CS112 Lecture: Introduction to Karel J. Robot

Last revised 1/17/08

Objectives:

1. To introduce Karel J. Robot as an example of an object-oriented system.
2. To explain the mechanics of writing simple Karel programs.

Materials:

1. Interactive demonstration program (Helicopter)
2. BlueJ project containing InitialDemo.world,.java and HarvestTwoRows.world, .java
3. Projectable version of problem 2-5 from complete book, plus Problem2_5.world
4. Projectable version of problem 2-7 from complete book, plus Problem2_7.world

I. Introduction

A. Before beginning our study of Java, we will look at a simpler example of an object-oriented system: Karel J. Robot. By so doing, we will be able to cover many of the key ideas of object-oriented programming in an initial way.

B. Karel the Robot was first introduced about 20 years ago as a vehicle for introducing students to programming, at a time when the dominant language for teaching programming was Pascal. It has been revised several times since then. The current version - called Karel J. Robot - is programmed using Java, and runs using a robot simulator, also written in Java.

C. I believe you did one lab using Karel in CS111. We are using this for first few classes of the course to get a quick look at a number of key programming concepts. We will then explore these concepts (and others) in more depth in the rest of the course.

II. Introduction to the Robots' World

A. As you saw last semester and/or read in the reserve reading, Karel and his relatives are robots who live in a world consisting of east-west Streets and north-south Avenues. At any time, each robot is positioned at the corner of a Street and an Avenue, facing in one of four directions: north, west, south, or east.

B. In addition to the streets, avenues, and robots themselves, the robot world includes impenetrable walls and objects called beepers, which robots can pick up from a corner or put down on a corner on command.

DEMO: Run interactive demonstration (Helicopter) using InitialDemo.world as world. Point out streets, avenues, walls, and beepers; then use new button to create a new robot.

C. Robots in this world have five basic capabilities:

1. Move forward one square in the direction the robot is facing.
2. Turn left 90 degrees.
3. Pick up a beeper.
4. Put down a beeper
5. Turn off upon completing a task.

DEMO: Show each operation

D. These operations are handled independently by each robot. To get a robot to do something, we must send a specific message to it - e.g. to get a robot called "karel" to move, we must send the message

karel.move();

DEMO: Create a second robot - demo sending messages to one or the other, to both, and to neither. (When a button is pressed in the simulator, the appropriate message is sent to each selected robot.)

E. Several of these primitive operations are effective only if certain requirements are met when they are attempted. When a robot is asked to do something it cannot do, it does an *error shutoff*, which is like a turnoff except that it announces that it has been unable to complete its assigned task. (*DEMO EACH FAILURE*)

1. Move is permitted only if the robot's path is not blocked by a wall.

2. A robot can only pick up a beeper if there is one on the corner for it to pick up.

3. A robot can only put down a beeper on a corner if it has one in its beeper bag.

(turnLeft() and turnOff(), however, are always possible.)

F. Let's now consider a demonstration of a single robot performing a complete task.

(REFRESH ROBOT WORLD from InitialDemo.world)

1. The robot will start at the corner of 1st st and 1st avenue, facing East.
2. The robot will go to 2nd and 2nd and pick up the beeper that is there

3. The robot will carry the beeper to 3rd and 3rd and drop it off there.
4. The robot will return to its starting position.
5. The robot will turn off.

   ASK CLASS TO DEVELOP ONE STEP AT A TIME, AND DEMONSTRATE SIMULATION - BEGINNING WITH CREATING THE ROBOT.

III. Robot Programs

A. Thus far, we have been issuing instructions individually to our robots, using a graphical user interface. In practice, robots are normally given tasks to perform, and then left to perform them.

B. A robot program will turn out to be a Java object in its own right, defined by a Java class. (Each program class will be used to create exactly one object, of course.)

C. The class defining the program will have two major parts. In the first part - called the *main program* - we set up the world, construct the program object, and start it executing. In the second part, we order the robot(s) we need and issue instructions to them. The entire program is "wrapped" in some additional statements and punctuation to make it a valid Java program. Part of the form of our programs will also be dictated by some of the requirements of the simulator we will be using in our lab.

D. To produce and run robot programs, we will proceed as follows:

   1. We will create and run our programs using a very simple integrated development environment (designed specifically for educational use) called BlueJ - in fact, we will use BlueJ throughout this course.

      BlueJ works with basic units of work called "projects". In fact, each lab and project you do for the course will be done using its own BlueJ project.

      DEMO: Start BlueJ, open Karel Intro Lecture project

   2. We will decide on a name for the program. For reasons relating to the requirements of Java, this name will have to consist of only letters, digits, and the underscore (_), with the first character being a letter. We will follow the convention for class names in Java, because our programs will, in fact, be Java classes.

      Example: Our initial program will be called InitialDemo

3. We will create a class containing the program. (I've already done this, so I won't be able to demonstrate this now, though later we will create a new one).

   DEMO: Show class InitialDemo on screen - explain structure

   a) prologue comment
   b) import statement to get access to simulator, including robot class
   c) definition of class InitialDemo - opened and closed by { }. The phrase "implements Runnable" indicates that this class defines a method called run() - required by the simulator because this class contains a complete definition of a robot task.
   d) main program - note statements to create world, make it visible, setup some requirements for the simulation
   e) run() method - note this specific name is required.
   f) embedded comments using //

4. It is worth noting that a BlueJ project can contain any number of programs, and a program can consist of any number of classes.

   In fact, the demonstration project we are using now contains two classes, each of which is part of a separate program.

5. When we have finished creating the file containing the definition for the program class, we will compile it into a form that can be interpreted by the system.

   a) Compilation is needed because the Karel simulation we are using cannot directly interpret the language we are using. The compiler translates our program into a language called "Java bytecodes", which can be interpreted by the simulation system.

   b) The result of compilation will be a translated file whose name will be the program name followed by .class - in this case, InitialDemo.class

      DEMO: Open project folder in the Finder. Compile using BlueJ. Note resultant class file

   c) If the program had contained any spelling or grammatical errors, the java compiler would reject it and we would have to fix them before proceeding.

      DEMO: Change the spelling of turnOff to turnoff and compile again, then fix

6. To run the program, we will use a feature of BlueJ that allows us to easily run any class that has a main method.

   DEMO RUNNING THE PROGRAM

   IMPORTANT: You must close the simulation window after every run of the program. (*SHOW*)

7. Note: if the program had contained any logical errors, it might run to completion, but would fail to do what it was supposed to do; or it might encounter an error shutoff.

   DEMO: Delete final move() - note how robot turns off at wrong corner

   DEMO: Delete move() before pickBeeper() - note error shutoff

IV. Programs with More than One Robot

A. Sometimes, it is advantageous to put more than one robot to work on a task.

   DEMO: Show HarvestTwoRows.world using Helicopter

   The task of picking up all the beepers could be done by a single robot, moving down one row and back the other - or by two robots each doing one row.

B. Let's look at a solution that uses the second approach. (For simplicity, we'll let each robot turn off in place when its job is done.)

   DEMO: Show, then run, HarvestTwoRows.java

V. Further Exercise As time permits, do the following as class exercises (develop on the spot). Note that files containing the world descriptions are already in the project.

A. Problem 2.5 from the complete book

   PROJECT

   Enter, compile, and run program

B. Problem 2.7 from the complete book

   PROJECT

   Enter, compile, and run program