MICHAEL KÖLLING

Introduction to Programming with Greenfoot

Object-Oriented Programming in Java™
with Games and Simulations

2/E

Introduction to Programming *with* Greenfoot

Object-Oriented Programming in Java
With Games and Simulations

This page intentionally left blank

Introduction to Programming *with* Greenfoot

Object-Oriented Programming in Java
With Games and Simulations

Michael Kölling

PEARSON

Boston Columbus Indianapolis New York San Francisco Hoboken
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montréal Toronto
Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Editorial Director: *Marcia Horton*
Executive Editor: *Tracy Johnson*
Editorial Assistant: *Kelsey Loanes*
VP of Marketing: *Christy Lesko*
Director of Field Marketing: *Tim Galligan*
Product Marketing Manager: *Bram van Kempen*
Field Marketing Manager: *Demetrius Hall*
Marketing Assistant: *Jon Bryant*
Director of Product Management: *Erin Gregg*
Team Lead Product Management: *Scott Disanno*
Program Manager: *Carole Snyder*
Production Project Manager: *Camille Trentacoste*
Procurement Manager: *Mary Fischer*
Senior Specialist, Program Planning and Support: *Maura Zaldivar-Garcia*
Manager, Rights Management: *Rachel Youdelman*
Senior Project Manager, Rights Management: *Timothy Nicholls*
Cover Designer: *Black Horse Designs*
Cover Art: *Ivan kmit/Fotolia*

Library of Congress Cataloging-in-Publication Data on File

Kölling, Michael.
Introduction to programming with greenfoot object-oriented programming in java with games and simulations / Michael Kölling. — 2nd edition.
pages cm
Includes bibliographical references and index.
ISBN 978-0-13-405429-2 — ISBN 0-13-405429-6 1. Greenfoot (Electronic resource) 2. Object-oriented programming (Computer science)—Study and teaching. 3. Java (Computer program language) 4. Computer games—Programming. I. Title.
QA76.64.K657 2016
794.8'1526—dc23

2015000976

10 9 8 7 6 5 4 3 2 1

PEARSON

ISBN-13: 978-0-13-405429-2
ISBN-10: 0-13-405429-6

To my darling girl.

This page intentionally left blank

Contents

List of scenarios discussed in this book

Leaves and Wombats (Chapter 1)
This is a simple example showing wombats moving around on screen, occasionally eating leaves. The scenario has no specific purpose other than illustrating some important object-oriented concepts and Greenfoot interactions.

Asteroids 1 (Chapter 1)
This is a simple version of a classic arcade game. You fly a spaceship through space and try to avoid being hit by asteroids. At this stage, we only use the scenario to make some small changes and illustrate how to edit source code to change program behavior.

Little Crab (Chapters 2, 3, 4)
This is our first full development. Starting from almost nothing, we develop a simple game slowly, adding things such as movement, keyboard control, sound, and many other elements of typical games.

Fat Cat (Chapter 2)
This is a small scenario serving as a basis for exercises with methods calls and simple statements. Make the cat perform while you practice your Java.

Stickman (Chapter 3)
Another small exercise scenario. This does not do much to start with, and we use it to do some exercises with if-statements at the end of the chapter.

White Blood Cell (WBC) (Chapter 5)
A typical side-scrolling game. We develop it from a very primitive, rudimentary start to a full, playable game. You steer a white blood cell through an artery to catch and neutralize bacteria.

Piano (Chapter 6)
An on-screen piano that you can really play.

Bubbles (Chapter 6)
A small scenario serving as a platform to practice writing some loops.

Autumn (Chapter 7)
This scenario shows leaves floating in the air, occasionally blown around. It is not a game, or any completed project in any sense, but it gives a good first look at collision detection and lists.

Newton's Lab (Chapter 8)
Newton's Lab is a simulation of the motion of stars and planets in space. Gravity plays a central role here. We also make a variant of this that combines gravity with making music, ending up with musical output triggered by objects under gravitational movement.

Asteroids 2 (Chapter 9)
We come back to the asteroids example from Chapter 2. This time, we investigate more fully how to implement it and add some more game elements.

Loop Practice (Chapter 9)
As the name suggests: a scenario with the sole purpose of reinforcing the use of loops. This scenario could also be used much earlier for similar exercises.

Greeps (Interlude 2)
Alien creatures land on earth to collect tomatoes. This scenario is a competition: Program the *greeps* so that they collect as many tomatoes in a limited time.

Color Chart (Chapter 10)
A small scenario just to display a chart of RGB colors.

Smoke (Chapter 10)
This scenario demonstrated a visual effect: smoke trailing a moving ball. In general, it serves to discuss dynamic drawing, to create more interesting visuals.

Path Follower (Chapter 10)
A small scenario demonstrating a creature following a colored path on the ground. This example is used to practice more work with color.

Foxes and Rabbits (Chapter 11)
A predator/prey simulation. This scenario is fairly complete, and we use it to make some experiments and gain some understanding about the nature of simulations.

Ants (Chapter 11)
A simulation of ant colonies searching for food, communicating via drops of pheromones left on the ground.

Simple Camera (Chapter 12)
Showing a camera image on screen, using the Microsoft Kinect.

Greenscreen (Chapter 12)
Using Kinect input to create a greenscreen effect (placing a user in front of a fixed background image).

Stick Figure (Chapter 12)
A demonstration of skeleton tracking with the Microsoft Kinect.

Body Paint (Chapter 12)
We extend the skeleton tracking to allow multiple users to paint on screen by waving their hands in the air. Again, making use of the Microsoft Kinect.

Kinect Pong (Chapter 12)
A very simple game, but this time with gesture input instead of keyboard control.

Fred With Radio (Chapter 12)
A last demo scenario for the Microsoft Kinect. We do not discuss this scenario in the chapter, but it serves as a model demo for studying how a cartoon character could be controlled by gestures.

The following scenarios are presented in Chapter 13, and selected aspects of them briefly discussed. They are intended as inspiration for further projects.

Marbles
A simulation of a marble board game. Marbles have to be cleared of the board within a limited number of moves. Contains simple physics.

Lifts
A start of a lift simulation. Incomplete at this stage—can be used as a start of a project.

Boids
A demo showing flocking behavior: A flock of birds flies across the screen, aiming to stick together while avoiding obstacles.

Explosion
A demo of a more sophisticated explosion effect.

Breakout
This is the start of an implementation of the classic Breakout game. Very incomplete, but with an interesting visual effect.

Platform jumper
A demo of a partial implementation of an ever-popular genre of games: platform jumpers.

Wave
This scenario is a simple demonstration of a physical effect: the propagation of a wave on a string.

Map
A scenario showing use of live data from the Internet, in this case Google maps.

About the companion website

Additional material and resources for this book can be found at
http://www.greenfoot.org/book/

For students:

- The Greenfoot software
- The scenarios discussed in this book
- The Greenfoot Gallery—a scenario showcase
- Tutorial videos
- A discussion forum
- Technical support

For Instructors:

- The "Greenroom," a free, instructor-only community site containing many teaching resources, worksheets, project ideas, and a discussion forum. Sign up here and talk to thousands of other instructors who are using Greenfoot.
  http://greenroom.greenfoot.org
- Scenarios are available to qualified instructors. Contact your Pearson representative or visit the Pearson Instructor Resource Center.
  http://www.pearsonhighered.com/irc

Acknowledgments

This book is the tip of an iceberg. It is an introduction to programming with Java, but this kind of approach would not be possible without the Greenfoot ecosystem. This book builds on many years of work by several people who have helped build Greenfoot.

The book rests first and foremost on the software itself—Greenfoot—but this is not the whole story. Much time and effort has gone into the design of websites (the Greenfoot community website, the Greenroom), development of material, building and supporting a user community, workshops and outreach, and a number of people have played very important roles in this.

Poul Henriksen was the first person to join me in this project. He started the Greenfoot implementation as part of his Masters thesis and was the main contributor to the software for many of the early years. Davin McCall, Bruce Quig, and Neil Brown are the next wave of designers and developers who have worked on Greenfoot for many years and shaped large parts of the design and implementation of the system as it is today. It is not easy to maintain a software system of this size with such few people and resources, but all are outstanding programmers and have managed to develop a system that has survived for almost ten years so far and runs with few problems on millions of computers around the world. This is an outstanding achievement, and I have been very lucky to have these people on my team.

Other important contributions to the ecosystem were by Ian Utting and our more recent team members, Amjad Altadmri and Fabio Hedayioglu. A wide variety of activities has helped to make the Greenfoot community what it is today.

The development of Greenfoot is being supported by Oracle Inc., through charitable donations over many years. Their consistent and ongoing support have allowed us to maintain our group; without this, Greenfoot would not exist. We are very grateful for their commitment and substantial contribution to the education community.

The people at Pearson Education have struggled on bravely in the face of the many delays caused by my missing of every possible deadline for sending in this manuscript. Tracy Johnson has worked with me on this book from the very beginning, through the first edition, and now the second one. She has been consistently positive, excited and encouraging, and her support has made a huge difference. Camille Trentacoste and Carole Snyder have done a lot of the important detail work to get this book produced, and I am grateful for their input and help.

The first edition of this book was reviewed by a number of people who have provided very detailed, thoughtful and useful feedback. They are Carolyn Oates, Damianne President, Detlef Rick, Gunnar Johannesmeyer, Josh Fishburn, Mark Hayes, Marla Parker, Matt Jadud, Todd O'Bryan, Lael Grant, Jason Green, Mark Lewis, Rodney Hoffman, and Michael Kadri. They helped spotting many errors and pointed out many opportunities for improvement. Josh Buhl and Adrienne Decker made a number of very useful suggestions after the publication of the first edition that have helped improve the examples for the second edition.

I am very grateful to Kerstin Wachholz for her expert proofreading—she found and fixed many of my errors and removed the warts of my language—, and to my good friend Michael Caspersen for providing encouragement very early in the project that was very important to me, partly because it helped improve the book, but most importantly because it encouraged me to believe that the idea of the Greenfoot system itself might be interesting to teachers and worthwhile completing.

About the 2nd edition

This is the second edition of this book. It tries to stick with what worked well the first time around, and to improve the parts that were not as smooth as they could have been.

We maintain the overall style of the book: the hands-on presentation of programming projects, the practical work interspersed with discussion and explanation, and the general tone. This has worked very well.

However, there were points in the first edition where readers found progression challenging when the pace picked up in the second half of the book. We have now added two chapters to introduce some concepts more slowly and gradually, and to provide more practice with the most difficult concepts. We have also added a significant amount of exercises to each chapter to provide much more practice and reinforcement of the concepts covered. This includes the presentation and use of many more practice scenarios.

We have also added a chapter about programming Greenfoot with the Microsoft Kinect. While not every reader can make use of this (because it requires having the hardware available), the level of enthusiasm and excitement that these examples have generated when we presented them in workshops justify, in our view, inclusion here. There is so much potential.

And, of course, the book has been updated to make use of new features of more recent versions of the Greenfoot software. We have adapted Greenfoot to make some popular tasks possible or easier and to illustrate some concepts better. The book incorporates this in the new scenarios.

Overall, we hope that the added material serves to make your path through the maze that is the learning of programming even more smooth and more interesting.

This page intentionally left blank

Introduction

Welcome to Greenfoot! In this book, we will discuss how to program graphical computer programs, such as simulations and games, using the Java Programming Language and the Greenfoot environment.

There are several goals in doing this: one is to learn programming, another is to have fun along the way. While the examples we discuss in this book are specific to the Greenfoot environment, the concepts are general: working through this book will teach you general programming principles in a modern, object-oriented programming language. However, it will also show you how to make your own computer game, a biology simulation, or an on-screen piano.

This book is very practically oriented. Chapters and exercises are structured around real, hands-on development tasks. First, there is a problem that we need to solve, then we look at language constructs and strategies that help us solve the problem. This is quite different from many introductory programming textbooks that are often structured around programming language constructs.

As a result, this book starts with less theory, and more practical activity than most programming books. This is also the reason we use Greenfoot: It is the Greenfoot environment that makes this possible. Greenfoot allows us to play. And that does not only mean playing computer games; it means playing with programming: we can create objects, move them around on screen, call their methods, and observe what they do, all interactively and easily. This leads to a more hands-on approach to programming than what would be possible without such an environment.

A more practical approach does not mean that the book does not cover the necessary theory and principles as well. It's just that the order is changed. Instead of introducing a concept theoretically first and then doing some exercises with it, we often jump right in and use a construct, initially explaining only as much as necessary to solve the task at hand, then come back to the theoretical background later. We typically follow a spiral approach: we introduce some aspects of a concept when we first encounter it, then revisit it later in another context, and gradually deepen our understanding.

The emphasis throughout is to make the work we do interesting, relevant, and enjoyable. There is no reason why computer programming has to be dry, formal, or boring. Having fun along the way is okay. We think we can manage to make the experience interesting and pedagogically sound at the same time.

This book can be used both as a self-study book or as a textbook in a programming course. Exercises are worked into the text throughout the book—if you do them all, you will come out of this as a fairly competent programmer.

The projects discussed in this book are easy enough that they can be managed by high school students, but they are also open and extendable enough that even seasoned programmers can find interesting and challenging aspects to do. While Greenfoot is an educational environment, Java is not a toy language. Since Java is our language of choice for this book, the projects discussed here (and others you may want to create in Greenfoot) can be made as complex and challenging as you like.

While it is possible to create simple games quickly and easily in Greenfoot, it is equally possible to build highly sophisticated simulations of complex systems, possibly using artificial intelligence algorithms, agent technology, database connectivity, network communication, or anything else you can think of. Java is a very rich language that opens the whole world of programming, and Greenfoot imposes no restrictions as to which aspects of the language you can use.

In other words: Greenfoot scales well. It allows easy entry for young beginners, but experienced programmers can also implement interesting, sophisticated scenarios.

Programming is a creative discipline, and Greenfoot is a tool that helps you build what you invent.

CHAPTER

1 Getting to know Greenfoot

topics: the Greenfoot interface, interacting with objects, invoking methods, running a scenario

concepts: object, class, method call, parameter, return value

This book will show you how to develop computer games and simulations with Greenfoot, a development environment. In this chapter, we shall take a look at Greenfoot itself and see what it can do and how to use it. We do this by trying out some existing programs.

Once we are comfortable with using Greenfoot, we shall jump right into writing a game ourselves.

The best way to read this chapter (and indeed the whole book) is by sitting at your computer with Greenfoot open on your screen and the book open on your desk. We will regularly ask you to do things in Greenfoot while you read. Some of the tasks you can skip; however, you will have to do some in order to progress in the chapter. In any case, you will learn most if you follow along and do them.

At this stage, we assume that you have already installed the Greenfoot software and the book scenarios (described in Appendix A). If not, read through the appendix first.

1.1 Getting started

Start Greenfoot and open the scenario *leaves-and-wombats* from the *Greenfoot book scenarios* folder. You can do this by choosing *Scenario–Open*[1] from the menu.

[1] We use this notation to tell you to select functions from the menu. Scenario–Open refers to the *Open* item in the *Scenario* menu.

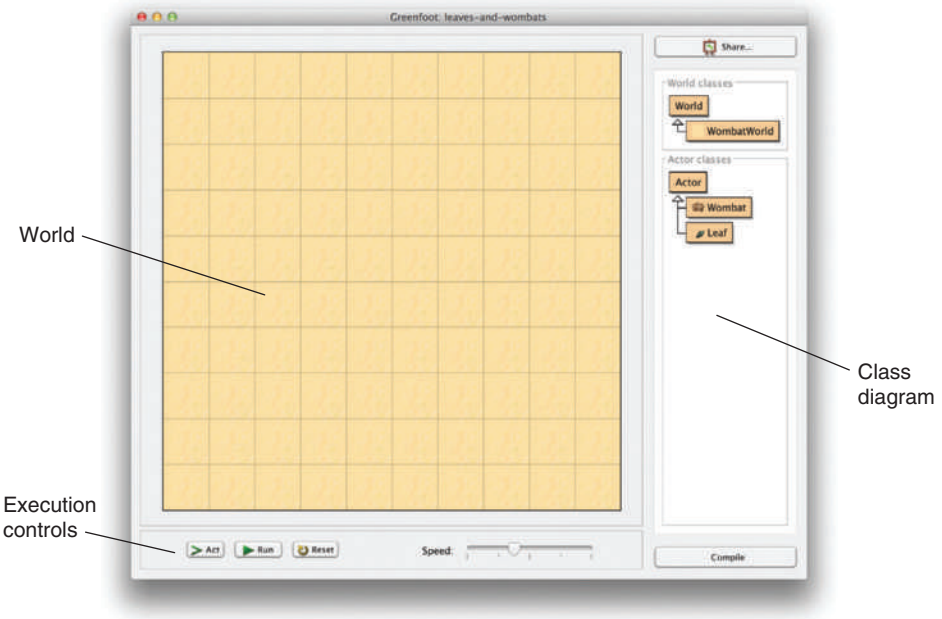


Figure 1.1
The Greenfoot main window

You will now see the Greenfoot main window, with the scenario open, looking similar to Figure 1.1.

The main window consists of three main areas and a couple of extra buttons. The main areas are:

- The *world*. The largest area covering most of the screen (a sand-colored grid in this case) is called the world. This is where the program will run and we will see things happen.
- The *class diagram*. The area on the right with the beige-colored boxes and arrows is the class diagram. We shall discuss this in more detail shortly.
- The *execution controls*. The *Act*, *Run*, and *Reset* buttons and the speed slider at the bottom are the execution controls. We'll come back to them in a little while, too.

1.2 Objects and classes

We shall discuss the class diagram first. The class diagram shows us the classes involved in this scenario. In this case, they are `World`, `WombatWorld`, `Actor`, `Wombat`, and `Leaf`.

Concept
Greenfoot scenarios consist of a set of **classes**.

We shall be using the Java programming language for our projects. Java is an *object-oriented* language. The concepts of classes and objects are fundamental in object orientation.

Let us start by looking at the `Wombat` class. The class `Wombat` stands for the general concept of a wombat—it describes all wombats. Once we have a class in Greenfoot, we

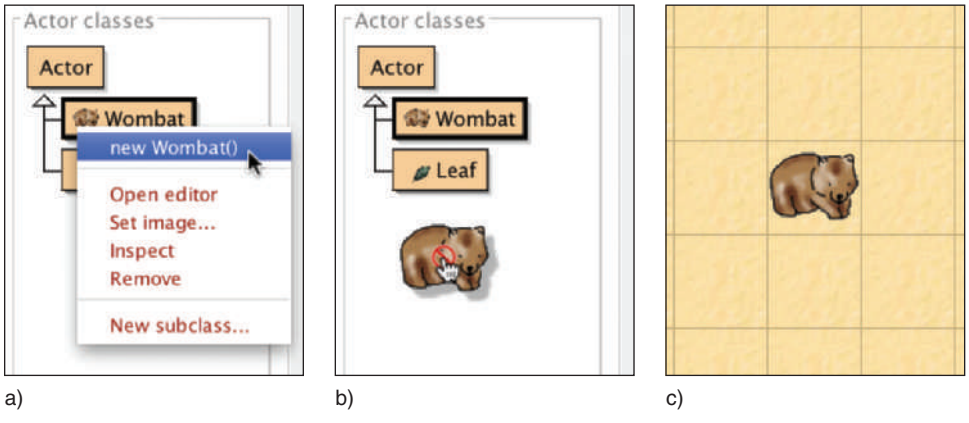
can create *objects* from it. (Objects are also often referred to as *instances* in programming—the two terms are synonyms.)

A wombat, by the way, is an Australian marsupial (Figure 1.2). If you want to find out more about them, do a Web search—it should give you plenty of results.

Right-click[3] on the `Wombat` class, and you will see the *class menu* pop up (Figure 1.3a). The first option in that menu, *new Wombat( )*, lets us create new wombat objects. Try it out.

You will see that this gives you a small picture of a wombat object, which you can move on screen with your mouse (Figure 1.3b). Place the wombat into the world by clicking anywhere in the world (Figure 1.3c).

Figure 1.2
A wombat[2]



a) b) c)

Figure 1.3
a) The class menu
b) Dragging a new object
c) Placing the object

[2] Image source: Marco Tomasini/Fotolia

[3] On Mac OS, use ctrl-click instead of right-click if you have a one-button mouse.

Concept

Many **objects** can be created from a **class**.

Once you have a class in Greenfoot, you can create as many objects from it as you like.

Exercise 1.1 Create some more wombats in the world. Create some leaves.

Currently, only the `Wombat` and `Leaf` classes are of interest to us. We shall discuss the other classes later.

1.3 Interacting with objects

Concept

Objects have **methods**. Invoking these performs an action.

Once we have placed some objects into the world, we can interact with these objects by right-clicking them. This will pop up the *object menu* (Figure 1.4). The object menu shows us all the operations this specific object can perform. For example, a wombat's object menu shows us what this wombat can do (plus two additional functions, *Inspect* and *Remove*, which we shall discuss later).

In Java, these operations are called *methods*. It cannot hurt to get used to standard terminology straight away, so we shall also call them methods from now on. We can *invoke* a method by selecting it from the menu.

Exercise 1.2 Invoke the `move()` method on a wombat. What does it do? Try it several times. Invoke the `turnLeft()` method. Place two wombats into your world and make them face each other.

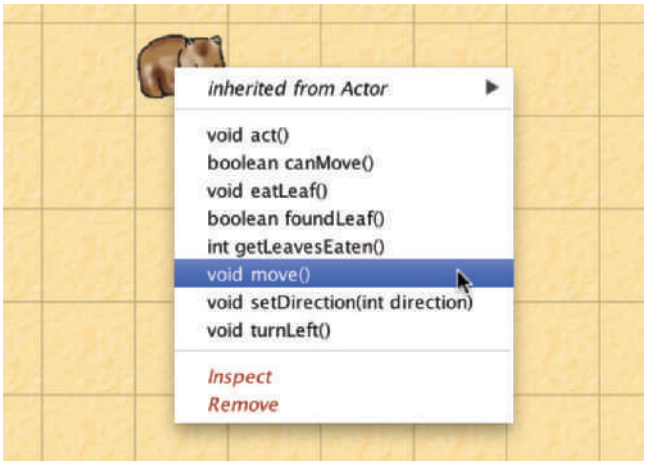


Figure 1.4
The wombat's object menu

In short: we can start to make things happen by creating objects from one of the classes provided, and we can give commands to the objects by invoking their methods.

Let us have a closer look at the object menu. The *move* and *turnLeft* methods are listed as:

```
void move()
void turnLeft()
```

We can see that the method names are not the only thing shown. There is also the word *void* at the beginning and a pair of parentheses at the end. These two cryptic bits of information tell us what data goes into the method call, and what data comes back from it.

1.4 Return types

Concept

The **return type** of a method specifies what a method call will return.

The word at the beginning is called the *return type*. It tells us what the method returns to us when we invoke it. The word `void` means "nothing" in this context: methods with a `void` return type do not return any information. They just carry out their action, and then stop.

Any word other than `void` tells us that the method returns some information when invoked, and of what type that information is. In the wombat's menu (Figure 1.4), we can also see the words `int` and `boolean`. The word `int` is short for "integer" and refers to whole numbers (numbers without a decimal point). Examples of integer numbers are 3, 42, –3, and 12000000.

Concept

A method with a **void** return type does not return a value.

The type `boolean` has only two possible values: `true` and `false`. A method that returns a `boolean` will return either the value `true` or the value `false` to us.

Methods with `void` return types are like commands for our wombat. If we invoke the `turnLeft` method, the wombat obeys and turns left. Methods with non-void return types are like questions. Consider the `canMove` method:

```
boolean canMove()
```

When we invoke this method, we see a result similar to that shown in Figure 1.5, displayed in a dialog box. The important information here is the word "true," which

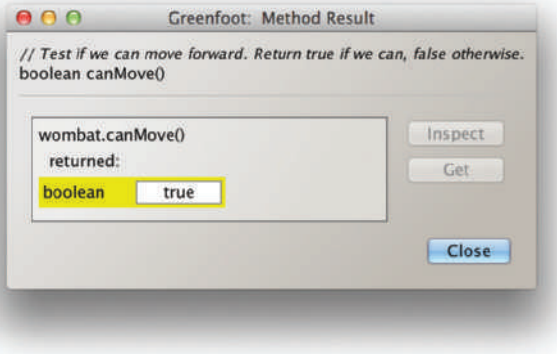


Figure 1.5
A method result

was returned by this method call. In effect, we have just asked the wombat "*Can you move?*", and the wombat has answered by saying "*Yes!*" (true).

Exercise 1.3 Invoke the `canMove()` method on your wombat. Does it always return *true*? Or can you find situations in which it returns *false*?

Try out another method with a return value:

```
int getLeavesEaten()
```

Using this method, we can get the information how many leaves this wombat has eaten.

Concept

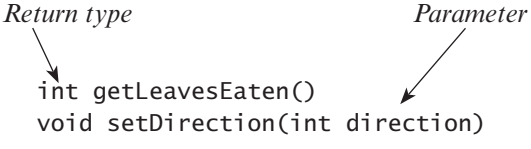
Methods with void return types represent **commands**; methods with non-void return types represent **questions**.

Exercise 1.4 Using a newly created wombat, the `getLeavesEaten()` method will always return zero. Can you create a situation in which the result of this method is not zero? (In other words: can you make your wombat eat some leaves?)

Methods with non-void return types usually just tell us something about the object (*Can it move? How many leaves has it eaten?*), but do not change the object. The wombat is just as it was before we asked it about the leaves. Methods with void return types are usually commands to the objects that make it do something.

1.5 Parameters

The other bit in the method menu that we have not yet discussed is the parentheses after the method name.



Concept

A **parameter** is a mechanism to pass additional data to a method.

The parentheses after the method name hold the *parameter list*. This tells us whether the method requires any additional information to run, and if so, what kind of information.

Concept

Parameters and return values have **types**. Examples of types are **int** for numbers, and **boolean** for true/false values.

If we see only a pair of parentheses without anything else between it (as we have in all methods so far), then the method has an *empty parameter list*. In other words, it expects no parameters—when we invoke the method, it will just run. If there is anything between the parentheses, then the method expects one or more parameters—additional information that we need to provide.

Let us try out the *setDirection* method. We can see that it has the words *int direction* written in its parameter list. When we invoke it, we see a dialog box similar to the one shown in Figure 1.6.

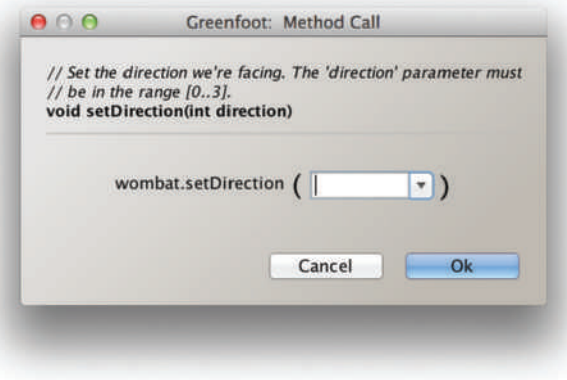


Figure 1.6
A method call dialog

The words *int direction* tell us that this method expects one parameter of type *int*, which specifies a *direction*. A parameter is an additional bit of data we must provide for this method to run. Every parameter is defined by two words: first the parameter type (here: *int*) and then a name, which gives us a hint what this parameter is used for. If a method has a parameter, then we must provide this additional information when we invoke the method.

In this case, the type *int* tells us that we now should provide a whole number, and the name suggests that this number somehow specifies the direction to turn to.

At the top of the dialog is a comment that tells us a little more: the direction parameter should be between 0 and 3.

Exercise 1.5 Invoke the `setDirection(int direction)` method. Provide a parameter value and see what happens. Which number corresponds to which direction? Write them down. What happens when you type in a number greater than 3? What happens if you provide input that is not a whole number, such as a decimal number (2.5) or a word (three)?

Concept

The specification of a method, which shows its return type, name, and parameters is called its **signature**.

The `setDirection` method expects only a single parameter. Later, we shall see cases where methods expect more than one parameter. In that case, the method will list all the parameters it expects between the parentheses.

The description of each method shown in the object menu, including the return type, method name, and parameter list, is called the *method signature*.

We have now reached a point where you can do the main interactions with Greenfoot objects. You can create objects from classes, interpret the method signatures, and invoke methods (with and without parameters).

1.6 Greenfoot execution

There is one other way of interacting with Greenfoot objects: The execution controls.