This course has already ended.
The latest instance of the course can be found at: O1: 2023

Luet oppimateriaalin englanninkielistä versiota. Mainitsit kuitenkin taustakyselyssä osaavasi suomea. Siksi **suosittelemme, että käytät suomenkielistä versiota**, joka on testatumpi ja hieman laajempi ja muutenkin mukava.
Suomenkielinen materiaali kyllä esittelee englanninkielisetkin termit. Myös suomenkielisessä materiaalissa käytetään ohjelmien koodissa englanninkielisiä nimiä kurssin alkupään johdantoesimerkkejä lukuunottamatta.
Voit vaihtaa kieltä A+:n valikon yläreunassa olevasta painikkeesta. Tai tästä: **Vaihda suomeksi**.

Chapter 8.1: Robots

About This Page

Questions Answered: Can I apply what I've learned on a larger program?

Topics: A particular simulator app. There are no new topics per se, but we will discuss a program that is more complex than any of the previous ones.

What Will I Do? Program.

Rough Estimate of Workload? A couple of hours?

Points Available: B50.

Related Modules: Robots **(new)**.

Introduction

Our virtual robots live in a grid. The robots take turns to act. Robot actions largely involve moving and turning.

Much of Week 8 revolves around a programming assignment in which you construct a robot simulator. In this simulator, a "robot world" is essentially a grid where "robots" of very little brain conduct their virtual existence.

The assignment has been broken down in nine parts. The first two are in this chapter; the rest are in Chapters 8.2 and 8.3.

Before we get started in earnest, let's form an overview of the program that you'll be working on.

The Robots Module

The Robots module contains two packages. The classes in `o1.robots` constitute the robot simulator's internal model; `o1.robots.gui` provides a user interface. The user interface is ready for use as given and we won't discuss it further here.

We'll build the simulator on the same `Grid` class that you used in Chapters 7.4 and 7.5.

The following table briefly describes each of the simulator's main components. Below the table, you'll find a diagram of the components' relationships.

Package	Component	Description	Status
`o1`	class `GridPos`	Represents pairs of coordinates. (Familiar from Chapter 6.3 and Week 7.)	ready
`o1`	abstract class `Grid`	Represents grids in general. (Familiar from Week 7.)	ready
`o1`	sealed abstract class `CompassDir`	Represents the main compass directions. There are exactly four instances of this class, which stand for north, east, south, and west. (Familiar from Chapter 6.3.)	ready
`o1.robots`	class `RobotBody`	Represents the robots' physical form. Each instance of `RobotBody` has properties such as location and facing. A robot can be either intact or broken.	partially implemented
`o1.robots`	abstract class `RobotBrain`	Represents the general properties of the robots' "artificial intelligence".	partially implemented
`o1.robots`	classes `Spinbot`, `Nosebot`, `Staggerbot`, `Lovebot`, and `Psychobot`	Subclasses of `RobotBrain`. Each of these subclasses represents a different sort of robot behavior.	`Spinbot` partially implemented; others missing
`o1.robots`	trait `Square`	Represents a single square of a robot world in general terms.	ready
`o1.robots`	class `Floor`	Extends `Square`. Represents floor squares that the robots move on.	partially implemented
`o1.robots`	singleton object `Wall`	Extends `Square`. Represents a wall. (A single object is enough for this purpose, since all walls are identical.)	partially implemented
`o1.robots`	class `RobotWorld`	A subclass of `Grid`. Represents robot-inhabited grids that are composed of `Square` objects. A `RobotWorld` object also tracks which robot's turn it is to act next.	partially implemented

Part 0 of 9: Answer Some Questions

Before you go on

Try to get a general sense of the Robots program by reading the description above and the Scaladocs. Browse the source code, too.

Points 8 **2 / 2** My submissions **3 / 5** Deadline Wednesday, 4 November 2020, 12:00
To be submitted alone or in groups of 2

⚠ This course has been archived (Tuesday, 31 August 2021, 23:59).

Assignment 1

You got 2/2 points from this questionnaire.

Question 1 **1 / 1**

For this question, assume that the program has already been implemented and works as specified. Also assume that we've created a robot world and a variable `testWorld` that refers to it. Moreover, we've created two robots and added them to the world. Neither of the two robots has yet had a turn to act. We now execute the following lines of code:

```
testWorld.advanceTurn()
testWorld.advanceFullRound()
testWorld.advanceTurn()
testWorld.addRobot(GridPos(1, 1), North)  // let's say this square is previously empty
testWorld.advanceTurn()
testWorld.advanceFullRound()
testWorld.advanceTurn()
```

Which of the robots has the next turn to act?

- ○ the one that was added first
- ○ the one that was added second
- ◉ the one that was added third
- ○ none of the robots will get a turn

- Correct. The first two `advanceTurn`s bring the turn back to the first robot. The next two again move those first two robots, leaving the turn with the newcomer. Calling `advanceFullRound` moves all the robots and have no impact on the answer.

Question 2 **1 / 1**

Read the claims below and select all the correct ones.

Note: below, the expression "each object knows" means "each object has stored in its instance variables or can trivially determine with a simple method call". This is not a trick question. Its only purpose is to encourage you to study the Scaladocs and learn about the given program.

- ☑ Each `RobotWorld` object "knows" which squares it contains.
- ☑ Each `RobotBody` object "knows" its own coordinates in a robot world.
- ☑ Each `RobotBody` object "knows" the square that it is located in.
- ☐ Each `Square` object "knows" which RobotWorld it is part of.
- ☐ Each `Square` object "knows" its coordinates within a robot world.
- ☑ Each `Square` object "knows" which robot (if any) it contains.

Submit

Part 1 of 9: Fundamental Repairs to `RobotWorld`

You've been given a whole bunch of code, much of which works in principle, but the program is not ready to run. IntelliJ spews a stream of errors.

`RobotWorld` has been only partially implemented. Fix it:

1. The error messages indicate that `RobotWorld` doesn't have the variables `width` and `height` as expected. It should inherit these methods from `Grid`. Add an `extends` clause that makes `RobotWorld` a subclass of `Grid`.
   - You may wish to look at `GameBoard` in the Viinaharava game for inspiration.
   - Read the documentation carefully. Make sure you pass in the right numbers as constructor parameters to the superclass.
   - Once you're done with this step, try launching the program via `o1.robots.gui.RobotApp`. You should see a robot world that's completely dark.
   - Be sure to give `Grid` the type parameter it needs. (If you receive error messages that feature `Nothing` as the type of the grid's squares, you probably overlooked this.)
2. Repair the `RobotWorld` method `initialSquare`, which now fills the entire world with walls. The method is private and not detailed in the Scaladocs, but there's a comment in the given code that explains what it should do: put walls on the edges and floors in the middle.
   - That is, you'll need to check the coordinates to determine whether you should create a `Floor` or use a reference to the `Wall` singleton.
3. Relaunch the application. You should be able to create empty robot worlds, but clicking the floor squares won't let you add robots or walls.

Part 2 of 9: Adding Content

1. Implement the `RobotWorld` method `addWall`.
   - Hint: use `update` in class `Grid`.
2. Run the program again and try right-clicking floors to add walls.
3. Implement `addRobot` in the same class.
   - Note that `addRobot` needs to handle several interrelated subtasks. Make sure you attend to each of the things mentioned in the Scaladoc.
   - Reveal additional hints below if you feel you want them.
4. Try adding `Spinbot`s in the GUI. They should appear in the robot world, but they don't do anything yet. You can break them and repair them, though.

Hint: subtasks in `addRobot`

Show the hint Hide the hint

The method should: 1) create a robot; 2) add it to the end of the robot list; 3) add it in the appropriate square within the robot world; and 4) return a reference to the added robot.

Tools for each subtask: 1) create a new `RobotBody` object; 2) update the list in `this.robots`; 3) pick the right square and call *that square's* `addRobot` method; and 4) return a reference to the `RobotBody` object that you created.

Hint: picking the appropriate square in `addRobot`

Show the hint Hide the hint

A `RobotWorld` is a `Grid`.

Grids have an `elementAt` method for accessing a single element (square) of the grid.

Use that method to pick the appropriate square. Then call the square's robot-adding method.

Submit your solution to Parts 1 and 2. The assignment continues in upcoming chapters.

Points 8 **48 / 48** My submissions **4 / 10** Deadline Wednesday, 4 November 2020, 12:00
To be submitted alone or in groups of 2

⚠ This course has been archived (Tuesday, 31 August 2021, 23:59).

Assignment 2 (Robots 1)

Select your files for grading

RobotWorld.scala

Choose File No file chosen

Submit

Feedback

Please note that this section must be completed individually. Even if you worked on this chapter with a pair, each of you should submit the form separately.

Accepted My submissions 1

⚠ This course has been archived (Tuesday, 31 August 2021, 23:59).

Time spent: (*) Required

Please estimate the total number of minutes you spent on this chapter (reading, assignments, etc.). You don't have to be exact, but if you can produce an estimate to within 15 minutes or half an hour, that would be great.

180

Written comment or question:

You aren't required to give written feedback. Nevertheless, please do ask something, give feedback, or reflect on your learning! (However, the right place to ask urgent questions about programs that you're currently working on isn't this form but Piazza or the lab sessions. We can't guarantee that anyone will even see anything you type here before the weekly deadline.)

Submit an update

Credits

Thousands of students have given feedback that has contributed to this ebook's design. Thank you!

The ebook's chapters, programming assignments, and weekly bulletins have been written in Finnish and translated into English by Juha Sorva.

The appendices (glossary, Scala reference, FAQ, etc.) are by Juha Sorva unless otherwise specified on the page.

The automatic assessment of the assignments has been developed by: (in alphabetical order) Riku Autio, Nikolas Drosdek, Joonatan Honkamaa, Jaakko Kantojärvi, Niklas Kröger, Teemu Lehtinen, Strasdosky Otewa, Timi Seppälä, Teemu Sirkiä, and Aleksi Vartiainen.

The illustrations at the top of each chapter, and the similar drawings elsewhere in the ebook, are the work of Christina Lassheikki.

The animations that detail the execution Scala programs have been designed by Juha Sorva and Teemu Sirkiä. Teemu Sirkiä and Riku Autio did the technical implementation, relying on Teemu's Jsvee and Kelmu toolkits.

The other diagrams and interactive presentations in the ebook are by Juha Sorva.

The O1Library software has been developed by Aleksi Lukkarinen and Juha Sorva. Several of its key components are built upon Aleksi's SMCL library.

The pedagogy of using O1Library for simple graphical programming (such as `Pic`) is inspired by the textbooks *How to Design Programs* by Flatt, Felleisen, Findler, and Krishnamurthi and *Picturing Programs* by Stephen Bloch.

The course platform A+ was originally created at Aalto's LeTech research group as a student project. The open-source project is now shepherded by the Computer Science department's edu-tech team and hosted by the department's IT services. Markku Riekkinen is the current lead developer; dozens of Aalto students and others have also contributed.

The A+ Courses plugin, which supports A+ and O1 in IntelliJ IDEA, is another open-source project. It was created by Nikolai Denissov, Olli Kiljunen, and Nikolas Drosdek with input from Juha Sorva, Otto Seppälä, Arto Hellas, and others.

For O1's current teaching staff, please see Chapter 1.1.

Additional credits appear at the ends of some chapters.