Uitwerking Derde deeltentamen Imperatief programmeren - versie 1
Vrijdag 9 november 2018, 11.00-13.00 uur

- Schrijf op elk ingeleverd blad je naam. Schrijf op het eerste blad ook je studentnummer en het aantal ingeleverde bladen.
- De lijst met standaardfuncties na afloop graag weer inleveren. De antwoorden komen binnenkort op Blackboard.
- Opgave 1 t/m 10 zijn meerkeuzevragen, die meetellen voor $10 \times 4 = 40$ punten.
  Opgave 11 en 12 zijn programmeervragen, die meetellen voor 20 en 40 punten.

Meerkeuzevragen: de letter van het goede antwoord volstaat.
Belangrijk: dit is versie **1** van het tentamen, vermeld dat boven je antwoorden.

1. De methode `Main` mag gedefinieerd worden met een parameter. Deze parameter bevat:

   (a) De tekstregels die door de gebruiker worden ingetikt
   (b) De files die door het programma gelezen moeten worden
   (c) • De woorden die bij het opstarten achter de programmanaam zijn ingetikt
   (d) De libraries die in het programma nodig zijn

   Toelichting op het antwoord: (a) is de string die je krijgt door aanroep van `ReadLine`. (b) lijkt aardig, omdat de ingetikte woorden vaak worden gebruikt als de namen van files die gelezen moeten worden, maar dat is niet altijd het geval: de woorden kunnen ook anders gebruikt worden. Bovendien zijn het dan de filenamen en niet de files zelf. (d) deze libraries staan in het programma achter `using`, maar dat heeft verder weinig met `Main` te maken.

2. Bepalen van de waarde van de expressie `a=>a`

   (a) geeft de waarde `true`
   (b) • geeft de identiteitsfunctie
   (c) resulteert in een exception
   (d) duurt oneindig lang

   Toelichting op het antwoord: Dit is een 'lambda-expressie', waarmee je een naamloze methode kunt definiëren. Voor de pijl staan de parameters, achter de pijl de waarde die wordt gereturnd. Met de gegeven expressie wordt dus de identiteitsfunctie aangeduid. Het antwoord `true` krijg je bij evaluatie van `a>=a` voor numerieke `a`.

3. Wat is het belangrijkste voordeel van een *virtuele* methode boven een *abstracte* methode?

   (a) Je krijgt een waarschuwing als je hem vergeet te overriden in een subklasse
   (b) Je hoeft er nog geen implementatie van te geven
   (c) • Je kunt alvast een default-gedrag vastleggen
   (d) Je kunt hem niet per ongeluk aanroepen terwijl dat nog geen zin heeft

   Toelichting op het antwoord: (a) en (b) zijn verkeerdom: het zijn voordelen van abstracte methoden boven virtuele methoden. (d) is onzin: zowel abstracte als virtuele methoden mag je juist wel aanroepen, zelfs als het nog geen zin heeft.

4. Stel dat je moet voortbouwen op een klasse die een collega-programmeur in je team geschreven heeft. Je zou een kopie van de sourcecode kunnen maken, de klassenaam veranderen, en de noodzakelijke uitbreidingen en aanpassingen maken. Maar het is beter om een subklasse te definiëren. Waarom?

   (a) Je schendt dan niet het auteursrecht van de oorspronkelijke auteur
   (b) Je kunt methoden uit de oorspronkelijke klasse aanroepen omdat de subklasse die erft
   (c) Er ontstaan minder snel fouten bij het maken van de uitbreidingen en aanpassingen
   (d) • Onderhoud van de software is dan eenvoudiger

   Toelichting op het antwoord: (d) is het goede antwoord: als er later bugs worden ontdekt in de oorspronkelijke code, moeten die tweemaal gefixt worden. (a) is onzin: je bent juist een team om samen te werken. (b) is geen verbetering: ook methoden van een gekopieerde klasse kun je aanroepen (c) is niet waar: de uitbreidingen blijven even moeilijk of makkelijk te maken

5. Een *interface* beschrijft

   (a) de opbouw van een object
   (b) • de mogelijkheden van een object
   (c) de werking van de methoden
   (d) de herdefinitie van de methoden

   Toelichting op het antwoord: De opbouw wordt beschreven door de declaraties in een klasse. De werking door de bodies van methoden in een klasse. Herdefinitie doe je in een subklasse.

6. Welk van onderstaande uitspraken is *niet* waar?

   (a) UTF8 kan dezelfde tekens opslaan als Unicode
   (b) UTF8 kost soms meer ruimte dan Unicode
   (c) UTF8 kost soms minder ruimte dan Unicode
   (d) • UTF8 codeert tekens met code 0 t/m 255 hetzelfde als Latin1

   Toelichting op het antwoord: UTF8 is een coderingsmethode, maar kan net als Unicode alle 65536 verschillende tekens opslaan. Voor Chinese tekens kost dat 3 bytes per karakter, terwijl Unicode er maar 2 nodig heeft. Voor westerse tekens kost dat 1 byte per karakter, terwijl Unicode er altijd 2 nodig heeft. Dus (a), (b) en (c) zijn correcte uitspraken. Maar (d) is niet waar (en dus het goede antwoord). Wel is het zo dat UTF8 de tekens met code 0 t/m 127 hetzelfde codeert als Ascii, maar code 128 t/m 255 krijgen nou juist codes van 2 bytes, terwijl Latin1 dat met 1 byte doet.

7. Welke declaratie is correct?

   (a) `IList<string> x = new IList<string>();`
   (b) `IList<string> x = new LinkedList<string>();`
   (c) `List<string> x = new IList<string>();`
   (d) • `ICollection<string> x = new List<string>();`

   Toelichting op het antwoord: De naam van deen interface mag nooit achter `new` staan, dus (a) en (c) vallen af. Een `LinkedList` is gek genoeg geen implementatie van `IList` (zie het klasse-diagram), dus (b) is fout. Bijna alles is wel een implementatie van `ICollection`, dus ook `List`.

8. Hoe kunnen menu-items van een typische tekstverwerker het best worden verdeeld tussen het MDI-containerwindow en het MDI-childwindow?

   (a) • Open en Exit in de container, Close en Save in het child
   (b) Open en Save in de container, Find en Close in het child
   (c) Help en Exit in de container, New en Save in het child
   (d) New en Close in de container, Help en Save in het child

9. Om een grafische animatie te maken, schrijf je een methode die steeds opnieuw `Invalidate` aanroept. Deze methode

   (a) • geef je als mee parameter bij de constructie van een `Thread`-object
   (b) geef je mee als parameter bij de aanroep van `Start` met een `Thread`-object onder handen
   (c) roep je aan met een `Thread`-object onder handen
   (d) roep je aan met een `Thread`-object als parameter

10. Het is handiger om een `List` te gebruiken in plaats van een array als je

    (a) van tevoren precies weet hoeveel elementen er zijn
    (b) • later nog elementen wilt kunnen tussenvoegen
    (c) de elementen later wilt kunnen sorteren
    (d) met een `foreach`-opdracht de elementen wilt langsgaan

    Toelichting op het antwoord: Als je van tevoren weet hoeveel elementen er zijn, kun je juist wel een array gebruiken. Sorteren en elementen langslopen zijn zowel voor arrays als string mogelijk en dus geen voordeel van lists.

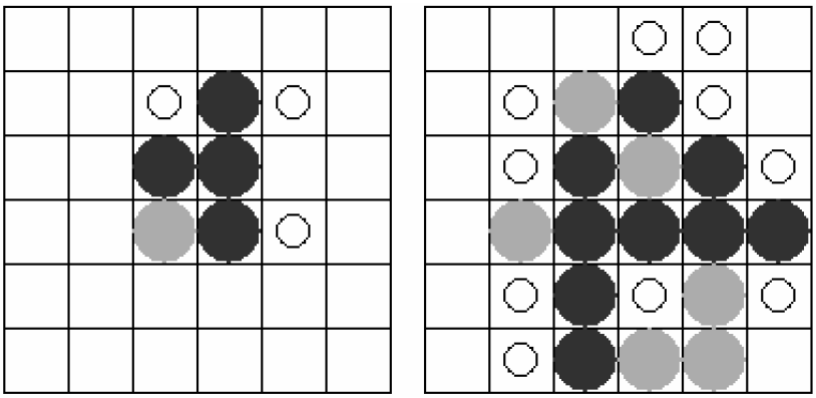
11. *(telt voor 20%)*

    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.

De opgave: Schrijf een methode

```
bool mag (int kleur, int x, int y)
```

die controleert of speler `kleur` een steen op veld (x, y) mag zetten. (De zet wordt dus nog niet uitgevoerd!).

Hint: het is toegestaan (en handig) om een extra methode te schrijven die het insluiten in één van de 8 richtingen controleert.

Antwoord:

```
bool mag (int kleur, int x, int y)
{
    if (bord[x,y]!=0)
        for (int dx=-1; dx<=1; dx++)
            for (int dy=-1; dy<=1; dy++)
                if (dx!=0 || dy!=0)
                {
                    if (test(kleur, x, y, dx, dy);
                        return true;
                }
    return false;
}
bool test (int kleur, int x, int y, int dx, int dy)
{
    for (int t=1; ; t++)
    {
        int ix = x+t*dx;
        int iy = y+y*dy;
        if (ix<0 || ix>=6 || iy<0 || iy>=6)
            return false;
        int s = bord[ix, iy];
        if (s==0)
            return false;
        if (s==kleur)
            return t>1;
    }
    return false;
}
```

12. *(telt voor 40%)*

Bekijk het programma `Blokken`, waarvan hiernaast een screenshot staat. In het window zijn een tekstveld en twee buttons zichtbaar. De gebruiker kan met de muis steeds een punt aanklikken. Op dat punt verschijnt, gecentreerd, een kleine zwarte cirkel.
Als de gebruiker een tweede keer klikt, verschijnt er een blok vanaf het eerste punt tot het tweede punt. Het cirkeltje verdwijnt weer. De kleur van het blok wordt gespecificeerd door het getal dat in het tekstveld staat ingevuld op het moment van de tweede klik: een grijstoon tussen 0=zwart en 255=wit. Je mag zonder controle aannemen dat in het tekstveld een getal tussen 0 en 255 is ingevuld.

Bij een derde klik verschijnt er weer een cirkeltje, dat bij de vierde klik weer wordt aangevuld tot blok, enzovoorts.

Het aantal blokken dat getekend kan worden mag niet aan een maximum zijn gebonden. Let op dat het tweede aangeklikte punt ook links en/of boven het eerste punt kan liggen.

Hieronder is al een deel van het programma gegeven.

```
using System;     using System.Windows.Forms; using System.Collections.Generic;
using System.IO;  using System.Drawing;       using System.Threading;

public class Blok
{   // opgave a
}

public class Blokken : Form
{   const string naam = "tekening.txt";
    TextBox tb;

    public Blokken()
    {   this.Text = "Blokken";
        this.tb  = new TextBox(); tb.Text = "0";       tb.Location = new Point( 10,10);
        Button b1 = new Button(); b1.Text = "opslaan"; b1.Location = new Point(100,10);
        Button b2 = new Button(); b2.Text = "inlezen"; b2.Location = new Point(200,10);
        this.Controls.Add(tb);
        this.Controls.Add(b1);     b1.Click += this.opslaan;
        this.Controls.Add(b2);     b2.Click += this.inlezen;
        this.Paint += this.teken;  this.MouseClick += this.muisklik;
    }
    // ontbrekende member-variabelen en methoden: opgave b en c
}

public class Program
{   static void Main()
    {   Application.Run(new Blokken());
    }
}
```

Vul hierop het volgende aan:

(a) Schrijf een hulpklasse `Blok`, zo dat in een object van die klasse alle gegevens die nodig zijn om een blok te kunnen tekenen beschikbaar zijn. Maak een constructormethode die zo'n object van beginwaardes voorziet, een methode `ToString` die de waarden 'inpakt' in een string, en een tweede constructormethode die zo'n string weer 'uitpakt'.

Antwoord:

```
public class Blok
{
    public int x, y, b, h, k;

    public Blok(Point p1, Point p2, int kl)
    {
        x = Math.Min(p1.X, p2.X);
        y = Math.Min(p1.Y, p2.Y);
        b = Math.Abs(p2.X - p1.X);
        h = Math.Abs(p2.Y - p1.Y);
        k = kl;
    }
    public Blok(string s)
    {
        string[] v = s.Split();
        x = int.Parse(v[0]);
        y = int.Parse(v[1]);
        b = int.Parse(v[2]);
        h = int.Parse(v[3]);
        k = int.Parse(v[4]);
    }
    public override string ToString()
    {
        return x + " " + y + " " + b + " " + h + " " + k;
    }
}
```

(b) Schrijf de methoden `muisklik` en `teken` in de klasse `Blokken`, en geef de declaraties van de daarvoor benodigde member-variabelen.

Antwoord:

```
List<Blok> blokken = new List<Blok>();
Point punt;
int n = 0;

void muisklik(object o, MouseEventArgs mea)
{
    if (n % 2 == 0)
        punt = mea.Location;
    else
        blokken.Add(new Blok(punt, mea.Location, int.Parse(tb.Text)));
    n++;
    this.Invalidate();
}
void teken(object o, PaintEventArgs pea)
{
    Graphics g = pea.Graphics;
    foreach (Blok blok in blokken)
    {   Color kleur = Color.FromArgb(blok.k, blok.k, blok.k);
        g.FillRectangle(new SolidBrush(kleur), blok.x, blok.y, blok.b, blok.h);
    }
    if (n % 2 == 1)
```

```
            g.FillEllipse(Brushes.Black, punt.X-5, punt.Y-5, 10, 10);
    }
```

(c) Als de gebruiker op de knop 'Opslaan' drukt, wordt de toestand van de tekening opgeslagen in een file waarvan de naam in de constante `naam` staat. Als de gebruiker op de knop 'Inlezen' drukt, wordt de huidige tekening vervangen door de opgeslagen tekening. Schrijf de methoden `opslaan` en `inlezen` die daarvoor nodig zijn.

Antwoord:

```
void opslaan(object o, EventArgs ea)
{
    StreamWriter w = new StreamWriter(naam);
    foreach (Blok blok in blokken)
        w.WriteLine(blok.ToString());
    w.Close();
}
void inlezen(object o, EventArgs ea)
{
    blokken.Clear();
    StreamReader r = new StreamReader(naam);
    string regel;
    while ((regel = r.ReadLine()) != null)
        blokken.Add(new Blok(regel));
    r.Close();
    this.Invalidate();
}
```