Verkiezingen in Java

Introductie *23*

Leerkern *25*

1 Inleiding *25*
  1.1 Programmeerstijlen *25*
  1.2 Java *27*
2 Een eerste programma *30*
  2.1 De stemmachine *30*
  2.2 Een programma dat stemmen uitbrengt *31*
  2.3 Het programma compileren en verwerken *34*
3 Objectoriëntatie *35*
  3.1 Wat is een object? *36*
  3.2 Attributen en methoden *37*
  3.3 Klassen *39*
4 Programma's nader bekeken *40*
  4.1 Applicaties *41*
  4.2 Declaraties, variabelen en typen *43*
  4.3 Toekenningen *45*
  4.4 Methodeaanroepen *48*
  4.5 Constructors en creatieopdrachten *51*
  4.6 Fouten in programma's *52*
  4.7 Netjes programmeren *54*
5 Een Java-programma dat tegels tekent *54*

Samenvatting *56*

Zelftoets *58*

Terugkoppeling *60*

1 Uitwerking van de opgaven *60*
2 Uitwerking van de zelftoets *65*

Bijlage: De interfaces van de klassen in de package verkiezingen *68*

Leereenheid 1

Verkiezingen in Java

INTRODUCTIE

Programmeren is een computer vertellen wat deze moet doen, in een taal die die computer kan begrijpen. In deze cursus leert u programmeren in zo'n taal, namelijk Java.

In deze leereenheid schrijft u uw eerste twee programma's. Het eerste programma bestuurt een stemmachine, die gebruikt wordt om een aantal stemmen uit te brengen op kandidaten van de drie grootste partijen. Vervolgens toont het programma de uitslag. Met behulp van het tweede programma tekent u tegelvloertjes zoals hieronder getoond.

Dat u deze programma's meteen kunt schrijven, komt doordat u niet al het werk zelf hoeft te doen; u krijgt hulp van voorgedefinieerde *objecten*. Een voorbeeld van zo'n object is een tegel, die u opdracht kunt geven zichzelf te tekenen. Hoe zo'n tegel dat precies doet, hoeft u niet te weten. Het begrip object speelt een centrale rol in Java. Java heet daarom een objectgeoriënteerde taal.

Voor u begint met programmeren, vertellen we in de eerste paragraaf in het kort iets over programmeren en programmeertalen in het algemeen, en over Java in het bijzonder. In paragraaf 2 tonen we een eerste Java-programma, dat u zelf op de computer gaat uitvoeren. Paragraaf 3 bevat een eerste inleiding in objectoriëntatie. Paragraaf 4 kijkt nauwkeuriger naar de elementen van de taal Java die we in het eerste blok zullen gebruiken. In de laatste paragraaf schrijft u het programma voor de tegels.

Deze programma's hebben, anders dan u misschien gewend bent, geen grafische interface. De stemmachine bevat geen rijen knoppen, en de uitslag verschijnt als tekst in een ouderwets commandvenster. In deze leereenheid gebruiken we bovendien alleen het meest eenvoudige gereedschap: het genoemde commandvenster, en Kladblok (Notepad) als tekstverwerker.
In de leereenheden hierna geven we u meer gereedschap in handen. In leereenheid 2 leert u werken met de ontwikkelomgeving Eclipse; in leereenheid 4 leert u grafische interfaces maken.

LEERDOELEN

Na het bestuderen van deze leereenheid wordt verwacht dat u

- kunt aangeven hoe een Java-programma wordt verwerkt en wat daarbij de rol is van de Java-compiler en van de Java Virtual Machine
- kunt uitleggen wat wordt bedoeld met een object en een klasse, en wat het verschil is tussen beide begrippen
- kunt uitleggen wat attributen en wat methoden zijn
- weet wat bedoeld wordt met de interface van een klasse
- het verschil kunt aangeven tussen een Java-applicatie en een Java-applet
- weet waarvoor de volgende sleutelwoorden in Java gebruikt kunnen worden: import, public, class, void, int, new
- weet wat wordt bedoeld met de signatuur van constructors en methoden
- weet wat wordt bedoeld met een variabele en een type
- de twee soorten types in Java kunt noemen en het verschil daartussen kunt aangeven
- in Java de eenvoudigste vormen van variabelendeclaraties, methode-aanroepen, creatie-expressies, toekenningen en printopdrachten kunt formuleren
- met behulp van deze elementen een eenvoudig programma kunt schrijven
- weet wat de functie is van commentaar en op twee manieren commentaar kunt opnemen in een programma
- twee soorten fouten kunt noemen die kunnen optreden in een programma
- de betekenis kunt geven van de volgende kernbegrippen: machinetaal, hogere programmeertaal, programmeerstijl, compiler, bytecode, syntaxis, package, .class-bestand, type, primitief type, referentietype, expressie, formele parameter, actuele parameter, terugkeerwaarde, exception.

Studeeraanwijzingen

We wijzen u er nogmaals op dat in het eerste blok niet alles tot in detail wordt uitgelegd; we zijn regelmatig beknopt in de uitleg of vragen iets in te typen dat u misschien niet precies begrijpt. Blijft u vooral niet op dergelijke details hangen; in het vervolg van de cursus (vanaf leereenheid 5) zullen uw vragen worden beantwoord.

De studielast van deze leereenheid bedraagt circa 7 uur.

LEERKERN

1 Inleiding

1.1 PROGRAMMEERSTIJLEN

Programma

Een computer doet niets uit zichzelf. Voor iedere taak die door een computer wordt uitgevoerd, is een *programma* nodig: een voorschrift dat door de computer kan worden verwerkt, waarin precies is vastgelegd wat de computer moet doen.
Hoe verloopt die verwerking? Het hart van een computer is de processor, ook wel de centrale verwerkingseenheid genoemd. Deze processor kan eenvoudige instructies verwerken, met betekenissen als 'tel deze twee getallen bij elkaar op', 'zet deze rij enen en nullen in geheugenplaats A' of 'ga verder met de instructie op geheugenplaats X als de inhoud van geheugenplaats A ongelijk is aan nul'. Deze eenvoudige instructies vormen de *machinetaal*. Ieder type processor heeft zijn eigen machinetaal. Een programma is niets anders dan een rij van die instructies.

Machinetaal

Toen computers pas bestonden, werden programma's direct in machinetaal geschreven. Inmiddels hoeven we als programmeur niet meer te weten hoe de machinetaal eruitziet van de computer die we gebruiken. We formuleren ons programma in een geschikte algemene programmeertaal en laten het aan de computer over daar machinetaal van te maken. Zo'n *hogere programmeertaal* is dus, in tegenstelling tot een machinetaal, *onafhankelijk* van welke processor dan ook.

Hogere programmeertaal

Voorbeeld

In het begin leken hogere programmeertalen nog vrij veel op machinetalen. Figuur 1.1 laat een stukje zien uit een programma geschreven in de allereerste hogere programmeertaal Fortran, waarvan de oudste versie stamt uit 1954.

```
1 IF (X) 2, 2, 3
2 STOP
3 X = X_- D
  N = N + 1
  GOTO 1
```

FIGUUR 1.1 Een stukje programma in Fortran

Een Fortran-programma bestond net als een machinetaalprogramma uit een rij eenvoudige instructies. Veel structuur had een Fortran-programma verder niet. Naarmate de hardware krachtiger werd en de programma's groter, werd de noodzaak van een goede structurering van programma's ook steeds groter. Een programma dat uit een lange lijst van een paar miljoen instructies bestaat, is erg moeilijk te begrijpen voor een menselijke lezer, zelfs als die lezer een ervaren programmeur is. Het is bijna onmogelijk om in zo'n programma een fout te vinden en te verbeteren, of het zonder fouten te introduceren aan te passen.

Programmeerstijl

Verschillende manieren om programma's te structureren, leiden tot verschillende *programmeerstijlen*. Elk van die stijlen komt overeen met een manier om tegen een programma aan te kijken; ze geven als het ware elk hun eigen antwoord op de vraag wat nu eigenlijk een programma is. Die vraag wordt daarbij niet langer benaderd vanuit de computer die het programma moet verwerken, maar vanuit de programmeur die het moet schrijven.

Imperatieve of procedurele programmeerstijl

De *imperatieve* of *procedurele programmeerstijl* brengt structuur aan in een programma via een verdeel-en-heerstactiek. Het op te lossen probleem wordt gesplitst in stappen en die weer in kleinere stappen, enzovoort. Deze stijl is lange tijd heel populair geweest. Tot de jaren negentig waren de meest gebruikte programmeertalen (bijvoorbeeld COBOL, Pascal, Basic en C) op deze programmeerstijl gebaseerd.

Voorbeeld

Stel, we willen een programma een straat laten tekenen in de vorm van een rijtje van tien huizen met aan weerszijden een boom. In de procedurele taal Pascal kan de eerste opzet van dat programma eruitzien als in figuur 1.2. Merk op dat dit codefragment een stuk begrijpelijker is dan dat uit figuur 1.1, ook als u nog nooit geprogrammeerd hebt. De derde regel, die met het woord for begint, kunt u lezen als 'doe tien keer: teken een huis'

```
begin
  tekenBoom();
  for i:=1 to 10 do tekenHuis();
  tekenboom();
end
```

FIGUUR 1.2 Een stukje programma in Pascal

De opdrachten tekenBoom en tekenHuis die in dit codefragment voorkomen, zijn geen opdrachten die direct vertaald kunnen worden naar opdrachten in de machinetaal die de computer kan verwerken; het zijn stappen waarin de programmeur het probleem heeft opgesplitst. De programmeur moet vervolgens zelf zogeheten procedures schrijven die de computer vertellen hoe die stappen moeten worden uitgevoerd. Zo'n procedure is zelf ook een stukje programma, waarin weer andere door de programmeur bedachte stappen mogen voorkomen. Het programma uit figuur 1.2 zal een procedure tekenHuis bevatten om een huis te tekenen. Daarin zouden stappen kunnen voorkomen die achtereenvolgens de gevel, het dak, de ramen en de deur tekenen. De splitsing in stappen gaat verder tot er alleen opdrachten over zijn die wél direct vertaald kunnen worden naar machinetaal.

Een procedureel programma is voor een menselijke lezer al een stuk makkelijker te begrijpen dan een ongestructureerd Fortran-programma, omdat het uit kleine stukjes bestaat die voor een kenner van de programmeertaal goed te volgen zijn (tenminste, als de programmeur goed werk heeft geleverd). Toch werden ook procedurele programma's op den duur te groot om nog begrijpelijk te zijn; onder meer omdat ze niet erg gestructureerd omgaan met gegevens. Als het programma bijvoorbeeld bestellingen verwerkt van klanten, dan is het moeilijk om snel uit te vinden waar die bestellingen en klanten in het programma precies zitten en welke procedures er iets mee doen.

Objectgeoriënteerde programmeerstijl

Sinds de jaren '90 is de *objectgeoriënteerde programmeerstijl* populair geworden. Deze staat nog verder af van machinetaal dan de procedurele stijl. In deze programmeerstijl wordt ervan uitgegaan dat een programmeertaak wordt uitgevoerd door een verzameling objecten die met elkaar samenwerken. Elk object is in staat om op verzoek bepaalde opdrachten uit te voeren. Bij het uitvoeren van zo'n opdracht kan het object ook weer andere objecten te hulp roepen.

Voorbeeld

Een codefragment om een straat te tekenen, kan in de objectgeoriënteerde taal Java er als volgt uitzien:

```
mijnStraat = new Straat(10);
mijnStraat.teken();
```

In de eerste regel wordt een Straat-object gemaakt met tien huizen, en in de tweede regel wordt aan dat object een tekenopdracht gegeven. Het is de taak van de programmeur om aan te geven wat voor soort objecten er zijn (bijvoorbeeld straten, huizen, bomen, daken...), wat die objecten kunnen (bijvoorbeeld zichzelf tekenen) en wanneer ze dat moeten doen.

Voorbeelden van objectgeoriënteerde talen zijn naast Java, de taal die u in deze cursus leert, C++, Delphi en C#.

Er zijn nog andere programmeerstijlen, zoals de functionele programmeerstijl die een programma beschouwt als een mechanisme om een gegeven invoer af te beelden op een bijbehorende uitvoer, en de logische programmeerstijl die een programma ziet als een geheel van logische relaties. Deze programmeerstijlen hebben echter buiten de universitaire wereld nog niet algemeen ingang gevonden.

Het gebruik van een bepaalde programmeerstijl of -taal is overigens op zich nog absoluut geen garantie voor een goed en begrijpelijk programma. Een taal als Java biedt de mogelijkheid om goed gestructureerde en begrijpelijke programma's te schrijven, maar het is aan de programmeur om die mogelijkheid te benutten. Ook in Java kun je onbegrijpelijke programma's schrijven die uit vrijwel niets anders bestaan dan een lange lijst van eenvoudige instructies. We willen u in deze cursus echter meer leren dan alleen de basis van de taal Java; we willen u leren hoe u Java kunt gebruiken om een begrijpelijk en daardoor onderhoudbaar programma te schrijven.

1.2 JAVA

De taal Java werd in mei 1995 door Sun uitgebracht en verwierf zich in korte tijd een enorme populariteit: er was indertijd sprake van een echte Java-hype. Dat had drie hoofdredenen:

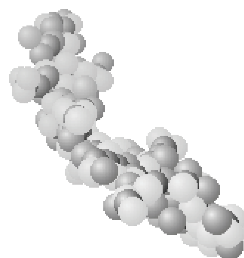
De eerste en belangrijkste reden was de positionering van Java als taal voor het internet. Tot de komst van Java waren webpagina's in het toen nog jonge world wide web statisch; de enige vorm van interactie was klikken op een hyperlink. Spelletjes spelen, iets bestellen, al die dingen waar we nu volkomen aan gewend zijn, waren er nog niet bij. Java bood als eerste een mogelijkheid om interactie in te bouwen, door middel van Java-programma's die opgenomen worden in een webpagina. Als zo'n pagina wordt opgevraagd, wordt het programma meegestuurd en verwerkt op de computer van de opvrager. Dergelijke programma's worden *applets* genoemd. De browser moet zo'n applet wel aankunnen, wat tot dan toe niet het geval was. Netscape, indertijd de meest populaire browser, zag onmiddellijk het potentieel van Java en bouwde het in zijn browser in; Internet Explorer moest toen wel volgen.

Zie ook leereenheid 14

Applet

Om een idee te geven van de impact van Java-applets, volgt hier een citaat uit een artikel over de eerste jaren van Java, afkomstig van de website van Sun (java.sun.com). Het citaat gaat over de allereerste openbare demo van een applet. Gosling is een van de ontwerpers van Java; Mosaic is een inmiddels in onbruik geraakte browser.

As the talk began, Gosling noticed that many people were only casually paying attention. After all, what was so exciting about a new language driving a page of text and illustrations in a clone of Mosaic? Then Gosling moved the mouse over an illustration of a 3D molecule in the middle of the text. The 3D molecule rotated with the mouse movement. Back and forth, up and around. 'The entire audience went "Aaaaaaah"', says Gosling. 'Their view of reality had completely changed because it MOVED'. Now everyone was paying close attention.



FIGUUR 1.3 De eerste demo-applet zag er ongeveer zo uit

Inmiddels is het belang van applets afgenomen. Er zijn meer technieken beschikbaar gekomen voor het inbouwen van mogelijke interactie op een webpagina, zoals JavaScript (de naam suggereert dat JavaScript een soort Java is, maar dat is niet waar) en Flash. Tegenwoordig vindt Java zijn belangrijkste toepassingsgebied weliswaar nog steeds in netwerkprogrammering, maar dan voor programma's die op de server draaien (op de computer die de webpagina beheert) en niet, zoals applets, op de client (op de computer die de webpagina opvraagt). In leereenheid 14 leert u hoe u een Java-applet kunt schrijven.

Een tweede reden voor de populariteit van Java was dat Sun Java gratis ter beschikking stelde, iets wat in die tijd zeer zeldzaam was (de opensourcebeweging kwam pas later op): iedereen mocht Java installeren, of Java inbouwen in een product.

De derde reden is, dat Java als programmeertaal goed in elkaar zit, hoewel er is ook kritiek mogelijk is op het progamma.

De positionering van Java als taal voor het internet werd niet pas bij de lancering bedacht; er was al bij het ontwerp terdege rekening mee gehouden. De mogelijkheid om Java-programma's in te bouwen in een webpagina heeft namelijk nogal wat consequenties. Webpagina's moeten op elk type computer bekeken kunnen worden, en dus moeten ook Java-applets op elk type computer kunnen draaien.

Om te begrijpen hoe daar in Java mee omgegaan is, moeten we eerst kijken naar de manier waarop programma's, geschreven in een hogere programmeertaal, meestal worden verwerkt.
We zagen al dat een dergelijk programma eerst vertaald moet worden naar machinetaal. Een programma dat een dergelijke vertaling voor zijn rekening neemt, heet een *compiler* (in het Nederlands: vertaler). Voor elk type processor is die vertaling weer anders. Voor elke combinatie van programmeertaal en type processor is er dus een compiler nodig. Voortaan gebruiken we vaak de term *compileren* in plaats van vertalen, en de term *compilatie* voor het vertaalproces.

Compiler

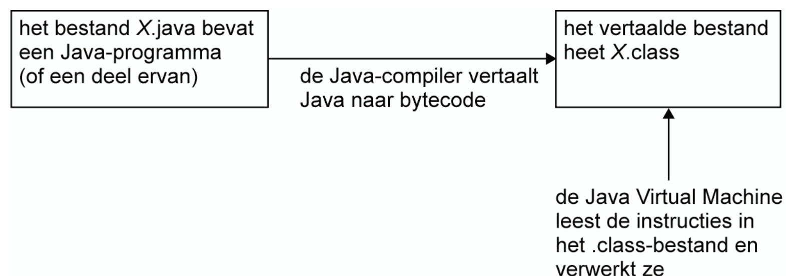
Java-compilers vertalen een Java-programma echter niet naar 'echte' machinetaal, maar naar de machinetaal voor een denkbeeldige Java-computer. De machinetaal van die computer wordt *bytecode* genoemd. Net als een gewoon machinetaalprogramma bestaat die bytecode uit een reeks eenvoudige instructies, alleen is er geen echte processor die ze kan verwerken. In plaats daarvan is er (wederom) een programma nodig dat de instructies een voor een leest en verwerkt: een *Java Virtual Machine* (vaak afgekort tot JVM).

Bytecode

Java Virtual Machine (JVM)

Figuur 1.4 illustreert deze gang van zaken. Het vertaalde (gecompileerde) bestand, dat de bytecode bevat, heeft extensie .class en wordt daarom het *.class-bestand* genoemd.

.class-bestand



FIGUUR 1.4 Compilatie en verwerking van een Java-programma

Deze werkwijze heeft één groot nadeel. De JVM is geen machine met een echte processor, maar een programma. Een echte processor kan instructies veel sneller verwerken dan een programma en dus leidt deze werkwijze als er geen maatregelen worden genomen, tot snelheidsverlies (van ongeveer een factor tien ten opzicht van echte machinecode). Via allerlei slimme trucs waar we hier niet op ingaan, is het snelheidsverlies echter vrijwel geëlimineerd.

Daar staan een aantal voordelen tegenover. Omdat een gecompileerd Java-programma uit bytecode bestaat, kan het op verschillende soorten computers draaien, mits die over een Java Virtual Machine beschikken

(en dat is niet zo'n ingewikkeld programma). Dat maakt Java-programma's minder systeemafhankelijk dan programma's in andere talen. Dat is met name nodig voor de inbouw van Java-applets in een webpagina: wat met de pagina wordt meegestuurd, is het gecompileerde .class-bestand.
Deze werkwijze maakt het ook mogelijk om een veiligheidscontrole uit te voeren op applets; we mogen er immers niet van uitgaan dat elke Java-programmeur goede bedoelingen heeft, en dus moet het technisch onmogelijk zijn dat de applet de computer misbruikt waar die op draait.

OPGAVE 1.1
Eerder merkten we op dat Netscape aangepast moest worden om Java-applets te kunnen tonen. Wat was die benodigde aanpassing?

Java-bibliotheek

Java biedt niet alleen een programmeertaal, maar ook een grote *bibliotheek* van standaardcomponenten die iedere programmeur kan gebruiken en waarvan de bytecode is ingebouwd in de JVM. Sinds 1995 heeft met name deze bibliotheek zich sterk ontwikkeld; de versie die wij gebruiken in deze cursus is versie 6. Bij het uitkomen van versie 5 werd niet alleen de bibliotheek maar ook de taal zelf uitgebreid.

2 Een eerste programma

2.1 DE STEMMACHINE

In deze paragraaf tonen we een eerste Java-programma. In dit programma gaan we met behulp van een stemmachine een aantal stemmen uitbrengen. De stemmachine waarmee we werken, heeft de volgende functionaliteit:

– De machine moet eerst worden aangezet, waarbij de lijsten met kandidaten worden geladen.
– Vervolgens kan de machine verzoeken verwerken om op een bepaalde kandidaat te stemmen.
– Ook kan de uitslag worden opgevraagd, naar keuze per partij of per kandidaat.

Om het eenvoudig te houden, kent de machine maar drie partijen met elk vijf kandidaten, als getoond in figuur 1.5 (lijsten en kandidaten zijn gebaseerd op de Tweede Kamerverkiezingen van 2006).

1 CDA	2 PvdA	3 VVD
1 Jan Peter Balkenende Capelle aan den IJssel	1 Wouter Bos Amsterdam	1 Mark Rutte Den Haag
2 Maxime Verhagen Voorburg	2 Nebahat Albayrak Rotterdam	2 Rita Verdonk Nootdorp
3 Gerda Verburg Woerden	3 Aleid Wolfsen Amsterdam	3 Henk Kamp Zutphen
4 Joop Atsma Surhuisterveen	4 Jet Bussemaker Amsterdam	4 Brigitte van der Burg Bergschenhoek
5 Jack Biskop Roosendaal	5 Ton Heerts Apeldoorn	5 Laetitia Griffith Amsterdam

FIGUUR 1.5 De kandidatenlijsten volgens de stemmachine

2.2 EEN PROGRAMMA DAT STEMMEN UITBRENGT

Zie ook paragraaf 1.2

In een objectgeoriënteerde taal zoals Java is het mogelijk om een representatie van een stemmachine te construeren: een stuk programma dat precies beschrijft wat stemmachines kunnen en ook hoe ze dat moeten doen. Stemmachines behoren niet tot de eerder genoemde Java-bibliotheek; daar zitten weinig zaken in die zo direct corresponderen met dingen uit de buitenwereld. We hebben daarom zelf dat stuk programma ontwikkeld en het klaar voor gebruik toegevoegd aan de bouwstenen.

Wat we in deze paragraaf gaan doen, is een klein programma schrijven dat werkt met zo'n stemmachine. Het hart van dat programma bestaat uit een serie opdrachten, die uiteindelijk een voor een uitgevoerd zullen worden. Daar moet nog het een en ander omheen, maar daar kijken we later naar: we beginnen met de serie opdrachten.

Het eerste wat ons programma gaat doen, is een Stemmachine-object creëren. Er bestaat weliswaar al een stuk programma dat beschrijft hoe stemmachines in elkaar zitten, maar de stemmachine zelf is er nog niet. U kunt dat enigszins vergelijken met het bestellen van een product uit een catalogus: de catalogus beschrijft wat u zoal kunt bestellen, maar pas als u een bestelformulier invult en opstuurt, krijgt u producten uit die catalogus ook daadwerkelijk in handen. De eerste opdracht bestelt dus als het ware een nieuwe stemmachine.

We beginnen de serie opdrachten daarom als volgt:

Programma versie 1

```
machine = new Stemmachine();
```

Sleutelwoord

Het woord new staat hier vetgedrukt omdat het een zogenaamd *sleutelwoord* is, een woord dat in Java een speciale betekenis heeft (in dit geval geeft het aan dat er een object gecreëerd moet worden). Dat geldt ook voor andere vetgedrukte woorden die u nog tegenkomt. Met deze opdracht wordt een nieuw Stemmachine-object gecreëerd en daaraan wordt de naam machine gegeven.

Vervolgens komt er een rijtje opdrachten die eerst de machine aanzetten en dan stemmen uitbrengen op Wouter Bos, Gerda Verburg, Nebahat Albayrak, nogmaals Gerda Verburg en tot slot Henk Kamp.

Met de eerste opdracht mee ziet het codefragment er als volgt uit (de grijze opdrachten zijn nieuw):

Programma versie 2

```
machine = new Stemmachine();
machine.zetAan();
machine.stem("Wouter Bos");
machine.stem("Gerda Verburg");
machine.stem("Nebahat Albayrak");
machine.stem("Gerda Verburg");
machine.stem("Henk Kamp");
```

Alle nieuwe opdrachten hebben de vorm

```
object.doeIets(...);
```

Vóór de punt staat een object genoemd, in dit geval machine. Na de punt staat een verzoek aan dat object: iets wat het object moet doen. Achter een verzoek staan altijd haakjes. Soms heeft het object extra informatie nodig om aan het verzoek te kunnen voldoen. Bij het verzoek om te stemmen hoort bijvoorbeeld de naam van een kandidaat. Die extra informatie staat tussen de haakjes. Een Java-opdracht wordt ten slotte altijd afgesloten met een puntkomma.

Het vervolg is iets ingewikkelder. We willen namelijk de uitslag weten, en die tonen op het scherm. We breiden daartoe het codefragment met twee opdrachten uit:

Programma versie 3

```
machine = new Stemmachine();
machine.zetAan();
machine.stem("Wouter Bos");
machine.stem("Gerda Verburg");
machine.stem("Nebahat Albayrak");
machine.stem("Gerda Verburg");
machine.stem("Henk Kamp");
uitslag = machine.geefUitslagPerPartij();
System.out.println(uitslag);
```

De eerste nieuwe opdracht vraagt de uitslag op bij de machine. Die vraag staat rechts van het isgelijkteken, en heeft dezelfde vorm die we eerst zagen: het object wordt genoemd, dan komt er een punt, dan het verzoek met een haakjespaar erachter, en dan een puntkomma. Het verschil met de vorige opdrachten is dat de machine nu een antwoord geeft, dat we willen onthouden. We geven er daarom een naam aan: uitslag. We kunnen het effect van deze opdracht dus als volgt omschrijven: vraag aan de machine om de uitslag per partij en geef het antwoord de naam uitslag.

De tweede nieuwe opdracht toont de uitslag op het scherm. De vorm van deze opdracht is, door de twee punten die er in staan, enigszins verwarrend. In leereenheid 7 kijken we er in detail naar, tot dan mag u deze voor kennisgeving aannemen:

Printopdracht

```
System.out.println(tekst);
```

zet de genoemde *tekst* op het scherm. De uitslag die we hebben opgevraagd is zo'n tekst; die kunnen we dus met deze opdracht op het scherm zetten.

OPGAVE 1.2

Stel dat we ook een tweede stemmachine willen hebben, die we machine2 noemen. Formuleer opdrachten om deze tweede machine te creëren, hem aan te zetten, stemmen uit te brengen op Jan Peter Balkenende, Wouter Bos en Mark Rutte en de uitslag (uitslag2) van deze machine te tonen.

De reeks opdrachten uit versie 3 van ons programma in wording vormt nog geen volledig Java-programma. Er moet nog een aantal zaken bij.

Ten eerste bevat dit codefragment twee zelfgekozen namen: machine voor de Stemmachine, en uitslag voor het stuk tekst dat de uitslag bevat.

We hadden ook andere namen kunnen kiezen, bijvoorbeeld m en u of desnoods x en y. Dat zou het programma weliswaar een stuk minder leesbaar maken voor menselijke lezers (en dat is een ernstig bezwaar tegen een dergelijke keuze), maar de taaldefinitie verbiedt het niet (en dus maakt het voor de Java-compiler geen verschil).

String

Declaraties

Java eist echter wel dat we expliciet aangeven dat de naam machine wordt gebruikt voor een Stemmachine-object, en dat de naam uitslag wordt gebruikt voor een stuk tekst (in Java wordt dat een *String* genoemd). Zulke aankondigingen heten *declaraties*. Als we ze toevoegen, krijgen we het volgende:

Programma versie 4

```
Stemmachine machine;
String uitslag;

machine = new Stemmachine();
machine.zetAan();
machine.stem("Wouter Bos");
machine.stem("Gerda Verburg");
machine.stem("Nebahat Albayrak");
machine.stem("Gerda Verburg");
machine.stem("Henk Kamp");
uitslag = machine.geefUitslagPerPartij();
System.out.println(uitslag);
```

Een Java-programma kan tot slot niet zomaar beginnen met declaraties of opdrachten; er moet nog het een en ander omheen. Het voltooide programma ziet er als volgt uit.

Voltooid programma

```
import verkiezingen.Stemmachine;

public class Verkiezingsprogramma {
  public static void main(String[] args){
    Stemmachine machine;
    String uitslag;

    machine = new Stemmachine();
    machine.zetAan();
    machine.stem("Wouter Bos");
    machine.stem("Gerda Verburg");
    machine.stem("Nebahat Albayrak");
    machine.stem("Gerda Verburg");
    machine.stem("Henk Kamp");
    uitslag = machine.geefUitslagPerPartij();
    System.out.println(uitslag);
  }
}
```

We gaan hier nu nog niet te diep op in. De eerste regel kondigt het gebruik van Stemmachine-objecten aan. De tweede en derde regel mag u voorlopig voor kennisgeving aannemen: elk programma moet dergelijke regels bevatten. Het enige variabele element erin is de naam Verkiezingsprogramma: dat is de naam die we aan dit programma hebben gegeven. Aan het eind van de tweede en derde regel staat een openingsaccolade; de bijbehorende sluitaccolades staan op de laatste twee regels. Merk op dat we na zo'n accolade een stukje inspringen; dat bevordert de leesbaarheid van het programma. De betekenis van de rest (public class, public static void main, String[] args) zal later duidelijk worden.

2.3 HET PROGRAMMA COMPILEREN EN VERWERKEN

We willen dit programma nu laten compileren door de Java-compiler en de resulterende bytecode laten verwerken door de Java Virtual Machine (zie ook figuur 1.4). We beginnen met een voorbereidende opgave.

Wie een programma schrijft, moet nauwkeurig zijn. Het programma moet aan allerlei vormeisen voldoen, anders is het volgens de taaldefinitie geen correct Java-programma, en herkent de compiler het niet. Het programma wordt dan niet gecompileerd en kan dus ook niet worden verwerkt. Een vergeten puntkomma, een haakje te veel of te weinig of een verkeerd gespelde naam zijn voorbeelden van dergelijke vormfouten.

Belangrijk!

Java maakt onderscheid tussen hoofdletters en kleine letters. Een Stemmachine-object bijvoorbeeld begrijpt het verzoek stem, maar kan niets met het verzoek Stem.

OPGAVE 1.3

Vind zoveel mogelijk fouten in het volgende codefragment:

```
Stemmachine machine;
String uitslag;
machine = new stemmachine();
machine.stem(Wouter Bos);
machine.Stem("Gerda Verburg");
machine.stem("Maxime Verhagen")
machine.sten("Laetitia Griffith");
uitsla = machine.geef UitslagperPartij;
System.println("uitslag")
```

In de volgende opdracht gaan we het programma intypen, compileren en verwerken op de eenvoudigste manier: we gebruiken alleen Kladblok (Notepad) en een opdrachtprompt.

Studeer-aanwijzingen

– Om deze opdracht uit te kunnen voeren, moet de software bij deze cursus op uw computer geïnstalleerd zijn. Zie paragraaf 4 van de introductieleereenheid.
– Gebruik voor de volgende opdracht uitsluitend Kladblok en vooral *niet* Word of een vergelijkbare tekstverwerker. Zelfs de optie Opslaan als tekstbestand geeft vaak problemen; zo kunnen de aanhalingstekens die u intypt door Word worden veranderd.
– Als een onderdeel van de opdracht niet meteen lukt of het niet duidelijk is wat u precies moet doen, kijk dan in de terugkoppeling voor extra aanwijzingen.

OPDRACHT 1.4

a Ga in de map ProjectenOPiJ1, waar u de bouwstenen geplaatst hebt, naar de submap Le01Verkiezingen en open in Kladblok het bestand Verkiezingsprogramma.java. U zult zien dat het grijze deel dat in de laatste versie werd toegevoegd, al is ingetypt.
b Typ op de aangegeven plek de declaraties en opdrachten in (zie programma versie 4); u kunt de regel die begint met // en die de juiste plek aangeeft, gewoon laten staan. Het woord new hoeft niet vet. Sla het programma op als u alle benodigde regels hebt ingetypt.

c Start een venster Opdrachtprompt en ga met behulp van de DOS-opdracht cd naar de map waarin het programma staat.
d Geef de Java-compiler opdracht om het programma te compileren. Gebruik daartoe de volgende opdracht (let op de c van compiler in javac):

```
javac Verkiezingsprogramma.java
```

e Geef de Java Virtual Machine opdracht om het .class-bestand te verwerken. Gebruik daartoe de volgende opdracht:

```
java Verkiezingsprogramma
```

In het Java-programma dat u als bouwsteen kreeg, staat een regel

```
// Typ na deze regel de opdrachten in
```

Commentaar

Het zal duidelijk zijn dat deze regel geen Java-opdracht is, maar voor de menselijke lezer is bedoeld. Zulke regels worden *commentaar* genoemd. De Java-compiler slaat commentaar gewoon over, maar moet dan natuurlijk wel weten wat tot het commentaar en wat tot het programma zelf behoort. Het commentaar moet dus herkenbaar zijn aan zijn vorm. In Java wordt dat op twee manieren aangegeven:
– Na // is alles tot het einde van de regel commentaar.
– Na /* is alles commentaar, tot er ergens */ staat (dat kan vele regels verderop zijn).
De eerste vorm wordt dus gebruikt voor relatief korte commentaren; de tweede vorm voor langere.

Voorbeeld

In het volgende codefragment worden beide vormen gebruikt. De sterretjes aan het begin van regel 4-6 zijn niet verplicht, maar wel gebruikelijk (de regelnummers in de marge maken geen deel uit van het programma).

```
1  machine.zetAan();  // kort commentaar tot einde regel
   machine.stem("Wouter Bos");
   /*
    * Commentaar dat zich uitstrekt over verschillende
5   * regels. Meestal wordt aan het begin van elke regel
    * een sterretje geplaatst.
    */
   machine.stem("Mark Rutte");
   ...
```

In paragraaf 1 is gesproken over het belang van onderhoudbaarheid van software. Het toevoegen van zinvol commentaar kan daar een grote hulp bij zijn.

3 Objectoriëntatie

In de vorige paragrafen hebt u uw eerste Java-programma geschreven. Aan dit programma was meteen te zien dat Java de objectgeoriënteerde programmeerstijl volgt: de opdrachten deden verzoeken aan een Stemmachine-object (zetAan, stem, geefUitslagPerPartij).
In deze paragraaf gaan we iets dieper in op objectoriëntatie. Na het eerste blok wijden we daar nog een hele leereenheid aan (leereenheid 5).

De objectgeoriënteerde stijl wordt niet alleen gebruikt voor programmeren, maar ook bij de analyse en het ontwerp van informatiesystemen. Objectoriëntatie is een succes geworden omdat deze stijl nauw aansluit bij de manier waarop we over de werkelijkheid denken.

Voorbeelden

– De Open Universiteit gebruikt programmatuur die de studentenadministratie verzorgt. Als we gaan uitleggen hoe zo'n administratie in elkaar zit, dan hebben we het onder andere over studenten, cursussen en inschrijvingen. Het komt de begrijpelijkheid van een programma ten goede als ook daarin studenten, cursussen en inschrijvingen voorkomen. Bij objectoriëntatie is dat het geval: dat worden uiteindelijk allemaal objecten in het programma.
– In het voorbeeld uit de vorige paragraaf ging het over verkiezingen. Partijen, kandidaten en stemmachines zijn in dat kader voor de hand liggende objecten. We hebben tot nu toe alleen een Stemmachine-object geïntroduceerd, maar dat object gebruikt zelf (tot dusverre onzichtbaar voor ons) weer Partij-objecten en Kandidaat-objecten.

In deze paragraaf beperken we ons niet tot objectoriëntatie als programmeerstijl; de uitleg betreft objectoriëntatie als algemener concept.

3.1 WAT IS EEN OBJECT?

Object

Bij objectoriëntatie staat een *object* voor iets dat we willen onderscheiden als een betekenisvolle eenheid. Dat kan iets zijn uit de fysieke werkelijkheid zoals een auto, een hotel, een kamer, een televisietoestel of een mens, iets abstracts zoals een reservering, een vonnis, een schaakpartij of een contract, of iets digitaals zoals een knop, een venster of een menu op ons computerscherm.

Objecten worden gekenmerkt door een *toestand* en door *gedragsmogelijkheden*.

Voorbeeld

Als voorbeeld bekijken we een object uit de fysieke werkelijkheid: een auto.

Toestand: kenmerken

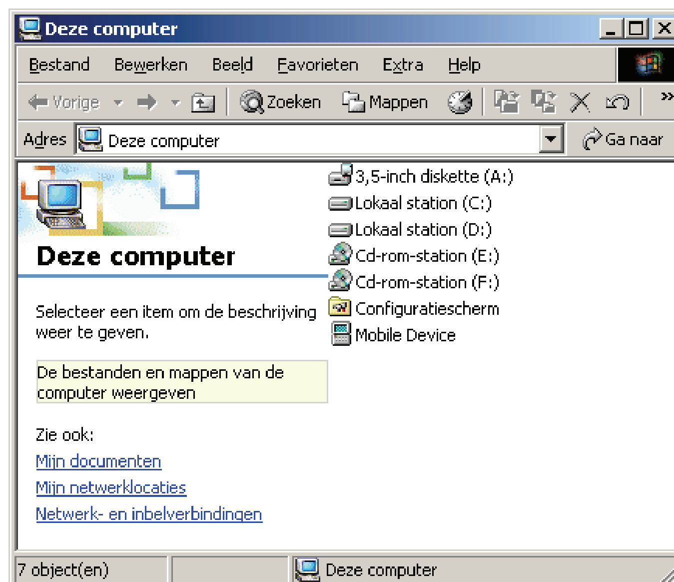
De *toestand* van de auto op een gegeven tijdstip wordt beschreven als het totaal van relevante *kenmerken* op dat tijdstip. Mogelijke relevante kenmerken van een auto zijn het merk, het type, de kleur, het kenteken, het aantal gereden kilometers, de snelheid, de rijrichting, de versnelling waarin de auto staat, welke lichten aan zijn, enzovoort. Merk op, dat sommige van deze kenmerken steeds hetzelfde blijven (merk, type, kleur) maar dat andere voortdurend veranderen (snelheid, versnelling, aantal gereden kilometers).

Gedrag: activiteiten

De *gedragsmogelijkheden* worden gevormd door alle *activiteiten* die het object op verzoek kan uitvoeren. Mogelijke activiteiten van een auto zijn: starten, rijden, naar een volgende versnelling overschakelen, ontkoppelen, remmen en lichten aan of uit doen. Een activiteit heeft invloed op de toestand: bij gas geven of remmen verandert bijvoorbeeld de snelheid van de auto.
We noemen dit niet voor niets activiteiten die het object *op verzoek* kan uitvoeren: een auto doet al deze dingen niet uit zichzelf, maar op verzoek van (in dit geval) de bestuurder.

Als een auto als object voorkomt in een computerprogramma, zijn we vaak maar in een deel van al deze kenmerken en gedragsmogelijkheden geïnteresseerd. In een verkeerssimulatie bijvoorbeeld zijn wel de snelheid en de rijrichting, maar niet het merk en de kleur van een auto van belang, dit in tegenstelling tot het systeem dat een autodealer gebruikt. De representatie van een object in een programma is dus altijd onvolledig: het is een abstractie.

OPGAVE 1.5
Ook een venster op een computerscherm kan beschouwd worden als een object. Noem eens enkele kenmerken en gedragsmogelijkheden van een venster op een computerscherm, zoals getoond in figuur 1.6.



FIGUUR 1.6 Een venster op een computerscherm

3.2 ATTRIBUTEN EN METHODEN

In een objectgeoriënteerd programma leggen we vast wat de kenmerken en de gedragsmogelijkheden zijn van de objecten.

Attributen

De kenmerken van een object worden vastgelegd in *attributen*, waarbij ieder attribuut overeenkomt met één kenmerk. Attributen zijn dus aanduidingen als kenteken of snelheid (van een auto) of naam en woonplaats (van een kandidaat). In een object hebben de attributen op elk moment een bepaalde *waarde*. De toestand van een object wordt bepaald door de waarden van alle attributen samen.

Gedrag: methoden

Het mogelijke *gedrag* van een object is vastgelegd in de *methoden* die het object kan uitvoeren. Elke *methode* beschrijft een bepaalde activiteit die het object (op verzoek) kan uitvoeren.

Voorbeeld

Figuur 1.7 toont als voorbeeld een schematische weergave van een Kandidaat-object uit het verkiezingenvoorbeeld.

Kandidaat	
naam	"Wouter Bos"
woonplaats	"Amsterdam"
aantalStemmen	24

methoden van **Kandidaat**
void stem() String getNaam() String getWoonplaats() int getAantalStemmen()

FIGUUR 1.7 Schematische weergave van een Kandidaat-object

Het object heeft drie attributen: naam, woonplaats en aantalStemmen. De waarden van die attributen voor dit object zijn respectievelijk "Wouter Bos", "Amsterdam" en 24.

Daarnaast staan de gedragsmogelijkheden ofwel de methoden van het object. Het object kan vier verzoeken verwerken. Met behulp van het verzoek stem wordt een stem uitgebracht op deze kandidaat. De andere drie zijn verzoeken om informatie, die dan ook een antwoord opleveren. Met behulp van de methode getNaam wordt de waarde van het attribuut naam opgevraagd, met behulp van de methode getWoonplaats wordt de waarde van het attribuut woonplaats opgevraagd, en met behulp van de methode getAantalStemmen ten slotte wordt de waarde van het attribuut aantalStemmen opgevraagd.

void

int

Het woord dat voor de methodenaam staat, geeft aan of het verzoek tot een antwoord leidt en zo ja, wat voor soort antwoord. Staat op die plek het woord *void* (Engels voor leeg), dan komt er geen antwoord. Staat er iets anders, dan komt er wel een antwoord; het woord geeft dan aan wat voor soort antwoord er gegeven zal worden. De verzoeken getNaam en getWoonplaats leveren als antwoord een stukje tekst (een String, zie ook paragraaf 2). Het verzoek getAantalStemmen levert als antwoord een geheel getal, dat in Java wordt aangeduid als een *int* (voor integer, het Engelse woord voor een geheel getal).

Opmerking

Het is gebruikelijk om methoden die de waarde van een attribuut opvragen, te laten beginnen met het woord get. Dat levert een mengsel van Engels en Nederlands, maar we houden ons toch aan deze conventie. Er zijn namelijk Java-tools die daarvan uitgaan.

OPGAVE 1.6

Teken een schematische weergave van een venster met als attributen titel, breedte en hoogte en methoden om het venster te sluiten, te minimaliseren en te maximaliseren, en om breedte en hoogte op te vragen. Verzin zelf geschikte waarden voor de attributen en geschikte namen voor de methoden.

De waarde van een attribuut kan een getal zijn (zoals 24 voor het aantal stemmen) of een String (zoals "Wouter Bos" voor de naam), maar de waarde kan ook een ander object zijn. Zo heeft de stemmachine uit de vorige paragraaf als attribuut een lijst van partijen, en een lijst is ook weer een object.

3.3 KLASSEN

In paragraaf 2 hebben we gezien dat het Verkiezingsprogramma bij het starten nog geen Stemmachine-object bevat; dat object werd in het programma zelf gecreëerd. Wel was er een beschrijving beschikbaar, een stuk programma dat precies aangeeft welke attributen en methoden stemmachines hebben en ook hoe ze deze methoden moeten uitvoeren.

Klasse

Een dergelijke algemene beschrijving heet een *klasse*. Je kunt een klasse opvatten als een blauwdruk voor een groep objecten, of (het beeld dat we in paragraaf 2 gebruikten) als een catalogusbeschrijving die aangeeft wat je krijgt als je een bepaald type object bestelt.

Er zijn twee redenen waarom in een programma klassen worden beschreven en niet direct objecten. De eerste is dat we meestal verschillende objecten nodig hebben van dezelfde soort. Het Verkiezingsprogramma bijvoorbeeld gebruikt in totaal 19 objecten. Een daarvan is de stemmachine. Bij het aanzetten (methode zetAan) maakt dit object zelf dan nog drie Partij-objecten en vijftien Kandidaat-objecten aan. Alle Kandidaat-objecten lijken op elkaar: ze hebben allemaal dezelfde attributen (naam, woonplaats, aantalStemmen) en dezelfde methoden (stem, getNaam, getWoonplaats, getAantalStemmen). Ook alle Partij-objecten hebben dezelfde attributen (een lijst met kandidaten) en dezelfde methoden (bijvoorbeeld een methode om een stem uit te brengen op een bepaalde kandidaat).

De tweede reden is, dat we bij het schrijven van een programma de objecten vaak nog niet kennen. Zo is het vooraf duidelijk dat een programma voor een studentenadministratie Student-objecten zal bevatten, met attributen zoals naam, adres en datum van inschrijving, en methoden om een adreswijziging uit te voeren, de student in of uit te schrijven enzovoort. Om welke studenten het gaat, is dan natuurlijk nog niet bekend.

Samenvattend

Een klasse specificeert een groep van objecten met overeenkomstige attributen en overeenkomstig gedrag. Objecten worden gecreëerd overeenkomstig die specificatie.

OPGAVE 1.7

Waarin *verschillen* de individuele objecten uit dezelfde klasse, dus bijvoorbeeld individuele Kandidaat-objecten?

Later, vanaf leereenheid 6, leert u om zelf klassendefinities in Java te schrijven. Voorlopig maken we echter alleen gebruik van de mogelijkheid om in een programma klassen te *gebruiken* die door andere programmeurs zijn geschreven.

Zo gebruiken de programma’s uit de vorige paragrafen de klassen Stemmachine, Partij en Kandidaat, die door ons zijn geschreven. Een programma moet wel aangeven welke bestaande klassen er in worden gebruikt; hiervoor dienen de importregels aan het begin. In het Verkiezingsprogramma stond bijvoorbeeld

```
import verkiezingen.Stemmachine;
```

Zonder deze regel herkennen de compiler en de JVM de aanduiding Stemmachine niet.

Hergebruik van bestaande klassen

De mogelijkheid om bestaande klassen te gebruiken vormt een van de aantrekkelijke kanten van objectoriëntatie. Java biedt standaard een enorme bibliotheek van klassen aan, waardoor je in een Java-programma heel gemakkelijk allerlei dingen kunt doen die je alleen met grote moeite zelf zou kunnen programmeren. In de loop van de cursus komt u met een klein deel van die klassen in aanraking.

Interface

Om een bestaande klasse te gebruiken, hoef je als programmeur niet te weten hoe die klasse in elkaar zit; je hoeft alleen te weten wat je ermee kunt: hoe je objecten van die klasse kunt creëren en welke verzoeken je daaraan kunt doen. We noemen dat de *interface* van de klasse.

In de bijlage bij deze leereenheid vindt u de interface van de drie bestaande klassen uit het verkiezingenvoorbeeld: Kandidaat, Partij en Stemmachine. In paragraaf 4 gaan we daar verder op in. Overigens blijft er, zelfs na paragraaf 4, nog een aantal taalelementen uit de bijlage onbesproken; in latere leereenheden worden die wel behandeld.

4 Programma's nader bekeken

Syntaxis

In deze paragraaf gaan we in op de algemene vorm van een programma en van de declaraties en opdrachten die daarin kunnen voorkomen. We geven daarbij steeds regels voor de *syntaxis*, ofwel de voorgeschreven vorm van de programmaonderdelen. We hebben al gezien dat Java daar strikte eisen aan stelt. Een programma dat niet aan deze eisen voldoet, kan niet gecompileerd en verwerkt worden.

In dit blok behandelen we maar een klein deel van de mogelijkheden van Java. Alle regels die we geven voor de syntaxis zijn dus voorlopig; in de volgende blokken zal blijken dat er meer mogelijk is dan we hier laten zien.

Afspraken

Bij het formuleren van de regels voor de syntaxis hebben we de volgende spelregels aangehouden:
– sleutelwoorden van Java staan altijd vetgedrukt (en niet schuin)
– woorden die rechtgedrukt staan, moet u letterlijk zo overnemen evenals punten, komma's, puntkomma's, haakjes en accolades (die staan nooit schuingedrukt, al is dat zeker bij een punt niet zo goed te zien)
– schuingedrukte woorden staan voor iets dat nog moet worden ingevuld. Uit de context zal blijken wat daar moet komen te staan.

Volledige regels voor de syntaxis van Java voor zover behandeld in deze cursus vindt u in bijlage 1 bij de cursus (deel 4).

4.1 APPLICATIES

Zie paragraaf 1.3

Applicaties

Er zijn twee soorten Java-programma's: applets, die kunnen worden ingebouwd in een webpagina, en 'gewone' programma's die los van webpagina's worden verwerkt. In leereenheid 14 bekijken we hoe u een applet kunt definiëren, maar verder houden we ons bezig met de gewone programma's. Ter onderscheid van applets heten dergelijke programma's in Java ook wel *applicaties*, en vanaf nu gebruiken we die term.

Voorlopig hebben de applicaties die we zullen schrijven de volgende vorm:

Java-applicatie

Voorlopige syntaxis

```
import packagenaam.Klassennaam;
import packagenaam.Klassennaam;
...

public class Programmanaam {
  public static void main(String[] args) {
    variabelendeclaraties

    opdrachten
  }
}
```

Package

Het programma begint met importopdrachten voor alle bibliotheekklassen die gebruikt worden. Dat kan een hele lijst zijn, maar het kunnen er ook nul zijn (als er helemaal geen bibliotheekklassen gebruikt worden). Voor de punt staat de naam van een *package*, een samenhangend deel van de bibliotheek (overeenkomend met een map in een filesysteem). Zo zitten de klassen Stemmachine, Partij en Kandidaat uit het verkiezingenvoorbeeld in een package verkiezingen. Achter de punt staat de naam van de klasse die gebruikt wordt.

Na de importopdrachten volgt een klassendefinitie:

```
public class Programmanaam {
...
}
```

We hebben in de vorige paragraaf al gezegd dat je in Java klassendefinities kunt schrijven, maar dat is te zwak uitgedrukt: *een Java-programma bestaat uit niets anders dan klassendefinities*. Ook een applicatie neemt dus de vorm aan van een klassendefinitie, al is dat hier een klasse zonder attributen en met maar een methode (main). Denk daar op dit moment vooral niet te diep over door!

Conventie

Zie verder paragraaf 4.2

De naam van deze klasse (ofwel de naam van het programma) kunt u zelf kiezen. We spreken af dat alle namen van klassen met een hoofdletter beginnen en dat we altijd een naam kiezen die aangeeft wat het programma doet (we noemen programma's dus geen X, Y of Programma).

Programma X altijd in bestand X.java

Verder *moet* het programma worden opgeslagen in een bestand met dezelfde naam, gevolgd door de extensie .java. We zagen dat al bij het Verkiezingsprogramma: dit was opgeslagen in het bestand Verkiezings-programma.java. Wijkt de naam van het bestand af, dan zal de Java-compiler het niet accepteren. Let hierbij op de hoofdletter: ook de bestandsnaam verkiezingsprogramma.java is voor de compiler niet acceptabel.

Tussen de openingsaccolade en de sluitaccolade van deze klasse staat de definitie van de enige methode van de klasse. Deze methode heeft altijd de naam *main*. U moet in uw programma's de eerste regel van deze definitie,

```
public static void main(String[] args) {
```

precies zo overnemen. Deze regel wordt de *signatuur* (of *kop*) van de methode main genoemd. De betekenis van de woorden static en String[] args leggen we voorlopig niet uit, maar ze mogen niet ontbreken.

De declaraties en opdrachten binnen de methode main bekijken we in de volgende deelparagrafen.

Commentaar en lay-out

Een programma kan op ieder punt commentaar bevatten, zoals al beschreven in paragraaf 2. Spaties, tabs en nieuwe regels hebben in het algemeen geen betekenis, alleen moet soms wel ergens een spatie, tab of nieuwe regel staan om twee elementen uit elkaar te houden. Zo mag u niet newStemmachine() schrijven in plaats van new Stemmachine().

OPGAVE 1.8

Schrijf een applicatie die niets anders doet dan "Hallo daar!" afdrukken.
U hoeft dit programma niet te compileren en verwerken.

De regels voor het verwerken van een applicatie zijn als volgt. Als aan de JVM wordt gevraagd om een bepaalde klasse te verwerken, dan zal deze op zoek gaan naar een methode main binnen deze klasse. Als die gevonden wordt, dan worden de opdrachten in die methode een voor een uitgevoerd. Het programma is klaar als de laatste opdracht is uitgevoerd (deze uitspraak zullen we in leereenheid 4, als we naar grafische interfaces gaan kijken, nuanceren). Als er geen methode met de naam main is, volgt een foutmelding.

De opdrachten binnen main maken meestal eerst wat objecten aan, en richten dan aan die objecten verzoeken om bepaalde methoden uit te voeren (we zagen dat in het Verkiezingsprogramma). Die methoden bestaan zelf ook weer uit een rij opdrachten, die op hun beurt weer andere objecten kunnen maken en kunnen verzoeken om andere methoden uit te voeren, enzovoort. Zo kan een methode main van een paar opdrachten toch tot een hele cascade van gebeurtenissen leiden.

4.2 DECLARATIES, VARIABELEN EN TYPEN

Na de signatuur van de methode main volgt een rij variabelendeclaraties (ook hier kunnen dat er nul zijn, zoals in het programma uit opgave 1.8).

De meest eenvoudige vorm (die we voorlopig gebruiken) van een dergelijke declaratie is

Variabelendeclaratie

Voorlopige syntaxis

type variabelennaam;

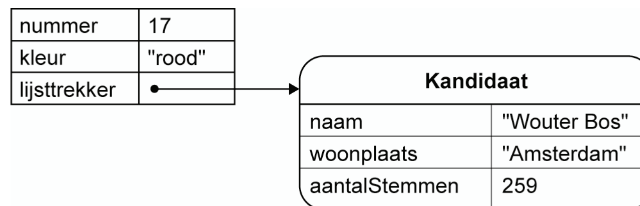
Voorbeelden hebben we al gezien in het Verkiezingsprogramma:

```
Stemmachine machine;
String uitslag;
```

We gaan nu wat dieper in op de begrippen variabele en type.

Variabele

We kunnen een *variabele* opvatten als een hokje in het computergeheugen, waarin een bepaalde waarde onthouden kan worden en dat aangeduid wordt met een bepaalde naam (Engels: identifier). Wat abstracter kunnen we ook zeggen dat een variabele een combinatie is van een naam en een bijbehorende waarde. Figuur 1.8 toont een schematische voorstelling van drie variabelen met hun waarden: een variabele nummer met de waarde 17, een variabele kleur met de waarde “rood”, en een variabele lijsttrekker met als waarde een Kandidaat-object.

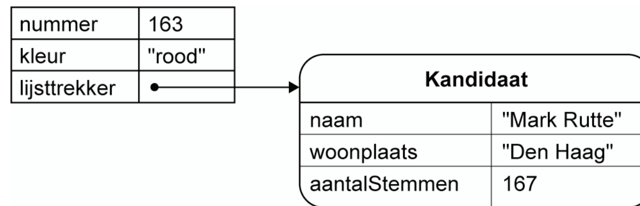


FIGUUR 1.8 Variabelen en waarden

Hoeveel variabelen bevat figuur 1.8?

We zeiden net dat het er drie waren, maar als u goed kijkt ziet u er zes: ook de attributen naam, woonplaats en aantalStemmen zijn variabelen met elk een bijbehorende waarde. We gaan daar voorlopig verder niet op in.

De waarde van een variabele kan tijdens de verwerking van een programma een of meer keer veranderd worden. De variabelen uit figuur 1.8 kunnen er na een tijdje uitzien als getoond in figuur 1.9. De waarden van nummer en lijsttrekker zijn veranderd, maar de waarde van kleur is hetzelfde gebleven.



FIGUUR 1.9 Variabelen kunnen nieuwe waarden krijgen

Namen: regels

In Java moeten alle namen (van variabelen, methoden of klassen) aan bepaalde *regels* voldoen. Ze moeten altijd beginnen met een letter (een hoofdletter of kleine letter), en mogen naast letters verder alleen cijfers en lage streepjes (_) bevatten. Sleutelwoorden mogen niet als namen worden gebruikt.

Namen: conventies

Verder is er een aantal afspraken of *conventies* waar alle Java-programmeurs zich aan houden. Zo is het gebruikelijk om namen van klassen altijd met een hoofdletter te laten beginnen, en namen van variabelen en methoden altijd met een kleine letter. Het is ook gebruikelijk om hoofdletters te schrijven op punten waar een nieuw woord begint, zoals in de naam aantalStemmen. Lage streepjes worden zelden gebruikt. Het is sterk aan te raden om ook die conventies te volgen; dit verhoogt de leesbaarheid van programma's.

OPGAVE 1.9
Bekijk de volgende namen: n, Stem_machine, brengStemUit, new, nummer3, 17deKandidaat, marleen@ou.
a Welke namen voldoen aan de regels voor namen in Java?
b Welke volgen ook de conventies voor variabelennamen?

Type

In een declaratie moet ook worden aangegeven van welk *type* de variabele zal zijn. Dit type bepaalt wat voor soort waarden de variabelen kan krijgen. Een variabele van het type int krijgt als waarden gehele getallen, een variabele van het type String krijgt als waarden stukjes tekst, en een variabele van het type Kandidaat krijgt als waarden Kandidaat-objecten.

Voorbeeld

De variabele nummer uit figuren 1.8 en 1.9 is van type int, kleur is van type String en lijsttrekker is van type Kandidaat. Declaraties van deze variabelen zien er als volgt uit:

```
int nummer;
String kleur;
Kandidaat lijsttrekker;
```

Primitief type

In deze declaraties is int (het type voor gehele getallen) een ander soort type dan String en Kandidaat. Java kent namelijk een klein aantal *primitieve typen*, waarvan de waarden geen objecten zijn. Dat betekent dat ze geen attributen hebben en geen methoden en dat ze niet met behulp van new gecreëerd hoeven te worden. Het type int is zo'n primitief type. Aan een waarde van een primitief type kunnen geen verzoeken worden gericht. Als n een variabele is van type int, zullen we dus nooit uitdrukkingen schrijven als n.plus(m) of n.abs(). We kunnen er wel gewoon mee rekenen; we schrijven bijvoorbeeld n + 3, of n / 4.

Referentietype

Alle andere typen in Java zijn klassen. Hun waarden zijn objecten die gecreëerd zijn met behulp van new en waar verzoeken aan gericht kunnen worden. String en Kandidaat zijn klassen, hun waarden zijn String-objecten (stukjes tekst) respectievelijk Kandidaat-objecten. Typen die horen bij een klasse worden ook wel *referentietypen* genoemd.

Merk op dat het verschil in de naamgeving te zien is: de namen van primitieve typen in Java beginnen met een kleine letter, die van referentietypen met een hoofdletter. Bovendien zijn de namen van primitieve typen sleutelwoorden in Java.

OPGAVE 1.10

In een verkeerssimulatie wordt onder meer gebruikgemaakt van de klassen Auto en Weggedeelte. Toon declaraties voor variabelen die achtereenvolgens moeten bevatten:
– het beschouwde weggedeelte
– het aantal auto's aan het begin van een bepaald tijdvak
– het aantal auto's aan het eind van het tijdvak
– de auto die als laatste het betreffende weggedeelte is binnengereden.

Belangrijk!

Tot slot een belangrijke opmerking: een declaratie is niets anders dan een aankondiging, een melding. Een declaratie introduceert een naam, maar creëert nooit een object. De gedeclareerde variabelen binnen de methode main krijgen niet vanzelf een waarde.

4.3 TOEKENNINGEN

De belangrijkste manier om een variabele een waarde te geven, is de toekenning. De algemene vorm van een toekenning is als volgt.

Toekenning

Voorlopige syntaxis

```
variabelennaam = waarde;
```

Let op!
Het isgelijkteken betekent: 'wordt gelijk aan'

Links staat de naam van de variabele die een waarde moet krijgen, rechts staat die waarde. Daartussen in staat een isgelijkteken, dat in Java gelezen moet worden als *wordt* gelijk aan en niet als *is* gelijk aan. Zoals altijd in Java eindigt de opdracht met een puntkomma. Eenvoudige voorbeelden van toekenningen zijn:

```
kleur = "zwart";
aantalStemmen = 0;
```

Expressie

Vaak moet de waarde aan de rechterkant eerst nog uitgerekend worden; er staat dan geen getal of een String, maar een zogenoemde *expressie* (iets dat na verwerking een waarde oplevert). Het Verkiezingsprogramma bevat daar twee voorbeelden van:

```
machine = new Stemmachine();
uitslag = machine.geefUitslagPerPartij();
```

De expressie new Stemmachine() wordt verwerkt en levert dan een nieuw Stemmachine-object; dit object wordt als waarde toegekend aan de variabele machine. Voor de tweede toekenning geldt hetzelfde: de methode geefUitslagPerPartij van het object machine berekent de uitslag per partij en geeft het resultaat als antwoord terug; dit resultaat wordt toegekend aan de variabele uitslag.

Een expressie kan ook een berekening zijn. We geven weer een paar voorbeelden (cda en pvda zijn variabelen van het type Partij; de andere variabelen zijn van het type int).

```
stemmenCda = cda.berekenAantalStemmen();
stemmenPvda = pvda.berekenAantalStemmen();
stemmenTotaal = stemmenCda + stemmenPvda;
stemmenGemiddeld = stemmenTotaal / 2;
```

In de eerste regel wordt bij het Partij-object cda het aantal stemmen opgevraagd en vervolgens toegekend aan de variabele stemmenCda; in de tweede regel gebeurt iets vergelijkbaars. In de derde regel worden de twee eerder opgevraagde aantallen bij elkaar opgeteld en toegekend aan de variabele stemmenTotaal. In de laatste regel wordt dit aantal door twee gedeeld en toegekend aan stemmenGemiddeld.

In expressies kunnen dus allerlei dingen staan: creatieopdrachten, verzoeken aan objecten die een antwoord teruggeven, getallen en variabelen, maar ook rekenkundige bewerkingen zoals optellen (+), aftrekken (–), vermenigvuldigen (*) en delen (/).

Bij het delen van twee int-waarden blijft het resultaat een waarde van type int; de waarde van 54 / 10 bijvoorbeeld is 5. We gaan hier in leereenheid 9 verder op in.

Eenvoudige waarden als “zwart” en 0 zijn feitelijk ook expressies. We kunnen de syntaxis van een toekenning daarom in zijn algemeenheid als volgt weergeven:

Toekenning

syntaxis *variabelennaam* = *expressie*;

OPGAVE 1.11

Welke waarden hebben de variabelen a, b, c en d na de volgende reeks toekenningen (a, b, c en d zijn van type int)?

```
a = 10;
b = 2 * a + 4;
c = a * b;
d = c / 12;
```

Niet iedere toekenning is toegestaan. Het type van de variabele moet namelijk overeenkomen met het type van de expressie.

Voorbeeld

Het volgende codefragment bevat verschillende fouten, die door de compiler zullen worden opgemerkt (de regelnummers in de marge maken geen deel uit van het programma).

```
1  int n;
   Partij p;
   Kandidaat k;

5  k = new Kandidaat("Jan Peter Balkenende", "Capelle aan den
                     IJssel");
   n = "Een tekstje";
8  p = new Stemmachine();
   n = 17 + k;
```

Ziet u zelf deze fouten?

De toekenning aan k in regel 5 is correct; de variabele k en de waarde van de expressie new Kandidaat("Jan Peter Balkenende", "Capelle aan den IJssel") zijn beide van type Kandidaat. De toekenning aan n in regel 6 is niet juist; n is van type int maar de waarde rechts is van type String. Ook de toekenning in regel 7 klopt niet; aan een variabele van het type Partij kan geen waarde van het type Stemmachine worden toegekend. In regel 8 is de expressie rechts van het isgelijkteken niet juist: we kunnen bij een waarde van het type int geen waarde van het type Kandidaat optellen. Wat overigens wel mag, is meer dan eens een waarde toekennen aan dezelfde variabele. De toekenningen aan n zijn incorrect, maar dat het er twee zijn is geen enkel probleem.

OPGAVE 1.12

Probeer de fouten te vinden in het volgende codefragment (kijk zowel naar declaraties als naar toekenningen).

```
int n;
int m;
int k;
String s;
Partij p;
Stemmachine s;

k = 12;
m = k;
m = n + 17;
p = (k + 14) * k;
s = p;
```

We bekijken nog een bijzonder geval. Stel dat we de waarde van een variabele aantalStemmen met 10 willen verhogen. We kunnen dat als volgt opschrijven:

```
aantalStemmen = aantalStemmen + 10;
```

Om deze toekenning te begrijpen, is het belangrijk om goed te beseffen dat de expressie aan de rechterkant altijd eerst wordt uitgerekend, en dat de resulterende waarde pas daarna aan de variabele die links staat wordt toegekend. Stel dat de variabele aantalStemmen vóór deze toekenning de waarde 60 heeft. De expressie aantalStemmen + 10 heeft dan dus de waarde 70. Die waarde wordt vervolgens toegekend aan aantalStemmen. Het resultaat is dan precies wat we wilden: de waarde van de variabele aantalStemmen is met 10 opgehoogd.

Dit voorbeeld illustreert een bezwaar tegen het gebruik in Java van het isgelijkteken voor de toekenning. Als dat isgelijkteken wordt gelezen als 'is gelijk aan' in plaats van als 'wordt gelijk aan', dan staat hierboven onzin: de waarde van aantalStemmen kan niet gelijk zijn aan diezelfde waarde plus tien (wiskundig gezien heeft de vergelijking aantalStemmen = aantalStemmen + 10 geen oplossing). Sommige programmeertalen vermijden dit soort verwarring door de toekenning te schrijven als :=, dus in dit geval aantalStemmen := aantalStemmen + 10. Helaas is daar in Java niet voor gekozen.

OPGAVE 1.13

Schrijf de volgende drie opdrachten als toekenningen (teller, a en b zijn variabelen van type int):

a verlaag de waarde van teller met 1
b verdubbel de waarde van a
c maak de waarde van b tien keer zo klein (ga er van uit dat de waarde van b een tiental is).

4.4 METHODEAANROEPEN

In de voorafgaande paragrafen hebben we steeds gesproken over verzoeken aan een object. Vanaf nu gebruiken we daarvoor steeds de term methodeaanroep. Bij een methodeaanroep hoort altijd een object; we spreken in dat verband van een methodeaanroep *op* een object.

Methodeaanroep

Syntaxis

Een *methodeaanroep* ziet er als volgt uit:

```
object.methodenaam(parameterlijst);
```

Het object zal voorlopig steeds aangeduid worden door een variabele. Dan volgt een punt, dan de naam van een methode met daarachter tussen haakjes de extra informatie die nodig is om de methodeaanroep uit te kunnen voeren. Die extra informatie bestaat uit een lijstje met waarden. Elke waarde die wordt meegegeven wordt een *parameter* genoemd (spreek uit: parámeter, met de klemtoon op de tweede a en twee toonloze e's). Als de methode geen parameters heeft, is deze lijst leeg; de haakjes moeten dan echter toch worden geschreven. De opdracht wordt uiteraard afgesloten met een puntkomma.

Parameter

Voorbeelden

Voorbeelden van methodeaanroepen hebben we al gezien in het Verkiezingsprogramma. Hier zijn er nog een paar (machine is van het type Stemmachine, groenLinks is van het type Partij, kand is van het type Kandidaat en vorigeLijst en laatsteNummer zijn van het type int):

```
machine.voegPartijToe(groenLinks);
kand = groenLinks.zoek("Femke Halsema");
machine.stemOpNummer(3, 4);
machine.stemOpNummer(vorigeLijst + 3, laatsteNummer - 1);
```

Al deze methodeaanroepen bevatten parameters. Het Partij-object groenLinks wordt als parameter meegegeven aan de aanroep voegPartijToe op het object machine. De String "Femke Halsema" wordt als parameter meegegeven aan de aanroep van de methode zoek op het object groenLinks. In het derde voorbeeld worden aan de aanroep twee parameters meegegeven, beide van type int. Het laatste voorbeeld toont dat de parameters ook expressies mogen zijn, die eerst nog uitgerekend moeten worden.

Al deze methodeaanroepen hebben de juiste vorm, maar dit betekent nog niet zonder meer dat ze ook correct zijn. De methodenaam moet namelijk ook een methode aanduiden van het betreffende object, en de parameterlijst moet met de definitie van die methode in overeenstemming zijn.

Welke methoden een bepaald object heeft en hoe de parameters eruit moeten zien, ligt vast in de interface van de bijbehorende klasse (herinner u dat de interface aangeeft wat u met objecten van die klasse kunt doen). De bijlage bij deze leereenheid bevat de interfaces van de klassen Kandidaat, Partij en Stemmachine. Deze interfaces bestaan uit twee delen: constructors (die bekijken we in de volgende paragraaf) en methoden.

We kijken als voorbeeld naar de specificatie van de methode voegPartijToe van de klasse Stemmachine:

```
public void voegPartijToe(Partij partij)
```
Voegt een Partij-object toe aan de lijst met Partij-objecten.

Signatuur van een methode

De tweede regel beschrijft in woorden wat de methode doet. Wat op de eerste regel staat, wordt de *signatuur* van de methode genoemd. Deze signatuur bestaat achtereenvolgens uit:
– het woord public, dat aangeeft dat de methode in elk programma mag worden gebruikt
– void voor methoden die bij aanroep geen antwoord teruggeven, of een typeaanduiding voor methoden die wel een antwoord teruggeven
– de naam van de methode
– een lijst met benodigde parameters, gescheiden door komma’s en tussen haakjes. Elke parameter bestaat uit een typeaanduiding gevolgd door een naam.

In dit voorbeeld is te zien dat er één parameter nodig is, en wel een object van type Partij. Daaruit kunnen we opmaken dat de methodeaanroep

```
machine.voegPartijToe(groenLinks);
```

correct is; de variabele groenLinks is immers van type Partij.

Ga na dat ook het tweede voorbeeld correct is:
```
kand = groenLinks.zoek("Femke Halsema");
```

De variabele groenLinks is van type Partij, en dus wordt hier een methode aangeroepen op een Partij-object. We vinden in de bijlage bij de methoden van de klasse Partij een methode met de volgende signatuur:

```
public Kandidaat zoek(String naam)
```

De methode zoek wil dus één parameter hebben, en wel een String. De aanroep geeft als antwoord bovendien een object van het type Kandidaat terug (dat zien we aan het woord Kandidaat na public). Aanroep en toekenning zijn dus beide correct: de aanroep bevat inderdaad een String als parameter, en het antwoord wordt toegekend aan een variabele van het juiste type.

Terugkeerwaarde

De waarde die door een methode wordt opgeleverd (het antwoord op een verzoek), duiden we meestal aan als de *terugkeerwaarde* (Engels: return value) van de methode.

OPGAVE 1.14

Controleer nu zelf dat ook de derde methodeaanroep correct is:

```
machine.stemOpNummer(3, 4);
```

Gebruik de bijlage bij deze leereenheid.

We hebben het woord parameter gebruikt zowel in verband met de signatuur van een methode, als in verband met de aanroep. Het gaat eigenlijk om twee verschillende zaken.

Formele parameters

De parameters in de signatuur bevatten een voorschrift: ze geven aan wat voor soort parameters er nodig zijn in de aanroep. Daarom staat er bij deze parameters ook een typeaanduiding. Ze hebben dezelfde vorm als declaraties (een type gevolgd door een naam), en dat zijn ze in feite ook. We noemen de parameters in de signatuur de *formele parameters* van de methode.

Actuele parameters

De parameters in de aanroep zijn expressies waarvan de waarden bij het verwerken van de methode aan de formele parameters worden toegekend. We noemen deze de *actuele parameters* van deze methodeaanroep.

We kunnen de regels voor de parameters in methodeaanroepen nu preciezer formuleren: de actuele parameters moeten overeenstemmen met de formele parameters, zowel in aantal als in type. Ze moeten ook precies in dezelfde volgorde staan. We kunnen ook zeggen: voor elke formele parameter in de signatuur moet in de aanroep een actuele parameter worden ingevuld van hetzelfde type.

In plaats van de termen formele parameter en actuele parameter worden in de Java-literatuur ook wel de termen parameter en argument gebruikt. De term parameter duidt dan dus altijd op een formele parameter.

OPGAVE 1.15

Elk van de volgende opdrachten bevat een fout; geef aan welke. De variabele machine is van het type Stemmachine, aantalStemmen is van het type int en vvd is van het type Partij.

```
machine.stemOpPartij("SGP");
machine.stemOpNummer("SGP", 1);
aantalStemmen = vvd.stem("Henk Kamp");
```

Verschillende klassen kunnen methoden hebben met dezelfde naam. Zo hebben de klassen Stemmachine en de klasse Partij allebei een methode met de signatuur

```
public void stem(String naam)
```

Welke methode wordt bedoeld, volgt uit het type object waarop de methode wordt aangeroepen.

4.5 CONSTRUCTORS EN CREATIEOPDRACHTEN

Creatie-expressie

Syntaxis

In het algemeen ziet een *creatie-expressie* er als volgt uit:

```
new Klassennaam ( parameterlijst )
```

Constructor

Initialisatie

Een creatie-expressie construeert een nieuw object van de genoemde klasse, en roept daar vervolgens een *constructor* op aan. Een klasse heeft namelijk naast methoden ook minstens één constructor (er kunnen er meer zijn, maar die mogelijkheid laten we voorlopig buiten beschouwing), die wordt aangeroepen op het zojuist gecreëerde object, en net als een methode parameters kan hebben. Een constructor verzorgt de *initialisatie* van het nieuwe object (heel vaak is dat de toekenning van geschikte beginwaarden aan de attributen van dat object).

Voorbeeld

Als voorbeeld bekijken we de constructor van de klasse Kandidaat:

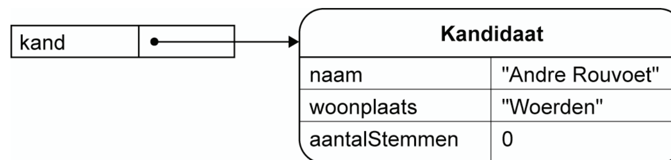
```
public Kandidaat(String naam, String woonplaats)
```
Geeft een nieuw Kandidaat-object de gegeven naam en woonplaats, en het aantal stemmen 0.

Merk op dat de signatuur van de constructor verschilt van die van een methode. De naam van een constructor is altijd gelijk aan die van de klasse (inclusief hoofdletter), en deze naam volgt meteen na het woord public (dus zonder void of andere typeaanduiding).

Een bijbehorende creatie-expressie (plus toekenning) is

```
kand = new Kandidaat("Andre Rouvoet", "Woerden");
```

Na deze toekenning heeft de variabele kand (mits, uiteraard, van type Kandidaat) als waarde een Kandidaat-object gekregen, als getoond in figuur 1.10.



FIGUUR 1.10 Kandidaat-object na constructie

De regels voor constructors zijn verder dezelfde als voor methoden: aantal en typen van parameters moeten overeenstemmen met die in de signatuur.

Belangrijk

Met één uitzondering worden nieuwe objecten *altijd* aangemaakt via een creatie-expressie. Zonder creatie-expressie is er dus geen nieuw object.

Uitzondering

Strings vormen deze uitzondering op de regel. Strings zijn objecten, maar het aanmaken van objecten van de klasse String komt zo vaak voor dat Java hiervoor een afkorting heeft. Voor iedere String-constante in de code(zoals “SP” in het codefragment hieronder), wordt vanzelf een nieuw String-object gecreëerd. We mogen dus bijvoorbeeld schrijven:

```
String partijnaam;
partijnaam = "SP";
```

in plaats van de langere vorm

```
String partijnaam;
partijnaam = new String("SP");
```

De tweede vorm is ook toegestaan, maar de eerste is veel gebruikelijker.

OPGAVE 1.16
Schrijf een opdracht die een nieuwe partij creëert met de naam “SP” en deze toekent aan een variabele sp van type Partij.

4.6 FOUTEN IN PROGRAMMA’S

Het compileren en verwerken van een programma gaat niet altijd goed. Er kunnen fouten optreden tijdens de *compilatie* en tijdens de *verwerking* van een programma. We gaan daar kort mee experimenteren door enkele fouten aan te brengen in het Verkiezingsprogramma.

Opdracht

Open het bestand Verkiezingsprogramma.java en breng daarin de volgende wijzigingen aan:
– verwijder de puntkomma in de declaratie van de variabele uitslag
– verwijder de aanhalingstekens rond “Wouter Bos”
– wijzig het eerste voorkomen van “Gerda Verburg” in “Gera Verburg” (tikfout in de naam)
– wijzig in de toekenning aan uitslag de methodenaam geefUitslagPerPartij in geefUitslagperPartij (hoofdletterfout).

Het programma ziet er nu als volgt uit:

```
1   import verkiezingen.Stemmachine;

    public class Verkiezingsprogramma {
      public static void main(String[] args) {
5
        Stemmachine machine;
        String uitslag

        machine = new Stemmachine();
10      machine.zetAan();
        machine.stem(Wouter Bos);
        machine.stem("Gera Verburg");
        machine.stem("Nebahat Albayrak");
        machine.stem("Gerda Verburg");
15      machine.stem("Henk Kamp");
        uitslag = machine.geefUitslagperPartij();
        System.out.println(uitslag);
      }
19  }
```

Compileer dit programma op dezelfde wijze als in opdracht 1.4. U krijgt dan foutmeldingen, als getoond in figuur 1.11

```
Verkiezingsprogramma.java:7: ';' expected
    String uitslag
                  ^
Verkiezingsprogramma.java:11: ')' expected
    machine.stem(Wouter Bos);
                       ^
Verkiezingsprogramma.java:11: not a statement
    machine.stem(Wouter Bos);
                        ^
Verkiezingsprogramma.java:11: ';' expected
    machine.stem(Wouter Bos);
                           ^
4 errors
```

FIGUUR 1.11 Foutmeldingen van de compiler

De compiler ziet meteen dat er in regel 7 een puntkomma ontbreekt, en raakt daar ook niet van in de war. Met de ontbrekende aanhalingstekens heeft de compiler veel meer moeite; er komen drie foutmeldingen op regel 11 maar geen van die drie geeft aan wat er echt aan de hand is. Dat is niet vreemd; we kunnen niet verwachten dat de compiler weet dat dit als String bedoeld is (het zou net zo goed een verkeerd gespelde variabelennaam kunnen zijn, zoiets als uit slag).

Opdracht

Voeg de puntkomma en de aanhalingstekens weer toe, en compileer het programma nogmaals.

Nu krijgt u de foutmelding uit figuur 1.12:

```
Verkiezingsprogramma.java:16: cannot find symbol
symbol  : method geefUitslagperPartij()
location: class verkiezingen.Stemmachine
    uitslag = machine.geefUitslagperPartij();
                     ^
```

FIGUUR 1.12 Foutmeldingen van de compiler (vervolg)

Pas nu ziet de compiler ook de tikfout in geefUitslagperPartij: deze naam wordt niet herkend. De compiler doet geen poging om te raden wat er wel had moeten staan.

Opdracht

Vervang de kleine p in de naam weer door een hoofdletter, compileer het programma (dat moet nu goed gaan) en laat het verwerken.

Exceptions

De foutieve spelling van Gerda Verburg is iets wat de compiler niet kan ontdekken, maar tijdens verwerking van de methode stem wordt wél geconstateerd dat de kandidaat Gera Verburg op geen enkele lijst staat. Als gevolg daarvan wordt een runtime-foutmelding gegeven; in Java heten deze *Exceptions* (zie figuur 1.13).

```
Exception in thread "main" verkiezingen.StemmachineException: Kandidaat Gera Ver
burg niet gevonden
        at verkiezingen.Stemmachine.stem(Stemmachine.java:59)
        at Verkiezingsprogramma.main(Verkiezingsprogramma.java:12)
```

FIGUUR 1.13 Een foutmelding tijdens verwerking (een Exception)

Exceptions kunnen worden veroorzaakt (opgegooid, heet dat in Java-jargon) door de Java Virtual Machine, maar ook door een stuk programma. Het eerste gebeurt bijvoorbeeld als er in een programma gedeeld wordt door 0. Het laatste is het geval in figuur 1.13; deze exception wordt opgegooid door de methode stem in de klasse Stemmachine. Vandaar de foutmelding in het Nederlands: die is door ons opgesteld bij het programmeren van die klasse.

Zeker in het begin zult u niet alle foutmeldingen onmiddellijk begrijpen. Wel hebt u altijd veel steun aan het feit dat gemeld wordt in welke regel de fout optreedt. Kijk goed naar die regel en eventueel ook naar de regels ervoor. Raadpleeg ook onze terugkoppelingen, waar altijd een correcte versie van het programma staat.

4.7 NETJES PROGRAMMEREN

In deze cursus willen we u meer leren dan programma's schrijven die werken; we willen u ook leren om goed gestructureerde, begrijpelijke programma's te schrijven die u over een jaar nog steeds zonder al te veel moeite kunt begrijpen. Hoe groter de programma's worden die u schrijft, hoe belangrijker dat is. Het helpt daarbij om vanaf het begin u te houden aan bepaalde gewoontes. We geven hier vast vier regels:

– Voorzie uw programma's waar nodig van commentaar.
– Zorg dat de programma's een overzichtelijke lay-out hebben. Spring in na iedere accolade openen. De ontwikkelomgeving die we gebruiken, zal daarbij helpen.
– Kies altijd betekenisvolle namen, voor programma's, voor variabelen en later ook voor klassen, attributen en methoden. Korte namen zoals n of m besparen typewerk, maar langere namen zijn vaak begrijpelijker.
– Houd u aan de naamgevingsconventies van de Java-gemeenschap, ook waar die niet door de taaldefinitie worden afgedwongen. De namen van klassen en dus ook van het programma beginnen altijd met een hoofdletter; de namen van variabelen en van methoden beginnen altijd met een kleine letter. Daarnaast gebruikt u een hoofdletter binnen de naam waar de Nederlandse taal een spatie zou eisen (bijvoorbeeld uitgebrachteStem, nieuwePartij, ...).

5 Een programma dat tegels tekent

Ter afsluiting van deze leereenheid vragen we u nog een ander programma te schrijven, dat gebruikmaakt van de klasse Tegel uit de package tegel.

Een tegel heeft een bepaalde afmeting en wordt op een standaardmanier opgebouwd uit twee motiefjes. Figuur 1.14 toont als voorbeeld twee Tegel-objecten.

```
                     ********************
                     oo****************oo
                     oooo************oooo
--------------       oooooo********oooooo
||----------||       oooooooo****oooooooo
||||------||||       oooooooo****oooooooo
||||||--||||||       oooooo********oooooo
||||------||||       oooo************oooo
||----------||       oo****************oo
--------------       ********************
```

FIGUUR 1.14 Twee tegels

De eerste heeft een afmeting van 7 bij 7, en is opgebouwd uit de motiefjes -- en ||. De tweede heeft een afmeting van 10 bij 10 en is opgebouwd uit de motiefjes ** en oo. Merk op dat de tegels vierkant ogen doordat elk motiefje uit twee tekens bestaat; de breedte in tekens is daardoor twee keer zo groot als de hoogte in regels. Het staat u overigens vrij om motiefjes te kiezen met meer of minder tekens (zolang beide motiefjes maar uit evenveel tekens bestaan).

De klasse Tegel heeft een constructor als volgt:

```
public Tegel(int afmeting, String motief1, String motief2)
```

Geeft een nieuwe tegel de gegeven afmeting en de gegeven motieven. Motief1 zit boven en onder; motief2 links en rechts. De motieven moeten uit evenveel karakters bestaan.

De klasse Tegel heeft verder twee methoden als volgt:

```
public void toon()
```

Toont de tegel op standaard uitvoer

```
public void toonErnaast(Tegel tegel2)
```

Toont deze tegel en tegel2 naast elkaar op standaard uitvoer, mits de afmetingen van de tegels gelijk zijn.

De term 'deze tegel' in de uitleg bij de methode toonErnaast verwijst naar het Tegel-object waarop de methode wordt aangeroepen.

OPDRACHT 1.17

a Net als bij het Verkiezingsprogramma, is er een bouwsteen met een programmaskelet. Ga in de map ProjectenOPiJ1 naar de submap Le01Tegels en open in Kladblok het bestand Tegelprogramma.java. Schrijf een programma dat een vloertje tekent van vier tegels, bijvoorbeeld zoals getoond in figuur 1.15. De tegels zijn twee aan twee gelijk. Kies zelf afmetingen en motieven.

b Kunt u het programma zo wijzigen dat het een vloertje tekent met vier gelijke tegels?

FIGUUR 1.15 Een vloertje van vier tegels

SAMENVATTING

Paragraaf 1

Een programma is een voorschrift dat door een computer kan worden verwerkt. Een programma kan gesteld zijn in machinetaal: de taal die door de computer direct verwerkt kan worden. Een programma kan ook gesteld zijn in een hogere programmeertaal; in dat geval moet het programma eerst nog vertaald worden naar machinetaal (compilatie). Hogere programmeertalen maken gebruik van verschillende programmeerstijlen.

Java behoort tot de groep talen die gebruikmaken van een objectgeoriënteerde programmeerstijl. Java dateert uit 1995 en werd populair vanwege de mogelijkheid om een bepaald type Java-programma's, applets genoemd, in een webpagina in te bouwen. Mede in verband daarmee wordt een Java-programma door de compiler niet meteen naar machinetaal vertaald, maar eerst naar bytecode. Een programma in bytecode (met de extensie .class) wordt verwerkt door een Java Virtual Machine.

Paragraaf 2

In deze paragraaf wordt een eerste, eenvoudig Java-programma geschreven, waarmee een stemmachine bediend wordt.

Paragraaf 3

De objectgeoriënteerde stijl beschouwt een programma als een beschrijving van een verzameling objecten die via samenwerking een bepaalde taak uitvoeren. Een object wordt gekenmerkt door een toestand, vastgelegd in de waarden van de attributen van het object, en door gedragsmogelijkheden, vastgelegd in methoden. Objecten worden beschreven door klassen. Een klasse specificeert de attributen en methoden van alle objecten van die klasse. Individuele objecten van dezelfde klasse verschillen in de waarden van hun attributen. Een van de aantrekkelijke kanten van objectoriëntatie is de mogelijkheid om klassen uit een bibliotheek te gebruiken. De programmeur hoeft dan alleen te weten welke methoden die klassen hebben en wat die methoden doen, maar niet hoe ze dat doen.

Paragraaf 4

In blok 1 van de cursus bekijken we Java-programma's met de volgende algemene vorm:

```
import packagenaam.Klassennaam;
import packagenaam.Klassennaam;
...

public class Programmanaam {

  public static void main(String[] args) {
    variabelendeclaraties

    opdrachten
  }
}
```

Een variabelendeclaratie heeft de volgende vorm:

```
type variabelennaam ;
```

Een type kan een primitief type zijn of een referentietype. Een referentietype is een klasse, de bijbehorende waarden zijn objecten. Waarden van primitieve typen zijn geen objecten, maar bijvoorbeeld gehele getallen (type int). Op dergelijke waarden kunnen we geen methoden aanroepen, en ze kunnen niet met behulp van new gecreëerd te worden.

Namen in Java beginnen met een letter en bevatten verder alleen letters, cijfers, en lage streepjes (_). Het is gebruikelijk om namen van klassen met een hoofdletter te laten beginnen, en namen van methoden en variabelen met een kleine letter. Een nieuw woord in een naam wordt aangeduid door daar een hoofdletter te gebruiken.

We hebben twee vormen van opdrachten bekeken: de toekenning en de methodeaanroep. Een toekenning heeft de algemene vorm

```
variabelennaam = expressie ;
```

Een expressie is een uitdrukking die na verwerking een waarde oplevert. In een expressie kunnen variabelennamen, rekenkundige bewerkingen en methodeaanroepen voorkomen. Een expressie kan ook een creatie-expressie zijn. De in een expressie genoemde variabelen moeten wel allemaal gedeclareerd zijn, en de waarde van de expressie moet van hetzelfde type zijn als de variabele waaraan deze waarde wordt toegekend.

Een methodeaanroep heeft de volgende vorm:

```
object.methodenaam ( parameterlijst ) ;
```

waarbij de parameterlijst bestaat uit een lijst expressies, gescheiden door komma's. Deze parameters heten de actuele parameters.
Een methodeaanroep is alleen correct als de interface van de klasse waartoe het object behoort inderdaad een methode met die naam en met een overeenkomstige signatuur bevat, dat wil zeggen dat aantal, typen en volgorde van de actuele parameters overeen moeten komen met die van de formele parameters in de signatuur.

Een creatie-expressie heeft de volgende vorm:

```
new Klassennaam ( parameterlijst )
```

Ook hier moet de klasse een constructor hebben met de juiste signatuur.

Programma's kunnen twee soorten fouten bevatten: compilatiefouten die door de compiler worden gesignaleerd, en verwerkingsfouten die bij verwerking tot uiting komen. Die laatste heten in Java Exceptions.

Paragraaf 5

In deze paragraaf wordt een tweede programma geschreven, dat tegels kan tekenen.

ZELFTOETS

1 Leg de betekenis uit van de volgende begrippen:
a bytecode, .class-bestand, Java-compiler en JVM
b signatuur van een methode, formele parameters, terugkeerwaarde en actuele parameters.

2 Welke van de volgende declaraties zijn correct? Wat is er mis met degene die niet correct zijn?
a `Int getal;`
b `Stemmachine mach3@amsterdam;`
c `String tekstje;`
d `Tegel t1`

3 Gegeven zijn de variabelen n en k van type int, uitslag van type String, p van type Partij en m van type Stemmachine. Welke van de volgende toekenningen zijn correct? Als een toekenning incorrect is, wat is er dan fout? Ga ervan uit dat alle variabelen een waarde hebben van het juiste type. Raadpleeg de bijlage!

a `n = m * 3 + k;`
b `p = new Partij();`
c `uitslag = p.geefUitslagPerPartij();`
d `n = m.stemOpNummer(n, k);`

4 Schrijf een serie Java-opdrachten waarin achtereenvolgens
– een Stemmachine-object wordt gecreëerd
– de gecreëerde stemmachine wordt aangezet
– Partij-objecten worden gecreëerd voor de partijen SP en CU (Socialistische Partij en ChristenUnie)
– Kandidaat-objecten voor Jan Marijnissen uit Oss en Andre Rouvoet uit Woerden worden gecreëerd en toegevoegd aan de zojuist gecreëerde Partij-objecten
– de Partij-objecten worden toegevoegd aan de stemmachine.
Toon ook de benodigde declaraties.
Raadpleeg de interfaces van deze klassen in de bijlage.

TERUGKOPPELING

1 Uitwerking van de opgaven

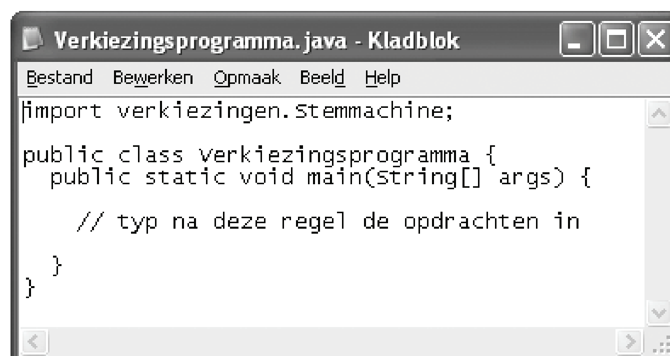
1.1 In Netscape moest een JVM worden ingebouwd.

1.2 De serie opdrachten ziet er als volgt uit:

```
machine2 = new Stemmachine();
machine2.zetAan();
machine2.stem("Jan Peter Balkenende");
machine2.stem("Wouter Bos");
machine2.stem("Mark Rutte");
uitslag2 = machine2.geefUitslagPerPartij();
System.out.println(uitslag2);
```

1.3 Het fragment bevat in totaal 12 fouten:
– In regel 3 is stemmachine met een kleine letter in plaats van een hoofdletter geschreven.
– In regel 4 ontbreken de aanhalingstekens rond de naam Wouter Bos.
– In regel 5 staat Stem ten onrechte met een hoofdletter geschreven.
– In regel 6 ontbreekt de puntkomma aan het eind.
– In regel 7 zit een tikfout ste*n* in plaats van ste*m*.
– Regel 8 bevat vier fouten: er staat uitsla in plaats van uitsla*g*, er staat een spatie tussen geef en Uitslag die daar niet mag staan, ‘per’ in geefUitslagperPartij is geschreven met een kleine letter in plaats van met een hoofdletter, en tot slot ontbreken de haakjes na geefUitslagPerPartij.
– Regel 9 ten slotte bevat drie fouten. Er ontbreekt een stuk: er staat System.println in plaats van System.*out*.println, rond uitslag staan aanhalingstekens die daar niet thuishoren en de puntkomma aan het eind ontbreekt.

1.4 a Kladblok vindt u in de lijst van programma’s bij de Bureau-accessoires. U kunt ook in de Verkenner rechtsklikken op het bestand Verkiezingsprogramma.java, kiezen voor Openen met... en uit de lijst programma’s die u dan krijgt, Kladblok kiezen.
Het bestand ziet er uit als in figuur 1.16



FIGUUR 1.16 De bouwsteen Verkiezingsprogramma.java in Kladblok

b Het scherm ziet er nu uit als getoond in figuur 1.17.

Verkiezingsprogramma.java - Kladblok

Bestand Bewerken Opmaak Beeld Help

```
import verkiezingen.Stemmachine;

public class Verkiezingsprogramma {
  public static void main(String[] args) {

    Stemmachine machine;
    String uitslag;

    machine = new Stemmachine();
    machine.zetAan();
    machine.stem("Wouter Bos");
    machine.stem("Gerda Verburg");
    machine.stem("Nebahat Albayrak");
    machine.stem("Gerda Verburg");
    machine.stem("Henk Kamp");
    uitslag = machine.geefUitslagPerPartij();
    System.out.println(uitslag);
  }
}
```

FIGUUR 1.17 Het voltooide programma in Kladblok

Controleer of u alles goed hebt ingetypt (haakjes, puntkomma's, aanhalingstekens).

c Kies in het menu Start voor Programma's, dan Bureauaccessoires en dan Opdrachtprompt. Ga in het venster eventueel eerst naar de juiste drive (bijvoorbeeld met het commando D: als de bouwstenen op de D-drive staan), en vervolgens met behulp van het commando cd *padnaam* naar de juiste map (bijvoorbeeld cd D:\ProjectenOPiJ1\Le01Verkiezingen).

d Als het goed gaat, krijgt u geen antwoord.

– Meldt de computer dat deze de opdracht javac niet herkent, wijzig dan uw executiepad zoals aangegeven in paragraaf 4.3 van de introductie tot de cursus.

– Meldt de computer dat de package verkiezingen niet bestaat en het symbool Stemmachine dus ook niet herkend wordt, plaats dan de bestanden verkiezingen.jar en tegels.jar in de juiste map zoals beschreven in paragraaf 4.4 van de introductieleereenheid.

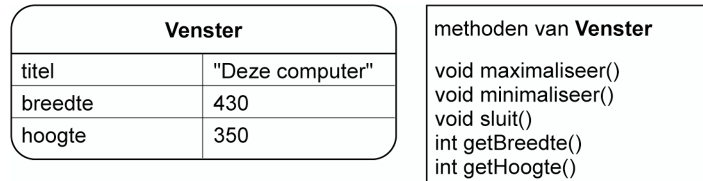
– Krijgt u andere meldingen, kijk dan nog één keer of u alles goed hebt ingetypt.

e Als dit goed gaat, krijgt u de volgende uitvoer:

```
CDA 1 stem
PvdA 2 stemmen
VVD 2 stemmen
```

1.5 Kenmerken van het venster zijn bijvoorbeeld de titel ("Deze computer" in figuur 1.6), de breedte en hoogte en de plaats op het scherm. Gedragsmogelijkheden zijn onder meer veranderen van de afmetingen, verplaatsen, sluiten, schermvullend maken en minimaliseren.

1.6 Zie figuur 1.18. Merk op, dat getBreedte en getHoogte verzoeken zijn waar een antwoord op komt, in tegenstelling tot maximaliseer, minimaliseer en sluit.



FIGUUR 1.18 Schematische weergave van een Vensterobject met drie attributen en drie methoden

1.7 Ze verschillen in de *waarden* van hun attributen.

1.8 Deze applicatie ziet er bijvoorbeeld als volgt uit:

```
public class HalloDaar {

  public static vold main(String[] args) {
    System.out.println("Hallo daar!");
  }
}
```

Er zijn geen importopdrachten. De naam HalloDaar kan ook anders zijn (als de filenaam overeenkomstig wordt aangepast).

1.9 a De namen n, Stem_machine, brengStemUit en nummer3 voldoen aan de regels. De namen new, 17deKandidaat en marleen@ou voldoen niet: new is een sleutelwoord, 17deKandidaat begint niet met een letter en marleen@ou bevat een teken dat niet in namen mag voorkomen.
b De namen n en nummer3 zijn het meest geschikt als variabelennamen. Stem_machine begint met een hoofdletter. De naam brengStemUit begint weliswaar met een kleine letter, maar is vanwege het karakter als verzoek geschikter als naam voor een methode.

1.10 Deze declaraties zien er bijvoorbeeld als volgt uit:

```
Weggedeelte huidigWeggedeelte;
int aantalAutosBegin;
int aantalAutosEind;
Auto laatstBinnengekomen;
```

Merk op dat we de namen zo kiezen, dat ze duidelijk maken wat de waarde betekent.

1.11 Zie de volgende tabel.

variabele	*expressie*	*expressie*	*waarde*
a	10	10	10
b	2 * a + 4	2 * 10 + 4	24
c	a * b	10 * 24	240
d	c / 12	240 / 12	20

1.12 We herhalen het codefragment, met regelnummers:

```
 1  int n;
    int m;
    int k;
    String s;
 5  Partij p;
    Stemmachine s;

    k = 12;
    m = k;
10  m = n + 17;
    p = (k + 14) * k;
    s = p;
```

De eerste fout zit in de declaraties: de variabele s wordt twee keer gedeclareerd en dat mag niet (de compiler moet weten of s nu een String of een Stemmachine als waarde moet krijgen).
De toekenning aan k in regel 8 is correct, en die aan m in regel 9 ook. Een expressie kan ook uit alleen een variabelennaam bestaan; na deze twee toekenningen hebben zowel k als m de waarde 12. De toekenning aan m in regel 10 is wat vorm betreft wel juist, maar toch gaat er iets mis: de variabele n heeft nooit een waarde gekregen en dus kan de expressie n+17 niet berekend worden. De compiler zal dit opmerken. In regel 11 wordt een waarde van type int toegekend aan een variabele van type Partij, en in regel 12 wordt een waarde van type Partij toegekend aan een variabele van type Stemmachine. Beide toekenningen bevatten dus een typefout (een fout in het type, geen tikfout).

1.13 Deze toekenningen zien er als volgt uit:

```
teller = teller - 1;
a = a * 2;   // of a = 2 * a, dat maakt niet uit
b = b / 10;
```

1.14 De aanroep luidt

```
machine.stemOpNummer(3, 4);
```

Bij de interface van de klasse Stemmachine vinden we een methode met de volgende signatuur:

```
public void stemOpNummer(int lijstnr, int kandidaatnr)
```

Deze methode wil dus twee parameters hebben, beide gehele getallen. Dat klopt precies met de aanroep, en dus is ook deze aanroep correct.

1.15 – De aanroep machine.stemOpPartij("SGP"); is incorrect omdat de klasse Stemmachine geen methode stemOpPartij heeft.
– De aanroep machine.stemOpNummer(vvd, 1); is incorrect omdat deze methode van Stemmachine twee int-waarden (een lijstnummer en een kandidaatnummer) als parameters vraagt, terwijl de eerste actuele parameter van type Partij is.
– De methodeaanroep in de opdracht aantalStemmen = vvd.stem("Henk Kamp"); is op zich juist. Alleen levert deze aanroep volgens de signatuur van de methode stem geen terugkeerwaarde op, en dus is er niets om aan de variabele toe te kennen.

1.16 In bijlage 1 zien we dat de constructor van de klasse Partij één parameter eist, en wel een String met de naam van de partij. We krijgen dus de volgende opdracht:

```
sp = new Partij("SP");
```

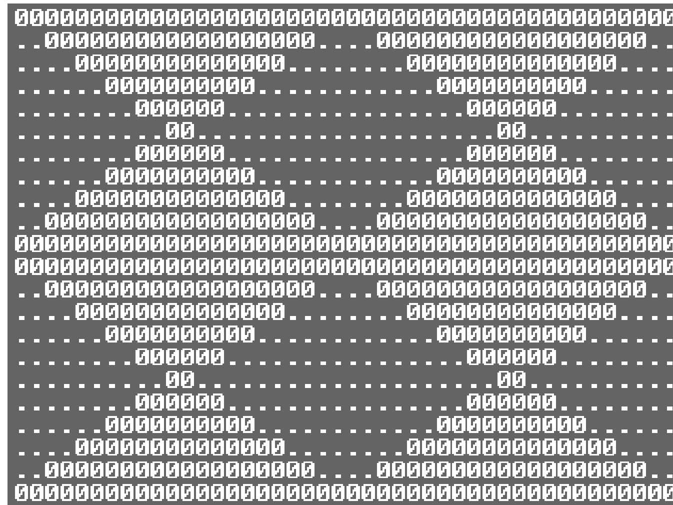
1.17 a We tonen het volledige programma:

```
import tegels.Tegel;
import tegels.Vloer;

public class Tegelprogramma {
  public static void main(String[] args) {

    // type na deze regel de opdrachten in

    Tegel tegel1;
    Tegel tegel2;

    tegel1 = new Tegel(11, "88", "..");
    tegel2 = new Tegel(11, "^^", "##");
    tegel1.toonErnaast(tegel2);
    tegel2.toonErnaast(tegel1);
  }
}
```

b Het programma ziet er nu als volgt uit:

```
import tegels.Tegel;

public class Tegelprogramma {
  public static void main(String[] args) {

    // type na deze regel de opdrachten in

    Tegel tegel;

    tegel = new Tegel(11, "00", "..");
    tegel.toonErnaast(tegel);
    tegel.toonErnaast(tegel);
  }
}
```

Merk op dat de methode toonErnaast wordt aangeroepen op tegel1, met diezelfde tegel als parameter. Daar is geen enkel bezwaar tegen. Figuur 1.19 toont de uitvoer.



FIGUUR 1.19 Een vloer van vier gelijke tegels

2 Uitwerking van de zelftoets

1 a Een Javaprogramma wordt door de Java-compiler vertaald naar bytecode. Het bestand waar deze bytecode in wordt opgeslagen, heeft de extensie .class. Dit .class-bestand wordt vervolgens verwerkt door de Java Virtual Machine (een programma, geen computer!). Zie ook figuur 1.4.

b De signatuur van een methode behoort tot de interface van een klasse en heeft de vorm

```
public void naam (formeleParameterlijst) of

public type naam (formeleParameterlijst)
```

In het eerste geval heeft de methode geen terugkeerwaarde (dat wil zeggen er komt na de aanroep geen antwoord terug), in het tweede geval is er een terugkeerwaarde van het aangegeven type. De lijst van formele parameters bestaat uit een rij *type1 naam1, type2 naam2, ...* enzovoort. Hiermee wordt aangegeven welke extra informatie meegegeven moet worden. In een aanroep van de methode op een object van de betreffende klasse staan de actuele parameters: expressies die bij verwerking een waarde opleveren. De actuele parameters moeten in aantal en type overeenkomen met de formele parameters, en in dezelfde volgorde staan.

2 Alleen de declaratie c is correct.
a Int wordt met een kleine letter geschreven en niet met een hoofdletter.
b mach3@amsterdam is geen correcte variabelennaam vanwege het @-symbool.
c Is correct.
d Er ontbreekt een puntkomma aan het eind.

3 Deze toekenningen zijn geen van alle correct.
a m is van het type Stemmachine en niet van het type int, en dus is de expressie m * 3 + k niet correct (als u hier overheen gekeken hebt, weet u meteen waarom u variabelen van het type Stemmachine beter niet m kunt noemen).
b De constructor van de klasse Partij heeft de volgende signatuur:

```
public Partij(String naam)
```

De aanroep bevat geen actuele parameter en is dus niet in overeenstemming met deze signatuur.
c De klasse Partij heeft geen methode geefUitslagPerPartij. De klasse Stemmachine wel, maar hier wordt een methode aangeroepen op een Partij-object.
d De methode stemOpNummer van de klasse Stemmachine heeft de volgende signatuur:

```
public void stemOpNummer(int lijstnr, int kandidaatnr)
```

Het sleutelwoord void geeft aan dat deze methode geen terugkeerwaarde heeft, en dus kan de aanroep niet in een toekenning staan.

4 Dit codefragment kan er bijvoorbeeld als volgt uitzien:

```
Stemmachine machine;
Partij sp;
Partij cu;
Kandidaat jan;
Kandidaat andre;

machine = new Stemmachine();
machine.zetAan();
sp = new Partij("SP");
cu = new Partij("CU");
jan = new Kandidaat("Jan Marijnissen", "Oss");
andre = new Kandidaat("Andre Rouvoet", "Woerden");
sp.voegKandidaatToe(jan);
cu.voegKandidaatToe(andre);
machine.voegPartijToe(sp);
machine.voegPartijToe(cu);
```

Een programmeur heeft echter een zekere vrijheid bij het formuleren van een programma. Om dat te verduidelijken, geven we nog een tweede versie, die ook correct is:

```
Stemmachine machine;
Partij nieuwePartij;

machine = new Stemmachine();
machine.zetAan();
nieuwePartij = new Partij("SP");
nieuwePartij.voegKandidaatToe(
               new Kandidaat("Jan Marijnissen", "Oss"));
machine.voegPartijToe(nieuwePartij);
nieuwePartij = new Partij("CU");
nieuwePartij.voegKandidaatToe(
               new Kandidaat("Andre Rouvoet", "Woerden"));
machine.voegPartijToe(nieuwePartij);
```

In deze versie is de volgorde van de opdrachten enigszins veranderd. De variabele nieuwePartij krijgt tweemaal een waarde toegekend. Het creëren van een nieuwe kandidaat en het toevoegen aan de partij gebeurt in één opdracht. Dat mag; een creatie-expressie is immers een expressie en mag als actuele parameter worden gebruikt.

Bijlage

De interfaces van de klassen in de package verkiezingen

1 De klasse Kandidaat

1.1 CONSTRUCTOR

```
public Kandidaat(String naam, String woonplaats)
```
Geeft een nieuw Kandidaat-object de gegeven naam en woonplaats, en het aantal stemmen 0

1.2 METHODEN

```
public int getAantalStemmen()
```
Levert het aantal op de kandidaat uitgebrachte stemmen

```
public String getNaam()
```
Levert de naam van de kandidaat

```
public String getWoonplaats()
```
Levert de woonplaats van de kandidaat

2 De klasse Partij

2.1 CONSTRUCTOR

```
public Partij(String naam)
```
Geeft een nieuw Partij-object de gegeven naam en een lege kandidatenlijst

2.2 METHODEN

```
public int berekenAantalStemmen()
```
Berekent het totaal aantal stemmen dat aan de partij is gegeven, dus de aantallen stemmen van alle kandidaten van de partij bij elkaar opgeteld

```
public String geefUitslagPerKandidaat()
```
Levert de uitslag per kandidaat als String bestaande uit regels van de vorm naam: aantal stemmen

```
public ArrayList<Kandidaat> getKandidaten()
```
Levert de lijst met kandidaten van de partij

```
public String getNaam()
```
Levert de naam van de partij

```
public void stem(String naam)
```
Brengt een stem uit op een kandidaat met gegeven naam, indien deze uniek op de lijst voorkomt

`public void stemOpNummer(int nummer)`
Brengt een stem uit op de kandidaat met het gegeven nummer

`public void voegKandidaatToe(Kandidaat k)`
Voegt het gegeven Kandidaat-object toe aan de lijst met Kandidaat-objecten

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

3 De klasse Stemmachine

3.1 CONSTRUCTORS

`public Stemmachine()`
Constructor van klasse Stemmachine

3.2 METHODEN

`public String geefUitslagPerKandidaat()`
Levert de uitslag per kandidaat als String bestaande uit regels van de vorm naam: aantal stemmen (alle kandidaten van alle partijen)

`public String geefUitslagPerPartij()`
Levert de uitslag per partij als String bestaande uit regels van de vorm naam: aantal stemmen

`public ArrayList<Partij> getPartijen()`
Levert de lijst met deelnemende partijen

`public void stem(String naam)`
Brengt een stem uit op een kandidaat met gegeven naam, indien deze op precies één van de lijsten staat

`public void stemNKeer(int n, int percentageLijsttrekker)`
Brengt *n* stemmen uit, verdeeld over alle partijen en alle kandidaten. Hoe lager het lijstnummer, hoe groter de kans dat een stem naar die lijst gaat. De tweede parameter geeft aan hoeveel procent van de stemmen naar de lijsttrekker gaat; de andere stemmen worden willekeurig verdeeld over de andere kandidaten op de lijst. Stemmen op een lege lijst gaan verloren.

`public void stemOpNummer(int lijstnr, int kandidaatnr)`
Brengt een stem uit op de kandidaat met het gegeven nummer van de lijst met het gegeven nummer

`public void voegPartijToe(Partij partij)`
Voegt een Partij-object toe aan de lijst met Partij-objecten

`public void zetAan()`
Initialiseert de lijst met partijen en voegt aan elke partij kandidaten toe