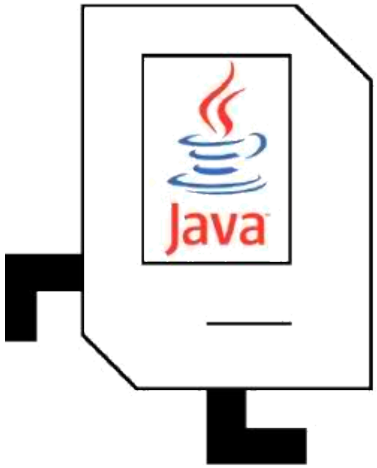
CS 106A, Lecture 3
Problem-solving with Karel

suggested reading:
Karel, Ch. 5-6

This document is copyright (C) Stanford Computer Science and Marty Stepp, licensed under Creative Commons Attribution 2.5 License. All rights reserved.
Based on slides created by Keith Schwarz, Mehran Sahami, Eric Roberts, Stuart Reges, and others.

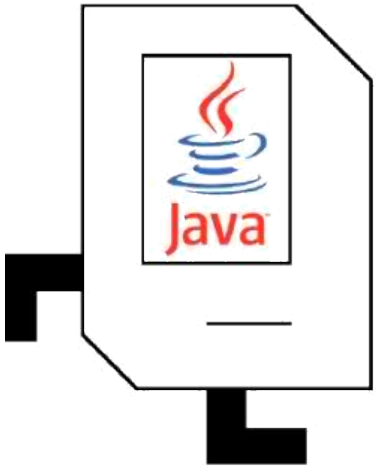
Plan For Today

- Announcements
- Recap: Control Flow
- Demo: HurdleJumper
- Decomposition
- Practice: Roomba
- Debugging



Plan For Today

- Announcements
- Recap: Control Flow
- Demo: HurdleJumper
- Decomposition
- Practice: Roomba
- Debugging



Karel Knows 4 Commands

```
move

turnLeft

putBeeper

pickBeeper
```

Karel Knows 4 Commands

```
move
turnLeft
putBeeper
pickBeeper
```

"methods"

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();
}
```

Control Flow: For Loops

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

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

Control Flow: While Loops

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

Repeats the statements in the body until ***condition*** is no longer true.

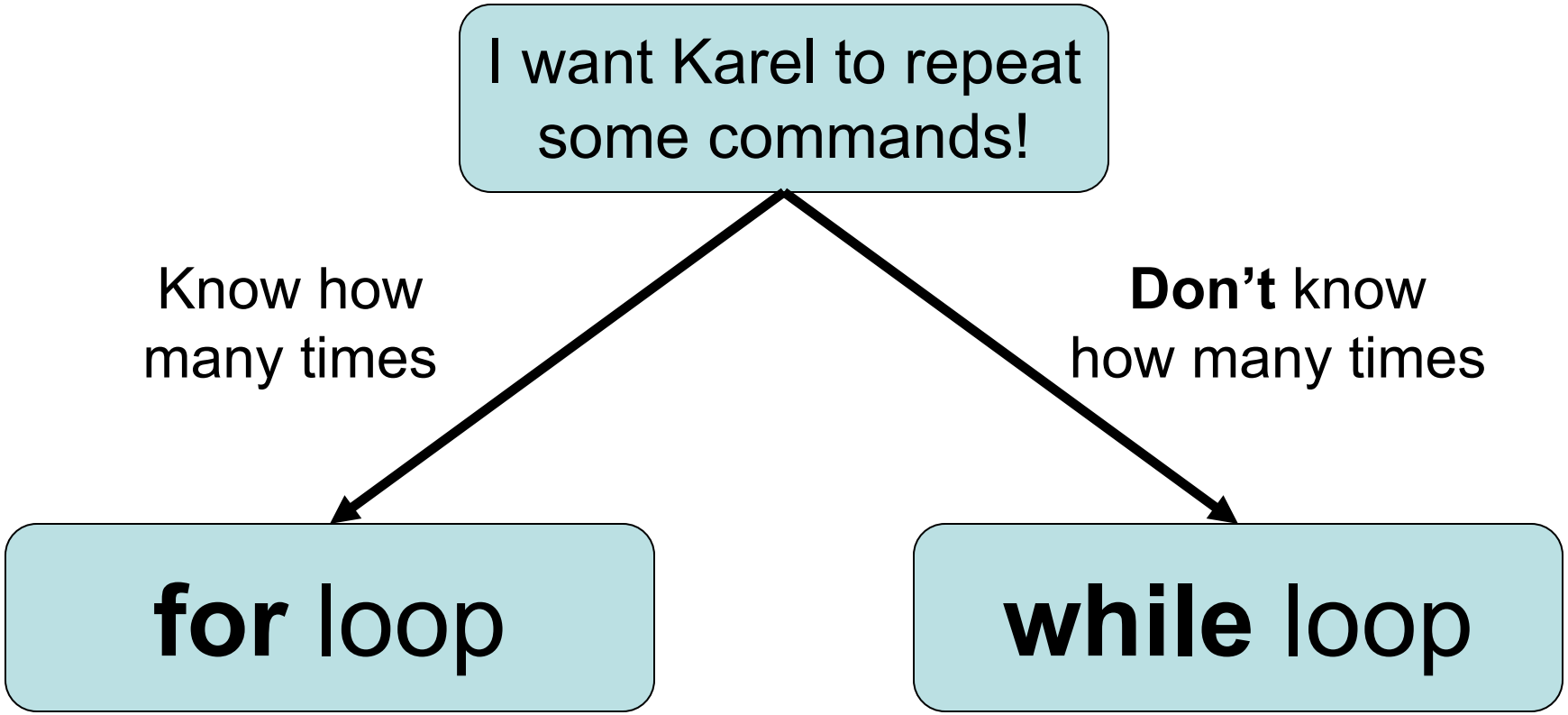
Each time, Karel executes *all statements*, and **then** checks the condition.

Possible Conditions

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 coursereader.*

Loops Overview



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
```

If/Else Statements

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

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

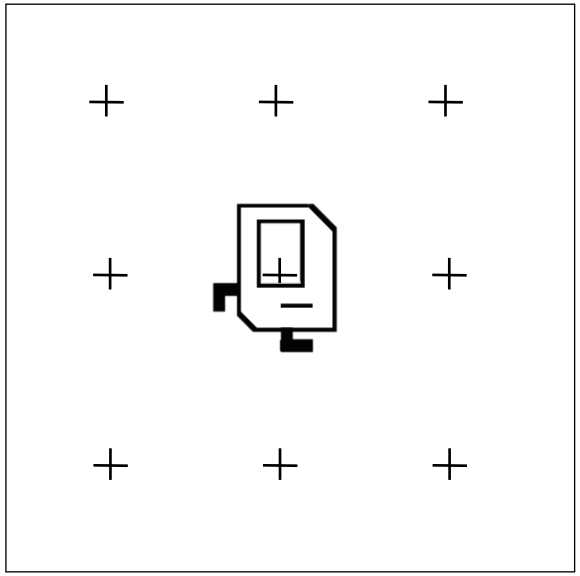
Infinite Loops

Infinite Loops

Rinse
Lather
Repeat

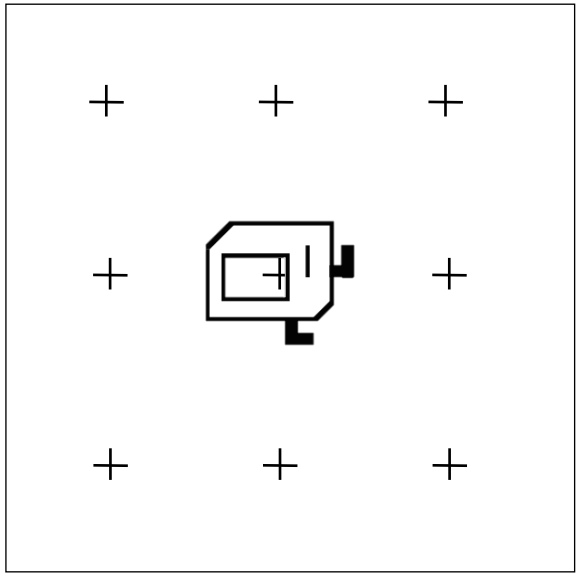
Infinite Loops

```
private void turnToWall() {
    while(leftIsClear()) {
        turnLeft();
    }
}
```



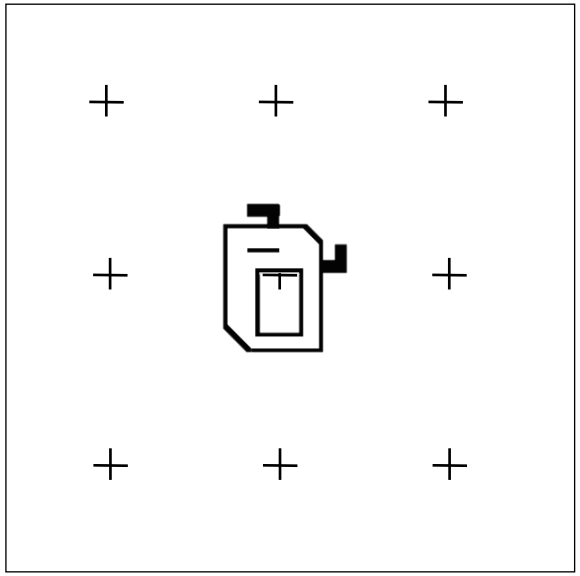
Infinite Loops

```
private void turnToWall() {
    while(leftIsClear()) {
        turnLeft();
    }
}
```



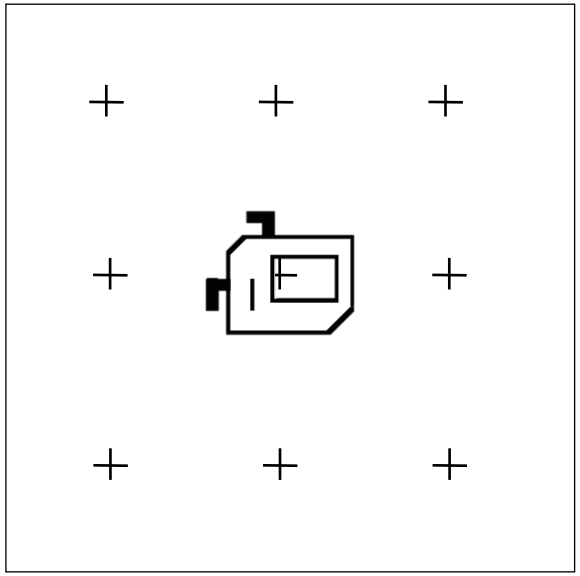
Infinite Loops

```
private void turnToWall() {
    while(leftIsClear()) {
        turnLeft();
    }
}
```



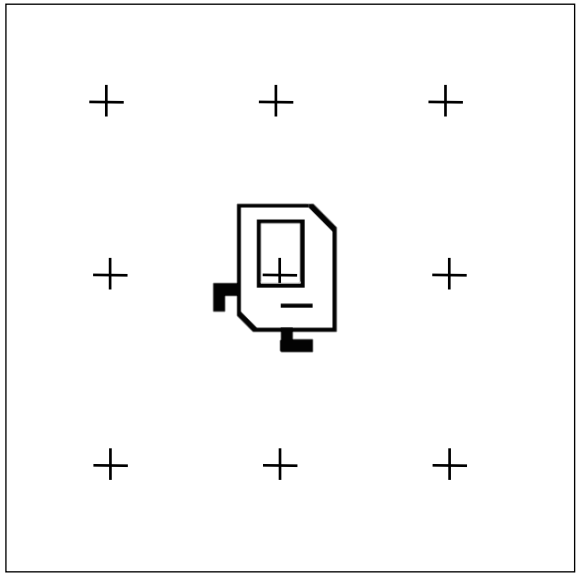
Infinite Loops

```
private void turnToWall() {
    while(leftIsClear()) {
        turnLeft();
    }
}
```



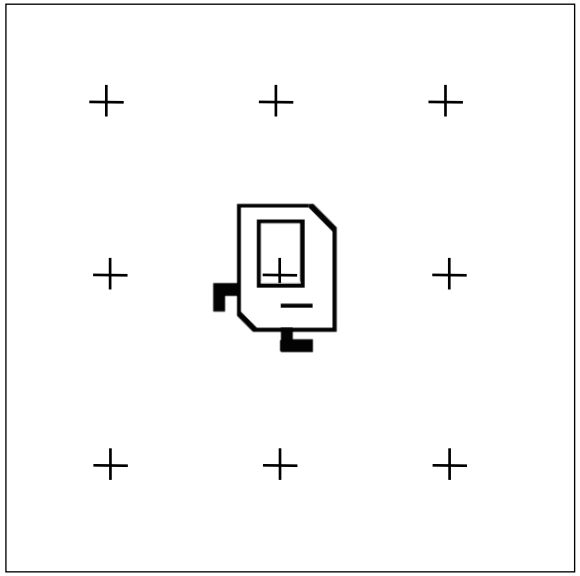
Infinite Loops

```
// Karel will keep turning left forever!
private void turnToWall() {
   while(leftIsClear()) {
      turnLeft();
   }
}
```



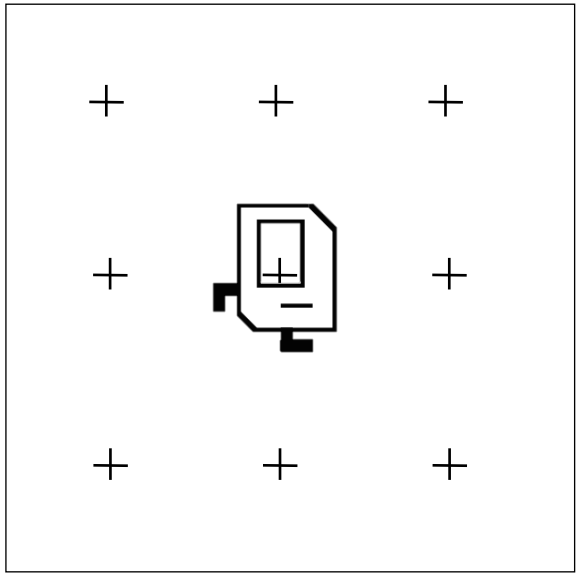
Infinite Loops

```
private void turnToWall() {
    while(leftIsClear()) {
        if (frontIsBlocked()) {
            turnLeft();
        }
    }
}
```



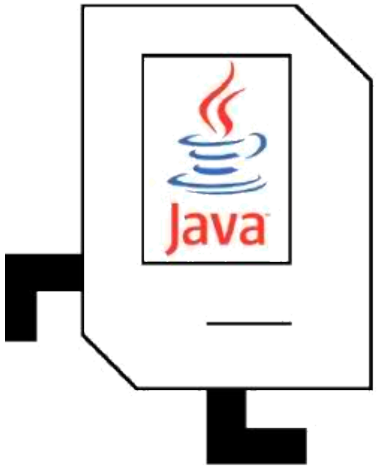
Infinite Loops

```
// Karel will be stuck in this loop forever!
private void turnToWall() {
   while(leftIsClear()) {
      if (frontIsBlocked()) {
         turnLeft();
      }
   }
}
```



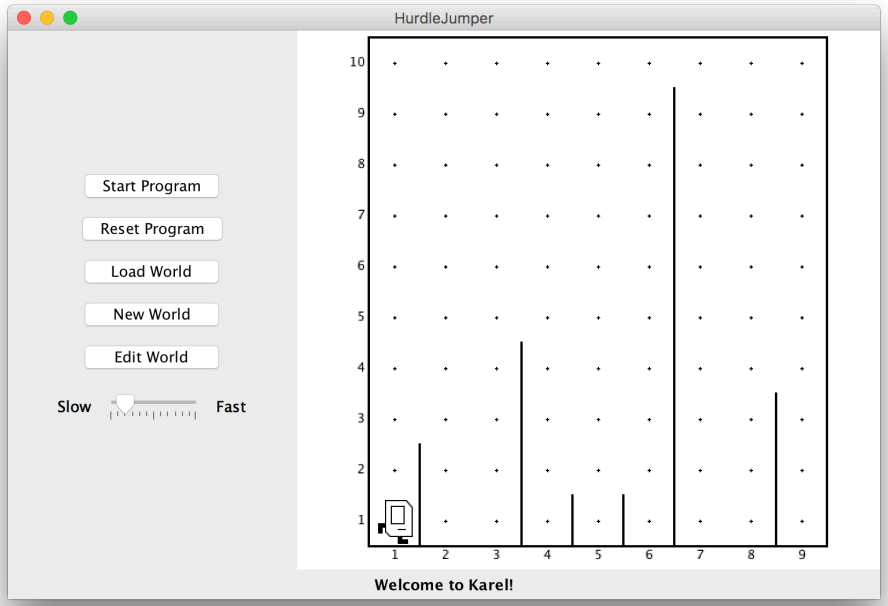
Plan For Today

- Announcements
- Recap: Control Flow
- Demo: HurdleJumper
- Decomposition
- Practice: Roomba
- Debugging



HurdleJumper

- We want to write a Karel program that hops hurdles.
  - Karel starts at (1,1) facing East, and should end up at the end of row 1 facing east.
  - The world has 9 columns.
  - There are an unknown number of ”hurdles” (walls) of varying heights that Karel must ascend and descend to get to the other side.



HurdleJumper

Demo

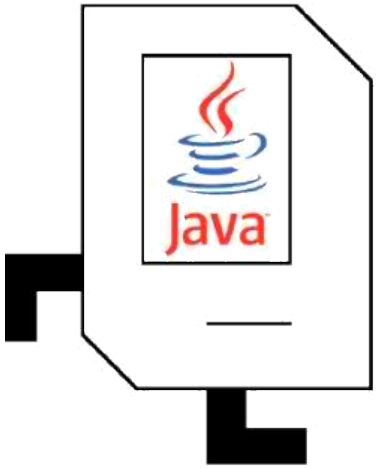
Pre/post comments

- **precondition**: Something you *assume* is true at the start of a method.
- **postcondition**: Something you *promise* is true at the end of a method.
  - pre/post conditions should be documented using comments.

```
/*
 * Jumps Karel over one hurdle of arbitrary height.
 * Pre:  Karel is facing east, next to a hurdle.
 * Post: Karel is facing east at the bottom of the other
 * side of the hurdle.
 */
public void jumpHurdle() {
    ascendHurdle();
    move();
    descendHurdle();
}
```

Plan For Today

- Announcements
- Recap: Control Flow
- Demo: HurdleJumper
- Decomposition
- Practice: Roomba
- Debugging



Decomposition

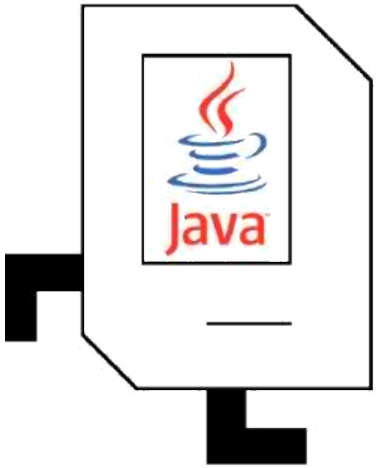
- Breaking down problems into smaller, more approachable sub-problems (e.g. our own Karel commands)
- Each piece should solve **one** problem/task (< ~ 20 lines of code)
  - Descriptively-named
  - Well-commented!
- E.g. getting up in the morning:
  - Wake up
  - Brush teeth
    - Put toothpaste on toothbrush
    - Insert toothbrush into mouth
    - Move toothbrush against teeth
    - …
  - …

Top-Down Design

- Start from a large task and break it up into smaller pieces
- Ok to write your program in terms of commands that don't exist yet
- E.g. HurdleJumper

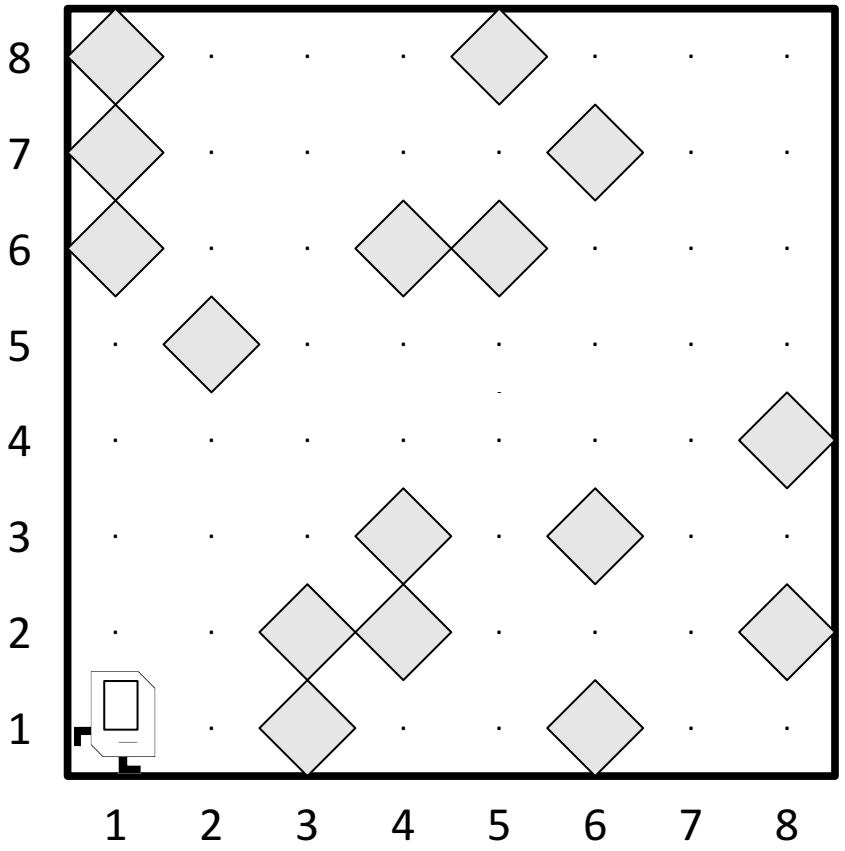
Plan For Today

- Announcements
- Recap: Control Flow
- Demo: HurdleJumper
- Decomposition
- Practice: Roomba
- Debugging



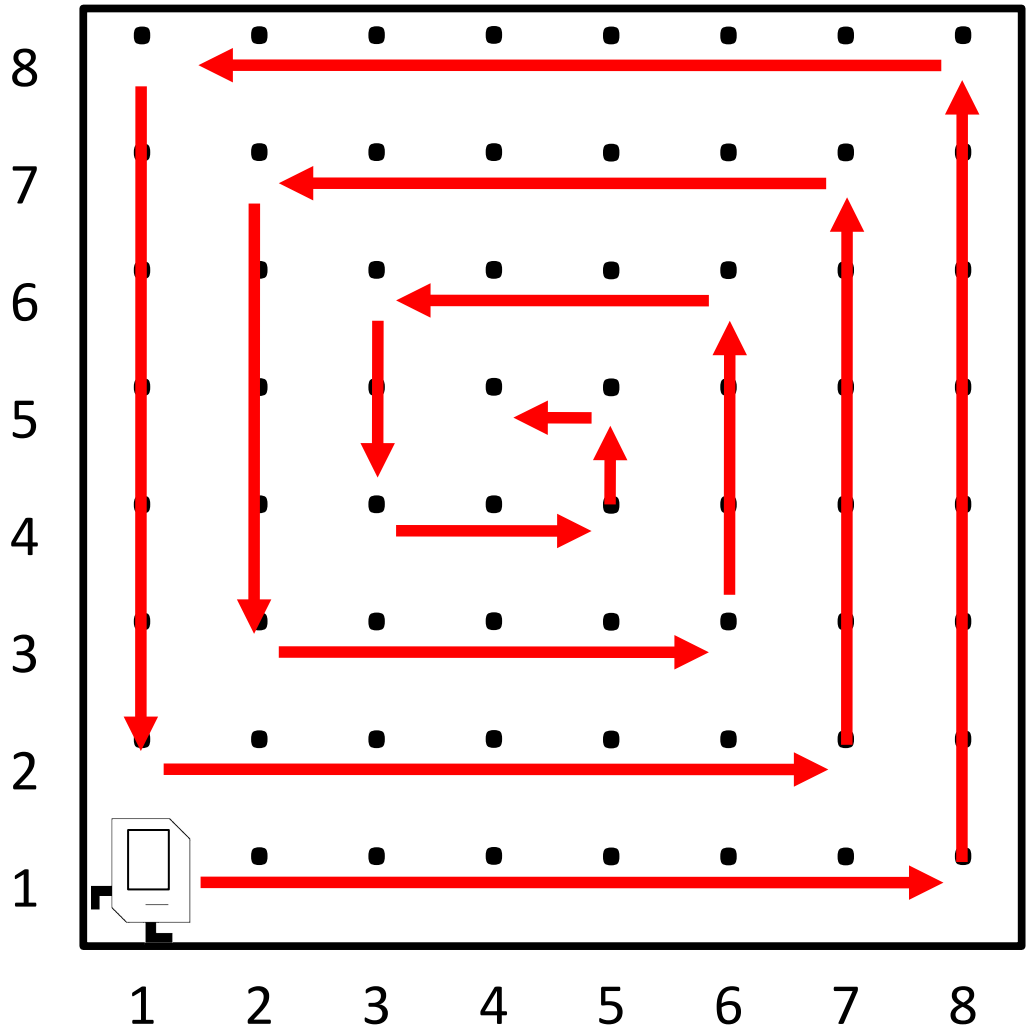
Practice: Roomba

- Write a **Roomba** Karel that sweeps the entire world of all beepers.
  - Karel starts at (1,1) facing East.
  - The world is rectangular, and some squares contain beepers.
  - There are no interior walls.
  - When the program is done, the world should contain 0 beepers.
  - Karel's ending location does not matter.

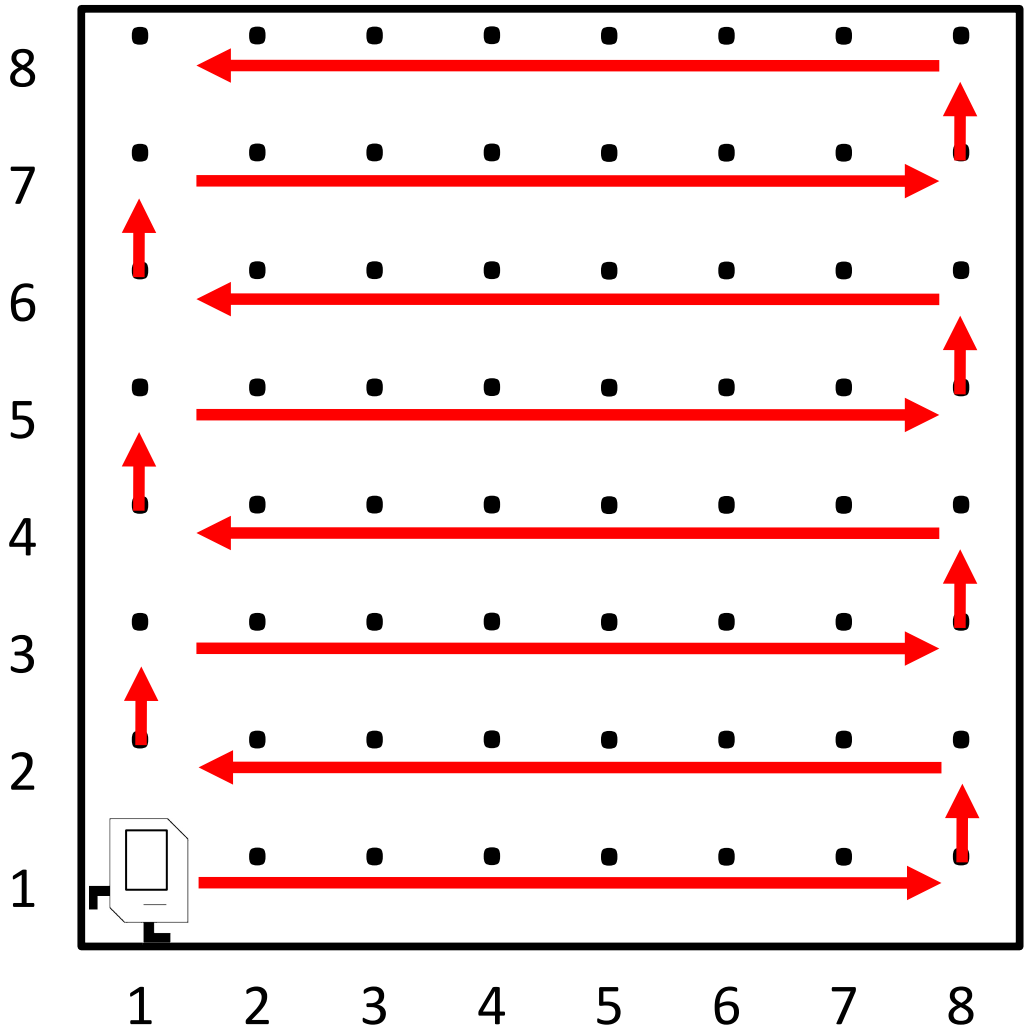


- How should we approach this tricky problem?

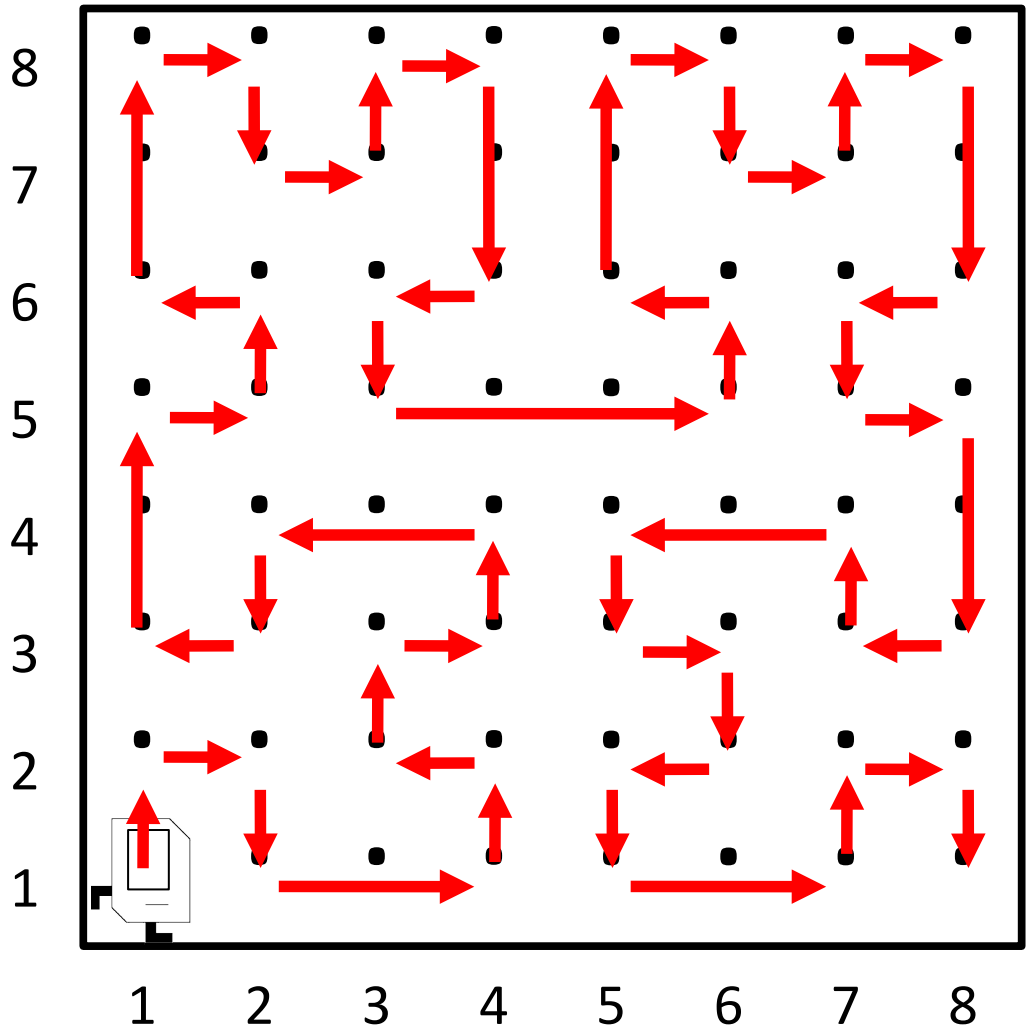
Possible algorithm 1



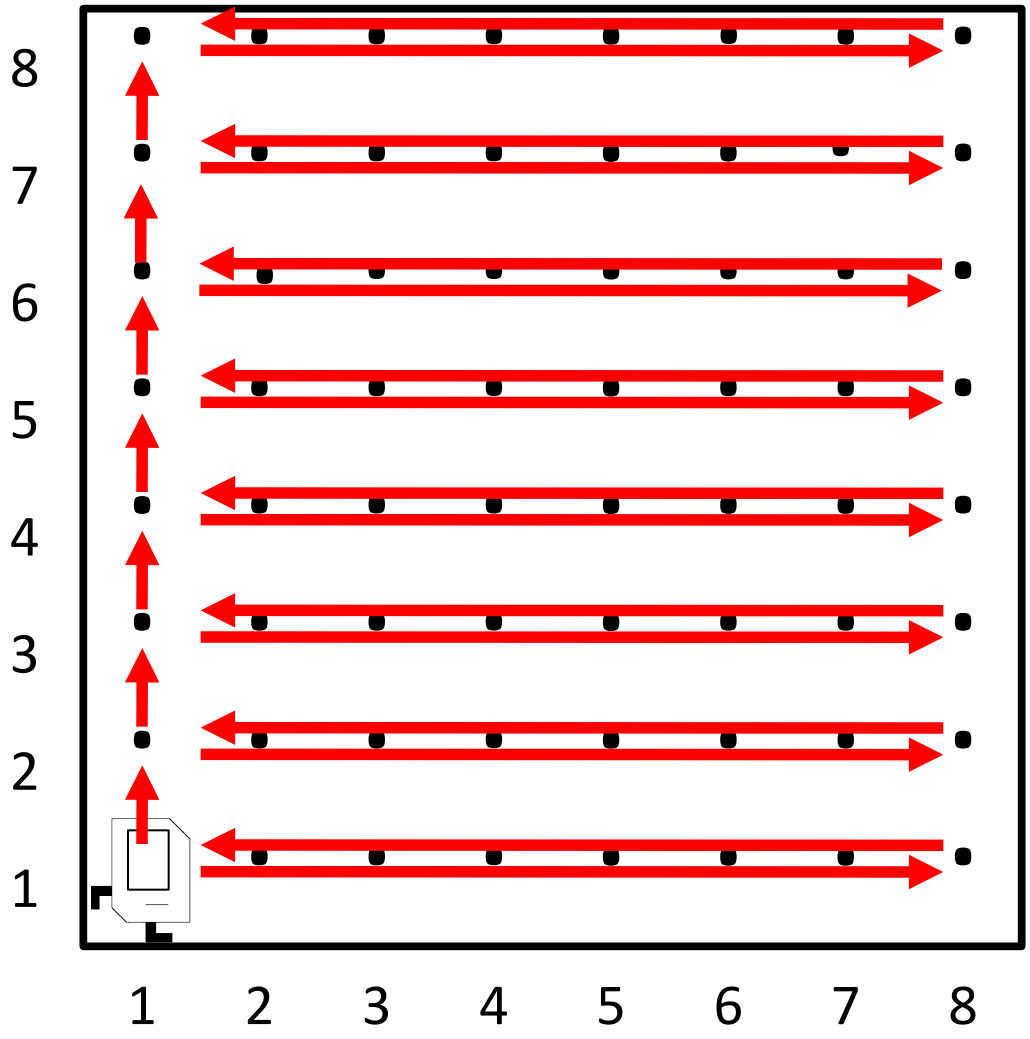
Possible algorithm 2



Possible algorithm 3



Possible algorithm 4

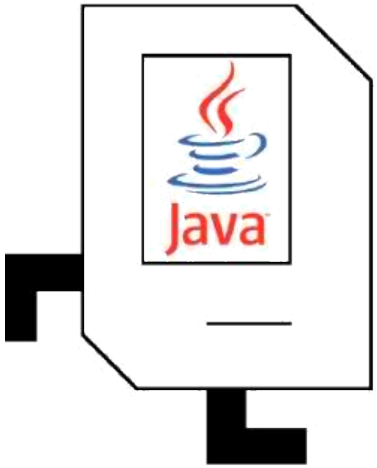


Roomba

Demo

Recap

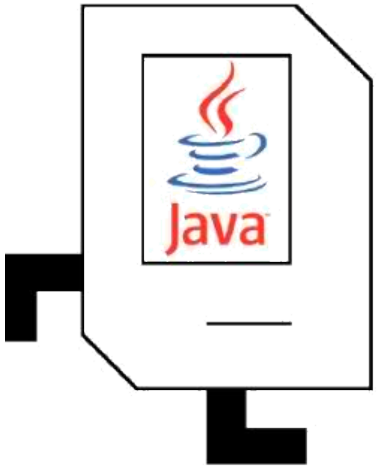
- Announcements
- Recap: Control Flow
- Demo: HurdleJumper
- Decomposition
- Practice: Roomba



Next time: An introduction to Java

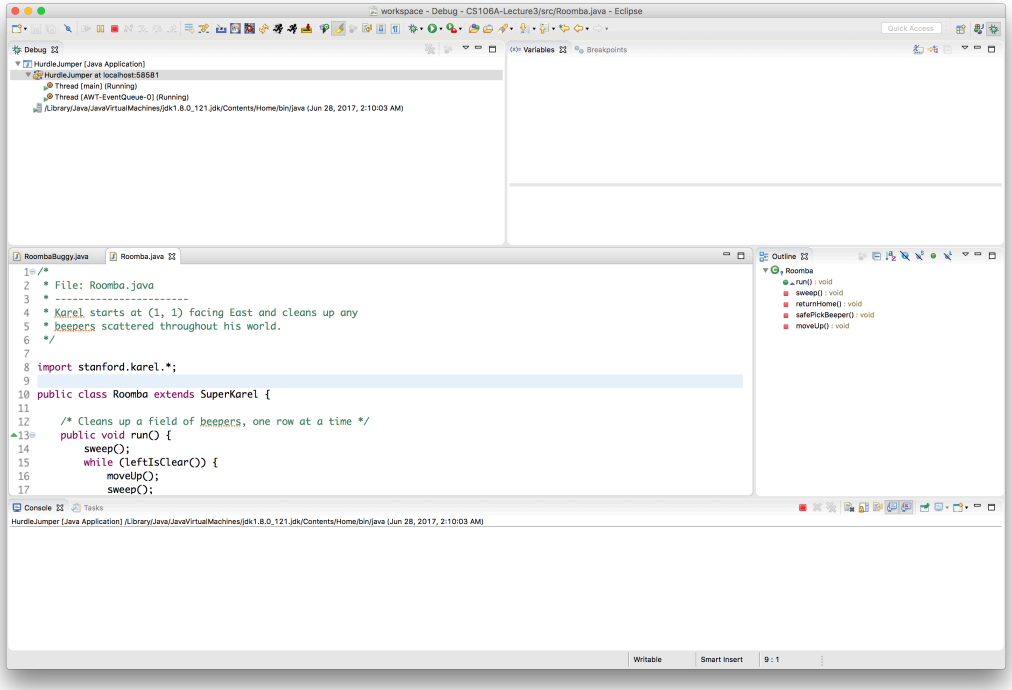
Plan For Today

- Announcements
- Recap: Control Flow
- Demo: HurdleJumper
- Decomposition
- Practice: Roomba
- Debugging



Debugging

- Finding and fixing unintended behavior in your programs.
- Try to narrow down *where* in your code you think the bug is occurring. (E.g. what command or set of commands)
- We can use Eclipse to help us figure out what our program is doing.

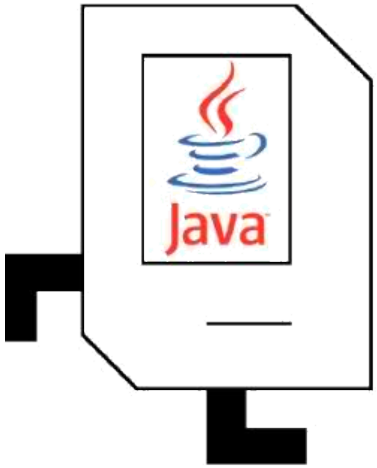


BuggyRoomba

Demo

Recap

- Announcements
- Recap: Control Flow
- Demo: HurdleJumper
- Decomposition
- Practice: Roomba
- Debugging



Next time: An introduction to Java