

# CS 106A, Lecture 2

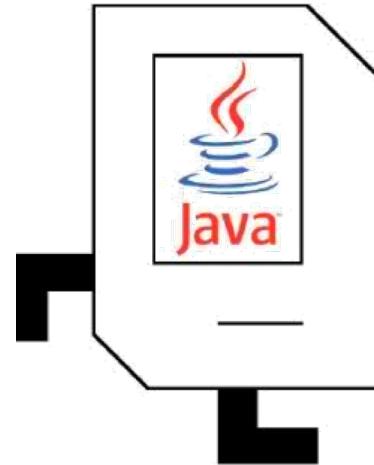
## Programming with Karel

suggested reading:

*Karel, Ch. 3-4*

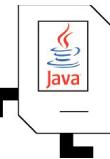
# Plan For Today

- (Re)Meet Karel the Robot
- Announcements
- Control Flow
  - For loops
  - While loops
  - If/else statements



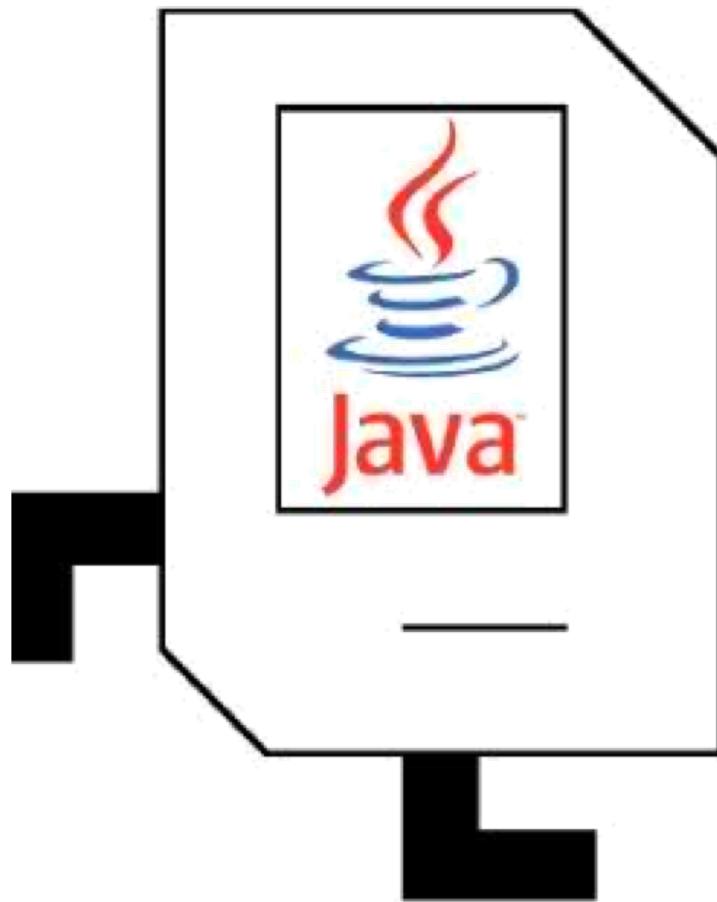
# Plan For Today

- (Re)Meet Karel the Robot

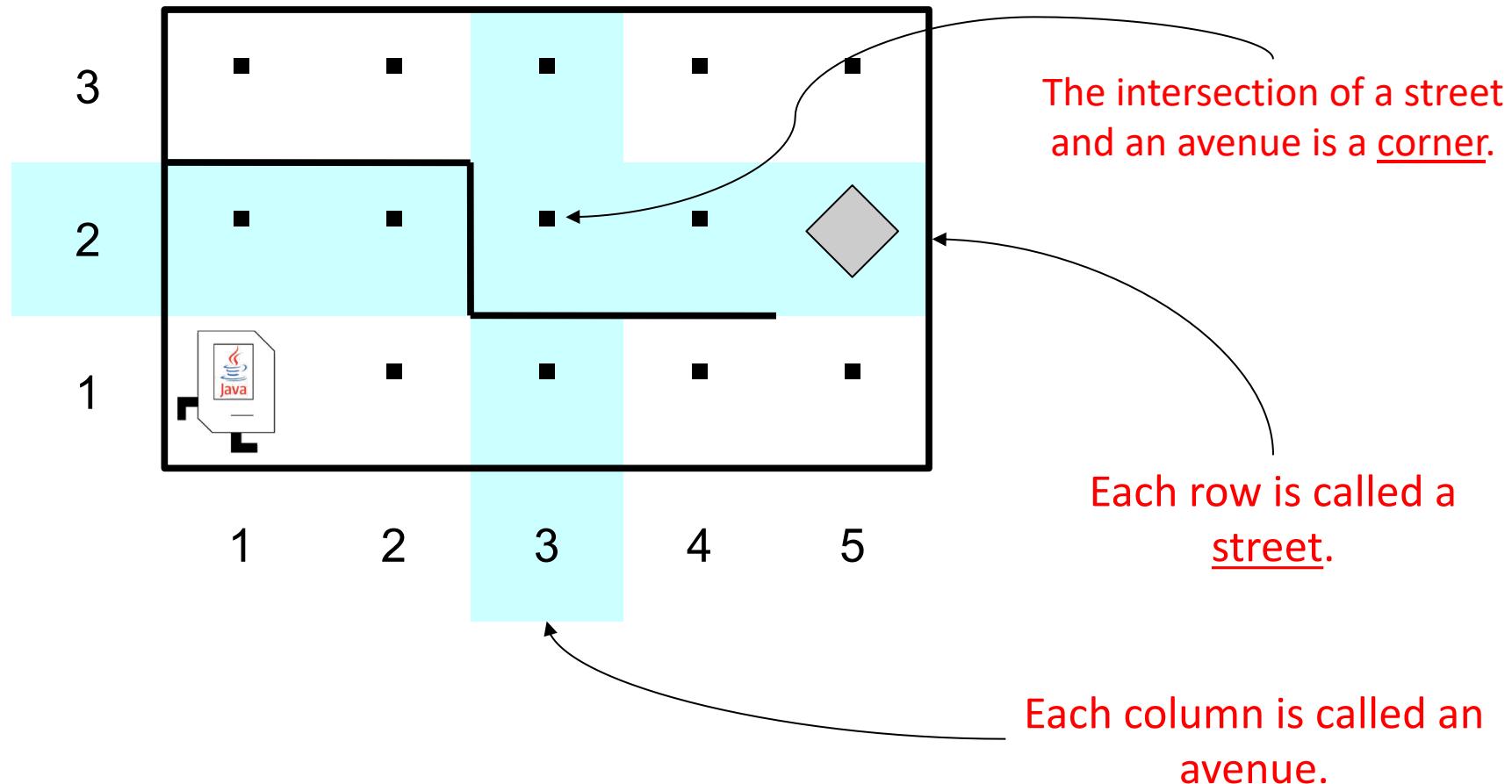


- Announcements
- Control Flow
  - For loops
  - While loops
  - If/else statements

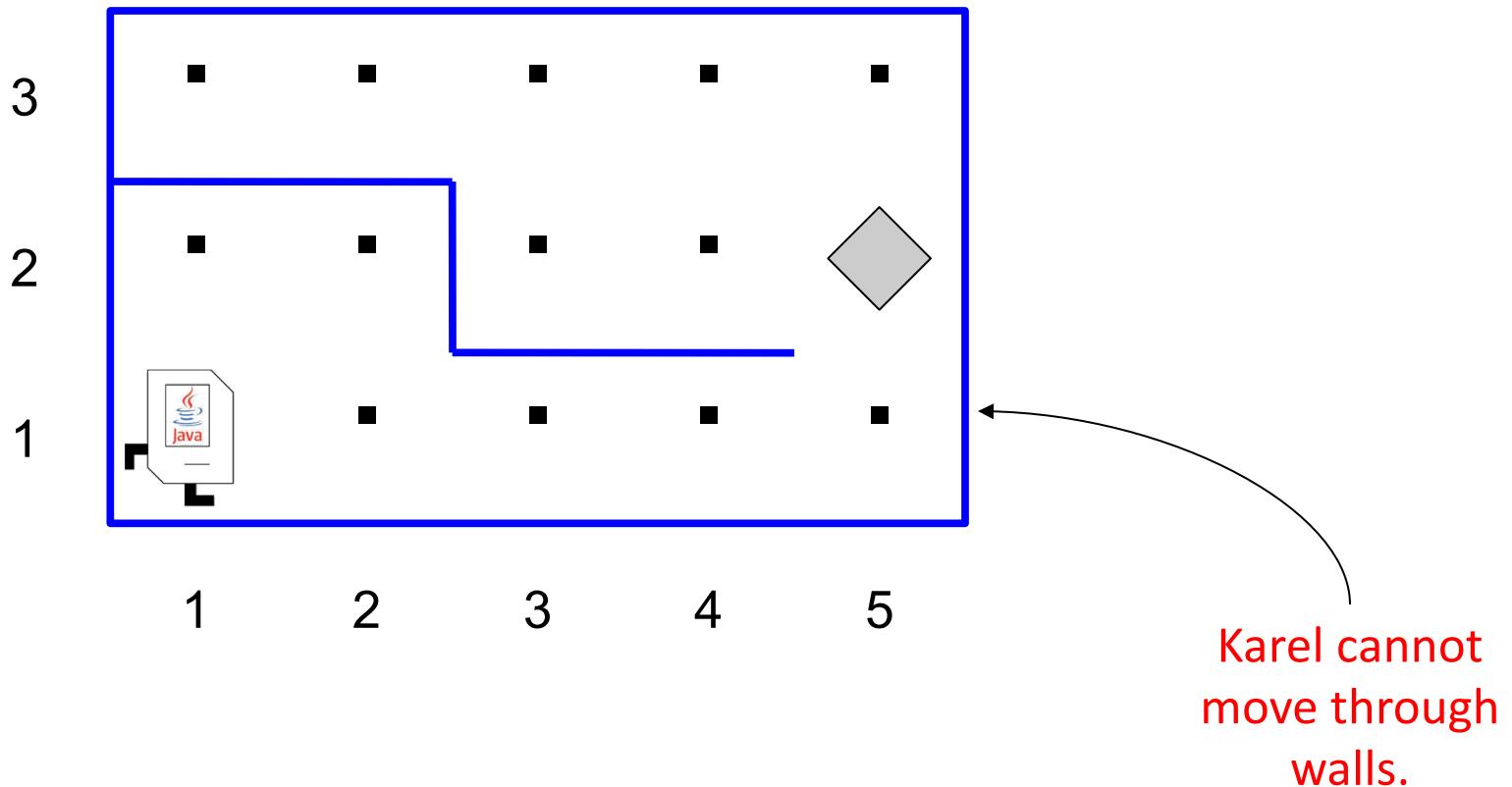
# Meet Karel the Robot!



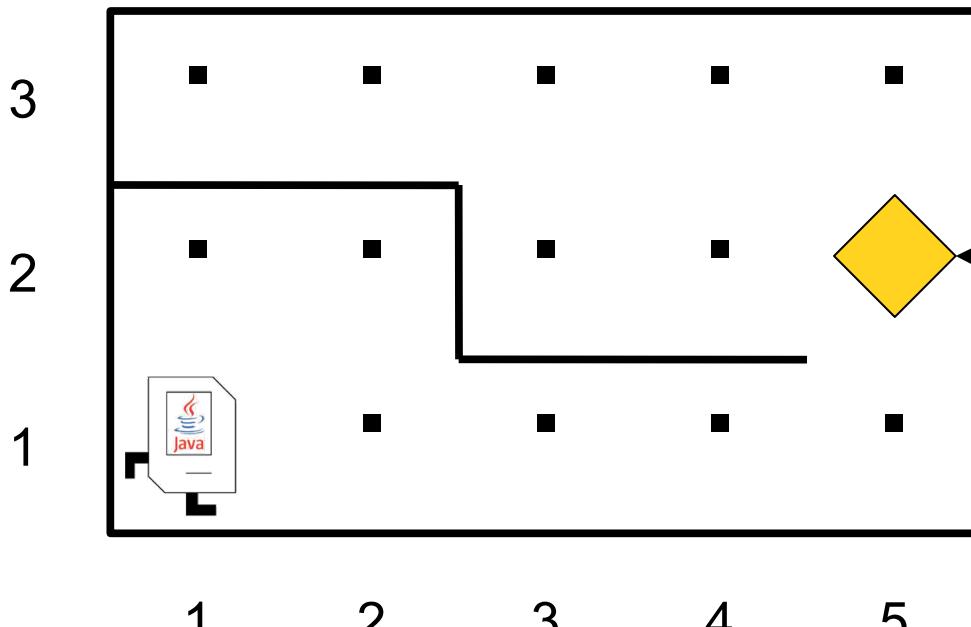
# Karel's World



# Walls



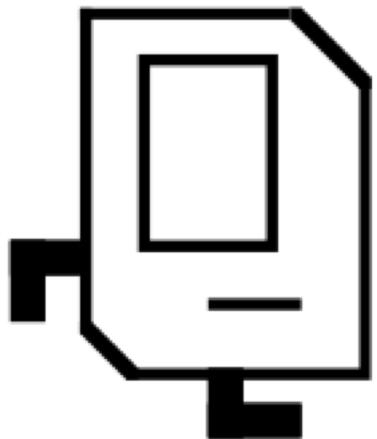
# Beepers



Beepers mark locations in Karel's world. Karel can pick them up and put them down.

# Karel Knows 4 Commands

move

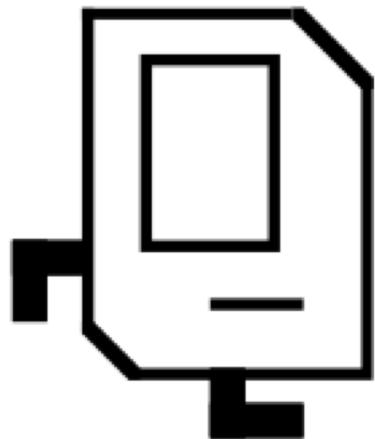


turnLeft

putBeeper

pickBeeper

# Karel Knows 4 Commands



move( )

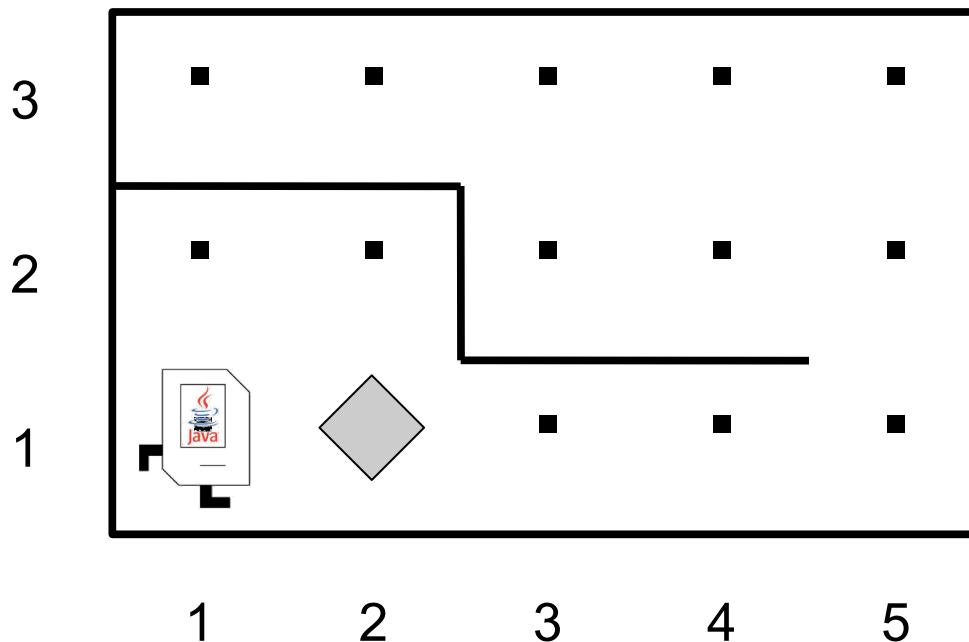
turnLeft( )

putBeeper( )

pickBeeper( )

“methods”

# Commands: move



Karel Commands

**move**

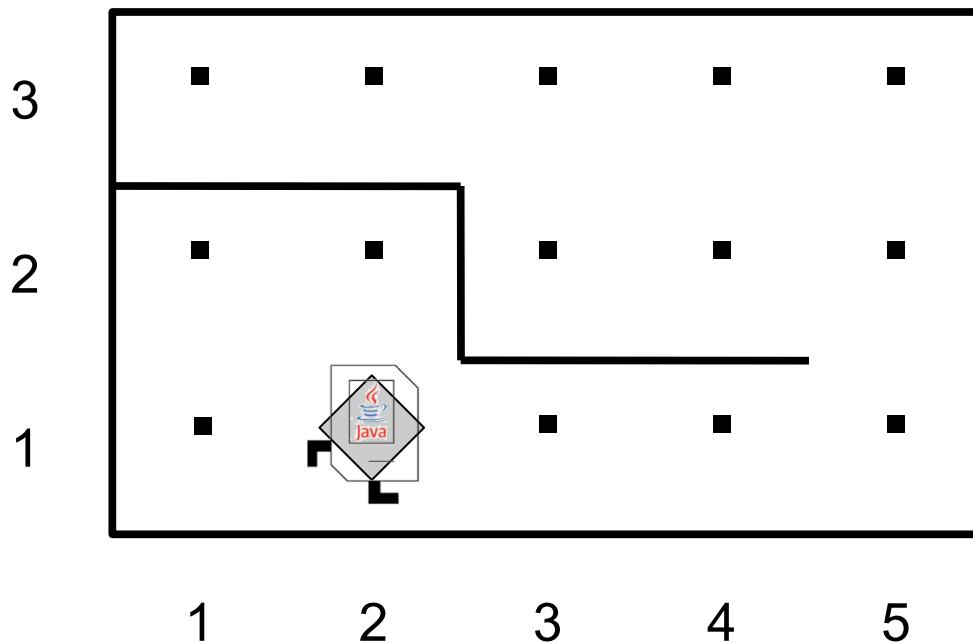
**turnLeft**

**pickBeeper**

**putBeeper**

- move makes Karel move forward one square in the direction it is facing.

# Commands: move



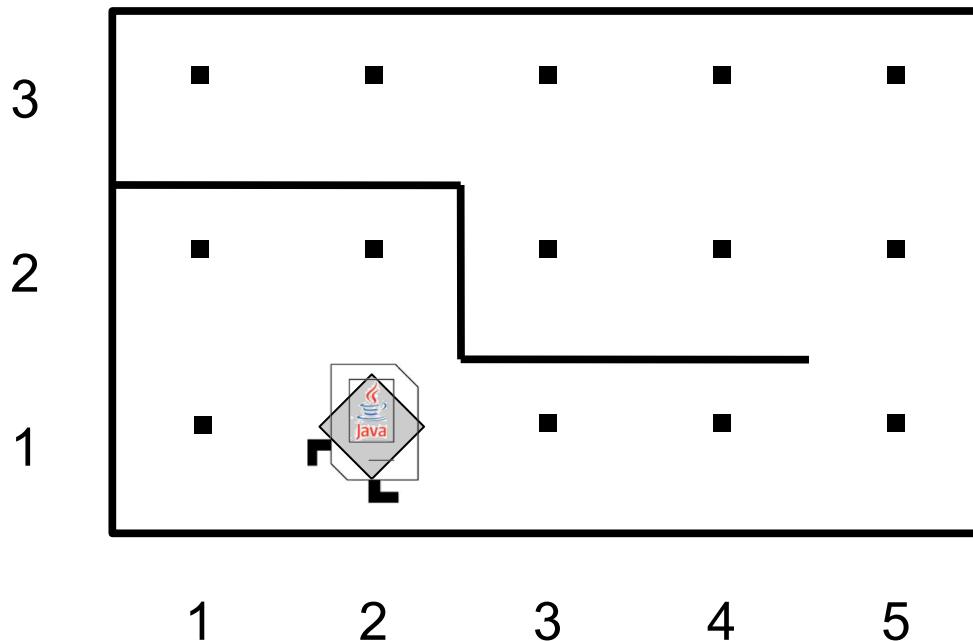
Karel Commands

---

**move**  
**turnLeft**  
**pickBeeper**  
**putBeeper**

- move makes Karel move forward one square in the direction it is facing.

# Commands: turnLeft



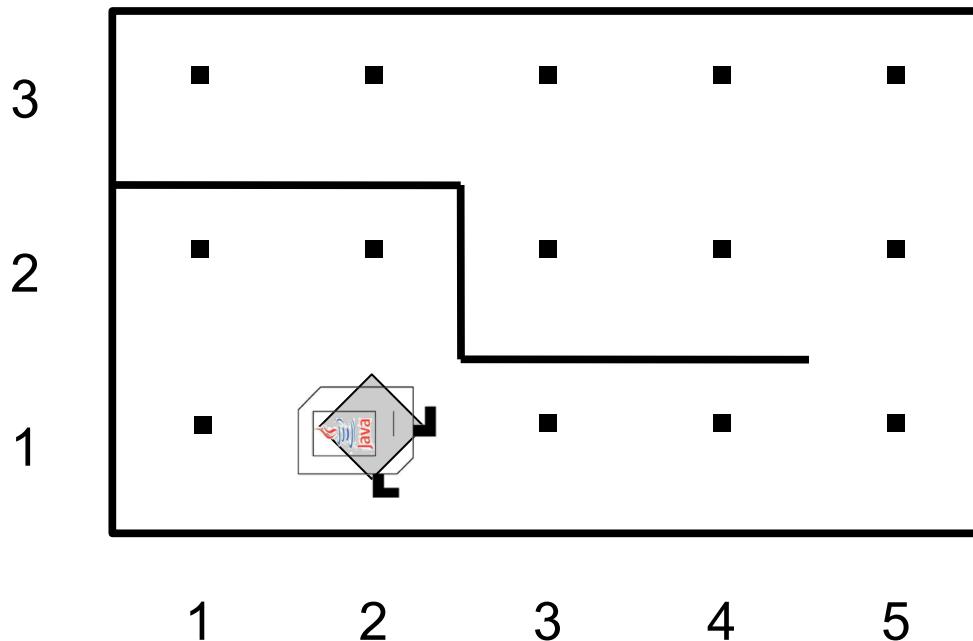
Karel Commands

---

move  
**turnLeft**  
pickBeeper  
putBeeper

- turnLeft makes Karel rotate 90° counter-clockwise.

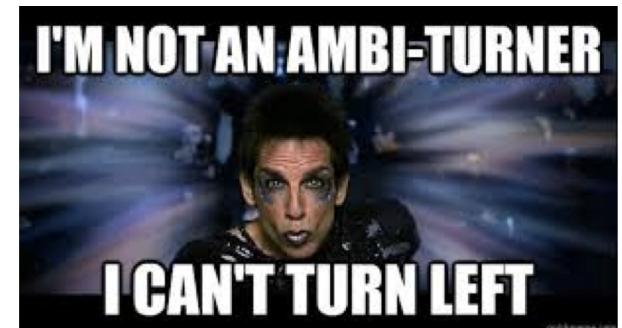
# Commands: turnLeft



Karel Commands

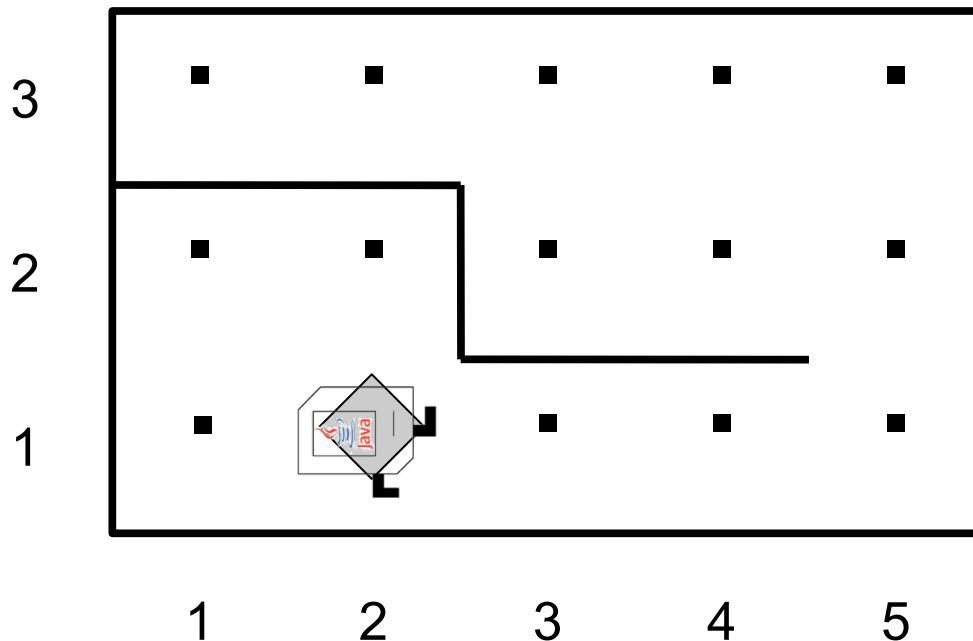
---

move  
**turnLeft**  
pickBeeper  
putBeeper



- turnLeft makes Karel rotate 90° counter-clockwise.
- There is no turnRight command. (Q: Why not?)

# Commands: turnLeft



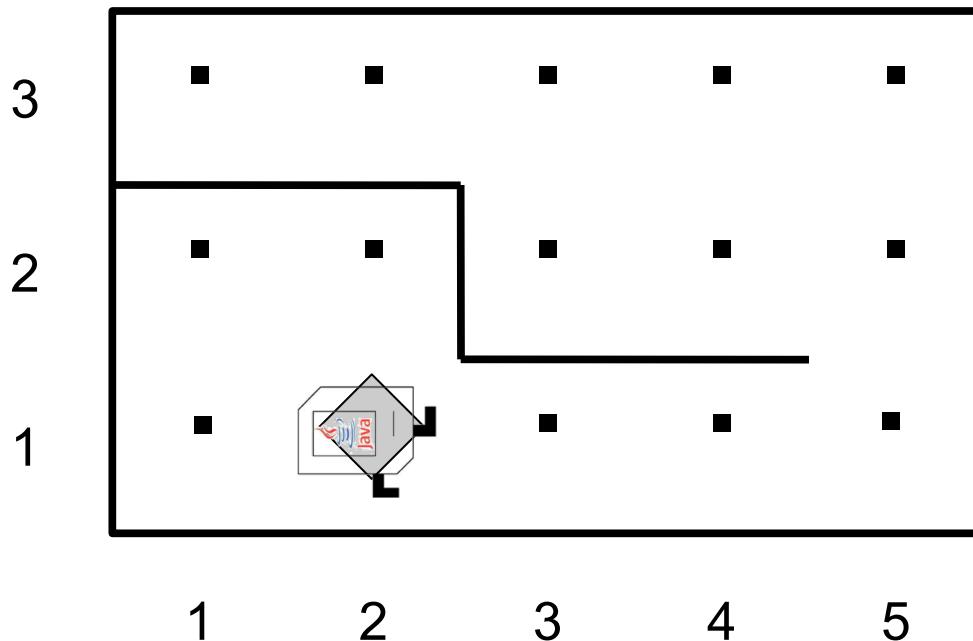
Karel Commands

---

move  
**turnLeft**  
pickBeeper  
putBeeper

- turnLeft makes Karel rotate 90° counter-clockwise.
- There is no turnRight command. (Q: Why not? A: No need!)

# Commands: pickBeeper



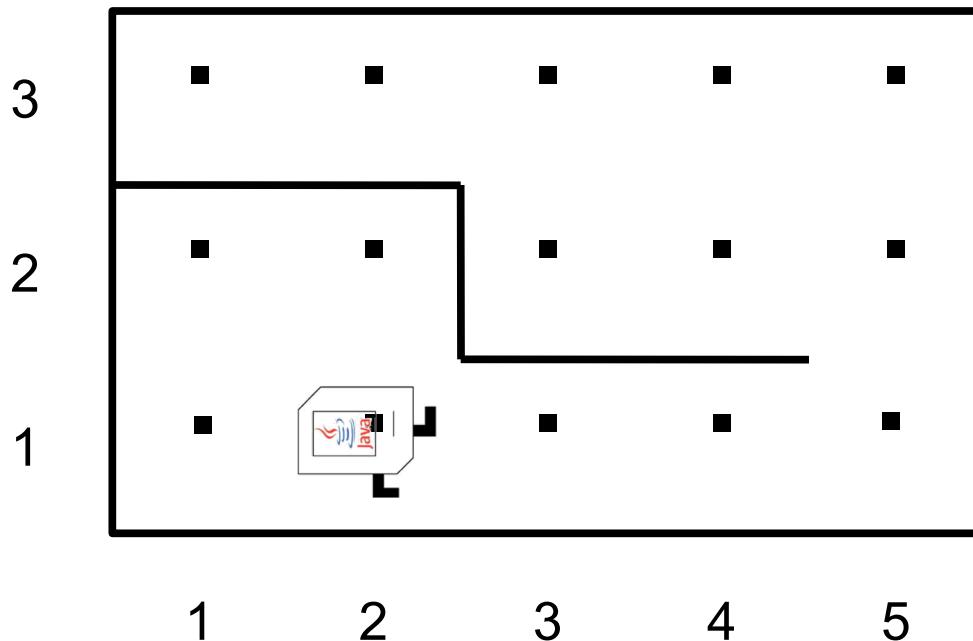
Karel Commands

---

move  
turnLeft  
**pickBeeper**  
putBeeper

- **pickBeeper** makes Karel pick up the beeper at the current corner. Karel can hold multiple beepers at a time in its "beeper bag".

# Commands: pickBeeper



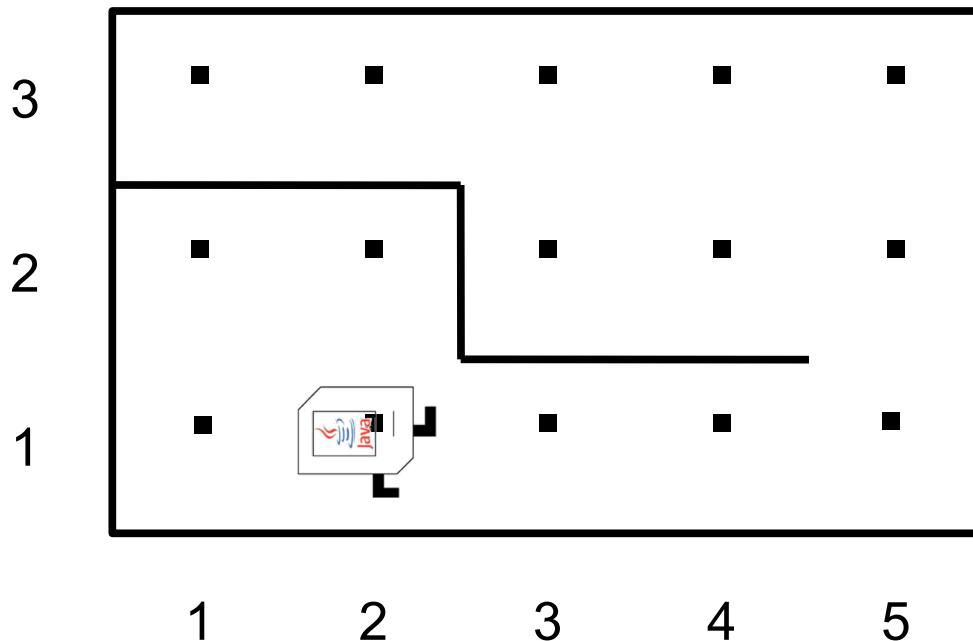
Karel Commands

---

move  
turnLeft  
**pickBeeper**  
putBeeper

- **pickBeeper** makes Karel pick up the beeper at the current corner. Karel can hold multiple beepers at a time in its "beeper bag".

# Commands: putBeeper



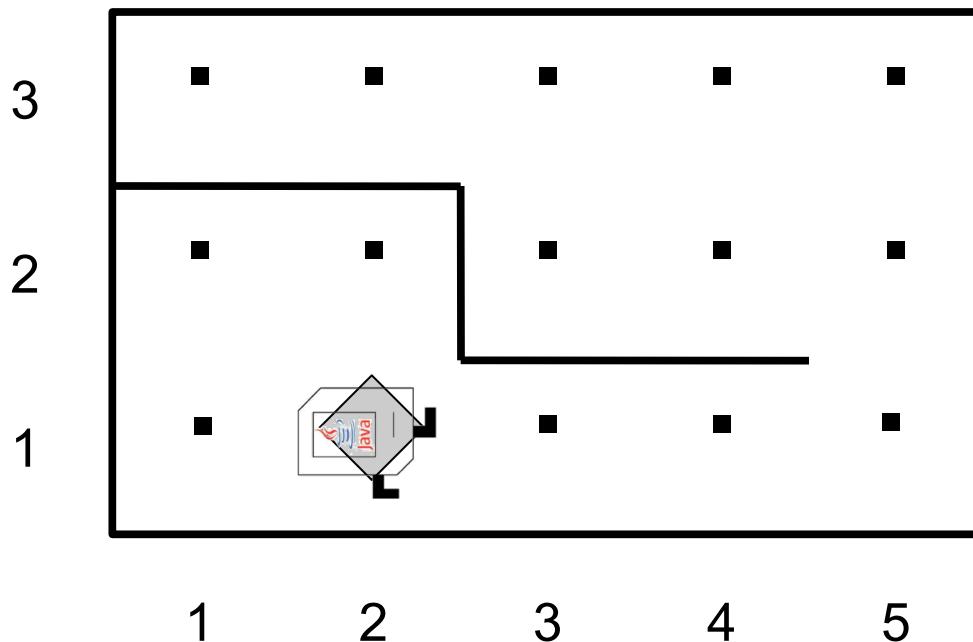
Karel Commands

---

move  
turnLeft  
pickBeeper  
**putBeeper**

- putBeeper makes Karel put a beeper down at its current location.
  - pickBeeper and putBeeper are used to move beepers around.

# Commands: putBeeper



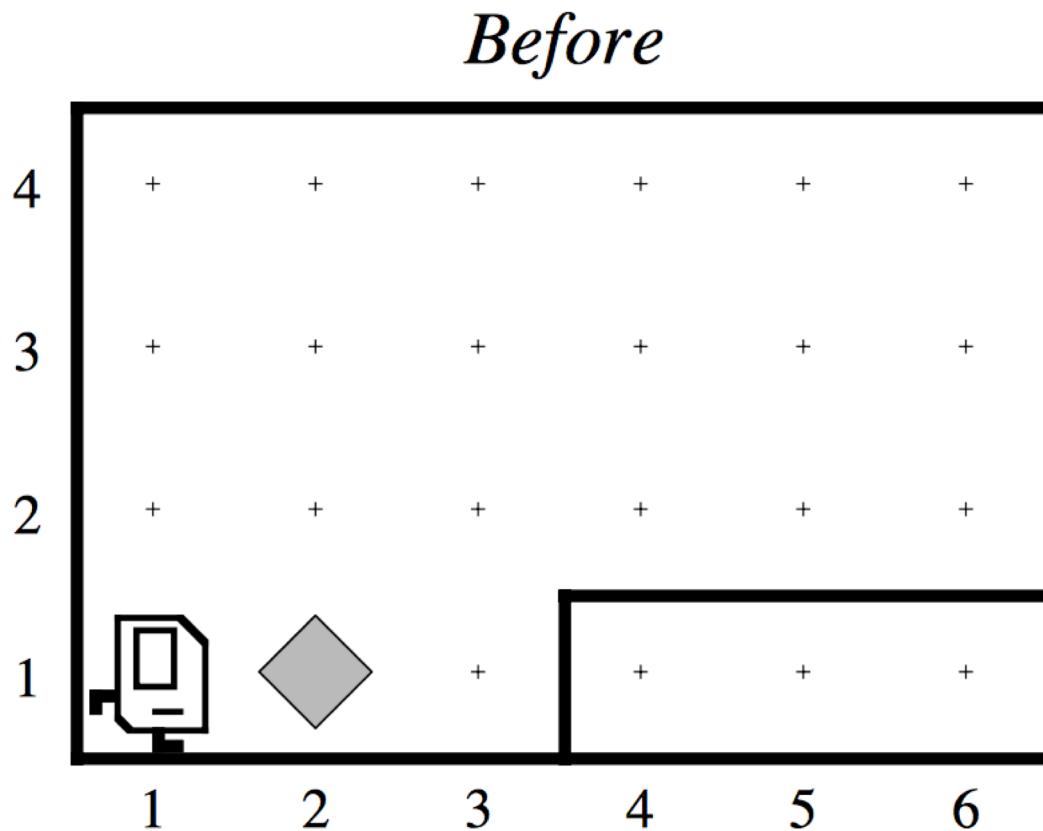
Karel Commands

---

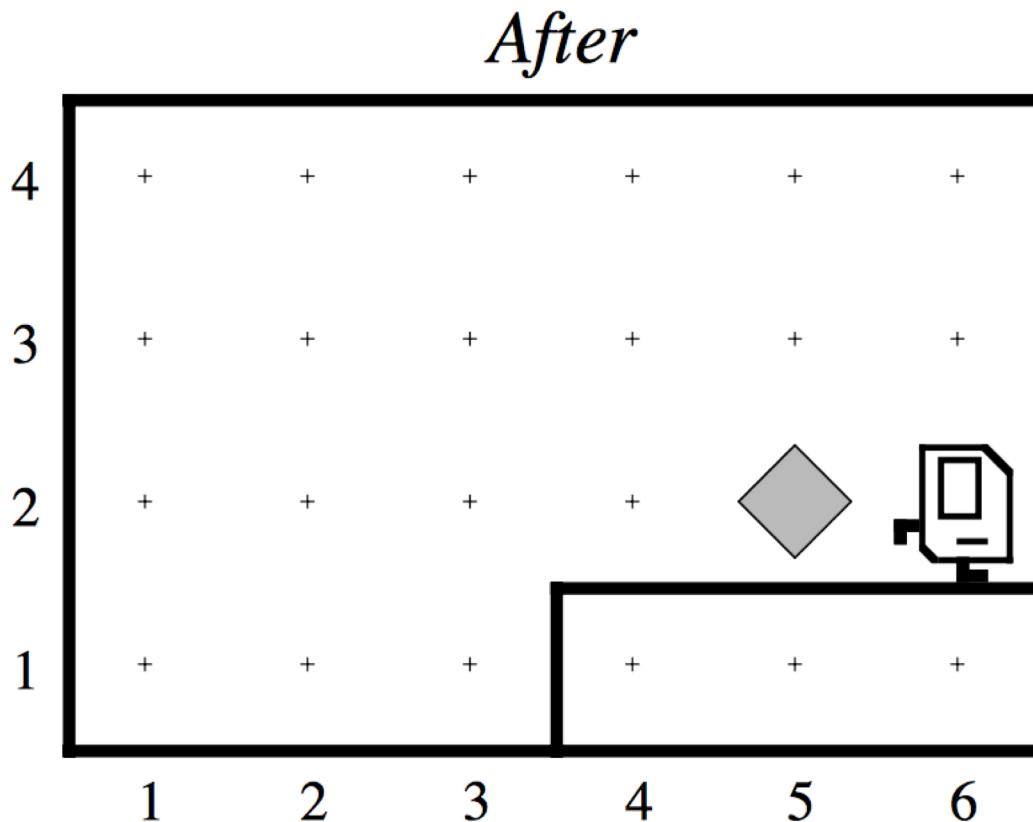
move  
turnLeft  
pickBeeper  
**putBeeper**

- putBeeper makes Karel put a beeper down at its current location.
  - pickBeeper and putBeeper are used to move beepers around.

# Our First Karel Program



# Our First Karel Program



# *Demo*

# Defining New Commands

We can make new commands (or **methods**) for Karel. This lets us *decompose* our program into smaller pieces that are easier to understand.

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

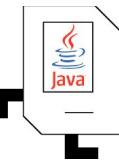
For example:

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

# Plan For Today

- (Re)Meet Karel the Robot

- Announcements



- Control Flow

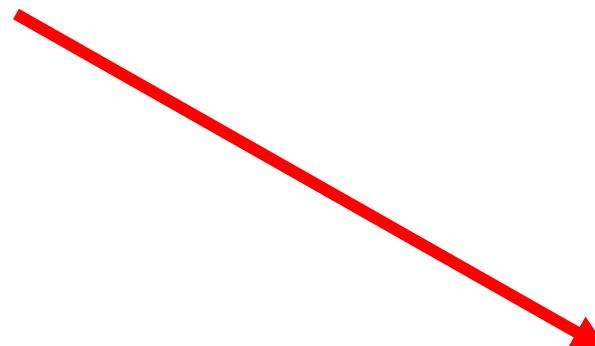
- For loops

- While loops

- If/else statements

# Announcements

- Section assignments out on Wednesday
- Section and LaIR start on Wednesday
- Eclipse help session tonight
- Extra Practice



Tuesday

JUNE 27

2: Programming with  
Karel

Read: Karel Ch. 3-4

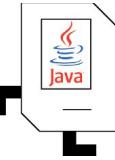
Slides (pdf)

Code (zip)

Practice

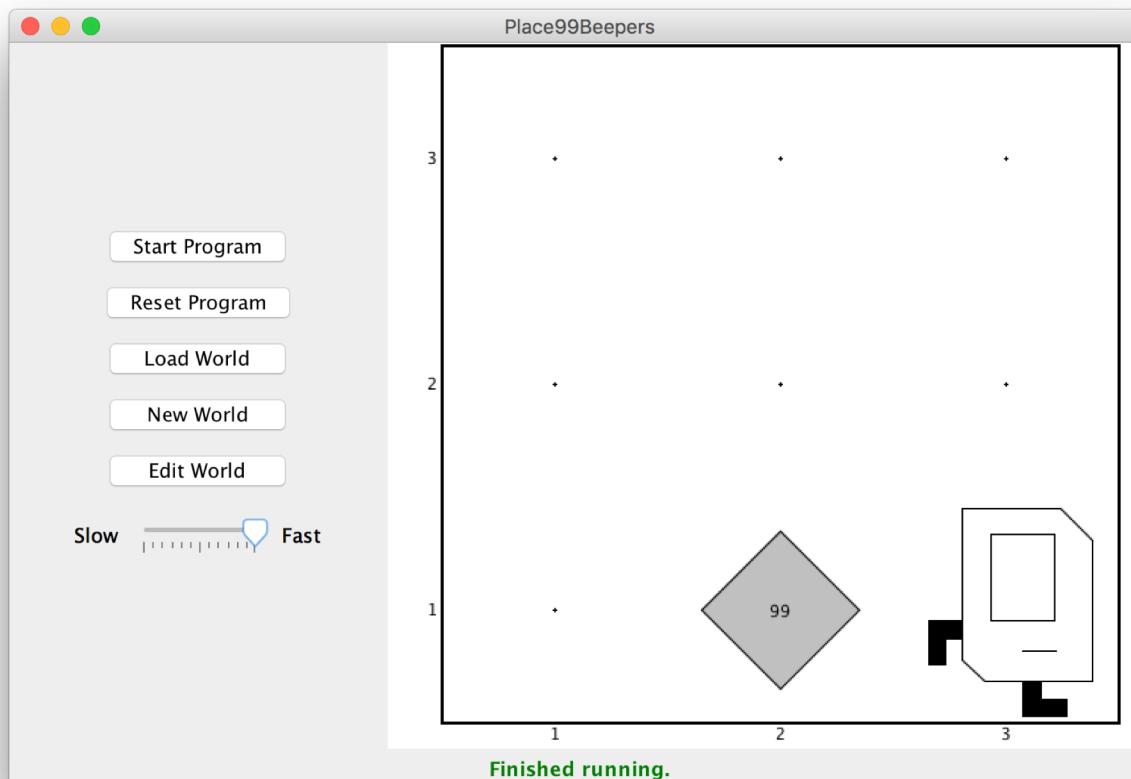
# Plan For Today

- (Re)Meet Karel the Robot
- Announcements
- Control Flow
  - For loops
  - While loops
  - If/else statements



# Control Flow: For Loops

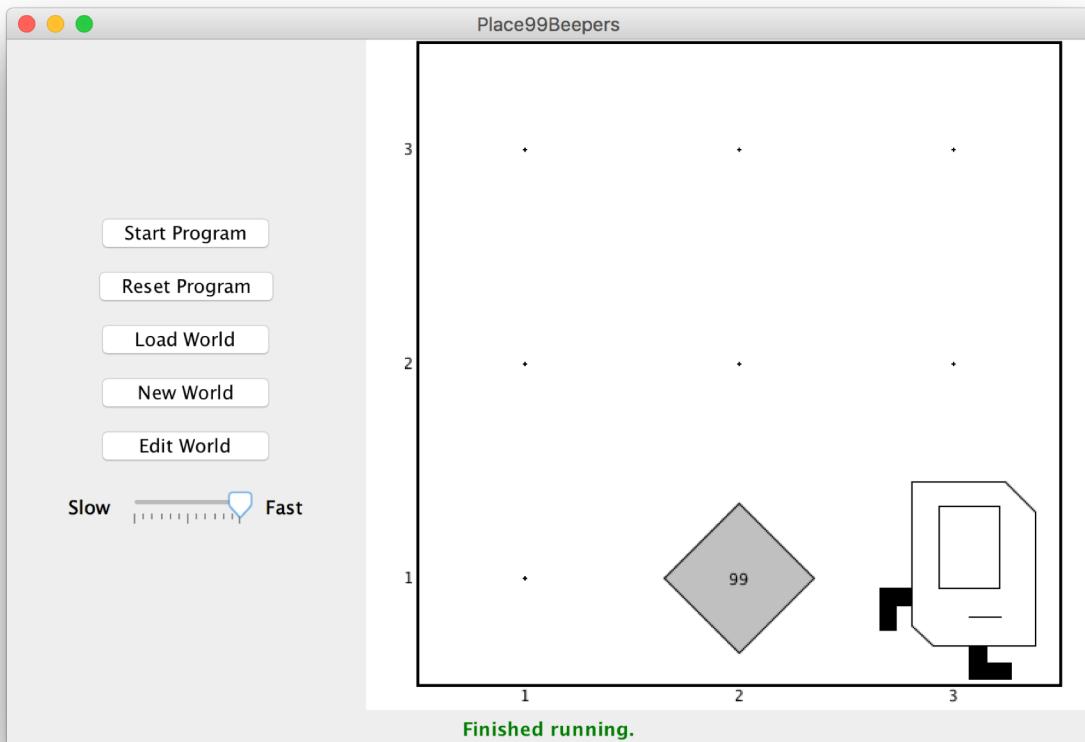
- I want to make Karel put 99 beepers down on a corner. How do I do this?



# Control Flow: For Loops

Can't just say:

```
move();  
putBeeper();  
putBeeper();  
putBeeper();  
...  
move();
```



This is too repetitive! Plus, it's difficult to change (e.g. to 25 beepers).

# Control Flow: For Loops

Instead, use a **for** loop:

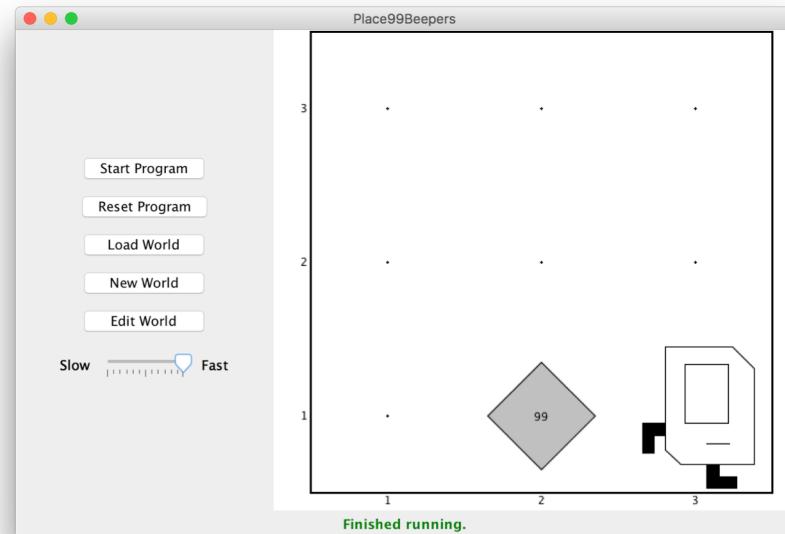
```
for (int i = 0; i < max; i++) {  
    statement;  
    statement;  
    ...  
}
```

Repeats the statements in the body **max** times.

# Control Flow: For Loops

Now we can say:

```
move();  
for (int i = 0; i < 99; i++) {  
    putBeeper();  
}  
move();
```



This is less repetitive, and is easier to change (e.g., to 25 beepers).

# Control Flow: For Loops

Some examples of using **for** loops:

```
// turns Karel right
for (int i = 0; i < 3; i++) {
    turnLeft();
}

// Moves Karel in a square
for (int i = 0; i < 4; i++) {
    move();
    turnLeft();
}
```

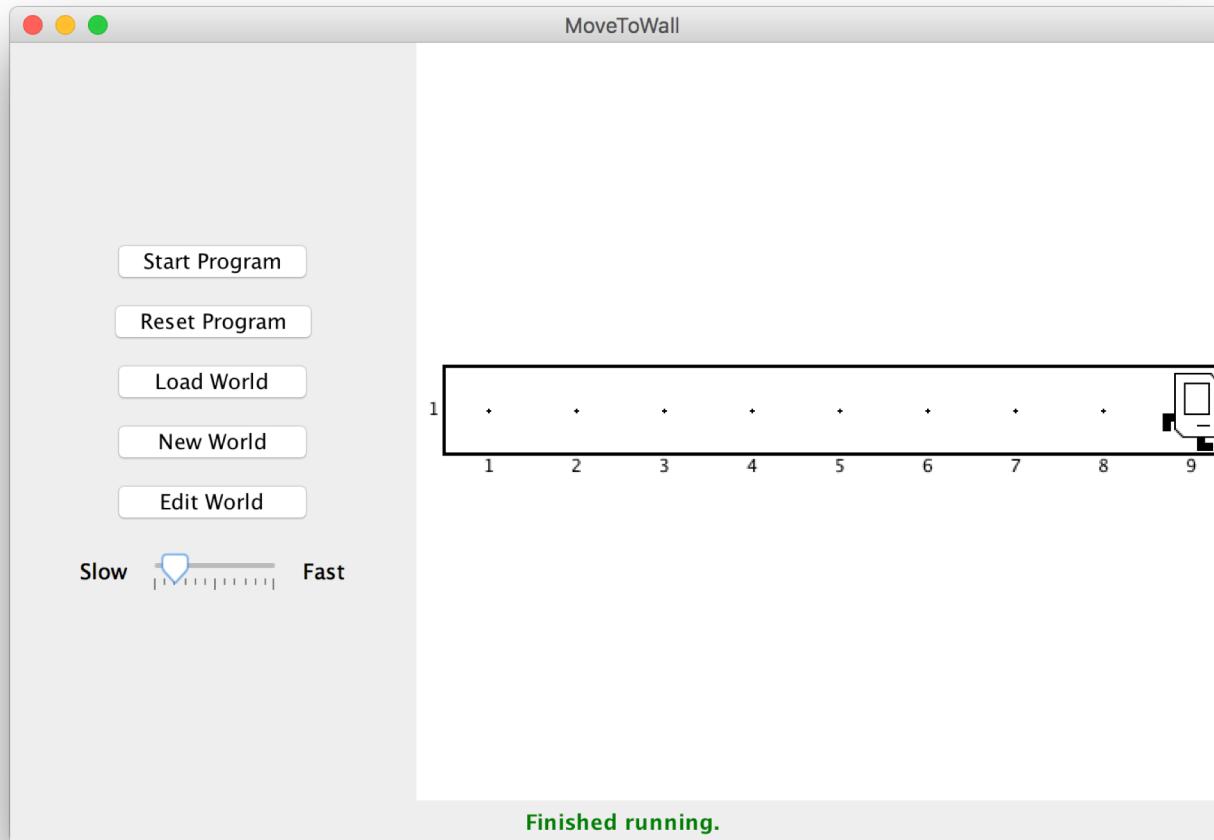
# Plan For Today

- Announcements
- (Re)Meet Karel the Robot
- Control Flow
  - For loops
  - While loops
  - If/else statements



# Control Flow: While Loops

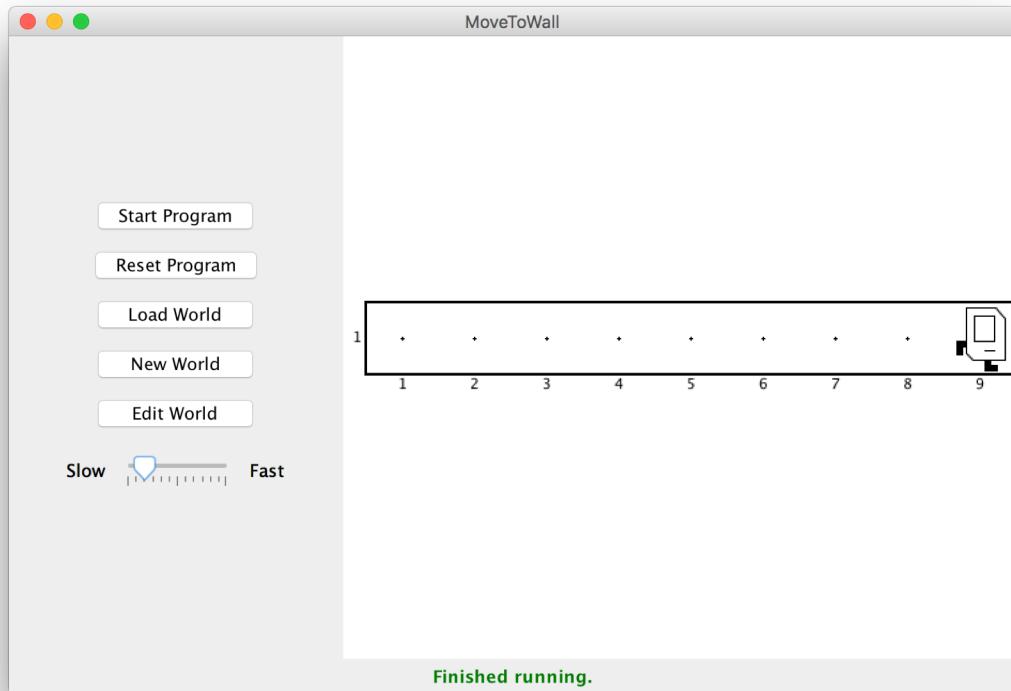
- I want Karel to move until it gets to a wall. How do I do this?



# Control Flow: While Loops

Can't just say:

```
move();  
move();  
move();  
move();  
move();  
...  
...
```



This is too repetitive! Also, we might not know how far away a wall is. Plus, we want our program to be as *generalized* as possible and work in many different worlds.

# Control Flow: While Loops

Instead, use a **while** loop:

```
while (condition) {  
    statement;  
    statement;  
    ...  
}
```

Repeats the statements in the body until **condition** is no longer true.  
Each time, Karel executes *all statements*, and **then** checks the condition.

# Possible Conditions

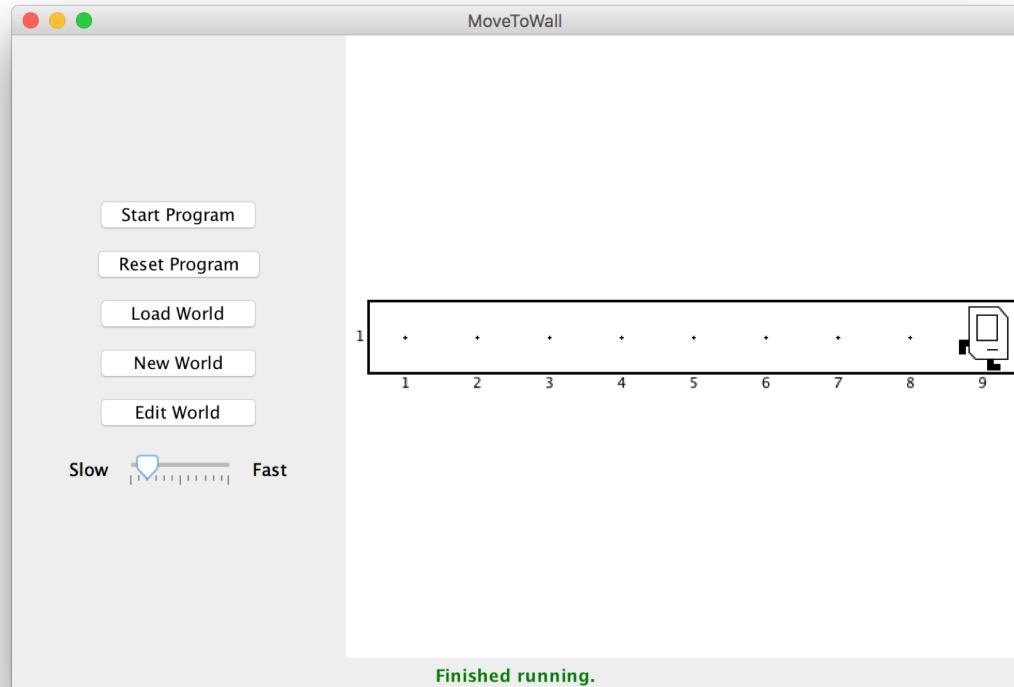
<i>Test</i>	<i>Opposite</i>	<i>What it checks</i>
<code>frontIsClear()</code>	<code>frontIsBlocked()</code>	Is there a wall in front of Karel?
<code>leftIsClear()</code>	<code>leftIsBlocked()</code>	Is there a wall to Karel's left?
<code>rightIsClear()</code>	<code>rightIsBlocked()</code>	Is there a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

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

# Control Flow: While Loops

**while** loop syntax for reference:

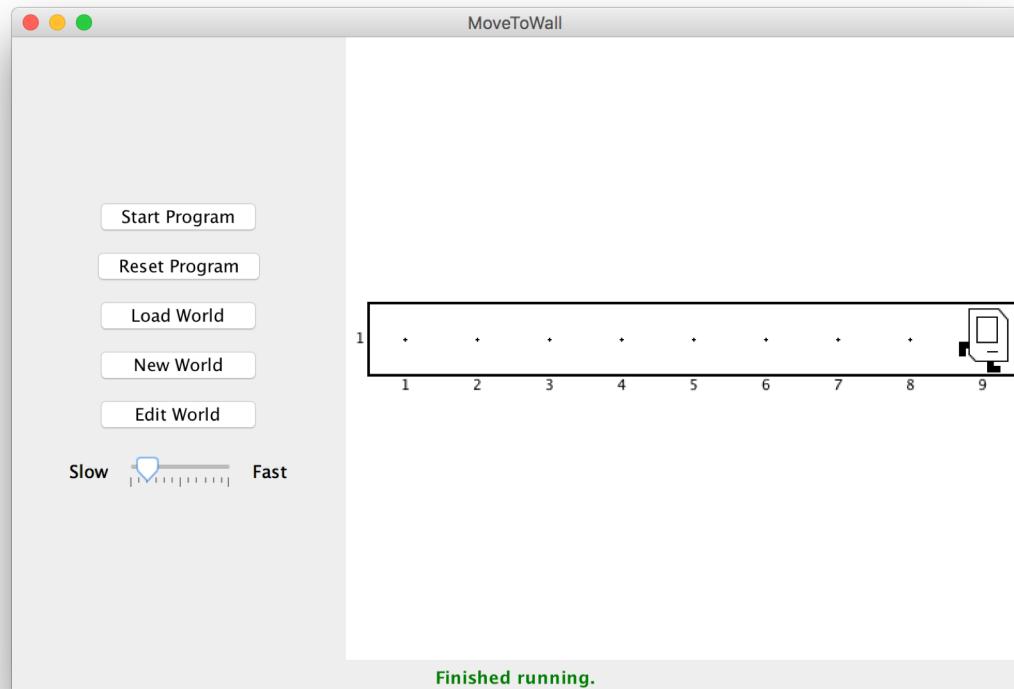
```
while (condition) {  
    statement;  
    statement;  
    ...  
}
```



# Control Flow: While Loops

Now we can say:

```
while (frontIsClear()) {  
    move();  
}
```



This is less repetitive, and it works in *any size* world!

# Control Flow: While Loops

**while** loops can have *compound* conditions as well:

```
// "and"
while (frontIsClear() && beepersPresent()) {
    ...
}
```

```
// "or"
while (leftIsClear() || rightIsClear()) {
    ...
}
```

# Loops Overview

I want Karel to repeat  
some commands!

Know how  
many times

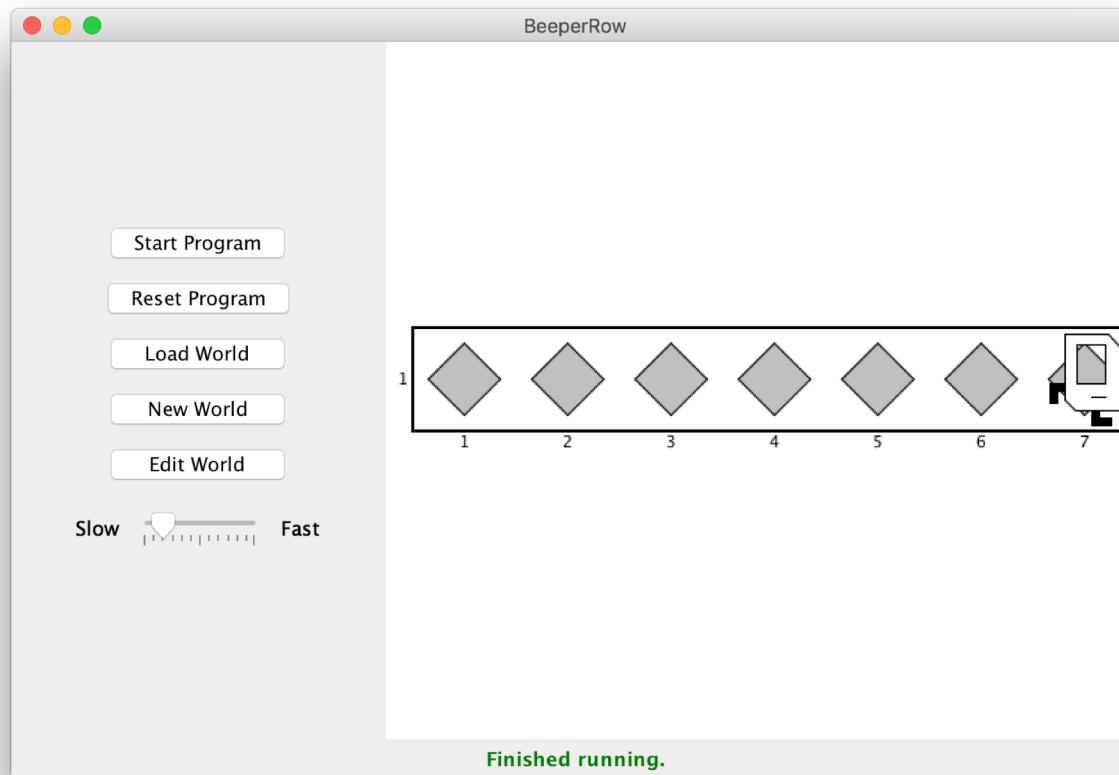
**for** loop

**Don't** know  
how many times

**while** loop

# Loops Overview

- I want Karel to put down a row of beepers until it reaches a wall. How do I do this?



# *Demo*

# Fencepost Problem



*8 fence segments, but 9 posts!*

# Fencepost Structure

The fencepost structure is useful when you want to loop a set of statements, but do one part of that set 1 *additional* time.

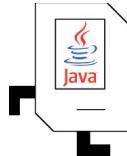
```
putBeeper();           // post
while (frontIsClear()) {
    move();           // fence
    putBeeper();       // post
}

while (frontIsClear()) {
    putBeeper();       // post
    move();           // fence
}

putBeeper();           // post
```

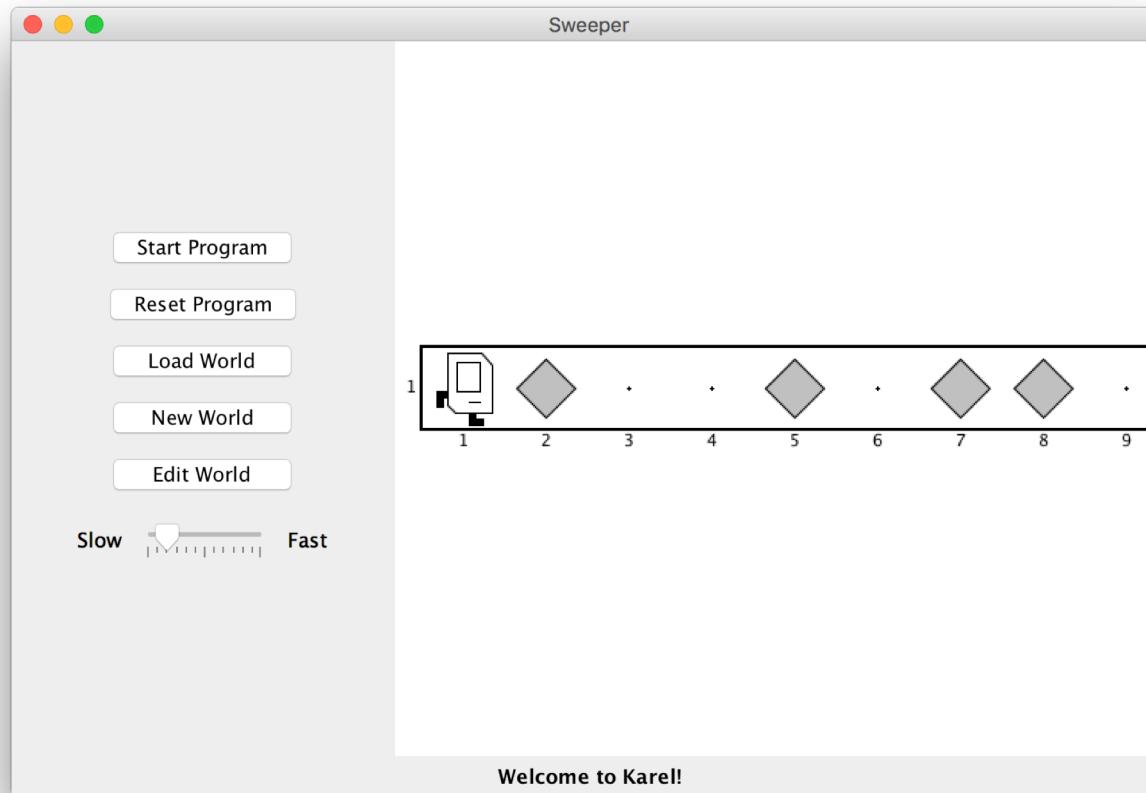
# Plan For Today

- Announcements
- (Re)Meet Karel the Robot
- Control Flow
  - For loops
  - While loops
  - If/else statements



# If/Else Statements

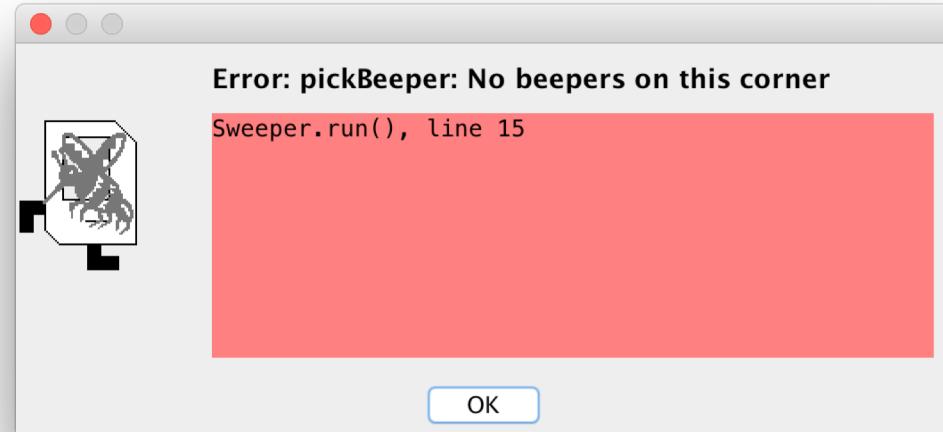
- I want to make Karel clean up all beepers in front of it until it reaches a wall. How do I do this?



# If/Else Statements

Can't just say:

```
while (frontIsClear()) {  
    move();  
    pickBeeper();  
}
```



This may crash, because Karel *cannot pick up beepers if there aren't any*. We don't **always** want Karel to pick up beepers; just when there is a beeper to pick up.

# If/Else Statements

Instead, use an **if** statement:

```
if (condition) {  
    statement;  
    statement;  
    ...  
}
```

Runs the statements in the body *once* if *condition* is true.

# If/Else Statements

You can also add an **else** statement:

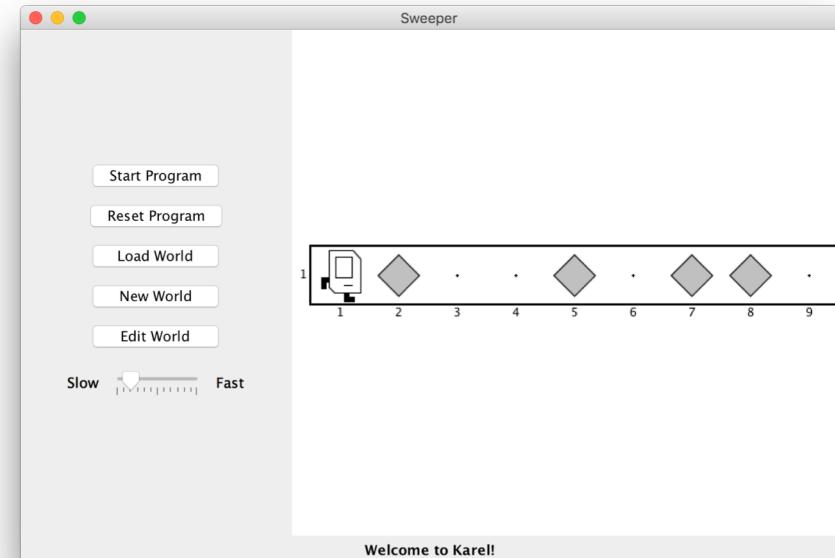
```
if (condition) {  
    statement;  
    statement;  
    ...  
} else {  
    statement;  
    statement;  
    ...  
}
```

Runs the first group of statements if ***condition*** is true; otherwise, runs the second group of statements.

# If/Else Statements

Now we can say:

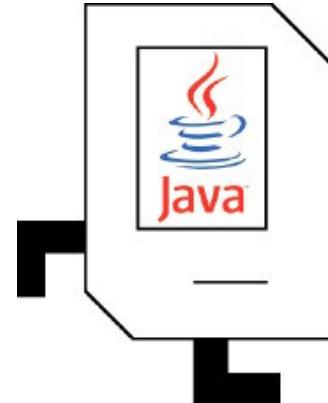
```
while (frontIsClear()) {  
    move();  
    if (beepersPresent()) {  
        pickBeeper();  
    }  
}
```



Now, Karel won't crash because it will only pickBeeper if there is one.

# Recap

- (Re)Meet Karel the Robot
- Announcements
- Control Flow
  - For loops
  - While loops
  - If/else statements



**Next time:** Karel Problem-Solving