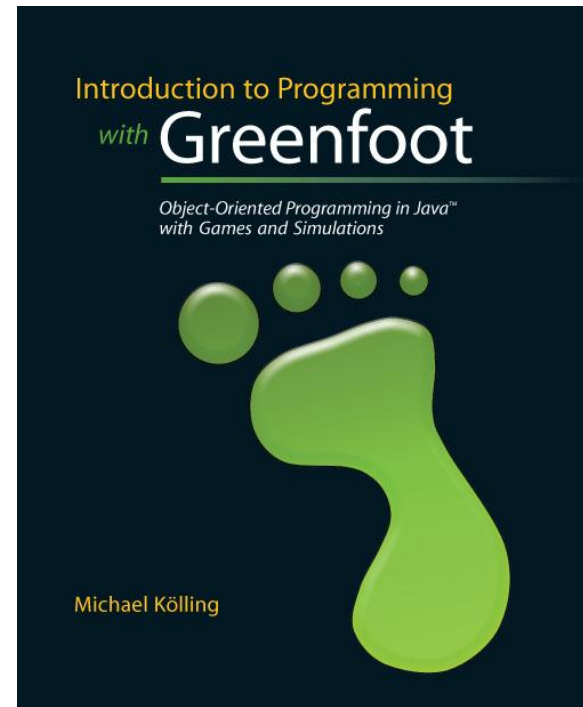


Chapter 1 - Getting to know Greenfoot

Acknowledgement: Michael Kolling & Bruce Chittenden

Greenfoot Resources

- Web site
 - <http://www.greenfoot.org>
- Scenarios
 - <http://www.greenfoot.org/scenarios/index.html>
- Tutorial
 - <http://www.greenfoot.org/doc/tutorial.html>
- Book: Introduction to Programming with Greenfoot by Michael Kölling



Desktop Icon



1.1 Getting Started

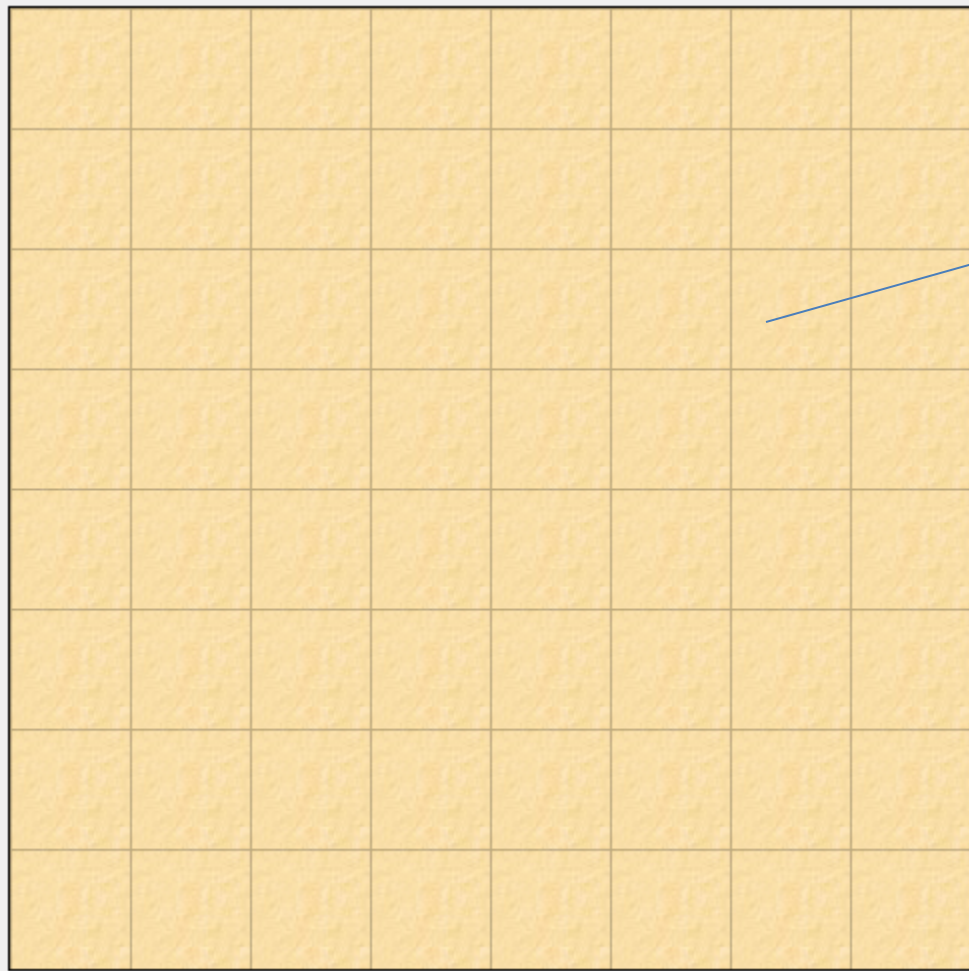
The screenshot displays the Greenfoot IDE interface for a project named "wombats". The main workspace shows a 10x10 grid labeled "wombatWorld". On the right, a class diagram shows "World" as a base class for "WombatWorld", and "Actor" as a base class for "Wombat" and "Leaf". At the bottom, there are execution controls: "Act", "Run", "Reset", a speed slider, and a "Compile all" button. Red arrows point from external text labels to these elements.

World → (points to the grid)

Execution Controls → (points to the Act, Run, Reset buttons)

Class Diagram → (points to the class diagram on the right)

World

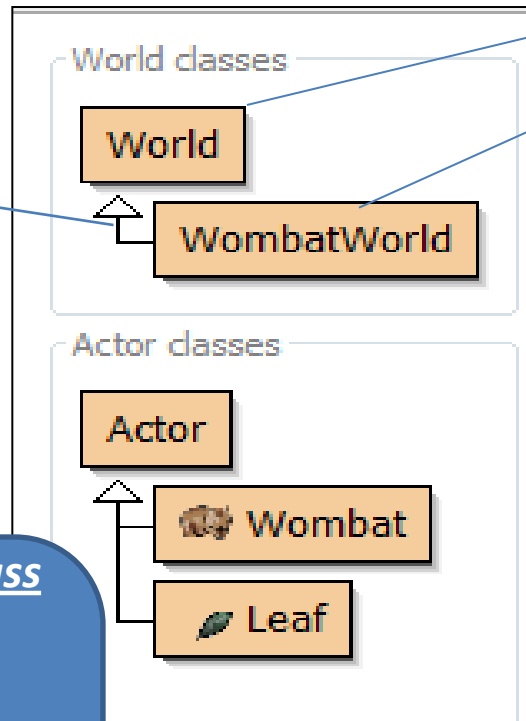


- A Finite space where our program runs.
- Space where we can introduce some actors
- Allow actors to Interact

Class Diagram

Diagram contains Boxes called Class boxes

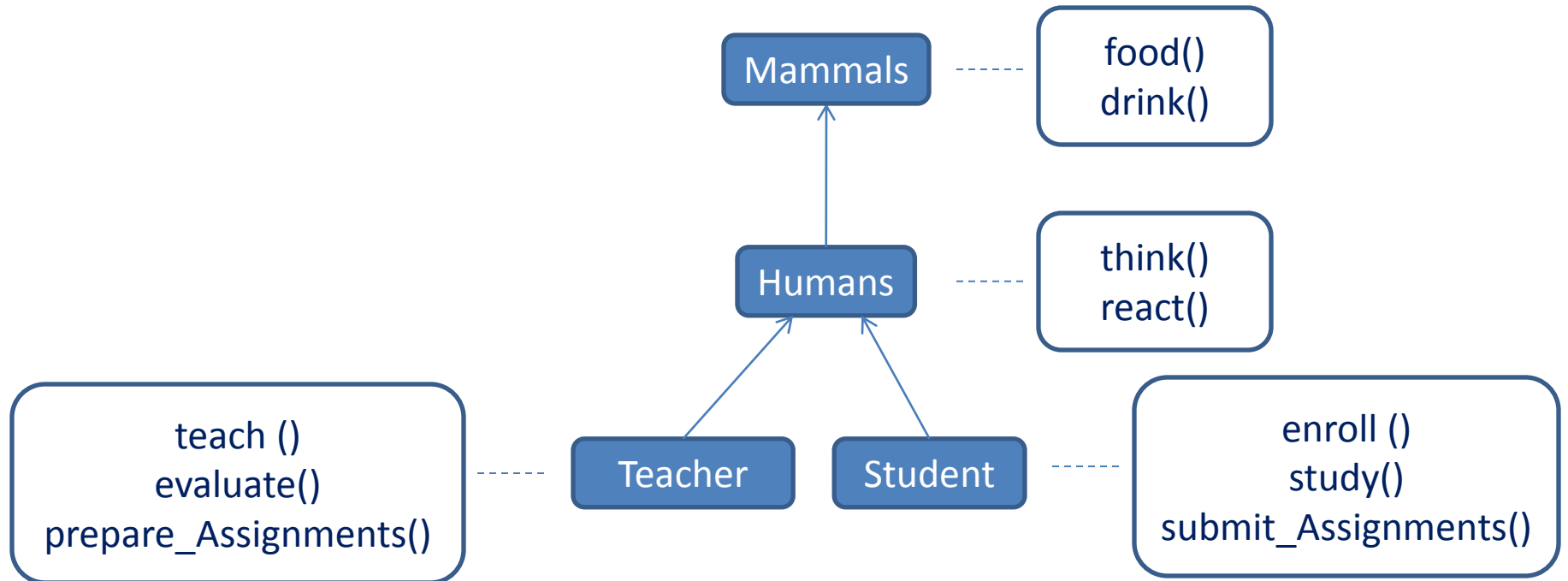
- Is-A-Relationship
- Subclass relationship



- WombatWorld is a subclass of World
- Wombat is a subclass for Actor
- Leaf is a subclass of Actor
- Leaf is not a subclass of Wombat
- Actor is the super class of Leaf and Wombat

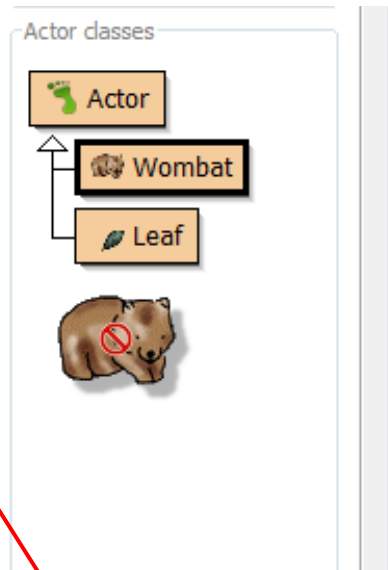
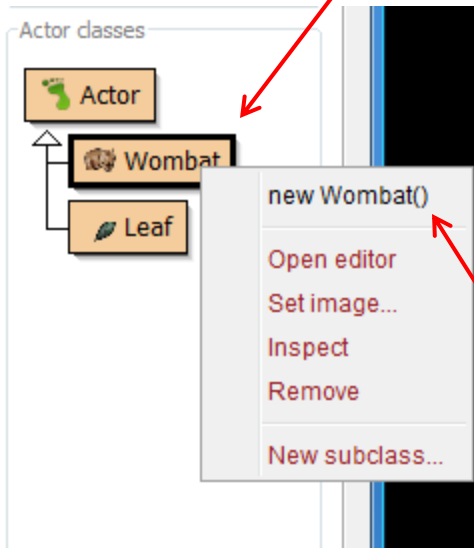
- Shows Classes involved in the scenario
- Classes define general characteristics or behavior of a group
- Characteristics that are easily differentiable

Class Hierarchy



1.2 Objects and Classes

Right Click on Wombat

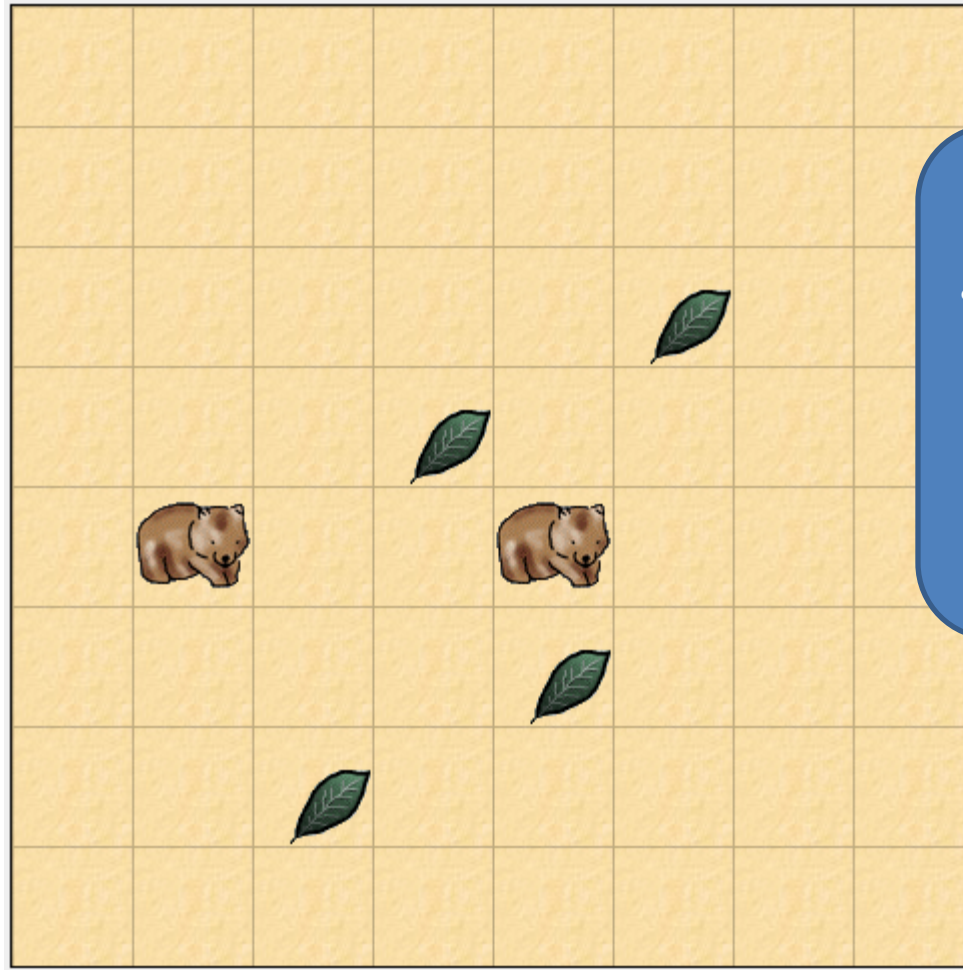


Drag to World



Click New Wombat()

Multiple Objects



- We can create multiple instances of same class

Exercise 1.1

The screenshot shows the Greenfoot IDE interface for a scenario named "leaves-and-wombats". The main workspace displays a 10x10 grid labeled "wombatWorld". Three wombats are positioned on the grid at (row, column) coordinates (1, 2), (2, 3), and (3, 4). Three leaves are positioned at (1, 5), (2, 6), and (3, 7). The interface includes a menu bar with "Scenario", "Edit", "Controls", and "Help". On the right, there is a "Scenario Information" panel and a class hierarchy diagram. The class hierarchy shows "World" as a superclass of "WombatWorld" (under "World classes") and "Actor" as a superclass of "Wombat" and "Leaf" (under "Actor classes"). At the bottom, there are control buttons for "Act", "Run", and "Reset", along with a "Speed:" slider and a "Compile all" button.

Greenfoot: leaves-and-wombats

Scenario Edit Controls Help

wombatWorld

Scenario Information

World classes

- World
- WombatWorld

Actor classes

- Actor
- Wombat
- Leaf

Act Run Reset Speed: Compile all

1.3 Interacting with Objects

inherited from Actor

- void act()
- boolean canMove()
- void eatLeaf()
- boolean foundLeaf()
- int getLeavesEaten()
- void move()
- void setDirection(int direction)
- void turnLeft()

Inspect
Remove

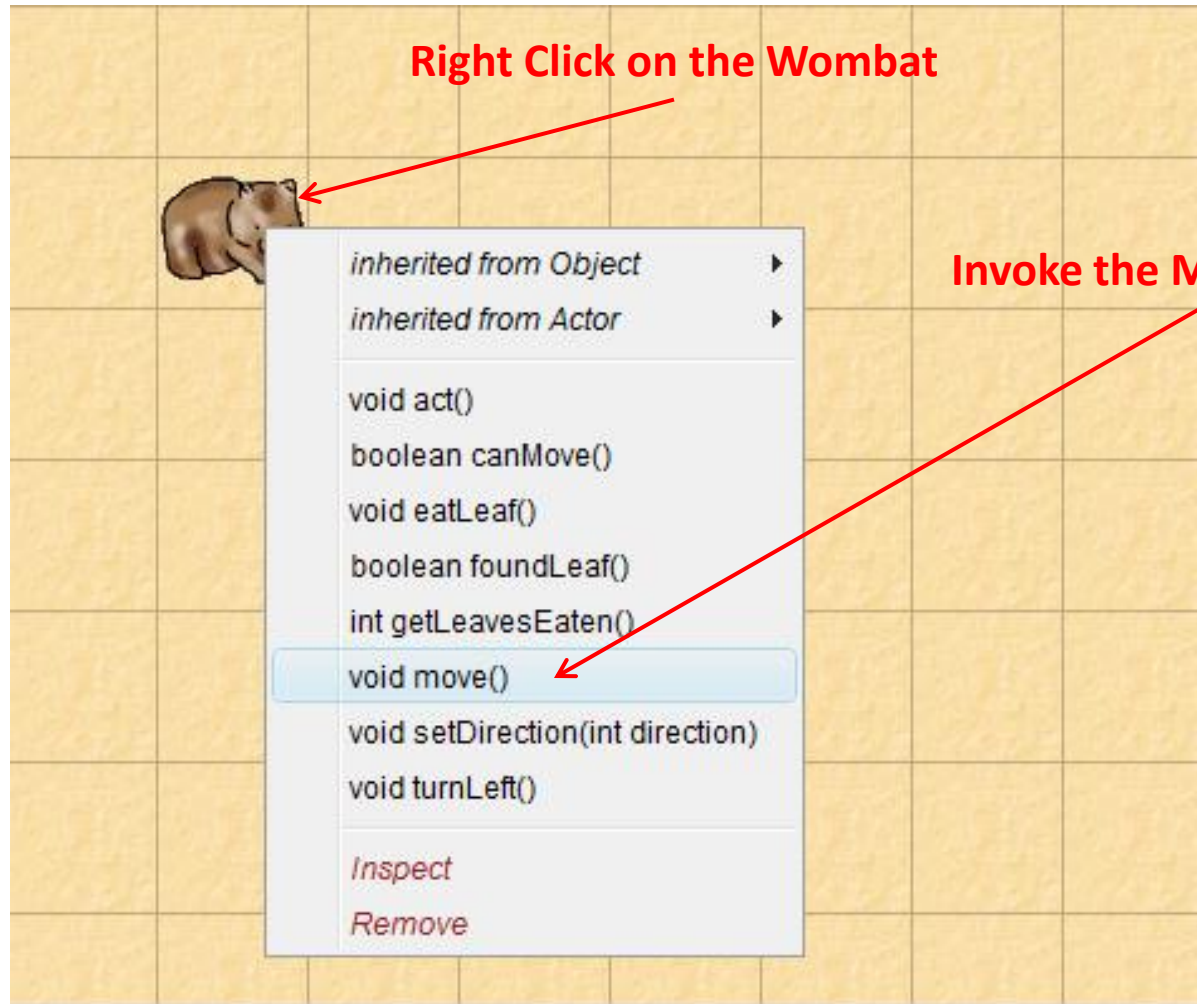
void act() [redefined in Wombat]

- GreenfootImage getImage()
- int getRotation()
- World getWorld()
- int getX()
- int getY()
- void move(int)
- void setImage(GreenfootImage)
- void setImage(String)
- void setLocation(int, int)
- void setRotation(int)
- void turn(int)

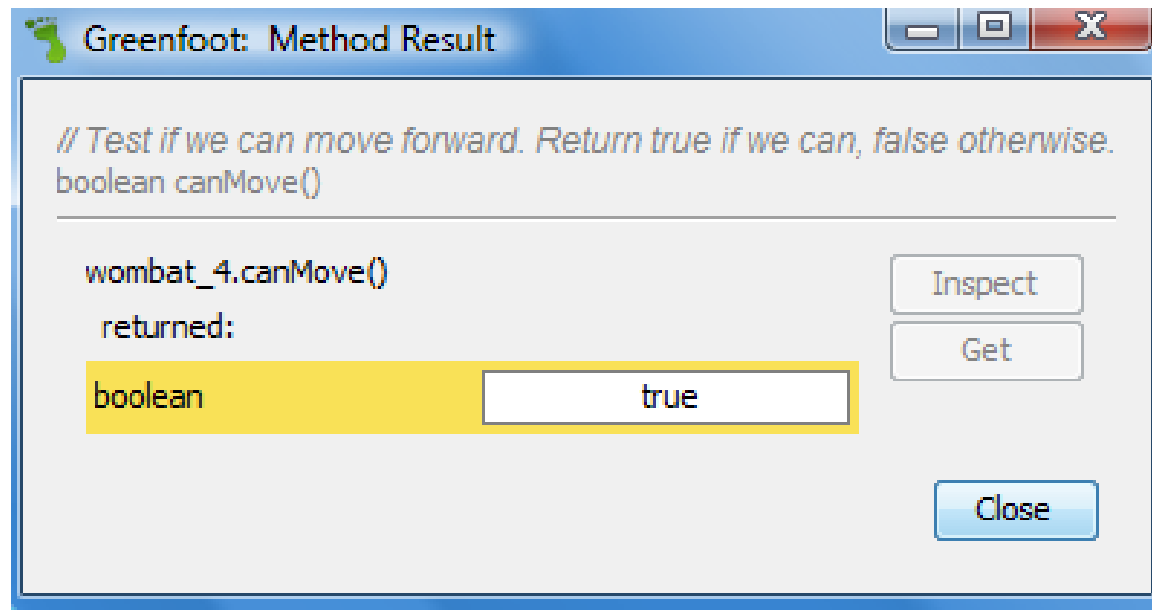
- Wombat has certain function of its own

- And Certain function inherited from Actor class

Interacting with Objects



1.4 Return Types



Exercise 1.3

The screenshot shows the Greenfoot IDE interface for a scenario named "wombatWorld". The main workspace displays a 10x10 grid with a brown Wombat actor on the second row, first column. A "Method Result" dialog box is open, showing the execution of the `wombat_2.canMove()` method, which returned the boolean value `true`. The dialog box includes "Inspect", "Get", and "Close" buttons. On the right side, the "Scenario Information" panel shows the class hierarchy: "World" is the superclass of "WombatWorld" (World classes), and "Actor" is the superclass of "Wombat" and "Leaf" (Actor classes). The bottom of the IDE features control buttons for "Act", "Run", and "Reset", a "Speed" slider, and a "Compile all" button.

Greenfoot: leaves-and-wombats

Scenario Edit Controls Help

wombatWorld

World classes

- World
- WombatWorld

Actor classes

- Actor
- Wombat
- Leaf

Greenfoot: Method Result

```
// Test if we can move forward. Return true if we can, false otherwise.  
boolean canMove()  
  
wombat_2.canMove()  
returned:  
boolean true
```

Inspect
Get
Close

> Act Run Reset Speed: [Slider] Compile all

Exercise 1.3

The screenshot shows the Greenfoot IDE interface for a project named "leaves-and-wombats". The main workspace displays a "wombatWorld" scene with a grid and a wombat icon. A "Method Result" dialog box is open, showing the execution of the `wombat_2.canMove()` method. The dialog indicates that the method returned a `boolean` value of `false`. The dialog also includes buttons for "Inspect", "Get", and "Close".

Scenario Information

World classes

- World
- WombatWorld

Actor classes

- Actor
- Wombat
- Leaf

Act Run Reset Speed: Compile all

Exercise 1.4

The screenshot shows the Greenfoot IDE interface for a project named "leaves-and-wombats". The main workspace displays a grid titled "wombatWorld" with a wombat and a leaf. A "Method Result" dialog box is open, showing the result of the `wombat_2.getLeavesEaten()` method call, which returned the integer value 0. The dialog box includes a comment: `// Tell how many leaves we have eaten. int getLeavesEaten()`. The IDE also features a class hierarchy on the right, showing "World" as a superclass for "WombatWorld", and "Actor" as a superclass for "Wombat" and "Leaf". At the bottom, there are control buttons for "Act", "Run", "Reset", a "Speed" slider, and a "Compile all" button.

Greenfoot: leaves-and-wombats

Scenario Edit Controls Help

wombatWorld

Scenario Information

World classes

- World
- WombatWorld

Actor classes

- Actor
- Wombat
- Leaf

Greenfoot: Method Result

```
// Tell how many leaves we have eaten.  
int getLeavesEaten()
```

wombat_2.getLeavesEaten() Inspect

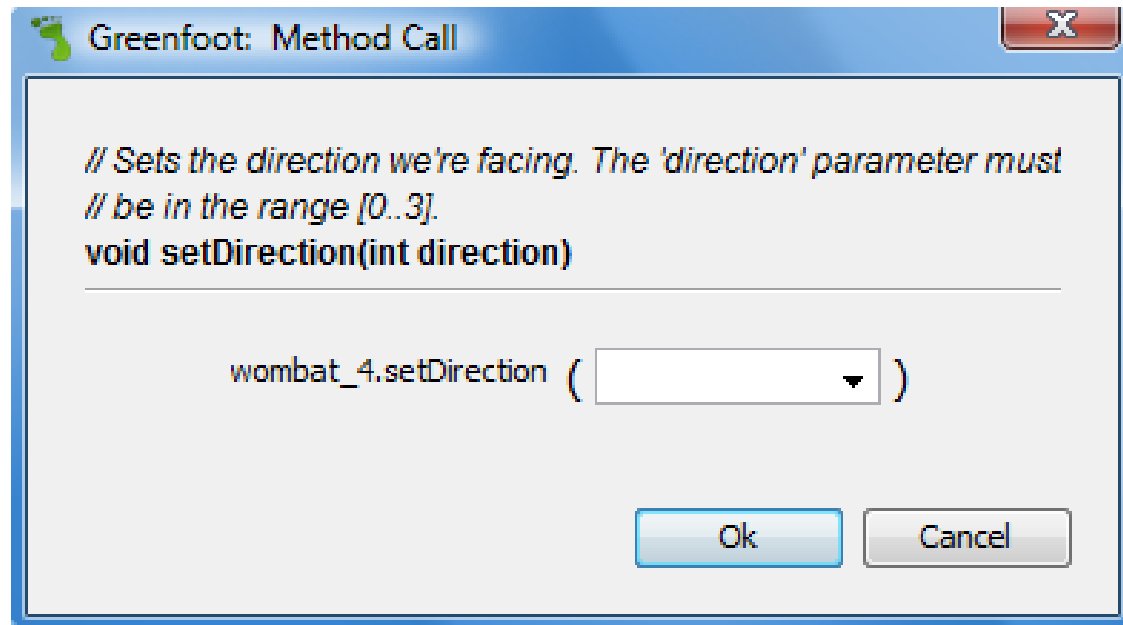
returned: Get

int 0

Close

> Act ▶ Run ↺ Reset Speed: Compile all

1.5 Parameters



Exercise 1.5

The screenshot shows the Greenfoot IDE interface for a project named "leaves-and-wombats". The main workspace displays a "wombatWorld" scene with a grid background and a wombat actor. A "Greenfoot: Method Call" dialog box is open, showing the following code:

```
// Sets the direction we're facing. The 'direction' parameter must  
// be in the range [0..3].  
void setDirection(int direction)
```

The dialog also shows the method call: `wombat_5.setDirection (3)`, where the value "3" is entered in a dropdown menu. The dialog has "Ok" and "Cancel" buttons.

On the right side of the IDE, there is a "Scenario Information" panel. It contains two sections: "World classes" and "Actor classes".

- World classes:** A hierarchy showing "World" as the base class and "WombatWorld" as a subclass.
- Actor classes:** A hierarchy showing "Actor" as the base class, with "Wombat" and "Leaf" as subclasses.

At the bottom of the IDE, there are control buttons: "Act", "Run", "Reset", a "Speed" slider, and a "Compile all" button.

Exercise 1.5

The screenshot shows the Greenfoot IDE interface for a project named "leaves-and-wombats". The main simulation window, titled "wombatWorld", displays a 10x10 grid of yellowish-brown cells. A small illustration of a brown wombat is positioned on the second cell of the first row. The interface includes a menu bar with "Scenario", "Edit", "Controls", and "Help". On the right side, there is a "Scenario Information" panel and a class hierarchy. The "World classes" section shows "World" as a base class with "WombatWorld" as a subclass. The "Actor classes" section shows "Actor" as a base class with "Wombat" and "Leaf" as subclasses. At the bottom, there are control buttons for "Act", "Run", and "Reset", a "Speed" slider, and a "Compile all" button.

Greenfoot: leaves-and-wombats

Scenario Edit Controls Help

wombatWorld

Scenario Information

World classes

- World
- WombatWorld

Actor classes

- Actor
- Wombat
- Leaf

Act Run Reset Speed: Compile all

Exercise 1.5



Exercise 1.5

Greenfoot: leaves-and-wombats

Scenario Edit Controls Help

wombatWorld

**Parameter > 3
Does Nothing**

Greenfoot: Method Call

// Sets the direction we're facing. The 'direction' parameter must
// be in the range [0..3].
void setDirection(int direction)

wombat_6.setDirection (100)

Ok Cancel

Scenario Information

World classes

- World
- WombatWorld

Actor classes

- Actor
- Wombat
- Leaf

Act Run Reset Speed: Compile all

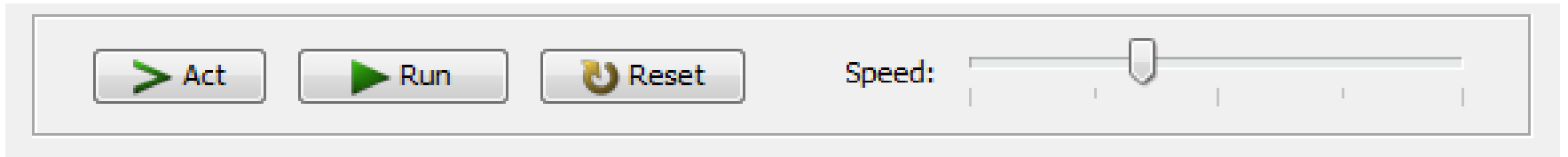
Exercise 1.5

The screenshot shows the Greenfoot IDE interface. The main window displays a scene titled "wombatWorld" with a grid background and a wombat actor. A red text overlay reads "Non-Integer Returns Error". A dialog box titled "Greenfoot: Method Call" is open, showing the following code:

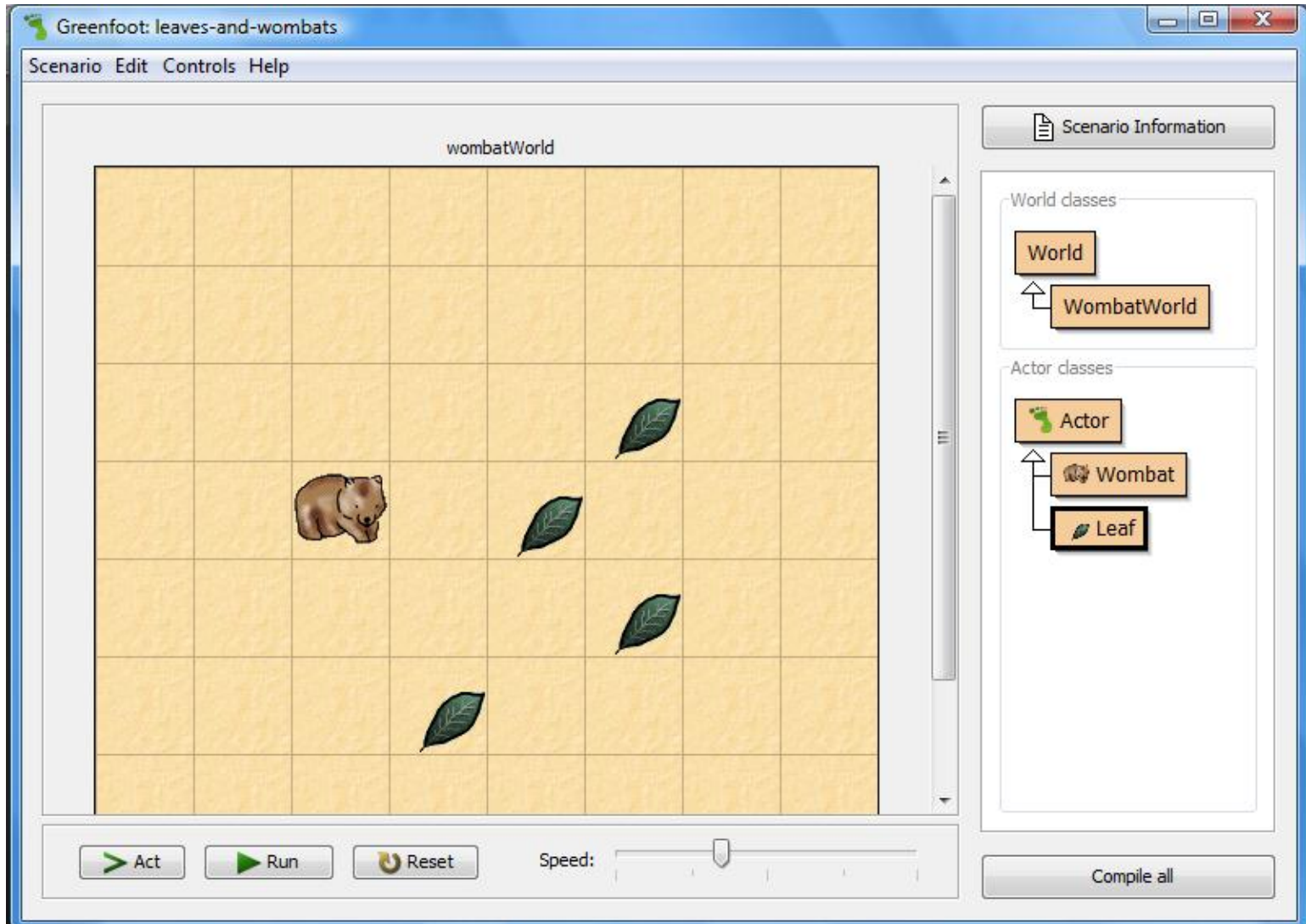
```
// Sets the direction we're facing. The 'direction' parameter must  
// be in the range [0..3].  
void setDirection(int direction)  
-----  
wombat_6.setDirection ( 2.5 )
```

Below the code, the text "possible loss of precision" is displayed. The dialog has "Ok" and "Cancel" buttons. A red arrow points from the text "Non-Integer Returns Error" to the value "2.5" in the code. The right sidebar shows the class hierarchy: "World classes" (World, WombatWorld) and "Actor classes" (Actor, Wombat, Leaf). The bottom of the IDE has buttons for "Act", "Run", "Reset", a "Speed" slider, and a "Compile all" button.

1.6 Greenfoot Execution



Exercise 1.6



Exercise 1.6

The screenshot displays the Greenfoot IDE interface for a project named "leaves-and-wombats". The main workspace shows a 10x10 grid with a yellowish background. A brown wombat is positioned in the center-left, and several green leaves are scattered to its right and bottom. A red arrow points from the text "Wombat Moves toward the Leaves" to the wombat. The right-hand panel contains a "Scenario Information" section and two class diagrams. The "World classes" diagram shows "World" as a base class with "WombatWorld" as a subclass. The "Actor classes" diagram shows "Actor" as a base class with "Wombat" and "Leaf" as subclasses. At the bottom, there are control buttons for "Act", "Run", and "Reset", along with a "Speed" slider and a "Compile all" button.

Greenfoot: leaves-and-wombats

Scenario Edit Controls Help

wombatWorld

Wombat Moves toward the Leaves

Scenario Information

World classes

- World
- WombatWorld

Actor classes

- Actor
- Wombat
- Leaf

Act Run Reset Speed: [Slider] Compile all

Exercise 1.6

The screenshot shows the Greenfoot IDE interface for a scenario named "leaves-and-wombats". The main workspace, titled "wombatWorld", displays a 10x10 grid. A brown wombat actor is positioned at the intersection of the 4th column and 6th row. Three green leaf actors are located at (6, 4), (5, 5), and (7, 6). A red arrow points from the text "Wombat Moves to Leaves" to the wombat actor. The bottom control bar includes "Act", "Run", and "Reset" buttons, a "Speed" slider, and a "Compile all" button.

Scenario Edit Controls Help

wombatWorld

Wombat Moves to Leaves

Scenario Information

World classes

- World
- WombatWorld

Actor classes

- Actor
- Wombat
- Leaf

Act Run Reset Speed: Compile all

Example 1.7

The screenshot displays the Greenfoot IDE interface for a project titled "leaves-and-wombats". The main workspace shows a 10x10 grid representing a world named "wombatWorld". A wombat actor is positioned at the center of the grid, and three leaf actors are scattered around it. A red arrow points from the text "Wombat Eats Leaf" to the wombat actor. The right-hand panel contains a "Scenario Information" section with two class diagrams. The "World classes" diagram shows "World" as a base class and "WombatWorld" as a subclass. The "Actor classes" diagram shows "Actor" as a base class, with "Wombat" and "Leaf" as subclasses. The bottom control bar includes buttons for "Act", "Run", and "Reset", a "Speed" slider, and a "Compile all" button.

Greenfoot: leaves-and-wombats

Scenario Edit Controls Help

wombatWorld

Wombat Eats Leaf

Scenario Information

World classes

- World
- WombatWorld

Actor classes

- Actor
- Wombat
- Leaf

Act Run Reset Speed: Compile all

Exercise 1.8

The screenshot shows the Greenfoot IDE window titled "Greenfoot: leaves-and-wombats". The main workspace is a grid labeled "wombatWorld" containing several wombats and leaves. The right sidebar shows the class hierarchy for "World classes" (World, WombatWorld) and "Actor classes" (Actor, Wombat, Leaf). The bottom control bar includes buttons for "Act", "Run", and "Reset", along with a "Speed" slider and a "Compile all" button.

Greenfoot: leaves-and-wombats

Scenario Edit Controls Help

wombatWorld

World classes

- World
- WombatWorld

Actor classes

- Actor
- Wombat
- Leaf

> Act ▶ Run ↻ Reset Speed: [Slider] Compile all

Exercise 1.8

Greenfoot: leaves-and-wombats

Scenario Edit Controls Help

wombatWorld

The >Act Execution Control Affects All the Wombats

World classes

- World
- WombatWorld

Actor classes

- Actor
- Wombat
- Leaf

> Act Run Reset Speed: [Slider] Compile all

Exercise 1.9

The screenshot shows the Greenfoot IDE interface for a scenario named "leaves-and-wombats". The main workspace, titled "wombatWorld", displays a 10x10 grid. A wombat is positioned at the left edge of the grid. Three leaves are scattered in the lower-right quadrant. The text "Wombat Runs Around the Edge of the World" is overlaid in red. The right sidebar shows the class hierarchy: "World" is the superclass of "WombatWorld", and "Actor" is the superclass of "Wombat" and "Leaf". The bottom control bar includes buttons for "Act", "Run", and "Reset", a "Speed" slider, and a "Compile all" button.

Greenfoot: leaves-and-wombats

Scenario Edit Controls Help

wombatWorld

Wombat Runs Around the Edge of the World

Scenario Information

World classes

- World
- WombatWorld

Actor classes

- Actor
- Wombat
- Leaf

Act Run Reset Speed: [Slider] Compile all

Exercise 1.9

- Act Method
 - If we're sitting on a leaf, eat the leaf
 - Otherwise, if we can move forward, move forward
 - Otherwise, turn left

1.7 A Second Example

The screenshot displays the Greenfoot IDE interface for a scenario named "asteroids1". The main workspace shows a "space" world with a black background and white stars. A white rocket is positioned in the center, and two grey, irregularly shaped asteroids are visible on the left and right sides. The interface includes a menu bar with "Scenario", "Edit", "Controls", and "Help". On the right side, there is a "Scenario Information" panel and a class hierarchy diagram. The class hierarchy shows "World" as the base class for "Space". "Actor" is the base class for "Explosion", "Mover", "Bullet", "Rocket", and "Asteroid". "Mover" is the base class for "Bullet", "Rocket", and "Asteroid". "Vector" is listed under "Other classes". At the bottom, there are control buttons for "Act", "Run", and "Reset", along with a "Speed" slider and a "Compile all" button.

Greenfoot: asteroids1

Scenario Edit Controls Help

space

Scenario Information

World classes

- World
- Space

Actor classes

- Actor
- Explosion
- Mover
- Bullet
- Rocket
- Asteroid

Other classes

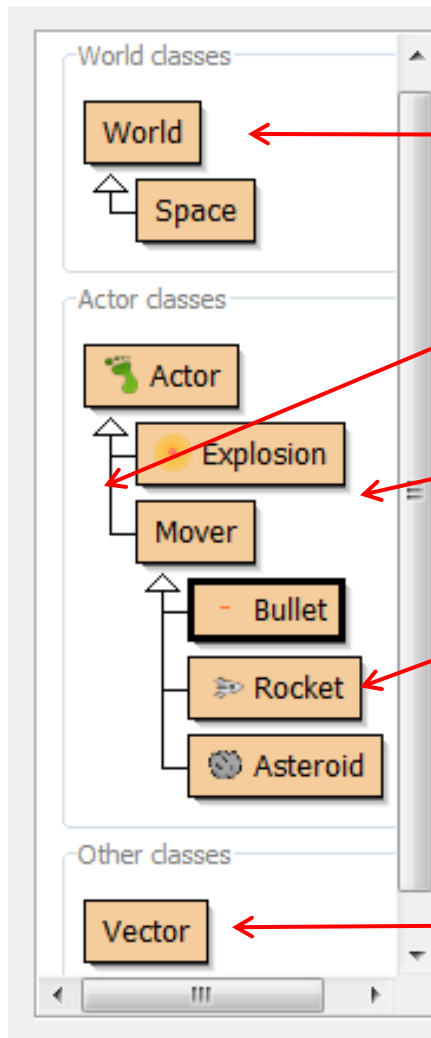
- Vector

Act Run Reset

Speed: [Slider]

Compile all

1.8 Understanding the Class Diagram



World Class is always there in Greenfoot scenarios, it is built-in. Space represents a specific world for this scenario

Arrows show relationships

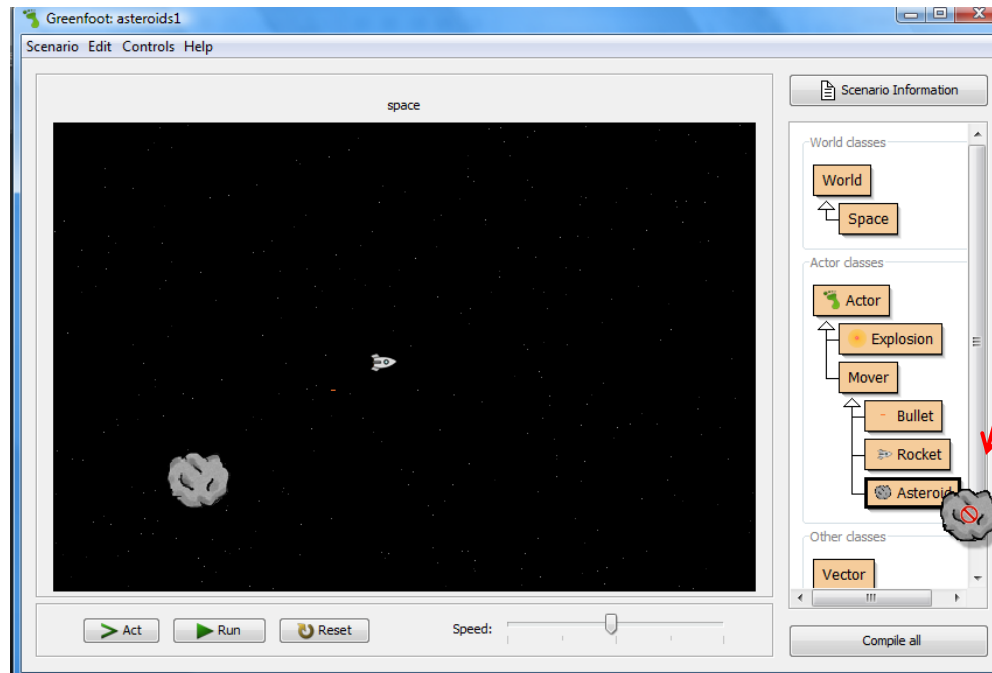
Explosion and Mover are subclasses of Actor

Bullet, Rocket, and Asteroid are subclasses of Mover.

Vector is a helper class

1.9 Playing with Asteroids

Start Playing by Creating Some Actor Objects (Objects of the Subclass of Actor). Create Objects for Rocket, Bullet, Asteroid, and Explosion



Exercise 1.10

The screenshot shows the Greenfoot IDE interface for a scenario named "asteroids1". The main workspace displays a "space" scene with a black background and several grey, rock-like objects. A red arrow points to a small white object, and a context menu is open over it. The menu lists inheritance information and methods: "inherited from Object", "inherited from Actor", "inherited from Mover", "void act()", "int getShotsFired()", "int getSpeed()", and "void setGunReloadTime(int reloadTime)". At the bottom of the menu are "Inspect" and "Remove" options. The right sidebar shows a class hierarchy: "World classes" (World, Space) and "Actor classes" (Actor, Explosion, Mover, Bullet, Rocket, Asteroid). The "Asteroid" class is highlighted. The "Other classes" section shows "Vector". The bottom of the IDE has "Act", "Run", and "Reset" buttons, a "Speed:" slider, and a "Compile all" button.

Greenfoot: asteroids1

Scenario Edit Controls Help

space

Right Click on the Object

- inherited from Object
- inherited from Actor
- inherited from Mover
- void act()
- int getShotsFired()
- int getSpeed()
- void setGunReloadTime(int reloadTime)
- Inspect
- Remove

Scenario Information

World classes

- World
- Space

Actor classes

- Actor
- Explosion
- Mover
- Bullet
- Rocket
- Asteroid

Other classes

- Vector

Act Run Reset

Speed:

Compile all

Exercise 1.11

Greenfoot: asteroids1

Scenario Edit Controls Help

space

Right Click on the Object and Select Inspect

- inherited from Object*
- inherited from Actor*
- inherited from Mover*
- void act()
- int getShotsFired()
- int getSpeed()
- void setGunReloadTime(int reloadTime)
- Inspect**
- Remove

Scenario Information

World classes

- World
- Space

Actor classes

- Actor
- Explosion
- Mover
 - Bullet
 - Rocket
 - Asteroid

Other classes

- Vector

Act Run Reset Speed: [Slider]

Compile all

Exercise 1.11

The image shows a screenshot of the Greenfoot software interface. The main window is titled "Greenfoot: asteroids1" and contains a "space" scene with several rocket objects. A red arrow points from the text "Right Click on the Object and Select Inspect" to one of the rocket objects. An "Object Inspector" window is open, displaying the properties of a selected "rocket : Rocket" object. The properties are listed in a table-like format with input fields for values.

Right Click on the Object and Select Inspect

Property	Value
private int gunReloadTime	20
private int reloadDelayCount	44
private Vector acceleration	[Vector icon]
private int shotsFired	12
private GreenfootImage rocket	[Image icon]
private GreenfootImage rocketWith...	[Image icon]
private Vector movement	[Vector icon]
private double x	213.0
private double y	91.0
int x (hidden)	213
int y (hidden)	91
private int rotation	-145
World world	[World icon]
private GreenfootImage image	[Image icon]

Buttons: Act, Run, Reset, Scenario Information, World classes, Inspect, Get, Show static fields, Close

Exercise 1.12

Greenfoot: asteroids1

Scenario Edit Controls Help

space

Right Click on the Object and Select getSpeed ()

inherited from Object ▶
inherited from Actor ▶
inherited from Mover ▶

void act()
int getShotsFired()
int getSpeed() ←
void setGunReloadTime(int reloadTime)

Inspect
Remove

Scenario Information

World classes

- World
- Space

Actor classes

- Actor
- Explosion
- Mover
- Bullet
- Rocket
- Asteroid

Other classes

- Vector

Act Run Reset

Speed: [Slider]

Compile all

Exercise 1.12

The screenshot shows the Greenfoot IDE interface for a project named "asteroids1". The main workspace displays a "space" scene with a rocket icon. A "Method Result" dialog box is open, showing the following information:

```
// Return the approximate current travelling speed of this rocket.  
int getSpeed()  
  
rocket.getSpeed()  
returned:  
int 3
```

The dialog includes "Inspect", "Get", and "Close" buttons. The right-hand sidebar contains a class hierarchy:

- World classes: World, Space (inherits from World)
- Actor classes: Actor, Explosion (inherits from Actor), Mover (inherits from Actor), Bullet (inherits from Mover), Rocket (inherits from Mover), Asteroid (inherits from Mover)
- Other classes: Vector

At the bottom of the IDE, there are control buttons: "Act", "Run", "Reset", a "Speed" slider, and a "Compile all" button.

Exercise 1.13

The screenshot displays the Greenfoot IDE interface for a project named "asteroids1". The main window shows a game scene with a rocket and several asteroids. Three "Greenfoot: Object Inspector" windows are overlaid, each showing the state of an "Asteroid" object. The top window, titled "asteroid : Asteroid", shows "private int size" at 64 and "private int stability" at 64. The middle window, titled "asteroid 2 : Asteroid", shows "private int size" at 32 and "private int stability" at 32. The bottom window, titled "asteroid 3 : Asteroid", shows "private int size" at 16 and "private int stability" at 16. The right sidebar contains a class hierarchy diagram with "World" and "Space" under "World classes", and "Actor", "Explosion", "Mover", "Bullet", "Rocket", and "Asteroid" under "Actor classes". The "Asteroid" class is highlighted. At the bottom, there are control buttons for "Act", "Run", "Reset", and "Compile all".

Greenfoot: asteroids1

Scenario Edit Controls Help

Greenfoot: Object Inspector

asteroid : Asteroid

private int size 64

private int stability 64

Inspect

Get

Greenfoot: Object Inspector

asteroid 2 : Asteroid

private int size 32

private int stability 32

Inspect

Get

Greenfoot: Object Inspector

asteroid 3 : Asteroid

private int size 16

private int stability 16

Inspect

Get

Show static fields

Close

Scenario Information

World classes

- World
- Space

Actor classes

- Actor
- Explosion
- Mover
- Bullet
- Rocket
- Asteroid

Other classes

- Vector

Act Run Reset Compile all

Exercise 1.14

Greenfoot: asteroids1

Scenario Edit Controls Help

space

Right Click on the Object and Select setSize(int Size) and Set the Size to 256

inherited from Object ▶
inherited from Actor ▶
inherited from Mover ▶

void act()
int getStability()
void hit(int damage)
void setSize(int size)

Inspect
Remove

Greenfoot: Method Call

// Set the size of this asteroid. Note that stability is directly // related to size. Smaller asteroids are less stable.
void setSize(int size)

asteroid.setSize (256)

Ok Cancel

Scenario Information

World classes

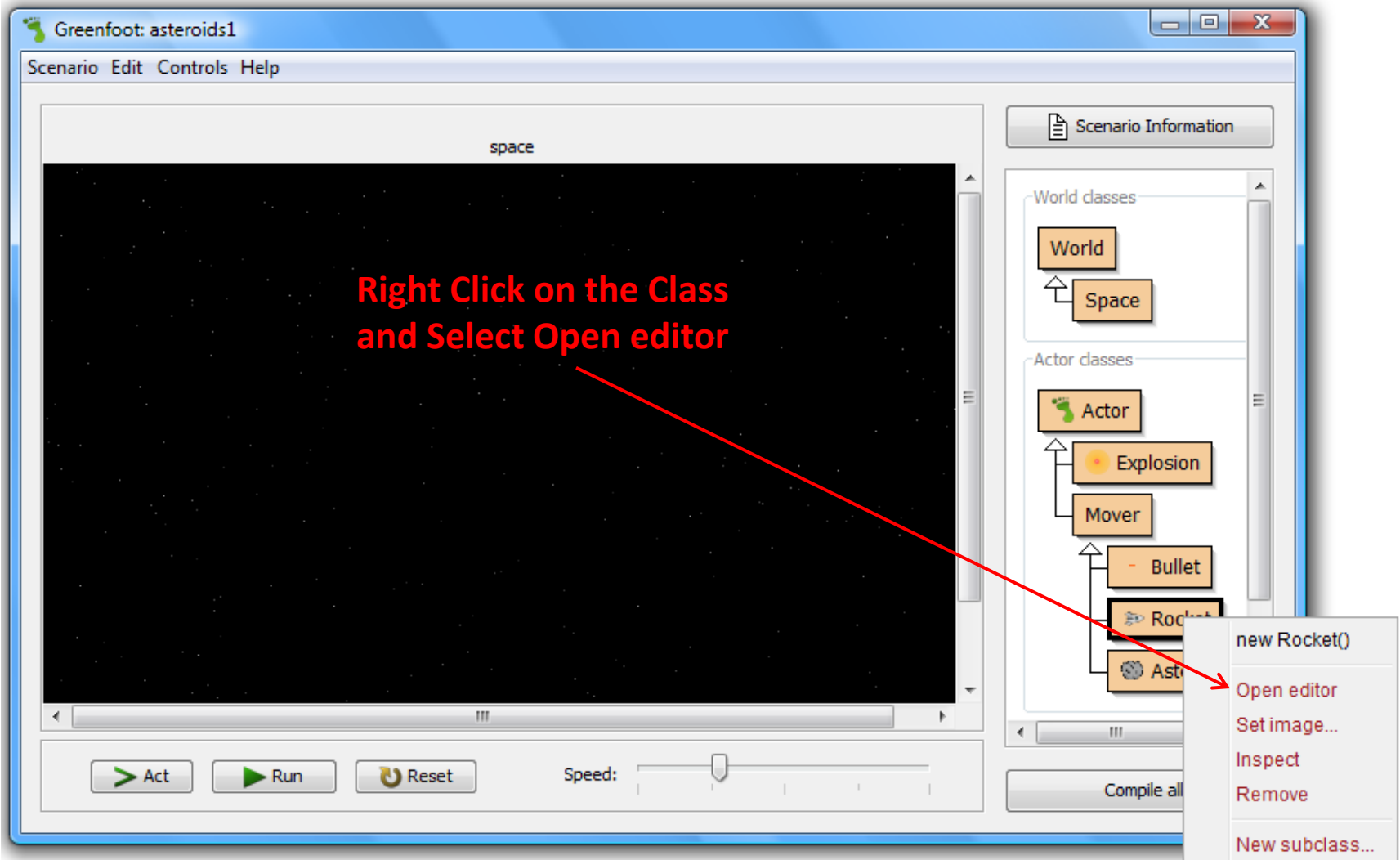
- World
- Space

Actor classes

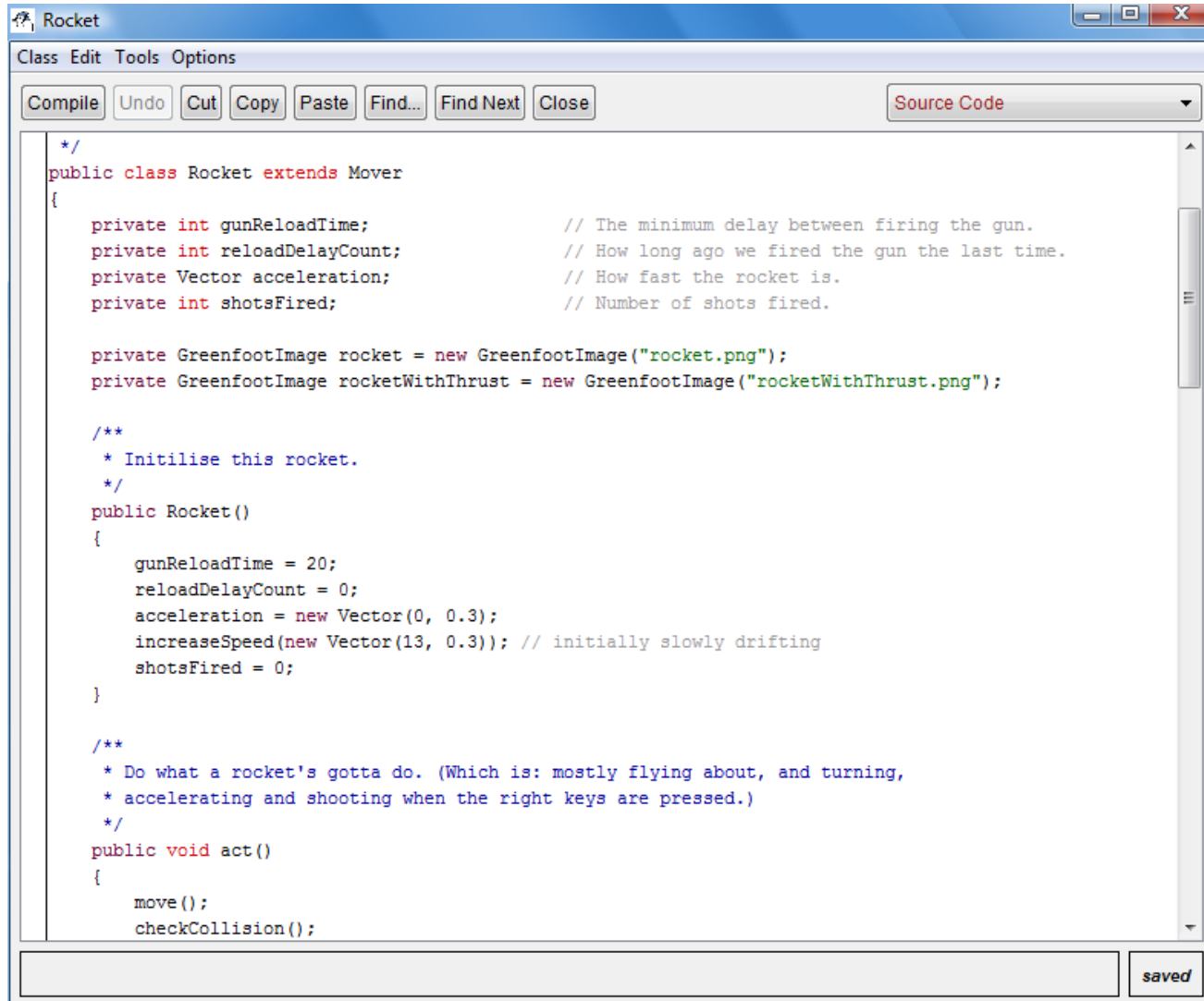
- Actor
- Explosion
- Mover
- Bullet
- Rocket
- Asteroid

Act Run Reset Speed: Compile all

1.10 Source Code



Source Code for Rocket



```
*/
public class Rocket extends Mover
{
    private int gunReloadTime;           // The minimum delay between firing the gun.
    private int reloadDelayCount;       // How long ago we fired the gun the last time.
    private Vector acceleration;        // How fast the rocket is.
    private int shotsFired;            // Number of shots fired.

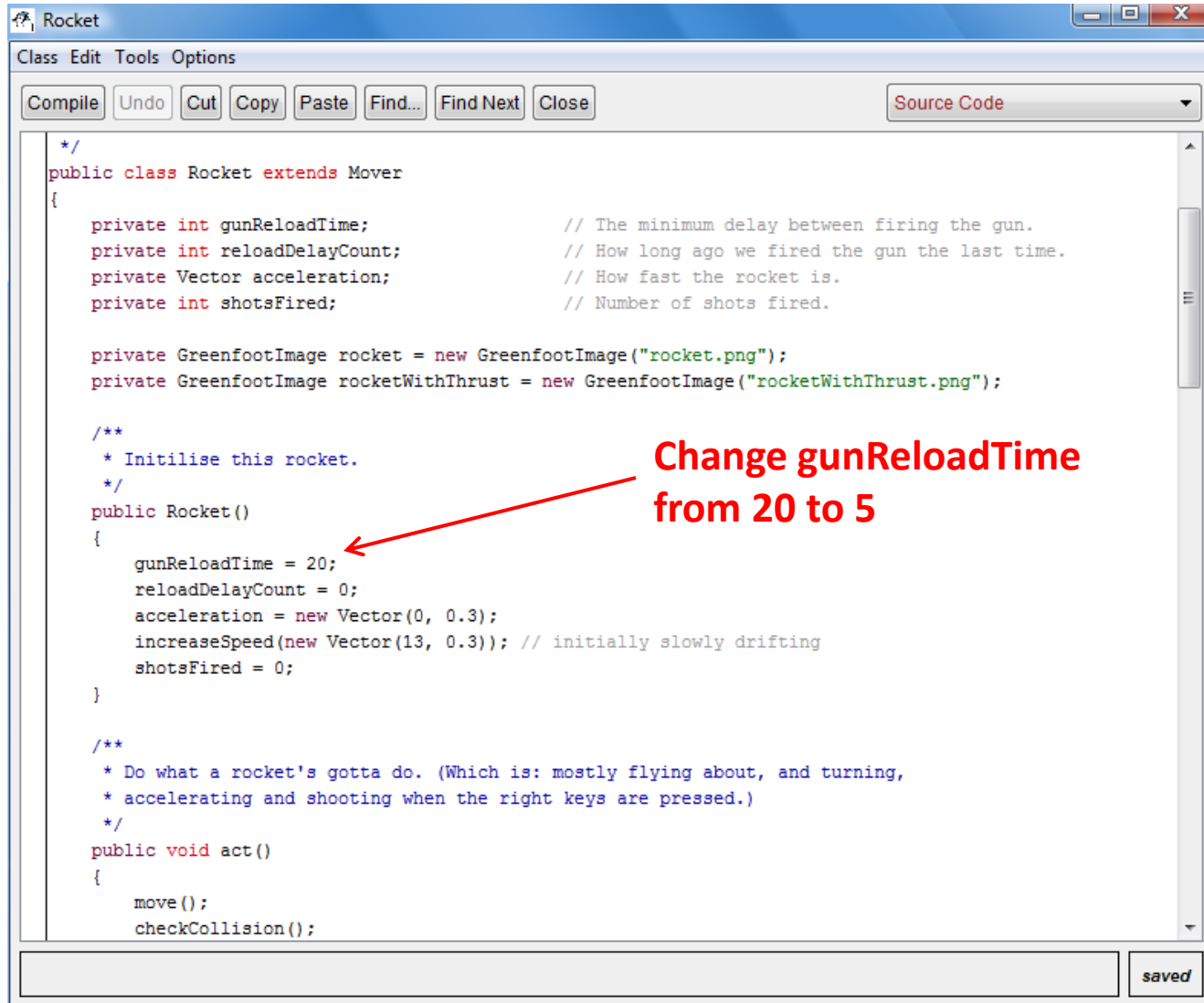
    private GreenfootImage rocket = new GreenfootImage("rocket.png");
    private GreenfootImage rocketWithThrust = new GreenfootImage("rocketWithThrust.png");

    /**
     * Initilise this rocket.
     */
    public Rocket()
    {
        gunReloadTime = 20;
        reloadDelayCount = 0;
        acceleration = new Vector(0, 0.3);
        increaseSpeed(new Vector(13, 0.3)); // initially slowly drifting
        shotsFired = 0;
    }

    /**
     * Do what a rocket's gotta do. (Which is: mostly flying about, and turning,
     * accelerating and shooting when the right keys are pressed.)
     */
    public void act()
    {
        move();
        checkCollision();
    }
}
```

saved

Exercise 1.15



```
public class Rocket extends Mover
{
    private int gunReloadTime;           // The minimum delay between firing the gun.
    private int reloadDelayCount;       // How long ago we fired the gun the last time.
    private Vector acceleration;        // How fast the rocket is.
    private int shotsFired;            // Number of shots fired.

    private GreenfootImage rocket = new GreenfootImage("rocket.png");
    private GreenfootImage rocketWithThrust = new GreenfootImage("rocketWithThrust.png");

    /**
     * Initilise this rocket.
     */
    public Rocket()
    {
        gunReloadTime = 20;
        reloadDelayCount = 0;
        acceleration = new Vector(0, 0.3);
        increaseSpeed(new Vector(13, 0.3)); // initially slowly drifting
        shotsFired = 0;
    }

    /**
     * Do what a rocket's gotta do. (Which is: mostly flying about, and turning,
     * accelerating and shooting when the right keys are pressed.)
     */
    public void act()
    {
        move();
        checkCollision();
    }
}
```

Change gunReloadTime from 20 to 5

saved

Exercise 1.15

```
public class Rocket extends Mover
{
    private int gunReloadTime;           // The minimum delay between firing the gun.
    private int reloadDelayCount;       // How long ago we fired the gun the last time.
    private Vector acceleration;        // How fast the rocket is.
    private int shotsFired;            // Number of shots fired.

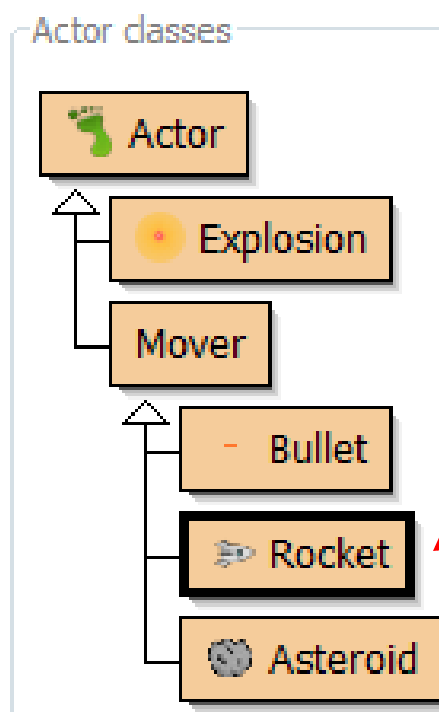
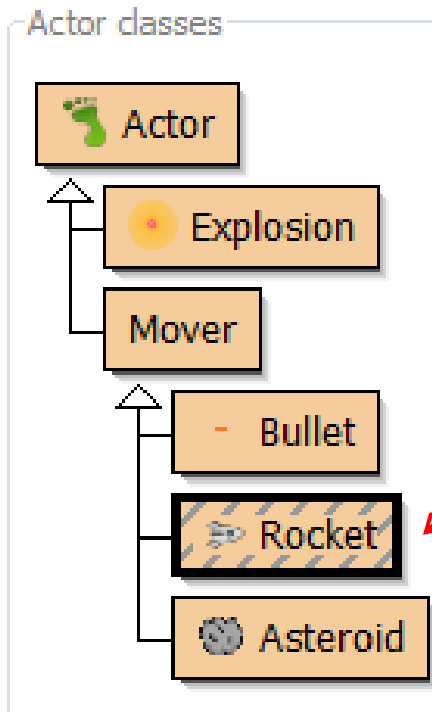
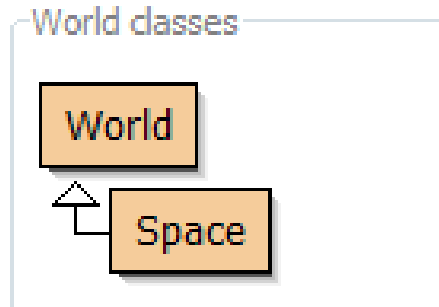
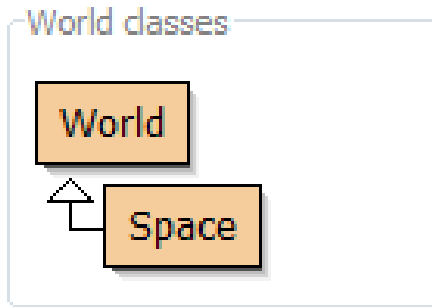
    private GreenfootImage rocket = new GreenfootImage("rocket.png");
    private GreenfootImage rocketWithThrust = new GreenfootImage("rocketWithThrust.png");

    /**
     * Initilise this rocket.
     */
    public Rocket()
    {
        gunReloadTime = 5;
        reloadDelayCount = 0;
        acceleration = new Vector(0, 0.3);
        increaseSpeed(new Vector(13, 0.3)); // initially slowly drifting
        shotsFired = 0;
    }

    /**
     * Do what a rocket's gotta do. (Which is: mostly flying about, and turning,
     * accelerating and shooting when the right keys are pressed.)
     */
    public void act()
    {
        move();
        checkCollision();
    }
}
```

changed

Exercise 1.15



1.11 Summary

In this chapter, we have seen what Greenfoot scenarios can look like and how to interact with them. We have seen how to create objects and how to communicate with these objects by invoking their methods. Some methods are commands to objects, while other methods return information about the object. Parameters are used to provide additional information to methods, while return values pass information back to the caller.

Concept Summary

Concept summary

- Greenfoot scenarios consist of a set of **classes**.
- Many **objects** can be created from a **class**.
- Objects have **methods**. Invoking these performs an action.
- The **return type** of a method specifies what a method call will return.
- A method with a **void** return type does not return a value.
- Methods with void return types represent **commands**; methods with non-void return types represent **questions**.
- A **parameter** is a mechanism to pass in additional data to a method.
- Parameters and return values have **types**. Examples of types are **int** for numbers, and **boolean** for true/false values.
- The specification of a method, which shows its return type, name, and parameters, is called its **signature**.
- Objects that can be placed into the world are known as **actors**.
- A **subclass** is a class that represents a specialization of another. In Greenfoot, this is shown with an arrow in the class diagram.
- Every class is defined by **source code**. This code defines what objects of this class can do. We can look at the source code by opening the class's editor.
- Computers do not understand source code. It needs to be translated to machine code before it can be executed. This is called **compilation**.