad>();
        this.Lees(filenaam);
        this.Paint += this.Teken;
        this.MouseClick += this.Muisklik;
    }
```

(c) Schrijf het ontbrekende deel van de methode `Lees` van de klasse `KaartControl`, die de steden inleest vanuit een bestand, waarvan de naam als parameter wordt meegegeven. Je mag zonder controle aannemen dat het bestand bestaat en goed leesbaar is.
De regels van dat bestand bevatten steeds twee getallen (de x- en y-coordinaat) en precies één woord, dus regels zoals

```
250 110 Amsterdam
280 180 Utrecht
```

Antwoord:

```
    public void Lees(string naam)
    {
        TextReader invoer = new StreamReader(naam);
        string regel;
        while ((regel = invoer.ReadLine()) != null)
        {
            string[] woorden = regel.Split();
            this.steden.Add(new Stad( new Point(int.Parse(woorden[0]), int.Parse(woorden[1]))
                                    , woorden[2]
                                    , Color.Black));
        }
    }
```

(d) Waarom heeft de auteur van `KaartControl` de methoden `Muisklik` en `Teken` gedefinieerd als `virtual` methoden?

(e) Schrijf een methode `ZoekStad` in de klasse `KaartControl` met als parameter een punt, die een `Stad` oplevert waarvan het vierkantje dat punt omvat. Als het punt met geen enkele stad corespondeert, dan wordt een neutrale waarde teruggegeven.

Antwoord:

```
public Stad ZoekStad(Point p)
{
    foreach (Stad s in steden)
        if (s.Raak(p))
            return s;
    return null;
}
```

(f) We gaan het programma nu aanpassen. In de constructor van de klasse `Hoofdscherm` wordt de initialisatie van variabele `kaart` vervangen door

```
kaart = new WegenkaartControl();
```

Schrijf de klasse `WegenkaartControl` zodat het programma zich als volgt gaat gedragen: als de gebruiker met de *rechter* muisknop een stad aanklikt, en nog een, dan worden die twee steden door een lijn verbonden. (Als de gebruiker mis klikt, dus niet op het vierkantje van een stad, dan gebeurt er niets, en mag hij het nog eens proberen).

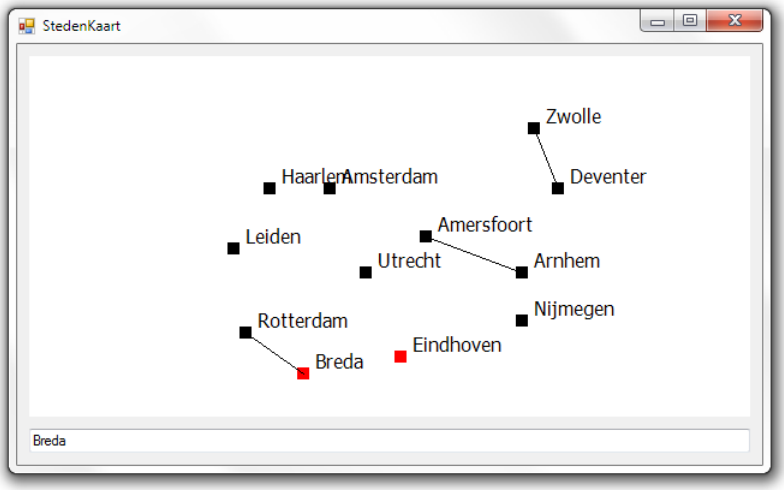
Antwoord:

```
public class WegenkaartControl : KaartControl
{
    IList<Weg> lijnen = new List<Weg>();
    Stad start;

    public override void Teken(object o, PaintEventArgs pea)
    {
        base.Teken(o, pea);

        foreach (Weg l in lijnen)
            pea.Graphics.DrawLine(Pens.Black, l.st1.plek, l.st2.plek);
    }

    public override void Muisklik(object o, MouseEventArgs mea)
    {
        if (mea.Button == MouseButtons.Right)
        {
            Stad s = ZoekStad(mea.Location);
            if (start==null)
                start = s;
            else if (s!=null)
            {   lijnen.Add(new Weg(start, s));
                start = null;
            }
            this.Invalidate();
        }
        else base.Muisklik(o, mea);
    }
}
```



3. *(telt voor 35%)*

(a)
- Wat is een abstracte methode?
- Wat is een abstracte klasse?
- Van een abstracte klasse kun je geen nieuwe objecten maken met een `new`-expressie. Waarom kan het toch zinvol zijn om een abstracte klasse te schrijven?

Antwoord:

Een abstract methode is een methode zonder body. Een abstracte klasse is een klasse waar abstracte methoden in voorkomen. Hoewel je er geen `new` objecten van kunt maken, is een abstracte klasse wel nuttig als subklasse van concrete klassen.

(b)
- Kun je een `StreamReader` ook als `Stream` gebruiken? Waarom wel of niet?
- Bekijk de subklassen van de klasse `Stream`. Wat is het verschil een *store stream* en een *decorator stream* ?
- Hoe zou je een `StreamReader` kunnen omschrijven: als *store* of als *decorator*? Waarom?

Antwoord: Je kunt een `StreamReader` niet als `Stream` gebruiken, want het is er geen subklasse van. Een *store stream* verzorgt de eigenlijke opslag op een opslagmedium, een *decorator stream* voegt een nieuwe faciliteit toe aan een reeds bestaande stream. Een `StreamReader` is weliswaar geen stream, maar omdat het een faciliteit toevoegt aan een bestaande stream (namelijk het decoderen van character-encodings) zou je het als een *decorator* kunnen beschouwen. (Maar het is ook te verdedigen dat het een *store* is, omdat het niet een bestaande `StreamReader` uitbreidt.)

(c)
- Wat is een *lambda-expressie*?
- Geef met een voorbeeld aan hoe je een lambda-expressie kunt gebruiken om een handler aan het `Paint` event te koppelen, die een vierkantje op het scherm zet.

Antwoord: Een lambda-expressie is een methode zonder naam. Hier staat er eentje aan de rechterkant van de `+=`, om te dienen als event-handler van het `Paint` event:

```
this.Paint +=  (object o, PaintEventArgs pea) => { pea.Graphics.DrawRectangle(Pens.Black, 0, 0, 10, 10);
```

(d) Schrijf een programma dat door de gebruiker vanaf de console kan worden gestart. De gebruiker moet daarbij twee namen van tekstfiles meegeven. Het programma zorgt ervoor dat de tweede file een kopie wordt van de eerste.

Als de gebruiker te weinig of te veel filenamen opgeeft, moet daar melding van worden gemaakt, en ook als er bij het kopiëren iets fout is gegaan.

Antwoord:

```
class Kopie
{
    static void Main(string[] namen)
    {
        if (namen.Length!=2)
            Console.PrintLine("usage: Kopie origineel kopie");
        else try
             {
                 StreamReader invoer = new StreamReader(namen[0]);
                 StreamWriter uitvoer = new StreamWriter(namen[1]);
                 uitvoer.Write(invoer.ReadToEnd());
             }
             catch (Exception e)
             {
                 Console.PrintLine("er ging iets fout");
             }
    }
}
```

Bijlage bij opgave 2

```
public class Hoofdscherm : Form
{
    private TextBox naambox;
    KaartControl kaart;

    public Hoofdscherm()
    {
        this.Text = "StedenKaart";
        this.ClientSize = new Size(620, 350);

        kaart = new KaartControl();
        kaart.Hoofd = this;
        kaart.Location = new Point(10, 10);
        kaart.Size = new Size(600, 300);

        naambox = new TextBox();
        naambox.Location = new Point(10, 320);
        naambox.Size = new Size(600, 20);

        this.Controls.Add(kaart);
        this.Controls.Add(naambox);
    }
    // TODO: nog een property schrijven (opgave a)
}
```

```
public class Stad
{
    public string naam;
    public Point plek;
    public Color kleur;
    static Font font = new Font("Tahoma", 12);

    public Stad(Point p, string s, Color k)
    {
        this.plek = p;
        this.naam = s;
        this.kleur = k;
    }
    public void LaatZien(Graphics g)
    {
        Brush b = new SolidBrush(this.kleur);
        g.FillRectangle(b, plek.X - 5, plek.Y - 5, 10, 10);
        g.DrawString(naam, font, Brushes.Black, plek.X + 7, plek.Y - 20);
    }
    public bool Raak(Point p)
    {
        return Math.Abs(p.X - plek.X) < 5 && Math.Abs(p.Y - plek.Y) < 5;
    }
}
```

```
public class Weg
{
    public Stad st1, st2;

    public Weg(Stad s1, Stad s2)
    {
        st1 = s1;
        st2 = s2;
    }
}
```

```
public class KaartControl : UserControl
{
    const string filenaam = "../../steden.txt";
    protected ICollection<Stad> steden;
    Hoofdscherm hoofd;

    // TODO: constructor schrijven (opgave b)

    // TODO: nog een property schrijven (opgave a)

    public void Lees(string naam)
    {
        // TODO: body invullen (opgave c)
    }
    public virtual void Muisklik(object o, MouseEventArgs mea)
    {
        Stad s = new Stad(mea.Location, hoofd.DeNaam, Color.Red);
        steden.Add(s);
        this.Invalidate();
    }
    public virtual void Teken(object o, PaintEventArgs pea)
    {
        // roept LaatZien aan voor elke stad
        // (de details zijn hier weggelaten, je hoeft deze methode niet te schrijven).
    }

    // TODO: methode ZoekStad schrijven (opgave e)
}
```

```
public class WegenkaartControl  // TODO (opgave f)
```
