Opdracht 6: Dodo wordt blijvend slimmer

– Algoritmisch Denken en Gestructureerd Programmeren in Greenfoot –

©2015 Renske Smetsers-Weeda & Sjaak Smetsers

Op dit werk is een creative commons licentie van toepassing.

https://creativecommons.org/licenses/by/4.0/

1 Inleiding

In de vorige opgave heb je geleerd om gebruik te maken van variabelen. Daardoor kon je Mimi dingen laten onthouden waardoor ze slimmer werd.

Je hebt daarbij lokale variabelen gebruikt. Elke keer als je een methode waarin zo'n variabele gedeclareerd is opnieuw aanroept (direct of indirect via `act`) worden die variabelen opnieuw geïnitialiseerd en zijn ze hun 'oude' waardes kwijt. Als je een ingewikkelder programma of een spelletje wilt maken wil je ook waardes gedurende het hele programma kunnen bijhouden. Daarvoor gebruik je een ander soort variabele, een *instantievariabele*. Hoe je die gebruikt leer je in deze opdracht. In opdracht 3 hadden we afgeproken dat we, als dat niet al te ingewikkeld is, liever de **`while`**-loop van *Run* gebruiken dan zelf een **`while`**-loop in de `act`-methode te zetten. Oftewel, we proberen `act` steeds zo te definiëren dat hierin één stap van ons algoritme wordt uitgevoerd. We zullen zien dat, door het gebruik van instantievariabelen, je jouw programma's zo kunt schrijven dat je **`while`**-loops kunt vermijden.

2 Leerdoelen

Na het voltooien van deze opdracht kun je:

- uitleggen waarvoor **instantievariabelen** gebruikt worden;
- in eigen woorden uitleggen wat het verschil is tussen een instantievariabele en een lokale variabele;
- de stappen benoemen en uitvoeren die nodig zijn voor het gebruik van een instantievariabele;
- instantievariabelen **initialiseren**;
- de constructor van een klasse herkennen;
- in eigen woorden uitleggen wat de rol van een **constructor** is;
- in eigen woorden uitleggen wat de rol is van **`private`** en **`public`** bij information hiding en modularisatie;
- redeneren over de **zichtbaarheid** van variabelen en methoden;
- in eigen woorden uitleggen wat de rol is van **getter- en settermethodes**.

3 Instructies

Bij deze opdracht ga je verder met jouw code uit opdracht 5. Je hebt dus het scenario nodig dat je na opdracht 5 hebt opgeslagen `Opdr5_jouwNaam`. Als eerste gaan we hiervan een kopie maken zodat je altijd terug kan naar jouw oorspronkelijke bestand:

- Open het scenario dat je na opdracht 5 hebt opgeslagen `Opdr5_jouwNaam`.

- Kies in Greenfoot 'Scenario' in het bovenste menu, en dan 'Save As ...'.
- Controleer dat je in de map staat waar je jouw werk wilt opslaan.
- Vul de bestandsnaam aan met jouw eigen naam en het opgavenummer, bijvoorbeeld: `Opdr6_Michel`.

Je zult ook een paar vragen beantwoorden. Vragen met een '(IN) ' moeten 'IN'geleverd worden. Voor die vragen heb je nodig:

- pen en papier om de stroomdiagrammen te tekenen die ingeleverd moeten worden ('(IN) '),
- een document (bijvoorbeeld, Word) open waarin je de antwoorden voor de '(IN) ' vragen kunt typen.

De overige vragen (die niet ingeleverd hoeven te worden) moet je met jouw programmeerpartner bespreken en jullie antwoord kort noteren op het opgavenblaadje.

Een opmerking vooraf: in deze opdracht mogen alléén de klassen `MyDodo` en `Egg` worden aangepast.

4 Uitleg

Zichtbaarheid
In (de code van) klassen wordt de zichtbaarheid/toegankelijkheid van methoden en instantievariabelen (verdere uitleg volgt hieronder) aangegeven. De mogelijkheden voor de zichtbaarheid zijn:

Zichtbaarheid	Beschrijving
public	toegankelijk van buitenaf buiten de klasse
private	alléén toegankelijk vanuit de klasse zelf
protected	alléén toegankelijk vanuit de klasse zelf en zijn subklasses

Private elementen zijn niet zichtbaar voor andere klassen. En een element dat gedefiniëerd is als **`public`**, is toegankelijk overal binnen het programma.

Bij objectgeoriënteerd programmeren is het de bedoeling dat je per klasse een aantal methoden kiest die **`public`** zijn. Deze kunnen andere objecten in andere klassen aanroepen om informatie op te vragen of om een object iets te laten doen. Een voorbeeld hiervan is de methode **`int`** `nrEggsHatched()` waarmee je aan Mimi vraagt hoeveel eieren ze uitgebroed heeft. Ook kun je haar verzoeken een ei uit te broeden **`void`** `hatchEgg()`. Objecten van andere klassen weten dus wat Mimi allemaal kan en kunnen bepaalde publieke accessor- of mutatormethodes aanroepen.

Maar *hoe* Mimi dat precies doet is aan haarzelf. De implementatie van de methode is private. Dus wat Mimi allemaal precies moet doen om een eitje uit te broeden, daar heeft een ander object uit een andere klasse niks mee te maken. Soms wil je uit beveiligings- of privacy-oogpunt niet dat anderen alle details weten over hoe iets precies gebeurt. Dit principe wordt *information hiding* genoemd. Dit principe zegt dat de interne informatie over de implementatie van een klasse onzichtbaar moet zijn voor andere klassen. Dit principe verhoogt de modularisatie van jouw programma.

Tot nu toe hebben we voor het gemak alle methodes public gemaakt. Dat is eigenlijk niet zo netjes. Het idee bij object oriëntatie is juist om de kennis en controle bij een object zelf te

laten. Dat is veiliger. Zo dwing je af dat een ander object aan Mimi vraagt of ze een ei wil afstaan, in plaats van dat een ander object zomaar eieren van Mimi kan afpakken.

Instantievariabelen
Instantievariabelen zijn een soort 'geheugen' van het object. Ze bevatten gegevens die door een object worden opgeslagen en gebruikt. Een instantievariabele is een soort variabele: net als elke andere variabele heeft deze een type en een naam, bijvoorbeeld `int myNrOfEggsHatched`. Ook gelden dezelfde naamgevingsafspraken. Instantantievariabelen worden ook wel eens *attributen* of *velden* genoemd.

Instantievariabelen behoren tot het object zelf. We spreken daarom af dat we de zichtbaarheid altijd op **`private`** zetten (principe van *information hiding*). Als een ander object de waarde van een instantievariabele wil weten, kan dit aan het object gevraagd worden door een public accessormethode aan te roepen.

Declareren
Een instantievariabele wordt in een klasse gedeclareerd. Zoals gezegd maken we deze **`private`**. Hier geven we de variabele een type en een naam. Bijvoorbeeld: `private int myNrOfEggsHatched`. Hiermee houdt een MyDodo bij hoeveel eieren ze uitgebroed heeft. Omdat deze **`private`** is, weet alleen zij hoeveel het er zijn en kan alleen zij dit aantal wijzigen.

Initialiseren
We geven deze een initiële waarde in de constructor (zie hiervoor het volgend theorieblok). Als het object wordt aangemaakt, dan krijgt de variabele meteen deze waarde. Bijvoorbeeld als een nieuwe MyDodo in de wereld wordt gezet, dan heeft deze nog geen eieren uitgebroed. In de constructor geven we dit aan met: `myNrOfEggsHatched` = 0.

Levensduur:
Een instantievariabele heeft dezelfde levensduur als het object waartoe deze behoort. Dus zodra een object wordt gecreëerd (met **`new`**) en in de constructor geïnitialiseerd, dan worden ook alle instantievariabelen die het object heeft, aangemaakt. De levensduur van object en instantievariabelen is hetzelfde.

Getter accessormethode
Met een public accessormethode (een zogehegeten *gettermethode*) kan een ander object de waarde van een **`private`** instantievariabele opvragen. Het is trouwens niet zo dat elke instantievariabele een **`public`** gettermethode heeft. Getters hoeven niet per se gemaakt te worden. Deze worden alleen beschikbaar gesteld als dit echt zinvol is.

Naamgevingsafspraken van een gettermethode
Hoe je een gettermethode noemt is afhankelijk van het resultaat dat die methode oplevert. Indien die een

- **`boolean`** als resultaat oplevert is het gebruikelijk om 'is....' te kiezen, bijvoorbeeld `boolean isHatched();`
- in alle andere gevallen kies je een naam als 'get...', bijvoorbeeld `int getNrOfEggsHatched()`.

Waarbij op '...' de variabelenaam komt.

Voorbeeld code voor een gettermethode
Om te vragen hoeveel eieren uitgebroed zijn:

```
public int getNrOfEggsHatched( ) {
    return myNrOfEggsHatched( );
}
```

Omdat de methode **public** is, kan een ander object met `getNrOfEggsHatched()` opvragen hoeveel eieren er zijn uitgebroed. Een ander object kan verder niets aan de waarde veranderen. Dat zou hij wel kunnen doen als we, tegen onze afspraak in, **int** `myNrOfEggsHatched` public hadden gemaakt.

Setter mutatormethode
Met een public mutatormethode (een zogehegeten settermethode) kan een ander object de waarde van één of soms zelfs meerdere **private** instantievariabelen (laten) aanpassen.

Tip: Wees zuinig met settermethodes. Voeg ze alleen aan een klasse toe als ze echt zinvol zijn. Dat is veiliger want zo worden de variabelen beter beschermd.

Voorbeeld code voor een settermethode:
Stel een `MyDodo` heeft een instantievariabele **private int** `eggsToHatch` waarmee Mimi bijhoudt hoeveel eieren ze moet uitbroeden. Alléén zij weet hoeveel het er zijn. Omdat de eieren wellicht door een slang opgegeten worden, kan ze de boodschap krijgen (van de slang) dat ze één ei minder hoeft uit te broeden.

De volgende settermethode kan door een ander object aangeroepen worden om de variabele van `MyDodo` aan te passen:

```
public void setOneEggLessToHatch( ) {
    eggsToHatch--; // waarde met 1 verminderen
}
```

Omdat de methode `setOneEggLessToHatch()` **public** is, kan een ander object met `setOneEggLessToHatch()` de waarde van de variabele `eggsToHatch` met ééntje verlagen.

Stappenplan gebruik instantievariabelen
Om een object 'geheugen' te geven maak je gebruik van instantievariabelen. Hiervoor moet je een aantal dingen doen:

- **Declareren** bovenin de klasse (aanmaken geheugenplekje): zichtbaarheid (als regel maken we deze private), type en naam aangeven.
- **Initialiseren** (initiële waarde toekennen) in de constructor (zie hiervoor de volgende theorieblok): zodra een object wordt aangemaakt krijgen de opgegeven instantievariabelen de opgegeven waardes.
- (optioneel) Een **getter accessormethode** maken zodat een object uit een andere klasse de waarde van de variabele kan opvragen;
- (optioneel) Een **setter mutatormethode** maken zodat een object uit een andere klasse de waarde van de variabele kan aanpassen;
- De waarde **aanpassen** in de code. Bv, de variabele **int** `myNrOfEggsHatched` moet telkens opgehoogd worden zodra een ei is uitgebroed.

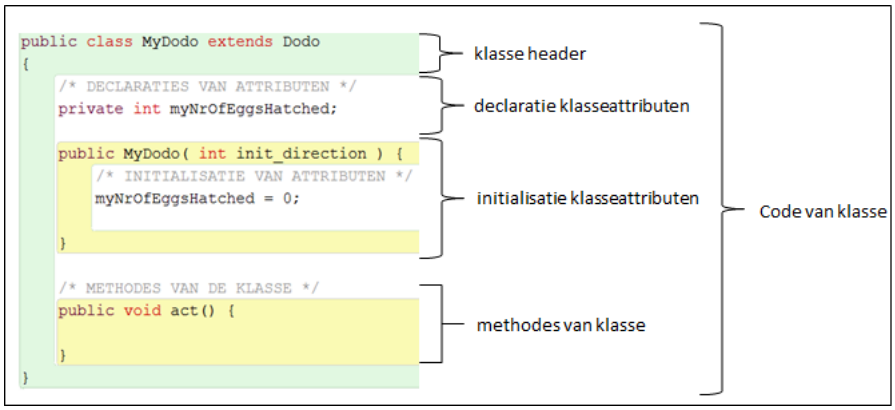
Toelichting: Als je niet wilt dat andere objecten gegevens kunnen opvragen of aanpassen, dan maak je geen getter- en settermethodes.

Constructor
De *constructor* is een speciale methode die wordt aangeroepen zodra een object wordt aangemaakt (met javnew). In een constructor geef je de instantievariabelen van een object de juiste beginwaarden. De constructor lijkt veel op een normale methode, op twee eigenschappen na:

- de naam van de constructor is altijd gelijk aan de klasse zelf;
- en de constructor heeft nooit een return-type, ook niet **`void`**.

In Greenfoot roep je de constructor van `MyDodo` aan door met de rechtermuisknop in de klassediagram te klikken en **`new`** `MyDodo()` te selecteren. In de code ziet dat er zo uit: `MyDodo mimi = new MyDodo ();`

Voorbeeld:
Open de klasse `MyDodo` in de editor. Deze klasse heeft één instantievariabele (**`private int`** `myNrOfEggsHatched`) en een aantal methoden waarvan één constructor.



5 Opgaven

5.1 (Instantie)variabelen

1. In opdracht 5 opgave 5.1.2 'Draai naar het oosten' heb je een tabel ingevuld met **`int`** `myDirection` en **`int`** `getDirection( )`. Inmiddels heb je het een en ander geleerd over **`private`** variabelen en gettermethodes. Leg uit wat de relatie is tussen **`int`** `myDirection` en **`int`** `getDirection( )`.
2. Waar herken je een constructor aan?
3. Leg in jouw eigen woorden uit wat het verschil is tussen een instantievariabele en een lokale variabele.
4. Waar worden getter- en settermethodes voor gebruikt?

5.2 Eieren tellen tot aan het hek

We laten Mimi tot een hek lopen. Ondertussen telt ze hoeveel eitjes ze tegenkomt. We maken hierbij gebruik van instantievariabelen. Hierdoor kunnen we
het tellen stapsgewijs laten uitvoeren en dus de **`while`**-loop van *Run* gebruiken.

1. Open 'world_layEggTrailUntilFence'.

2. Bedenk een geschikte naam voor een instantievariabele om het aantal gevonden eieren bij te houden.
3. Bedenk een algoritme waarbij Mimi tot aan het hek loopt en ondertussen telt hoeveel eieren ze tegenkomt. Elke keer als ze een ei vindt, wordt het aantal gevonden eieren in een console getoond.
4. Teken het bijbehorende stroomdiagram.
5. We schrijven hiervoor een methode **public void** `walkToFenceAndCountEggs()`. Open de code voor `MyDodo`. Vergelijk de code in de editor met het stuk hieronder. Tik datgene over wat nog niet in de code staat.

```
public class MyDodo extends Dodo {
    // HIER DE INSTANTIEVARIABELE DECLAREREN

    public MyDodo (int init_Direction) {
        // HIER DE INSTANTIEVARIABELE INTIALISEREN

    }
    public void walkToFenceAndCountEggs() {
        if (!fenceAhead() ) {
            move();
            if( foundEgg() ) {
                // HIER DE INSTANTIEVARIABELE MET 1 VERHOGEN

                // toon console
                System.out.println("I found " + VARIABELE_NAAM + " eggs!");
            }
        }

    }
}
```

6. Vervang al het commentaar in hoofdletters door de bijbehorende code.
7. Voeg commentaar toe aan de methode.
8. Roep de methode **void** `walkToFenceAndCountEggs()` aan vanuit de methode `act()`.
9. Compile, run and test jouw methode met *Run*.
10. Wat zie je gebeuren? Tip: Er opent nog een Greenfoot venster, dit zit mogelijk verscholen achter een ander venster.
11. Doet het programma wat je verwacht?

We hebben nu gezien dat Mimi kan bijhouden hoeveel eieren ze gevonden heeft met een instantievariabele die geïnitialiseerd wordt bij het creëren van Mimi (in de constructor).

In opdracht 5 hebben we gebruik gemaakt van een **while**-loop en een lokale variabele voor het tellen van de eieren. In dat geval kun je de **while**-loop niet zomaar weglaten en vervangen door de **while** van *Run*. De`act`-methode wordt dan telkens weer opnieuw aangeroepen en bij elke aanroep wordt de lokale variabele opnieuw aangemaakt en geïnitialiseerd (in dit geval op 0 gezet). De oude waarde gaat daarbij steeds verloren. Daardoor kan Mimi de tel niet bijhouden. Wat dus wél werkt is een instantievariabele gebruiken. Deze wordt immers alleen bij de creatie op 0 gezet en blijft daarna steeds z'n waarde behouden ook tussen verschillende aanroepen van `act` door. Dit maakt het niet langer noodzakelijk om de herhaling in `act` zelf te zetten, maar plaats te laten vinden in *Run*.

5.3 Bijhouden hoeveel stappen er gezet zijn

Mimi is niet zo slim. Ze kan nog net onthouden hoeveel eieren ze vandaag uitgebroed heeft, maar dat was het dan. We gaan Mimi ietsje slimmer maken door haar ook te laten bijhouden hoeveel stappen ze gezet heeft.

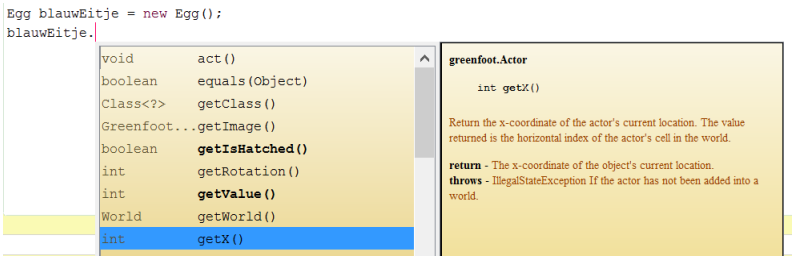
1. Open de code voor de klasse `MyDodo`.
2. Zoek in de code naar de plek waar instantievariabelen van deze klasse gedeclareerd worden. Bekijk de instantievariabel **`int`** `myNrOfEggsHatched`.
   (a) Wat is het type van die instantievariabele?
   (b) Er staat **`private`** voor, wat betekent dat?
   (c) Waar wordt er voor het eerst een waarde aan toegekend?
   (d) Wanneer wordt deze waarde aangepast?
   (e) Heb je inmiddels ook de constructor kunnen vinden? Hoe herken je die?
3. Voeg op een geschikte plek de **`private`** instantievariabele `nrOfStepsTaken` toe.
4. Wat lijkt jou een geschikte initiële waarde? Dus als Mimi voor het eerst in de wereld wordt zet, wat is dan de waarde van `nrOfStepsTaken`? Leg uit.
5. Geef de instantievariabele de juiste initiële waarde. Tip: Doe dit in de constructor.
6. Mimi kan nu bijhouden hoeveel stappen ze gezet heeft. We voegen nu een gettermethode toe. Hiermee krijgt een ander object de mogelijkheid om te vragen hoeveel stappen ze gezet heeft. Verzin een geschikte naam voor deze methode. Implementeer en test deze.
7. Als Mimi een stapje zet, dan moet `nrOfStepsTaken` verhoogd worden. Het verhogen met één doe je door `nrOfStepsTaken++;`. Waar kunnen we dat het beste doen? Je kunt dat natuurlijk na elke aanroep van `move` doen. Je kan het ook na elke aanroep van `step` doen. Beredeneer welke van de twee mogelijkheden de beste is.
8. Ga in de code van `MyDodo` op zoek naar waar `step` aangeroepen wordt. Zorg ervoor dat na elke `step` aanroep de variabele `nrOfStepsTaken` met één verhoogd wordt.
9. Test jouw aanpassingen.
10. Leg uit waarom we geen settermethode maken.

Een methode aanroepen uit andere klasse
Bij het aanroepen van methoden onderscheid je altijd twee gevallen:

1. De methode komt uit je eigen klasse (of de klasse waarvan je erft). In dit geval kun je de methode 'rechtstreeks' aanroepen (zoals je al gewend bent).
2. De methode behoort tot een andere klasse. Om dan zo'n methode aan te kunnen roepen heb je een object van de klasse waaruit de methode komt nodig. Stel `object` is zo'n object en `methode` een methode uit de klasse van `object`. De juiste aanroep van deze methode is `object.methode( );`, ervan uitgaande dat `methode` geen parameters heeft. Je kunt zo'n aanroep als volgt interpreteren: aan `object` wordt gevraagd zijn methode `methode` uit te voeren.

Voorbeeld van een methode aanroep vanuit een andere klasse:
Stel je hebt een `BlueEgg` object genaamd `blauwEitje`. Mimi wil weten op welke coördinaat het eitje ligt. `BlueEgg` heeft een methode **`int`** `getX( )` (geërfd uit `Actor`) die de x-coördinaat van het ei oplevert. De x-coördinaat van `blauwEitje` kan door Mimi opgevraagd door: `blauwEitje.getX( );`. Deze aanroep zelf staat dan in de klasse `MyDodo`.

Toevoeging:
Tip: Gebruik 'Ctrl+Spatie' na het intikken van het objectnaam en '.' (zoals `blauwEitje.` in het voorbeeld hierboven) om een lijst te krijgen van alle methodes van dat object die je kunt aanroepen.



Figuur 1: Voor een lijst van beschikbare methodes: gebruik 'Ctrl+spatie'

5.4 Ei uitbroeden

Een uitgebroed ei verdwijnt uit de wereld. Mooier zou zijn als het ei niet zomaar verdwijnt, maar getoond wordt als een uitgebroed ei.

Figuur 2: Een uitgebroed ei

We moeten hiervoor ons scenario op een aantal plaatsen wijzigen. We denken eerst na over welke aanpassingen nodig zijn. De aanpassingen in de code doen we samen, stap-voor-stap, in de volgende opgave.

1. Het ei moet onthouden of het is uitgebroed of niet. Hiervoor voegen we een **instantievariabele** aan het ei toe: **`private boolean`** `iAmHatched`.
2. Nieuwe methodes toevoegen om de toestand van de instantievariabele `iAmHatched` op te vragen of aan te passen:
   (a) Een **gettermethode** waarmee aan het ei gevraagd kan worden of deze al wel of niet is uitgebroed:

   ```
   public boolean isHatched( )
   ```

   (b) Een **settermethode** waarmee aan het ei doorgegeven wordt dat het uitgebroed is:

   ```
   public void setHatched( )
   ```

   Hierin moet het volgende gebeuren:
   i. de **instantievariabele aanpassen**, zodat `iAmHatched` **`true`** is;
   ii. het **plaatje** van een ei moet vervangen worden door een uitgebroed ei.

3. Er voor zorgen dat Mimi de getter- en settermethodes aanroept als ze een ei wil uitbroeden.

Deze wijzigingen voegen we stapsgewijs toe:

1. Allereerst gaan we de klasse `Egg` aanpassen en wel zodanig dat het `Egg`-object zelf bijhoudt of het ei al is uitgebroed of niet. We maken eieren dus slimmer door ze zelf hun toestand bij te laten houden. We gebruiken hiervoor een **`boolean`** instantievariabele `iAmHatched`.

   (a) Open de code voor de klasse `Egg`.

   (b) Voeg op een geschikte plek de **`private boolean`** instantievariabele toe met de naam `iAmHatched`.

   (c) Wat lijkt jou een geschikte initiële waarde? Dus als een ei voor het eerst in de wereld wordt gezet, wat is dan de waarde van **`boolean`** `iAmHatched`? Leg uit.

   (d) Geef de instantievariabele de juiste initiële waarde. Dit doe je in de constructor.

   De eigenschap "uitgebroed zijn" is nu aan de klasse toegevoegd.

2. We voegen methodes toe om de toestand van de instantievariabele `iAmHatched` op te vragen of aan te passen:
   We breiden nu de klasse `Egg` met een gettermethode en een settermethode:

   (a) **Gettermethode**: We voegen nu de mogelijkheid toe om aan een ei te vragen of het is uitgebroed of niet. Als er een ander object (bijvoorbeeld Mimi) zou willen weten of dat ei wel of niet uitgebroed is, dan moet Mimi dat aan het ei vragen.

      i. Maak een nieuwe methode **`public boolean`** `isHatched( )` in de klasse `Egg`.

      ii. Deze levert de waarde van `iAmHatched` op.

      iii. Test jouw methode.

   (b) **Settermethode**: We voegen nu de mogelijkheid toe om de toestand van het ei van 'niet uitgebroed' naar 'uitgebroed' te wijzigen. Ook vervangen we het plaatje door een plaatje met een uitgebroed ei, want dat hoort bij de toestand.

      i. Maak een nieuwe methode **`void`** `setHatched( )` in de klasse `Egg`.

      ii. De methode moet de waarde van de instantievariabele **`boolean`** `iAmHatched` aanpassen.

      iii. Als het ei uitgebroed is moet een ander plaatje getoond worden (het rode uitgebroede ei in plaats van het blauwe). Hiervoor kun je het volgende stukje code gebruiken:

      ```
      GreenfootImage hatched_egg = new GreenfootImage("egg_hatched.png");
      setImage(hatched_egg);
      ```

      Toelichting: In de map 'images' vind je de plaatjes terug die in dit scenario gebruikt worden. Een van die plaatjes heet 'egg_hatched.png'. Met de twee instructies die hierboven gebruikt worden koppel je dit plaatje aan een Greenfoot klasse. Je mag ook zelf op internet een leukere afbeelding zoeken. Kopiëer die dan naar de images map om het daarna te gebruiken in je scenario.

      iv. De methode `setHatched( )` is een mutatormethode. Hoe herken je dat?

      v. Compileer en test jouw code. Met de rechtermuisknop kun de methode `setHatched( )` aanroepen en daarna met 'Inspect' de waarden van instantievariabele te bekijken. Werkt jouw code zoals verwacht?

   Nu hebben we in stap 1 aan `Egg` een **`private`** instantievariabele `iAmHatched` toegevoegd. Daar hebben we in stap 2 **`public`** getter- en settermethodes voor geschreven. Hiermee kunnen andere objecten (zoals Mimi) de waarde opvragen of het ei verzoeken de waarde aan te passen.

3. We zorgen ervoor dat Mimi de getter- en settermethodes aanroept als ze een ei wil uitbroeden.
   Onze eieren zijn nu slimmer geworden. Ze weten of ze uitgebroed zijn of niet. Maar het is Mimi die de eieren daadwerkelijk uitbroedt.

   - Als zij een ei wil uitbroeden, dan moet ze eerst aan het ei vragen of dit niet al uitgebroed is. Mimi vraagt dit door de gettermethode `boolean isHatched( )` van het gevonden ei aan te roepen.
   - Als zij het ei daarna uitbroedt, dan moet ze dit aan het ei vertellen. Mimi roept hiervoor de settermethode `setHatched( )` van het gevonden ei aan. Het ei past dan zelf het plaatje en zijn instantievariabele `iAmHatched` aan.

   Op het eerste gezicht lijkt dit allemaal wat vreemd, maar dit is dé manier waarop dit in de objectgeoriënteerde wereld gebeurt: objecten kunnen allerlei dingen, maar doen die niet uit zichzelf. Ze moeten hiervoor expliciet de opdracht krijgen.

   We passen de code aan zodat Mimi aan het ei doorgeeft dat het uitgebroed is.

   (a) Open de klasse `MyDodo`. Zoek de methode `hatchEgg( )` op. Omschrijf wat de methode nu met een ei doet.
   (b) Gebruik eerst `Egg eggFound = getEgg( );` om het ei-object te krijgen. Met deze instructie maak je een lokale variabele aan die een `Egg`-object kan bevatten. Deze variabele krijgt als initiële waarde het ei dat door `getEgg()` wordt opgeleverd. Daarna kun je pas methoden van het gevonden ei aanroepen, waarvoor je dan de variable `eggFound` gebruikt.
   (c) Voordat Mimi een ei uitbroedt moet ze eerst controleren of dat nog niet uitgebroed is. Voeg een controle toe met `eggFound.isHatched()` voordat ze dit uitbroedt.
   (d) Pas het commentaar bij de methode aan.
   (e) Compileer en test jouw code. Wat levert `isHatched( )` op bij een uitgebroed ei? En bij een niet uitgebroed ei?
   (f) Om nu de toestand van het ei te veranderen moet Mimi voor het gevonden ei de settermethode `void setHatched( )` aanroepen. Het aanroepen doe je dan met `eggFound.setHatched( );`.
   (g) Pas het commentaar bij de methode aan.
   (h) Compileer en test jouw code. Wat gebeurt er als je met de rechtermuisknop `setHatched( )` aanroept bij een niet uitgebroed ei? En bij een ei dat al wel is uitgebroed?

5.5 Ga naar een locatie (met instantievariabelen)

In opdracht 5 opgave 'Ga naar een bepaalde locatie' heb je de methode

```
void goToLocation( int coordX, int coordY)
```

geschreven die Mimi naar een bepaalde locatie met coördinaten (coordX, coordY) stuurt. Jouw methode maakt gebruik van een **while**. In deze opdracht gaan we een methode schrijven die hetzelfde doet, maar die geen **while** gebruikt: het herhalen van stappen gebeurt in *Run*.

Om dit voor elkaar te krijgen maken we gebruik van instantievariabelen.

1. Bekijk de methode `goToLocation` die je in opdracht 5 opgave 'Ga naar een bepaalde locatie' geschreven hebt.
2. Declareer in de `MyDodo` klasse twee nieuwe instantievariabelen: `private int coordX` en `private int coordY`.
3. Schrijf een settermethode:

```
public void setDestination( int locationCoordX, int locationCoordY )
```

die aan `coordX` en `coordY` de gegeven waarden in de parameters toekent. Hierbij is het handig om te controleren of de opgegeven cooördinaten geldig zijn. Verder hebben we geen gettermethodes nodig.

4. Als `goToLocation` aangeroepen wordt, naar welke concrete locatie wil je dat Mimi naar toe gaat? Kies een bestemming en initialiseer de instantievariabelen in de constructor door `setDestination` aan te roepen.

5. Pas de code van `goToLocation` als volgt aan:

   (a) De methode `goToLocation` krijgt geen parameters meer mee. Pas dit in de signatuur aan.

   (b) Vervang de **`while`** door een **`if`**.

   (c) Ook de submethode die je geschreven hebt om te controleren of Mimi al op de gewenste locatie staat moet aangepast worden. Deze krijgt geen parameters meer mee. Pas dit zowel bij de aanroep vanuit `goToLocation` aan, als in de signatuur van de submethode zelf.

   (d) Loop de methode langs om te controleren of het juist is.

6. Roep `goToLocation()` aan vanuit de `act`.

7. Schrijf commentaar bij de aangepaste methode.

8. Compileer en test jouw methode met *Run*.

9. Roep vervolgens met de rechtermuisknop `setDestination` aan, en druk daarna op de *Run*. Wat gebeurt er?

10. Open nu de `Madagaskar` klasse en zoek hierin de `act` methode op. Hierin staat een aanroep van `setDestination`, maar die is nu nog uitgecommentarieerd. Haal het commentaar weg. Compileer en test jouw scenario met *Run*. Wat gebeurt er? Klik op een willekeurig vakje in de wereld. Wat zie je? Kun je dit verklaren?

5.6 Bijhouden hoeveel eieren er gevonden zijn (met instantievariabelen) (A)

In opdracht 4, opgave 5.4 'Eitjes volgen tot het nest' heb je een methode geschreven `followEggTrailUntilNest()` die een pad van eieren tot het nest volgt. Net als bij de vorige opgave gaan we de methode aanpassen zodat deze gebruik maakt van instantievariabelen.

1. Bedenk een instantievariabele waarmee Mimi bijhoudt hoeveel eieren ze onderweg tegenkomt.

2. Pas jouw code aan zodat deze instantievariabele steeds wordt bijgewerkt.

3. Test jouw code.

4. Als het nest gevonden is, toon in een console de volgende regel: "Je hebt een pad van xx eitjes gevolgd om bij het nest te komen". In plaats van 'xx' geef je aan hoeveel eieren ze gevonden heeft.

5. Test de code opnieuw.

6 Samenvatting

Je hebt geleerd:

- instantievariabelen te gebruiken om waardes (blijvend) bij te houden;
- wat een constructor is;
- hoe je met behulp van instantievariabelen een **`while`**-loop in `act` door een **`if`** kunt vervangen.

7 Jouw werk opslaan

Je bent klaar met de zesde opdracht. Sla je werk op, want je hebt het nodig voor de volgende opdrachten.

1. Kies in Greenfoot 'Scenario' in het bovenste menu, en dan 'Save As ...'.
2. Vul de bestandsnaam aan met jouw eigen naam en het opgavenummer, bijvoorbeeld: `Opdr6_Michel`.

Alle onderdelen van het scenario bevinden zich nu in een map die dezelfde naam heeft als de naam die je hebt gekozen bij 'Save As ...'.