PROJECT VOSSEN & KONIJNEN

ADVIESRAPPORT



ICT Bureau Ambler  
Frank Noorlander, Tsjeard de Winter en  
Rick van der Poel

PROJECT VOSSEN & KONIJNEN   
ADVIESRAPPORT

**Auteurs:** ICT Bureau AmblerFrank Noorlander, Tsjeard de Winter en Rick van der Poel   
  
**Opdrachtgever**: de heer H. Ripper van SimulationFirst   
  
**Begeleiding:** de heer B. Heijne van Hanzehogeschool Groningen  
 **Datum:**  02-2015  
  
**Locatie**: Groningen  
  
**Bron afbeelding voorpagina:**  
Rick van der Poel. (2015, 30 januari). Screenshot Vossen & Konijnen applicatie [Leertaak 2].  
Hanzehogeschool Groningen, opleiding Informatica.

# Samenvatting

In dit document vindt u de documentatie van het project Vossen & Konijnen. Deze samenvatting geeft u een indruk van de werkzaamheden die verricht zijn tijdens het project.

De opdrachtgever heeft ons van een incomplete applicatie voorzien waarin het gedrag van vossen en konijnen wordt gesimuleerd. De simulatie miste echter nog veel functionaliteiten. De originele applicatie kan alleen vossen en konijnen simuleren en de opdrachtgever wil graag dat ook andere dieren gesimuleerd kunnen worden. Daarnaast moet de GUI van de applicatie zo uitgebreid worden dat er meer informatie over de simulatie in beeld komt voor de gebruiker. In de gewenste GUI moet het ook mogelijkheid zijn de parameters van de simulatie te kunnen wijzigen. De simulatie wordt verder nog uitgebreid met diverse andere onderdelen zoals plaatjes, geluiden en ziektes.  
  
Als eerste is het programma zo aangepast dat het gebruikt kan worden door meer programmeerprogramma’s. Ook is de GUI uitgebreid met een menubalk en knoppen voor het bedienen van de applicatie. Daarnaast zijn er verschillende unittests aangemaakt die testen of de functionaliteit van functies uit het originele programma niet beïnvloed worden door de aanpassing die gaan komen.

De eerste grote uitbreiding van het programma was het introduceren van nieuwe wezens aan de simulatie. De klassen zijn zo aangepast dat er nieuwe dieren en mensen aan de simulatie toegevoegd kunnen worden. Vervolgens zijn beren en jagers toegevoegd aan de simulatie die elk hun eigen invloed hebben op de simulatie. Beren jagen op vossen en konijnen en jagers jagen op de dieren waar er teveel van zijn. De wijzigingen in de applicatie wordt ondersteund met verschillende UML-diagrammen die gemaakt zijn tijdens en na het project. Deze diagrammen geven u een goed beeld van de werkzaamheden die zijn verricht tijdens het project.

Daarna waren de nieuwe simulatie weergaven aan de beurt. Om deze weergaven beter te kunnen implementeren is de hele applicatie gerefactored. Hierdoor is er een betere cohesie tussen de klassen en word ook het implementeren van nieuwe functionaliteit veel makkelijker. Om dit te realiseren is er gebruik gemaakt van package structuur en is er een MVC-model toegepast tijdens het refactoren.   
  
Na het refactoren is het parameter paneel er ingekomen waarmee het gedrag van de dieren beïnvloed kan worden. Ook is het programma zo aangepast dat het nu ook rekening houd met de voedselvoorraad van de konijnen. Als er veel konijnen zijn komen er minder nakomelingen dan wanneer er weinig konijnen zijn omdat er dan meer gras beschikbaar is. Verder is het programma nog uitgebreid met geluiden en een toerist die af en toe even langs komt om een brand te stichten.

Tot slot zijn er nog een aantal bonus uitbreidingen toegevoegd waaronder een konijnen ziekte die ook invloed heeft om de andere dieren. Konijnen die deze ziekte opvangen gaan vervolgens dood na 5 dagen. Zodra een ziek konijn opgegeten wordt door een vos raakt de vos geïnfecteerd. Word de vos vervolgens weer opgegeten door een beer dan wordt de beer geïnfecteerd en zal ook de beer last hebben van de ziekte.   
  
Door al deze verbeteringen is de applicatie efficiënter en veelzijdiger geworden met veel ruimte tot nog meer uitbreidingen. De gebruiker kan makkelijker met de simulatie overweg en krijgt ook veel meer informatie terug tijdens de simulatie.

die wij als projectgroep hebben uitgebreid en verbeterd. + doelstelling

De nieuwe applicatie is voorzien van een uitgebreide GUI zodat de simulatie gemakkelijk en correct weergeven wordt op het scherm. In deze GUI is het ook mogelijk om parameters van de dieren en de omgeving te veranderen om de simulatie zo realistisch mogelijk te maken.

De nieuwe applicatie is ook voorzien van verschillende Junittests. Met deze tests wordt het onderhouden en uitbreiden van de applicatie vereenvoudigd.

De nieuwe applicatie is tevens uitgebreid met verschillende dieren, jagers en voedselbronnen. Met deze uitbreiding is de simulatie een stuk realistischer geworden.

De packagestructuur van de nieuwe applicatie is nu volgens MVC- standaarden opgezet. Met deze verandering is de overzichtelijkheid van de code enorm verbeterd.

Tot slot is de GUI van de nieuwe ? applicatie uitgebreid met verschillende grafieken en afbeeldingen. Deze uitbreiding zorgt ervoor dat u de simulatie gemakkelijker en beter kunt volgen.

Meer informatie over de hierboven besproken onderwerpen vindt u in de verschillende hoofdstukken in dit document.

Inhoudsopgave

[Samenvatting 2](#_Toc409969070)

[Verklarende woordenlijst 4](#_Toc409969071)

[Voorwoord > niet in inhoudsopgave 5](#_Toc409969072)

Inleiding

[1. Probleemstelling 7](#_Toc409969073)

Hoofdstukken + titels

[2. Analyse huidige situatie 8](#_Toc409969074)

[3. Uitbreiding 1 9](#_Toc409969075)

[4. MVC 1](#_Toc409969076)3

[5. Uitbreiding 2 1](#_Toc409969077)6

[6. Bonus uitbreidingen 1](#_Toc409969078)7

[7. Conclusie 1](#_Toc409969079)8

[8. Advies / aanbevelingen / actieplan 1](#_Toc409969080)9

[9. Noten](#_Toc409969081) 20

[10. Literatuurlijst](#_Toc409969082) 21

[Bijlage 1](#_Toc409969083) 22

[Bijlage 2](#_Toc409969084) 23

[Register](#_Toc409969085) 24

# Verklarende woordenlijst > alles via tabs laten lijnen, recht onder elkaar. Vergelijkbaar formuleren. Alles met een hoofdletter.

Refactoren Een programma of een stuk code herschrijven zodat dit efficiënter wordt.   
  
BlueJ Programma om Java in te programmeren   
   
Eclipse Programma om o.a. Java in te programmeren  
  
GUI De grafische gebruikersomgeving waarin de gebruiker werkt  
  
V&K Vossen & Konijnen

MVC Een programmeermodel waarin de software uit de onderdelen model, view en controller bestaan.

Model Verwerkt alle informatie.

View Toont alle informatie van het model op het scherm.

Controller Ontvangt alle gebruikersinvoer en geeft dit door aan het model.

Package Geeft een groep bestanden weer. Een package kan ook andere packages bevatten.

Subclasse Een klasse die een andere klasse uitbreidt.

# Voorwoord

Voorwoord > Dit adviesrapport is geschreven in het kader van een opdracht voor het vak... aan de Hanzehogeschool enz. + bedanken.

Inleiding

Je kruipt hier in de huid van het ICT-bureau en je schrijft richting de lezer: de opdrachtgever, docent en geïnteresseerde leek. Professioneel en zakelijk formuleren.

Inleiding toespitsen op de opdracht.

Verzin een naam voor jullie bureau en een naam voor de opdrachtgever, bijvoorbeeld Pets & Partners + ondertitel. Je introduceert hier o.a. de opdrachtgever, de applicatie, de aanleiding voor de opdracht en de opdracht en de doelstelling. Je geeft aan dat je de complete werkwijze stap voor stap in het adviesrapport beschrijft.

In de inleiding hoort te staan:

> aanleiding + beschrijving onderwerp (aanleiding: de opdrachtgever Pets & Partners (fictieve naam) heeft de wens de applicatie van.. te verbeteren, heeft behoefte aan.. teneinde ((= doelstelling))...;

> probleemstelling (opdracht/vraagstelling, eventuele deelvragen, doelstelling);

> operationalisering gebruikte begrippen (maak alle gehanteerde begrippen meetbaar en werkbaar > afbakenen).

> vooruitblik.

Hoofdstuk 1 beschrijft.... In hoofdstuk 2 komt ... aan de orde... Daarna behandelt hoofdstuk 3 .... Vervolgens komt in hoofdstuk 4.... aan bod enz. Het adviesrapport eindigt met een conclusie en advies.

IN de opdracht staan vragen > die kun je eventueel per hoofdstuk behandelen.

Of: deel het probleem op in een van de tekststructuren die ik uitgedeeld heb.

**PAK DE BEOORDELINGSCHECKLIST erbij en volg die!**

Inleiding

Voor u ligt het adviesrapport   
Voor u ligt het reflectieverslag ?? > het is een adviesrapport waarin jullie verslag uitbrengen van de verbeteringen die jullie aangebracht hebben aan de applicatie, in opdracht van de fictieve opdrachtgever o.i.d.

van het project Vossen & Konijnen. Het project wordt uitgevoerd in de tweede periode van het eerste schooljaar, in opdracht van de Hanzehogeschool. De projectleden zijn: Frank Noorlander, Rick van der Poel en Tsjeard de Winter.

Dit verslag zal een reflectie zijn op de werkzaamheden die we als projectgroep hebben verricht om dit project tot een succes te brengen. Meer informatie over het project vindt u in de volgende hoofdstukken. > naar het voorwoord

In de inleiding hoort te staan:

> aanleiding + beschrijving onderwerp (de opdrachtgever Pet & Co (fictieve naam) heeft de wens de applicatie van.. te verbeteren, heeft behoefte aan...;

> probleemstelling (opdracht/vraagstelling, eventuele deelvragen, doelstelling);

> afbakening/definiëring gebruikte begrippen;

> vooruitblik.

In het eerst volgende hoofdstuk vindt u de probleemstelling. De probleemstelling wordt gevolgd door een analyse van de huidige situatie. Daarna volgen de verschillende werkzaamheden gesorteerd per projectonderdeel > benoemen per hoofdstuk. Hoofdstuk 1 beschrijft.... IN hoofdstuk 2 komt ... aan de orde... Daarna behandelt hoofdstuk 3 .... Vervolgens komt in hoofdstuk 4.... aan bod enz.

Na de projectonderdelen > deze hoofdstukken volgt een conclusie met ~~eventueel~~ een advies voor de opdrachtgever. Na de conclusie komen er nog een aantal hulppagina’s (noten, bijlagen en een literatuurlijst). > Alles volledig uitschrijven! Na de conclusie volgen..

# Probleemstelling > zie boven > mijd de we-vorm.

Onze opdrachtgever; de Hanze Hogeschool heeft veel verzoeken gekregen om een applicatie te maken waarmee het > waar verwijst het naar?? de levenscyclus van dierenpopulaties kan simuleren. Omdat er tegenwoordig veel meer aandacht is voor de natuur en het behouden ervan, zijn er verschillende partijen geïnteresseerd in zo’n applicatie. Te denken valt aan + voorbeelden.

De opdrachtgever heeft zelf al een poging gedaan om een dergelijke applicatie te maken. > Professioneler formuleren. Deze applicatie genaamd ‘*Vossen & Konijnen’* voldoet echter niet aan de eisen om een goede simulatie te genereren.

Hieronder volgt een opsomming van de problemen met de huidige applicatie:

* De applicatie heeft geen GUI.
* De applicatie heeft geen Junittests.
* Er zijn nog geen UML diagrammen van de applicatie.
* De applicatie is niet uitgebreid genoeg.
* De packagestructuur van de applicatie is niet volgens MVC.
* De applicatie heeft extra views nodig.
* De parameters van de diersoorten moeten in de GUI aangepast kunnen worden.
* De applicatie moet ondersteund worden met afbeeldingen en/of geluiden.

.

Als projectgroep hebben we de taak gekregen om de huidige applicatie van de opdrachtgever aan te vullen en te verbeteren om tot een realistisch resultaat te komen. > Wat verstaan jullie onder realistisch? Begrip operationaliseren. De meegegeven applicatie is momenteel erg simpel, [wat verstaan jullie onder simpel?] dat betekent dat we ons momenteel alleen zullen gaan richten op een bosgebied met daarin vossen en konijnen. Later tijdens dit project zal de applicatie echter uitgebreid worden met meer functionaliteiten. > Je schrijft een verslag achteraf. Geen zullen.

Doel >? Wat willen jullie bereiken met het verbeteren van deze applicatie?

Als de applicatie een succes wordt, kan het gebruikt kunnen worden om veel verschillende dierenpopulaties te simuleren en daarmee een enorme schat aan informatie te verkrijgen. Deze informatie kan op zijn beurt helpen de natuur en haar inwoners te behouden.

# Analyse huidige situatie > Mijd de we-vorm. Elk hoofdstuk een titel geven + beginnen met een inleiding. Wat komt er in dit hoofdstuk aan de orde? Wat wordt achtereenvolgens behandeld? Waarom komt dit aan de orde (gezien de opdracht)?

Betere alinea's maken + tussenkopjes. Leid alles in en licht alles toe. Jullie weten meer dan de lezer.

In de huidige versie van het programma dat we hebben gekregen van de opdrachtgever is het mogelijk om via een visuele simulator het leven van vossen en konijnen te simuleren.

Steeds benoemen of het gaat om de huidige of de gewenste situatie . Waar bestaat de huidige applicatie uit?

De vossen en konijnen hebben de volgende eigenschappen:

* Beide dieren hebben een maximale leeftijd. Als deze leeftijd overschreden wordt zullen de dieren verdwijnen uit het programma.
* Beide dieren hebben de mogelijkheid om te paren.
* De vossen kunnen de konijnen opeten als deze naast elkaar komen te staan in de simulator.
* De vossen kunnen dood gaan als ze te lang geen konijnen gegeten hebben.

Zoals hier boven al blijkt, is de applicatie nog erg kaal. Wat is kaal? De enige overlevingscondities waar de applicatie momenteel rekening mee houdt zijn de leeftijd en de honger. Er wordt bijvoorbeeld geen rekening gehouden met weersomstandigheden, andere dieren die hoger in de voedselketen staan zoals beren of jagers, vervuiling enzovoort. > Meer uitleggen, toelichten!! Onderwerp inleiden, ook paragrafen of subonderwerpen inleiden.

Ook heeft de huidige applicatie nog geen GUI. Door het gebrek van een GUI is het erg moeilijk om de simulatie te volgen en dat is juist de bedoeling van de applicatie.

Het doel van deze applicatie is om de levenscyclus van dierenpopulaties zo goed en realistisch mogelijk te simuleren. Om dat doel te kunnen realiseren moet de applicatie uitgebreid worden met nieuwe elementen.

Elk hoofdstuk eindigen met een slotpassage / slotalinea (samenvatting van het voorafgaande).

1. Uitbreiding 1 Zie boven. Overal bronvermelding, in de tekst (auteur + jaartal).

# De originele versie van de applicatie is verbeterd door een aantal wijzigingen door te voeren en daarna nieuwe onderdelen toe te voegen. In dit hoofdstuk worden de wijzigingen besproken die in de eerste week zijn toegepast. **Applicatie aangepast zodat het gebruikt kan worden buiten BleuJ** Als eerst is de applicatie aangepast zodat het ook in andere programma’s gebruikt kan worden. Hierdoor is het niet alleen afhankelijk van een programma zoals BleuJ. Door het toevoegen van een main methode kan het programma ook geopend worden in andere programmeer programma’s zoals bijvoorbeeld Eclipse. **Ontwikkeling van een simpele GUI** De applicatie is uitgebreid met een simpele GUI waardoor gebruikers acties kunnen uitvoeren binnen in de applicatie. Ook is er een menu aangemaakt waar basis functionaliteiten ingezet kunnen worden zoals bijvoorbeeld het afsluiten van de applicatie. Daarnaast is er een menu gemaakt met twee knoppen die het uitvoeren van de stappen regelen en een reset knop om te simulatie opnieuw te starten.

Figuur 1 – Plaatje van de nieuwe Simulatie GUI

**Unittests**Om wijzigingen in het programma te controleren op fouten zijn er ook een aantal unittests aangemaakt. Deze testen kunnen worden uitgevoerd als de applicatie is gewijzigd om te controleren of bepaalde functionaliteit van de applicatie nog steeds goed werkt. Slaagt de applicatie er niet in om door de test te komen dan is er te zien dat er iets mis de code. Omdat unittests specifieke onderdelen controleren kan ook het probleem snel terug gevonden worden.

De volgende unittests zijn gemaakt waarbij vooral het gedrag van de dieren wordt getest:  
- Worden de dieren ouder per stap?   
- Wordt een konijn opgegeten als het naast een vos staat?  
- Sterft een vos ook als er niets te eten is doordat het ingesloten zit?  
  
**Klasse- en sequentiediagram na deze uitbreidingen**In het klassendiagram van de applicatie hieronder ziet u hoe de applicatie er uit ziet na het toevoegen van de veranderingen. In het klassendiagram zijn alle klassen opgenomen samen met alle velden en methodes die er bij horen. Ook ziet u welke klassen iets met elkaar te maken hebben door de lijnen die de klassen met elkaar verbinden.   
  


Figuur 2 – Klassendiagram van de V&K applicatie na het toevoegen van de laatste paar uitbreidingen

Naast een klasse-diagram is ook een sequentie-diagram gemaakt. In dit diagram ziet u de interacties tussen verschillende objecten van de applicatie.   


Figuur 3 - Sequentie-diagram

**Toevoeging van een Actor interface, beren en jagers**Omdat het programma met meerdere diersoorten en zelfs met mensen moet kunnen werken is ook de structuur van de applicatie aangepast zodat dit makkelijker gedaan kan worden in de toekomst. Om dit te realiseren is er een Interface genaamd Actor toegevoegd.  
   
Vervolgens is de klasse Animal abstract gemaakt en is er ook een abstracte klasse Human toegevoegd. Deze klassen implementeren de klasse Actor en zijn op hun beurt weer de super klasse geworden van de wezens die daar bij horen. Om duidelijk de veranderingen weer te geven is er een klassendiagram gemaakt die deze veranderingen weergeeft.   
  
  
****

Figuur 4 - klassendiagram (Actoren) van de gewenste situatie

1. MVC Check overal de komma's, samenstellingen en d's en t's. Vermijd de hulpwerkwoorden willen en zullen.

De applicatie zoals deze aangeleverd is, bevatte naast het gebrek aan functionaliteit ook gebrek aan flexibiliteit. Wanneer er een functie toegevoegd of juist gewijzigd moest worden, zou er te veel veranderd moeten worden waardoor dit veel tijd in beslag zou nemen. Er is gekozen om het programma flexibeler te maken door het MVC-model toe te passen. Door dit model verandert er niks aan het uiterlijk van de applicatie maar het is wel duidelijk merkbaar tijdens het wijzigen of toevoegen van functies.

**Wat is MVC?**

MVC staat voor *Model, View, Controller* en is een ontwerppatroon dat het ontwerp van toepassingen opdeelt in drie onderdelen met elk hun eigen verantwoordelijkheden. Door het opdelen wordt de leesbaarheid en herbruikbaarheid van de code vergroot.

Het belangrijkste onderdeel is het *model*. Deze zorgt voor de informatie door de applicatie logica toe te passen op de data. Het model weet welke *views* bij hem horen en geeft steeds een melding aan al deze *views* dat de informatie veranderd is.

De *view* zorgt dat de informatie op het scherm getoond wordt. Wanneer de *view* gemaakt wordt, geeft deze door aan een *model* dat hij van zijn informatie afhankelijk is. De *view* doet zelf geen verwerking van de gegevens en geeft ook niks aan het *model* door.

De *controller* zorgt voor de invoer van gebruikers. Alle knoppen, tekstvelden, menu’s, etc. zijn dus van de controller. De controller wordt aan één of meerdere *models* verbonden en geeft elke gebruikers invoer direct door aan het *model* zonder iets met de informatie te doen.

Wanneer deze drie onderdelen goed geïmplementeerd zijn kan je heel makkelijk een nieuwe view toevoegen of weghalen zonder dat de applicatie daardoor niet werkt. Maar dit geldt ook voor controller en view hoewel deze vaak wat in elkaar verwikkeld zijn, de kunst is om dit zo beperkt mogelijk te houden.

**MVC in Vossen en Konijnen**

In de applicatie moest er op minstens vijf plekken code gewijzigd worden voordat er een nieuwe view toegevoegd kon worden. Een nieuwe view zou bijvoorbeeld een cirkel- of staafdiagram kunnen zijn. Daarom is er in de vernieuwde applicatie gekozen om het MVC-model te implementeren. Dit is gedaan door eerst een duidelijke packagestructuur te maken en daarna de bestaande classes op te splitsen in de drie onderdelen.

De packagestructuur is veranderd om meer overzicht te krijgen en optimaal gebruik te kunnen maken van de opties die Java biedt. De eerste drie packesages (hanze > PIV1E > leertaak2) zijn om duidelijk te maken van welk bedrijf de code is en waar de code voor dient (zie figuur 4.1). In dit geval is de code van de Hanze en het project is leertaak2. Binnen leertaak2 zijn de drie onderdelen van het MVC-model opgenomen in de packages model, view en controller. Alle packages met uizondering van de main-package ondersteuen de *model*.



Figuur 5 De packagestructuur van V&K

Na het aanbrengen van de packagestructuur kon deze gevuld worden. Veel klassen bestonden uit delen die zowel de UI deden als het verwerken van de informatie die getoond moest worden. Deze klassen moesten uit elkaar worden gehaald en opgedeeld worden in hun eigen onderdelen, *model* en *view* in dit geval.

Wanneer alle klassen één onderdeel van het MVC-model bevatten kunnen deze aan elkaar gekoppeld worden. Dit wordt gedaan in de klasse *Simulator.java* in de main-package. Hieronder ziet u een fragment code uit Simulator.java waarin de onderdelen aan elkaar gekoppeld worden.

simulation = new SimulationModel(DEFAULT\_DEPTH, DEFAULT\_WIDTH);

view = new SimulatorView(DEFAULT\_DEPTH, DEFAULT\_WIDTH, simulation);

…

graph = new GraphView(140, 140, 100, simulation);

…

pie = new PieView(140, 140, simulation);

…

chart = new ChartView(140, 140, simulation);

…

RunController controller = new RunController(simulation);

menuController = new MenuController(simulation);

Tekstfragment 1 Codefragment uit Simulator.java

Eerst wordt het model gemaakt, *simulation* in dit geval. Deze voert alle berekeningen uit en maakt houdt bij wat er in welk hokje van de simulatie zit. Daarom krijgt deze het aantal vakjes waaruit de simulatie bestaat mee.

Daarna wordt de belangrijkste view aangemaakt. *View* laat de informatie die *simulation* heeft zien, om deze informatie op te halen moet de view weten van welk model hij deze vandaan moet halen. Daarom krijgt *view* als laatste parameter *simulation* mee. Dan worden er drie views gemaakt die alle informatie op een overzichtelijke manier weergeven. Elke view heeft een afmeting nodig en het model waarvan deze zijn informatie vandaan moet halen. Die drie views zijn: *graph*, *pie* en *chart*. Als laatst worden er 2 controllers aangemaakt, deze hoeven allen te weten waar ze de gebruikers invoer naar toe moeten sturen en krijgen daarom ook *simulation* mee.

Elke view is een subclasse van *AbstractView*, in deze klasse staat een constructor die er voor zorgt dat elke view aangemeld word bij het model. Het fragment code hieronder laat deze constructor zien.

public AbstractView(AbstractModel simulation) {

this.simulation = simulation;

simulation.addView(this);

}

Tekstfragment 2 Codefragment uit AbstractView.java

Elk model is een subclasse van *AbstractModel* en hierin staan onder andere de methoden voor het toevoegen en notificeren van views. Het fragment code hieronder laat deze methodes zien.

public void addView(AbstractView view) {

views.add(view);

}

public void notifyViews() {

for(AbstractView v: views) v.updateView();

}

Tekstfragment 3 Codefragment uit AbstractModel.java

Op deze manier wordt in Vossen & Konijnen het MVC-model gebruikt.

1. Uitbreiding 2
2. Bonus uitbreidingen
3. Conclusie
4. Advies / aanbevelingen / actieplan > uit elkaar halen als je ze alledrie benoemt. Vaak: Conclusie en aanbevelingen. Daarna volg Advies.
5. Noten
6. Literatuurlijst

Programmeren in Java met BlueJ – editie: 5 - Auteurs: David J.Barnes & Michael Kölling > check de Schrijfwijzer TEchniek voor de notatie van de bronnen.

Bijlage 1

Bijlage 2

Register