Gebruikersinterfaces

Introductie *2*

Leerkern *3*

1 Swing *3*
2 WindowBuilder in Eclipse *6*
  2.1 Constructie van een eenvoudige GUI *6*
  2.2 Het gebruik van packages *11*
3 De code van visuele klassen *12*
  3.1 De structuur van de gegenereerde code *13*
  3.2 Attributen *14*
  3.3 Methoden aanroepen op de aanroeper zelf *15*
  3.4 Private methoden definiëren *16*
4 Interactie toevoegen *17*
  4.1 Een event handler *17*
  4.2 Stemmen op een partij *20*
  4.3 Stemmen op een kandidaat *22*
5 Nabeschouwing *24*
  5.1 Volgorde *24*
  5.2 Het openen van een GUI-klasse *25*

Samenvatting *26*

Zelftoets *27*

Terugkoppeling *28*

1 Uitwerking van de opgaven *28*
2 Uitwerking van de zelftoets *32*

Bijlage: Een deel van de interface van vijf standaardklassen *34*

Leereenheid 4

Gebruikersinterfaces

INTRODUCTIE

We hebben in de leereenheden 1 en 3 de eerste objectgeoriënteerde programma's geschreven. Deze programma's laten de computer door ons voorgeschreven handelingen verrichten. De uitvoer werd zichtbaar in een DOS-venster of (later) in het uitvoervenster (console) van Eclipse en oogde nogal beperkt. Invoer was helemaal niet mogelijk. Dat is wel heel anders dan u gewend bent: normaal communiceert u met een programma via een grafische gebruikersinterface, met menu's, lijsten, knoppen, tekstvelden enzovoort. Figuur 4.1 toont een voorbeeld van een dergelijke interface: een stemmachine met drie knoppen om een stem uit te brengen op (de lijsttrekkers van) CDA, PvdA en VVD, een knop om de uitslag op te vragen, en een tekstgebied om die uitslag te laten zien.

FIGUUR 4.1 Een grafische gebruikersinterface

Java biedt via de klassenbibliotheek uitgebreide ondersteuning voor het maken van dergelijke gebruikersinterfaces. Voor Eclipse is bovendien een visuele editor (WindowBuilder) beschikbaar, die het mogelijk maakt om een gebruikersinterface als het ware op het scherm te tekenen. De programmeur hoeft dan maar een beperkt deel van het programmeerwerk zelf te doen. Hoewel een dergelijke benadering ook zijn beperkingen kent, maken we er in de rest van deze cursus dankbaar gebruik van.

Paragraaf 1 van deze leereenheid bevat een algemene introductie over Swing, het deel van de Java-bibliotheek dat het maken van gebruikers-interfaces ondersteunt. In paragraaf 2 leert u om met behulp van WindowBuilder gebruikersinterfaces te maken, die echter nog niet reageren op acties van de gebruiker. In paragraaf 3 kijken we, voor zover nodig, naar de code die WindowBuilder genereert. Naar aanleiding daarvan behandelen we enkele nieuwe Java-concepten, die in de rest

van de leereenheid gebruikt worden. In paragraaf 4 leert u hoe u de gebruikersinterfaces wel kunt laten reageren op acties van de gebruiker. We bouwen in die paragraaf twee stemmachines – die uit figuur 4.1 en een verbeterde versie waarmee niet alleen op partijen maar ook op kandidaten gestemd kan worden. Paragraaf 5 bevat enkele kanttekeningen bij het voorafgaande.

LEERDOELEN

Na het bestuderen van deze leereenheid wordt verwacht dat u

- weet hoe een Swing-GUI is opgebouwd
- een hoofdscherm (JFrame) kunt toevoegen aan een Eclipse-project
- weet waar u in WindowBuilder het ontwerpscherm, de componentenlijst, de eigenschappenlijst en het palette kunt vinden
- met behulp van WindowBuilder componenten aan het ontwerpscherm toe kunt voegen en deze met behulp van de eigenschappenlijst de gewenste eigenschappen kunt geven
- de globale structuur van de gegenereerde code begrijpt
- de overeenkomst en het verschil weet tussen een attribuut en een lokale variabele
- weet waar een private methode toe dient en hoe deze aangeroepen kan worden
- via WindowBuilder event handling kunt toevoegen aan grafische componenten
- aan de gegenereerde code attributen en private methoden kunt toevoegen
- de toegevoegde private methoden voor event handling en initialisatie op de juiste manier kunt aanroepen
- in grote lijnen begrijpt hoe een programma met een GUI verwerkt wordt
- de betekenis kunt geven van de volgende kernbegrippen: GUI, Swing, contentPane, component, domeinlaag, interfacelaag, refactoring, event, event handling, this.

Studeeraanwijzingen

– Deze leereenheid bevat een aantal schermafdrukken. Deze kunnen afwijken van de schermen die u krijgt (afhankelijk van versies van Eclipse en het besturingssysteem), maar zijn functioneel wel gelijkwaardig.

De studielast van deze leereenheid bedraagt circa 8 uur.

LEERKERN

1 Swing

GUI

Grafische gebruikersinterfaces ofwel *GUI's* (Graphical User Interfaces) vormen een ideale illustratie van het objectgeoriënteerde paradigma. Zulke interfaces zijn immers opgebouwd uit een beperkt aantal componenten (knoppen, invoervelden, keuzelijsten, …) in steeds wisselende configuraties. Elke component is een object; overeenkomstige objecten zoals knoppen behoren allemaal tot één klasse. Voor de meeste objectgeoriënteerde talen bestaat dan ook een bibliotheek van klassen waarmee de programmeur GUI's kan bouwen.

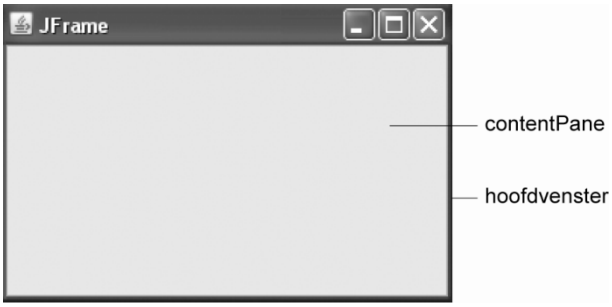
Swing
java.awt
javax.swing

De bibliotheek die Java hiervoor beschikbaar stelt wordt aangeduid als de Swing-bibliotheek, of kortweg *Swing*. De klassen uit deze bibliotheek zijn verdeeld over twee packages, namelijk *java.awt* en *javax.swing*. Die verdeling heeft historische redenen. De eerste grafische bibliotheek van Java heette de Abstract Windowing Toolkit, en bestond alleen uit de klassen in de package java.awt. Deze package had echter beperkingen, die er uiteindelijk toe leidden dat er een nieuwe grafische package bij kwam: javax.swing. De namen van de klassen uit deze package beginnen vaak met een hoofdletter J. Deze package bevat bijvoorbeeld de klasse JButton voor knoppen, JFrame voor vensters en JTextArea voor tekstgebieden.

JFrame

contentPane

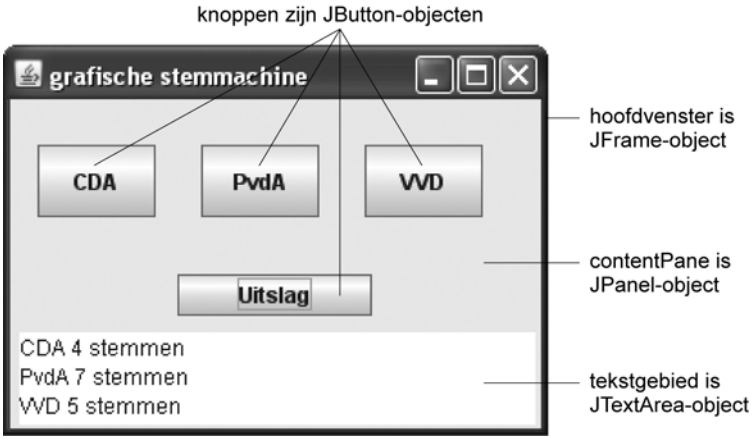
De basis van iedere Swing-GUI is een hoofdvenster. Er zijn verschillende typen hoofdvensters, maar wij gebruiken daarvoor steeds een *JFrame-object*. Het hoofdvenster bevat nooit meer dan twee elementen: een menubalk met menuopties en een zogeheten *contentPane* (dat is een JPanel-object) waarin alle andere grafische elementen zoals knoppen en tekstgebieden worden geplaatst. De menubalk is optioneel en gebruiken we niet in deze cursus. Ons uitgangspunt is daardoor steeds een GUI als getoond in figuur 4.2. Het hoofdvenster beslaat de hele figuur, inclusief de balk bovenin met de titel JFrame en de knopjes om te minimaliseren, maximaliseren en sluiten. De contentPane is het grijze vlak binnen het hoofdvenster. De elementen die we zelf toevoegen, komen daarin te staan.



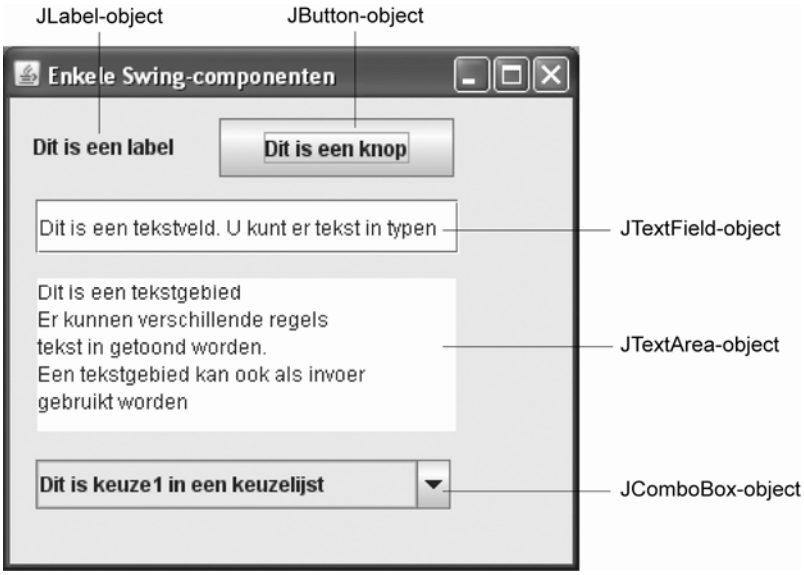
FIGUUR 4.2 Hoofdvenster en contentPane

Ter illustratie tonen we in figuur 4.3 nogmaals de gebruikersinterface uit de introductie, maar nu hebben we ook aangegeven uit welke objecten van welke klassen deze is opgebouwd. De vier knoppen en het tekstgebied zijn in de contentPane geplaatst. De knopjes in de titelbalk maken standaard deel uit van elk JFrame-object; die hoeven we dus niet zelf toe te voegen.

Wij gebruiken in deze leereenheid slechts enkele soorten Swing-componenten: labels, knoppen, tekstvelden, tekstgebieden en keuzelijsten, overeenkomend met de klassen JLabel, JButton, JTextField, JTextArea en JComboBox (zie figuur 4.4).



FIGUUR 4.3 Uit deze objecten is de GUI uit de introductie opgebouwd



FIGUUR 4.4 Enkele Swing-componenten

We laten u zelf het gedrag van deze componenten bekijken.

– Start zonodig Eclipse en open het project Le04Componenten.
– Ga binnen dit project naar de klasse VoorbeeldFrame, en verwerk deze. U krijgt dan een scherm te zien als figuur 4.4.
– Probeer de tekst van de label te veranderen; u zult merken dat dit niet lukt.
– Klik op de knop. Dat is mogelijk, maar er gebeurt niets.
– Ga na dat u de teksten in het tekstveld en het tekstgebied wel kunt wijzigen.
– Bekijk ook de andere keuzes in de keuzelijst.
– Sluit het venster op de gebruikelijke manier.

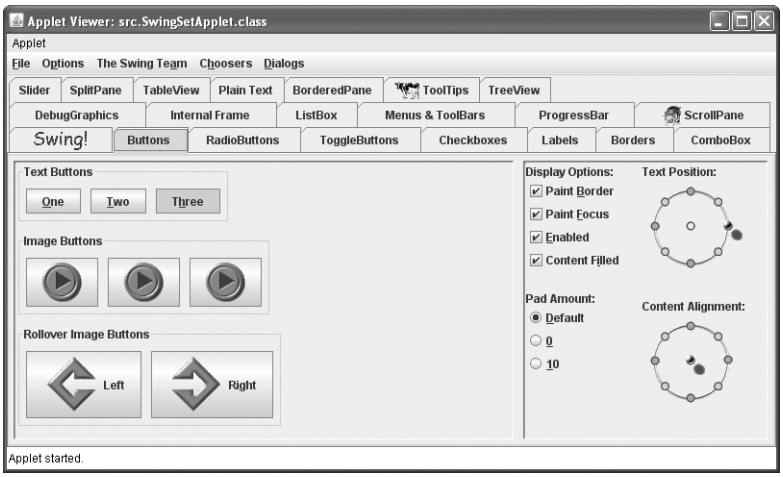
U hoeft niet naar de Java-code van dit project te kijken; het gaat alleen om de componenten in de GUI.

Wij gebruiken daarmee slechts een zeer klein gedeelte van de mogelijkheden die de Swing-package biedt. Met behulp van een demo geven we u daar een indruk van. Deze demo is een applet, een programma dat geschreven is om opgenomen te worden in een webpagina.

– Open het project Le04SwingDemo.
– Rechtsklik in het projectvenster op de projectnaam Le04SwingDemo, selecteer Run As en vervolgens Java Applet.

U krijgt een scherm te zien met een groot aantal tabbladen. Elk tabblad gaat over een Swing-component.

– Kies als eerste het tabblad Buttons (zie figuur 4.5).
– U ziet voorbeelden van knoppen met tekst, plaatjes en met beide. U kunt het uiterlijk beïnvloeden door aan de rechterkant verschillende mogelijkheden te kiezen.
– Bekijk ook de tabbladen Labels, Plain Text (bevat zowel tekstvelden als een tekstgebied) en ComboBox.
– Bekijk als u wilt ook de andere tabbladen en probeer de menu-opties uit. U krijgt daarmee een (overigens nog steeds onvolledig) beeld van wat Swing te bieden heeft.



FIGUUR 4.5 Verschillende soorten Swing-knoppen

2 WindowBuilder in Eclipse

In deze paragraaf maken we een programma met een uiterst eenvoudige grafische gebruikersinterface, die niets anders doet dan de tekst “Hallo daar!” tonen. Dit programma vormt het grafische equivalent van het programma van zes regels uit opgave 1.8 in leereenheid 1.

2.1 CONSTRUCTIE VAN EEN EENVOUDIGE GUI

– Start Eclipse.
– Maak een nieuw project met de naam Le04Basis.

Frameklasse maken

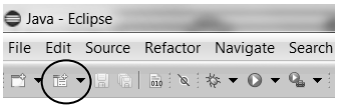
Binnen dit project gaan we een GUI aanmaken. De basis daarvoor is een JFrame. We kunnen deze op twee manieren maken:

Methode 1

– Selecteer het nieuwe project in het projectvenster, klik op de rechtermuisknop, kies New en vervolgens Other.
–Open in het venster dat u te zien krijgt WindowBuilder, en vervolgens Swing Designer. Kies JFrame en klik op Next.

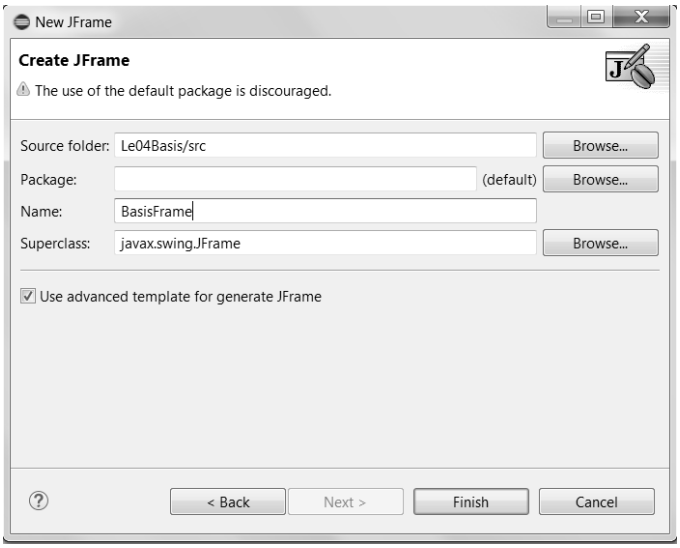
Methode 2

– Selecteer het nieuwe project in het projectvenster, en klik in de knoppenbalk op de knop Create new visual classes (zie figuur 4.6).
– Kies vervolgens Swing – JFrame.



FIGUUR 4.6 De knop Create new visual classes in de knoppenbalk

U krijgt vervolgens het scherm te zien van figuur 4.7.
– Vul als naam voor de klasse in: BasisFrame (let op de beginletter B; klassennamen beginnen met een hoofdletter). Eclipse geeft nu een waarschuwing: 'The use of the default package is discouraged'. Deze negeren we voorlopig.
– Klik op Finish.



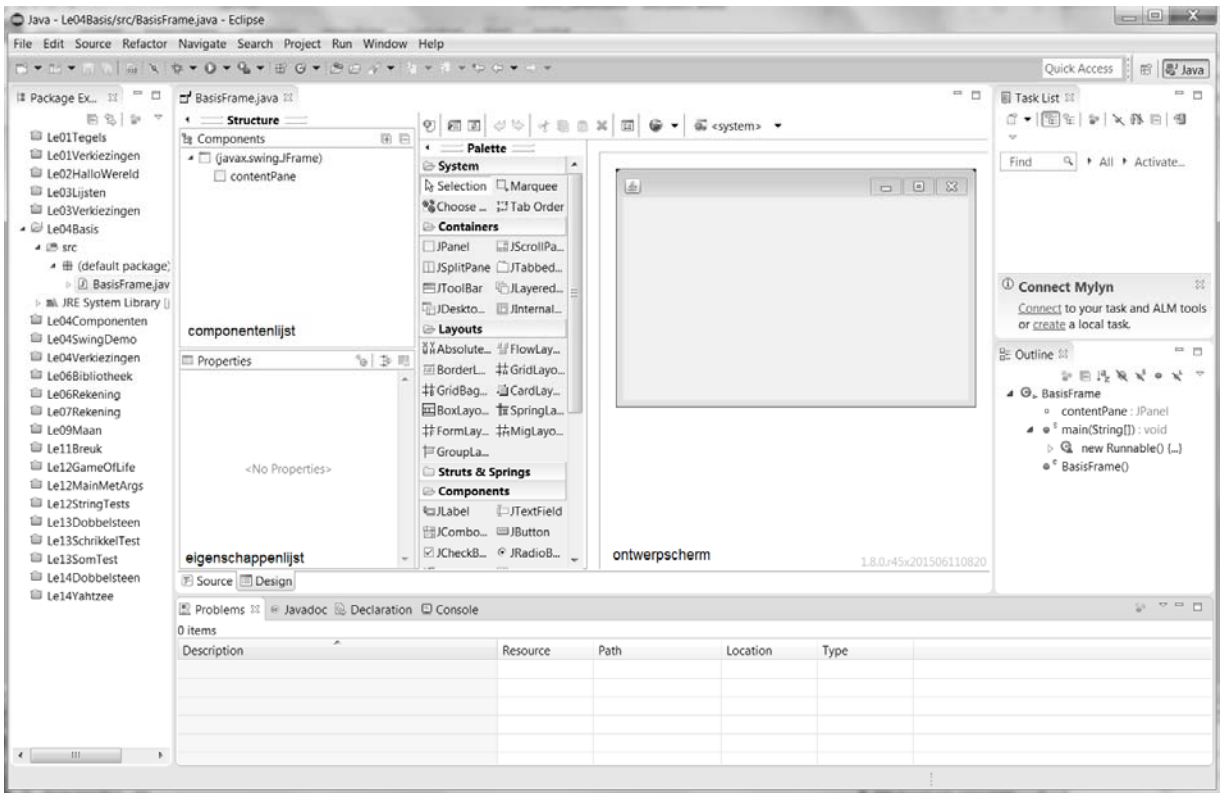
FIGUUR 4.7 Het scherm New JFrame

Ontwerpscherm

Het Eclipse-scherm bevat nu op de plaats van het editorvenster twee tabbladen, Design en Source (zie figuur 4.8). Het tabblad Design bevat een grafische weergave van de GUI die we aan het ontwerpen zijn. Dit tabblad noemen we het *ontwerpscherm*. Het tabblad Source bevat de bijbehorende programmacode, maar daar kijken we nog even niet naar.

Componentenlijst

De elementen van de GUI kunt u zien in het venster Components (de *componentenlijst*): er is een JFrame (aangeduid met javax.swing.JFrame), met daarin een contentPane.



FIGUUR 4.8 Eclipse na creatie van een JFrame

– Selecteer in de componentenlijst nu eerst de eerste regel (javax.swing.JFrame). Kijk wat er gebeurt in het ontwerpscherm en in de eigenschappenlijst. Selecteer dan de tweede regel (contentPane) en kijk weer wat er gebeurt. Ga desnoods een paar keer heen en weer. U zult zien dat in het ontwerpscherm steeds het overeenkomstige element geselecteerd is: hetzij het volledige venster, hetzij alleen de binnenkant. Ook ziet u dat de eigenschappen van het geselecteerde element worden weergegeven bij Properties (we noemen dit de *eigenschappenlijst*).

Eigenschappenlijst

OPDRACHT 4.1
Wijzig in de eigenschappenlijst de titel van het JFrame in BasisFrame.
Waar wordt deze wijziging nog meer zichtbaar?

defaultClose-Operation

Een belangrijke eigenschap van het JFrame is de defaultCloseOperation. Deze bepaalt wat er met de applicatie gebeurt wanneer het frame wordt gesloten. Om er voor te zorgen dat de applicatie wordt gestopt wanneer het frame wordt gesloten, dient de waarde van defaultCloseOperation gelijk te zijn aan EXIT. Dit is de standaardwaarde in WindowBuilder, en dit hoeft u normaal gesproken niet zelf meer in te stellen.

Belangrijke stap!

De volgende stap moet u altijd (bij voorkeur als eerste) doen als u een GUI maakt. Vergeet u deze, dan kunt u straks de grafische elementen geen zelfgekozen plek op het scherm geven.

Layout contentPane absolute (= null) maken

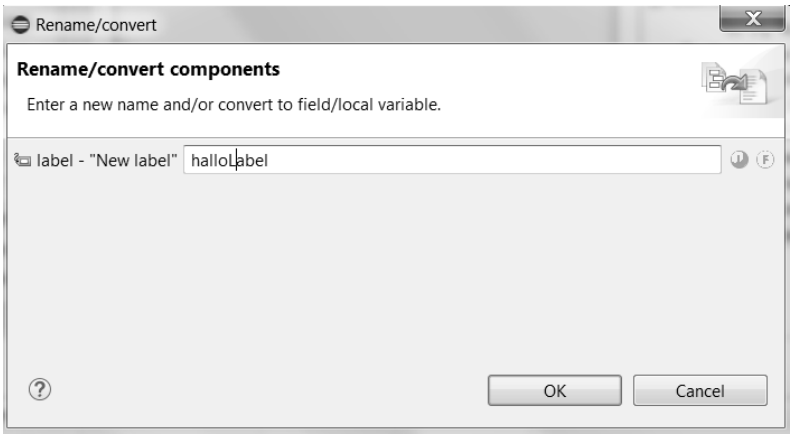
–Klik met de rechtermuisknop op het contentPane en kies vervolgens voor Absolute layout.

In de volgende stap voegen we een component toe aan het contentPane en wel een label voor een stukje uitvoertekst, waar de gebruiker niets aan kan veranderen.

Component op ontwerpscherm plaatsen

– Klik in het palette op JLabel. Klik vervolgens in de contentPane op de plek waar u de component wilt plaatsen en maak een rechthoek van de gewenste afmetingen.

WindowBuilder maakt nu een component (als attribuut). De nieuwe component is te zien in de componentenlijst. Het is voor de leesbaarheid van het programma van belang dat de componenten een herkenbare naam krijgen.
– Geef de component een herkenbare naam, bijvoorbeeld halloLabel. Rechtsklik op de component en selecteer Rename. Let op de beginletter van de naam: gebruik een kleine letter (zie figuur 4.9).



FIGUUR 4.9 Geef het nieuwe label de naam halloLabel

– Wijzig via de eigenschappenlijst de eigenschap text van dit label in "Hallo daar!" (selecteer eventueel het label als dat nog niet is gebeurd). Pas als dat nodig is, de afmetingen aan; u kunt het label op het ontwerpscherm groter of kleiner maken en het ook verplaatsen.

Gegenereerde code

We zijn nu klaar met deze eerste GUI. We hebben nog geen regel code gezien: het schrijven van de code voor de klasse BasisFrame hebben we overgelaten aan WindowBuilder. We spreken in dit verband van *gegenereerde code*. Realiseer u daarbij wel dat het niet per se nodig is om WindowBuilder te gebruiken; met voldoende kennis van Java en Swing kunt u een dergelijke klasse ook helemaal zelf schrijven. Het gebruik van WindowBuilder bespaart echter wel tijd. Bovendien kunnen we daardoor ook zonder veel kennis van Java al grafische gebruikersinterfaces maken.

De gegenereerde klasse is zelfs een volledige applicatie. WindowBuilder heeft namelijk ook een main-methode gegenereerd.

– Verwerk tot slot dit programma. U ziet uw eerste GUI verschijnen (zie figuur 4.10).

– Stel vast dat het hier een normaal venster betreft, dat groter en kleiner gemaakt kan worden met de muis en met de knopjes bovenin geminimaliseerd, gemaximaliseerd en tot slot gesloten kan worden.

FIGUUR 4.10 Het BasisFrame getoond

In de volgende opdracht laten we u zelf een tweede en wat complexere GUI maken, namelijk de stemmachine uit figuur 4.1 en 4.3. We doen daarbij niet meer dan wat we nu kunnen en dat is de GUI 'tekenen'. Pas in paragraaf 4 zorgen we ervoor dat de knoppen ook werken.

OPDRACHT 4.2

a Start een nieuw project met de naam Le04Verkiezingen.
b Maak binnen dit project een nieuw JFrame StemOpPartijFrame.
c Geef de eigenschap layout van de contentPane de waarde Absolute layout.
d Geef de eigenschap title van het hoofdvenster de waarde “grafische stemmachine”.
e Voeg vier knoppen (component JButton) en een tekstgebied (component JTextArea) toe, als getoond in figuur 4.3. Geef de componenten herkenbare namen. U kunt eventueel de afmeting van het frame aanpassen door dit met de muis te vergroten.
f Geef de knoppen het juiste opschrift. Gebruik hiervoor de eigenschap text.
g Verwerk StemOpPartijFrame en controleer zo of de GUI zichtbaar wordt.

Het is mooi als de drie knoppen voor de partijen even groot zijn en op één lijn staan. WindowBuilder kan u daar bij helpen.

Lay-out van groep componenten op elkaar afstemmen

– Selecteer alle drie de knoppen (door de shifttoets ingedrukt te houden terwijl u de knoppen met de muis selecteert). Boven het ontwerpscherm verschijnen een aantal knoppen als getoond in figuur 4.11.
De betekenis van de icoontjes wijst zich (eventueel met behulp van de tooltip text) vanzelf.

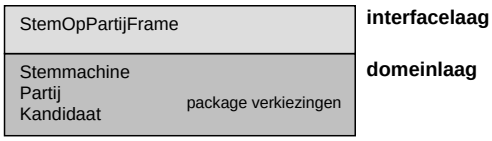
FIGUUR 4.11 Knoppen om de lay-out van een groep componenten op elkaar af te stemmen

2.2 HET GEBRUIK VAN PACKAGES

Bij het aanmaken van een nieuwe klasse kregen we van Eclipse een waarschuwing 'The use of the default package is discouraged', die we hebben genegeerd. Eclipse wil dus kennelijk dat we onze klassen in packages plaatsen. Aan de hand van het verkiezingenvoorbeeld leggen we uit waarom dat is.

In paragraaf 4 verbinden we de grafische gebruikersinterface uit opdracht 4.2 met de package verkiezingen, zodat we een applicatie krijgen waarmee we via die GUI ook echt stemmen kunnen uitbrengen (op de lijsttrekkers van de partijen). Die applicatie omvat (naast de klassen uit de Java-bibliotheek) in totaal vier klassen, namelijk Stemmachine, Partij, Kandidaat, en StemOpPartijFrame. Deze klassen vallen uiteen in twee groepen van duidelijk verschillende aard.
– Stemmachines, partijen en kandidaten zijn zaken die we aantreffen in de werkelijkheid. De klassen uit de package verkiezingen vormen daar een model van (een zeer beperkt model, maar toch een model).
– De klasse StemOpPartijFrame heeft een totaal andere functie. Deze verwijst niet naar zaken in de werkelijkheid, maar dient uitsluitend om ervoor te zorgen dat gebruikers met de klassen uit de package verkiezingen kunnen communiceren.

De uiteindelijke applicatie heeft dus een duidelijke structuur, die we kunnen weergeven als getoond in figuur 4.12. Beide lagen gebruiken naast de vier genoemde klassen bovendien klassen uit de Java-bibliotheek.



FIGUUR 4.12 Structuur van een applicatie met een GUI

Domeinlaag

Interfacelaag

Het vaststellen van de structuur van een applicatie is een belangrijke stap in het ontwerp ervan en hoe groter de applicatie is, des te belangrijker dat wordt. Bijna alle applicaties uit deze cursus hebben een structuur vergelijkbaar met die uit figuur 4.12. Er is een *domeinlaag*, met klassen die het deel van de werkelijkheid modelleren waar het programma over gaat. In dit geval hebben wij de klassen uit deze domeinlaag voor u geprogrammeerd, maar in het tweede blok gaat u dat zelf doen. En er is een *interfacelaag* met klassen die de communicatie met de gebruiker verzorgen.

Het is een goed gebruik om de structuur van een applicatie in het programma zichtbaar te maken door het gebruik van packages. Met andere woorden: niet alleen de klassen Stemmachine, Partij en Kandidaat horen samen in een package te zitten, maar ook de klasse StemOpPartijFrame hoort in een package.

We hebben dat laatste tot nu toe verzuimd. Dat was vooral omdat we niet alles tegelijk willen uitleggen, maar het biedt ook een goede

gelegenheid om u kennis te laten maken met de ondersteuning van Eclipse bij de herstructurering van een bestaand programma.

Nieuwe package maken

– Rechtsklik in het projectvenster op het project Le04Verkiezingen, kies New en dan package. Vul als naam in verkiezingengui (let op de kleine letters) en klik op Finish. U ziet de nieuwe package in het projectvenster.

Verplaatsen klasse

– Rechtsklik nu, eveneens in het projectvenster, op de klasse StemOpPartijFrame en kies uit het menu voor Refactor en vervolgens voor Move… U krijgt dan een scherm te zien met alle packages van projecten die open zijn. Kies de package verkiezingengui en klik op OK. De klasse StemOpPartijFrame wordt daarmee verplaatst naar deze package.

Packagedeclaratie

– Bekijk de code van de klasse StemOpPartijFrame. U ziet dat Eclipse ervoor heeft gezorgd dat de eerste regel nu een packagedeclaratie bevat:

```
package verkiezingengui;
```

Alle klassen die tot een specifieke package behoren beginnen met een dergelijke packagedeclaratie. We typen die nooit zelf in, maar laten Eclipse die genereren.

Refactoring

Het verplaatsen van een klasse, waarbij de code wordt aangepast, is slechts een voorbeeld van een mogelijke herstructurering (Engels *refactoring*). Eclipse biedt daarvoor veel ondersteuning. Refactoring kan bijvoorbeeld ook worden gebruikt om namen te veranderen (van een project of een package, maar ook van een attribuut of een methode). Eclipse zorgt er dan voor dat de naam overal in de code wordt aangepast. We gaan hier niet in detail op in, maar het is goed dat u weet dat dat kan. U kunt er dan zelf mee experimenteren.

In figuur 4.7, in het scherm New JFrame, kunt u ook direct de naam van de package opgeven waarin de nieuwe GUI-klasse geplaats wordt. Wanneer de package nog niet bestaat, dan wordt deze automatisch gecreëerd.

Samenvatting

We vatten de gang van zaken bij het maken van een GUI nog eens samen.
– Maak indien nodig een nieuw project.
– Voeg aan het project een JFrame toe. Plaats deze bij voorkeur in een package. Wij houden de conventie aan dat de naam van de nieuwe klasse eindigt op Frame en die van de package op gui.
– Selecteer in het ontwerpscherm de contentPane (klik binnen in het venster) en geef in de eigenschappenlijst de eigenschap layout de waarde Absolute layout.
– Voeg met behulp van de componentenpalette componenten toe aan het scherm. Geef iedere component een betekenisvolle naam. Gebruik de eigenschappenlijst om de component de gewenste eigenschappen te geven.

3 De code van visuele klassen

Om echte interactieve GUI's te kunnen maken, moeten we eigen code toevoegen aan de code die door WindowBuilder is gegenereerd. We

kijken daarom nu eerst naar die gegenereerde code. Aan de hand daarvan behandelen we ook enkele nieuwe Java-concepten.

3.1 DE STRUCTUUR VAN DE GEGENEREERDE CODE

In deze deelparagraaf kijken we heel globaal naar de code van de klassen BasisFrame en StemOpPartijFrame.

– Kies van de klasse BasisFrame nu in plaats van het tabblad Design het tabblad Source. U gaat daarmee van het ontwerpscherm naar het editorvenster. Figuur 4.13 toont een deel van de code.

```
1  import java.awt.BorderLayout;
8
9  public class BasisFrame extends JFrame {
10
11   private JPanel contentPane;
12   private JLabel halloLabel;
13
14   /**
15    * Launch the application.
16    */
17   public static void main(String[] args) {
18     EventQueue.invokeLater(new Runnable() {
19       public void run() {
20         try {
21           BasisFrame frame = new BasisFrame();
22           frame.setVisible(true);
23         }
24         catch (Exception e) {
25           e.printStackTrace();
26         }
27       }
28     });
29   }
30
31   /**
32    * Create the frame.
33    */
34   public BasisFrame() {
35     initialize();
36   }
```

FIGUUR 4.13 Een deel van de gegenereerde code

Code inklappen en uitklappen

We bekijken hier alleen de globale structuur van de code. Eclipse maakt dat makkelijk door ons de gelegenheid te geven om delen van de code te verbergen. Er staan daartoe op bepaalde plekken minnetjes in de kantlijn. Klikt u op zo'n minnetje, dan wordt het codedeel daarnaast ingeklapt tot alleen de essentie over is.
– Klik op alle minnetjes naast de code. In figuur 4.14 ziet u wat u overhoudt. De minnetjes zijn veranderd in plusjes. Ook aan de regelnummers kunt u zien waar code is ingeklapt.

De eerste regels van de code bevatten importopdrachten. Deze regels waren door Eclipse al ingeklapt. Als u wilt, kunt u ze even uitklappen. U ziet dan dat er klassen uit de packages java.awt en javax.swing zijn geïmporteerd. Eclipse geeft waarschuwingen (gele onderstrepingen) dat niet alle geïmporteerde klassen worden gebruikt. Deze waarschuwingen kunnen we negeren.

```
1  import java.awt.BorderLayout;
8
9  public class BasisFrame extends JFrame {
10
11   private JPanel contentPane;
12   private JLabel halloLabel;
13
15   * Launch the application.
17   public static void main(String[] args) {
30
32   * Create the frame.
34   public BasisFrame() {
37   private void initialize() {
47   private JLabel getHalloLabel() {
54 }
```

FIGUUR 4.14 Structuur van de gegenereerde code

Regel 9 laat zien dat onze ontwerpactiviteiten inderdaad hebben geresulteerd in de definitie van een klasse BasisFrame, die een uitbreiding is van de klasse JFrame (wat dat precies inhoudt, leggen we uit in leereenheid 5).

Zoals we al weten uit leereenheid 1, heeft een klasse attributen, constructors en methoden. Die structuur is terug te zien in figuur 4.14.

De regels 11 en 12 bevatten declaraties voor de attributen van de klasse BasisFrame. Zoals u kunt zien lijken deze declaraties sterk op variabelendeclaraties, alleen het sleutelwoord private is extra. We bekijken deze attribuutdeclaraties nader in paragraaf 3.2.

Op regel 17 ziet u dat er ook een methode main is gegenereerd. Daarom kunnen we deze klasse als applicatie uitvoeren. In de methode main wordt het frame gecreëerd en zichtbaar gemaakt. De gegenereerde main van een frame zal er altijd exact zo uitzien. Wijzig deze methode niet.

Op de regels 34, 37 en 47 ziet u de signaturen van de constructor en van de twee methoden (initialize en getHalloLabel) van de klasse BasisFrame. De signatuur van de constructor heeft de vorm die u al kent, maar de signatuur van de methoden bevat een nieuw element: in plaats van public, zoals u gewend bent in de signatuur van een methode, staat ook hier het sleutelwoord private. In paragraaf 3.3 bekijken we wat dat betekent en hoe we daar gebruik van zullen maken.

OPDRACHT 4.3
Bekijk nu zelf de structuur van de code van de klasse StemOpPartijFrame. Welke attributen en welke methoden heeft deze klasse?

3.2 ATTRIBUTEN

Lokale variabelen

In de vorige leereenheden hebben we variabelen gedeclareerd binnen de methode main en soms zelfs binnen een herhaling of keuzeopdracht. Deze variabelen hebben alleen betekenis binnen de methode of het blok waarbinnen ze gedeclareerd zijn: het zijn l*okale variabelen*.

Attributen zijn feitelijk ook variabelen. Hoewel een attribuut een geheel andere rol speelt dan een lokale variabele, duiden beide een in principe veranderlijke ('variabele') waarde aan die ergens in het computergeheugen moet worden opgeslagen. Deze waarde wordt in beide gevallen beheerd door het object. Het verschil ligt in de levensduur van de opgeslagen waarde en – als gevolg daarvan – in de zichtbaarheid ervan. De waarde van een lokale variabele gaat verloren zodra de methode of het blok waarbinnen deze voorkomt, verwerkt is. Daarom kan de lokale variabele alleen worden gebruikt binnen de methode of het blok waarin zij is gedeclareerd. De waarde van het attribuut heeft andere eigenschappen: zij blijft bewaard zolang het object bestaat. Een attribuut kan daarom gebruikt worden binnen alle methoden van de klasse die dat attribuut bevat.
De attribuutdeclaraties in deze leereenheid hebben de volgende vorm:

Attribuut-declaratie

```
private type variabelennaam = expressie;
```

Private

Deze vorm lijkt veel op die van een variabelendeclaratie (met toekenning van een beginwaarde), met één verschil: het woord *private* aan het begin. Bij attributen kan namelijk aangegeven worden of ze ook buiten de klasse toegankelijk zijn. Het woord private geeft aan dat dit niet het geval is.

Voorbeeld

Bekijk als voorbeeld de declaraties van contentPane en halloLabel van de gegenereerde code.

```
private JPanel contentPane;
private JLabel halloLabel;
```

Zoals u ziet, zijn dit precies de componenten die zijn opgenomen in het BasisFrame. Deze verschijnen dus in de code als attributen van deze klasse. Het BasisFrame-object beheert daardoor een JPanel-object met de naam contentPane en een JLabel-object met de naam halloLabel. Alle methoden van de klasse BasisFrame hebben toegang tot deze attributen. Vanwege de toegangsspecificatie private hebben methoden van andere klassen er echter geen toegang toe.

3.3 METHODEN AANROEPEN OP DE AANROEPER ZELF

Tot nu toe hebben we methoden steeds aangeroepen op een object. Dat is de kern van objectoriëntatie: er is een taakverdeling tussen objecten, waarbij het ene object een verzoek kan richten aan een ander object om een methode uit te voeren.

Het kan echter ook zinvol zijn om een object een verzoek te laten richten aan zichzelf om een methode uit te voeren. Dat object delegeert dan een deeltaak aan een andere methode van zichzelf. Het object dat de methode aanroept is dan dus hetzelfde als het object dat de methode uitvoert.

Om te bekijken hoe dat in zijn werk gaat, kijken we iets dieper in de code van de constructor van de klasse BasisFrame.

Net als we al gewend zijn van de methode main, bevat de constructor een reeks opdrachten (in dit geval echter maar één) die een voor een worden uitgevoerd. Die ene opdracht is:

```
initialize();
```

Deze aanroep noemt geen object. Dat betekent dat het object zelf (het object dat net gecreëerd is) nu zijn eigen methode initialize gaat verwerken.

Er is voor een dergelijke aanroep ook een tweede schrijfwijze:

```
this.initialize();
```

this

In deze vorm is de objectaanduiding niet weggelaten, maar bestaat uit het sleutelwoord *this*.

Het aanroepen van een eigen methode draagt bij aan een goede structurering van de code. Nette, onderhoudbare code bestaat uit kleine, overzichtelijke methoden. Wordt een methode te omvangrijk, dan kan het helpen om deeltaken onder te brengen in aparte methoden. Omdat zulke methoden alleen van belang zijn binnen de klasse, krijgen ze de aanduiding private. Het toevoegen van een private methode aan een klasse wijzigt niets aan de interface van de klasse (dat wil zeggen, aan het aanbod van public methoden).

Zowel de methode initialize als de methode getHalloLabel zijn op deze manier gedefinieerd. De signaturen van deze methoden zijn dus

```
private void initialize()
private JLabel getHalloLabel()
```

We voegen in de volgende paragraaf onze eigen code toe aan de gegenereerde code. Om de scheiding tussen onze eigen code en de gegenereerde code goed te bewaren, brengen we de eigen code steeds onder in nieuwe private methoden. Aan de methoden die gegenereerd zijn door WindowBuilder hoeven we dan alleen een aanroep van de betreffende nieuwe private methode toe te voegen.

3.4 PRIVATE METHODEN DEFINIËREN

De methoden die we schrijven hebben in deze leereenheid steeds de volgende vorm:

Private methode

```
private void methodenaam() {
  opdrachten
}
```

De opdrachten in zo'n methode wijken in principe niet af van de opdrachten uit de programma's die we tot nu toe schreven. Ook hier kunnen declaraties, toekenningen, methodeaanroepen, herhalingen en keuzeopdrachten voorkomen. Er is echter één aspect dat we nog niet eerder gezien hebben omdat we nog niet met attribuutdeclaraties te maken hebben gehad: methoden hebben toegang tot de attributen van de klasse waar ze in voorkomen. U kunt dus bijvoorbeeld een methode

toevoegen aan de klasse BasisFrame die de tekst van het attribuut halloLabel wijzigt door daarop een methode aan te roepen.

Methodeaanroepen op grafische objecten (objecten die we via WindowBuilder hebben toegevoegd) zullen regelmatig voorkomen in de private methoden die we schrijven. In de bijlage bij deze leereenheid is een klein deel van de interfaces van enkele relevante klassen opgenomen: JLabel, JTextArea, JTextField, JButton en JComboBox.

Voorbeeld

Stel we hebben een visuele klasse TestFrame gemaakt en daaraan een JLabel toegevoegd met de naam label1. We definiëren een private methode die de tekst van deze label wijzigt in "Dit is label 1". Er wordt gebruikgemaakt van de methode setText van de klasse JLabel (zie paragraaf 1 van de bijlage).

```
private void test1(){
  label1.setText("Dit is label1");
}
```

OPGAVE 4.4

Stel aan de klasse TestFrame zijn met behulp van WindowBuilder behalve label1 ook nog de volgende componenten toegevoegd:
– een tekstveld met de naam tekstVeld1
– een tekstgebied met de naam tekstGebied1
– een keuzelijst met de naam keuzeLijst1.
Schrijf met behulp van de informatie in de bijlage de volgende twee private methoden:
a een methode test2 die de strings in label1 en in tekstVeld1 toevoegt aan tekstGebied1.
b een methode vulKeuzeLijst die aan keuzeLijst1 de items "keuze1", "keuze2" en "keuze3" toevoegt.

4 Interactie toevoegen

4.1 EEN EVENT HANDLER

Veel communicatie met programmatuur verloopt via GUI's. Dat gaat als volgt. De GUI staat op het scherm en zolang wij niets doen, doet het programma ook niets. Tot wij bijvoorbeeld op een knop klikken, een tekstje invoeren en op enter drukken of een mogelijkheid kiezen uit een menu of een uitrollijst. Vanuit het programma bezien heet zo'n gebruikersactie een *event*. Het programma reageert daar dan op door iets te doen (hopelijk hetgeen wij wilden): het *handelt de event af*.

Event
Event afhandelen

Wij laten events afhandelen door private methoden, die we vanuit de gegenereerde code aanroepen. De code van deze methoden kan uiteraard niet automatisch worden gegenereerd.

Als eerste voorbeeld breiden we de klasse BasisFrame uit als getoond in figuur 4.15: naast een label bevat deze nu ook een tekstveld. Anders dan een label, laat een tekstveld ook invoer toe. We kunnen er iets in intypen en dan op enter drukken. We willen nu dat ons programma daarop reageert door de ingetypte naam op te nemen in de label (zie figuur 4.15 rechts).

FIGUUR 4.15 BasisFrame handelt een event af

OPDRACHT 4.5
Wijzig BasisFrame in het ontwerpscherm als getoond in de linkerkant van figuur 4.15. Gebruik voor het tekstveld de component JTextField uit de palette en geef dit tekstveld de naam 'naamVeld'.

We voegen aan het tekstveld naamVeld *event handling* toe. Als eerste stap schrijven we een private methode die de gewenste actie uitvoert. We noemen deze private methode naamVeldAction: de naam van de betreffende component (naamVeld) met het achtervoegsel Action. We benoemen alle private methoden die een event afhandelen op deze manier.

OPDRACHT 4.6
Ga naar het editorvenster en voeg aan de code een nieuwe private methode toe met de naam naamVeldAction. Deze methode moet de inhoud van het tekstveld uitlezen en dan de tekst van de label overeenkomstig wijzigen. Raadpleeg zonodig de bijlage.

Nu moeten we er nog voor zorgen dat deze methode ook wordt aangeroepen.

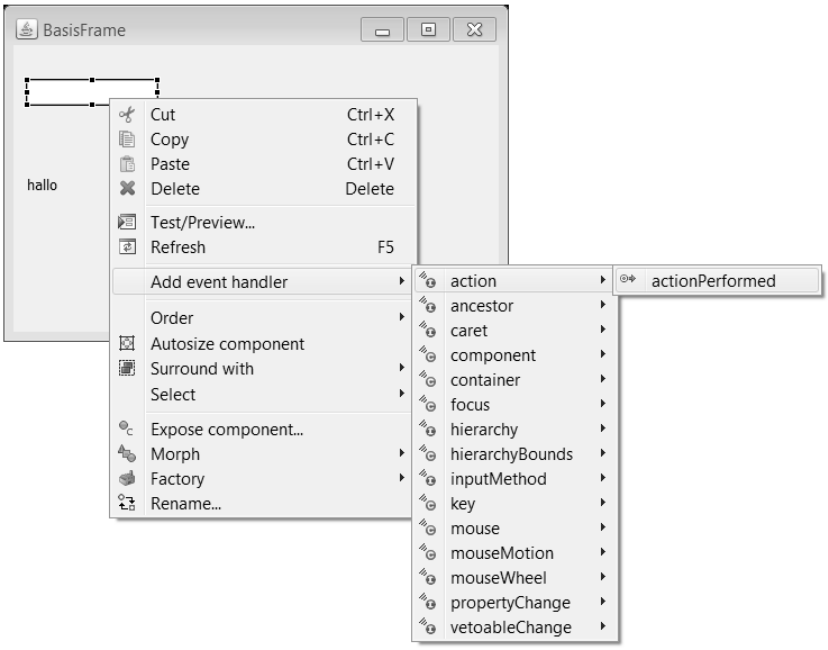
– Ga weer naar het ontwerpvenster, rechtsklik op het naamVeld, selecteer uit het menu de optie Add event handler en vervolgens action en actionPerformed (zie figuur 4.16). In plaats van in het ontwerpscherm, kunt u naamVeld ook in de componentenlijst selecteren.

Aan de gegenereerde code is nu event handling toegevoegd. We moeten er nu nog voor zorgen dat de event handler de private methode naamVeldAction aanroept.

– Ga naar het editorvenster en bekijk de methode getNaamVeld (zie figuur 4.17).

Deze code is op dit moment niet inzichtelijk te maken. Dat hoeft ook niet; we hoeven alleen te weten waar we iets moeten veranderen.

– Voeg aan de methode actionPerformed de aanroep van de private methode naamVeldAction toe (zie figuur 4.18).



FIGUUR 4.16 Event handling toevoegen

```
65  private JTextField getNaamVeld() {
66    if (naamVeld == null) {
67        naamVeld = new JTextField();
68        naamVeld.addActionListener(new ActionListener() {
69          public void actionPerformed(ActionEvent e) {
70          }
71        });
72        naamVeld.setText(" ");
73        naamVeld.setBounds(12, 31, 116, 22);
74        naamVeld.setColumns(10);
75    }
76    return naamVeld;
77  }
```

FIGUUR 4.17 De event handler is toegevoegd aan de code

```
65  private JTextField getNaamVeld() {
66    if (naamVeld == null) {
67        naamVeld = new JTextField();
68        naamVeld.addActionListener(new ActionListener() {
69          public void actionPerformed(ActionEvent e) {
70            naamVeldAction();
71          }
72        });
73        naamVeld.setText(" ");
74        naamVeld.setBounds(12, 31, 116, 22);
75        naamVeld.setColumns(10);
76    }
77    return naamVeld;
78  }
```

FIGUUR 4.18 Een aanroep van methode naamVeldAction is toegevoegd

– Verwerk de klasse BasisFrame. Voer uw naam in in het tekstveld en druk op enter. Uw naam wordt opgenomen in de label.

Voor knoppen en keuzelijsten kunnen we precies dezelfde werkwijze aanhouden. We zetten deze nog eens op een rij.

Event handling toevoegen

Werkwijze

Om event handling toe te voegen aan een knop, tekstveld of keuzelijst met de naam comp gaan we als volgt te werk.
– Voeg in het editorvenster een private methode compAction die de gewenste actie onderneemt toe aan de gegenereerde code.
– Rechtsklik op het ontwerpscherm of in de componentenlijst op comp, selecteer Add event handler en vervolgens action en actionPerformed (er zijn ook andere events, maar die gebruiken we voorlopig niet).
– Voeg de aanroep van compAction toe aan de gegenereerde code van actionPerformed binnen de methode getComp.

OPDRACHT 4.7

In deze opdracht vragen we u een applicatie te maken met een GUI met twee knoppen, waarvan er steeds maar één zichtbaar is. Klikt u op de zichtbare knop, dan wordt deze onzichtbaar en wordt de andere knop zichtbaar.

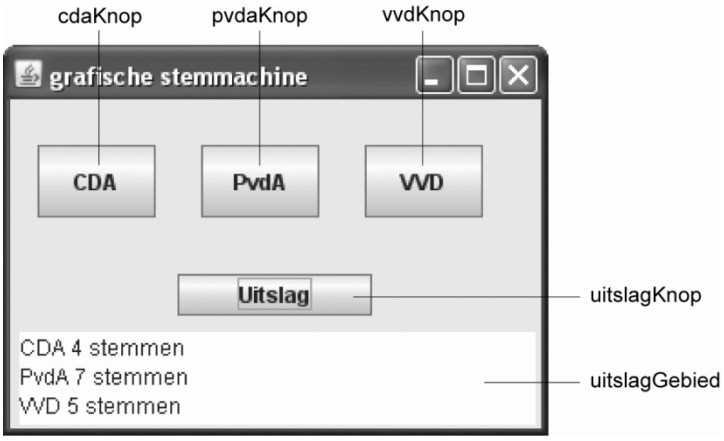
a Maak een nieuw project Le04Knoppen. Maak daarbinnen volgens de procedure beschreven aan het eind van paragraaf 2 een applicatie met een GUI met twee knoppen, waarvan er slechts één zichtbaar is.

b Voeg event handling toe aan de knoppen zodat het gewenste gedrag ontstaat. Raadpleeg de bijlage voor de te gebruiken methode(n).

c Test uw applicatie door de applicatie te verwerken.

4.2 STEMMEN OP EEN PARTIJ

In deze paragraaf voltooien we de stemmachine waar we in opdracht 4.2 mee begonnen zijn. Figuur 4.19 toont nog eens de interface, met de door ons gekozen namen van de componenten (die zijn van belang omdat ze ook in de code voorkomen).



FIGUUR 4.19 Het StemOpPartijFrame met de namen van de componenten

We gaan deze GUI nu koppelen aan de package verkiezingen. Allereerst moeten we zorgen dat deze package binnen het project bekend is.

– Open zo nodig het project Le04Verkiezingen.

– Ga naar het editorvenster. Vouw de lijst met importopdrachten uit en voeg aan het eind daarvan een importopdracht toe voor de klasse Stemmachine:

```
import verkiezingen.Stemmachine;
```

Nu kunnen we event handling toevoegen. Op zich is het duidelijk wat er moet gebeuren als er op de knoppen wordt geklikt: er moet een stem worden uitgebracht of de uitslag moet worden getoond. Dat kan echter alleen als we over een Stemmachine-object beschikken en als daarop vooraf ook de methode zetAan is aangeroepen.

We beginnen met de beschikbaarheid. Het Stemmachine-object dat de stemmen registreert, moet toegankelijk zijn voor alle event handlers. We moeten dit object dus declareren als attribuut.

– Voeg aan de gegenereerde code, meteen na de kop van de klasse StemOpPartijFrame, de volgende regel toe.

```
private Stemmachine stemmachine = new Stemmachine();
```

Deze regel code declareert een attribuut stemmachine en kent daar meteen een nieuw Stemmachine-object aan toe.

Nu moeten we er nog voor zorgen dat al bij de constructie van de StemOpPartijFrame de methode zetAan wordt aangeroepen op dit attribuut. We voegen daartoe opnieuw een private methode toe, ditmaal met de naam mijnInit en roepen die aan vanuit de constructor.

– Voeg aan de code een private methode mijnInit toe als volgt:

```
private void mijnInit() {
  stemmachine.zetAan();
}
```

Plaats deze methode meteen na de door WindowBuilder gegenereerde code. Deze methode mag overal in de klasse staan, maar het is handig om deze niet tussen de gegenereerde code te plaatsen en om altijd dezelfde structuur te gebruiken. Dat maakt de code beter leesbaar.

– Voeg aan de constructor, na de aanroep van de methode initialize, een aanroep van de methode mijnInit toe.

```
public StemOpPartijFrame() {
  initialize();
  mijnInit();
}
```

We hebben er nu voor gezorgd dat de klasse StemOpPartijFrame beschikt over een attribuut stemmachine met een geschikte waarde. Het toevoegen van de event handling is nu niet moeilijk meer; we volgen gewoon de procedure die uiteen is gezet in paragraaf 4.1.

OPDRACHT 4.8

Voeg nu zelf event handling toe aan alle vier de knoppen. Het klikken op een van de knoppen CDA, PvdA of VVD moet leiden tot een stem op

de lijsttrekker van die partij. Het klikken op de knop Uitslag moet leiden tot het tonen van de uitslag (per partij) in het tekstgebied.

Importeren via Quick Fix

In deze paragraaf hebben we netjes eerst een importopdracht toegevoegd voor de klasse Stemmachine en daarna pas een attribuut van dat type gedeclareerd. We kunnen echter het toevoegen van de juiste importopdrachten vrijwel altijd aan Eclipse overlaten. U gebruikt dan gewoon het betreffende type. U declareert bijvoorbeeld een attribuut van type Stemmachine. De compiler signaleert op die regel een fout. Gaat u met de cursor op de foute code staan of klikt u op het lampje naast de regel met de foutmelding, dan verschijnt een aantal suggesties om de fout op te lossen. Het toevoegen van de importopdracht staat vrijwel altijd als eerste bovenaan in het lijstje. Klik hierop en de import wordt toegevoegd. Dat scheelt veel type- en denkwerk. Bovendien worden dan altijd precies die klassen geïmporteerd die u ook echt gebruikt.

4.3 STEMMEN OP EEN KANDIDAAT

Tegen de stemmachine uit de vorige paragraaf is een aantal bezwaren in te brengen. Ten eerste lijkt hij niet erg op een echte stemmachine. We kunnen alleen maar op een partij stemmen en niet, zoals voorgeschreven bij Tweede Kamerverkiezingen, op een kandidaat. Ook het tussentijds kunnen opvragen van de uitslag is niet volgens de democratische regels. Een tweede bezwaar is uit het oogpunt van programmeren interessanter: de GUI is niet erg algemeen. Zodra er aan de stemmachine een partij wordt toegevoegd, is de GUI niet meer bruikbaar. Eigenlijk willen we een GUI die gewoon de partijen en de kandidaten toont die er in de stemmachine voorhanden zijn en dus bruikbaar is voor iedere stemmachine, hoe deze ook is ingericht.
De meeste echte stemmachines hebben een knop per kandidaat, zoals in figuur 4.20. Om deze GUI te maken zouden we in WindowBuilder alle vijftien knoppen een voor een toe kunnen voegen en dan met elke knop een event handler verbinden. Dat levert een GUI op die afhangt van de specifieke kandidatenlijsten. Dat doen we dus niet. Het is overigens wel goed mogelijk om deze GUI te maken zonder kennis vooraf van de partijen en de kandidaten, maar dat lukt niet met een tool zoals WindowBuilder.

FIGUUR 4.20 Een knop per kandidaat

Een goed alternatief is de GUI getoond in figuur 4.21. Deze bevat twee keuzelijsten, een met alle partijen en een met de kandidaten van de geselecteerde partij. De kiezer kiest eerst links een partij. De keuze voor een partij leidt ertoe dat de kandidaten van die partij in de rechterlijst worden geplaatst. De kiezer kiest daar vervolgens een kandidaat uit. Beide keuzes kunnen vrijelijk veranderd worden. Pas wanneer op de knop Stem wordt geklikt, wordt een stem uitgebracht op de gekozen kandidaat. Een knop uitslag linksonder zorgt ervoor dat de uitslag in het uitvoervenster verschijnt, dus buiten de stemmachine zelf.

FIGUUR 4.21 Een stemmachine met twee keuzelijsten

In de volgende opdrachten vragen we u deze GUI zelf te programmeren en wel zo dat deze werkt voor iedere stemmachine, ook met andere partijen en kandidaten. Het programmeerwerk is complexer dan dat voor de vorige GUI en doet ook een beroep op hetgeen u hebt geleerd in de vorige leereenheid.

OPDRACHT 4.9

a Voeg aan het project Le04Verkiezingen een JFrame StemOpKandidaatFrame toe, met twee labels, twee keuzelijsten (JComboBoxes) en twee knoppen als getoond. Noem de keuzelijsten partijKeuze en kandidaatKeuze en de knoppen stemKnop en uitslagKnop.

b Declareer een attribuut stemmachine met als waarde een nieuwe stemmachine. Gebruik Quick Fix om de bijbehorende importopdracht toe te voegen.

c Zorg ervoor dat bij constructie van de GUI de stemmachine wordt aangezet en de namen van alle bij die stemmachine geregistreerde partijen worden opgenomen in de keuzelijst van de partijen. Zorg er ook voor dat er bij het starten van de GUI geen partijnaam te zien is en dat de lijst met kandidaten leeg is (we willen de kiezer niet sturen in zijn keuze). Kijk in de bijlage voor de methoden van klasse JComboBox.

d Verwerk de klasse StemOpKandidaatFrame om te kijken of de GUI tot zover juist is geprogrammeerd.

In de volgende opdrachten voegen we event handling toe. We hebben drie event handlers nodig: voor partijKeuze, voor stemKnop en voor uitslagKnop. We beginnen met de eerste.

Als er een andere partij wordt gekozen, moet de keuzelijst voor de kandidaten eerst worden leeggemaakt. Vervolgens moeten de

kandidaten van de gekozen partij worden toegevoegd. U moet daarbij controleren of er echt wel een partij gekozen is, dus of de geselecteerde index niet –1 is. Dat is nodig omdat elke wijziging van die index een event genereert, ook als die wijziging niet van de gebruiker komt (via de GUI) maar expliciet in de code van het programma staat. De opdracht partijKeuze.setSelectedIndex(–1) uit mijnInit genereert dus ook een event.

OPDRACHT 4.10

a Beschrijf in woorden wat de methode partijKeuzeAction moet doen.
b Implementeer deze methode en voeg event handling toe aan partijKeuze.

Aanwijzing

Zoals blijkt uit de bijlage, worden de keuzen in de keuzelijst vanaf 0 genummerd. Dat geldt ook voor de elementen in een ArrayList (zie leereenheid 3). Als in de lijst index i gekozen is, dan moeten we dus de kandidaten van de i-de partij in de ArrayList hebben.

De volgende event handler is die voor de stemKnop. Deze moet de uitgebrachte stem vastleggen, waarbij de indices uit beide lijsten nodig zijn. Om fouten te voorkomen is het verstandig om, alvorens de methode stemOpNummer aan te roepen, te controleren of er echt een kandidaat gekozen is. Verder is het niet de bedoeling dat de volgende kiezer die in het stemhokje komt, iets te weten kan komen over de keuze van de vorige. We wissen daarom na het uitbrengen van een stem de keuze van de partij en we maken de keuzelijst voor de kandidaten leeg (anders kun je nog steeds zien op welke partij gestemd is door te kijken welke kandidaten er in de lijst staan).

OPDRACHT 4.11

a Beschrijf in woorden wat de methode stemKnopAction moet doen.
b Implementeer deze methode en voeg event handling toe aan stemKnop.

Aanwijzing

Let op: voor de methode stemOpNummer van klasse Stemmachine zijn de partijen en kandidaten vanaf 1 genummerd en niet vanaf 0. Bij de indices uit de keuzelijsten moet dus 1 worden opgeteld.

Tot slot verbinden we nog event handling met de uitslagKnop. Dit is eenvoudig.

OPDRACHT 4.12

Schrijf een methode uitslagKnopAction en voeg event handling toe aan de uitslagKnop.

OPDRACHT 4.13

Test nu de volledige applicatie. Breng tenminste één stem uit op alle kandidaten en controleer de uitslag met behulp van de uitslagKnop.

5 Nabeschouwing

5.1 VOLGORDE

De gegenereerde code binnen een JFrame-klasse bestaat uit achtereenvolgens:

– declaratie van het attribuut contentPane
– attribuutdeclaraties voor de componenten toegevoegd in WindowBuilder
– een methode main
– een constructor
– een private methode initialize
– private methoden getComp voor componenten toegevoegd in WindowBuilder.

Wij voegen daaraan toe:
– declaraties van attributen die we zelf nodig hebben (vaak voor objecten uit de domeinlaag)
– een private methode mijnInit
– private methoden compAction die worden aangeroepen als er een event op component comp optreedt
– aanroepen van compAction in de getComp-methoden.

Om de code overzichtelijk te houden, is het verstandig om voor al deze elementen een vaste volgorde aan te houden. Bovendien is het handig om die volgorde zo te kiezen, dat uw eigen code zoveel mogelijk bij elkaar blijft staan. Wij raden u daarom aan om de volgende volgorde aan te houden:
– plaats uw eigen attributen meteen na de kop van de klasse
– plaats de private methode mijnInit meteen na de gegenereerde code
– plaats de event handlers (private methoden compAction) na mijnInit of, indien er geen mijnInit is, na de gegenereerde code.

De volgorde van de code binnen de grafische klasse wordt dan als volgt:
– eigen attributen
– gegenereerde attributen (contentPane en de toegevoegde componenten)
– gegenereerde methode main
– gegenereerde constructor (waar eventueel een aanroep van mijnInit aan is toegevoegd)
– gegenereerde methode intialize
– gegenereerde methoden getComp
– mijnInit
– methoden compAction.

5.2 HET OPENEN VAN EEN GUI-KLASSE

Wanneer u een GUI-klasse opent die met behulp van WindowBuilder gemaakt is, dan toont Eclipse deze klasse meestal met behulp van WindowBuilder. In het hoofdscherm ziet u meteen zowel het ontwerpscherm als het gewone editorvenster (op verschillende tabbladen). Bij het openen van GUI-klassen die tot de bouwstenen behoren, gaat dit echter soms niet goed. U krijgt dan alleen de code te zien, en geen ontwerpscherm.
In dat geval moet u de klasse weer sluiten en vervolgens expliciet aangeven dat deze geopend moet worden in WindowBuilder. Dat gaat als volgt:
– Rechtsklik op de klasse in het projectvenster, selecteer ‘Open with’ en vervolgens ‘WindowBuilder Editor’.

Het ontwerpscherm komt dan ook tevoorschijn.

SAMENVATTING

Paragraaf 1

Voor de constructie van grafische gebruikersinterfaces (GUI's) stelt Java de Swing-bibliotheek beschikbaar. De klassen uit deze bibliotheek zijn verdeeld over twee packages, namelijk java.awt en javax.swing. Wij gebruiken slechts enkele componenten: labels, knoppen, tekstvelden, tekstgebieden en keuzelijsten, overeenkomend met de klassen JLabel, JButton, JTextField, JTextArea en JComboBox.

De basis van elke Swing-GUI is een hoofdvenster (een JFrame-object) met daarbinnen een contentPane (een JPanel-object). De andere elementen worden binnen het contentPane geplaatst.

Paragraaf 2

Om met behulp van Eclipse een programma te maken met een grafische gebruikersinterface gaan we als volgt te werk.

– Maak zo nodig een nieuw project.
– Voeg aan het project een JFrame toe. Plaats deze bij voorkeur in een package. Wij houden de conventie aan dat de naam van deJFrame eindigt op Frame en die van de package op gui.
– Selecteer in het ontwerpscherm de contentPane (klik binnen in het venster) en geef in de eigenschappenlijst de eigenschap layout de waarde Absolute layout.
– Voeg met behulp van de componentenpalette componenten toe aan het contentPane. Geef iedere component een betekenisvolle naam. Gebruik de eigenschappenlijst om de component de gewenste eigenschappen te geven..

Paragraaf 3

In de code van de visuele klasse is te zien dat er attributen, een main-methode, een constructor en methoden zijn gegenereerd. Met uitzondering van de main-methode en de constructor hebben al deze elementen de toegang private, wat betekent dat ze alleen vanuit de klasse zelf toegankelijk zijn.

Declaraties van attributen hebben (voorlopig) de volgende vorm:

```
private type variabelennaam = expressie;
```

De declaratie van een private methode heeft (voorlopig) de volgende vorm:

```
private void methodenaam() {
  opdrachten
}
```

Paragraaf 4

Om event handling toe te voegen aan een component met de naam comp gaan we als volgt te werk:
– Voeg in het editorvenster een private methode compAction die de gewenste actie onderneemt toe aan de gegenereerde code.
– Rechtsklik op het ontwerpscherm of in de componentenlijst op comp, selecteer Add event handler en vervolgens action en actionPerformed (er zijn ook andere events, maar die gebruiken we voorlopig niet).
– Zoek in de gegenereerde code de methode getComp en voeg daaraan de aanroep naar private methode comAction toe

Als er bij constructie van de GUI bepaalde acties nodig zijn, dan plaatsen we die in een private methode mijnInit. Een aanroep van die methode plaatsen we in de constructor, na de aanroep van initialize.

Als we methoden willen aanroepen op objecten buiten de GUI-klasse, dan moeten we die objecten eerst creëren en toekennen aan een variabele (meestal aan een attribuut).

ZELFTOETS

1 Maak binnen een nieuw project een applicatie met de gebruikersinterface getoond in figuur 4.22. Als de gebruiker de naam van een kandidaat intypt, worden woonplaats en partij getoond (linkerscherm). Is de naam onbekend, dan wordt niets getoond (rechterscherm).

FIGUUR 4.22 Gegevens over kandidaten opvragen

Aanwijzing
Gebruik de volgende methode van de klasse Partij:

```
public Kandidaat zoek(String naam)
```
Levert het Kandidaat-object met de gegeven naam, mits er zo'n object is in de lijst met kandidaten van deze partij. Zo niet, dan wordt null opgeleverd.

TERUGKOPPELING

1 Uitwerking van de opgaven

4.1 U zult zien dat de titelbalk in het ontwerpscherm nu ook is aangepast, evenals de aanduiding van het element in componentenlijst op het scherm: daar staat nu (javax.swing.JFrame)-"BasisFrame".

4.2 a Zie desgewenst paragraaf 2.4 van leereenheid 2.
b Zie desgewenst figuren 4.6 en 4.7 en de bijbehorende tekst.
c Zorg er voor dat u de contentPane hebt geselecteerd en niet het frame zelf.
d Ook dit gaat via de eigenschappenlijst. Zorg nu echter dat het frame zelf geselecteerd is en niet de contentPane.
e Het toevoegen van een knop en een tekstgebied gaat op dezelfde manier als het toevoegen van een label, alleen gebruikt u nu uit het palette de elementen JButton en JTextArea. Kies als namen bijvoorbeeld cdaKnop, pvdaKnop, vvdKnop, uitslagKnop en uitslagGebied.
f Geef een knop een opschrift door de knop op het ontwerpscherm te selecteren en in de eigenschappenlijst de waarde van de eigenschap text aan te passen.
g Als u dit programma verwerkt, krijgt u een gebruikersinterface te zien als in de figuren 4.1 en 4.3 (maar zonder uitslag). U kunt klikken op de knoppen, maar dat heeft uiteraard geen effect. We hebben nog nergens aangegeven wat er bij zo'n klik moet gebeuren.

4.3 Deze klasse is al een stuk groter dan BasisFrame. Ook hier is het handig om de code zover mogelijk in te klappen om de structuur te zien.
Net als bij BasisFrame zijn er attributen contentPane, plus attributen voor alle grafische componenten die we op het ontwerpscherm hebben geplaatst: cdaKnop, pvdaKnop, vvdKnop, uitslagKnop en uitslagGebied.
Ook het aantal methoden is flink toegenomen. Naast de constructor en de methoden initialize en main, die we al van BasisFrame kennen, zijn er nu ook methoden getCdaKnop, getPvdaKnop, getVvdKnop, getUitslagKnop en getUitslagGebied: een methode voor elke toegevoegde component. Ook al deze methoden hebben de aanduiding private.

4.4 a Zowel de klasse JLabel als de klasse JTextField heeft een methode getText om de tekst in de component op te vragen. Om deze toe te voegen aan het tekstgebied gebruiken we de methode append. De methode kan er daarmee als volgt uitzien:

```
private void test2(){
  String s1 = label1.getText();
  String s2 = tekstVeld1.getText();
  tekstGebied1.append(s1);
  tekstGebied1.append(s2);
}
```

b Om een keuze toe te voegen aan een keuzelijst gebruiken we de methode addItem. Dit doen we drie keer. De methode wordt daarmee als volgt:

```
private void vulKeuzeLijst(){
```

```
  keuzeLijst1.addItem("keuze1");
  keuzeLijst1.addItem("keuze2");
  keuzeLijst1.addItem("keuze3");
}
```

4.5 – Wijzig de tekst van de label in “Hallo ?”. U kunt dit doen via de eigenschappenlijst van de label.
– Voeg met behulp van de palette een component van het type JTextField toe. Geef deze de naam naamVeld.

4.6 De methode kan als volgt uitzien:

```
private void naamVeldAction(){
  String naam = naamVeld.getText();
  halloLabel.setText("Hallo " + naam + "!");
}
```

Eclipse geeft een waarschuwing dat deze methode nergens wordt aangeroepen. Dat klopt. Die aanroep gaan we nog toevoegen.

4.7 a We zetten de stappen nog eens op een rij.
– Maak een nieuw project aan volgens de procedure beschreven in paragraaf 2.4 van leereenheid 2.
– Voeg een JFrame KnoppenFrame toe aan dit project. Vul bijvoorbeeld 'knoppengui' in als packagenaam. U hoeft dus niet apart eerst een nieuwe package aan te maken. Dit gebeurt vanzelf.
– Ga naar het ontwerpscherm, selecteer de contentPane en geef in de eigenschappenlijst de eigenschap layout de waarde Absolute layout.
– Voeg met behulp van de componentenpalette twee knoppen toe (knop1 en knop2). Geef ze eventueel een opschrift.
– Maak één knop onzichtbaar door in de eigenschappenlijst van die knop de eigenschap visible de waarde false te geven (hiervoor dient u de advanced properties te selecteren).

b Volg de procedure beschreven aan het eind van paragraaf 4.1.
– Schrijf voor elke knop een private methode die de knop zelf onzichtbaar maakt en de andere zichtbaar.
De code van deze methoden ziet er bijvoorbeeld als volgt uit.

```
private void knop1Action(){
  knop1.setVisible(false);
  knop2.setVisible(true);
}

private void knop2Action(){
  knop1.setVisible(true);
  knop2.setVisible(false);
}
```

– Voeg aan beide knoppen via het ontwerpscherm event handling toe door te rechtsklikken op de knop en uit het menu eerst Add event handler en dan action en actionPerformed te kiezen.
– Voeg in getKnop1 een aanroep van knop1Action en in getKnop2 een aanroep van knop2Action toe. We tonen nog een keer deze code.

```
private JButton getKnop1() {
  if (knop1 == null) {
    ...
```

```
    knop1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        knop1Action();
      }
    });
    ...
  }
}

private JButton getKnop2() {
  if (knop2 == null) {
    ...
    knop2.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        knop2Action();
      }
    });
    ...
  }
}
```

4.8 Volg weer de procedure beschreven in paragraaf 4.1. Voeg ten eerste aan de code de benodigde private methoden toe, bijvoorbeeld als volgt.

```
private void cdaKnopAction(){
  stemmachine.stemOpNummer(1, 1);
}

private void pvdaKnopAction(){
  stemmachine.stemOpNummer(2, 1);
}

private void vvdKnopAction(){
  stemmachine.stemOpNummer(3, 1);
}

private void uitslagKnopAction(){
  String uitslag = stemmachine.geefUitslagPerPartij();
  uitslagGebied.setText(uitslag);
}
```

In plaats van de methode stemOpNummer kunt u ook gebruikmaken van de methode stem, waarbij u de lijsttrekker benoemt; de opdracht in cdaKnopAction wordt dan dus stemmachine.stem("Jan Peter Balkenende"). Het gebruik van stemOpNummer is echter iets algemener (als het CDA een andere lijsttrekker krijgt, gaat het nog steeds goed).

Voeg vervolgens in het ontwerpscherm aan alle knoppen event handling toe (rechtsklikken, Add event handler en vervolgens action en actionPerformed selecteren).

Vul ten slotte de aanroepen van de private methoden in in de code. We laten deze code niet meer zien. U kunt eventueel het volledige project raadplegen via de cursussite.

4.9 a We nemen aan dat u dit inmiddels kunt. Vergeet niet de eigenschap layout van de contentPane de waarde Absolute layout te geven.

Eclipse geeft bij de gegenereerde code voor de comboboxen waarschuwingen. Negeer deze waarschuwing. Pas nooit zelf de door Eclipse gegenereerde code aan.

b De declaratie ziet er als volgt uit:

```
private Stemmachine stemmachine = new Stemmachine();
```

Gebruik Quick Fix om de importopdracht voor de klasse Stemmachine toe te voegen.

c We schrijven een private methode mijnInit die achtereenvolgens:
– de stemmachine aanzet
– de lijst met partijen opvraagt
– van elke partij de naam opvraagt en deze toevoegt aan de keuzelijst met behulp van de methode addItem
– de methode setSelectedIndex aanroept met parameter -1 op beide lijsten om te zorgen dat er in de lijsten niets geselecteerd is

De code ziet er als volgt uit (gebruik Quick Fix om de importopdrachten voor ArrayList en Partij toe te voegen).

```
private void mijnInit(){
  stemmachine.zetAan();
  ArrayList<Partij> partijen = stemmachine.getPartijen();
  for (Partij p: partijen) {
    String naam = p.getNaam();
    partijKeuze.addItem(naam);
  }
  partijKeuze.setSelectedIndex(-1);
  kandidaatKeuze.setSelectedIndex(-1);
}
```

Een aanroep van deze methode wordt toegevoegd aan de constructor, als volgt:

```
public StemOpKandidaatFrame() {
  initialize();
  mijnInit();
}
```

4.10 a De methode partijKeuzeAction moet het volgende doen:

Bepaal de gekozen index in partijKeuze
Als die niet gelijk is aan -1, doe dan het volgende:
Maak de keuzelijst met kandidaten leeg.
Bepaal het Partij-object dat bij de keuze hoort.
Haal daarvan de lijst kandidaten op en plaats hun namen in de keuzelijst voor de kandidaten.
Zorg ervoor dat er nog geen kandidaat gekozen is.

De code kan er als volgt uitzien (gebruik Quick Fix om de importopdracht voor Kandidaat toe te voegen).

```
private void partijKeuzeAction() {
  int nummer = partijKeuze.getSelectedIndex();
  if (nummer != -1) {
    kandidaatKeuze.removeAllItems();
    ArrayList<Partij> partijen =
              stemmachine.getPartijen();
    Partij gekozenPartij = partijen.get(nummer);
```

```
    ArrayList<Kandidaat> kandidaten =
             gekozenPartij.getKandidaten();
    for (Kandidaat kandidaat: kandidaten) {
      kandidaatKeuze.addItem(kandidaat.getNaam());
    }
  }
  kandidaatKeuze.setSelectedIndex(-1);
}
```

Uiteraard moet er ook voor gezorgd worden dat deze methode wordt aangeroepen vanuit getPartijKeuze.

4.11 a De methode stemKnopAction moet het volgende doen:

Bepaal de nummers van de partij en van de kandidaat waarop wordt gestemd (beide 1 hoger dan de gekozen index in de keuzelijst).
Als het nummer van de kandidaat ongelijk aan 0 is, doe dan het volgende:
Breng een stem uit op de gekozen partij en kandidaat.
Zet de geselecteerde index van partijKeuze op -1.
Verwijder de elementen uit kandidaatKeuze.

b De code ziet er dan als volgt uit:

```
private void stemKnopAction(){
  int partijnr = partijKeuze.getSelectedIndex() + 1;
  int kandidaatnr = kandidaatKeuze.getSelectedIndex() + 1;
  if (kandidaatnr != 0) {
    stemmachine.stemOpNummer(partijnr, kandidaatnr);
  }
  partijKeuze.setSelectedIndex(-1);
  kandidaatKeuze.removeAllItems();
}
```

Vergeet ook nu weer niet het toevoegen van de event en het aanroepen van deze methode vanuit getStemKnop.

4.12 De methode uitslagKnopAction is eenvoudig:

```
private void uitslagKnopAction(){
  String uitslag = stemmachine.geefUitslagPerKandidaat();
  System.out.println(uitslag);
}
```

4.13 Geen terugkoppeling.

1 Uitwerking van de zelftoets

1 – Maak een nieuw project en voeg aan dit project een JFrame toe. Wij noemden deze klasse KandidaatFrame en plaatsten deze in de package kandidaatgui.
– Ga naar het ontwerpscherm. Zet de eigenschap ‘layout’ van het contentPane op Absolute layout. Wijzig in de eigenschappenlijst ook de titel van het JFrame.
– Voeg aan het ontwerpscherm een tekstveld en drie labels toe; wij noemden de componenten (van boven naar beneden) kandidaatLabel, kandidaatVeld, woonplaatsLabel en partijLabel. Geef kandidaatLabel de

tekst “Typ de naam van een kandidaat in”; maak de tekst van de andere drie componenten leeg.
– Ga nu naar het editorvenster. Voeg aan de code importopdrachten toe als volgt:

```
import verkiezingen.Stemmachine;
import verkiezingen.Partij;
import verkiezingen.Kandidaat;
import java.util.ArrayList;
```

– Voeg ook een eigen attribuut toe:

```
public class KandidaatFrame extends JFrame {
  // eigen attribuut
  private Stemmachine stemmachine = new Stemmachine();
```

– Voeg aan de code een private methode mijnInit toe als volgt en roep deze aan vanuit de constructor:

```
private void mijnInit(){
  stemmachine.zetAan();
}
```

– Schrijf een private methode kandidaatVeldAction, die er als volgt uit kan zien:

```
private void kandidaatVeldAction(){
  woonplaatsLabel.setText("");
  partijLabel.setText("");
  ArrayList<Partij> partijen = stemmachine.getPartijen();
  String naam = kandidaatVeld.getText();
  for (Partij p : partijen) {
    Kandidaat k = p.zoek(naam);
    if (k != null) {
      woonplaatsLabel.setText(k.getWoonplaats());
      partijLabel.setText(p.getNaam());
    }
  }
}
```

– Voeg via het ontwerpscherm event handling toe aan het tekstveld en roep de methode kandidaatVeldAction aan vanuit getKandidaatVeld.

– Verwerk de klasse en controleer of alles naar behoren werkt.

Bijlage

Een deel van de interface van vijf standaardklassen

1 Methoden van de klasse JLabel

`public String getText()`
Returns the text string that the label displays.

`public void setText(String text)`
Defines the single line of text this component will display.

`public void setVisible(boolean b)`
Makes the component visible or invisible.

2 Methoden van de klasse JTextArea

`public String getText()`
Returns the text contained in this TextArea.

`public void setText(String text)`
Sets the text of this TextArea to the specified text.

`public void append(String text)`
Appends the given text to the end of the document.

`public void setVisible(boolean b)`
Makes the component visible or invisible.

3 Methoden van de klasse JTextField

`public String getText()`
Returns the text contained in this TextField.

`public void setText(String t)`
Sets the text of this TextField to the specified text.

`public void setVisible(boolean b)`
Makes the component visible or invisible.

4 Methoden van de klasse JButton

`public String getText()`
Returns the button's text.

`public void setText(String t)`
Sets the button's text.

`public void setVisible(boolean b)`
Makes the component visible or invisible.

5 Methoden van de klasse JComboBox

`public void addItem(String item)`
Adds an item to the item list.

`public int getSelectedIndex()`
Returns an integer specifying the currently selected list item, where 0 specifies the first item in the list; or -1 if no item is selected.

`public void setSelectedIndex(int anIndex)`
Selects the item at index anIndex, where 0 specifies the first item in the list and -1 indicates no selection.

`public void removeAllItems()`
Removes all items from the item list.