sheets Programmeren 1 — Java

college 4, Walter Kosters

De sheets zijn gebaseerd op met name hoofdstuk 8 van:
D. Bell en M. Parr, Java voor studenten,
Prentice Hall, 2002

`http://www.liacs.nl/home/kosters/java/`

Java — **functies**

Kort samengevat: Java kent functies zonder returnwaarde (dat zijn `void` functies; aanroepvoorbeeld: `schrijf ( );`) en functies die iets, bijvoorbeeld een `int`, teruggeven:

```
public void schrijf ( ) {
  System.out.println ("Tekst op scherm");
}  // schrijf

public int som (int x, int y) {
  return x + y;
}  // som
```

Bij de tweede functie hebben we twee parameters: `x` en `y`. Aanroepvoorbeeld: `t = som(y,12);`.

Als er `public` voor staat, kan "iedereen" de functie gebruiken; staat er `private` voor, dan is het gebruik "beperkt" — later meer hierover. Voorlopig gebruiken we zoveel mogelijk `public`.

Java — **parameters**

Parameters worden in Java *call by value* doorgegeven:

```
public int som (int x, int y) {
  int a = x;
  x = 0;
  return a + y;
}  // som
```

rekent nog steeds de som van `x` en `y` uit. Maar bij de aanroep `t = som (x,y);` bevat `x` na afloop nog steeds zijn oorspronkelijke waarde! Eigenlijk gaan alleen de waardes (values) van `x` en `y` naar de functie toe.

De variabele `a` heet wel een *locale variabele*, en is alleen binnen de functie `som` geldig.

De buiten de functies aangemaakte (meestal `private`) variabelen kun je "overal" gebruiken (bijvoorbeeld de `Scrollbar trekker`).

Java — **globale structuur**

De globale stuctuur van een Java-applet (met *OOP*, ObjectgeOriënteerd Programmeren) is:

```
import-regels ...
public class Iets extends Applet
    implements AdjustmentListener,
               ActionListener {
  private variabelen ...
  public void init ( ) { ... }
  public void paint (Graphics g) { ... }
  public void actionPerformed ( ... ) { ... }
  public void adjustmentValueChanged ( ... ) {
    ... }
  public int eigen1 ( ... ) { ... }
  public void eigen2 ( ... ) { ... }
  ...
}  // Iets
```

Let er op dat sommige functies verplicht zijn, bijvoorbeeld `adjustmentValueChanged` (die de gebruikers-acties opvangt) als je "gebruik maakt" van een `AdjustmentListener`.

Java — **while**

Het volgende stukje Java bevat een herhaling in de vorm van een “while-loop”:

```
int teller = 0, x = 10;
while ( teller < 8 ) {
  g.drawString ("*",x,20);
  x = x + 10;
  teller++;
}  // while
```

Het programma tekent 8 sterretjes naast elkaar. Achtereenvolgens heeft `teller` de waarde 0, 1, 2, 3, 4, 5, 6, 7 en 8; als `teller` 8 is wordt de “body” van de loop niet meer uitgevoerd.

Als er in plaats van `<` had gestaan `!=` (ongelijk) had het programma hetzelfde gedaan, maar was het gevaarlijk geweest: begin maar eens met `teller = 20`!

Java — **for**

Hetzelfde wordt bereikt met de volgende “for-loop”:

```
int teller;
int x = 10;
for ( teller = 0; teller < 8; teller++ ) {
  g.drawString ("*",x,20);
  x = x + 10;
}  // for
```

Het programma tekent wederom 8 sterretjes naast elkaar.

Bij een for-loop is het aantal herhalingen van te voren bekend.

Java — **stopt while-loop?**

Een while-loop wordt vaak gebruikt in situaties waarin je niet weet hoe vaak de de herhaling zal plaatsvinden: je vraagt bijvoorbeeld aan een gebruiker om een getal kleiner dan 100, en herhaalt dat net zolang totdat hij/zij een getal geeft dat aan de eisen voldoet.

Van het programma

```
while ( x != 1 ) // met int x
  if ( x % 2 == 0 )  // x is even
    x = x / 2;
  else  // x is oneven
    x = 3 * x + 1;
```

is het nog onbekend of het voor iedere waarde van `x` stopt! Als je begint met `x` gelijk aan 11, krijg je achtereenvolgens: 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.

Java — **voorbeeld**

We breiden ons “Scrollbar-voorbeeld” als volgt uit:

```
// waarde is de waarde van de Scrollbar
public void paint (Graphics g) {
  int tel = 0;
  int x = 20, y = 20;
  for ( tel = 1; tel <= waarde; tel++ ) {
    g.drawLine (x,y,x+20,y);
    g.drawLine (x+20,y,x+20,y+20);
    x = x + 20;
    y = y + 20;
  }  // for
}  // paint
```

Het programma tekent nu een trapje ter grootte van de door de Scrollbar aangegeven waarde.

Java — **do...while**

Soms wil je iets herhalen (minstens één keer) totdat . . . , bijvoorbeeld: vraag iets aan een gebruiker, totdat het antwoord goed genoeg is. Of hoeveel vakjes heb je nodig om minstens 100 rijstkorrels te krijgen, als ieder nieuw vakje er twee keer zoveel bevat als het vorige?

```
public void paint (Graphics g) {
  int y = 20; int vakjes = 0;
  int rijst = 1; int totaal = 0;
  do {
    vakjes++;
    g.drawString ("Vakje " + vakje + " heeft "
                  + rijst + " korrels",10,y);
    totaal = totaal + rijst;
    rijst = rijst * 2;
    y = y + 20;
  } while ( totaal < 100 );
  g.drawString ("Nodig: " + vakjes
                + " vakjes",10,y+20);
}  // paint
```

Java — **geneste loops**

Loops binnen loops (binnen loops . . .) komen ook vaak voor, zogeheten geneste loops:

```
int i, j; int x = 10, y = 10;
for ( i = 0; i < 4; i++ ) {
  g.drawString (i + ":",10,y);
  x = 30;
  for ( j = 0; j < 5; j++ ) {
    g.drawString (" " + (i+j),x,y);
    // let op de " " + die er voor zorgt
    // dat het getal i+j een string wordt
    x = x + 20;
  }  // for j
  y = y + 20;
}  // for i
```

levert op:

```
0: 0 1 2 3 4
1: 1 2 3 4 5
2: 2 3 4 5 6
3: 3 4 5 6 7
```

Java — **voorbeeld: snelle stuiterbal**

```
private int x = 70, xDelta = 7;
private int y = 20, yDelta = 2;

public void paint (Graphics g) {
  int n;
  Color achtergrond;
  g.drawRect (0,0,100,100);
  for ( n = 1; n <= 1000; n++ ) {
    achtergrond = getBackground ( );
    g.setColor (achtergrond);
    g.fillOval (x,y,10,10);  // "wis" oude bal
    if ( x <= 10 || x >= 90 ) xDelta = -xDelta;
    if ( y <= 10 || y >= 90 ) yDelta = -yDelta;
    x = x + xDelta;
    y = y + yDelta;
    g.setColor (Color.red);
    g.fillOval (x,y,10,10);  // en maak nieuwe
  }  // for
}  // paint
```

Merk op dat het "object" `g` steeds nieuwe opdrachten krijgt toebedeeld.