Karel J Robot Chapter 5

The two new instructions are *for* loops and *while* loops. Both instructions will repeatedly execute any instructions the robot understands. When we program a robot, having it repeat instructions a **known** number of times is sometimes necessary. We previously handled this problem by writing (copy/pasting) the instructions as many times as needed. The *for* loop command gives us a mechanism which allows the robot to repeat one or more instructions a known number of times.

```
    public void turnRight()
    {
        for (int counter=1; counter<=3; counter++)
        {
               turnLeft();
        }
    }

    public static void main(String[] args)
    {
        SuperRobot karel=new SuperRobot(3, 4, East, 0);
        for (int steps=1; steps<=4; steps++)
        {
               karel.turnLeft();
               karel.move();
               karel.turnRight();
               karel.move();
        }
    }
```

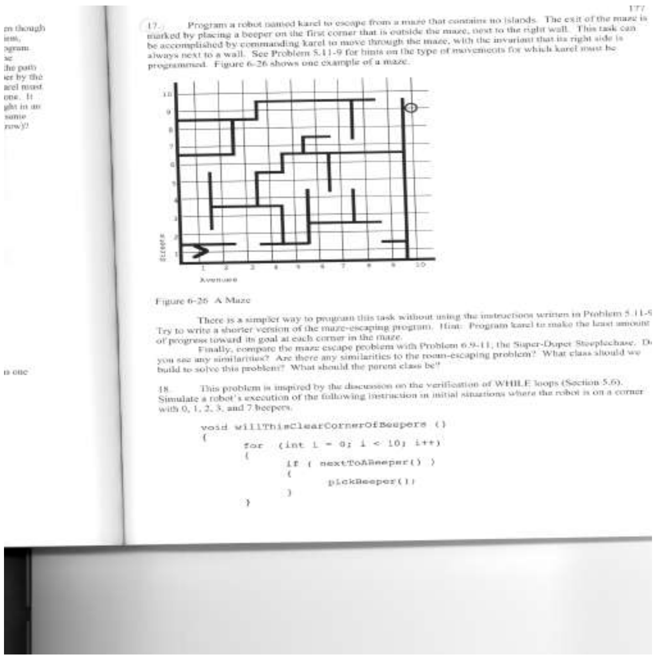
The *while* loop commands a robot to repeat instructions an **unknown** number of times (as long as the test condition remains true). The *while* command is executed somewhat similarly to an *if* command, except that the *while* loop repeatedly executes as long as the test condition is true versus the *if* command just executes only once if the test condition is true. Use the same boolean methods that you used in the previous chapter.

```
    public void getAllBeepers()
    {
        while (nextToABeeper())
        {
               pickBeeper();
        }
    }


    public static void main(String[] args)
    {
        SuperRobot karel=new SuperRobot(3, 4, North, 0);
        while (karel.frontIsClear())
        {
               karel.move();
               karel.pickBeeper();
        }
    }
```

Problem Set

1. Maze: Program the robot to escape from a maze that contains no islands. The exit of the maze is marked by a beeper that is placed on the first corner outside the maze, next to the wall. This task can be accomplished by moving through the maze, always following the wall to the robot's right.



177

17. Program a robot named karel to escape from a maze that contains no islands. The exit of the maze is marked by placing a beeper on the first corner that is outside the maze, next to the right wall. This task can be accomplished by commanding karel to move through the maze, with the invariant that its right side is always next to a wall. See Problem 5.11-9 for hints on the type of movements for which karel must be programmed. Figure 6-26 shows one example of a maze.

Figure 6-26 A Maze

There is a simpler way to program this task without using the instructions written in Problem 5.11-9. Try to write a shorter version of the maze-escaping program. Hint: Program karel to make the least amount of progress toward its goal at each corner in the maze.

Finally, compare the maze escape problem with Problem 6.9-11, the Super-Duper Steeplechase. Do you see any similarities? Are there any similarities to the room-escaping problem? What class should we build to solve this problem? What should the parent class be?

18. This problem is inspired by the discussion on the verification of WHILE loops (Section 5.6). Simulate a robot's execution of the following instruction in initial situations where the robot is on a corner with 0, 1, 2, 3, and 7 beepers.

```
void willThisClearCornerOfBeepers ()
{
        for  (int i = 0; i < 10; i++)
        {
                if ( nextToABeeper() )
                {
                        pickBeeper();
                }
        }
```

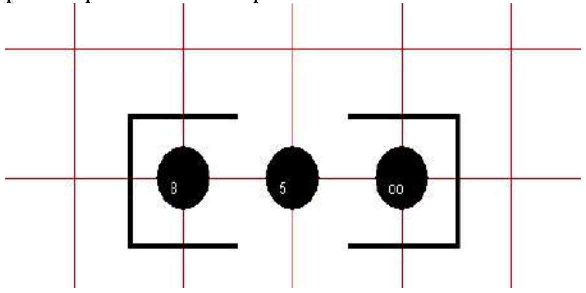
2. In the SuperRobot class, rewrite the faceNorth(), faceEast(), faceSouth(), and faceWest() methods using for each method, just one while loop with one included instruction. Test these methods using the TestFacing class from the last chapter, changing the comment markers to test the various directions.

   Write a pickupAllBeepers() method that picks up beepers from a corner until they are all picked up. Test this method by using the Test class and having a robot pickup all the beepers and then move off the corner so you can see.

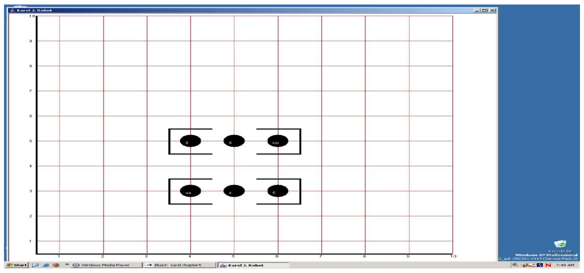
   Write a countBeepers() method that returns the number of beepers on the corner that the robot occupies. Remember that the robot might have beepers already in its bag before counting the beepers at the corner. Test these new methods using the Test class. Put a robot on a corner with some beepers down on the corner and also in the robot's bag. Move the robot off the corner to make sure that the same number of beepers remains at the corner as there was in the beginning. Do a System.out.println (see page 13) of the result from the countBeepers() method. Look at the Terminal Window to see your print out and to check that the robot ended up with the same number of beepers he started with. The countBeepers() method just counts the beepers; it does not keep them!

3. DangerousTask: There is a menace in Karel's world – an infinite pile of beepers. If Karel accidentally tries to pick up an infinite pile of beepers, it is forever doomed to pick beepers from the pile. Karel's current situation places the robot in grave danger from such a pile. The robot is standing outside two rooms: one is to the west and one is to the east. Only one of these rooms has a pile of beepers that Karel can pick. The other room has the dreaded infinite pile of beepers. Karel must decide which room is the safe room, enter it and pick all the beepers in that safe room. To help the robot decide which room is safe, there is a middle pile of beepers on the corner at which Karel is currently standing. If this middle pile has an even number of beepers (2,4,6,8,10), the safe room is the eastern room. If the pile has an odd number of beepers (1,3,5,7,9), the safe room is the western room. There is at least one beeper and no more than 10 beepers in the middle pile. At the end of the program, Karel should only have beepers in his bag that are from the safe room only, not from the middle pile and he should be back in the middle.

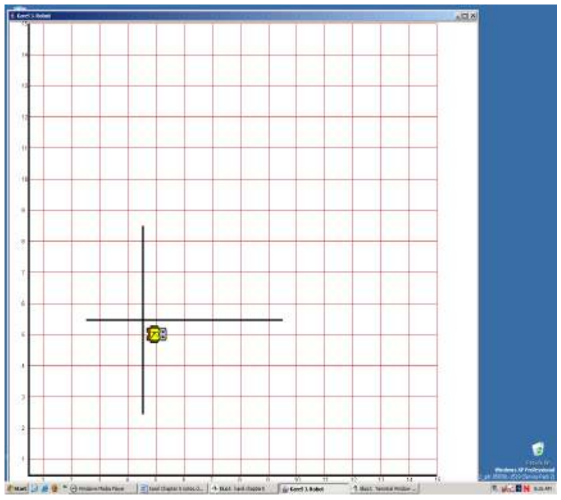
Since 5 is odd, robot needs to turn to the West, pick up those 8 beepers and return to the middle.

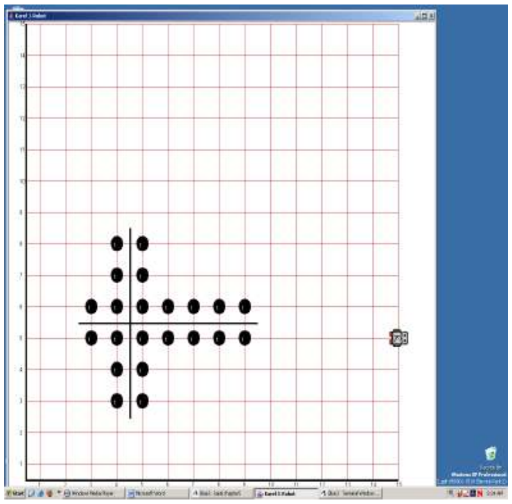


Since 4 is even, robot needs to turn to the East, pick up those 5 beepers and return to the middle.

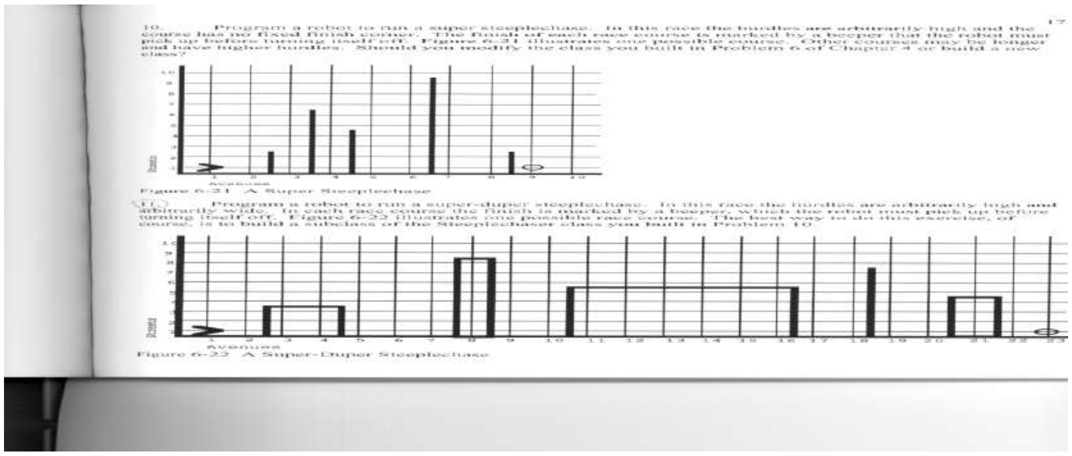


4. GardeningJob: Karel is working once again as a gardener. Karel must outline the wall segment shown with beepers. The walls can be any length, so don't program just for the diagram below. One and only one beeper is to be planted on each corner that is adjacent to a wall. Karel always starts in the same relative position however the walls have an arbitrary length. Karel starts with an infinite number of beepers. When done, move the robot out of the way.
Write a *for* loop that repeats 4 times. Inside that loop, write all the code to do one complete out and back wall. (Hint: Make the *for* loop only repeat once while working on the code for one out and back wall. When that is successful, put the *for* loop back to repeating 4 times.) The code inside the *for* loop must not be more than 11 lines (not counting squiggly braces).





5. SuperDuperSteeplechase: In this race, the hurdles are arbitrarily high and arbitrarily wide. In each race the finish is marked by a beeper; that is where the robot ends. The code for SuperDuperSteepleChase has the same *if-else* block as SteepleChase – just put it in a loop. You need to then modify the climbWall() method.



17

10. Program a robot to run a super steeplechase. In this race the hurdles are arbitrarily high and the course has no fixed finish corner. The finish of each race course is marked by a beeper that the robot must pick up before turning itself off. Figure 6-21 illustrates one possible course. Other courses may be longer and have higher hurdles. Should you modify the class you built in Problem 6 of Chapter 4 or build a new class?

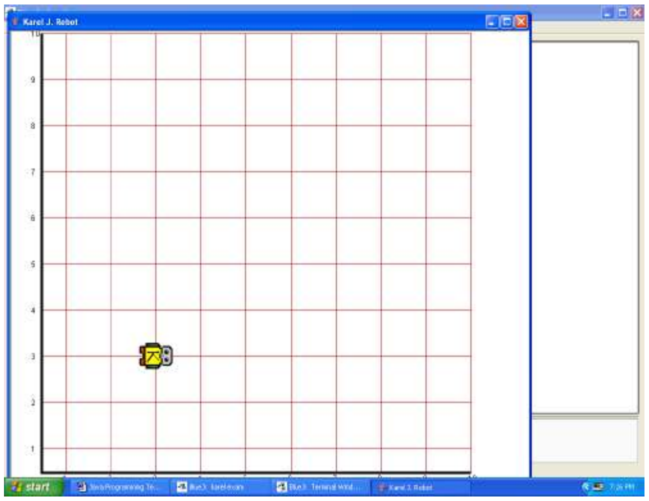
Figure 6-21 A Super Steeplechase

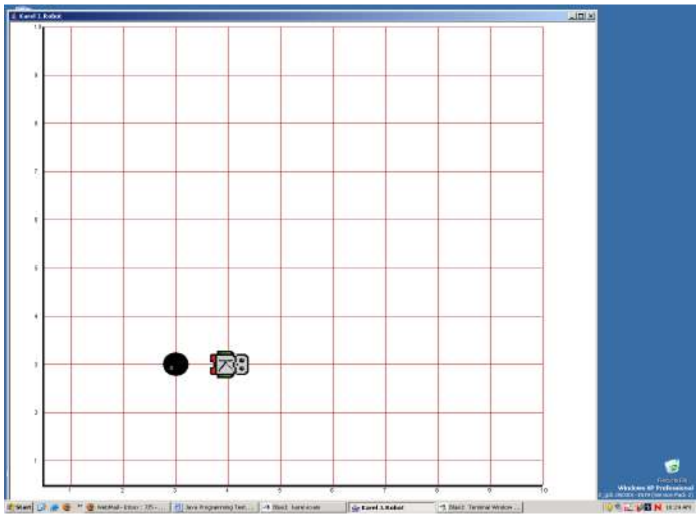
11. Program a robot to run a super-duper steeplechase. In this race the hurdles are arbitrarily high and arbitrarily wide. In each race course the finish is marked by a beeper, which the robot must pick up before turning itself off. Figure 6-22 illustrates one possible race course. The best way to do this exercise, of course, is to build a subclass of the Steeplechaser class you built in Problem 10.

Figure 6-22 A Super-Duper Steeplechase

6. Make a method in the SuperRobot class called countBag that counts how many beepers are in the robot's beeper bag. Test this new method using the Test class. Put a robot on a corner with some beepers down on the corner and also in the robot's bag. Move the robot off the corner to make sure that the same number of beepers remains at the corner as there was in the beginning. Do a System.out.println of the result from the countBag() method. Look at the Terminal Window to see your print out and to check that the robot ended up with the same number of beepers he started with.

   A robot is standing on a corner with some beepers in its beeper bag. Write the instructions in a DepositHalf class that will have the robot count the beepers that he's carrying and then deposit half of the beepers from the bag onto the corner (leaving the other half of the beepers in the beeper bag). Assume there is always an even number of beepers. Here is a run example in which the robot started with 8 beepers in its beeper bag (so 4 beepers remain in its bag and 4 beepers are on the corner).

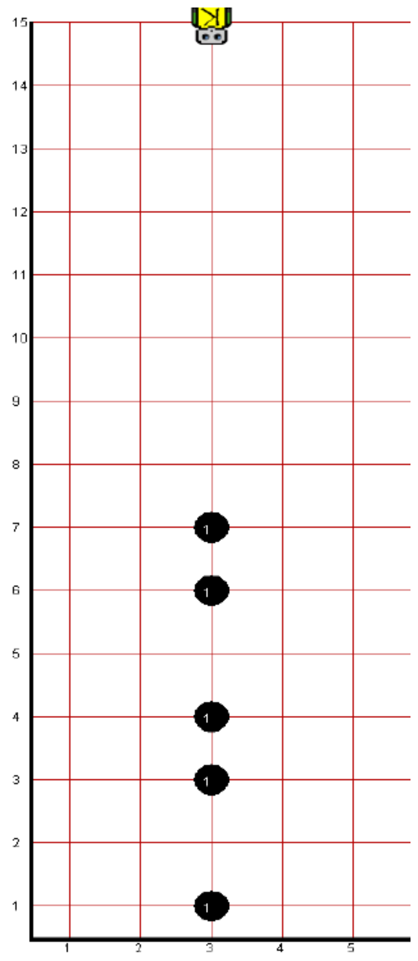


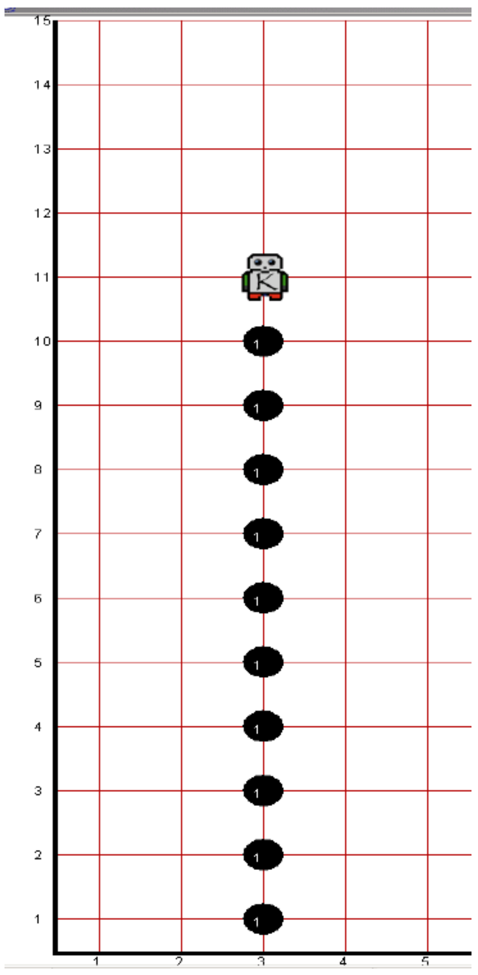


7. Karel has discovered that there are beepers that are scattered between him and the wall.
   - He starts 15 corners up, with some beepers (could be any number) in his bag
   - He wants to pick the beepers all up that are in front of him (could be any number there)
   - There aren't any beepers at the corner that he starts at
   - There isn't more than 1 beeper at any of the corners
   - Print a message that tells how many beepers he has in his bag once he reaches the wall
   - He then needs to put them back down so they are next to each other one per corner

Write a class named OrganizeBeepers that does the above.

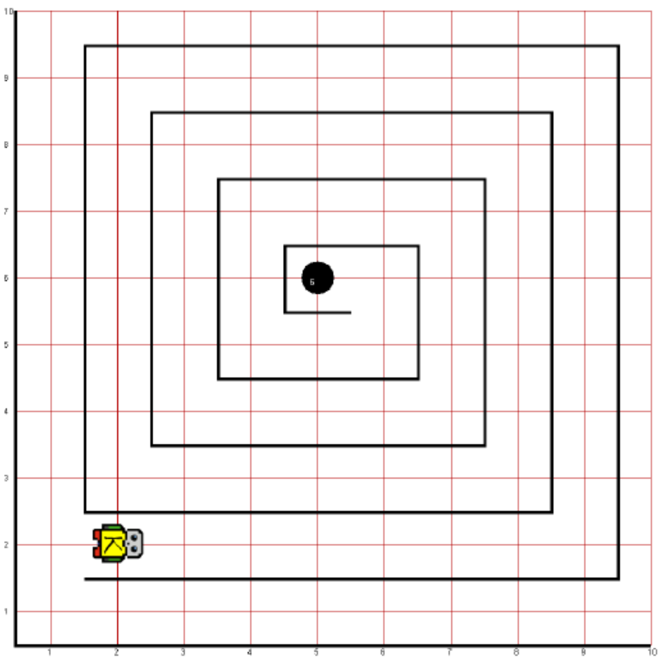
In this example, the robot started with 5 beepers in his bag and picked up 5 more as he walked to the wall so that he had 10 beepers to put down on his way back

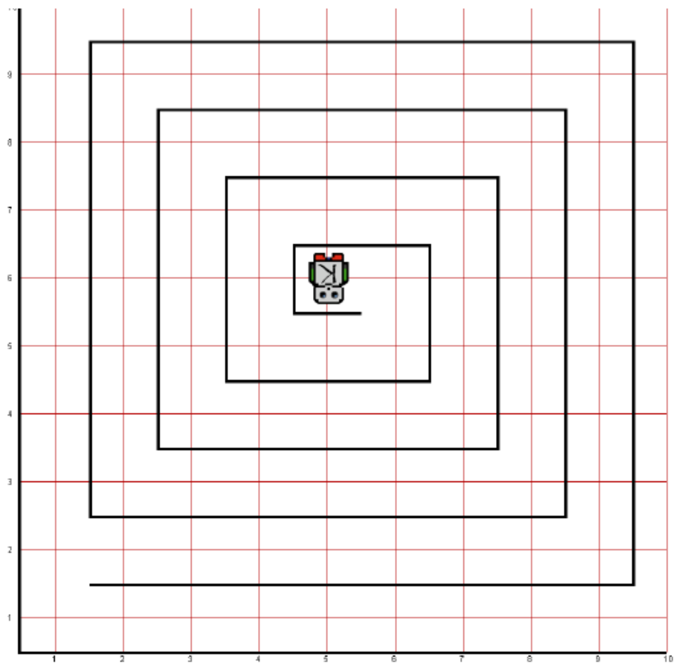




Number of beepers in bag at wall is 10

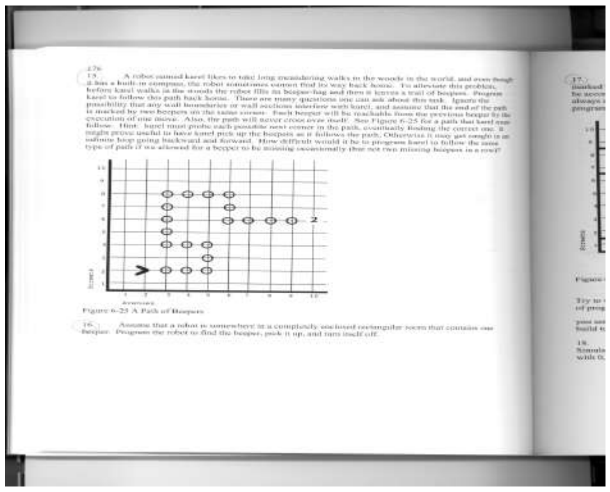
8. Karel is on a treasure hunt. He needs to follow the spiral inwards until he finds the treasure. When he finds the treasure, he is to count the beepers that he found and print out a message. He always starts at 2,2 but doesn't have any idea how long the walls are. Name this class Spiral. To get your robot to the center, it takes no more than 4 lines of code (not including squiggly braces). You may not use the followWallRight() method.





---BONUS---

PathOfBeepers: Karel robot likes to take long meandering walks in the woods in the world. Even though he has a built-in compass, the robot sometimes cannot find its way back home. To alleviate this problem, Karel leaves a trail of beepers. Program Karel to follow this path back home. Ignore the possibility that any wall boundaries or wall sections interfere with Karel and assume that the end of the path is marked by two beepers on the same corner. Each beeper is reachable from the previous beeper by the execution of one move. Also the path will never cross over itself. Hint: Probe each possible next corner in the path, eventually finding the correct one. It would be useful to have Karel pick up the beepers as it follows the path; otherwise it may get caught in an infinite loop.



176

15. A robot named karel likes to take long meandering walks in the woods in the world, and even though it has a built-in compass, the robot sometimes cannot find its way back home. To alleviate this problem, before karel walks in the woods the robot fills its beeper-bag and then it leaves a trail of beepers. Program karel to follow this path back home. There are many questions one can ask about this task. Ignore the possibility that any wall boundaries or wall sections interfere with karel, and assume that the end of the path is marked by two beepers on the same corner. Each beeper will be reachable from the previous beeper by the execution of one move. Also, the path will never cross over itself. See Figure 6-25 for a path that karel can follow. Hint: karel must probe each possible next corner in the path, eventually finding the correct one. It might prove useful to have karel pick up the beepers as it follows the path. Otherwise it may get caught in an infinite loop going backward and forward. How difficult would it be to program karel to follow the same type of path if we allowed for a beeper to be missing occasionally (but not two missing beepers in a row)?

Figure 6-25 A Path of Beepers

16. Assume that a robot is somewhere in a completely enclosed rectangular room that contains one beeper. Program the robot to find the beeper, pick it up, and turn itself off.