Bijlage B

Werkcollege-opgaven

2.1 *Programmeerparadigma's*
Waar of niet waar (en waarom?)
a. Alle imperatieve talen zijn object-georiënteerd.
b. Er zijn object-georiënteerde talen die niet procedureel zijn.
c. Procedurele talen moeten worden gecompileerd.
d. Declaratieve talen kunnen niet op dezelfde processor runnen als imperatieve, omdat die processor een toekenningsopdracht kan uitvoeren, die in declaratieve talen niet bestaat.
Antwoord: a. Nee, Fortran is wel imperatief maar niet objectgeoriënteerd.
b. Nee, objecten worden onder handen genomen door methoden, en dat zijn een soort procedures.
c. Nee, PHP is procedureel en wordt toch geïnterpreteerd.
d. Nee, de programmeertaal kan bepaalde eigenschappen van de onderliggende processor afschermen.

2.2 *Namen veranderen*

a. Bekijk de klasse `HalloWin3` in het voorbeeldprogramma HalloWin3 in hoofdstuk 2. Wat moet er allemaal veranderen als we de naam van deze klasse willen veranderen in `Hoi`? Wat hoeft er strikt genomen niet te veranderen, maar zou logisch zijn om ook te veranderen?
b. In het programma HalloWin3 gebruiken we twee klassen: `HalloWin3` en `HalloForm`. Zouden we dit programma ook met één klasse kunnen schrijven? Wat moeten we dan veranderen?

Antwoord: De naam van de klasse, de naam van de constructormethode, en de aanroep van de constructormethode vanuit `Main`. De tekst in de titelbalk hoeft niet te veranderen. De methodes van de twee klassen hadden ook gecombineerd kunnen worden in één klasse.

2.3 Wat wordt er in een C#-methode aangeduid met `this`? In welke situatie is het niet toegestaan om `this` te gebruiken?
Antwoord: `this` staat voor het object dat momenteel onder handen genomen wordt. We mogen `this` niet gebruiken in static methoden omdat die geen object onder handen hebben.

2.4 *Syntax en semantiek*
Wat wordt er verstaan onder de *syntax* van een (programmeer)taal-constructie? En wat is de *semantiek* van een taal-constructie?
Antwoord: De syntax van een taalconstructie is de grammaticale opbouw. De semantiek is de betekenis ervan.

3.1 *Commentaar*
Wat zijn de twee manieren om in C# commentaar bij een programma te schrijven?

3.2 *Declaratie, opdracht, expressie*
Wat is het verschil tussen een *declaratie*, een *opdracht* en een *expressie*?
Antwoord: Een declaratie legt het type van een variabele vast. Een opdracht kun je uitvoeren, waardoor het geheugen wordt veranderd. Een expressie kun je uitrekenen om de waarde ervan te bepalen.

3.3 *Vermenigvuldigen en delen*
Is er verschil tussen de opdrachten:

```
topy = y - 3*br / 2;
topy = y - 3/2 * br;
```

```
topy = y - br/2 * 3;
```

Antwoord: Wiskundige gesproken zou je zeggen dat er geen verschil is. Maar als het hier om int-variabelen gaat (zoals in het voorbeeldprogramma waaruit dit fragment afkomstig is) dan maakt de afronding van het resultaat van de deling dat het resultaat verschillend is. Bekijk bijvoorbeeld het geval dat `br` de waarde 5 heeft. Dan is `3*5/2` = `15/2` = 7. Maar `3/2*5` = `1*5` = 5, en `5/2*3` = `2*3` = 6.

3.4 *Methode schrijven en gebruiken*
Maak een schetsje van de uitvoer van onderstaand programma.
Herschrijf nu de methode `TekenScherm`, zo dat er een extra methode wordt gebruikt om de regelmaat explicieter te maken.

```
class Iets : Form
{
    Iets()
    {    this.Paint += this.tekenScherm;
    }
    public void tekenScherm(object o, PaintEventArgs pea)
    {   Graphics g = pea.Graphics;
        g.DrawLine(Pens.Black, 10,10,20,20);
        g.DrawLine(Pens.Black, 20,10,10,20);
        g.DrawLine(Pens.Red,   30,10,50,30);
        g.DrawLine(Pens.Red,   50,10,30,30);
        g.DrawLine(Pens.Blue,  60,10,90,40);
        g.DrawLine(Pens.Blue,  90,10,60,40);
    }
}
```

4.1 *Technische begrippen*
Geef korte definities van de begrippen *opdracht*, *variabele*, *methode*, en *object*.
Welke twee relaties heeft het begip *klasse* met de genoemde begrippen?
Antwoord: Een opdracht is een programmaconstructie die uitgevoerd kan worden. Een variabele is een geheugenplaats met een naam. Een methode is een groepje opdrachten met een naam. Een object is een gropjes variabelen dat bij elkaar hoort. Een klasse is een groepje methoden, en bovendien het type van een object.

4.2 *Toekenningen aan variabelen*
Bekijk een situatie waarin vier variabelen zijn gedeclareerd, en twee toekenningen zijn gebeurd, zoals na:

```
int x, y;
string s, t;
x = 40;
y = 12;
```

Beantwoord voor elke van de 9 groepjes opdrachten hieronder apart de vraag: wat zijn de waarden van `x` en `y` na het uitvoeren van het groepje opdrachten in bovengenoemde situatie?

```
y = x+1;        x = y;       x = y+1;       x = x+y;       y = x/3;
x = y+1;        y = x;       y = x-1;       y = x-y;       x = y*3;
                                            x = x-y;

y = 2/3*x;      y = x%6;     s = "hallo";                  s = "hallo";
x = 2*x/3;      x = x/6;     t = "\\//";                   t = s;
                             x = s.Length;                 s = s + "!";
                             y = t.Length;                 x = s.Length;
                                                           y = t.Length;
```

Antwoord:

```
y = 41;         x = 12;      x = 13;        x = 12;        y = 13;
x = 42;         y = 12;      y = 12;        y = 40;        x = 39;

y = 0;          y = 4;       x = 5;                        x = 6;
```

```
x = 26;        x = 6;       y = 3;                        y = 5;
```

Bij een van bovenstaande gevallen worden de waarden van `x` en `y` omgewisseld. Werkt dat voor alle mogelijke waarden van `x` en `y`? Zo ja, hoe is dat te verklaren? Zo nee, voor welke waarden niet?

Antwoord: Het omwisselen van `x` en `y` werkt voor alle gevallen. Dit kun je zien als je de constanten 40 en 12 voorstelt door `a` en `b` en ze aan de rechterkant invult. De uiteindelijke situatie is dan `y = a, x = b`. Echter, `x` en `y` mogen niet zo groot zijn dat `x+y` niet meer in een `int` past.

4.3 *Conversies*

Stel dat is gedeclareerd:

```
int x;     string s;    double d;
```

Vul de volgende toekenningen aan met de benodigde conversies (waarbij we er van uitgaan dat de string inderdaad een getal voorstelt).

```
x = d;      x = s;
s = x;      s = d;
d = x;      d = s;
```

Antwoord:

```
x = (int)d;
x = int.Parse(s);
s = x.ToString();
s = d.ToString();
d = x; // geen conversie nodig
d = double.Parse(s);
```

4.4 *Syntactische categorieën*

Hieronder staan 15 fragmenten uit een programma (in een blok van 3 bij 5). Schrijf bij elk fragment een letter passend bij het overeenkomstige fragment:

- T als het programmafragment een type is
- E als het programmafragment een expressie is
- O als het programmafragment een opdracht is
- D als het programmafragment een declaratie is
- H als het programmafragment een methode-header is
- X als het programmafragment geen van bovenstaande dingen is

```
double           double x;             (double)x*x
void x()         x==y+1                y=x!=x;
a%=x;            2xa0                  0x2a
Button b=ok;     Button                new Button("")
this.add(b);     String ok(Button b)   class OK : Form
```

Antwoord:

```
T double           D double x;             E (double)x*x
H void x()         E x==y+1                O y=x!=x;
O a%=x;            X 2xa0                  E 0x2a
D Button b=ok;     T Button                E new Button("")
O this.add(b);     H String ok(Button b)   X class OK : Form
```

4.5 *nog eentje: Syntactische categorieën* Hieronder staat 16 fragmenten uit een programma. Schrijf op je antwoordblad een blok van 4 bij 4 vakjes en zet in elk vakje een letter passend bij het overeenkomstige fragment:

- **T** als het programmafragment een **type** is
- **E** als het programmafragment een **expressie** is
- **O** als het programmafragment een **opdracht** is
- **D** als het programmafragment een **declaratie** is
- **H** als het programmafragment een **methode-header** is

- **X** als het programmafragment geen van bovenstaande dingen is

Antwoord:

```
[T] Button            [E] (bool)true           [H] int t()       [X ] class A : Size
[D] Button b;         [X] const bool true;     [D] int t=1;      [E] this.Size=this.ClientSize;
[O] b.Text = "ok";    [T] bool                 [E] t==t+1        [H] Size s(Size t)
[X] new Button b;     [O] while(true) t=1;     [O] t=t+1;        [E] new Size(x,y)
```

5.1 *Keywords*
Wat betekent het Engelse woord `void`, en in welke situatie is dit keyword nodig?
Wat betekent de Engelse afkorting `int`, en in welke situatie is dit keyword nodig?
Wat betekent in een C#-opdracht het woord `return`, en in welke situatie is dit keyword nodig?
Wat wordt in een C#-methode aangeduid door `this`, en in welke situatie is dit keyword nodig?
In welk soort methodes kan het *niet* gebruikt worden?
Antwoord: 'Void' betekent 'leeg'. Je schrijft het als resultaattype van methodes die eigenlijk geen resultaat hebben. 'Int' is een afkorting van 'integral', oftwel 'geheel'. Het is het type van gehele getallen (van 32 bits lang). 'Return' betekent zowel 'teruggeven' als 'terugkeren', en wordt gebruikt in de body van een methode met een resultaat, om een waarde terug te geven aan de aanroeper, en ook de programmacontrole te doen terugkeren naar de aanroeper.

5.2 *Methodes met een resultaat*

a. Schrijf een methode `restBijDeling` met twee parameters `x` en `y`, die de waarde van `x%y` teruggeeft, *zonder* daarbij de operator `%` te gebruiken.
Antwoord:

```
int restBijDeling(int x, int y)
{
    return x - y*(x/y);
}
```

b. Schrijf een methode `omtrek`, die die als resultaat oplevert wat de omtrek is van een rechthoek waarvan de lengte en breedte als parameters worden meegegeven.
Antwoord:

```
double berekenOmtrek(double lengte, double breedte)
{
     return 2*lengte + 2*breedte;
}
```

c. Schrijf een methode `diagonaal`, die als resultaat oplevert hoe lang de diagonaal is van een rechthoek waarvan de lengte en breedte als parameters worden meegegeven.
Antwoord:

```
double berekenDiagonaal(double lengte, double breedte)
{
     return Math.Sqrt(lengte*lengte + breedte*breedte);
}
```

d. Schrijf een methode `driewerf`, die drie aan elkaar geplakte kopieën oplevert van de string die als parameter wordt meegegeven, dus `driewerf("hoera!")` moet de string `"hoera!hoera!hoera!"` opleveren.
Antwoord:

```
string driewerf(string s)
{
     return s + s + s;
}
```

e. Schrijf een methode `keer64`, die 64 aan elkaar geplakte kopieën oplevert van de string die als parameter wordt meegegeven. Probeer daarbij het schrijfwerk in de methode te beperken.
Antwoord:

```
string keer2(string s)
{   return s+s;
}
```

```
string keer4(string s)
{   return keer2(s) + keer2(s);
}
string keer8(string s)
{   return keer4(s) + keer4(s);
}
string keer16(string s)
{   return keer8(s) + keer8(s);
}
string keer32(string s)
{   return keer16(s) + keer16(s);
}
string keer64(string s)
{   return keer32(s) + keer32(s);
}
```

5.3 *Uren, minuten, seconden*
Stel dat de variabele `tijd` een (mogelijk groot) aantal seconden bevat. Schrijf een aantal opdrachten, waardoor de variabelen `uren`, `minuten` en `seconden` een daarmee overeenkomende waarde krijgen, waarbij de waarden van `minuten` en `seconden` kleiner dan 60 moeten zijn.
Antwoord:

```
int uren = tijd/3600;
int minuten = (tijd%3600)/60;
int seconden = tijd%60;
```

Schrijf bovendien een methode die, het omgekeerde probleem, uit drie parameters `uren`, `minuten` en `seconden` de totale hoeveelheid seconden berekent.
Antwoord:

```
int totaalSeconden(int uren, int minuten, int seconden)
{
     return uren*3600 + minuten*60 + seconden;
}
```

6.1 *Spijkerschrift*
a. Schrijf een methode `streepjes` met een getal als parameter. Je mag zonder controle aannemen dat de parameter 0 of groter is. De methode moet als resultaat een string opleveren met daarin zoveel verticale streepjes als de parameter aangeeft. Bijvoorbeeld: de aanroep `this.streepjes(5)` levert `"|||||"` op.
b. Schrijf een methode `spijker` met een getal als parameter. Je mag zonder controle aannemen dat de parameter 1 of groter is. De methode moet als resultaat een string opleveren met daarin het getal in spijkerschrift-notatie. Elk cijfer wordt daarin weergegeven met verticale streepjes, en de cijfers worden gescheiden door een liggend streepje. Er staan ook liggende streepjes aan het begin en het eind. Hier zijn een paar voorbeelden:
`this.spijker(25)` geeft `"-||-|||||-"`
`this.spijker(12345)` geeft `"-|-||-|||-||||-|||||-"`
`this.spijker(7)` geeft `"-|||||||-"`
`this.spijker(203)` geeft `"-||--|||-"`
Hint: verwerk eerst het laatste cijfer, en herhaal dan voor de rest van de cijfers.
Antwoord:

```
public String streepjes(int n)
{
    String s; int t;
    s = "";
    for (t=0; t<n; t++)
        s += "|";
    return s;
}
public String spijker(int x)
{
```

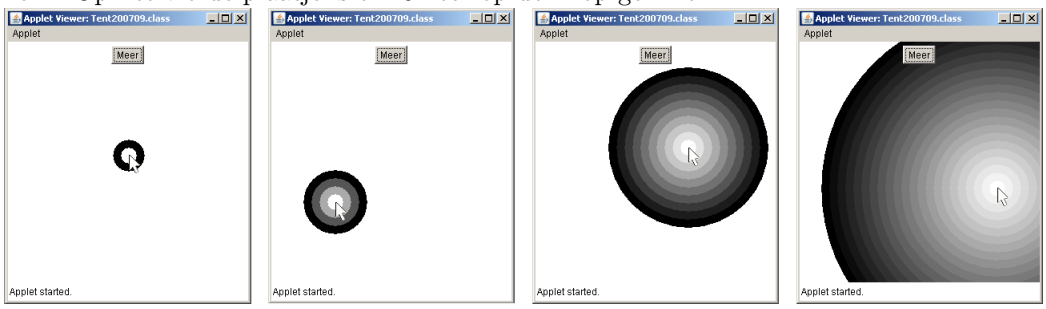
```
    String s;
    s = "";
    while (x>0)
    {
        s = streepjes(x%10) + "-" + s;
        x = x/10;
    }
    return "-" + s;
}
```

6.2 *Ringen*

Neem het volgende raamwerk voor een programma:

```
class Ring : Form
{
    Ring()
    {
        Button knop = new Button();
        knop.Text = "Meer";
        knop.Location = new Point(50,10);
        this.Controls.Add(knop);
        this.Paint     = this.teken;
        this.MouseMove = this.beweeg;
        knop.Click     = this.klik;
    }
    static void Main()
    {
        Application.Run(new Ring());
    }
}
```

Schrijf de ontbrekende declaraties en methoden in de klasse, zo dat het programma de volgende werking krijgt: een zwarte ring met een doorsnede van 40 beeldpunten beweegt mee met de bewegingen van de muis. De witte kern in het midden van de ring heeft een doorsnede van 20 beeldpunten. (Je kunt ook zeggen dat de dikte van het zwarte gedeelte van de ring 10 beeldpunten is). Zie het eerste plaatje hieronder.
In het window is een knop zichtbaar met het opschrift “Meer”. Elke keer dat de gebruiker daarop drukt, wordt de ring groter: er komt een band bij met een dikte van 10 beeldpunten; de totale diameter wordt dus 20 beeldpunten groter. De buitenste band is altijd zwart, de kern blijft altijd wit, en de banden daartussen krijgen een vloeiend verloop van zwart naar wit. Zie het tweede plaatje: er is tweemaal op de knop geklikt, en er zijn dus een zwarte buitenring, een donkergrijze band, een lichtgrijze band, en een witte kern.
Op het derde plaatje is er 8 keer op de knop ‘Meer’ geklikt, en zijn er dus 9 banden en een witte kern. Op het vierde plaatje is er 20 keer op de knop geklikt.



Antwoord:

```
int x, y, n;
void tekenScherm(object o, PaintEventArgs pea)
{
Graphics g = pea.Graphics;
    int i, k;
```

```
    for (i=n; i>=1; i--)
    {   k = 255*(n-i)/(n-1);
        Brush br = new Brush( new Color(k,k,k) );
        g.FillEllipse(br, mx-10*i, my-10*i, 20*i, 20*i);
    }
}
void beweeg(object o, MouseEventArgs mea)
{   mx = mea.X;
    my = mea.Y;
    this.Invalidate();
}
void klik(object o, EventArgs ea)
{   n++;
    this.Invalidate();
}
```

6.3 *Totaal van een rij getallen*
Schrijf een statische methode `totaal` met een getal n als parameter, die het totaal van de getallen van 0 tot en met n als resultaat oplevert.
Als n kleiner of gelijk is aan 0, moet het antwoord 0 zijn.
Hint: gebruik een variabele om het resultaat in op te bouwen.
Antwoord:

```
int totaal(int n)
{
    int resultaat = 0;
    for (int i=1; i<=n; i++)
        resultaat += i;
    return resultaat;
}
```

Het kan trouwens ook zonder while- of for-opdracht, al moet je daar maar net opkomen:

```
int totaal(int n)
{
    return n * (n+1) / 2;
}
```

6.4 *Product van een rij getallen*
Schrijf een statische methode `faculteit` met een getal n als parameter, die het product van de getallen 1 tot en met n als resultaat oplevert, dus alle getallen van één tot en met n met elkaar vermenigvuldigd.
Als n kleiner of gelijk is aan 1, moet het antwoord 1 zijn.
Antwoord:

```
int faculteit(int n)
{
    int resultaat = 1;
    for (int i=1; i<=n; i++)
        resultaat *= i;
    return resultaat;
}
```

Het is niet nodig om een apart geval voor $n \leq 1$ te programmeren: in die gevallen stopt de for-opdracht direct, en is de initialisatie van `resultaat` meteen het eind-antwoord.
Een alternatieve techniek gebruikt een techniek die bekend staat als *recursie*: de methode roept zichzelf aan, maar dan wel met een kleinere parameter. Die methode roept zichzelf dan ook weer aan, enzovoorts, net zolang todat de clausule $n \leq 1$ voorkomt dat de methode zichzelf oneindig vaak blijft aanroepen.

```
int faculteit(int n)
{   if (n <= 1)
        return 1;
```

```
    return n * this.faculteit(n-1);
}
```

6.5 *Machtsverheffen*
Schrijf een statische methode `macht` met een grondtal x en een exponent n als parameter. Het resultaat moet x^n zijn, dus x wordt n keer met zichzelf vermenigvuldigd.
Je mag aannemen dat n een natuurlijk getal is (dus niet negatief is). De methode moet ook goed werken als n gelijk is aan 0, en als x niet een geheel getal is.
Hint: gebruik een variabele om het resultaat in op te bouwen. Vergeet niet om die resultaat-variabele ook een beginwaarde te geven!
(Nota bene: in veel programmeertalen kan dit ook worden gedaan met de operator `^`. In C# kan dat niet, wel is er een methode `Math.Pow`, maar die mag je hier niet gebruiken want anders wordt het te makkelijk :–).
Antwoord:

```
double macht(double grondtal, int n)
{
    double resultaat = 1.0;
    for (int i=0; i<n; i++)
        resultaat *= grondtal;
    return resultaat;
}
```

6.6 *Reeksen*
De 'faculteit' van een natuurlijk getal is de uitkomst van alle getallen vanaf 1 tot en met dat getal met elkaar vermenigvuldigd. Bijvoorbeeld: de faculteit van 3 is $1 \times 2 \times 3 = 6$. Schrijf een statische methode `faculteit` die de faculteit van zijn parameter uitrekent. Je mag er zonder controle van uitgaan dat de parameter ≥ 1 is.
Een benadering van cosinus hyperbolicus van een reel getal x kun je berekenen door:

$$1 + x^2/2! + x^4/4! + x^6/6! + x^8/8! + x^{10}/10! + \cdots$$

De notatie 6! betekent hierin de faculteit van 6. Schrijf een statische methode `coshyp` die deze benadering berekent door 20 van deze termen te sommeren, en dat als resultaat oplevert. Je mag (maar hoeft niet) zelf extra hulp-methoden definiren.
Antwoord: Dit is het kortste:

```
private double coshyp(double x)
{    double res, a, b; int s, t;
     a=1; b=1; res=0;
     for (t=0; t<40; t+=2)
     {    res += a/b;
          a *= x*x;
          b *= (t+1)*(t+2);
     }
     return res;
     }
```

Maar je kunt natuurlijk ook eerst een methode `macht` en een methode `fac` schrijven:

```
private double macht(double x, int n)
{   double res; int t;
    res = 1;
    for (t=0; t<n; t++)
        res *= x;
    return res;
}
private int fac(int n)
{   int res, t;
    res = 1;
    for (t=2; t<=n; t++)
        res *= t;
```

```
        return res;
    }
```

en dan:

```
    private double coshyp(double x)
    {   double res; int t;
        res=0;
        for (t=0; t<40; t+=2)
            res += this.macht(x,t)/this.fac(t);
        return res;
    }
```

6.7 *String verveelvoudigen*
Eerder schreven we een methode `driewerf`, die drie kopieën van een string aan elkaar plakte.
Schrijf nu een statische methode `veelwerf`, die behalve een string ook een getal als parameter heeft, die aangeeft hoe vaak de string herhaald moet worden. Bijvoorbeeld, de aanroep `veelwerf("ha", 4)` levert `"hahahaha"`. Als het getal 0 is of negatief, moet deze methode een lege string opleveren.
Hint: gebruik een variabele om het resultaat in op te bouwen. Vergeet niet om die resultaatvariabele ook een beginwaarde te geven!
Antwoord:

```
string veelwerf(string s, int aantal)
{
    string resultaat = "";
    for (int i=0; i<aantal; i++)
        resultaat += s;
    return resultaat;
}
```

6.8 *Priemgetallen*

a. Schrijf een statische methode `even` die als resultaat oplevert of zijn parameter een even getal is. Bedenk goed wat een handig type is voor de parameter en het resultaat van de methode.
Antwoord:

```
bool even(int x)
{
    return x%2 == 0;
}
```

b. Schrijf een statische methode `deelbaar` met twee parameters x en y, die als resultaat oplevert of x een deelbaar is door y, dat wil zeggen de deling x/y precies opgaat.
Antwoord:

```
bool deelbaar(int x, int y)
{
    return x%y==0;
}
```

c. Schrijf een statische methode `kleinsteDeler`, die het kleinste getal ≥ 2 bepaalt waar de parameter door deelbaar is.
Hint: probeer één voor een de mogelijke delers, en stop als je er eentje gevonden hebt.
Antwoord:

```
int kleinsteDeler(int x)
{
    int deler = 2;
    while (!this.deelbaar(x,deler))
        deler++;
    return deler;
}
```

d. Schrijf een statische methode die bepaalt of een getal een priemgetal is, dat wil zeggen alleen maar deelbaar is door 1 en zichzelf.

Antwoord:

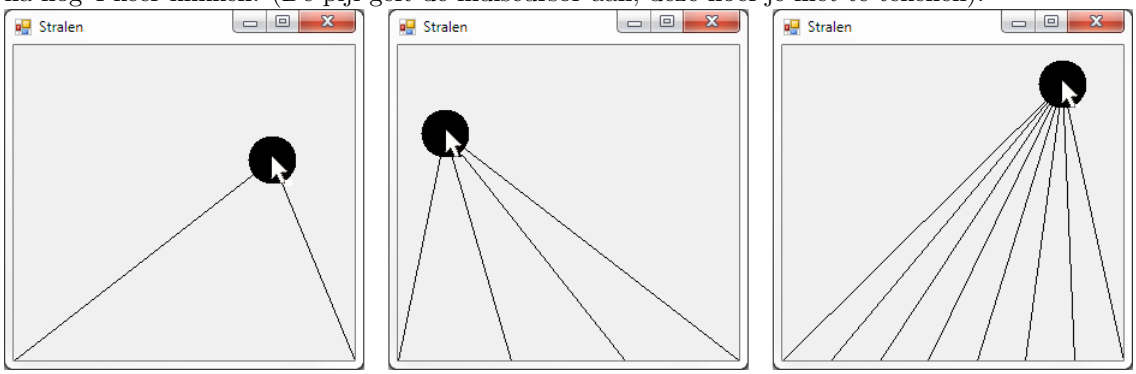
```
bool isPriemGetal(int x)
{
    return this.kleinsteDeler(x) == x;
}
```

6.9 *Stralen*

Gegeven is de volgende klasse:

```
class Program
{   public static void Main()
    {   Stralen s = new Stralen();
        s.Text = "Stralen";
        Application.Run(s);
    }
}
```

Schrijf de klasse `Stralen`, zo dat het programma zich als volgt gaat gedragen.
Er is een zwart opgevulde cirkel met een diameter van 40 pixels in beeld. Het middelpunt van de cirkel bevindt zich op de positie van de muis; de cirkel beweegt dus mee met de muis.
Twee lijnen verbinden het midden van de cirkel met de twee onderhoeken van het window. Elke keer als de gebruiker met de muis klikt komt er een lijn bij. De lijnen monden op gelijke afstanden uit op de onderrand van het window.
Zie onderstaande figuur, met daarin: de beginstituatie, de situatie na 2 keer klikken, en de situatie na nog 4 keer klikken. (De pijl geft de muiscursor aan, deze hoef je niet te tekenen).



Antwoord:

```
public class Stralen : Form
{
    int x, y, n;
    public Stralen()
    {   n = 1;
        this.Paint += this.teken;
        this.MouseMove += this.beweeg;
        this.MouseClick += this.klik;
    }
    void teken(object o, PaintEventArgs pea)
    {   Graphics gr = pea.Graphics;
        gr.FillEllipse(Brushes.Black, this.x - 20, this.y - 20, 40, 40);
        int t = 0;
        while (t <= n)
        {   gr.DrawLine(Pens.Black, this.x, this.y, t * this.ClientSize.Width / n, this.ClientSize.He
            t = t+1;
        }
    }
    void beweeg(object o, MouseEventArgs mea)
    {   this.x = mea.X;
        this.y = mea.Y;
```

```
            this.Invalidate();
        }
        void klik(object o, MouseEventArgs mea)
        {   n = n+1;
            this.Invalidate();
        }
    }
```

8.1 *Try/Catch*
Wat is de semantiek van een opdracht van de volgende vorm: `try` A `catch (`E`)` B ?
Antwoord: Probeer de opdrachten in A uit te voeren; als er daarbij een exception van type E optreedt, sla de rest van A over en doe in plaats daarvan de opdrachten in B.

8.2 *Lichtkrant*
Schrijf een programma'tje genaamd "Lichtkrant". Deze laat een tekst langzaam door het beeld bewegen, van rechts naar links. Als de tekst helemaal uit beeld is verdwenen, begint hij weer opnieuw.
De tekst die gebruikt wordt staat in een constante-declaratie. Ga ervan uit dat elke letter gemiddeld 10 beeldpunten breed is. Gebruik een Thread-object voor de animatie!
Antwoord:

```
public class Lichtkrant : Form
{
    string tekst = "Hallo allemaal";
    Font font = new Font("Tahoma", 20);
    int x = 0;
    public Lichtkrant()
    {   this.Paint += this.teken;
        Thread animatie = new Thread(this.run);
        animatie.Start();
    }
    public void teken(object o, PaintEventArgs pea)
    {   Graphics g = pea.Graphics;
        g.DrawString(tekst, font, Brushes.Black, x, 30);
        g.DrawString(tekst, font, Brushes.Black, x-this.ClientSize.Width, 30);
    }
    public void run()
    {   while (true)
        {   x = (x + 1) % this.ClientSize.Width;
            this.Invalidate();
            Thread.Sleep(50);
        }
    }
}
```

8.3 *Stabiel beeld bij animaties*
Als het Control opnieuw getekend moet worden (bijvoorbeeld omdat zijn Invalidate-methode is aangeroepen), dan worden alle "abonnees" van het Paint-event aangeroepen. Maar daarvoor nog wordt eerst de `virtual` methode `OnPaintBackground` aangeroepen, die er verantwoordelijk voor is om de achtergrond neutraal te kleuren.
Bij snelle animaties kan dat een hinderlijk knipper-effect geven: je ziet steeds de achtergrondkleur even oplichten, waarna het plaatje weer opnieuw getekend wordt. Verzin een manier om dat te voorkomen, om een stabieler beeld te krijgen.
Antwoord: Herdefinieer de methode `OnPaintBackground` met een lege body:

```
override void OnPaintBackground(PaintEventArgs pea)
{
}
```

Nu wordt de achtergrond niet meer uitgewist. In de Paint-eventhandler moet je dus zorgen dat het plaatje 'dekkend' getekend wordt. Zo nodig kun je een gedeelte van het scherm uitwissen door er een wit vierkant te tekenen. Ingewikkelde plaatjes kun je prepareren in een Bitmap, om die daarna

in 1 keer op het scherm te zetten.

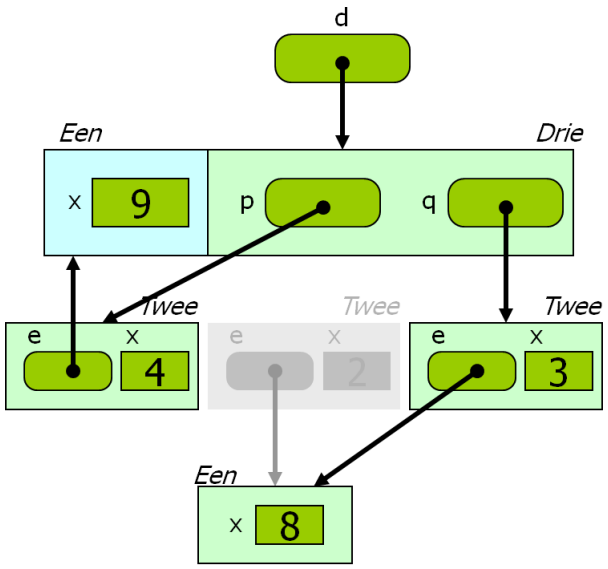
8.4 *Geheugen tekenen*
Gegeven zijn de volgende klasse-definities:

```
class Een
{   int x;
    public Een()
    {   x = 0;
    }
    public void setX(int a)
    {   x = a;
    }
}
class Twee
{   int x; Een e;
    public Twee(Een b, int c)
    {   e = b;
        x = c+1;
    }
    public Een getE()
    {   return e;
    }
}
class Drie : Een
{   Twee p, q;
    public Drie()
    {   p = new Twee( new Een(), 1 );
        p.getE().setX(7);
        q = new Twee( p.getE() , 2 );
        q.getE().setX(8);
        p = new Twee( this     , 3 );
        p.getE().setX(9);
    }
}
```

Teken, in dezelfde stijl als figuur 22 en 24 van het diktaat, de situatie die in het geheugen ontstaat na uitvoering van

```
Drie d = new Drie();
```

Maak, net als in het voorbeeld, duidelijk onderscheid tussen de naam en de waarde van de variabelen: de naam staat naast de hokjes, de waarde er in. Object-verwijzingen moeten, net als in het voorbeeld, met een duidelijke stip beginnen in het hokje van de verwijzings-variabele, en wijzen naar de rand van het object.
Antwoord:



Het middelste Twee-object is grijs aangegeven. Omdat dit niet meer via verwijzingen bereikbaar is had het er net zo goed niet meer kunnen zijn. Je mag het dus weglaten uit de tekening, maar het is ook goed om het te laten staan.

8.5 *Type van Add*
Iemand schrijft:

```
class Hallo : Form
{
    Hallo()
    {    b = new Button();
         t = new TextBox();
         this.Controls.Add(b);
         this.Controls.Add(t);
    }
}
```

a. Hoe kan het correct zijn dat `Add` zowel een Button als een TextBox als parameter accepteert? Wat is het type van de parameter van `Add`?
b. Mag je in plaats van de laatste opdracht ook schrijven:

```
b.Controls.Add(t);
```

zo nee, waarom niet? zo ja, wat gebeurt er dan?

Antwoord: a. De parameter van Add is een Control, en de methode accepteert dus alle subklassen van Control, zoals onder andere Button en TextBox.
b. Ja, dit kan gek genoeg inderdaad. In de Forms-library heeft *elke* Control een property `Controls` waarin een hoeveelheid sub-controls kan worden opgeslagen. Deze worden bovenop het control getekend; in dit geval krijg je dan een TextBox bovenop de Button.

8.6 *Klassen en overerving*
Gegeven de volgende klassen:

```
class A
{
    public float var1;
    protected int var2;
    private bool var3;
    public float Var1
    {   get { return var1; }
    }
    public int Var2
    {   get { return var2; }
        set { if (value > 0) var2 = value; }
```

```
    }
    public void methode_in_A()
    {   ...
    }
}
class B : A
{
    public int var4;
    private int var5;
    public void methode_in_B()
    {  ...
    }
}
```

a. Geef aan of de volgende expressies mogelijk zijn in `methode_in_A`:

```
this.var1      this.var2    this.var3
this.var4      this.Var2    base.var1
```

Antwoord:

```
this.var1 Y    this.var2 Y  this.var3 Y
this.var4 N    this.Var2 Y  base.var1 N
```

b. Geef aan of de volgende expressies mogelijk zijn in `methode_in_B`:

```
this.var1      this.var2    this.var3
this.var5      this.Var2    base.var1
base.var2      base.var3    base.Var2
```

Antwoord:

```
this.var1 Y    this.var2 Y  this.var3 N
this.var5 Y    this.Var2 Y  base.var1 Y
base.var2 Y    base.var3 N  base.Var2 Y
```

8.7 *Type-controle*

Worden de types van expressies over het algemeen gecontroleerd tijdens het compileren of tijdens het runnen van het programma?
Er is een uitzondering op deze regel. In welk geval is dat? (En waarom is die uitzondering nodig?)
Antwoord: Types worden gecontroleerd tijdens het compileren.
Je mag een object van een subklasse opslaan in een variabele van de superklasse.

```
class A     {...}
class B : A {...}
class Test
{   void Main()
    {
        A a;
        B b;
        a = new B();  // dit mag, want B is een subklasse van A
        b = new A();  // dit mag niet: compile-time fout
        b = a;        // dit is ook een compile-time fout
        b = (B)a;     // zo mag het wel!
    }
}
```

Met de cast geef je als programmeur aan dat je weet dat, met deze voorgeschiedenis, de toekenning toch veilig is. Tijdens het runnen wordt gecontroleerd of op dat moment de variabele a inderdaad een object van type `Subklasse` bevat.
Deze controle kan niet door de compiler worden gedaan, omdat het in alle gevallen kunnen analyseren van de voorgeschiedenis gelijk staat aan het oplossen van het Halting-probleem.

9.1 *Arrays*

a. Schrijf een methode `aantalNullen` die als parameter een array van integers meekrijgt. Het resultaat van de methode is het aantal nullen dat in de array staat.
Antwoord:

```
int aantalNullen(int[] arr)
{
    int aantal = 0;
    for (int i=0; i<arr.Length; i++)
        if (arr[i] == 0)
            aantal++;
    return aantal;
}
```

b. Schrijf een methode `optellen` die twee integer arrays van gelijke lengte als parameters meekrijgt. De methode telt de waarden in de twee arrays bij elkaar op en geeft als resultaat een nieuwe array. Bijvoorbeeld, gegeven een array `array1 = { 0, 3, 8, -4 }` en een array `array2 = { 10, 2, -8, 8 }`. De aanroep van de methode `optellen(array1, array2)` levert als resultaat `{ 10, 5, 0, 4 }` op.
Antwoord:

```
int[] optellen(int[] arr1, int[] arr2)
{
    int[] res = new int[arr1.Length];
    for (int i=0; i<arr1.Length; i++)
        res[i] = arr1[i] + arr2[i];
    return res;
}
```

c. Schrijf een methode `eerstePositie` die als parameters een `string` en een `char` meekrijgt en die de eerste positie van het karakter in de string als resultaat teruggeeft. Als het karakter niet in de string voorkomt, dan levert de methode -1 op. Bijvoorbeeld:

```
int resultaat = this.eerstePositie("arjan", 'j'); // levert 2 op
resultaat = this.eerstePositie("jeroen", 'j'); // levert 0 op
resultaat = this.eerstePositie("hans", 'j'); // levert -1 op
```

Antwoord:

```
int eerstePositie(string s, char c)
{
    int pos = 0;
    while (pos < s.Length)
    {
        if (s[pos] == c)
            return pos;
        pos++;
    }
    return -1;
}
```

9.2 String-methoden

a. In de klasse String zit de methode `ToUpper`, die een hoofdletterversie van de string oplevert. Die kun je bijvoorbeeld aanroepen met

```
h = s.ToUpper( );
```

Als je zelf de klasse String zou moeten schrijven, hoe zou je deze methode dan kunnen definiëren (gebruikmakend van andere methoden in de klasse String)?
Antwoord:

```
public string ToUpper()
{
    string res = "";
    for (int n=0; n<this.Length; n++)
    {   char c = this[n];
```

```
            if (c>='a' && c<='z')
                c = (char)(c-32);
            res += c;
        }
        return res;
    }
```

b. In de klasse String zit een methode `Replace`. Deze methode levert een nieuwe string op, waarin elk voorkomen van het character dat als eerste parameter wordt meegegeven, is vervangen door het character dat als tweede parameter wordt meegegeven. Bijvoorbeeld:

```
"Utrecht".replace('t','x')     // geeft "Uxrechx"
"A+2+#?".replace('+','9')      // geeft "A929#?"
```

Stel dat je de auteur van de klasse String bent. Veel andere methoden van die klasse zijn al geschreven (die mag je dus gebruiken). Schrijf de methode `Replace`.
Antwoord:

```
public string Replace(char x, char y)
{
    string res;
    int n;
    char c;
    res = "";
    for (n=0; n<this.Length; n++)
    {
        c = this[n];
        if (c==x)
            c = y;
        res += c;
    }
    return res;
}
```

c. Ook is er een methode `EndsWith`, die oplevert of een string eindigt met de string die als parameter wordt meegegeven. Bijvoorbeeld:

```
"Utrecht".endsWith("recht")  // geeft true
```

Stel dat je de auteur van de klasse String bent. Veel andere methoden van die klasse zijn al geschreven (die mag je dus gebruiken), maar nog niet de `Substring` en `IndexOf` methoden (die mag je dus niet gebruiken). Schrijf de methode `endsWith`.
Antwoord:

```
public boolean EndsWith(string s)
{
    int n, sl, tl;
    tl = this.Length;
    sl = s.Length;
    if (sl>tl)
        return false;
    for (n=0; n<sl; n++)
    {   if (s[n] != this[tl-sl+n])
            return false;
    }
    return true;
}
```

9.3 *Tweedimensionale arrays*
Er zijn twee manieren om een twee-dimensionale array te declareren:

```
int [,] eerste;
int [][] tweede;
```

Wat is het verschil? In welke situatie is het handig om de `tweede` te gebruiken?
Antwoord: `eerste` bevat alle elementen in het array-object, `tweede` is eigenlijk een array van

verwijzingen naar arrays. Dat laatste is handig als de rijen van de array niet allemaal even lang zijn.

9.4 *String-methoden*
In de klasse `String` zitten onder andere de volgende methoden:

```
static bool IsNullOrWhiteSpace(string)
static int Compare(string, string)
```

a. De naam van `IsNullOrWhiteSpace` spreekt voor zichzelf. Onder 'whitespace' verstaan we spaties, tab-tekens en newline-tekens.
Schrijf deze methode, *zonder* gebruik te maken van de bestaande `IsNullOrEmpty` en `IsNullOrWhiteSpace` methoden.
Antwoord:

```
static bool IsNullOrWhiteSpace(string s)
{
    if (s==null) return true;
    for (int t=0; t<s.Length; t++)
        if (s[t]!=' ' && s[t]!='\t' && s[t]=='\n')
            return false;
    return true;
}
```

b. De methode `Compare` levert 0 op als de twee parameters precies gelijk zijn. Hij levert een negatief getal op (bijvoorbeeld -1, maar iets anders mag ook) als de eerste parameter kleiner is dan de tweede, en een positief getal (bijvoorbeeld 1) als die groter is. Met kleiner en groter wordt hier de woordenboek-ordening bedoeld: de eerste letter waar de strings verschillen bepaalt de ordening (volgens de Unicodes van die letters). Is de ene string een beginstuk van de andere, dan is de kortste de kleinste. Spaties en leestekens tellen gewoon mee, die hoeven dus niet speciaal behandeld te worden.
Voorbeelden:

`String.Compare("aap", "noot")`	geeft een negatief getal, want `'a'<'n'`
`String.Compare("noot", "nieten")`	geeft een positief getal, want `'o'>'i'`
`String.Compare("niet", "nietmachine")`	geeft een negatief getal vanwege de lengte
`String.Compare("noot", "noot")`	geeft 0, want precies gelijk
`String.Compare("noot", "NOOT")`	geeft een positief getal, want `'n'>'N'`

De methode neemt aan dat de parameters niet `null` zijn (en controleert dat ook niet).
Schrijf deze methode, *zonder* gebruik te maken van de bestaande `Compare` en `CompareTo` methoden.
Antwoord:

```
static int Compare(string s, string t)
{
    int m = s.Length;
    int n = t.Length;
    for (int i=0; i<Math.Min(m,n); i++)
    {
        char c = s[i];
        char d = t[i];
        if (c!=d) return c-d;
    }
    return m-n;
}
```

9.5 Zoeken en sorteren

a. Schrijf een statische methode `grootste` met als parameter een array van doubles, met als resultaat de *hoogste waarde* die in de array voorkomt.
Antwoord:

```
double grootste(double [] a)
{
    double result = a[0];
```

```
    for (int t=1; t<a.Length; t++)
        if (a[t] > result)
            result = a[t];
    return result;
}
```

b. Schrijf een andere statische methode `plaatsGrootste`, met als resultaat het *volgnummer* (de index in de array) van de grootste waarde.
Antwoord:

```
int plaatsGrootste(double [] a)
{
    int result = 0;
    for (int t=1; t<a.Length; t++)
        if (a[t] > a[result])
            result = t;
    return result;
}
```

c. Schrijf een methode met als parameter een array van doubles. De methode moet opleveren hoe vaak de kleinste waarde van de array voorkomt.
Bijvoorbeeld: als de array de waarden 9,12,9,7,12,7,8,25,7 bevat, dan is het resultaat 3, omdat de kleinste waarde (7) drie maal voorkomt.
Antwoord:

```
int hoeVaakKleinste(double [] a)
{
    int result = 1;
    double kleinste = a[0];
    for (int t=1; t<a.Length; t++)
        if (a[t] < kleinste)
        {   result = 1;
            kleinste = a[t];
        }
        else if (a[t]==kleinste)
            result++
    return result;
}
```

d. Maak een variant van de methode `plaatsGrootste`, waarbij niet de hele array doorzocht wordt, maar alleen de eerste `n` elementen, waarbij `n` een aparte parameter is.
Antwoord:

```
int plaatsGrootste(double [] a, int n)
{
    int result = 0;
    for (int t=1; t<n; t++)
        if (a[t] > a[result])
            result = t;
    return result;
}
```

e. Gebruik deze methode in de volgende methode: schrijf een statische methode `Sorteer`, met als parameter een array van doubles. Het resultaattype is `void`. Na afloop van het uitvoeren van de methode moeten de elementen in de array in opklimmende volgorde van grootte staan. Hint hierbij: zoek eerst de plaats van de grootste waarde in de hele array. Verwissel deze waarde met de waarde op de laatste plaats. Dan staat de grootste waarde alvast achteraan, waar hij hoort. Zoek nu de grootste waarde van het resterende deel van de array. Die waarde wordt verwisseld met de op één na laatste waarde in de array. Dit gaat zo verder: zoek weer het grootste element van de array minus de laatste twee elementen, verwissel die met op het twee na laatste element, enzovoorts.
Antwoord:

```
void Sorteer(double [] a)
```

```
{
    for (int t=a.Length; t>0; t++)
    {
        int p = plaatsGrootste(a,t);
        double h = a[p];
        a[t-1] = a[p];
        a[p] = h;
    }
}
```

f. Schrijf een statische methode met als parameter een array van integers en een losse integer, die oplevert op welke plaats de losse integer als eerste in de array voorkomt. Als het getal nergens voorkomt, moet de methode -1 opleveren.
Antwoord:

```
int eerste(int [] a, int x)
{
    for (int t=0; t<a.Length; t++)
        if (a[t]==x)
            return x;
    return -1;
}
```

g. (moeilijker) Als je weet dat de array op volgorde gesorteerd is, kun je op een slimmere manier zoeken: kijk in het midden van de array of je daar de gezochte waarde aantreft. Zo ja, mooi. Zo nee, dan weet je of je in de helft links ervan of in de helft rechts ervan moet verder zoeken. Op deze manier wordt het te doorzoeken stuk van de array bij elke stap twee keer zo klein. Gebruik twee integers die de grenzen van het te doorzoeken stuk array aanduiden.
Antwoord:

```
int eerste(int [] a, int x)
{
    int laag = 0;
    int hoog = a.Length;
    while (hoog>laag)
    {
        int mid = (laag+hoog)/2;
        if (a[mid]==x)
            return mid;
        else if (a[mid]<x)
            laag = mid+1;
        else hoog = mid;
    }
    return -1;
}
```

9.6 *Strings*
In de klasse String zitten onder andere methoden met de volgende headers:

```
public string Substring(int startIndex);
public string Substring(int startIndex, int length);
public int IndexOf(char value)
```

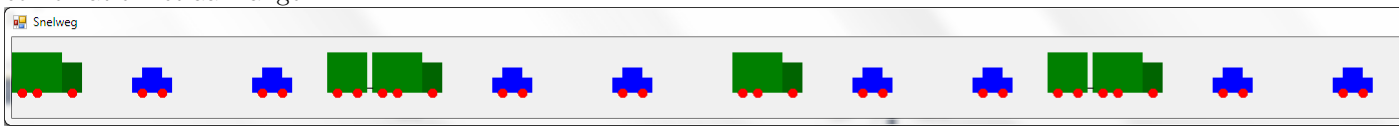
Stel dat je de auteur van de klasse String bent, en deze methoden zijn nog niet aanwezig. Schrijf deze drie methoden. Je mag daarbij niet gebruik maken van methoden Substring en IndexOf (niet de genoemde drie, maar ook niet van andere varianten met deze namen, want we nemen aan dat die er nog niet zijn), maar wel van de andere methoden die in de klasse String al aanwezig zijn, en van je eigen methoden.
Antwoord:

```
public string Substring(int startIndex, int length)
{
    string s = "";
    for (int i=startIndex; i<startIndex + length && i<this.Length; i++)
```

```
            s += this[i];
        return s;
    }
    public string Substring(int startIndex)
    {
        return this.Substring(startIndex, this.Length);
    }
    public int IndexOf(char value)
    {
        for (int i=0; i<this.Length; i++)
            if (this[i] == value)
                return i;
        return -1;
    }
```

9.7 *subklassen en arrays*

Bekijk het onderstaande programma. Het moet een file auto's op de snelweg tekenen, zoals in de screendump hieronder. Elke derde auto is een vrachtwagen, en elke tweede vrachtwagen is een combinatie met aanhanger.




```
public class Snelweg : Form
{   public Snelweg()
    {   this.Text = "Snelweg";
        this.ClientSize = new Size(1800, 80);
        this.Paint += this.tekenSnelweg;
        // TODO: ontbrekend deel van de constructor
    }
    public void tekenSnelweg(object o, PaintEventArgs pea)
    {   for (int t = 0; t < rijbaan.Length; t++)
            rijbaan[t].Teken(pea.Graphics, t*120, 60);
    }
    static void Main()
    {   Application.Run(new Snelweg());
    }
}
class MotorVoertuig
{   public void Teken(Graphics gr, int x, int y)
    {
    }
}
class PersonenAuto : MotorVoertuig
{   public void Teken(Graphics gr, int x, int y)
    {   gr.FillRectangle(Brushes.Blue, x, y - 20, 40, 15);
        gr.FillRectangle(Brushes.Blue, x+10, y - 30, 20, 10);
        gr.FillEllipse(Brushes.Red, x + 5, y - 10, 10, 10);
        gr.FillEllipse(Brushes.Red, x + 25, y - 10, 10, 10);
    }
}
class Vrachtwagen : MotorVoertuig
{   public void Teken(Graphics gr, int x, int y)
    {   gr.FillRectangle(Brushes.Green, x, y - 45, 50, 40);
        gr.FillRectangle(Brushes.DarkGreen, x+50, y - 35, 20, 30);
        gr.FillEllipse(Brushes.Red, x + 5, y - 10, 10, 10);
        gr.FillEllipse(Brushes.Red, x + 20, y - 10, 10, 10);
        gr.FillEllipse(Brushes.Red, x + 55, y - 10, 10, 10);
    }
```

```
}
class Combinatie : Vrachtwagen
{   public void Teken(Graphics gr, int x, int y)
    {   // de vrachtwagen
        gr.FillRectangle(Brushes.Green, x, y - 45, 50, 40);
        gr.FillRectangle(Brushes.DarkGreen, x + 50, y - 35, 20, 30);
        gr.FillEllipse(Brushes.Red, x + 5, y - 10, 10, 10);
        gr.FillEllipse(Brushes.Red, x + 20, y - 10, 10, 10);
        gr.FillEllipse(Brushes.Red, x + 55, y - 10, 10, 10);
        // de aanhanger
        gr.DrawLine(Pens.Black, x - 5, y - 10, x, y - 10);
        gr.FillRectangle(Brushes.Green, x-45, y - 45, 40, 40);
        gr.FillEllipse(Brushes.Red, x -40, y - 10, 10, 10);
        gr.FillEllipse(Brushes.Red, x -20, y - 10, 10, 10);
    }
}
```

a. Er ontbreekt nog een declaratie. Schrijf deze declaratie, en geef aan waar die moet staan.
Antwoord: Boven in de klasse `Snelweg` komt de declaratie

```
MotorVoertuig[] rijbaan = new MotorVoertuig[25];
```

b. Schrijf het ontbrekende deel van de constructor van `Snelweg`.
Antwoord:

```
for (int t = 0; t < rijbaan.Length; t++)
{   if (t % 3 != 0)
        rijbaan[t] = new PersonenAuto();
    else if (t % 6 == 0)
        rijbaan[t] = new Vrachtwagen();
    else rijbaan[t] = new Combinatie();
}
```

c. In het gegeven programma zit nog een fout, waardoor er helemaal niets zichtbaar wordt. Hoe komt dat, en hoe kan de fout worden verbeterd?
Antwoord: Het element-type van de array is `MotorVoertuig`, dus wordt in `tekenSnelweg` steeds de lege methode `Teken` van `MotorVoertuig` aangeroepen. Deze methode had `virtual` gedeclareerd moeten worden, en in de subklasse als `override`.

d. De programmeur heeft een flink stuk code met copy&paste gedupliceerd. Waarom is dat geen goed idee?
Antwoord: Er ontstaat nu een versie-management probleem: als de vorm van de vrachtauto later verandert, moet dat op twee plaatsen worden aangepast.

e. Hoe had het dupliceren van de code het beste vermeden kunnen worden?
Antwoord: In de klasse `Combinatie` had de trekker getekend kunnen worden met de aanroep `base.Teken(gr,x,y)`.

f. We willen de object-georiënteerde opzet van het programma nog verder doorvoeren, zo dat ook de concepten 'wiel' en 'aanhanger' met klassen worden gemodelleerd. Hoe kan dat netjes worden aangepakt? Zorg ervoor dat code-duplicatie zo veel mogelijk vermeden kan worden, en dat er nooit door een programmeerfout een losse aanhanger op de weg terecht kan komen. Je hoeft dit niet helemaal uit te programmeren; geef alleen aan welke klassen er komen, hoe hun subklasse-relatie is, en welke declaraties er in (bestaande en/of nieuwe) klassen komen te staan.
Antwoord: Nieuwe klasse `Wiel`, los van alle andere klassen. Nieuwe klasse `Wagen` als superklasse van `MotorVoertuig`. Daarin een array met `Wiel`-objecten. Nieuwe klasse `Aanhanger` als subklasse van `Wagen`. In de klasse `Combinatie` een declaratie van een `Aanhanger`.

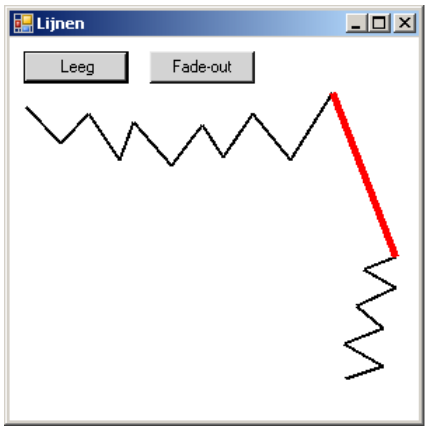
9.8 *Lijnen*
Gegeven is de volgende klasse, met daarin de methode `Main` en twee andere handige methoden.

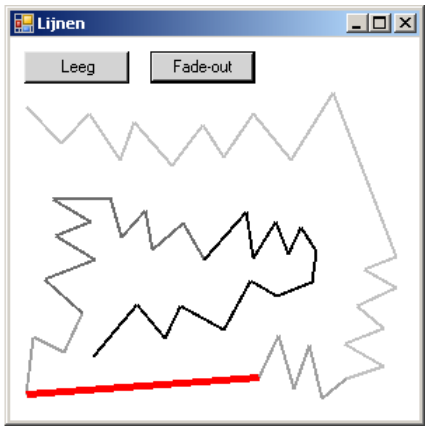
```
public class Help
{   public static void Main()
    {   Lijnen lijnen = new Lijnen();
        lijnen.Text = "Lijnen";
```

```
            lijnen.BackColor = Color.White;
            Application.Run(lijnen);
        }
        public static Button MaakKnop(string tekst, int x, int y, EventHandler eh)
        {   Button b = new Button();
            b.Text = tekst;
            b.BackColor = Color.LightGray;
            b.Location = new Point(x, y);
            b.Click += eh;
            return b;
        }
        public static int Kwadraat(int x)
        {   return x * x;
        }
    }
```

Schrijf nu de klasse `Lijnen`, zo dat het programma zich als volgt gaat gedragen:

- De gebruiker ziet twee knoppen met het opschrift 'Leeg' en 'Fade out'.
- De gebruiker kan verder overal in het window klikken. De aangeklikte punten worden verbonden door lijnen. (Na de eerste klik ziet de gebruiker nog niets, bij de tweede klik verschijnt er een lijn, bij de derde klik en tweede lijn, enz).
- Er zijn maximaal 100 lijnen zichtbaar. Als de gebruiker daarna toch meer punten aanklikt, gebeurt er niets (ook geen foutmelding!).
- Alle lijnen zijn zwart met dikte 2, behalve de langste lijn: die is rood met dikte 5. (Als er meerdere lijnen de langste zijn, mag je er daar een van kiezen, of ze allemaal rood maken).
- Na het indrukken van de knop 'Leeg' verdwijnen alle lijnen. De gebruiker kan dan weer met 100 nieuwe lijnen beginnen.
- Na het indrukken van de knop 'Fade out' beginnen de lijnen langzaam te vervagen: elke seconde worden ze 10% grijzer. Theoretisch gesproken worden ze dus nooit helemaal wit, maar in de praktijk zijn ze na een halve minuut onzichtbaar geworden op de witte achtergrond. Nieuw aangeklikte lijnen beginnen wel weer zwart. De langste lijn blijft rood.
- Na nogmaals indrukken van de knop 'Fade out' stopt het vervagen, na een derde keer indrukken gaat het weer verder waar het gebleven was, enz.





In het voorbeeld links heeft de gebruiker 19 punten aangeklikt. De langste lijn is dikker en rood. In het voorbeeld rechts is de 'Fade out' knop gebruikt. Later gemaakte lijnen zijn nog donkerder. Inmiddels is ook een andere lijn de langste.

Antwoord:

```
public class Lijnen : Form
{
    Point[] ps = new Point[100];
    int[] ks = new int[100];
    int n = 0;
```

```
    public Lijnen()
    {   this.Controls.Add( Help.MaakKnop("Leeg",      10, 10, this.leeg ));
        this.Controls.Add( Help.MaakKnop("Fade-out", 100, 10, this.fade ));
        this.Paint      += this.teken;
        this.MouseClick += this.klik;
    }
    public void teken(object o, PaintEventArgs pea)
    {   int maxlen = 0;
        int maxt = 0;
        Graphics gr = pea.Graphics;
        for (int t = 1; t < n; t++)
        {
            int len = Help.Kwadraat(ps[t].X - ps[t-1].X) + Help.Kwadraat(ps[t].Y - ps[t-1].Y);
            if (len > maxlen)
            {   maxlen = len;
                maxt = t;
            }
            int k = 255 - ks[t];
            Color c = Color.FromArgb(k, k, k);
            Pen pen = new Pen(c, 2);
            gr.DrawLine(pen, this.ps[t-1], this.ps[t]);
        }
        if (maxt>0)
            gr.DrawLine(new Pen(Color.Red,5), this.ps[maxt - 1], this.ps[maxt]);
    }
    public void klik(object o, MouseEventArgs mea)
    {   if (n < 100)
        {   ps[n] = mea.Location;
            ks[n] = 255;
            n++;
            this.Invalidate();
        }
    }
    public void leeg(object o, EventArgs ea)
    {   n = 0;
        this.Invalidate();
    }
    Thread animatie;
    public void fade(object o, EventArgs ea)
    {   if (animatie == null)
        {   animatie = new Thread(this.run);
            animatie.Start();
        }
        else animatie = null;
    }
    public void run()
    {   while (animatie != null)
        {   for (int t = 0; t < n; t++)
                ks[t] = ks[t] * 9 / 10;
            this.Invalidate();
            Thread.Sleep(1000);
        }
    }
}
```

10.1 *TextReader*

In de klasse `TextReader` zitten onder andere de volgende methoden:

```
char Read();        // geeft de eerstvolgende letter, of -1 als die er niet is
string ReadLine();  // geeft alles tot (maar zonder) de eerstvolgende newline
string ReadToEnd(); // geeft de rest van de file
```

Welk van deze methoden moet(en) `abstract` gedefinieerd worden? Implementeer de overige me-

thode(n). Is daarbij de modifier `virtual` en/of `override` nodig?
Antwoord: De methode `Read` kan nog niet gedefinieerd worden, omdat het in de abstracte klasse `TextReader` nog niet duidelijk is waarvandaan de char moet komen. Deze methode is dus `abstract`. De overige methoden kunnen in termen van deze methode worden gedefinieerd:

```
string ReadLine()
{   string res = "";
    char c;
    c = this.Read();
    while ( c!=-1 && c!='\n' )
    {   res += c;
        c = this.Read();
    }
    return res;
}
string ReadToEnd()
{   string res = "";
    char c;
    c = this.Read();
    while ( c!=-1 )
    {   res += c;
        c = this.Read();
    }
    return res;
}
```

Deze methoden kunnen niet met `override` gedefinieerd worden, want ze bestaan nog niet in de superklasse. Wel zou `virtual` kunnen, als we de auteurs van de subklassen willen toestaan om deze methoden te overriden. Erg zinvol lijkt dat echter niet.

10.2 *Tekstfiles en collections*
Schrijf een programma (met commandline interface) met de volgende specificatie. Het programma wordt door de gebruiker opgestart vanaf een commandoregel. De gebruiker specificeert daarbij twee filenamen. Het programma leest een tekstfile met de eerste filenaam als naam. Daarna schrijft het een tekstfile met de tweede filenaam als naam De uitvoer moet elk woord uit de invoer op aparte regels vermelden. De woorden staan daarbij op (alfabetische, of eigenlijk Unicode) volgorde, waarbij dubbele woorden slechts éénmaal worden vermeld.
Elk groepje karakters zonder spatie erin beschouwen we als woord. Je mag ervan uitgaan dat er tussen de woorden precies 1 spatie staat.
Als de gebruiker te weinig of te veel filenamen opgeeft, of als er een fout optreedt bij het lezen of schrijven, krijgt de gebruiker daarvan een korte melding.
Voorbeeld: als de invoer de volgende twee regels bevat:

```
dit IS een *%#$ voorbeeld
van een tekst!
```

dan moet de uitvoer de volgende zeven regels bevatten:

```
*%#$
IS
dit
een
tekst!
van
voorbeeld
```

Als dit niet helemaal lukt, kun je je de volgende versimpelingen permitteren: gebruik vaste filenamen in plaats van door de gebruiker gespecificeerde; laat het sorteren en ontdubbelen achterwege; verwerk niet de woorden, maar de regels van de tekst; laat de foutmeldingen achterwege.
Antwoord:

```
public class Woord
{
    static void Main(string [] namen)
```

```
    {
        if (namen.length!=2)
            Console.WriteLine("Usage: Woord input output");
        else
        {   try
            {
                TextReader invoer = new StreamReader(namen[0]);
                TextWriter uitvoer = new StreamWriter(namen[1]);
                SortedSet<String> alles = new SortedSet<String>();
                String regel;
                while ((regel=invoer.readLine())!=null)
                    foreach (string woord in regel.Split(" "))
                        alles.Add(woord);
                foreach (string t int alles)
                    uitvoer.println(t);
            }
            catch (Exception e)
            {   Console.WriteLine("foutje bij het lezen");
            }
        }
    }
}
```

10.3 *Decorator streams*

De klasse `Stream` heeft onder andere de volgende methoden:

```
int ReadByte();    // geeft het eerstvolgende byte, of -1 als dat er niet meer is
int Read(byte[] doel, int n);  // leest maximaal n bytes, en zet die neer in doel.
                               // geeft het aantal gelezen bytes terug.
```

Het is vaak efficiënter om, met methode `Read`, een heel blok tegelijk te lezen. Maar het is gemakkelijker om een losse byte te kunnen lezen als je hem nodig hebt.
Ziedaar het nut van de klasse `BufferedStream`. Bij de constructor van deze klasse geef je hem een reeds bestaande `Stream` in beheer. Als je daarna `ReadByte` aanroept, spreekt hij het object dat hij in beheer heeft aan om meteen maar 1000 bytes te lezen. De eerste daarvan geeft hij terug, de overige bewaart hij zolang in een array. Roep je daarna nog eens `ReadByte` aan, dan kan die 'uit voorraad'geleverd worden, en hoeft de onderliggende file dus niet aangesproken te worden. Pas als de voorraad op is, wordt er weer een nieuw blok ingelezen.
Opgave: schrijf de klasse `BufferedStream`, met een constructormethode en de methode `ReadByte`.
Antwoord:

```
class BufferedStream : Stream
{
    Stream onderliggend;
    byte[] buffer = new byte[1000];
    int tel, max;
    BufferedStream(Stream s)
    {    onderliggend = s;
         tel = 0; max=0;
    }
    override int ReadByte()
    {
        if (tel==max)
        {    // de voorraad is op, lees een nieuw blok
             max = onderliggend.Read(buffer, buffer.Length);
             if (max==0)
                 return -1;  // er was niets meer te lezen
             else tel = 0;
        }
        byte res = buffer[tel];
        tel++;
        return res;
```

```
    }
}
```

De laatste drie regels kunnen ook worden gecombineerd tot:

```
    return buffer[tel++];
```

Je zou misschien denken dat de teller nu te vroeg wordt opgehoogd, of juist helemaal niet omdat hij er al uitgevlogen is, maar dat is niet het geval: dit werkt precies zoals je zou hopen: de *oude* waarde van `tel` wordt gebruikt om de array te indexeren, en *daarna* wordt de waarde van `tel` opgehoogd.
Indien je de notatie `++tel` gebruikt, wordt de teller juist *eerst* opgehoogd, maar dat willen we hier natuurlijk niet hebben.

10.4 *Lists*
In de klasse `List<string>` zitten onder andere methoden met de volgende headers:

```
public void Reverse()
public int LastIndexOf(string item)
public bool Contains(string item)
```

Stel dat je de auteur van de klasse List bent, en deze methoden zijn nog niet aanwezig. Schrijf deze drie methoden. Je mag daarbij niet gebruik maken van de bestaande methoden met dezelfde naam en ook niet van andere varianten met deze namen, want we nemen aan dat die er nog niet zijn, maar wel van de andere methoden die in de klasse List al aanwezig zijn, en van je eigen methoden.
Antwoord:

```
public void Reverse()
{
    for (int i=this.Count-2; i>=0; i--)
    {
        this.Add(this[i]);
        this.RemoveAt(i);
    }
}
public int LastIndexOf(string item)
{
    int i=this.Count-1;
    while (i>=0)
    {
        if (this[i] == item)
            return i;
        i--;
    }
    return i;
}
public bool Contains(string item)
{
    for (int i=0; i<this.Count; i++)
        if (this[i] == item)
            return true;
    return false;
}
```

10.5 *Collections*
Schrijf een methode `verwijderDubbeleGetallen` die als parameter een `List` van `int`s meekrijgt. De methode verwijdert alle dubbele getallen in de lijst die meegegeven is als parameter. Bijvoorbeeld, de lijst bevattende `0, 1, 3, 2, 1, 5, 2` wordt `0, 1, 3, 2, 5`. Let op: het return-type van deze methode is `void`!
Antwoord:

```
void verwijderDubbeleGetallen(List<int> lijst)
{
    for (int i=lijst.Count-1; i>=0; i--)
```

```
    {
        if (lijst.IndexOf(lijst[i], 0, i-1) != -1)
        {
            lijst.RemoveAt(i);
            i--;
        }
    }
}
```

10.6 *Klassen en interfaces*
Eén van de volgende drie declaraties-met-toekenningen is correct. Welke is dat, en waarom zijn de andere twee niet correct?

```
List a  = new IList();  // versie 1
IList b = new List();   // versie 2
IList c = new IList();  // versie 3
```

Beschrijf een situatie waarin de correcte versie van bovenstaande regels een voordeel heeft, vergeleken met het in ieder geval ook correcte:

```
List d  = new List();   // versie 4
```

Waarin onderscheiden abstracte klassen zich van interfaces? Geef een voorbeeld van een situatie waar je een abstracte klasse zou gebruiken. Geef ook een voorbeeld van een situatie waar je een interface zou gebruiken.
Antwoord:
In versie 1 en 3 wordt een object van `IList` gecreëerd, maar dat kan niet want `IList` is een interface en geen klasse. Versie 2 is correct, want klasse `List` implementeert de interface `IList`, en mag dus die rol vervullen.
Versie 2 is beter dan versie 4 als je de mogelijkheid wilt openhouden om in een later stadium toch voor een andere implementatie van `IList` te kiezen. De rest van het programma kan dan niet per ongeluk toch gebruikmaken van eventuele eigenschappen van `List` die niet in `IList` zijn gespecificeerd.
In een interface staan alleen maar methode-headers. In een abstracte klasse kan dat ook (als je de modifier `abstract` voor de header van de methode schrijft), maar ju kunt ook sommige methoden al wel van body voorzien. Ook kunnen er in een abstracte klasse membervariabelen worden gedeclareerd; in een interface kan dat niet.
Je gebruikt een abstracte klasse als basis van een hiërarchie, waarin sommige methoden al in termen van andere methoden geschreven kunnen worden, zoals in de klasse `Stream`, waar `Read` alsvast in termen van `ReadByte` geschreven kan worden.
Je gebruikt een interface als je wilt specificeren wat er allemaal moet kunnen, zonder je erover uit te laten *hoe* dat zou moeten. Dit is bijvoorbeeld het geval bij `ICollection`.

10.7 *Abstracte klassen*
Gegeven de volgende klassen:

```
abstract class A
{
    public abstract void methode1();
    public void methode2()
    {
        return;
    }
}
class B : A
{
    public override void methode1()
    {
        return;
    }
    public void methode3(A a)
    {
```

```
            a.methode1();
        }
    }
```

Geef voor elk van de volgende opdrachten aan of ze mogen:
Antwoord:

```
A obj; // ja, de referentie declareren mag altijd
obj = new A(); // nee, van een abstracte klasse kun je geen object aanmaken
obj = new B(); // ja, van een concrete subklasse wel
obj.methode1(); // ja, de compiler weet dat elke subklasse deze methode heeft
obj.methode2(); // ja, deze methode wordt geerfd
obj.methode3(obj); // nee, de compiler kan niet controleren dat het object deze methode heeft
B anderObject = (B)(new A()); // nee, van een abstracte klasse kun je geen object aanmaken
A nogEenObject = (A)obj; // ja, je mag altijd casten naar een "hoger" type
obj.methode3(nogEenObject); // nee, de compiler kan niet controleren dat het object deze methode heeft
A[] lijst; // ja
lijst = new A[10]; // ja, hier wordt een array referenties aangemaakt
lijst[0] = new A(); // nee, van een abstracte klasse kun je geen object aanmaken
lijst[1] = new B(); // ja, van een concrete subklasse wel
List<A> nogEenLijst = new List<A>(); // ja
```

10.8 *List en foreach*
Gegeven de volgende klasse:

```
class Tellertje
{
    int waarde = 0;
    public Tellertje(int w)
    {   waarde = w;
    }
    public void increment()
    {
        waarde++;
    }
}
```

en de volgende opdrachten in een `Applicatie`-klasse:

```
class Applicatie
{
    static void Main()
    {
        List<Tellertje> lijst = new List<Tellertje>();
        for (int i=0; i<25; i++)
            lijst.Add(new Tellertje(i));
    }
}
```

a. Schrijf een methode `ophogen` in de klasse `Applicatie` die als parameter een `IList<Tellertje>` meekrijgt en die alle tellertjes ophoogt (via de `increment`-methode). Maak een versie die de `foreach`-opdracht gebruikt, en een versie die een `for`- of een `while`-opdracht gebruikt.
Antwoord:

```
void ophogen(IList<Tellertje> lijst)
{
    for (int i=0; i<lijst.Count; i++)
        lijst[i].increment();
}
void ophogen(IList<Tellertje> lijst)
{
    foreach (Tellertje t in lijst)
        t.increment();
```

```
    }
```

b. Stel dat we in plaats van een `List<Tellertje>` nu een `EigenLijst<Tellertje>` zouden willen gebruiken, en de `EigenLijst`-klasse is onze eigen versie van een lijst, die ook de `IList` interface implementeert, wat moeten we dan veranderen aan de `ophogen`-methode zodat we hem ook met onze eigen lijstklasse kunnen gebruiken?
Antwoord: We hoeven dan niks te wijzigen, want de klasse `EigenLijst` implementeert de `IList`-interface. Alle benodigde methoden en properties, zoals de `Count`-property en de rechte-hakentoegang zijn dus geimplementeerd.

10.9 *Klassen en overerving*
Beschouw de volgende twee klassen:

```
class A {
    public void func1() { Console.WriteLine("A::func1"); }
    public virtual void func2() { Console.WriteLine("A::func2"); }
}
class B : A {
    public void func1() { Console.WriteLine("B::func1"); }
    public override void func2() { Console.WriteLine("B::func2"); }
}
```

Wat is de uitvoer van de volgende serie opdrachten?

```
A x = new A();
A y = new B();
B z = new B();
x.func1();
x.func2();
y.func1();
y.func2();
z.func1();
z.func2();
```

Antwoord:

```
A::func1
A::func2
A::func1
B::func2
B::func1
B::func2
```

10.10 *Nog meer tekstfiles*
Schrijf een programma met de volgende specificaties. Het programma wordt door de gebruiker vanaf een commandoregel opgestart, en toont zijn uitvoer ook via de console. Het programma heeft dus geen eigen window.
Bij het starten van de programma geeft de gebruiker ook één of meer namen van tekstfiles op. Van elke file wordt gerapporteerd:

- welke regel het langste is
- welke regel het meeste woorden bevat

Als er een probleem is bij het lezen van de file wordt dat gemeld. Aan het eind wordt bovendien gemeld welk van de files het meeste regels heeft.
In het programma moet er een aparte methode zijn voor het verwerken van n file. Bij het rapporteren van regels tekst, moet de geciteerde regel tussen aanhalingstekens worden getoond (zoals in het voorbeeld). Bij het rapporteren van de langste tekst moet ook de naam van de file genoemd worden (zoals in het voorbeeld).
Een woord is een aaneengesloten reeks van n of meer letters, cijfers en leestekens. Woorden worden dus gescheiden door spaties. Let op: er kunnen meerdere spaties tussen twee woorden staan, maar dat telt niet als extra woorden.
Voorbeeld: De inhoud van de file `pipo.txt` is:

```
Kort he!
```

```
Eenentwintig vijfendertig zevenenveertig.
Een twee drie vier vijf zes zeven.
```

De inhoud van de file `test.txt` is:

```
Dit is de eerste regel.
Dit de tweede.
Vanwege de waarschuwingsknipperlichtinstallatie is dit een hele lange regel geworden.
Op deze regel staan dus wel erg veel maar niet zo erg lange woorden.
Dit is de vijfde regel.
Op deze regel     staan       veel extra    spaties.
Deze zevende regel is behoorlijk lang, maar overtreft toch niet tot de langste.
```

De file `fuot.txt` bestaat niet.
De gebruiker start het programma bijvoorbeeld met:

```
TekstAnalyse pipo.txt fuot.txt test.txt
```

De output is dan:

```
Langste regel van pipo.txt is: "Eenentwintig vijfendertig zevenenveertig."
Meeste woorden in: "Een twee drie vier vijf zes zeven."
Probleem bij het lezen van fuot.txt
Langste regel van test.txt is: "Vanwege de waarschuwingsknipperlichtinstallatie is dit een hele lange reg
Meeste woorden in: "Op deze regel staan dus wel erg veel maar niet zo erg lange woorden."
De meeste regels zitten in file test.txt
```

Antwoord:

```
class TekstAnalyse
{
    static int maxn = 0;
    static string langste;
    static void Main(string[] namen)
    {
        foreach (naam in namen)
            TekstAnalyse.verwerk(naam);
        Console.WriteLine("meeste regels in " + langste );
    }
    static void TekstAnalyse(string naam)
    {   try
        {   TextReader reader = new StreamReader(naam);
            string regel, maxregel="", maxwoorden="";
            for (int n=0; (regel=reader.ReadLine()) != null; n++)
            {
                if (regel.Length > maxregel.Length)
                    maxregel = regel;
                if (regel.Split(" ") > maxwoorden.Split(" ").Length)
                    maxwoorden = regel;
            }
            Console.WriteLine("Langste regel van " + naam + " is: \"" + maxregel + "\"");
            Console.WriteLine("Meeste woorden in: \"" + maxwoorden + "\"");
            if (n>maxn)
            {    maxn = n;
                 langste = naam;
            }
        }
        catch (Exception e)
        {   Console.WriteLine("Probleem bij lezen van " + naam );
        }
    }
}
```