CS112 - INTRODUCTION TO PROGRAMMING

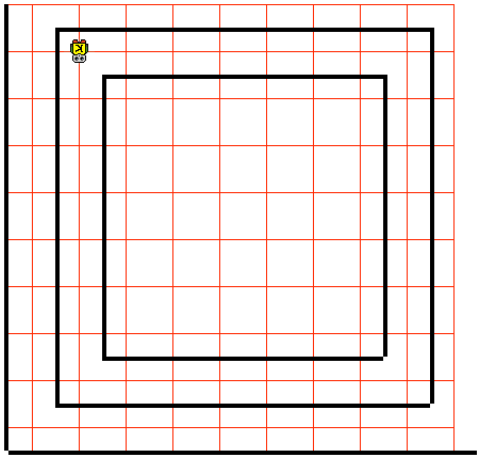
Lab #3 - Due: Friday, February 8, at the start of class

Objectives:

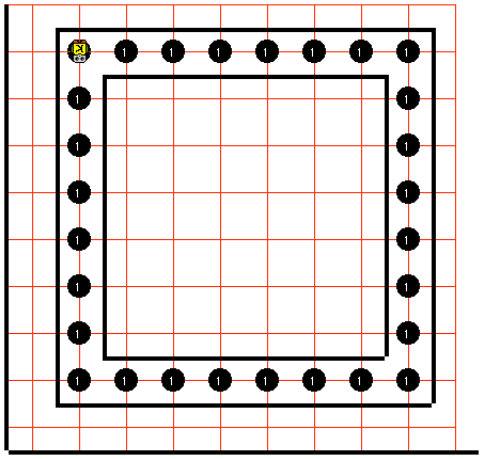
1. To familiarize you with extending existing classes to add new methods
2. To introduce you to using for and while loops
3. To introduce you to using if and if ... else

In this lab session, you will be making a series of modifications to a Karel J. Robot program, resulting in increasingly sophisticated capabilities. You will be supplied with an incomplete initial program which you will progressively modify during the course of the lab.

The problem you will work with is adapted from problem 3.8 in the book on reserve. Karel has taken a job installing carpets (made from beepers) in his world. In the first part of the lab, you will be completing a program that will allow Karel to carpet the building shown below. There must be no "lumps" in the carpet, so Karel must place exactly one beeper on each corner. Subsequent parts of the lab will involve improvements to Karel's proficiency as a carpet layer.



BEFORE



AFTER

- At the start of the lab session, log in to your computer and to your server account on the Computer Science server. (You may use the account for either partner). Be sure to mount *both* your server volume and the common volume.

- Copy the folder labeled `Lab03` from common to your volume, just like you did for Lab02 last week. You will do all your work for this lab in this `Lab03` folder.

- Open `Lab03` as a BlueJ project, as you did last week. You should see two classes, called `Main` and `CarpeterRobot`. You will complete and modify these classes during this lab, and will eventually (in Part IV) create an additional class.

- Edit the class documentation file to include your names and today's date as you did last week.

Part I - Extending class Robot to Create a New Class

An almost complete program for accomplishing the initial task is found in classes `Main` and `CarpeterRobot`.

- Edit both classes to modify the prologue comment appropriately (by adding "Modified by")

- Complete the `CarpeterRobot` class by defining the method `carpetCorridor()`. This code should go at the indicated place in the file - replacing the line you are told to remove. (*Note*: do *not* use a loop for this - simply repeat the statements the required number of times. Loops are coming in the next part!)

  Note that Karel should *first* put a beeper on the corner he is on and then move to the next corner, not vice versa. (Either way would work here, but doing put first will make life simpler later .)

- Compile the modified program by using the Compile button in the main BlueJ window - not the compile button associated with a specific class (which only compiles that class). Correct syntax errors as necessary until you get a clean compile.

- Run the compiled program and make sure it runs successfully, producing a result like that shown as "After" above. Make corrections and recompile if necessary.

- Show the correctly working program to the professor or a TA, and have him/her initial your lab writeup form before proceeding.

- **Print a copy of class `CarpeterRobot` before making the changes specified for the next part.**

Part II - Using for Loops (Definite Iteration)

In the program you just wrote, there are two places where the same sequence of two statements is repeated several times. The program can be made shorter, clearer, and easier to modify by replacing each group of statements with a for loop. You will do this by modifying class `CarpeterRobot`.

- Change the opening comment appropriately.

- Replace the four repetitions of `carpetCorridor(); turnLeft();` in `carpetBuilding()` with a single for loop containing one copy of each statement.

- Replace the seven repetitions of `putBeeper(); move();` in `carpetCorridor()` with a single for loop containing one copy of each statement.

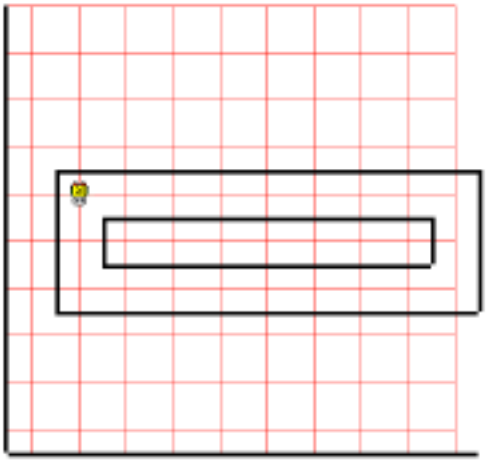
- Compile and test this modified program, fixing errors as necessary until it is correct.

- Show the correctly working program to the professor or a TA, and have him/her initial your lab writeup form before proceeding. Note that the professor / TA will verify *both* correct operation and appropriate use of for in the two places.

- **Print a copy of class `CarpeterRobot` before making the changes specified for the next part.**

Part III - Using while Loops (Indefinite Iteration)

The program you have used thus far is specifically written to carpet a building whose corridors are all of length 8, and would be useless for any other kind of building. In this part, you will modify the program to deal with any size rectangular building - e.g. a building like the following.



- Modify your `CarpeterRobot` class as follows:
  - Change the opening comment appropriately.
  - Replace the for loop in `carpetCorridor()` with an appropriate while loop. The key idea is that Karel should keep laying down carpeting and moving as long as his front is clear. (The `carpetBuilding()` method can still use a for loop, since rectangular buildings always have four sides!)
- Test your program to be sure it still correctly handles the world you used in parts I and II of the lab (`Part1.world`).
- Now modify your `Main` class so that it reads `Part3.world` instead. (This is the world shown in the picture above). You will need to change the name of the file that is read by `readWorld()`, and you will also need to appropriately alter the robot's starting position and initial beeper supply. Test your program again, and be sure it handles this world correctly.
- Double-click on the icon for `Part3.world` to open it in a text editor called TextWrangler, and use Save As ... in the File menu and save a copy as `Mine.world`. (Note that this implies that you will not change the original `Part3.world` file.) Edit this world file to represent some other rectangular building different from either of the two you have used so far. (If you study this file in conjunction with the display it produces, you should be able to figure out its format.) Also edit your `Main` class to read from this file instead, and to start the robot at the appropriate position with the correct number of beepers. Test your program with this world.
- Show the correctly working program (final version using `Mine.world`) to the professor or a TA, and have him/her initial your lab writeup form before proceeding.
- **Print a copy of class `CarpeterRobot` before making the changes specified for the next part.**

Part IV: Using an if Statement (Conditional Instruction)

We said at the outset that part of doing a quality job of carpet-laying is to avoid lumps. Karel got his carpet laying job as a replacement for a previous carpet layer whose work was not up to snuff. As a result, Karel sometimes has to remove the previous robot's work before he can lay down his new carpet.

- ☐ Modify your `Main` class to read its world from `Part4.world`, and to start Karel at 6, 2 with 20 beepers in his beeper bag.

- ☐ Modify the opening comment in your `CarpeterRobot` class appropriately.

- ☐ Modify the `carpetCorridor()` method in your `CarpeterRobot` class to check each corner before laying down a beeper to see if there is one there already. If so, it should pick up the old beeper first. Then, in either case, it should putting down a new one. (The effect is the same as if Karel simply skipped a corner already containing a beeper - but we want Karel to be neat and take up the old carpet and then put down new.)

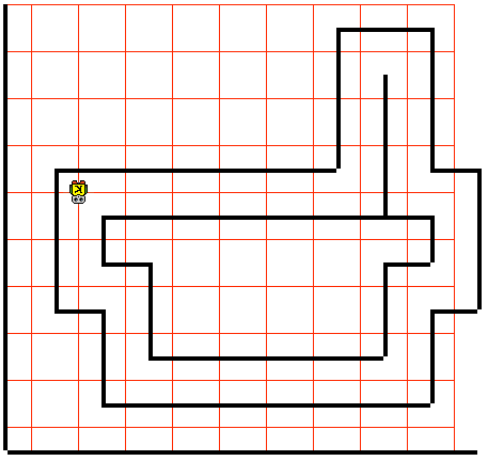
  Use an `if` statement to check to see if you need to pick up old carpeting. You may assume that there will only be one current beeper on any one corner, so you don't need a loop.

- ☐ Compile, test, and run this program, show the working version to a professor or a TA, and have him/her initial your lab writeup before proceeding.

- ☐ **Print a copy of class `CarpeterRobot` before making the changes specified for the next part.**

Part V: A More Sophisticated Carpet-Layer program

For the previous parts, you have been working with a carpeting program that deals only with rectangular buildings. Extending the program to handle more complex shapes like the following turns out to not be difficult: (This world is described in the file `Part5.world`)



Two basic changes are needed, both involving the loop in the `carpetBuilding()` method:

- Instead of always turning left when he has finished with a corridor, Karel must be prepared to turn either left or right as appropriate. How can Karel tell which way to turn? (Answer in the space provided on your writeup form.)
- Instead of having to carpet exactly four corridors, Karel must be prepared to carpet any number of them. (In this particular case, there are 12). How can Karel know when he has done all the corridors? (Answer in the space provided on your writeup form.)

- This time, instead of modifying the `CarpeterRobot` class, you will create a new class called `SmartCarpeterRobot` that deals with this issue.
  - Use the New Class button to create a new class. Choose Robot from the list of choices for type of class, and call it `SmartCarpeterRobot.`
  - Modify the class that BlueJ creates by making it extend `CarpeterRobot` instead of `Robot`.
  - Write appropriate prologue comments.
  - Give this new class a method called `turnTheCorrectWay()`. This method should make the decision whether to turn left or right, based on the criterion you identified above. There are many ways this can be done, but for the purposes of this lab (in order to get experience with using key features of Java), you must do it as follows:

    Use an `if .. else` statement with a `leftIsClear()` method to turn one way or the other as appropriate.

    Note: it is possible to do this without using an "`else`" and a `leftIsClear()` method. However, please do it this way to gain experience with using"`else`" and with defining a `boolean` method.
  - Create a test predicate called `leftIsClear()`.

    See the discussion of creating new predicates in section 5.2.1 of the book. Be sure you understand this section thoroughly before writing your code.
  - You will also need to create a `turnRight()` method.
  - Define a `carpetBuilding()` method in this new class (which overrides the inherited one). This should be just like the method of the same name in `CarpeterRobot`, except :
    - Use a while loop that stops when all the corridors are carpeted, using the criterion you identified above, instead of using a for loop that counts four corridors.
    - Call your new `turnTheCorrectWay()` method instead of `turnLeft()`.
- Modify the `Main` class as follows:
  - Change the line that reads the world file to read `Part5.world`.
  - Change the line that creates a new robot to create a `SmartCarpeterRobot` instead of just a plain, ordinary `CarpeterRobot`, and to start the robot at 6,2 with the 30 beepers.
- Compile, test, and run this program, show the working version to a professor or a TA, and have him/her initial your lab writeup before proceeding.
- Print a copy of the new class `SmartCarpeterRobot`.
- **Log off the server and the computer before leaving**.

Attach the following to the writeup form before turning it in:

Printouts of `CarpeterRobot` class from parts I, II, III, IV, `SmartCarpeterRobot` from part V.

LEAVE A COPY OF ALL THE FILES YOUR PRODUCED ON THE SERVER FOR POSSIBLE CHECKING DURING THE GRADING PROCESS