# Hertentamen Object-oriëntatie (NWI-MOL098)

Vrijdag 12 mei 2017, 18:00-21:00

Dit is een gesloten boek tentamen. In totaal kun je voor dit tentamen 58 punten halen.

## 1 Enkele kleine opgaven **totaal 14 punten**

We beginnen met enkele korte vragen. Voor onderdeel a, b en c kun je ieder 3 punten en voor onderdeel d 5 punten krijgen.

**1.a**) Wat is het verschil tussen een *parameter* en een *lokale variabele*? Geef van beide een voorbeeld.

**1.b**) Bekijk de volgende declaratie:

```
private double averageWeight (List<Body> bodies)
```

Beschrijf **alle** vijf onderdelen in deze declaratie kort en krachtig.

**1.c**) Wat is het verschil tussen een statische methode (soms ook wel *service* genoemd) en een 'gewone' (niet-statische) methode?

**1.d**) Bekijk de volgende methode:

```
public static int calculateSomething( int n ) {
    int j = 3;
    int i = 0;
    int s = 1;
    while ( s <= n ) {
        s = s + j;
        j = j + 2;
        i++;
    }
    return i;
}
```

Wat is het resultaat van de aanroep `calculateSomething`$(16)$? Gebruik eventueel een tracing table. En wat is het resultaat van `calculateSomething` $(n*n)$, voor willekeurige niet-negatieve $n$?

## 2 Complexiteit: graafalgoritmen **totaal 12 punten**

In een wijk van een of andere stad in Nederland is het sinds langere tijd onrustig. Het aantal incidenten (inbraken, vechtpartijen, etc) is significant hoger dan in vergelijkbare gemeenten, vandaar dat de politie besloten heeft de controle in de wijk te intensiveren. Hiertoe gaat men over op het plaatsen van bewakingscamera's zodat *alle* straten in beeld gebracht kunnen worden. Het is hiervoor noodzakelijk dat camera's op kruispunten geplaatst worden en wel zodanig dat *ieder* kruispunt door tenminste een camera in beeld gebracht wordt. Het bereik van een camera loopt van de plek waar de camera staat tot de direct aangrenzende kruispunten.

Daarnaast moeten de camera's verbonden worden met de centrale. Om in vervolgprojecten ook nog andere apparatuur op het netwerk aan te kunnen sluiten wil men ervoor zorgen dat elk kruispunt van de wijk toegang heeft tot dit netwerk. De bekabeling die hiervoor nodig is wordt direct langs de wegen gelegd en, om kosten te besparen, wil men de totale lengte van de hiervoor benodigde kabel *minimaliseren*.

Stel we hebben de wijk gerepresenteerd als een (ongerichte) graaf waarin knopen kruispunten voorstellen en verbindingen wegen tussen twee kruispunten. Iedere verbinding is gelabeld met de lengte van de bijbehorende weg.

**2.a**) Abstraheer van de concrete situaties en beschrijf beide problemen in termen van de graafrepresentatie. Oftewel, met welke twee graafproblemen/algoritmen hebben we hier te maken?

**2.b**) Geef voor één van beide problemen een oplossing in de vorm van een algoritme gespecificeerd door middel van een 'hoog niveau' flowchart (dat wil zeggen, een flowchart zonder veel details). Houd je daarbij wel aan de

**2.c**) Wat is het fundamentele verschil tussen de beide problemen wanneer we de complexiteit van mogelijke oplossingen bekijken? Gebruik in je antwoord termen als Niet-Polynomiaal of Exponentieel.

## 3 Loops: tel het aantal gemeenschappelijke elementen **totaal 10 punten**

In deze opdracht werken we met twee lijsten van eieren. Ieder ei (type `Egg`) heeft een waarde die je met **public int** `getValue` () kunt opvragen.

**3.a**) Definieer een methode

```
public int gemeenschappelijk(List<Egg> eiLijst1, List<Egg> eiLijst2)
```

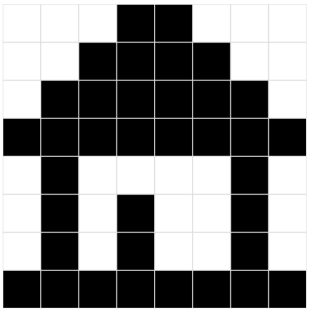
die voor de meegegeven lijsten van eieren bepaalt hoeveel eieren beide lijsten gemeenschappelijk hebben. Dat wil zeggen, je dient te bepalen hoeveel eieren uit lijst `eiLijst1` gelijk zijn aan eieren uit lijst `eiLijst2`, waarbij gelijk betekent: hebben dezelfde waarde. Enkele `List`-methodes die wellicht van pas kunnen komen zijn:

```
int size ()
Object get(int index)
void remove (Object o)
boolean isEmtpy ()
```

Je mag zonodig aannemen dat in iedere lijst afzonderlijk geen dubbele eieren voorkomen.

## 4 Compressie: Run-length encoding **totaal 10 punten**

Voor het comprimeren van plaatjes in pixel-formaat wordt soms gebruik gemaakt van *run-length encoding* (RLE). We gaan in deze opdracht uit van zwart-wit plaatjes. In plaats van deze plaatjes pixelgewijs te representeren worden bij RLE hele reeksen van dezelfde kleur door hun lengte gerepresenteerd. Onderstaand plaatje geeft hiervan een voorbeeld; bij het tellen (rechterplaatje) wordt begonnen met het aantal witte pixels/blokjes.



(a) Pixelplaatje

3, 2, 3
2, 4, 2
1, 6, 1
0, 8
1, 1, 4, 1, 1
1, 1, 1, 1, 2, 1, 1
1, 1, 1, 1, 2, 1, 1
0, 8

(b) RLE

Figuur 1: Run-length encoding

Als we zo'n plaatje over het internet willen versturen, dan moeten we voor de aantallen zelf die in deze representatie voorkomen (3,2,3,...) natuurlijk ook weer bits gebruiken.

**4.a**) Waarom is het nodig om van tevoren een afspraak te maken over een vaste maximale grootte van deze getallen.

**4.b**) Hoeveel bits per aantal hebben we dan nodig om bovenstaand voorbeeld geheel over te kunnen sturen?

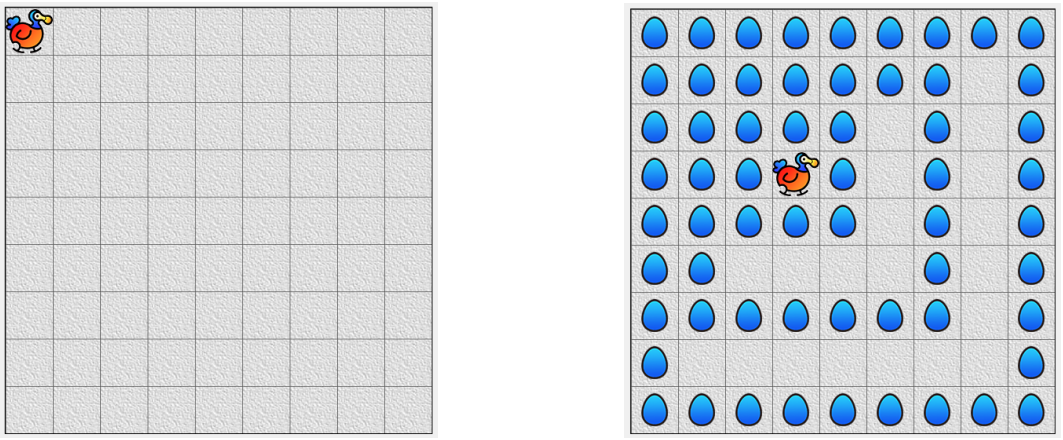
**4.c**) Stel we ook hebben afgesproken hoeveel pixels breed een plaatje is. We hoeven dan bij het tellen niet steeds op te houden aan het einde van de regel, maar kunnen dan doorgaan met tellen aan het begin van de volgende regel. Hoe ziet de resultaatreeks er nu uit als we RLE op bovenstaand plaatje toepassen.

**4.d**) Stel we gebruiken 4 bits voor de aantallen. Ook hebben we een zo gunstig mogelijk plaatje (een plaatje dat maximale compressie oplevert). Hoe ziet dat plaatje er uit en wat is het resultaat na RLE? Ook hier dien je gewoon door te gaan met tellen op de volgende regel.

**4.e**) En wat is het meest ongunstige plaatje? Hoeveel bits heb je nodig om het totale resultaat na RLE te kunnen representeren?

## 5 Algoritmen: Dodo tekent een compositie van vierkanten **totaal 12 punten**

Dodo staat in de wereld (zie onderstaand linkerplaatje). Ze kijkt naar het oosten. Haar doel is om het eierenpatroon zoals getoond wordt in het rechterplaatje, te produceren.



(a) Beginsituatie (b) Eindsituatie

Figuur 2: Dodo maakt een kunstwerk

**5.a**) Geef, in Java, een algoritme dat het rechterplaatje oplevert. Dit algoritme dient generiek te zijn, dat wil zeggen, het moet werken voor willekeurige groottes van de wereld. Wel mag je aannemen dat de wereld vierkant is. De grootte van de wereld kun je opvragen met de gecombineerde methode-aanroepen `getWorld ()`. `getWidth ()`. Hint: Definieer eerst een methode **void** `vierkant (` **int** `grootte )` die een vierkant tekent van de meegegeven grootte.

– EINDE TENTAMEN –