CS 106A, Lecture 2
Programming with Karel

reading:

Karel the Robot Learns Java, Chapters 2-3

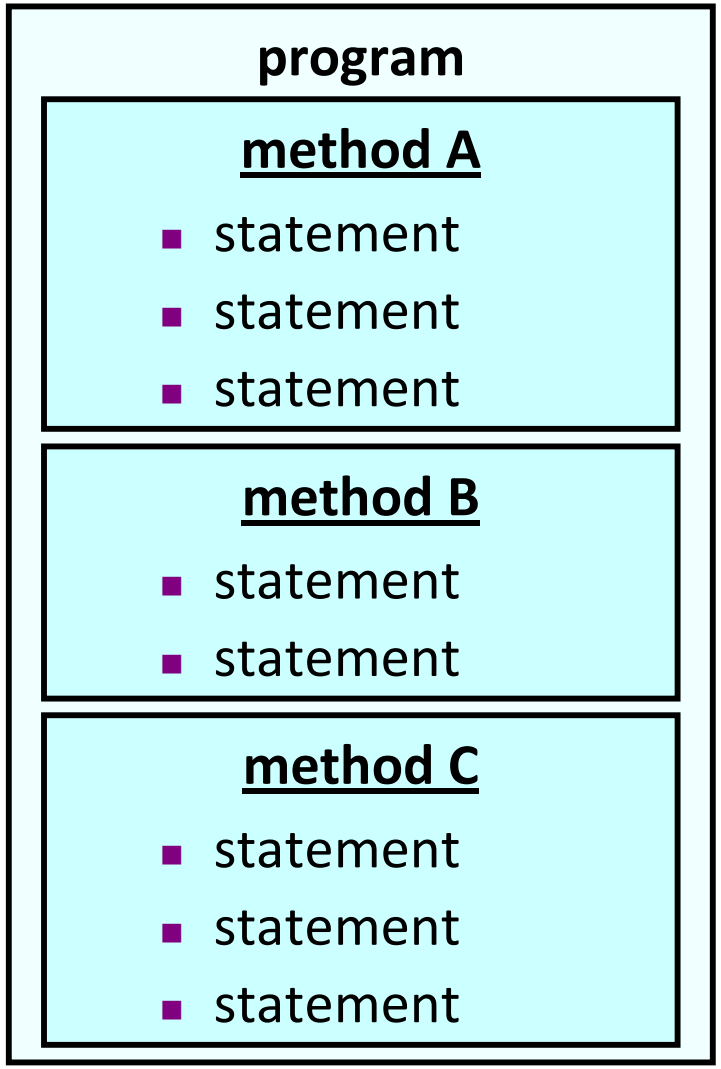
Program Decomposition with Methods

Methods

- **method**: A named group of statements.
  - denotes the *structure* of a program
  - eliminates *redundancy* by code reuse

  – **program decomposition**:
  dividing a problem into methods

- Writing a method is like adding a new command to Karel/Java.



1) Declaring a method

Gives your method a name so it can be executed

- Syntax:

```
public void name() {
    statement;
    statement;
    ...
    statement;
}
```

- Example:

```
public void moveThreeTimes() {
    move();
    move();
    move();
}
```

2) Calling a method

Executes the method's code

- Syntax:

```
name();
```

  – You can call the same method multiple times if you like.

- Example:

```
public void run() {
    moveThreeTimes();
    turnLeft();
    move();
}
```

Hey, I just wrote it
and this is crazy
but here's my method
so call it maybe?

Control flow

- When a method is called, the program's execution...
  - "jumps" into that method, executing its statements, then
  - "jumps" back to the point where the method was called.

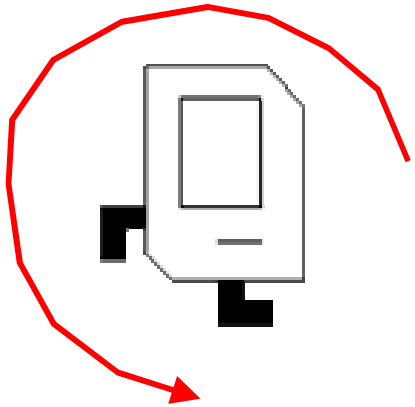
```
public class MethodsExample extends Karel {
    public void run() {
        moveThreeTimes();
        turnLeft();
        moveThreeTimes();
        move();
    }
}
```

```
public void moveThreeTimes() {
    move();
    move();
    move();
```

```
public void moveThreeTimes() {
    move();
    move();
    move();
}
```

Method exercises

- Write a Karel method **`turnRight`** that performs the equivalent of a right turn by turning left 3 times.



- Write a method **`woopWoop`** that spins Karel in a circle by having him turn left 4 times.

- Write a method **`cycle`** that moves Karel in a square by having him move-and-turn-left 4 times in a row.

- Write a method **`advanceBeeper`** that grabs a beeper from Karel's current location, moves it forward by 1 square, and puts it down.

Exercise solutions

```
public void turnRight() {
    turnLeft();
    turnLeft();
    turnLeft();
}

public void woopWoop() {
    turnLeft();
    turnLeft();
    turnLeft();
    turnLeft();
}

public void advanceBeeper() {
    pickBeeper();
    move();
    putBeeper();
}
```

```
public void cycle() {
    move();
    turnLeft();
    move();
    turnLeft();
    move();
    turnLeft();
    move();
    turnLeft();
}
```

The SuperKarel class

- Your program can extend **`SuperKarel`** if you like.
  - `SuperKarel` is the same as `Karel` but already includes built-in methods named **`turnRight`** and **`turnAround`**.

```
import stanford.karel.*;

public class DubstepKarel extends SuperKarel {
    public void run() {
        move();
        turnRight();   // 3 left turns
        move();
        turnAround();  // 2 left turns
    }
}
```

Methods calling methods

- One method can call another:

```
public void woopWoop() {
    turnRight(); // 3 lefts
    turnLeft();
}

public void cycle() {
    moveAndTurn();
    moveAndTurn();
    moveAndTurn();
    moveAndTurn();
}
```

```
public void moveAndTurn() {
    move();
    turnLeft();
}
```

When to use methods

- Place statements into a method if:
  - The statements are related structurally, and/or
  - The statements are repeated.

- You should *not* create a method for:
  - An individual statement.
  - Unrelated or weakly related statements.
    (Consider splitting them into two smaller methods.)

Java's Control Statements

Redundant repetition

- So far, repeating a command many times is redundant:

```
// move Karel forward 7 times
move();
move();
move();
move();
move();
move();
move();
```

- Creating a method doesn't help very much. (Why not?)
  - We really want a concise way to say, "Move 7 times."

The for loop

```
for (int i = 0; i < max; i++) {
    statement;
    statement;
    ...
    statement;
}
```

– Repeats the given statements ***max*** times.

```
// move Karel forward 7 times
for (int i = 0; i < 7; i++) {
    move();
}
```

– Exercise: Update our previous methods to use `for` loops as needed.

Example for loops

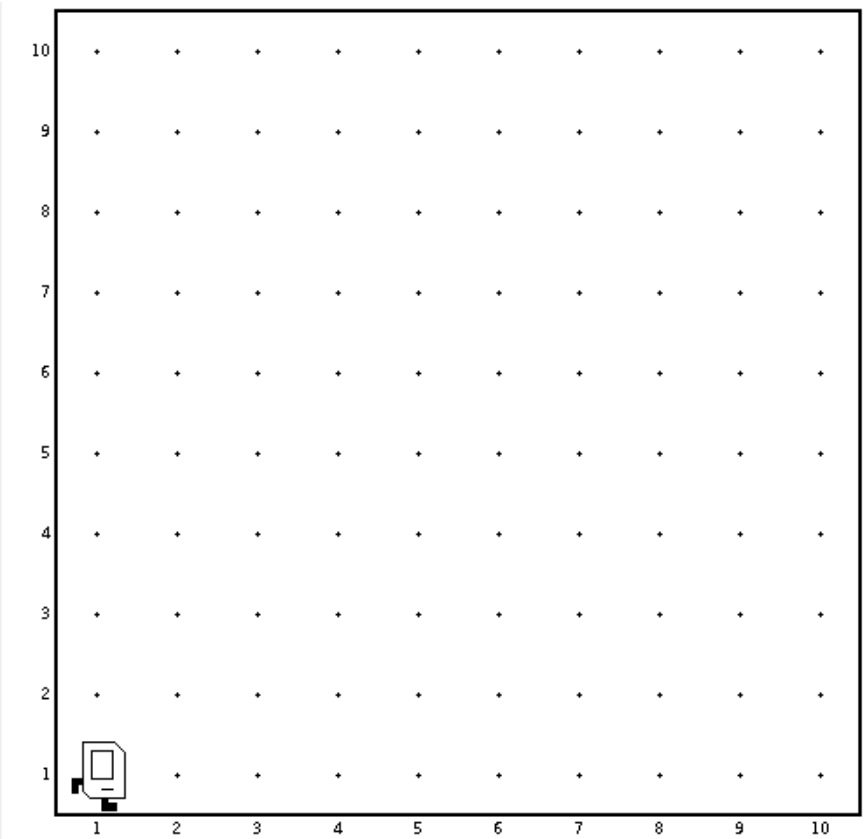
```
// Turns right by turning left 3 times.
public void turnRight() {
    for (int i = 0; i < 3; i++) {
        turnLeft();
    }
}

// Moves in a square cycle pattern.
public void cycle() {
    for (int i = 0; i < 4; i++) {
        move();
        turnLeft();
    }
}
```

Exercise

karelLoop1

- **Q:** What row/column will Karel be at after the following code?

```
for (int i = 0; i < 5; i++) {
    move();
}
for (int i = 0; i < 2; i++) {
    turnLeft();
    move();
}
```



A. (5, 3)

B. (6, 2)

C. (5, 2)

D. (5, 1)

E. none of the above

Logic errors

- **logic error**: Incorrect program behavior.
  - Example: Calling `pickBeeper` when there is no beeper at Karel's current location
  - Usually caused due to programmer error.
  - How can we avoid logic errors in our code?

Karel condition methods

- Karel has some commands that are not meant to be complete statements, but rather are used to ask questions:

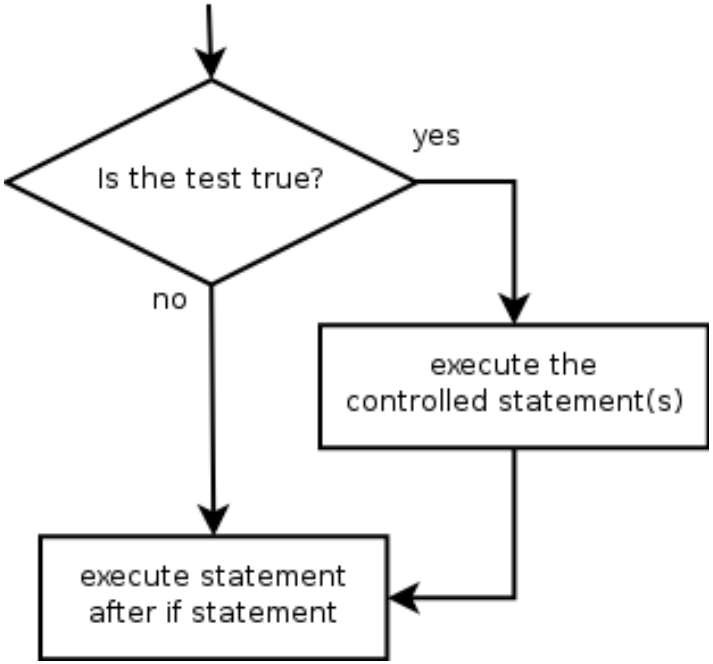
Test	*Opposite*	*What it checks*
`frontIsClear()`	`frontIsBlocked()`	Is there a wall in front of Karel?
`leftIsClear()`	`leftIsBlocked()`	Is there a wall to Karel's left?
`rightIsClear()`	`rightIsBlocked()`	Is there a wall to Karel's right?
`beepersPresent()`	`noBeepersPresent()`	Are there beepers on this corner?
`beepersInBag()`	`noBeepersInBag()`	Any there beepers in Karel's bag?
`facingNorth()`	`notFacingNorth()`	Is Karel facing north?
`facingEast()`	`notFacingEast()`	Is Karel facing east?
`facingSouth()`	`notFacingSouth()`	Is Karel facing south?
`facingWest()`	`notFacingWest()`	Is Karel facing west?

*This is **Table 1** on page 18 of the Karel course reader.*

The if statement

Executes a group of statements only if a test is true

```
if (test) {
    statements;
}
```



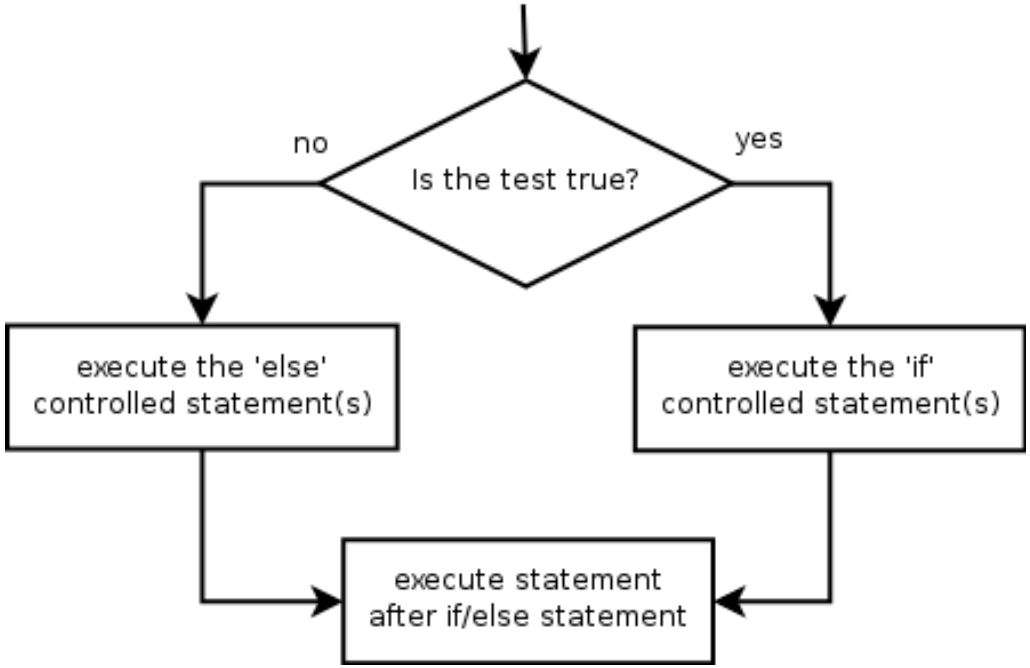
- Example:

```
turnLeft();
if (frontIsClear()) {
    move();
}
turnLeft();
```

The if/else statement

Executes one group if a test is true, another if false

```
if (test) {
    statements;
} else {
    statements;
}
```



- Example:

```
if (frontIsClear()) {
    move();
} else {
    turnLeft();
    turnLeft();
}
```

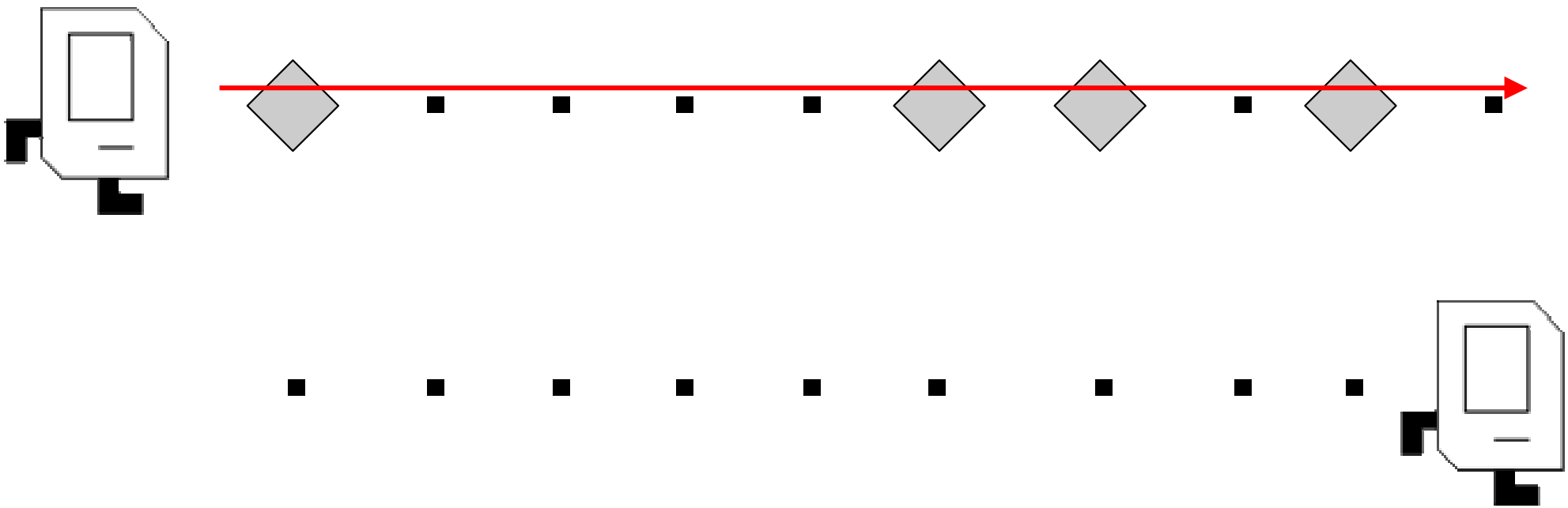
Nested statements

- You can combine any statement inside any other.
  - `if` inside `if`, `if` inside `for`, …

```
// try to move forward and take a beeper
if (frontIsClear()) {
    move();
    if (beepersPresent()) {
        pickBeeper();
    } else {
        for (int i = 0; i < 3; i++) {
            turnLeft();
        }
    }
}
```

If/else exercise

- Write a method called `sweep` that causes Karel to walk forward 10 squares and pick up any beepers he finds along the way.
  - Don't try to pick up a beeper if there isn't one on the current square.

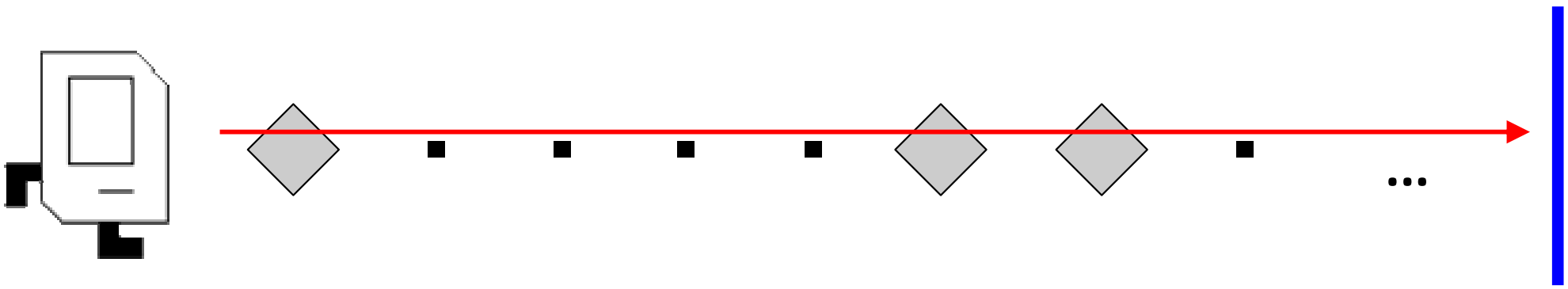


If/else solution

```
public void sweep() {
    for (int i = 0; i < 10; i++) {
        move();
        if (beepersPresent()) {
            pickBeeper();
        }
    }
}
```

Problem

- Suppose we want our sweep method to walk Karel all the way to the edge of the world (or the nearest wall), regardless of the world's size.
  - What should we set our `for` loop's ***max*** to?
  - Is a `for` loop really the right tool for solving this problem?



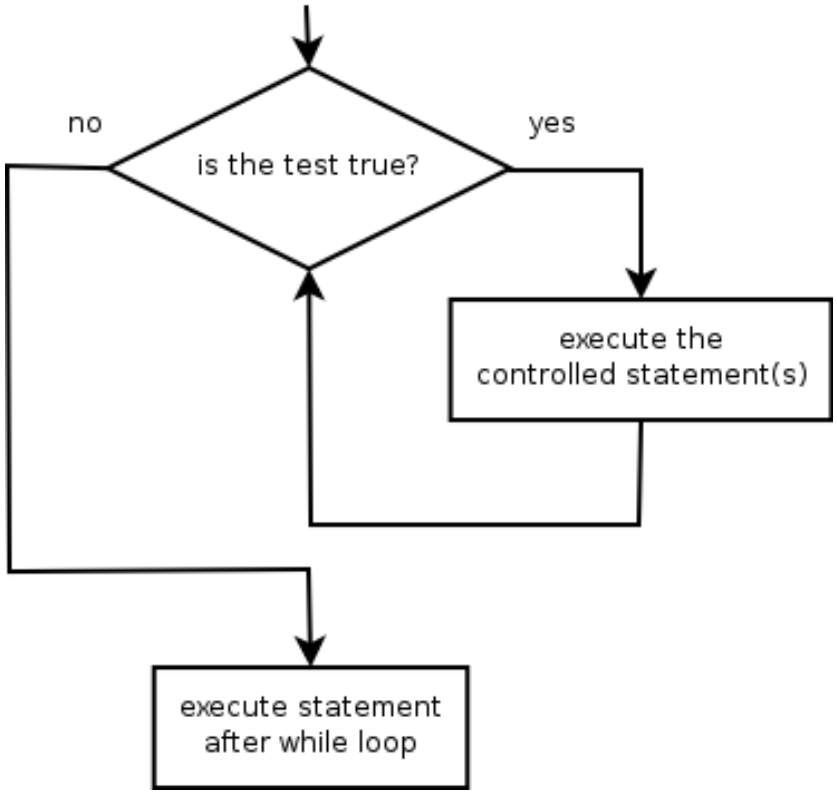
The while loop

Repeatedly executes its body as long as a logical test is true

```
while (test) {
    statements;
}
```

- Example:

```
// walk until I hit a wall
while (frontIsClear()) {
    move();
}
```



Sweep solution 2

```
public void sweep() {
    while (frontIsClear()) {
        move();
        if (beepersPresent()) {
            pickBeeper();
        }
    }
}
```

More practice

- Write a program named **RoadRepairKarel** that looks for "potholes" in the road ahead of him and fills in each with a single beeper.
  - Assume Karel starts at (2, 1) facing East and holds infinite beepers.
  - Walk all the way to the East edge of the world, filling any potholes.
  - *See Karel book Ch. 3 for a discussion of how to solve this problem.*