Inleiding Programmeren
College 2

Universiteit van Amsterdam

Robin de Vries

6 september 2017

Practicum

Hoe gaat het met het practicum?

Wat hebben jullie onthouden van het vorige college?

Waar hebben we het over gehad?

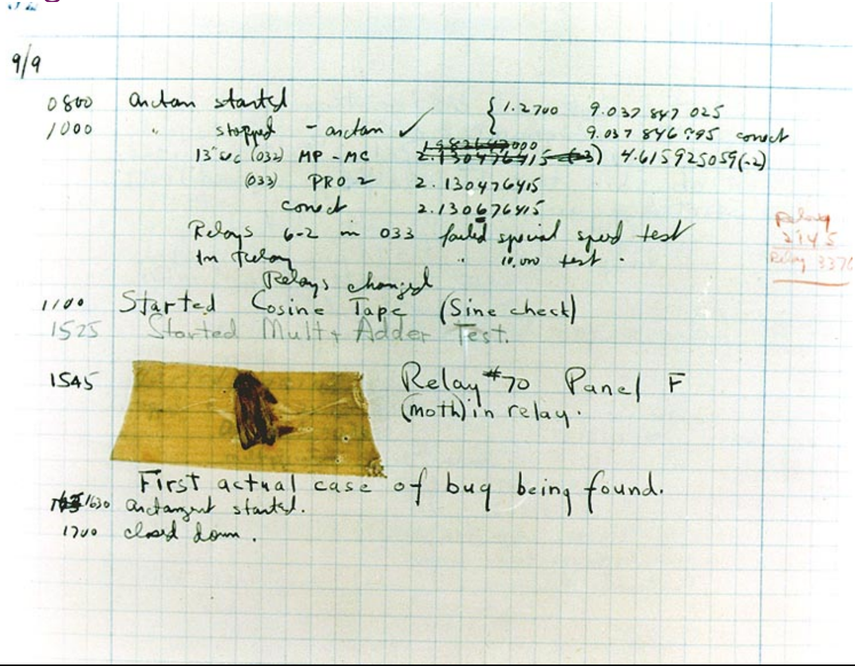
Laten we testen hoeveel jullie onthouden hebben!

Pak je mobieltje en ga naar `kahoot.it`

(Deze vragen komen achteraf **niet** op Canvas, hoorcollege-only!)

Inhoud

Bugs


9/9
0800 Antan started
1000 stopped - antan
1.2700 9.037 847 025
9.037 846 995 conect
13" uc (032) MP - MC
4.615925059(-2)
(033) PRO 2 2.130476415
conect 2.130676415
Relays 6-2 in 033 failed special speed test
In relay 10,000 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1630 Antangent started.
1700 closed down.

Fout bij compileren (1)

```java
/* Fout1.java */
import java.util.Scanner;
public class Fout1 {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.println(scanner.nextInt());
   }
]
```

```
$ javac Fout1.java
Fout1.java:8: error: illegal start of type
]
^
Fout1.java:8: error: reached end of file while parsing
]
 ^
Fout1.java:9: error: reached end of file while parsing
3 errors
```

Fouten bij compileren (2)

```java
/* Fout2.java */
import java.util.Scanner;
public class Fout1 {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.println(scanner.nextInt());
   }
}
```

```
$ javac Fout2.java
Fout2.java:3: error: class Fout1 is public, should be declared
in a file named Fout1.java
public class Fout1 {
       ^
1 error
```

Fouten bij compileren (3)

```java
/* Fout3.java */
import java.util.Scanner;
public class Fout3 {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.println(scanner.nextInt())
   }
}
```

```
$ javac Fout3.java
Fout3.java:6: error: ';' expected
      System.out.println(scanner.nextInt())
                                           ^
1 error
```

Fouten bij compileren (4)

```java
/* Fout4.java */
public class Fout4 {
   public static void main(String[] args) {
      int n;
      n = n + 1;
      System.out.println(n);
   }
}
```

```
$ javac Fout4.java
Fout4.java:5: error: variable n might not have been initialized
      n = n + 1;
          ^
1 error
```

Fouten bij compileren (5)

```java
/* Fout5.java */
public class Fout5 {
    public static void main(String[] args) {
        Random random = new Random();
        System.out.println(random.nextInt());
    }
}
```

```
$ javac Fout5.java
Fout5.java:4: error: cannot find symbol
        Random random = new Random();
        ^
  symbol:   class Random
  location: class Fout5
Fout5.java:4: error: cannot find symbol
        Random random = new Random();
                            ^
  symbol:   class Random
  location: class Fout5
2 errors
```

Inhoud

Loops

We onderscheiden op dit moment drie soorten loops:

- For-loop: als je van te voren **exact** weet hoeveel iteraties je hebt.
- While-loop: als je niet van te voren weet hoeveel iteraties je hebt, maar mogelijk **geen**.
- Do-while-loop: als je niet van te voren weet hoeveel iteraties je hebt, maar **minstens één**.

Kun je voor ieder type een loop een situatie bedenken?

De for-loop revisited

Met onderstaande code doen we exact 10 iteraties van de code:

```
for(int i = 0; i < 10; i++)
{
    System.out.println("Getal is: " + i);
}
```

`i++` is een verkorte schrijfwijze van `i = i + 1`, later meer daar over.

De while-loop

De code in de `while-loop` wordt uitgevoerd, zolang de evaluatie van de conditie **true** oplevert.

```
while(conditie) {
  // code die uitgevoerd wordt, zolang conditie == waar
}
```

De while-loop (2)

```
int count = 1;
while(count < 11) {
    System.out.println("Teller: " + count);
    count++;
}
```

Aan welke for-loop is dit equivalent?

De do-while-loop

De code in de `do-while-loop` wordt altijd minstens één maal uitgevoerd, en vervolgens net zolang herhaald zolang de evaluatie van de conditie **true** oplevert.

```
do {
 // code
} while(conditie);


// code die uitgevoerd wordt, zodra conditie != waar
```

Kunnen we een eenvoudig programmaatje verzinnen dat hier gebruik van maakt?

Som van invoer met do-while

```
Scanner scanner = new Scanner(System.in);
int som = 0;
int getal = 0;

do {
    System.out.println("Geef een geheel getal, 0 is stoppen:");
    getal = scanner.nextInt();
    som = som + getal;
} while(getal != 0);

System.out.println("Som van de getallen: " + som);
```

Inhoud

Functies

Een **functie** is een stuk code zelfstandige code, met eigen variabelen, die elders in een programma kan worden aangeroepen.

Indien een functie is gedefinieerd binnen een `class`, wordt het een **methode** genoemd. In Java bestaan dus, theoretisch gezien, geen functies, enkel methoden.

Een functie kan meerdere variabelen als argumenten meekrijgen, deze variabelen worden de **parameters** van de functie genoemd.

De aanroep van de functie wordt onderbroken of beëindigd met het commando `return`.

Aan het einde van de functieaanroep kan een functie een waarde teruggeven, de **return value**.

Functie definieren

Structuur:

```
modifiers returnType naam(parameters)
{
    // .. code
    return a; // a = variabele van het type returnType
}
```

Enkele aandachtspunten:

- Indien een functie geen waarde teruggeeft, is het return type: **void**.
- Een aanroep vanuit een `static` functie naar een andere functie van dezelfde klasse, moet altijd naar een andere `static` functie zijn[1].

[1] Later meer hier over.

Functie definieren (2)

```
public class FunctieProgramma
{

     public static void main(String[] args)
     {
        int som = telOp(3, 4);
        System.out.println(som);
     }

     static int telOp(int a, int b)
     {
         return a + b;
     }
}
```

Functie definieren (3)

Een functie zonder return value heeft als type **void**.

```
public class FunctieProgrammaVoid
{

     public static void main(String[] args)
     {
        telOpEnPrint(3, 4);
     }

     static void telOpEnPrint(int a, int b)
     {
        int som = a + b;

        System.out.println("De som van " + a + " en " + b + " is " + som);

        return;
     }
}
```

Early returns

Het is ook mogelijk om de uitvoer van een functie vroegtijdig af te breken door de **return** niet als laatste regel te plaatsen.

Kunnen we een priemgetal-tester schrijven?

Priemgetal tester

```
public static boolean isPriemOnderHonderd(int getal)
{
    if(getal < 2 || getal > 100)
        return false;

    int max = Math.min(getal - 1, 10);

    for(int i = 2; i <= max; i++)
    {
        if(getal % i == 0)
            return false;
    }

    return true;
}
```

Functies: waarom en wanneer?

Wanneer moet je code in een eigen functie stoppen?

- Als je merkt dat er verschillende dingen door elkaar lopen
- Als je code aan het herhalen bent
- Als een stuk code onafhankelijk is van de boven en onderstaande code

Wat levert dit voor voordelen op?

- Je code wordt overzichtelijker, DRY, modulair
- En dus...
- ... minder foutgevoelig!

Inhoud

Scanner

Vorig college hebben we al gezien hoe een integer in te lezen via de `Scanner` klasse.

```
import java.util.*;

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

        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter an integer: ");
        int intValue = scanner.nextInt();

        System.out.println("You entered integer: " + intValue);
    }
}
```

next() en nextDouble()

Naast de `nextInt()`-methode hebben we ook `next()` en `nextDouble()` om de volgende **String** of **Double** in te lezen.

Foutafhandeling

Echter crashen de programma's bij een foutieve invoer.

```
Scanner scanner = new Scanner(System.in);
System.out.println("Wat is mijn getal?");
int mijnGetal = scanner.nextInt();
System.out.println("Mijn getal: " + mijnGetal);
```

```
Wat is mijn getal?
a
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:864)
        at java.util.Scanner.next(Scanner.java:1485)
        at java.util.Scanner.nextInt(Scanner.java:2117)
        at java.util.Scanner.nextInt(Scanner.java:2076)
        at MijnInvoer.main(MijnInvoer.java:9)
```

Dit kunnen we afhandelen met `hasNextInt()`.

Foutafhandeling (2)

```
Scanner scanner = new Scanner(System.in);
System.out.println("Voer een geheel getal in:");

if(! scanner.hasNextInt())
{
    System.out.println("Foutieve invoer!");
    return; // Stop met de uitvoer van deze functie / het programma
}

int getal = scanner.nextInt();
System.out.println("Jouw getal: " + getal);
```

```
$ java MijnInvoer
Voer een geheel getal in: a
Foutieve invoer!
$ java MijnInvoer
Voer een geheel getal in: 3
Je hebt het volgende getal ingevoerd: 3
```

Foutafhandeling (3)

Toch is vorige code weinig elegant, vanwege het return-statement bij een foutieve invoer?

Hoe kunnen we dit verbeteren?

Invoer opvragen met behulp van een loop

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    boolean geldigeInvoer = false;
    do {
        System.out.println("Voer een geheel getal in:");

        if(! scanner.hasNextInt()) {
            System.out.println("Foutieve invoer!");
        } else {
            int getal = scanner.nextInt();
            geldigeInvoer = true;
            System.out.println("Jouw getal: " + getal);
        }
    }while(! geldigeInvoer);
}
```

Wat gaat hier fout?

Foutieve invoer overslaan

Met behulp van **scanner.next()** kunnen we de ongeldige invoer overslaan.

```
Scanner scanner = new Scanner(System.in);

boolean geldigeInvoer = false;
do {
    System.out.println("Voer een geheel getal in:");

    if(! scanner.hasNextInt()) {
        System.out.println("Foutieve invoer!");
        scanner.next();
    } else {
        int getal = scanner.nextInt();
        geldigeInvoer = true;
        System.out.println("Jouw getal: " + getal);
    }
}while(! geldigeInvoer);
```

Hoera, het werkt! Toch?! Niet? Nouuu :-(

```
boolean geldigeInvoer = false;
do {
    System.out.println("Voer een geheel getal in:");
    if(! scanner.hasNextInt()) {
        System.out.println("Foutieve invoer!");
        scanner.next();
    } else {
        int getal = scanner.nextInt();
        geldigeInvoer = true;
    }
}while(! geldigeInvoer);
System.out.println("Jouw getal: " + getal);
System.out.println("Jouw getal maal 2: " + (2 * getal));
```

```
$ javac MijnInvoer.java
MijnInvoer.java:21: error: cannot find symbol
System.out.println("Jouw getal: " + getal);
                                     ^
```

Inhoud

Scope van een variabele

Een variabele is alleen te gebruiken binnen het **block** waarin de variabele is gedeclareerd.
Een block wordt omsloten door accolades: {}

Scope van een variabele (2)

```
public static void main(String args[]) {
    int a = 0;

    if(a < 10) {
        int b = 3;
        a = a + 3;
    }

    // a bestaat hier nog steeds
    // b is hier niet bruikbaar
}
```

Dit gedrag is niet in alle programmeertalen vergelijkbaar

```python
# Python
som = 0
for i in range(10):
    som = som + i
print(i)
```

```php
<?php
$som = 0;
for($i = 0; $i < 10; $i++)
    $som = $som + $i;
echo $i;
```

```
$ python3 voorbeeld.py
9
$ php voorbeeld.php
10
```

Variabelen zijn dus gescheiden tussen verschillende methode

```
public static void main(String args[])
{
    int a = 0;
    int b = 3;

    a = 3;
    rekenUit();
}

public static rekenUit()
{
    int a = 4; // dit is een andere a!

    // hier is a=4 beschikbaar, en b niet
}
```

Inhoud

Variabelen

- **Gegevens** worden opgeslagen in geheugenlocaties
- Wij zijn als Java programmeur niet geinteresseerd in deze locaties
- Een **variabele** is een **naam** voor een geheugenlocatie
- Deze **naam** kun je gebruiken om toegang te krijgen tot de **gegevens** in deze geheugenlocatie
- Iedere variabele heeft:
  1. **naam** (een voor de scope unieke identifier, gekozen door de programmeur)
  2. **type** (dat waar we verder naar gaan kijken dit college)
  3. **waarde** (de gegevens die opgeslagen zijn op de desbetreffende geheugenlocatie)
  4. **scope** (zichtbaarheid en levensduur van de variabele)

8 primitieve datatypen

In Java onderscheiden we 8 primitieve datatypen.

1. **boolean** (voor waar/onwaar)
2. **char** (voor karakters)
3. **byte**, **short**, **int**, **long** (voor gehele getallen)
4. **float**, **double** (voor reële getallen)

Bereik van de primitieve datatypen[2]

datatype	aantal bits	ondergrens	bovengrens	aantal
boolean	**1**	false	true	2
char	**16**	\u0000	\uFFFF	2^{16}
byte	**8**	-128	127	2^8 = 256
short	**16**	-32768	32767	2^{16} = 65536
int	**32**	-2^{31}	$2^{31}-1$	2^{32} = 4.2 miljard
long	**64**	-2^{63}	$2^{63}-1$	2^{64}
float	**32**	$-3,40 \cdot 10^{38}$	$3,40 \cdot 10^{38}$	
double	**64**	$-1,80 \cdot 10^{308}$	$1,80 \cdot 10^{308}$	

[2] Tentamen: niet stampen, wel begrijpen!

Literals

"A `literal` is the source code representation of a fixed value; literals are represented directly in your code without requiring computation."

```
boolean b = true;
char    c = 'r';
byte    b = 8;
short   s = 300;
int     i = 100000;
float   f = 1.1f;
double  d = 123.4;
long    l = 1234567890000L;
```

Literals (2)

En de compiler vindt het niet leuk als je de verkeerde literals in het verkeerde type stopt:

```
byte  b = 300;
float f = 123.4;
```

```
$ javac Literals.java
HelloWorld.java:4: error: incompatible types: possible lossy conversion from
int to byte
        byte  b = 300;
                  ^
HelloWorld.java:5: error: incompatible types: possible lossy conversion from
double to float
        float f = 123.4;
                  ^
2 errors
```

Literals (3)

Je kunt literals van een **kleiner** datatype wel kwijt in een **groter** datatype:

```
double d = 123.4f;
long   l = 100000;
System.out.println(d);
System.out.println(l);
```

```
$ javac Literals.java
$ java Literals
123.4000015258789
100000
```

Karakters

De waarden van het type `char` zijn karakters.

- een **char** is een 16-bit **unicode** karakter
  minimum `\u0000` (0) en maximum `\uffff` (65.535)
- Literals schrijf je tussen enkele aanhalingstekens (hexadecimaal): `'\u0061'`
- De meest gebruikte karakters vallen binnen 7-bits ASCII alfabet (128)
- De ASCII karakters zijn, tussen aanhalingstekens, ook direct gebruiken:
  `'A'`, `'3'`, en `'!'`

Karakters (2): De ASCII tabel [3]

	30	40	50	60	70	80	90	100	110	120
0:		(	2	<	F	P	Z	d	n	x
1:		)	3	=	G	Q	[	e	o	y
2:		*	4	>	H	R	\	f	p	z
3:	!	+	5	?	I	S	]	g	q	{
4:	"	,	6	@	J	T	^	h	r	|
5:	#	-	7	A	K	U	_	i	s	}
6:	$	.	8	B	L	V	‘	j	t	~
7:	%	/	9	C	M	W	a	k	u	DEL
8:	&	0	:	D	N	X	b	l	v	
9:	’	1	;	E	O	Y	c	m	w	

[3] Probeer eens: `man ascii` in je terminal

Karakters (3)

Intern zijn karakters dus 'gewoon' getallen volgens de ASCII-tabel / unicode.

```
char a = 'q' - ('a' - 'A');
char b = 'Q' + ('b' - 'B');
System.out.println(a);
System.out.println(b);
```

Wat is de uitvoer van bovenstaande code?

```
Q
q
```

Wordt vervolgd

Volgende keer meer over primitieve datatypen voor de 2e opgave!

Verwacht dus ‘uitdagende’ vragen bij de volgende quizzz!