Opdracht 4: Dodo wordt slimmer

Algoritmisch Denken en Gestructureerd Programmeren (in Greenfoot)

Inhoudsopgave

Opslaan en inleveren **24**

Inleiding

Als Mimi een moeilijk probleem tegenkomt maakt ze altijd eerst een plan. Met een plan voelt Mimi zich sterker. Sommige soorten problemen komen vaak voor. Informatici hebben daar dan ook standaard plannen voor bedacht. Er zijn plannen voor allerlei dingen, zoals tellen, herhalen, dingen verwisselen, de grootste van iets vinden .... Je leert standaard plannen en past deze aan zodat je daarmee een specifiek probleem kunt oplossen.

Een van die plannen is het *Variabelenplan*. Variabelen gebruik je om informatie bij te houden. Als Mimi haar eieren gaat tellen moet ze natuurlijk een getal kunnen onthouden. Met variabelen geven we Mimi 'geheugen'. We gaan Mimi dus slimmer maken! Door haar slimmer te maken kan ze ook complexere opdrachten uitvoeren.

Doelen

De doelen van deze opdracht zijn:

- **Variabelen gebruiken**
- **Generieke algoritmes bedenken die als oplossing voor meerdere problemen kunnen dienen**

Leerdoelen

Na het voltooien van deze opdracht kun je:

- uitleggen waarvoor **variabelen** gebruikt worden;
- een variabele **declareren**, **initialiseren** en een waarde toekennen;
- code en de daarin gebruikte variabelen traceren met behulp van een **tracetabel**;
- de **(waarde)toekenningsoperator** '=' gebruiken;
- de **vergelijkingsoperatoren** '==', '!=', ' <', ' <=', ' >' en ' >=' gebruiken;
- de **rekenkundigeoperatoren** '+, '-', ' *', '/' en '%' gebruiken;
- de **verhogingsoperator** ' ++' (en verlagingsoperator ' --') gebruiken;
- **plannen** voor tellen, herhalingen en voor het verwisselen van waarden herkennen en toepassen;
- een **teller** gebruiken om te bepalen hoe vaak bepaalde code herhaald moet worden (**Plan: Teller-gestuurde herhaling**);
- een conditie gebruiken om steeds te bepalen of bepaalde code herhaald moet worden (**Plan: Conditie-gestuurde herhaling**);
- de rol van een **parameter** (van een methode of constructor) uitleggen;
- controleren op ongeldige waarden.

Instructies

In deze opdracht ga je verder met jouw eigen code uit de vorige opdracht. Maak een kopie van jouw scenario zodat je altijd nog terug kan naar een vorige versie. Een kopie maken doe je zo:

- Open jouw scenario van de vorige opdracht.
- In de Greenfoot menu, bovenaan het scherm, selecteer je 'Scenario' en dan 'Save As ...'.
- Ga naar de folder waarin je jouw werk voor opdracht 4 wilt opslaan.
- Kies een bestandsnaam met jouw eigen naam en opdrachtnummer, bijvoorbeeld: `Opdr4_John`.

Tijdes de opdracht zul je wat vragen moeten beantwoorden. Je moet het volgende inleveren:

- Alle stroomdiagrammen: gebruik potlood en papier, of maak gebruik van de software op `http://course.cs.ru.nl/greenfoot/flowchart/flowcharttool.html`;
- Jouw code: het bestand `MyDodo.jav` bevat al jouw code en moet ingeleverd worden;
- Het reflectieblad: vul in en lever het in.

Je moet al jouw antwoorden met je partner te bespreken. Noteer kort jullie antwoorden.

Er zijn drie soorten taken:

	**Aanbevolen**. Leerlingen met weinig programmeerervaring, of leerlingen die wat meer oefening nodig hebben, moeten deze taken allemaal afmaken.
	**Verplicht**. Iedereen moet deze taken afmaken.
	**Uitdagend**. Complexere taken, ontwikkeld voor leerlingen die de 2-ster opdrachten voltooid hebben en klaar zijn voor een grotere uitdaging.

Leerlingen die 1-ster taken overslaan moeten alle 3-ster taken maken.
Opmerking:

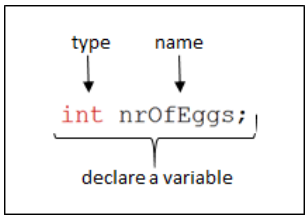
- In deze opdracht mag je alleen aanpassingen maken in `MyDodo`;
- Je mag methodes gebruiken uit de `MyDodo` of `Dodo` klasse, maar niet uit de `Actor` klasse;
- 'Teleporteren' is niet toegestaan: als Mimi ergens moet zijn, dan moet ze daar zelf stap voor stap heenlopen (ze mag er niet in één keer naar toe springen).

Theorie

Theorie 4.1: Variabelenplan

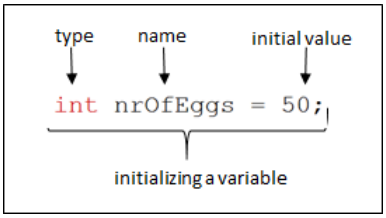
Variabelen worden gebruikt om gegevens in op te slaan. Voordat je een variabele gebruikt moet je deze eerst:

1. **Declareren** (of aanmaken): een naam en type geven.



- *Type*: bijvoorbeeld `int`, `boolean`, `String`, of zoals we straks zullen zien, een klasse. Het type van een variabele bepaalt wat voor soort waarden deze variabele kan bevatten.
- *Naam*: bijvoorbeeld `Mimi` of `aantalEieren`.

2. **Initialiseren**: een *beginwaarde* geven. Deze kan later aangepast worden.[2].

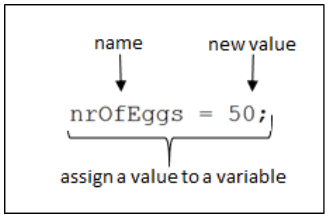


Voorbeelden van declaratie en gebruik van variabelen

Type	Betekenis	Voorbeeld van variabele declaratie	Voorbeeld van gebruik van variabele
`int`	Geheel getal	`int nrOfEggsHatched = 2`	`if ( nrOfEggsHatched == 12)...`
`boolean`	`true` of `false`	`boolean doneLookingForEgg = false`	`if ( doneLookingForEgg )...`
`String`	Tekst	`String naam = "Mimi"`	`showCompliment( naam );`

Waarde toekennen

Een waarde toekennen aan een variabele doe je met '='. Het '=' teken spreek je uit als 'wordt'.



Figuur 1: De variabele `nrOfEggs` wordt 50

Je kunt alleen een waarde aan een variabele toekennen als deze waarde van **hetzelfde** type is als de variabele. Dat betekent dat het type aan de linkerkant van de '=' moet gelijk zijn aan het type aan de rechterkant. In bovenstaand voorbeeld is het type:

- aan de linkerkant `int`, want het type van `nrOfEggs` is een `int`,
- aan de rechterkant ook `int`, want 50 is een `int`.

[2]Het is goed gebruik om een variabele altijd een beginwaarde te geven. In Java hoeft het echter niet per se.

Het gebruik van een variabele

Wat je met een variabele kan doen, dus welke *operaties* je kunt toepassen, hangt van zijn type af. `Strings` kun je bijvoorbeeld aan elkaar plakken. Getallen van het type `int` kun je vergelijken, vermenigvuldigen, optellen, verhogen ...
Voorbeelden zijn:

- `aantalDozijnen = aantalEieren / 12;`
  de variabele `aantalDozijnen` krijgt de waarde van aantalEieren gedeeld door 12. Opmerking: de deling die hier gebruikt wordt is een zogenaamde *integerdeling*, dat wil zeggen dat de rest na deling weggegooid wordt (7/4 levert bijvoorbeeld 1 op).
- `aantalEieren = aantalEieren + 1;`
  de variabele `aantalEieren` wordt verhoogd met 1 (bijvoorbeeld als Mimi nog een ei legt).
- `aantalEieren++;`
  de variabele `aantalEieren` wordt verhoogd met 1 (het effect van '++' is hetzelfde als de verhoging met 1 zoals hierboven).
- `totaalAantalEieren += aantalNieuweEierenGevonden;`
  is een verkorte schrijfwijze voor:
  `totaalAantalEieren = totaalAantalEieren+ aantalNieuweEierenGevonden;`

Toevoeging:

- Als je een variabele declareert, geef je meteen het type aan. Als je de variabele gebruikt, dan geef je het type niet aan.[3]
- Een methode kan ook parameters hebben. Een variabele kan je ook als parameter gebruiken. Bijvoorbeeld in de volgende methode aanroep:
  ```
  legAantalEieren( aantalEieren );
  ```
  In deze aanroep wordt de waarde van de variabele `aantalEieren` als parameter aan de methode `legAantalEieren` meegegeven.

Naamgevingsafspraken

De naam van een variabele:

- is betekenisvol: hij komt overeen met het doel van de variabele. De enige uitzondering is de naam van een teller-variabele in een loop. Deze mag één letter zijn, bijvoorbeeld `i`.
- bevat één of meer zelfstandige naamwoorden.
- is geschreven in *lowerCaseCamel*: begint met een kleine letter, elk volgend 'woord' begint met een hoofdletter.
- bevat geen spaties, komma's of andere 'rare' karakters ('_' uitgezonderd).
- Voorbeeld: `nrEggsFound`.

[3] Dit is vergelijkbaar met het aanroepen van een methode. In zo'n aanroep gebruik je alleen de naam en niet de hele signatuur.

Levensduur

Variabelen kunnen op verschillende plaatsen in de code worden aangemaakt (gedeclareerd). De plek waar de declaratie staat bepaalt het *geldigheidsgebied* (ook wel *bereik* of *scope* genoemd) van deze variable. Een variabele die in een methode wordt aangemaakt is alleen beschikbaar binnen deze methode.

Variabelen kunnen ook buiten een methode in de klasse zelf gedeclareerd worden. Het bereik van deze variabele is de hele klasse: de variabele kan dus overal gebruikt worden.[4].

Een begrip dat nauw samenhangt met scope is de *levensduur* van een variable. De levensduur van variabelen (en ook van objecten) zegt iets over hun bestaan terwijl een programma wordt uitgevoerd. Zo is de levensduur van een variable die in een methode gedeclareerd is beperkt tot één aanroep van de methode. Dat wil zeggen, de variabele bestaat zolang de methode wordt uitgevoerd. Is de methode klaar dan verdwijnt de variable (en daarmee ook zijn waarde). Bij een volgende aanroep van dezelfde methode wordt de variabele weer opnieuw aangemaakt. Variabelen die in een klasse gedeclareerd hebben een levensduur die gelijk is aan die van het object (de instantie van de klasse). En sommige objecten blijven bestaan totdat de uitvoering van het hele programma stopt (hierover leer je meer in de volgende opdracht).

De scope kun je in Greenfoot herkennen doordat in de editor gebruik wordt gemaakt van verschillende achtergrondkleuren. In het volgende plaatje is de scope van de variabele `int stepsTaken` beperkt tot het witte gebied en alle gebieden die binnen het witte gebied vallen, in dit geval het roze gebied.

```
public int walkAndCountSteps(){
    int stepsTaken=0;

    while( canMove() ){
        stepsTaken++;
        move();
    }
    return stepsTaken;
}
```

Parameters hebben ook een bereik: dit is beperkt tot de body van de methode (of constructor) waartoe ze behoren. In het volgende plaatje is te zien dat de scope van de parameter `int nrStepsToTake` beperkt is tot het witte gebied (en het roze gebied, want dat valt binnen het witte gebied) van de methode.

```
public void takeSteps (int nrStepsToTake){
    int stepsTaken = 0;

    while (stepsTaken < nrStepsToTake){
        stepsTaken++
        move();
    }

}
```

Voorbeeld

We schrijven een methode die het kwadraat van een getal oplevert:

[4]Eigenlijk zijn de echte scope-regels van Java nog wat ingewikkelder. Om het zo eenvoudig mogelijk te houden laten we een preciezere omschrijving achterwege

```java
/**
* This method returns the square of a given number
*
* @param  number    the integer to be squared
* @return           the value after squaring
*/
public int square( int number ) {
   // declareren en initialiseren van de uitkomst, kwadraat uitrekenen
   int result = number*number;

   // de methode levert het resultaat van het kwadrateren op
   return result;
}
```

Toelichting:

We noemen een variabele die in een methode gedeclareerd is (en dus alleen binnen die methode gebruikt kan worden) een *lokale variabele*. In dit voorbeeld is `int result` dus een lokale variabele van de methode `square`. Ook kan de parameter `int number` alléén binnen deze methode gebruikt worden.

Theorie 4.2: Operatoren

Toekenningsoperator

Met de ' =' operator geef je een bepaalde waarde aan een variabele. Bijvoorbeeld:

Instructie	Resulteerde waarde
`aantalEierenInNest = 4;` `aantalEierenDoorSlangOpgegeten = aantalEierenInNest;`	`aantalEierenInNest` heeft de waarde 4 `aantalEierenDoorSlangOpgegeten` is 4 (en `aantalEierenInNest` blijft 4)

Toevoeging:

Als je de waarde van een variable toekent aan een andere variabele, wordt er een kopie van deze waarde gemaakt. In het voorbeeld hierboven krijgt `aantalEierenDoorSlangOpgegeten` de waarde 4, en `aantalEierenInNest` blijft ook 4.

Vergelijkingsoperatoren

Met de volgende operatoren kun je getallen (gehele getallen en kommagetallen) vergelijken:

Operator	Betekenis	Voorbeeld
`==`	is gelijk aan	`getal == 4`
`!=`	is NIET gelijk aan	`getal1 != getal2`
`>`	is groter dan	`getal > 3`
`>=`	is groter of gelijk aan	`getal1 >= getal2`
`<`	is kleiner dan	`getal < 5`
`<=`	is kleiner of gelijk aan	`getal <= 5`

Hier komt altijd een `boolean` als resultaat uit (dus `true` of `false`).

Rekenkundige operatoren

Met de volgende operatoren kun je rekenen met getallen:

Operator	Betekenis	Voorbeeld
`+`	optellen	`uitkomst = getal + 4`
`-`	aftrekken	`uitkomst = getal - 4`
`*`	vermenigvuldigen	`uitkomst = getal * 5`
`/`	delen door	`uitkomst = getal / 5`

Verhogings- en verlagingsoperatoren

Met de volgende operatoren kan de waarde van een variabele met 1 worden verhoogd of verlaagd:

Operator	Betekenis	Voorbeeld
`++`	met één verhogen	`uitkomst++`
`--`	met één verlagen	`uitkomst--`

Voorbeelden

- Declareren en initialiseren: Stel we willen bijhouden hoeveel eieren Mimi gevonden heeft. Mimi heeft aan het begin al 4 eieren gevonden. Met de volgende instructie wordt een variabele gedeclareerd en geïnitialiseerd: `int aantalEierenGevonden = 4;`
- Optellingsoperator: Als ze daarna nog eens twee eieren vindt, dan zal de volgende instructie de waarde van `aantalEierenGevonden` met twee verhogen:
  `aantalEierenGevonden = aantalEierenGevonden + 2;`
- Verlagingsoperator: Als ze een ei kwijtraakt, dan zal de volgende instructie de waarde met ééntje verlagen: `aantalEierenGevonden--;`

De operaties hierboven passen steeds de waarde van de variabele aan. Om te controleren of Mimi inmiddels vijf eieren gevonden heeft gebruiken we: `aantalEierenGevonden == 5`. Dit is inderdaad `true`.

Theorie 4.3: Variabelen traceren

Het nalopen van waarden die variabelen op bepaalde momenten tijdens de uitvoering van het programma zullen krijgen heet *traceren*. Traceren is een handige vaardigheid bij het opsporen van fouten in de code.

Voorbeeld:

Voor de volgende programmacode gaan we de variabelen `a`, `b` en `c` traceren:

```
public void oefenenTracerenVariabelen( ){
   int a = 2;
   int b = 3;

   int c = a * b;
   System.out.println("De waarde van a is: " + a + ", b is: " + b + ", c is: " + c);

   b = c - a;
   System.out.println("De waarde van a is: " + a + ", b is: " + b + ", c is: " + c);
```

```
    a = a + b + c;
    System.out.println("De waarde van a is: " + a + ", b is: " + b + ", c is: " + c);

    c = b * a;
    System.out.println("De waarde van a is: " + a + ", b is: " + b + ", c is: " + c);
}
```

We gebruiken een *tracing tabel* om de waardeveranderingen van de variabelen bij te houden na het uitvoeren van iedere instructie:

Instructie	Waarde na uitvoeren van de instructie		
	a	b	c
`int a = 2;`	**2**	-	-
`int b = 3;`	2	**3**	-
`int c = a * b;`	2	3	**6**
`b = c - a;`	2	**4**	6
`a = a + b + c;`	**12**	4	6
`c = b * a;`	12	4	**48**

Met een streepje (’-’) geven we aan dat de variabele nog niet gedeclareerd of geïnitialiseerd is. Aan het einde van het programma is de waarde van `a` gelijk aan 12, die van `b` gelijk aan 4 en die van `c` gelijk aan 48.

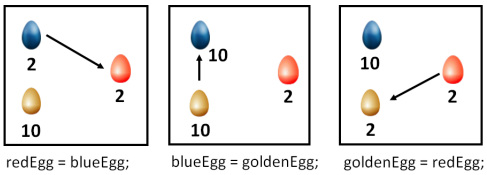
Theorie 4.4: Verwisselplan

Om de waarden van twee variabelen te verwisselen heb je een extra tijdelijke ’hulp-variabele’ nodig. Deze algemene techniek noemen we het *driehoekig verwisselplan*.

Voorbeeld:

Het doel is om de waarden van twee eieren te verwisselen. Een blauw ei is 2 punten waard en een gouden 10. De waarden van de eieren houden we bij met de variabelen `waardeBlauweEi` en `waardeGoudenEi`. Beide variabelen hebben het type `int`. Het verwisselen van de waarden gaat als volgt:

1. We maken een (tijdelijke) hulpvariabele `int tijdelijkeWaardeEi` aan.
2. De `tijdelijkeWaardeEi` krijgt de waarde van `waardeBlauweEi`.
3. De `waardeBlauweEi` krijgt de waarde van `waardeGoudenEi`.
4. De `waardeGoudenEi` krijgt de waarde van `tijdelijkeWaardeEi`.



Figuur 2: Plan voor het verwisselen van waarden van twee variabelen

Instructie	Waarde na uitvoeren van instructie		
	`waardeBlauweEi`	`waardeGoudenEi`	`tijdelijkeWaardeEi`
beginwaarden	2	10	-
`waardeTijdelijkeEi = waardeBlauweEi;`	2	10	**2**
`waardeBlauweEi = waardeGoudenEi;`	**10**	10	2
`waardeGoudenEi = waardeTijdelijkeEi;`	10	**2**	2

Toevoeging:

Dit kun je ook zelf uitproberen met bestanden op jouw PC.

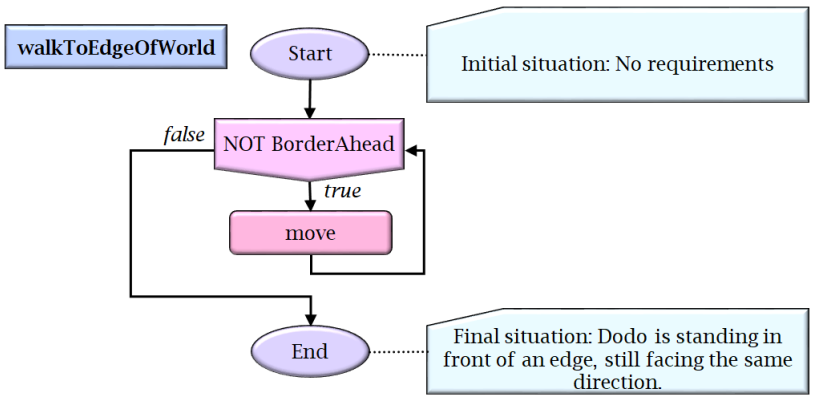
1. Open je (Windows) verkenner.
2. Maak in Notepad een bestand aan met de naam `A.txt`. Zet hier een tekst in.
3. Maak ook een bestand aan met de naam `B.txt`. Zet hier een andere tekst in.
4. Probeer nu, alléén door bestanden te kopiëren, de inhoud van de bestanden te verwisselen.

Theorie 4.5: Conditie-gestuurde herhalingsplan

Soms weet je van tevoren precies hoe vaak je wilt dat code herhaald wordt, maar soms ook niet. In dit laatste geval wil je dat code herhaald wordt zolang aan een bepaalde voorwaarde is voldaan. De situatie en de voorwaarde bepalen dan hoe vaak de code uitgevoerd moet worden. Dit heet een *conditie-gestuurde herhaling* (of een *sentinel-controlled loop*).

Voorbeeld:

Als concreet voorbeeld bekijken we `MyDodo`'s code voor `void walkToWorldEdge( )`.
Zolang de conditie `! borderAhead( )` waar is, zal de loop herhaald worden en zet Mimi een stapje vooruit.



Figuur 3: Stroomdiagram voor een conditie-gestuurde herhaling

Zodra niet meer aan de conditie wordt voldaan, zal de herhaling stoppen.

Theorie 4.6: Generieke strategieën

Programma's die generiek zijn kunnen gebruikt worden om meerdere vergelijkbare problemen op te lossen. Door handig generieke condities te kiezen, kun je jouw oplossingen in meerdere situaties gebruiken. Bijvoorbeeld, een methode die werkt voor iedere wereldgrootte in plaats van een methode die alleen voor een vaste grootte zoals '12' werkt. In dit specifieke voorbeeld kun je de methode generiek maken door:

- gebruik te maken van condities zoals `borderAhead()` of
- het aantal rijen `nrOfRowsInWorld` niet als 'hard' getal (zoals 12) op te geven, maar te bepalen met behulp van:

```
World world = getWorld();                // gets world to determine world height
int nrOfRowsInWorld = world.getHeight(); // stores world height in nrOfRowsInWorld
```

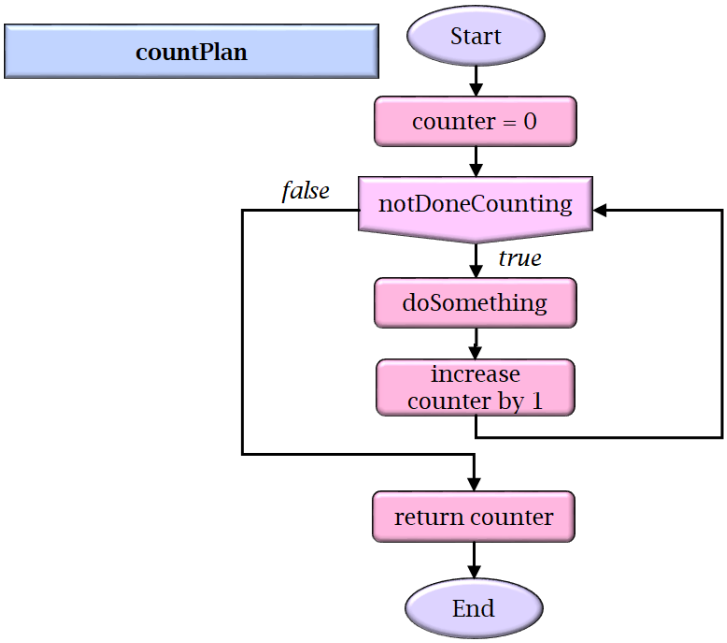
- parameters te gebruiken als invoer voor een methode. Een methode met parameters is flexibeler, en dus generieker, dan een methode zonder parameters. De parameter wordt dan gebruikt om specifieke waarden aan de methode mee te geven, zoals hoe vaak een gedeelte van een algoritme herhaald moet worden. Dezelfde methode kan dan gebruikt worden om meerdere vergelijkbare problemen op te lossen. Bijvoorbeeld, Mimi kan de methode `void jump( int distance )` gebruiken om verschillende afstanden te overbruggen, afhankelijk van de parameter die de methode meekrijgt. Dit in tegenstelling tot de niet-generieke methode `void move( )` waarbij Mimi maar één stapje kan zetten.

```
public void jump( int distance ) {
    int stepsTaken = 0;                 // set counter to 0
    while ( stepsTaken < distance ) {   // check if more steps must be taken
        move( );                        // take a step
        stepsTaken++;                   // increment the counter
    }
}
```

Theorie 4.7: Telplan

Een *teller* is een variabele die gebruikt wordt om bij te houden hoe vaak iets is gebeurd of nog moet gebeuren. Bijvoorbeeld, we kunnen een variabele gebruiken om te tellen hoe vaak de body van een `while` loop uitgevoerd wordt. Een *telplan* ziet er als volgt uit:

- Declareer een tellervariabele die je meteen initialiseert, vaak met beginwaarde 0;
- Binnen de loop pas je de teller aan, meestal met een ophoging van één.
- Aan het einde van de methode lever je de teller op met een `return`.



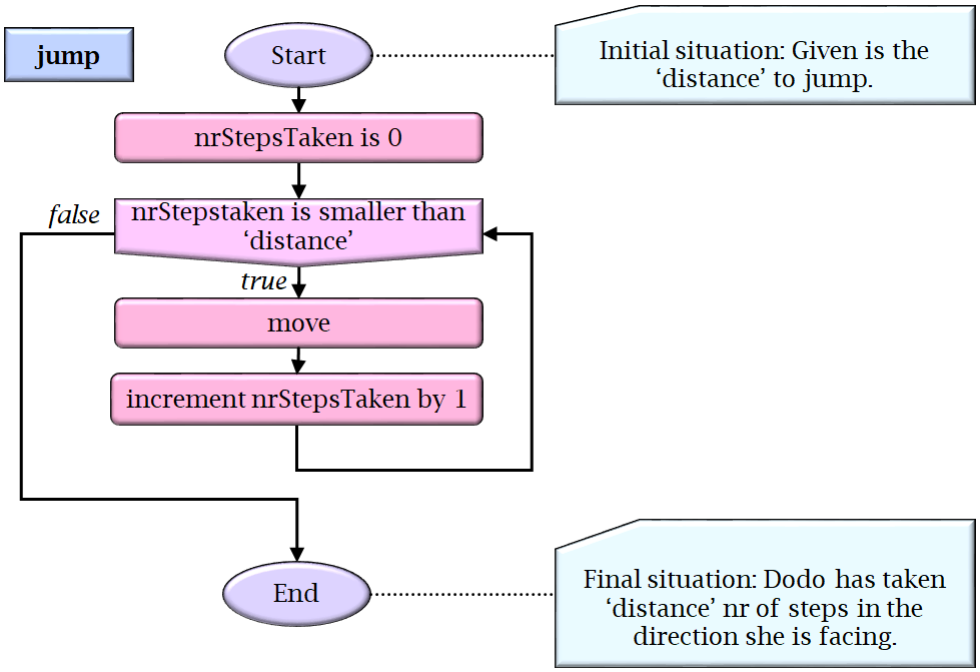

```
/**
 * Example of a count plan: counting steps till the end of the world
 */
public int countSteps( ) {
    int nrStepsTaken = 0;       // set counter to 0
    while ( !borderAhead() ) {  // check if more steps can be taken
        move();                 // take a step
        nrStepsTaken++;         // increment the counter
    }
    return nrStepsTaken++;
}
```

Theorie 4.8: Teller-gestuurde herhalingsplan

Om aan te geven hoe vaak de loop herhaald moet worden kun je een teller-variable gebruiken. Aan het begin van de loop wordt aan de hand van deze teller (in de conditie van de loop) bepaald of de body van de loop wel of niet uitgevoerd moet worden. Deze techniek heet ook wel *teller-gestuurde herhaling* (of *counter-controlled loop*).

Voorbeeld:

Als concreet voorbeeld bekijken we `MyDodo`'s code voor `void jump (int distance)` (zie het stroomdiagram en code hieronder). Hiermee zet Mimi herhaaldelijk stappen totdat ze een bepaalde afstand heeft afgelegd. Een teller (`nrStepsTaken`) wordt gebruikt om bij te houden hoeveel stappen Mimi heeft gezet. Met elke stap wordt de teller met één verhoogd. Zolang de teller nog niet het vereiste aantal stappen heeft bereikt, wordt de loop opnieuw uitgevoerd. Als de teller (`nrStepsTaken`) gelijk is aan het vereiste aantal stappen (`distance`) stopt de herhaling.



Figuur 4: Stroomdiagram voor de `jump` methode

```
public void jump ( int distance ) {
   int nrStepsTaken = 0;              // teller 'nrStepsTaken' initialiseren
   while ( nrStepsTaken < distance ){ // Afstand nog niet afgelegd? Dan doorgaan
      move( );                        // neem een stap
      nrStepsTaken++;                 // aantal stappen verhogen
   }
}
```

Toelichting:

Veelvoorkomende fouten of misvattingen zijn:

- vergeten de teller in de loop te verhogen. Hierdoor zal het programma nooit uit de loop komen en dus ook nooit stoppen.
- de verkeerde vergelijkingsoperator gebruiken in de conditie. Als de teller als beginwaarde 0 krijgt, dan moet je vaak ' <' in de conditie gebruiken in plaats van ' <=' om niet één stap te veel te zetten.
- denken dat de methode *direct* stopt als de teller verhoogd wordt naar de gewenste waarde. Dat klopt niet, want alle code in de loop zal nog uitgevoerd worden. Daarna, als de voorwaarde weer gecontroleerd wordt, zal pas blijken dat die niet voldoet en zal de loop worden beëindigd.

Uitdagingen

Lees eerst Theorie 4.1: Variabelenplan.

Lees eerst Theorie 4.2: Operatoren.

Lees eerst Theorie 4.3: Variabelen traceren.

Lees eerst Theorie 4.4: Verwisselplan.

Opgave 4.1: Code traceren

We gaan nu wat oefenen met variabelen en operaties. Bekijk de volgende stukken code. Geef telkens aan wat de waarden van de variabelen na het uitvoeren van de code zijn. Gebruik hiervoor een tracing tabel. Je mag daarna Greenfoot gebruiken om jouw antwoord te controleren. Gebruik de programmacode uit Figuur 5.

```
public void practiceTracingCode() {
   int nrOfEggsFound = 3;
   nrOfEggsFound++

   System.out.println( "Value of nrOfEggsFound is: " + nrOfEggsFound );
}
```

Figuur 5: Code voor het oefenen met traceren

a) Wat wordt de waarde van `aantalEierenGevonden`? Welke waarde van `aantalEierenGevonden` hoort bij de codefragmenten A,B,C en D?

Vraag	Code
A	`int aantalEierenGevonden = 3;` `aantalEierenGevonden ++;`
B	`int aantalEierenGevonden = 2;` `aantalEierenGevonden = aantalEierenGevonden + 4;`
C	`int aantalEierenGevonden = 1;` `aantalEierenGevonden--;`
D	`int aantalEierenGevonden = 1;` `aantalEierenGevonden +=2;`

aantalEierenGevonden
0
2
3
4
6

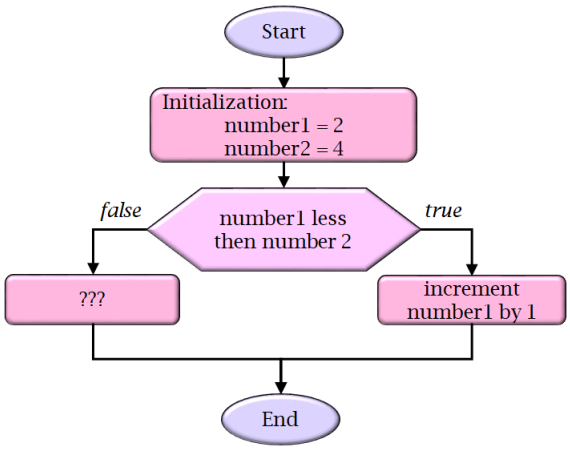
b) Vul de tracing tabel in met de juiste waarden van `getal1` en `getal2`.

```
int getal1 = 2;
int getal2 = 4;

getal1 = getal1 + getal2;
getal2 = getal1 + getal2;
```

Instructie	Waarde van `getal1`	Waarde van `getal2`
initialisatie	2	4
`getal1 = getal1 + getal2;`		
`getal2 = getal1 + getal2;`		

c) i) Vul de ontbrekende onderdelen in het stroomdiagram en in de programmacode in.

ii) Na het uitvoeren van de volgende code zouden de waarden van `getal1` en `getal2` aan elkaar gelijk moeten zijn. Pas het stroomdiagram en de code aan.




```
int getal1 = 2;
int getal2 = 4;

if ( ??? ){
    getal1++;
} else {
    getal2 = getal1;
}
```

d) Bekijk de volgende code:

```
int getal1 = 2;
int getal2 = getal1 * 3;

getal1 = getal2;
getal2 = getal1 + getal1;
```

i) Wat worden de waarden van `getal1` en `getal2`?

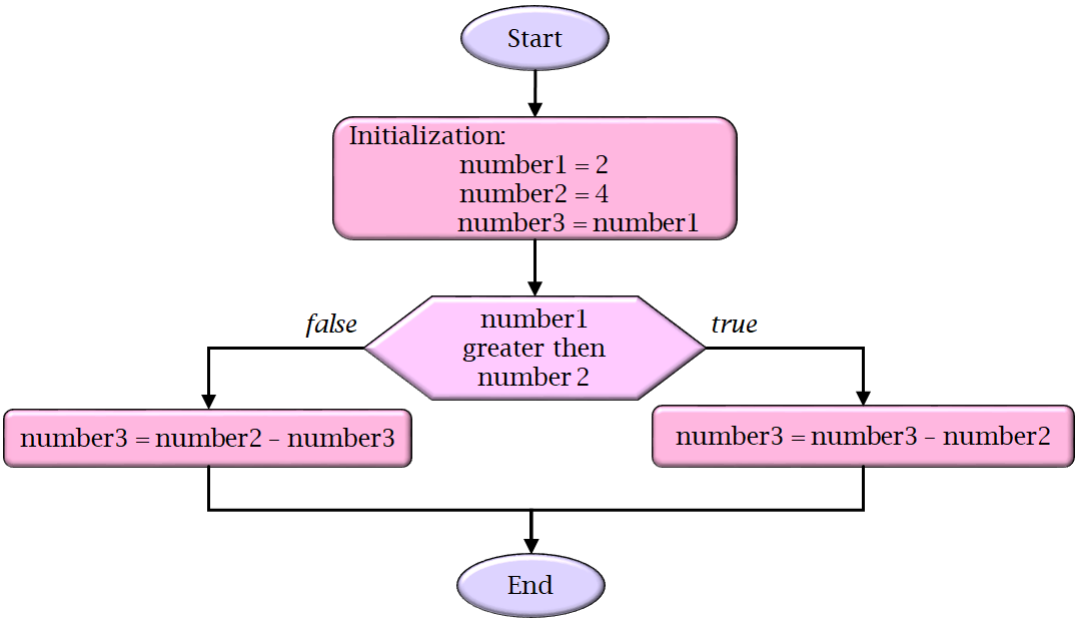
ii) Stel dat de waarde van `getal2` na uitvoer gelijk moeten zijn aan 36. Pas de fout in de code aan.

e) Vul de waarden in de tracing tabel aan voor het volgende programmaatje:

```
int getal1 = 6;
int getal2 = 3;

getal1 = getal2;
getal2 = getal1;
if( getal1 == getal2 ){
    getal1 = getal1 + getal2;
}
```

Instructie	`getal1`	`getal2`
initialisatie	6	3
`getal1 = getal2;`		

f) Wat wordt de waarde van `getal3`?



g) Wat wordt de waarde van `int getal3`?

```
int getal1 = 10;
int getal2 = 8;
int getal3 = 4;

getal3 = doeIetsMetTweeGetallen( getal1, getal2 );
```

waar:

```
public int doeIetsMetTweeGetallen ( int a, int b ) {
   int resultaat = ( a + b ) / 2;
   return resultaat;
}
```

Opgave 4.2: Namen verwisselen

Taak: *Mimi heeft haar eieren per ongeluk de verkeerde namen gegeven! Kun jij haar helpen deze te verwisselen?*

Figuur 6: Mimi heeft de namen van haar eieren per ongeluk verwisseld

a) Bepaal een strategie om de namen te verwisselen.

b) Vertaal jouw strategie naar code. Tip: Weet je niet waar je moet beginnen? Bekijk dan figuur 7.

c) Druk de namen af naar de console om te testen of jouw code goed werkt.

d) Hoeveel variabelen heb je nodig gehad? Wat is na afloop de waarde van elk variabele?

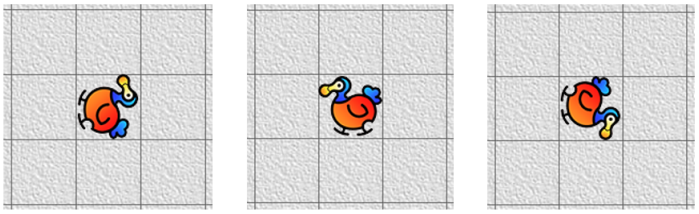
```
public void swapEggNames() {
   String nameOfBlueEgg   = "Gold";
   String nameOfGoldenEgg = "Blue";

   System.out.println( "Initial values before the swap: " );
   System.out.println( "Name of blue egg is: " + nameOfBlueEgg );
}
```

Figuur 7: Eerste stappen voor het verwisselen van waarden

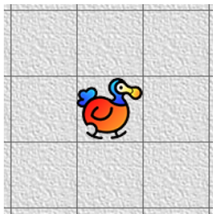
Opgave 4.3: Draai naar het oosten

Taak: *Schrijf een methode `void faceEast( )` waarmee Mimi naar het oosten draait.*
Tip: Gebruik de methode `int getDirection ()` om te bepalen in welke richting Mimi kijkt. Bijvoorbeeld, met `getDirection( )== EAST` controleer je of Mimi naar het oosten kijkt.

Begin scenario:



Eind scenario:



Lees eerst Theorie 4.5: Conditie-gestuurde herhalingsplan.

Lees eerst Theorie 4.6: Generieke strategieën.

Opgave 4.4: Ga naar bestemming

Taak: *Laat Mimi naar een bepaalde bestemming lopen.*
Hiervoor schrijven we een methode `void goToLocation( int coordX, int coordY )` die Mimi naar de cel met coördinaten (`coordX`, `coordY`) stuurt.

a) Bedenk een geschikt hoog-niveau algoritme en schets een stroomdiagram. Bepaal welke submethodes je nodig hebt. Tips:

- Vraag Mimi's coördinaten op en bepaal dan hoeveel stappen ze in welke richting moet zetten om op haar bestemming te komen.
- Het kan handig zijn om verschillende gevallen apart te bekijken, afhankelijk van welke kant ze uit moet lopen. Overweeg om in de specificatie van jouw algoritme woorden als 'WEST', 'OOST', 'NOORD' en 'ZUID' te gebruiken.
- Het kan handig zijn om een aparte submethode `boolean locationReached( int coordX, int coordY )` te schrijven om te controleren of de bestemming bereikt is.
- Controleer of jouw stroomdiagram verbeterd kan worden kan door gebruik te maken van een `while`.

b) Schrijf de bijbehorende code. Tips:

- Je kunt `setDirection( WEST );` gebruiken om Mimi naar het westen te laten draaien. Een andere richting gaat op een vergelijkbare manier.
- Je kunt `getX( )` en `getY( )` gebruiken om de coördinaten van Mimi op te vragen.

c) Voeg (JavaDoc) commentaar toe.

d) Test jouw methode door deze met de rechtermuisknop aan te roepen en verschillende waarden in te voeren. Probeer grensgevallen zoals $(0, 0)$ en $(11, 11)$. Controleer op ongeldige waarden zoals $(14, 14)$ en $(-3, 1)$.

e) Pas jouw programma aan zodat het ongeldige waarden juist afhandelt:

i) Schrijf een aparte submethode `boolean validCoordinates(int coordX, int coordY)` die controleert of de gegeven coördinaten geldig zijn. De coördinaten moeten in Mimi's wereld liggen.

ii) Bij ongeldige coördinaten toon je de volgende foutmelding: `showError( "Invalid coordinates");`.

iii) Bij ongeldige invoer moet Mimi niks doen; ze blijft staan.

iv) Jouw programma moet voor elk wereldgrootte werken, niet alleen voor een wereld van 12 bij 12. Vraag de breedte en hoogte van de wereld op. Tip: De breedte van de wereld krijg je door eerst de wereld op te vragen: `World world = getWorld( );` en daarna de breedte op te vragen: `world.getWidth( )` (zie Theorie 4.6).

f) Test deze aanpassing.

We hebben nu gezien hoe je variabelen kunt gebruiken en vergelijken. Ook hebben we gezien hoe je parameters, variabelen en getallen kunt combineren om ingewikkelde condities te maken.

Opgave 4.5: Draai naar het oosten met een `while`

Taak: *Schrijf een methode* `void faceEastUsingWhile( )` *die Mimi naar het oosten toe draait.*
Maak hierbij gebruik van een `while` (in plaats van een `if`). Tip: Je kunt gebruik maken van de methode `int getDirection( )` om na te gaan in welke richting Mimi kijkt.
We hebben nu gezien hoe je een variabele en een getal kunt vergelijken. Daarnaast heb je een `while`-loop gebruikt om stappen te herhalen.

Lees eerst Theorie 4.7: Telplan.

Lees eerst Theorie 4.8: Teller-gestuurde herhalingsplan.

Opgave 4.6: Tellers in een `while`-loop

We gaan nu oefenen met tellervariabelen in een `while`. Net als bij de eerste taak (taak 4.1) gaan we stukken code bekijken en nalopen wat er met de variabelen gebeurt. Dit geeft veel inzicht in hoe de `while` werkt. Daardoor zal je minder snel fouten maken.

Bekijk de volgende stukken code en maak gebruik van een tracing tabel om antwoord te geven op de vragen. Gebruik de methode `void practiceTracingCode ( )` die je bij taak 4.1 gemaakt hebt om jouw antwoorden te controleren.

a) Wat zijn na afloop de waarden van `teller` en `aantalStappenOmTeZetten`? Hoe vaak wordt `move( );` aangeroepen? Is de code goed?

```
/**
 * Dodo loopt aantalStappenOmTeZetten vooruit
 */
int aantalStappenOmTeZetten = 6;
int teller = 0;

while ( teller <= aantalStappenOmTeZetten ) {
    move();
    teller++;
}
```

b) Vul de tracing tabel aan voor elke keer dat de body van de `while` wordt uitgevoerd.

```
int getal1 = 2;
int getal2 = 5;
int teller = 0;

while( getal1 <= getal2 ){
   move();
   getal1 ++;
   teller ++;
}
```

Aantal `while`-loops uitgevoerd	Waarde `getal1` na `while`	Waarde `getal2` na `while`
0		
1		
2		

c) Pas de onderstaande **conditie** van de `while` aan zodat `move( )` drie keer wordt aangeroepen:

```
int getal1 = 10;
int getal2 = 8;

while( getal1 > getal2 ){
    move();
    getal1--;
}
```

d) Bekijk de methode `jump` uit Theorie 4.8.

i) Kopieer en plak de methode `void jump(int distance)` in jouw `MyDodo` code.

ii) Geef voor elk van de volgende gevallen aan hoe vaak de code in de `while` (van `void jump(int distance)`) uitgevoerd wordt als:

i. `distance` gelijk is aan 5.
ii. `distance` gelijk is aan 1.
iii. `distance` gelijk is aan 0.
iv. `nrStepsTaken` gelijk is aan 3 en `distance` aan 5.

Tip: Print een paar waarden naar de console.

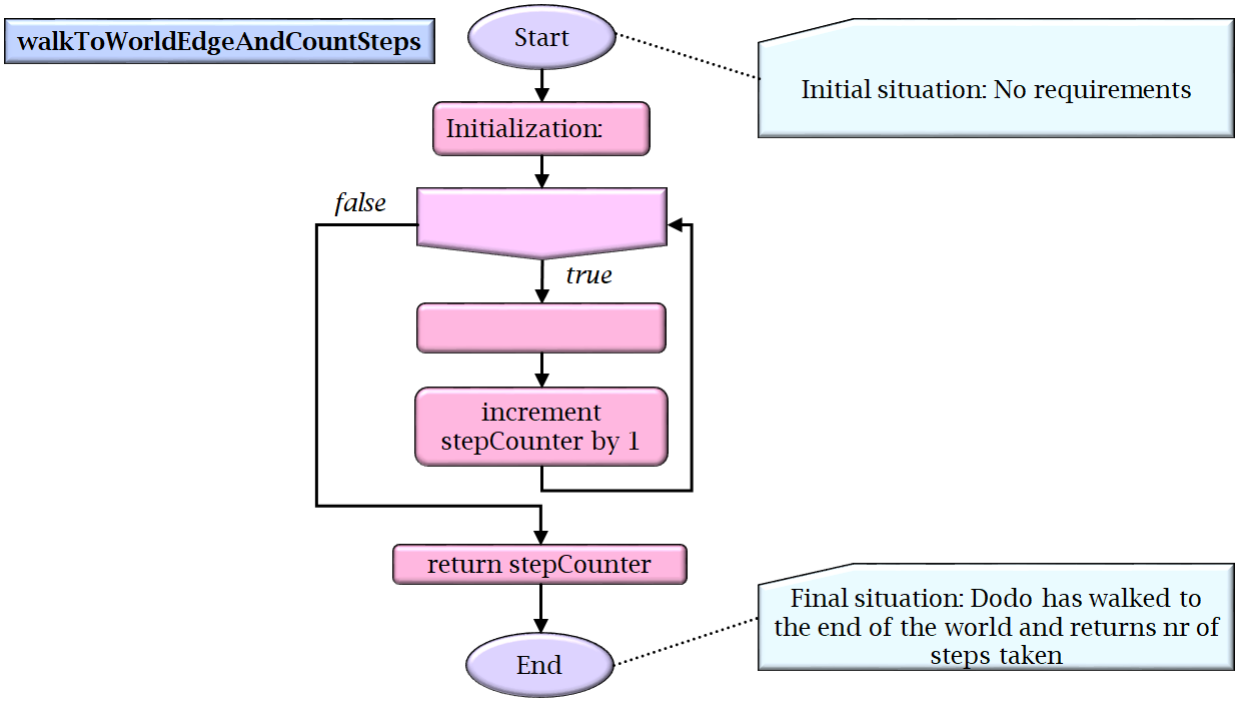
Je hebt nu gezien hoe je een teller kunt gebruiken om te bepalen hoe vaak code (bijvoorbeeld de body van een `while`-loop) uitgevoerd moet worden om de juiste resultaten te krijgen.

Opgave 4.7: Aantal stappen tot het einde van de wereld

Bij deze taak zul je het *telplan* gebruiken.

Taak: *Schrijf een methode die telt hoeveel stappen Mimi moet zetten om vanaf haar huidige positie tot het einde van de wereld te komen. Lever het resultaat op.*

Bekijk Theorie 4.7 voor meer informatie over het plan. Het volgende stroomdiagram en de bijbehorende code laten een raamwerk zien voor de methode `walkToWorldEdgeAndCountSteps( )` die gebaseerd is op het plan:



Stroomdiagram voor het tellen van het aantal stappen tot het einde van de wereldfig:flowchartWalkToWorldEgde

```
public int walkToWorldEdgeAndCountSteps(){
   int stepCounter = 0; // counter variable
   while (           ) {
```

```
        stepCounter ++;
    }
    return stepCounter;
}
```

a) Declareer een tellervariabele (vergeet de initialisatie niet).

b) Binnen de `while`-loop, pas je de teller aan.

c) Aan het einde van de methode lever je de teller op met een `return`-statement.

d) Test jouw methode.

e) Wat is het kleinste getal dat deze methode kan opleveren? En het grootste?

We hebben nu zelf een teller-variabele gebruikt om bij te houden hoe vaak een `while`-loop doorlopen wordt. We hebben ook een methode geschreven die de waarde van de variabele oplevert zodat deze in een ander deel van je programma gebruikt kan worden.

Opgave 4.8: Aantal eieren tot de rand van de wereld tellen

Taak: *Schrijf een accessormethode `int countEggsInRow` die oplevert hoeveel eieren er in een rij of kolom liggen.*

Om dit te doen laat je Mimi vooruit lopen (tot ze de rand van de wereld bereikt) en het aantal eieren tellen die ze tegenkomt. Hiervoor gebruik je een `while` en een teller-variabele. Omdat dit een accessormethode is (die alléén informatie oplevert) moet Mimi wel terug naar haar oorspronkelijke positie (beginsituatie gelijk aan eindsituatie). Aan het einde van de methode gebruik je een `return`-statement om het aantal eieren als resultaat op te leveren.

a) Kies een geschikte variabelenaam om het aantal gevonden eieren bij te houden.

b) Teken het bijbehorende stroomdiagram. Lever onderaan het aantal gevonden eieren als resultaat op. Tip: De `return` is de allerlaatste instructie die aangeroepen kan worden: alles wat daarna aan instructies volgt zal nooit worden uitgevoerd. Mimi moet dus eerst terug naar haar oorspronkelijke positie voordat de waarde als resultaat opgeleverd kan worden. Voor het teruglopen kun je een submethode aanroepen die je in taak 2.8 al geschreven hebt.

c) Schrijf de bijbehorende methode. Vergeet niet het JavaDoc commentaar toe te voegen.

d) Test jouw methode door deze met de rechtermuisknop aan te roepen. Test ook een situatie met een ei in de eerste of laatste cel van de rij.

We hebben nu zelf een variabele gebruikt om bij te houden hoeveel eieren er gevonden zijn.

Opgave 4.9: Leg een spoor eieren

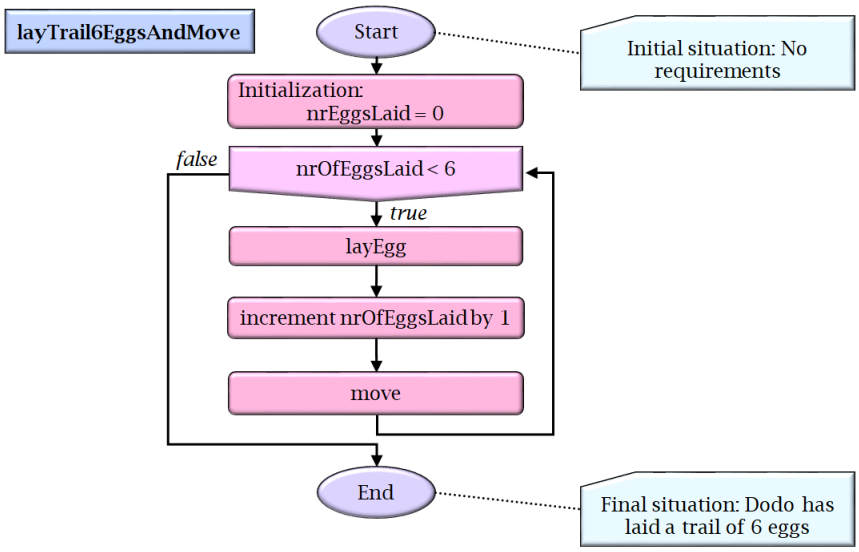
Taak: *Mimi maakt een spoor van zes eieren.*

De begin- en eindsituaties zijn als volgt:

Figuur 8: Beginsituatie

Figuur 9: Eindsituatie: Mimi heeft een spoor van zes eieren achtergelaten en staat achter het laatste ei

Schrijf een methode `void layTrailOf6EggsAndMove( )` die overeenkomt met het stroomdiagram in figuur 13.



Figuur 10: Stroomdiagram voor methode `layTrailOfEggsAndMove( )`

We hebben nu een teller-variabele (`nrOfEggsLaid`) gebruikt om te bepalen hoe vaak een `while`-loop doorlopen moet worden.

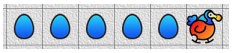
Opgave 4.10: Leg een spoor van eieren, zonder te ver te lopen

Als je de methode uit taak 4.9 goed test kom je een probleem tegen: Mimi kan het spoor van eieren niet helemaal tot de grens leggen: de laatste cel blijft leeg. We gaan dit nu oplossen.

Taak: *Mimi maakt een spoor van zes eieren en blijft na afloop op het laatste ei staan.*

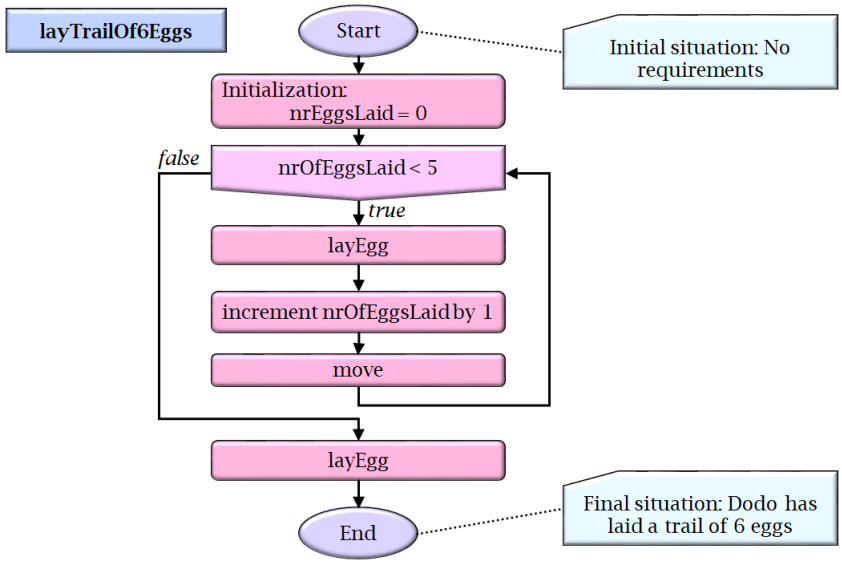
De begin- en eindsituaties zijn als volgt:

Figuur 11: Beginsituatie



Figuur 12: Eindsituatie: Dodo heeft een spoor van zes eieren gelegd en staat op het laatste ei

Schrijf een methode `void layTrailOf6EggsAndMove( )` die bij het stroomdiagram in figuur 13 hoort.



Figuur 13: Stroomdiagram voor methode `layTrailOfEggs( )`

We hebben nu een teller-variabele (`nrOfEggsLaid`) gebruikt om te bepalen hoe vaak een `while`-loop doorlopen moet worden.

Opgave 4.11: Leg een spoor van *n* eieren

We willen Mimi leren om een spoor van *n* (een bepaald aantal) eieren te maken. Dit lijkt op `layTrailOf6Eggs ( )` (taak 4.10) waar Mimi zes eieren moest leggen. Het verschil is dat we haar nu niet van tevoren vertellen hoeveel eieren ze precies moet gaan leggen.

Taak: *Schrijf een methode* `void layTrailOfEggs( int nrOfEggsToLay )` *waarmee Mimi een spoor van* `nrOfEggsToLay` *legt.* Aan het einde moet ze bovenop haar laatste ei staan. Voeg commentaar toe aan jouw code en test jouw methode.

We hebben nu een generieke oplossing gemaakt die voor meerdere problemen gebruikt kan worden.

Opgave 4.12: Leg een spoor van *n* eieren, controleer op invoer

We willen weer dat Mimi een spoor van *n* eieren legt. Deze keer controleren we op ongeldige waarden. Dit zijn waarden kleiner dan nul of zo groot dat Mimi uit de wereld zou stappen.

Taak: *Ga jouw methode* `layTrailOfEggs( int nrEggsToLay )` *uitbreiden zodat die ook werkt bij ongeldige invoer.*

Voeg commentaar toe aan jouw code en test jouw werk. Controleer dat jouw oplossing werkt ongeacht de kant waarop Mimi kijkt.

Je hebt nu gezien hoe je een methode makkelijk kan uitbreiden met andere eisen. Daarnaast heb je geleerd om ongeldige waarden af te vangen.

Reflectie

In deze opdracht heb je variabelen gebruikt om informatie bij te houden. Je hebt ook geleerd over herhalingsloops en hoe je methodes en parameters moet gebruiken.

Als je ergens echt goed in wilt worden, is een van de meest belangrijke stappen om even terug te blikken op wat je deed en hoe dat ging:

Resultaat

Ik weet dat mijn oplossing werkt omdat . . .
Ik ben trots op mijn oplossing omdat . . .
Ik kan mijn oplossing verbeteren door . . .

Aanpak

Mijn aanpak was goed omdat . . .
Wat ik volgende keer beter anders kan doen is . . .

Geef met een smiley aan hoe het ging.

	Ik kan het
	Het is me een beetje gelukt maar ik begreep het niet helemaal
	Ik snapte er helemaal niks van

	Ik kan variabelen gebruiken om data in op te slaan.
	Ik kan een tracetabel gebruiken om code te analyseren.
	Ik kan plannen toepassen voor tellen, herhalingen en voor het verwisselen van waarden.
	Ik kan aangeven hoe vaak bepaalde code herhaald moet worden door een teller of een conditie te gebruiken.
	Ik weet hoe ik methodes met parameters kan gebruiken.

Opslaan en inleveren

Sla je werk op. Je hebt het nodig voor de volgende opdrachten.

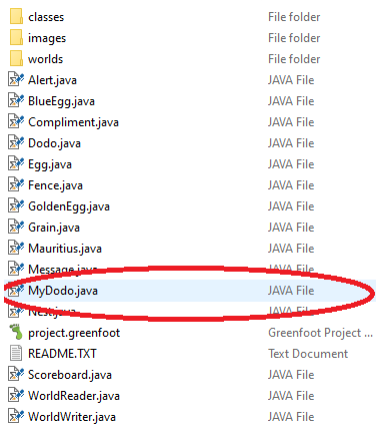
Opslaan

Selecteer ’Scenario’ in het Greenfoot menu, en dan ’Save’. Alle bestanden die bij het scenario horen zitten nu in één map.

Inleveren

Lever het volgende in:

- Naam: van jou (en je partner);
- Jouw code: Het `MyDodo.java` bestand;
- Stroomdiagrammen:
  - Als je ze op papier hebt gemaakt: plak (foto’s van) jouw stroomdiagrammen in een (Word) bestand.
  - Als je software gebruikt hebt: sla het bestand op als .pdf of .jpg.



Figuur 14: Het `MyDodo.java` bestand in bestandsbeheer