Informatica

Objectgeörienteerd leren programmeren

Van de theorie met BlueJ tot een spelletje met Greenfoot...

Bert Van den Abbeele

Objectgeörienteerd leren programmeren

Van de theorie met BlueJ tot een spelletje met Greenfoot...

door

Bert Van den Abbeele

Inhoudsopgave

Inleiding

Objectgeörienteerd programmeren sluit aan bij het menselijke denkpatroon. Deze methode van programmeren maakt gebruik van (voor)gedefinieerde bewerkingen van objecten.

Programmeertalen waarmee object-georiënteerd geprogrammeerd kan worden is bijvoorbeeld C++ en JAVA. Deze mogelijkheden bevinden zich in headers, bibliotheken (libraries), APIs (Application Programming Interfaces) enz. waar de mogelijke bewerkingen die de programmeur kan uitvoeren beschreven staan.

Een klasse is de basis binnen het object georiënteerd programmeren. Je kan een klasse vergelijken met een kookboek. Daarin kan je lezen hoe een recept vorm krijgt. Men omschrijft de methode hoe je tewerk moet gaan en de eigenschappen van het gerecht liggen vast. Ons programma heeft een klasse die vertelt wat acties en eigenschappen aanwezig moeten zijn om tot een bepaald resultaat te komen (= algoritmen).

In deze cursus leren we de basis van Object Oriented Programming (= object-georiënteerd programmeren). We bekijken van dichterbij wat een klasse en een object is. De belangrijkste principes van OOP komen aan bod zoals overerving, methoden en encapsulatie.

We maken gebruik van de gratis programma's BlueJ (http://www.bluej.org/) en Greenfoot (http://www.greenfoot.org/). Beide applicaties zijn beschikbaar voor de besturingssystemen Linux, MacOS en Windows.

In deze cursus worden ook UML diagrammen weergegeven. Het tekenen van deze diagrammen wordt niet stapsgewijs overlopen. Het programma dat hiervoor gebruikt wordt is Umbrello met Linux (http://uml.sourceforge.net/) of StarUML met Windows (http://staruml.sourceforge.net/en). Een ander programma waarmee je deze diagrammen kan tekenen is Dia (http://projects.gnome.org/dia/ en http://dia-installer.sourceforge.net/).

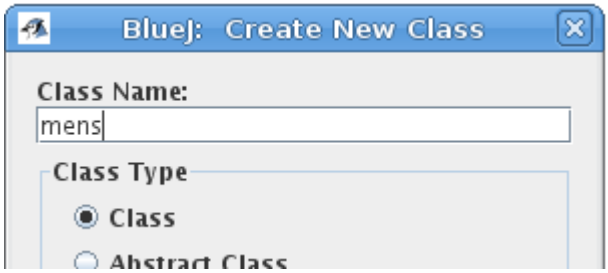
Veel succes!

Bert Van den Abbeele

De klasse

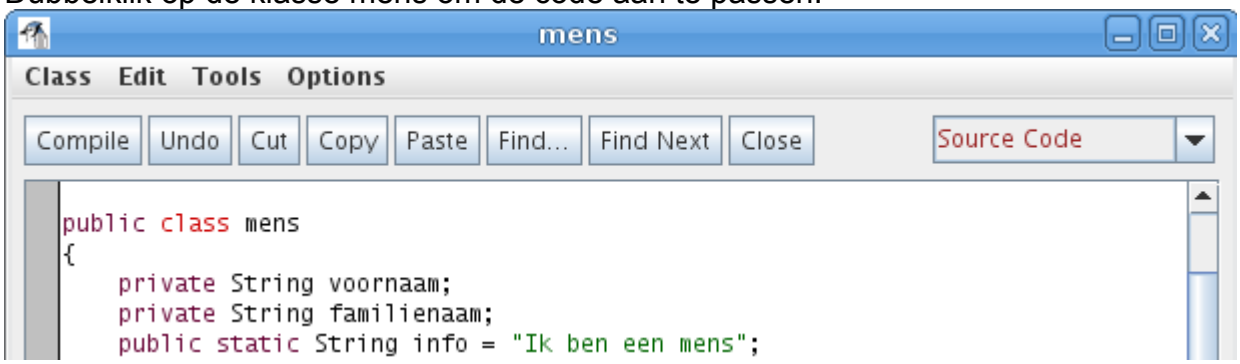
Een **klasse** is een model waarvan objecten afgeleid zullen worden. Zo kunnen we een klasse “mens” maken waarin vaststaat dat een mens een voornaam en familienaam heeft.

We maken in **BlueJ** een nieuw project aan via Project > New Project... Geef het project een locatie.
We maken vervolgens een nieuwe klasse “New Class...” 



In ons voorbeeld maken we een klasse “mens” met drie **variabelen**: voornaam, familienaam en info. Deze variabelen kunnen we ook **eigenschappen**, **attributen** of **properties** noemen.

Dubbelklik op de klasse mens om de code aan te passen.




```
public class mens
{
     private String voornaam;
     private String familienaam;
     public static String info = "Ik ben een mens";
}
```

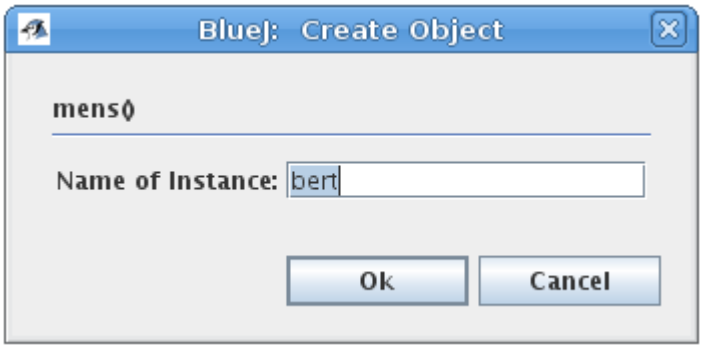
Compileer de aanpassingen: Tools > Compile.

De variabelen zijn “private” omdat ze enkel toegankelijk mogen zijn binnen de klasse, dit in tegenstelling tot een variabele die “public” is. Naast **private** of **public** kan een variabele ook **protected** zijn. Deze laatste is beschikbaar binnen dezelfde klasse, ervan afgeleide klassen en het pakket (**package**) waartoe het behoort.
Attributen die enkel nut hebben voor de interne werking van de klasse en waar afgeleide objecten eigenlijk geen boodschap aan hebben kunnen dus verborgen gehouden worden. Deze attributen (data) vormen de kern van de klasse en zijn enkel bereikbaar via een methode (zie verder). Dit is het principe van **encapsulation** (=data binding).

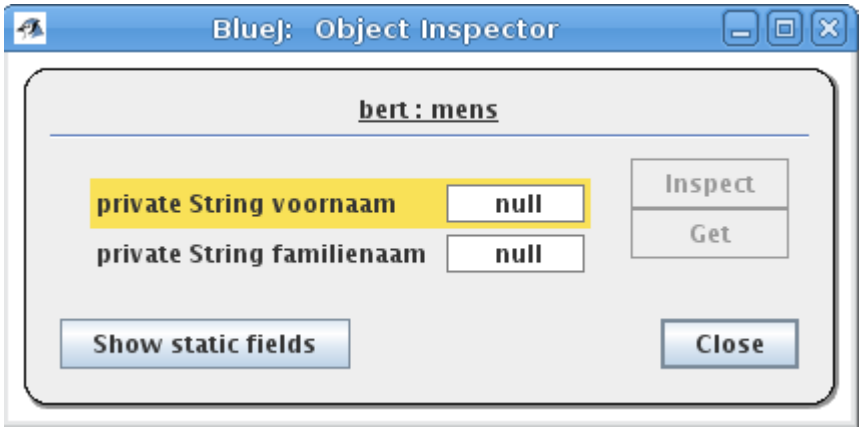
Een object

Een **object** is een **instantie** van een klasse. Het kader gemaakt in de klasse wordt overgenomen en we krijgen een object waarop bepaalde handelingen uitgevoerd kunnen worden.

In BlueJ klikken we met een rechtermuistoets op onze klasse en klikken op new mens(). Geef het object je eigen voornaam.



Dubbelklik op het object en bekijk de **instantievariabelenen** of eigenschappen (properties) van het object.



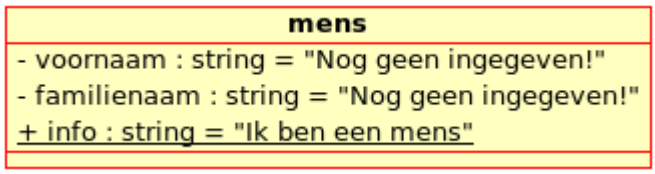
Klik ook eens op: 

Merk op dat er geen voornaam of achternaam is ingesteld. Indien we wel een naam willen meegeven moeten we de klasse aanpassen en opnieuw compileren. Hier voegen we een **constructor** mens() toe aan de klasse.

```
public class mens
{
     private String voornaam;
     private String familienaam;
```

```
	public static String info = "Ik ben een mens";

	public mens()
	{
		voornaam= "Nog geen ingegeven!";
		familienaam= "Nog geen ingegeven!";
	}
}
```

Onze klasse kunnen we in een schema weergeven. Zo'n schema heeft een **UML** diagram:



Een methode

De naam wijzigen was tot nu toe niet mogelijk omdat we hiervoor geen mogelijkheid gelaten hebben. We gaan nu een functionaliteit toevoegen aan de klasse die ervoor zorgt dat men een instantie van de klasse (= het object) een voornaam en familienaam kan geven. Hiervoor maken we een **methode**. Het principe van encapsulatie zorgt ervoor dat we enkel via een methode de kern van de klasse kunnen aanpassen (=de eigenschap of attribuut).

```
public void veranderVoornaam(String ingave)
{
	voornaam=ingave;
}
```

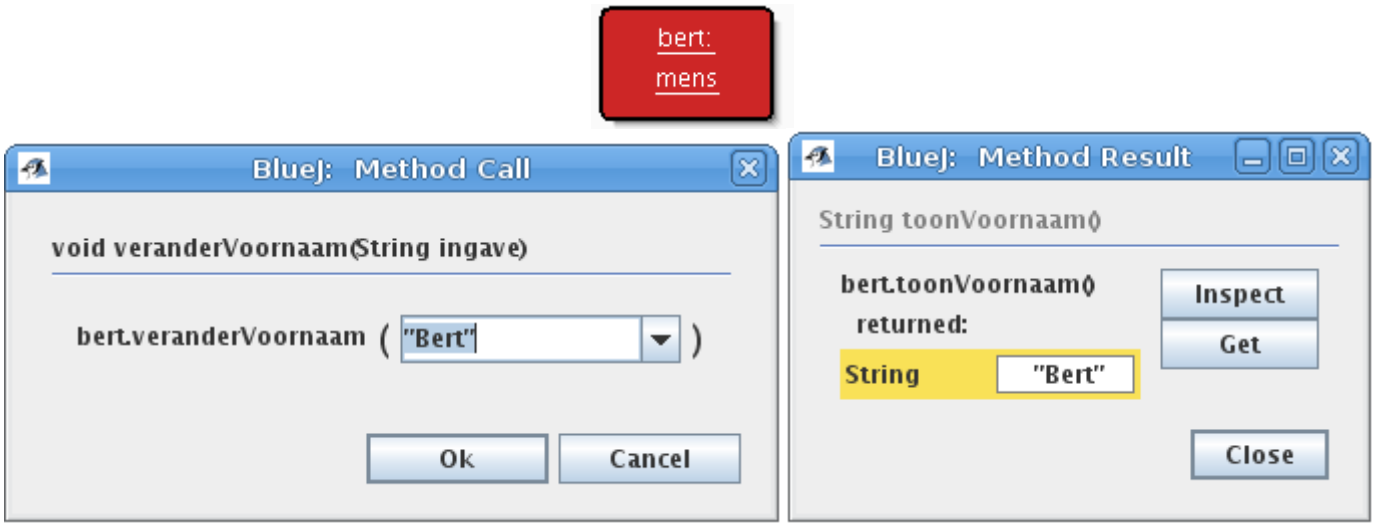
Deze methode is public dus vanuit andere klassen toegankelijk vanuit andere klassen. Het woord **void** betekent dat er geen waarden terug gegeven worden. Er is wel een parameter verwacht, de ingave.

Een volgende methode is het weergeven van de waarden.

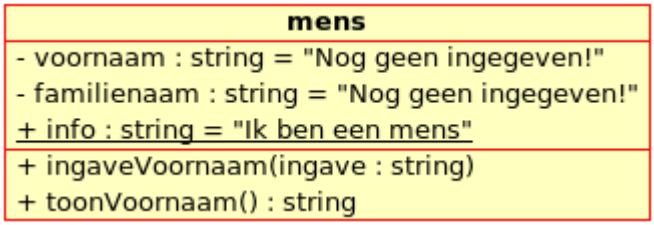
```
public String toonVoornaam()
{
     return voornaam;
}
```

Hier worden geen parameters verwacht maar wel een waarde teruggegeven, daarom is void veranderd in **string** (het gegevenstype van de uitvoer).

Indien we nu met een rechtermuistoets op het object klikken kunnen we de methoden uitvoeren.



Het UML diagram:



Opdracht: Maak ook methodes aan voor weergave en ingave van familienaam.

Overloading

Indien we bij initialisatie van het object onmiddellijk parameters wensen toe te voegen moeten we de declaratie van mens() aanpassen naar mens(String voornaam, String achternaam).

```
public mens(String voornaam, String achternaam)
{
}
```


```
public mens()
{
     voornaam= "Nog geen ingegeven!";
     familienaam= "Nog geen ingegeven!";
}
public mens(String ingaveVoornaam, String ingaveAchternaam)
{
     voornaam= ingaveVoornaam;
     familienaam= ingaveAchternaam;
}
```

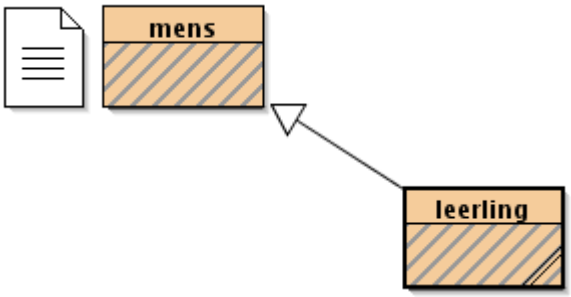
Het is perfect mogelijk om zowel de constructor mens() en mens(String voornaam, String achternaam) in de klasse te plaatsen. Dit geeft ons de mogelijkheid om al dan niet te kiezen de parameters bij construtie mee te geven.

Overerving

Het **overerven** maakt mogelijk dat we onze nieuwe klasse gaan baseren op een andere klassen en hierop verderbouwen. Zo kunnen we een klasse leerling maken waarin vaststaat dat de leerling een mens is dus een voornaam en achternaam heeft maar ook een studiejaar.

We maken vervolgens een nieuwe klasse “New Class...” 

Geef deze nieuwe klasse de naam “leerling”.
Voeg vervolgens een overerving toe van mens naar leerling. 



We noemen de klasse “leerling” een subklasse van “mens”. De klasse “mens” is de superklasse van “leerling”.
Dubbelklik op “leerling”, in de declaratie van de klasse lezen we de overerving:
public class leerling extends mens.

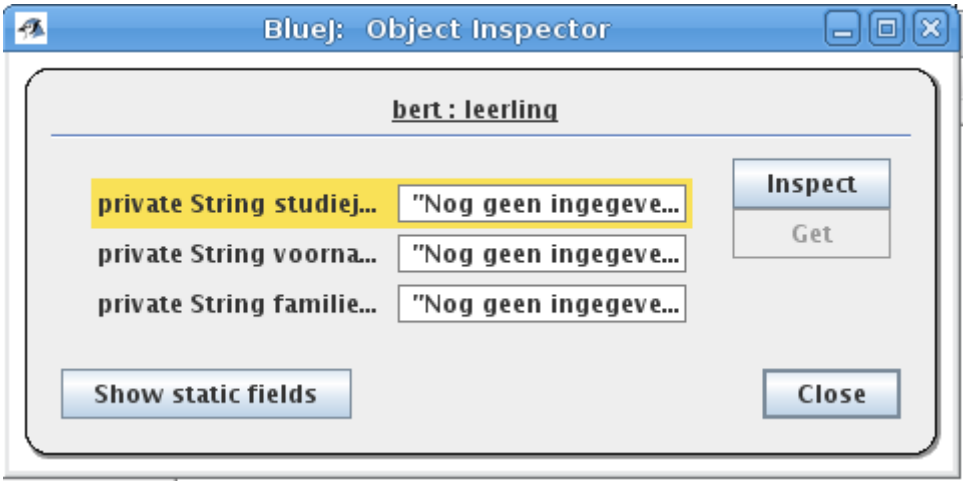
We voegen enkele methoden toe aan deze klasse.

```
public class leerling extends mens
{
     private String studiejaar;
     public static String info = "Ik ben een leerling";

public leerling()
{
     studiejaar= "Nog geen ingegeven!";
}

public void veranderStudiejaar(String ingave)
{
     studiejaar=ingave;
}

public String toonStudiejaar()
{
     return studiejaar;
}
}
```

Compileer de aanpassingen: Tools > Compile.

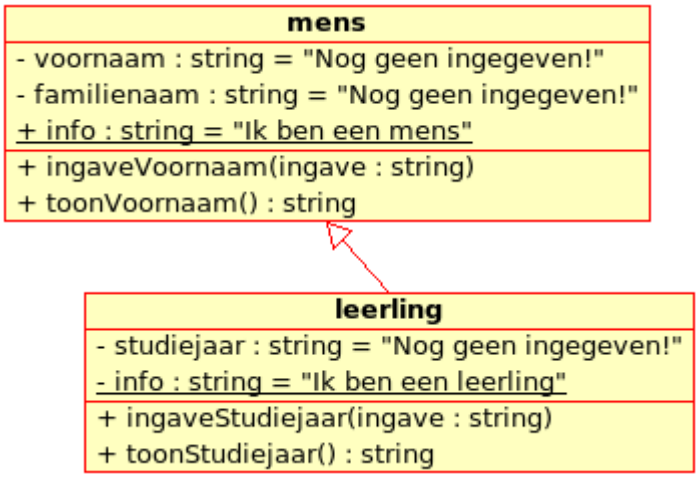
We klikken met een rechtermuistoets op onze klasse en klikken op new leerling(). Geef het object je eigen naam.

Dubbelklik op het object en bekijk de instantievariabelenen of eigenschappen (properties) van het object.



Klik ook eens op: 

Het UML diagram:

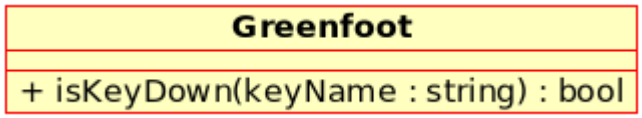


Van theorie naar praktijk

Je krijgt in deze verwerkingsopdracht over objectgeörienteerd een aantal klassen ter beschikking. We bekijken eerst de verschillende klassen waarop wij ons programma gaan baseren. Vervolgens programmeren we instanties van deze klassen om zo een spel te bekomen.

Vanaf dit punt maken we gebruik van **Greenfoot**.

Package en klasse Greenfoot

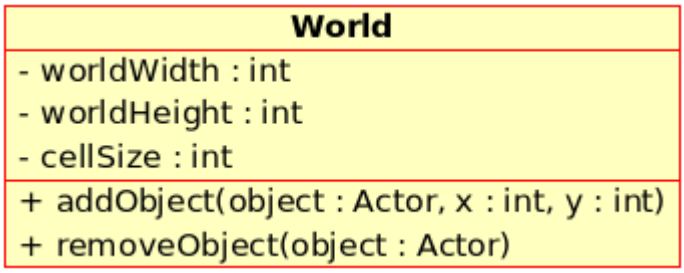


We maken gebruik van een aantal voorgedefinieerde klassen. Het voordeel is dat we gebruik kunnen maken van hun functionaliteit. We moeten niet weten hoe dit juist gebeurt, we moeten enkel de attributen (properties) en methoden juist aanroepen. Dit is het principe van **polymorfisme**.

De klasse Greenfoot met dezelfde naam als het pakket waarmee we werken is het raamwerk waarop onze objecten geplaatst zullen worden. Er is één methode in deze klasse aanwezig die belangrijk is voor deze oefening. De methode isKeyDown(KeyName : string) : bool controleert of een bepaalde toets ingedrukt is.

Indien we isKeyDown(“left”, “right”, “up” of “down”) aanroepen, wordt er 0 (false) of 1 (true) teruggegeven. De methode kan maar twee waarden terugggeven 0 (false) of 1 (true), vandaar het gegevenstype boolean.

De klasse Word (achtergrond)



Het spelletje dat we gaan programmeren heeft als speelveld een achtergrond nodig. Deze achtegrond heeft een aantal parameters:

- De breefte: worldWidth
- De hoogte: worldHeight
- De celgrootte: cellSize

Indien we aan onze achtergrond opbouwen met World(100,100,25) creëeren we een wereld van 100 bij 100 pixels met cellen van 25 bij 25 px groot.
Onze achtergrond heeft ook twee interessante methoden:

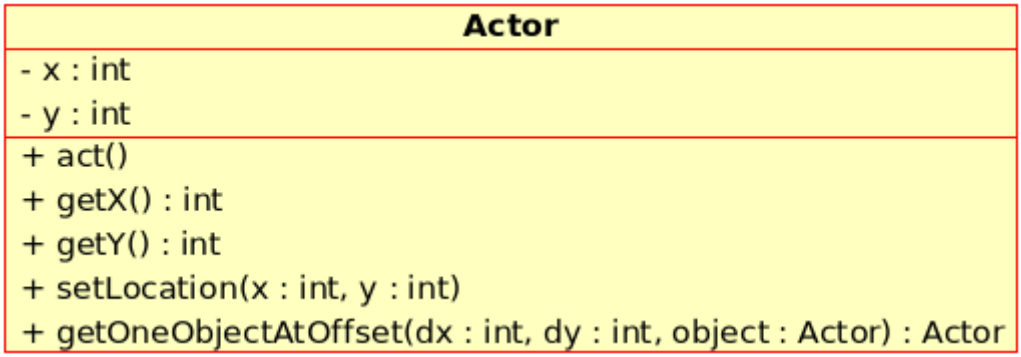
- addObject(object : Actor, x : int, y :int)

Bij het begin van ons spel zullen een aantal spelers op de achtergrond geplaatst moeten worden. Elke speler heeft een positie uitgedrukt in pixels (x en y) ten opzichte van de rechterbovenhoek.

- removeObhect(object:Actor)

Heeft een speler verloren, dan zal deze ook verwijderd moeten worden van het spelveld.

De klasse Actor



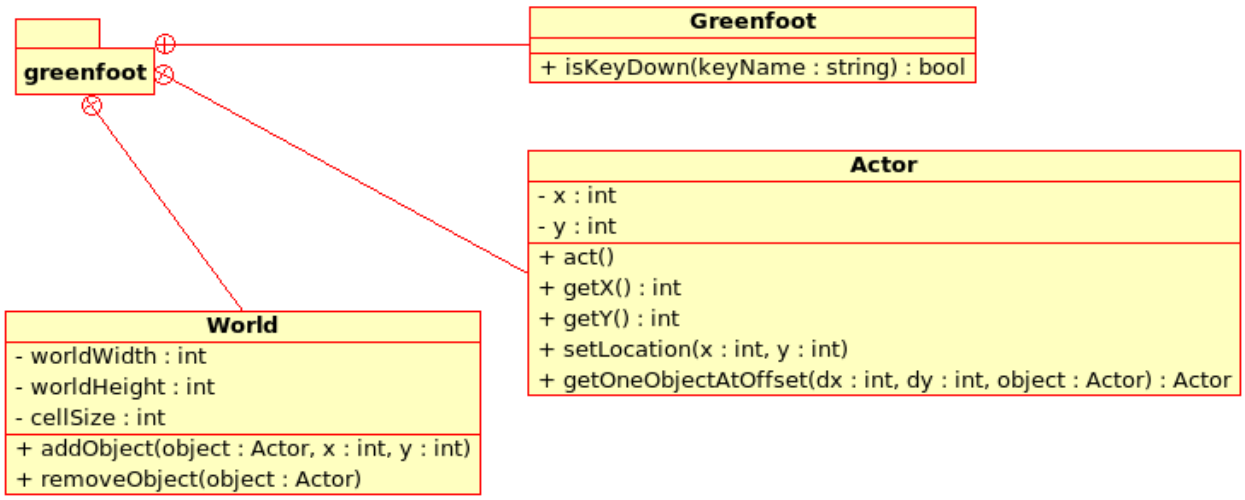
Hiermee definiëren we de spelers van ons spel. De actor heeft twee gehele getallen x en y als parameter. Dit is de huidige positie van onze speler.

De waarde van deze parameters kunnen we opvragen met de methoden getX() en getY().

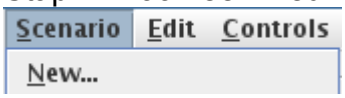
Om de actor te laten bewegen moeten we hem van plaats kunnen veranderen, daarvoor dient de methode setLocation(x :int, y :int).

Bij een spel is het belangrijk dat we te weten komen of twee objecten tegen elkaar aanlopen. De methode getOneObjectAtOffset(dx: int, dy :int, object:Actor) contoleert of er een andere Actor op de positie dx:dy aanwezig is.

De objecten definiëren

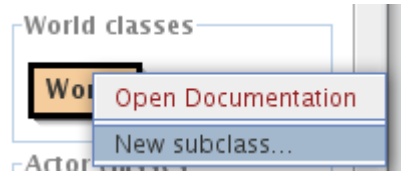


Stap 1: Maak een nieuw scenario: Scenario > New...

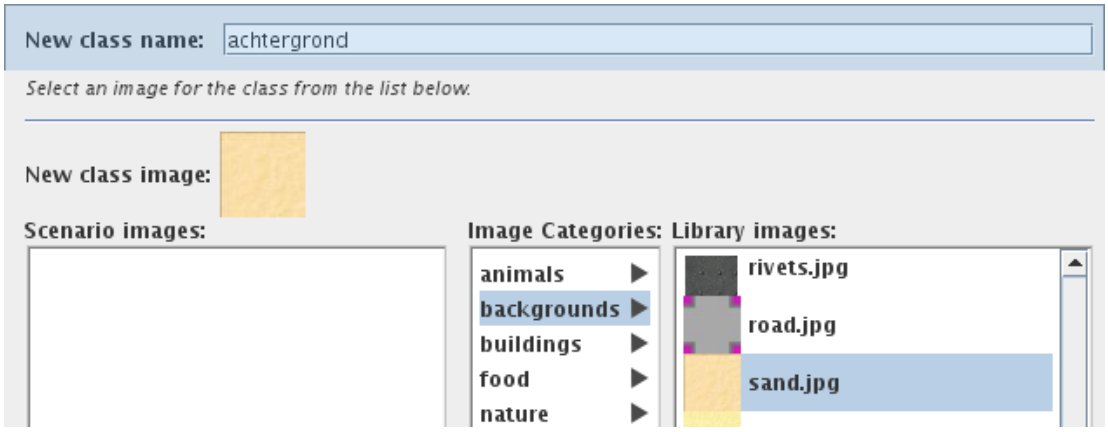


Geef je scenario de naam “Oefening zeehond” en een logische locatie...

Stap 2: Maak een nieuwe instantie van de klasse World: World > New subclass...



Geef de subklasse de naam “achtergrond” en kies een achtergrond (bvb.: sand.jpg).



Stap 3: Geef een implementatie aan de subklasse “achtergrond”.

We wensen dat onze achtergrond uit 60 bij 60 pixels bestaat met een celgrootte van 10px.

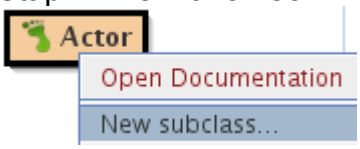
Hiervoor passen we de constructor van de klasse achtergrond aan (dubbelklik op de klasse achtergrond om de editor te openen):

```
public achtergrond()
{
     super(60, 60, 10);
}
```

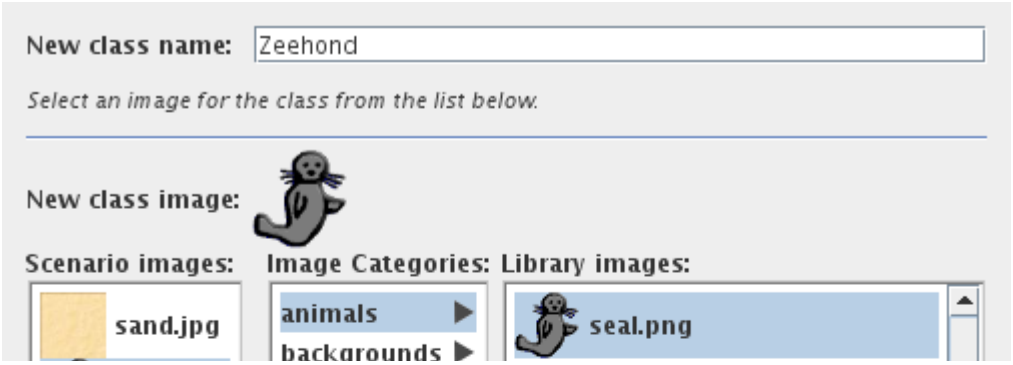
De methode **super()** geeft parameters door aan de klasse waaruit de subklasse ontstaan is, in dit geval de “parent” World. We moeten in onze klasse achtergrond geen eigenschappen aanmaken voor breedte, hoogte en celeigenschappen, deze eigenschappen heeft hij overgeërfd van de “parent”. Vandaar dat we de parameters doorspelen aan de klasse World.

Nu moeten we de code nog compileren om de aanpassingen te zien. Klik op “Compile all”:

Stap 4: We maken een nieuwe instantie van de klasse Actor:Actor > New subclass...



Geef de subklasse de naam “Zeehond” en kies een afbeelding (bvb.: seal.png).



Onze zeehond is onze hoofdspeler. Onze actor heeft een methode act(), hierin plaatsen we de acties die de speler onderneemt, deze worden continue doorlopen in de gameloop.

Hij moet kunnen bewegen. Hiervoor gaan we eerst de huidige x en y positie opvragen via de methoden getX() en getY().

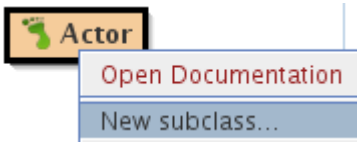
Vervolgens moeten we controleren of er een toets is ingedrukt. Indien dit het geval is moeten we de x en y positie aanpassen zodat de zeehond in beweging komt. Hiervoor gebruiken we de methode isKeyDown(Key :string) uit de klasse Greenfoot.

Hebben we onze zeehond op dezelfde plaats dan een vis gezet dan eet de zeehond de vis op. Hiervoor roepen we de methode getOneObjectAtOffset van de huidige positie op. Geeft deze methode ons de actor vis terug, dan stond er een vis op deze locatie. We geven in dat geval de opdracht aan onze klasse achtergrond om de vis te verwijderen.

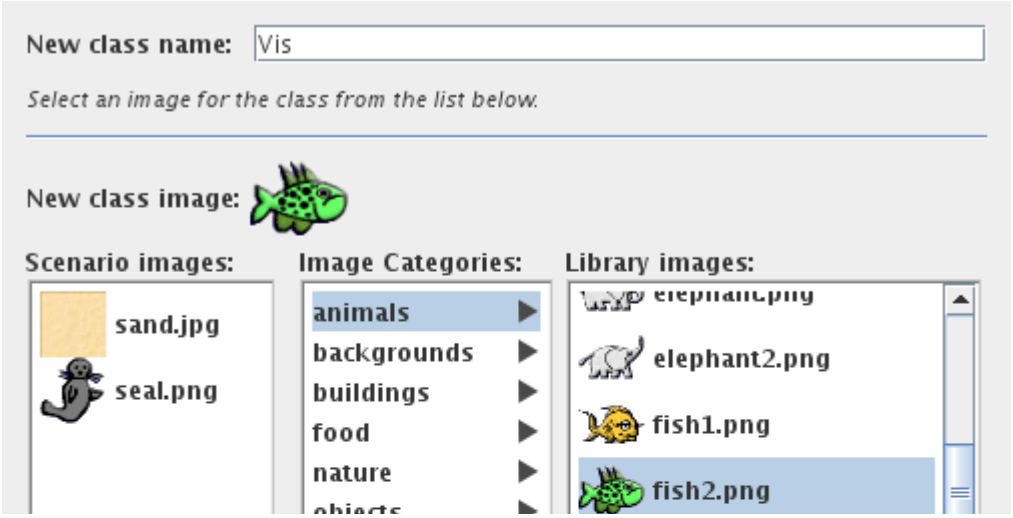
```
public void act()
{
     int x = getX();
```

```
        int y = getY();

        if (Greenfoot.isKeyDown("left")) {
            setLocation(x - 5, y);
        }
        if (Greenfoot.isKeyDown("right")) {
            setLocation(x + 5, y);
        }
        if (Greenfoot.isKeyDown("up")) {
            setLocation(x, y-5);
        }
        if (Greenfoot.isKeyDown("down")) {
            setLocation(x, y+5);
        }
        Actor object = getOneObjectAtOffset(0, 0, Vis.class);
        if(object != null) {
            getWorld().removeObject(object);
        }
}
```

Stap 5: We maken een nieuwe instantie van de klasse Actor:Actor > New subclass...



Geef de subklasse de naam “Vis” en kies een afbeelding (bvb.: fish2.png).



De vis is de tegenspeler van de zeehond. De vissen proberen zo snel mogelijk weg te lopen van de zeehond. We verplaatsen de vissen op een horizontale lijn (X-as). Indien de vis de rand heeft berijkt (= 55 px) verdwijnt de vis.

```
public void act()
{
     setLocation(getX() + 1, getY());
     if (getX() == 55) {
          getWorld().removeObject(this);
     }
}
```

Stap 6: Bij het opstarten van het spel moet er één zeehond en drie vissen aanwezig zijn op de achtergrond. Hiervoor maken we een private methode aan met de naam “populate”.

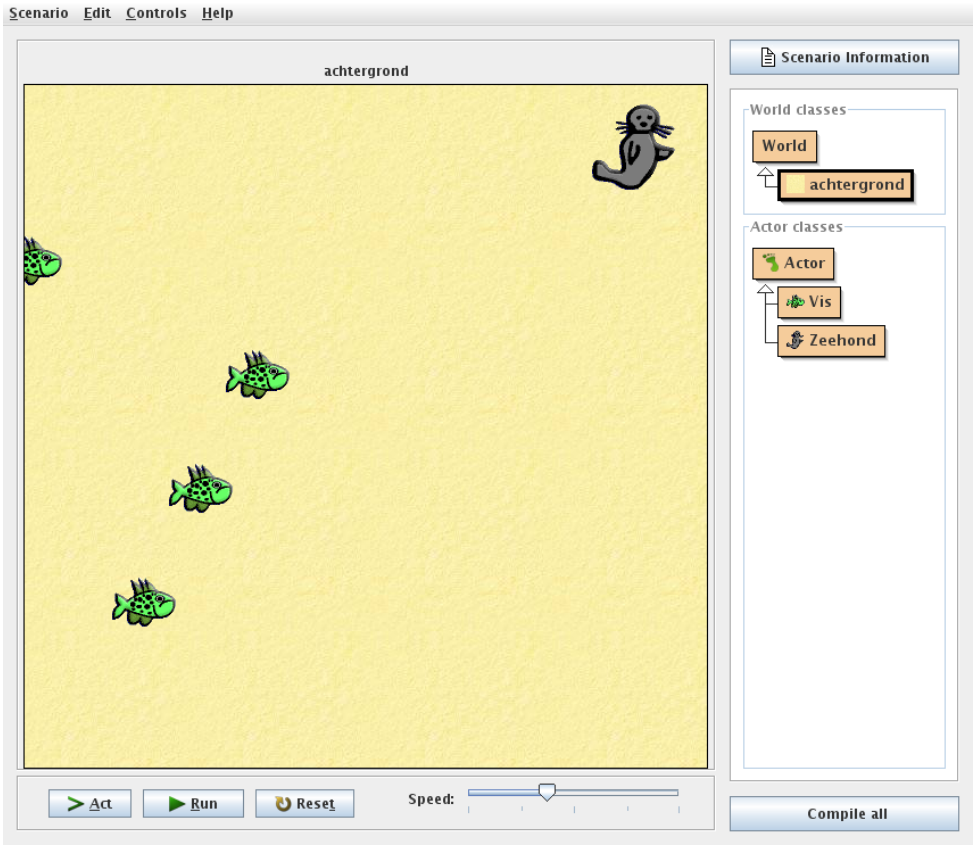
In deze methode maken we gebruik van de methode addObject(new object, x, y) uit de klasse Actor.

```
public achtergrond()
{
     super(60, 60, 10);
     populate();
}

private void populate()
{
     addObject(new Vis(), 10, 45);
     addObject(new Vis(), 15, 35);
     addObject(new Vis(), 20, 25);
     addObject(new Vis(), 0, 15);

     addObject(new Zeehond(), 53,5);
}
```

Nu moeten we de code nog compileren om de aanpassingen te zien. Klik op “Compile all”:

Vergeet je scenario niet op te slaan!

Let's play



We klikken onderaan op “Run” om het spel te starten. Gebruik de navigatietoetsen om de zeehond te bewegen en de vissen op te eten.

We kunnen het spel ook opslaan als een applet. Een applet kunnen we op een webpagina in de webbrowser openen. Iedere computer met Java Runtime erop kan je spel spelen!

1 Klik op Scenario > Export...

2 Klik op “Webpage”

3 Export

Wens je het spel als java applicatie op te slaan klik dan op “Application”. Hier maak je een jar-bestand aan. Dit bestand kan je op iedere computer met Java Runtime erop openen!