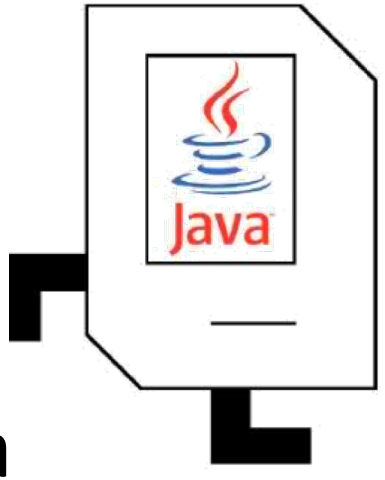
CS 106A, Lecture 3
Problem-solving with Karel

suggested reading:

Karel, Ch. 5-6

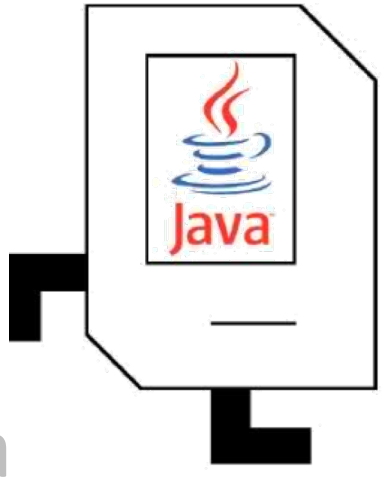
Plan For Today

- Announcements
- Recap: Control Flow
- Control Flow: If/else
- Decomposition
- Demo: HurdleJumper
- Practice: Debugging and Roomba



Plan For Today

- Announcements
- Recap: Control Flow
- Control Flow: If/else
- Decomposition
- Demo: HurdleJumper
- Practice: Debugging and Roomba

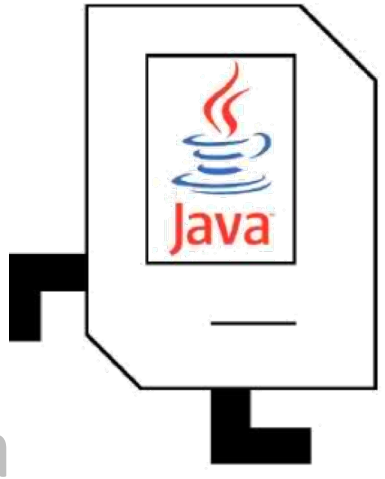


Announcements

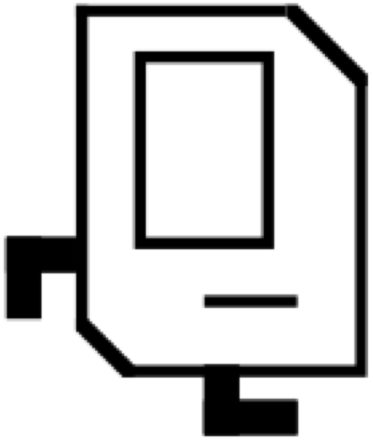
- Sections start today. Assignment on website
- Late signups open
- Email cs198@cs.stanford.edu with schedule conflicts
- Fill out Annie's form if not put with partner
- LaIR starts tonight: ground floor of Tressider
- Piazza for logistics + conceptual questions
- Please raise hands!

Plan For Today

- Announcements
- Recap: Control Flow
- Control Flow: If/else
- Decomposition
- Demo: HurdleJumper
- Practice: Debugging and Roomba



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.

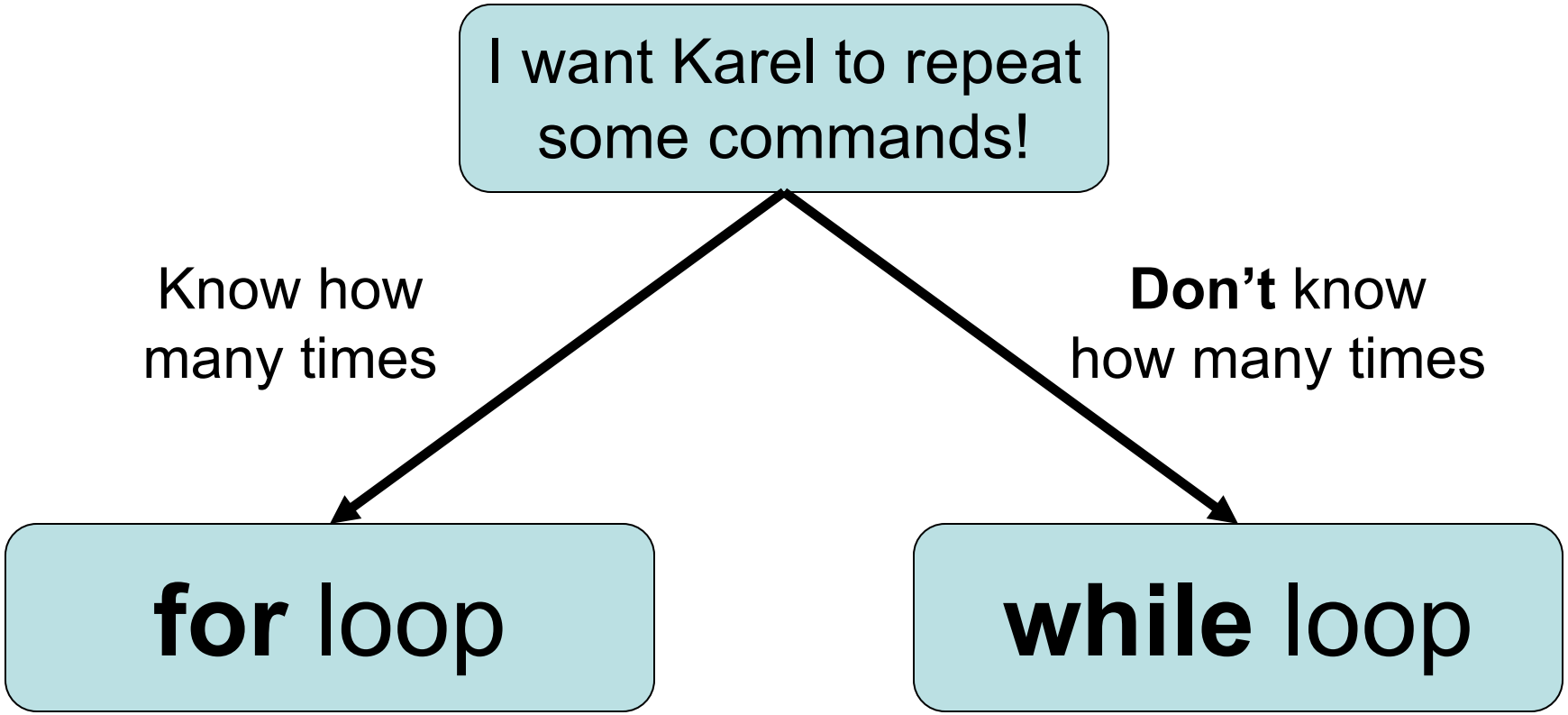
Each time, Karel executes *all statements*, and **then** checks the condition.

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

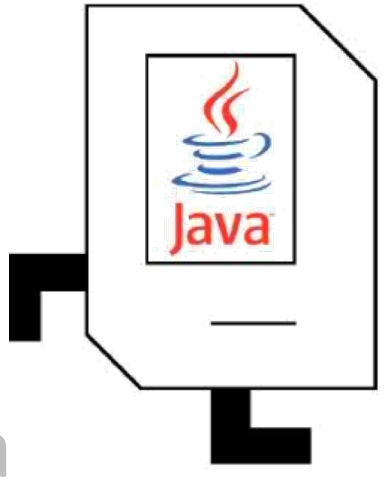
The fencepost structure is useful when you want to loop a set of statements, but do one part of that set 1 *additional* time.

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

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

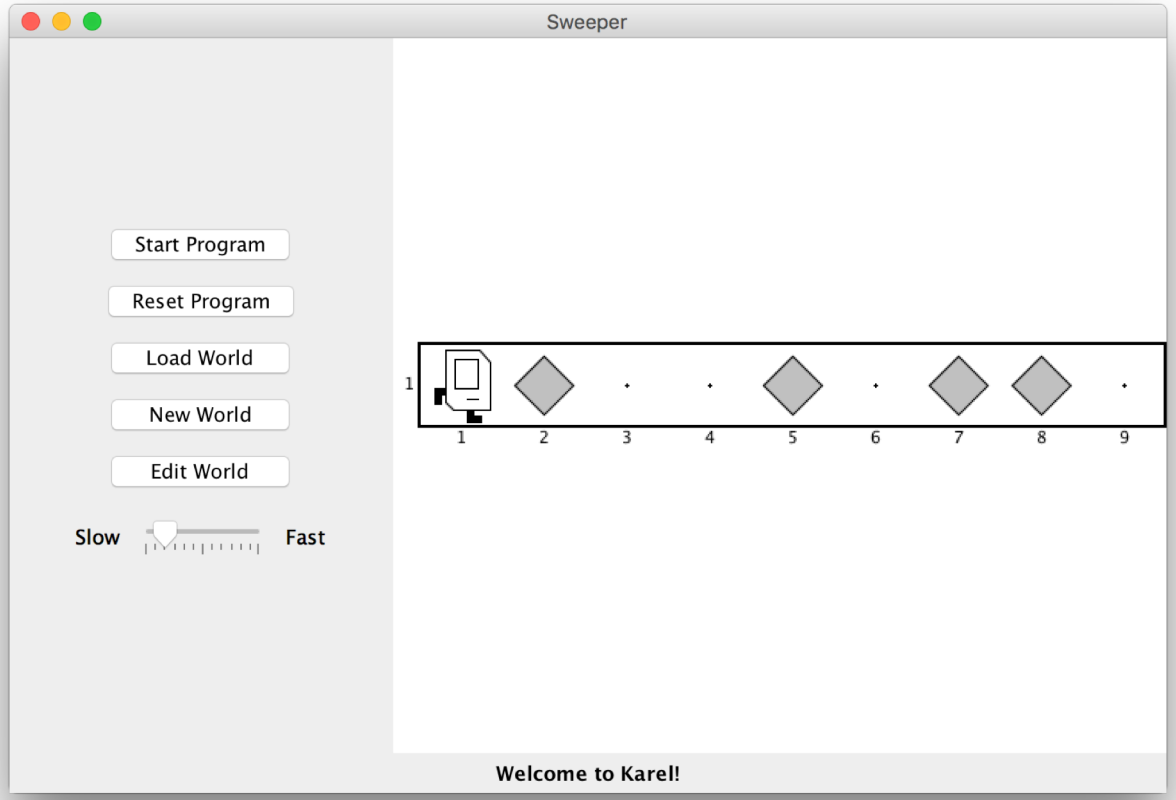
Plan For Today

- Announcements
- Recap: Control Flow
- Control Flow: If/else
- Decomposition
- Demo: HurdleJumper
- Practice: Debugging and Roomba



If/Else Statements

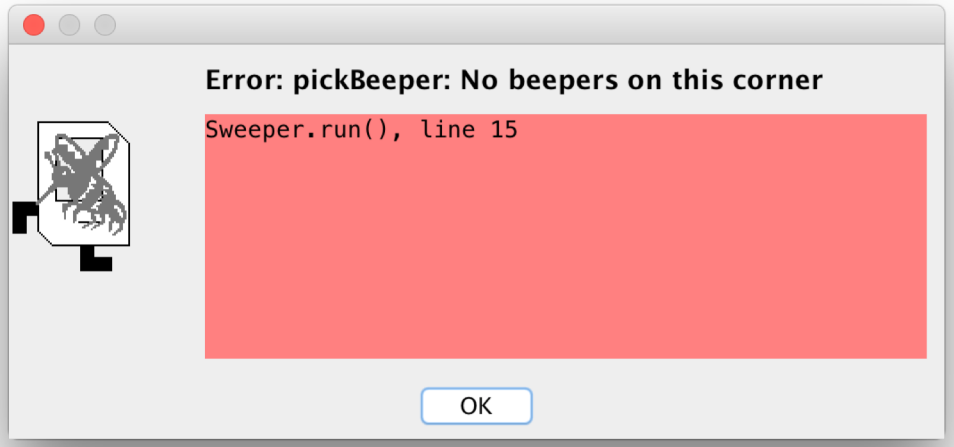
- I want to make Karel clean up all beepers in front of it until it reaches a wall. How do I do this?



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:

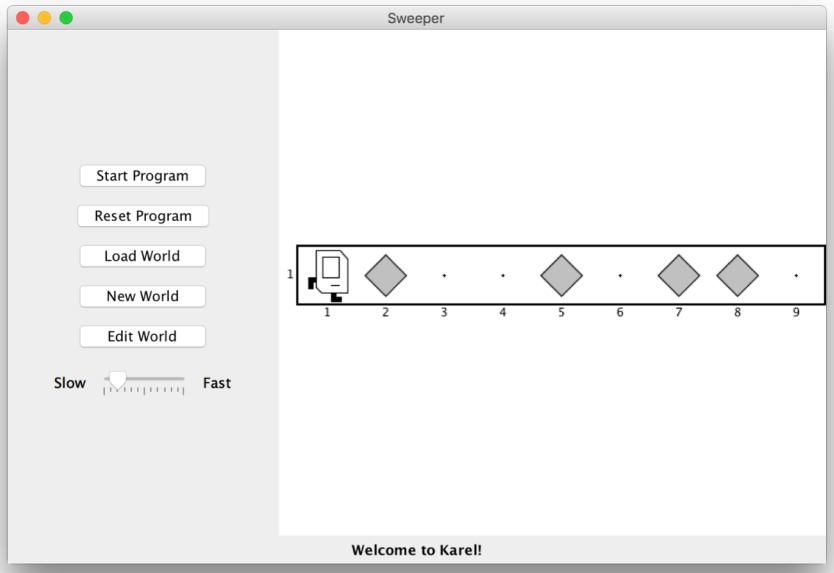
```
if (condition) {
    statement;
    statement;
    ...
} else {
    statement;
    statement;
    ...
}
```

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.

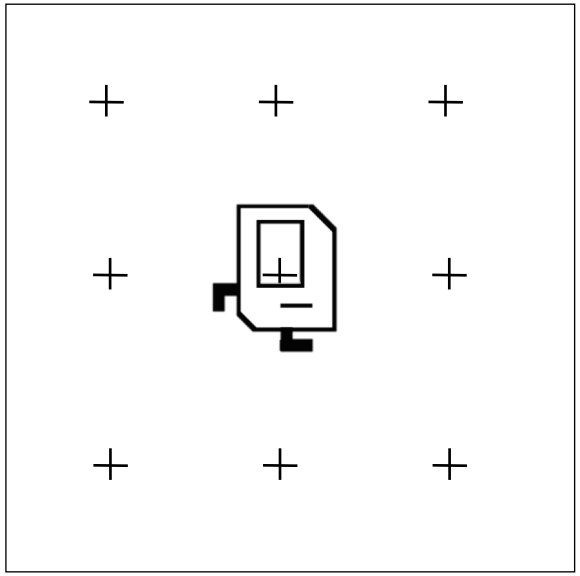
Infinite Loops

Infinite Loops

Lather
Rinse
Repeat

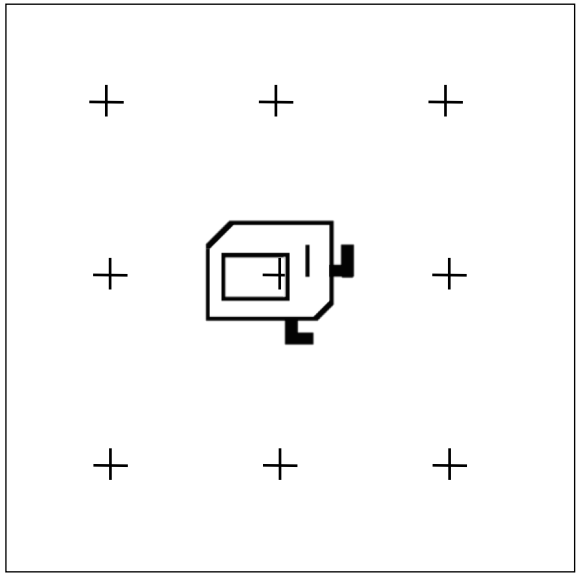
Infinite Loops

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



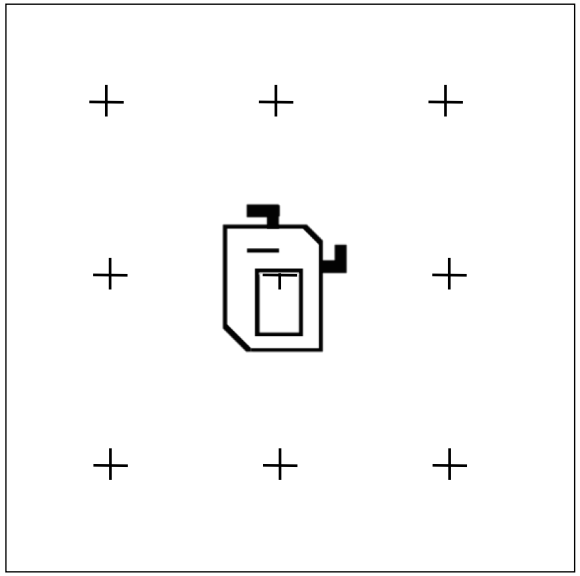
Infinite Loops

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



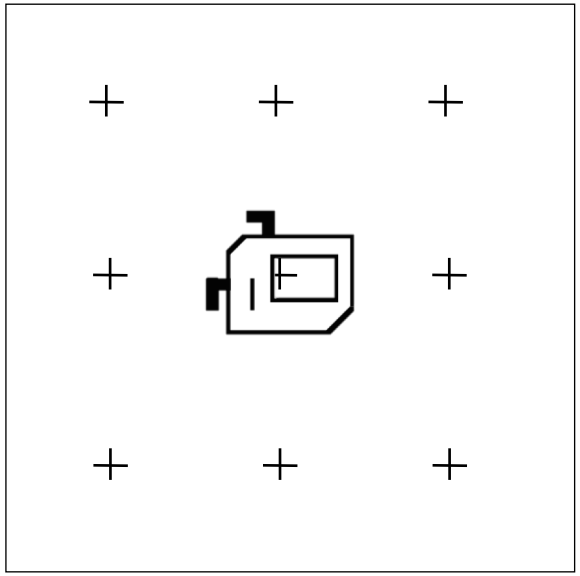
Infinite Loops

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



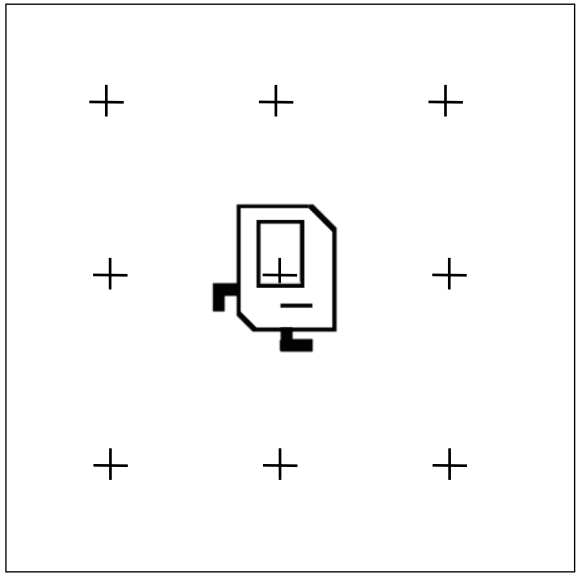
Infinite Loops

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



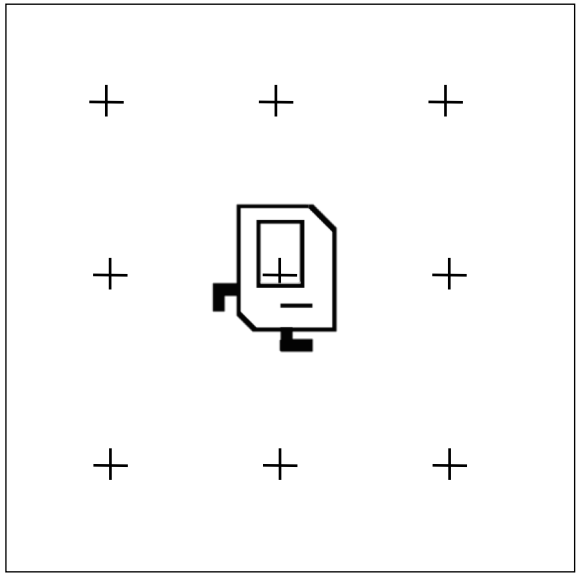
Infinite Loops

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



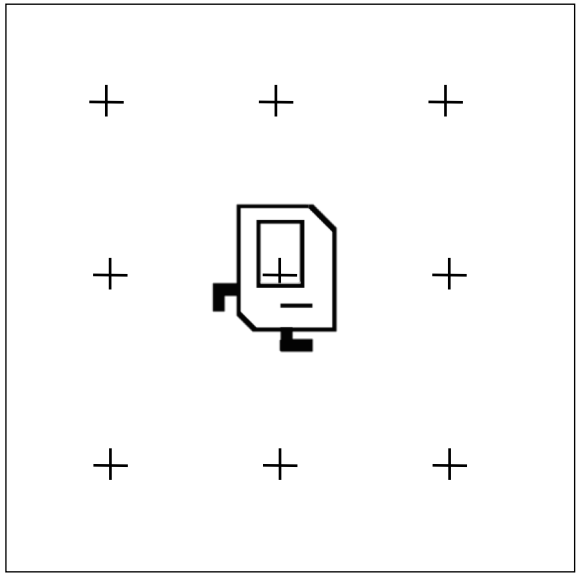
Infinite Loops

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



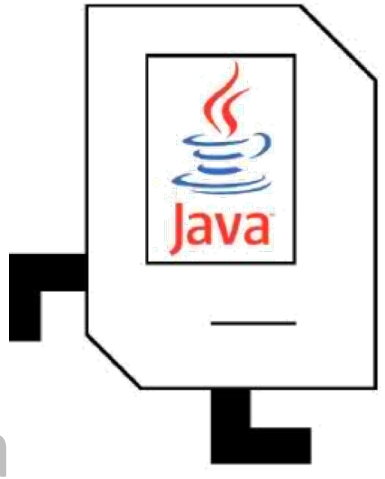
Infinite Loops

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



Plan For Today

- Announcements
- Recap: Control Flow
- Control Flow: If/else
- Decomposition
- Demo: HurdleJumper
- Practice: Debugging and Roomba



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 (in fact, recommended!) to write your program in terms of commands that don’t exist yet

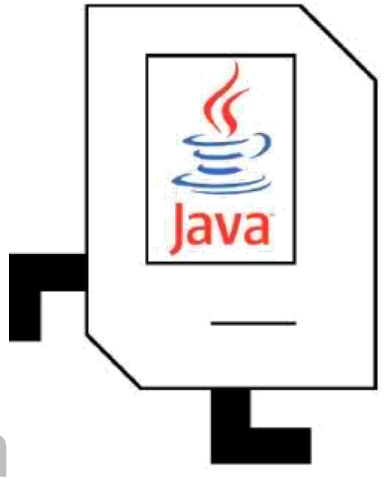
Pre/post comments

- **precondition**: Something you *assume* is true at the start of a method.
- **postcondition**: Something you *promise* is true at the end of a method.
  - Recommendation: write these comments **before** implementing!

```
/*
 * Karel picks up any beepers they find to their right.
 * Pre:  Karel is facing east at (1,1), which has no beeper.
 * Post: Karel is facing east at the end of the first
 * street. There are no beepers in the first street.
 */
private void sweepStreet() {
    while (frontIsClear()) {
        move();
        if (beepersPresent()) {
            pickBeeper();
        }
    }
}
```

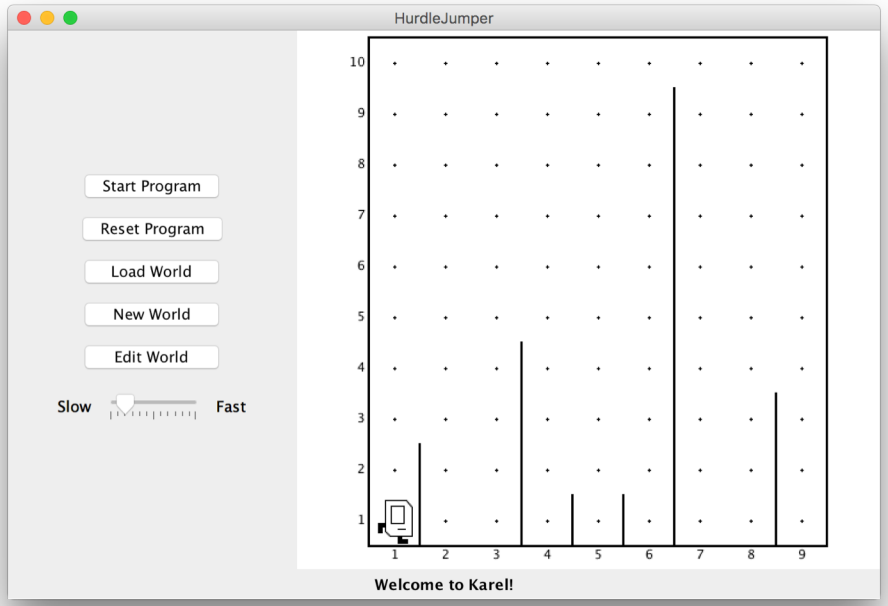
Plan For Today

- Announcements
- Recap: Control Flow
- Control Flow: If/else
- Decomposition
- Demo: HurdleJumper
- Practice: Debugging and Roomba



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.

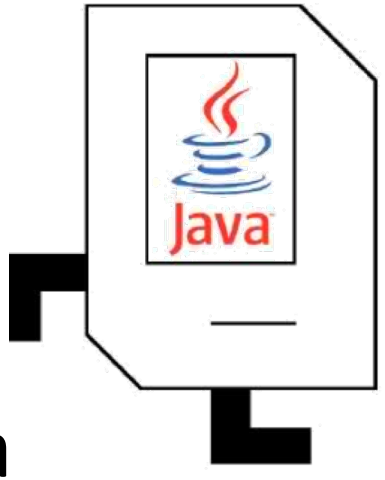


HurdleJumper

Demo

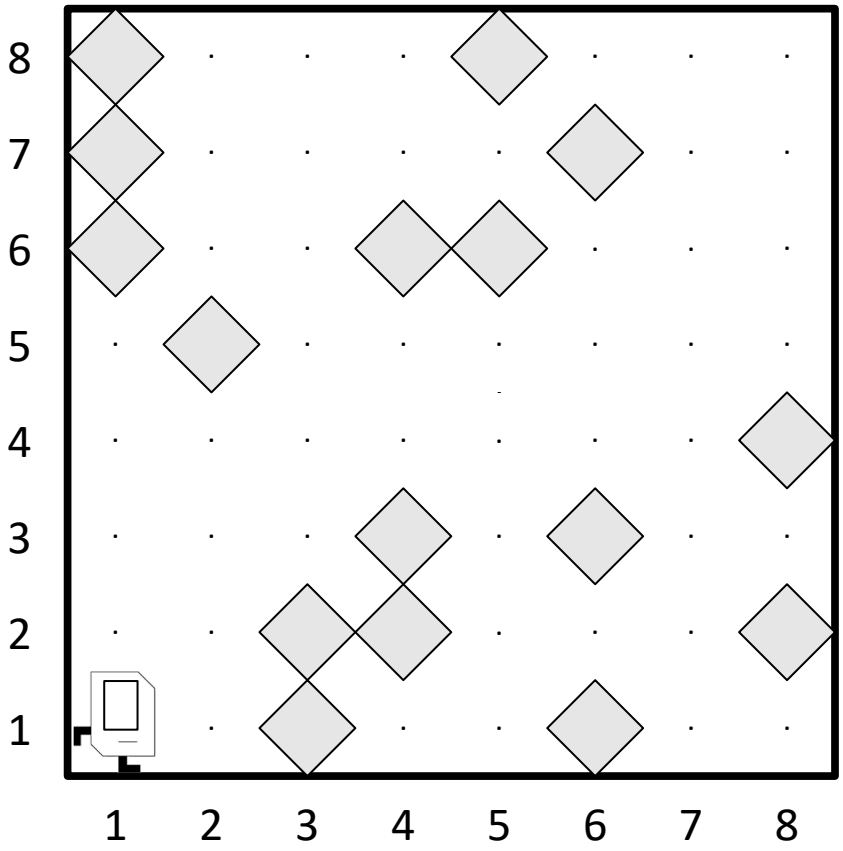
Plan For Today

- Announcements
- Recap: Control Flow
- Control Flow: If/else
- Decomposition
- Demo: HurdleJumper
- Practice: Debugging and Roomba



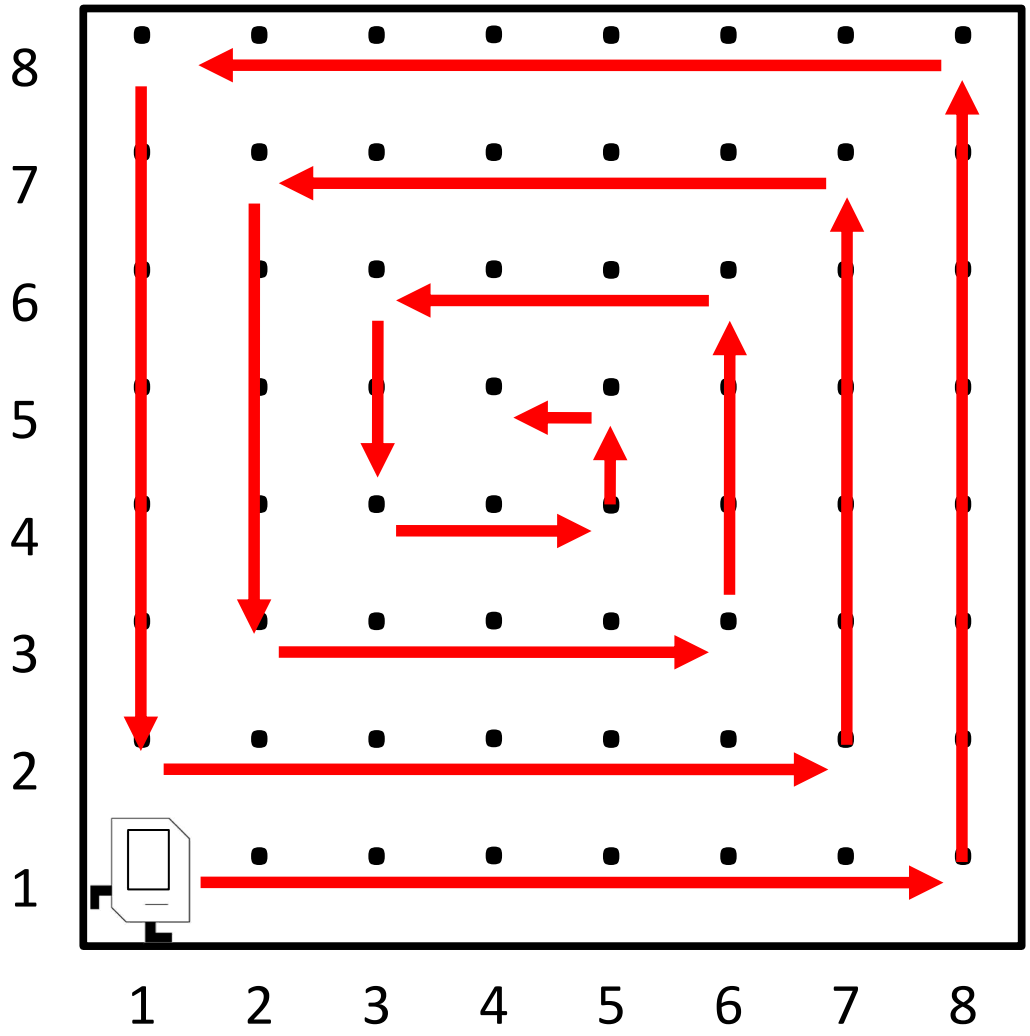
Roomba

- 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.

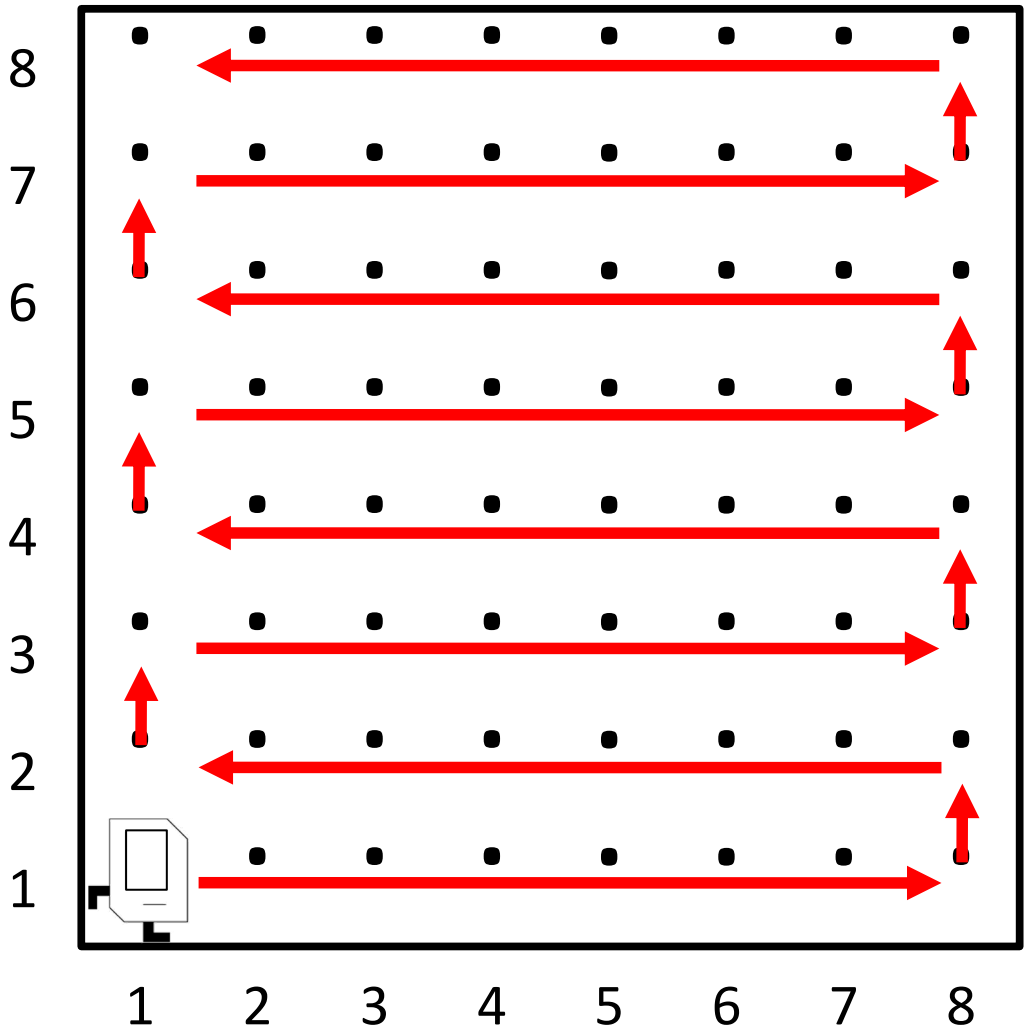


- How should we approach this tricky problem?

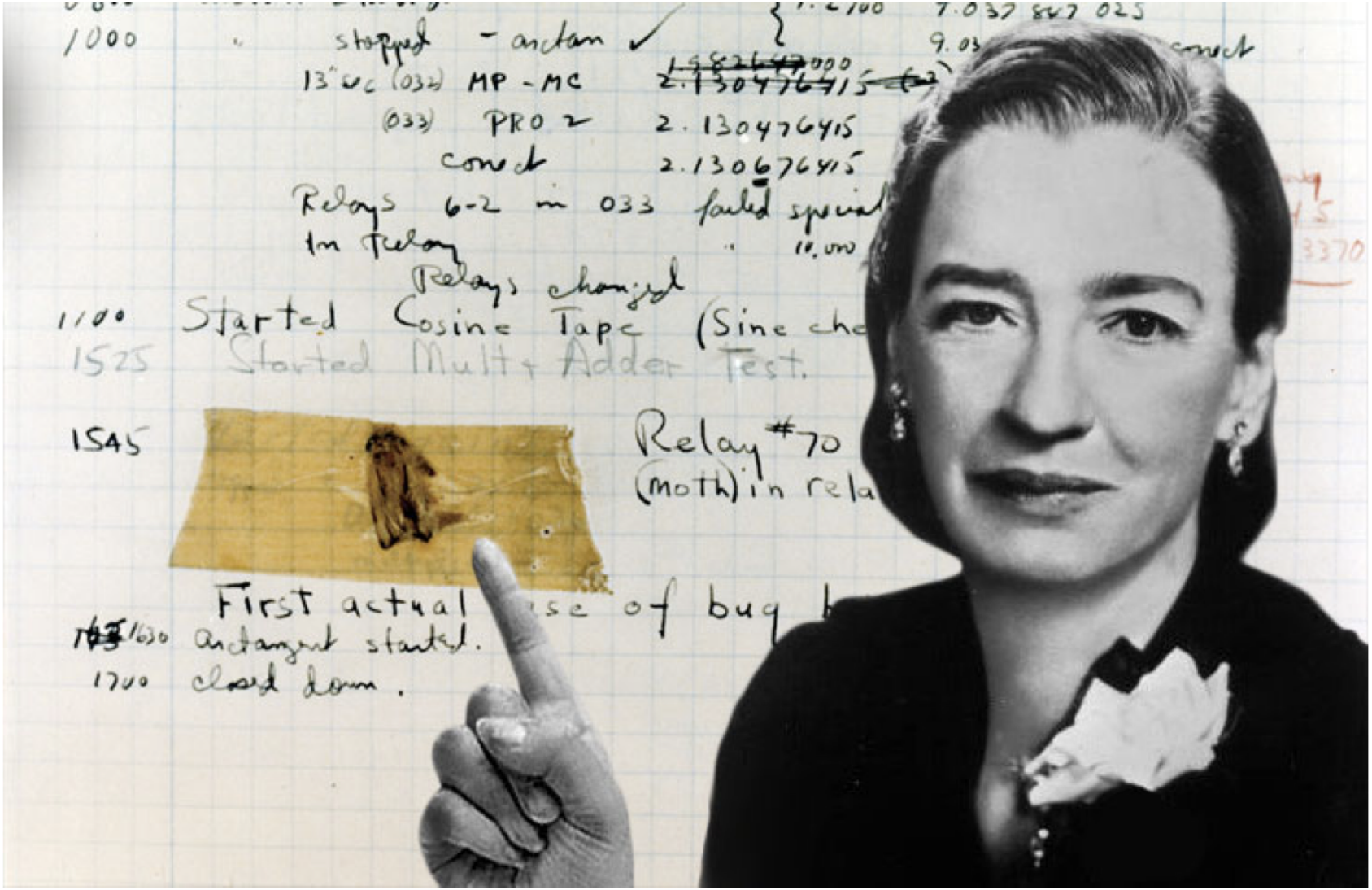
Possible algorithm 1



Possible algorithm 2

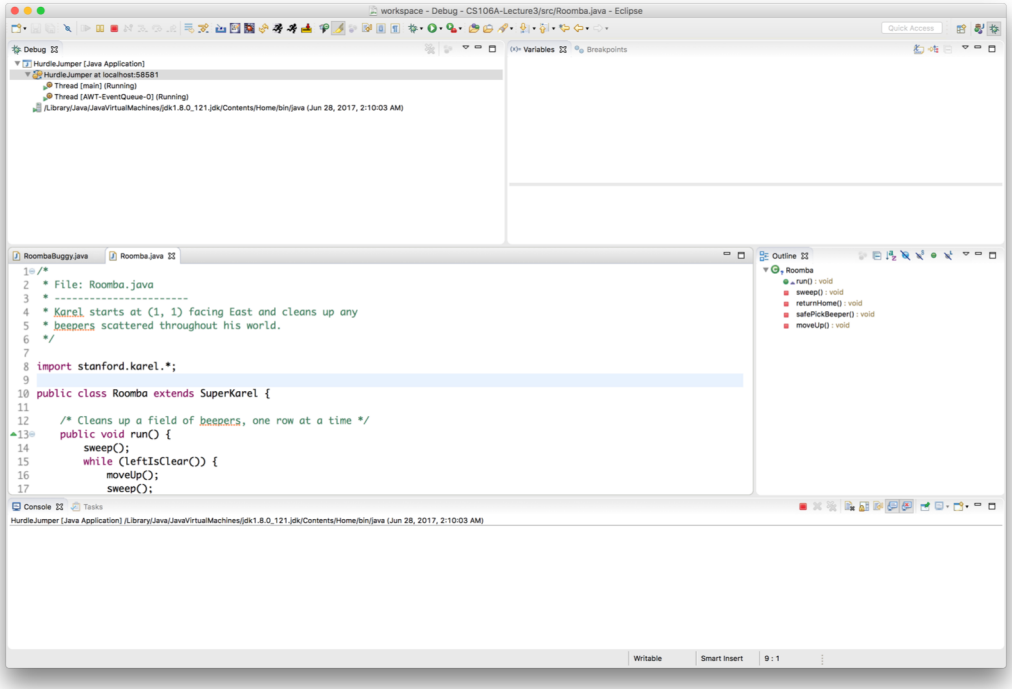


Debugging


1000
stopped - arctan
13"UC (032) MP - MC
(033) PRO 2
correct
2.13047 6415
2.13067 6415
Relays 6-2 in 033 failed special
In relay
Relays changed
1100 Started Cosine Tape (Sine che
1525 Started Mult + Adder Test.
1545
Relay #70
(moth) in rela
First actual
of bug
1700 closed down.

Debugging

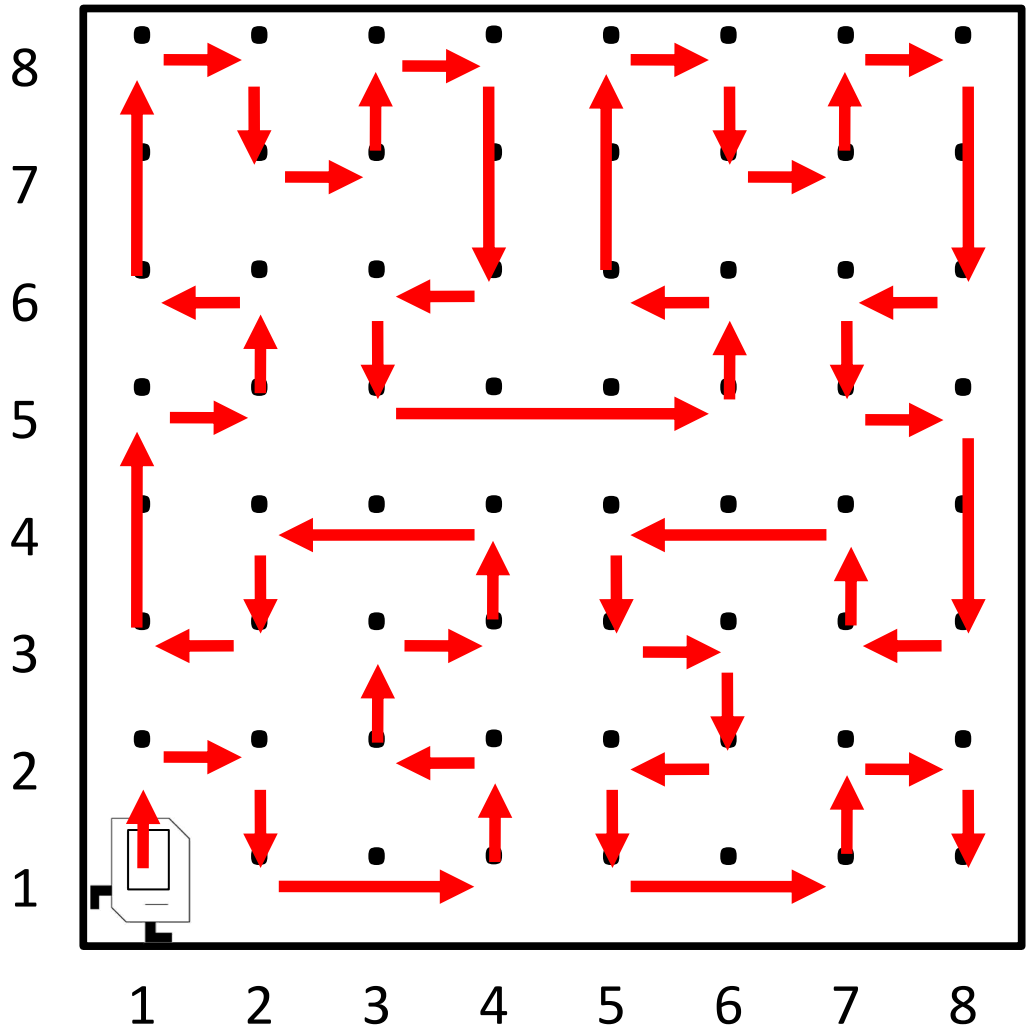
- Finding and fixing unintended behavior in your programs.
- Try to narrow down *where* in your code you think the bug is occurring. (i.e., what command or set of commands)
- We can use Eclipse to help us figure out what our program is doing.



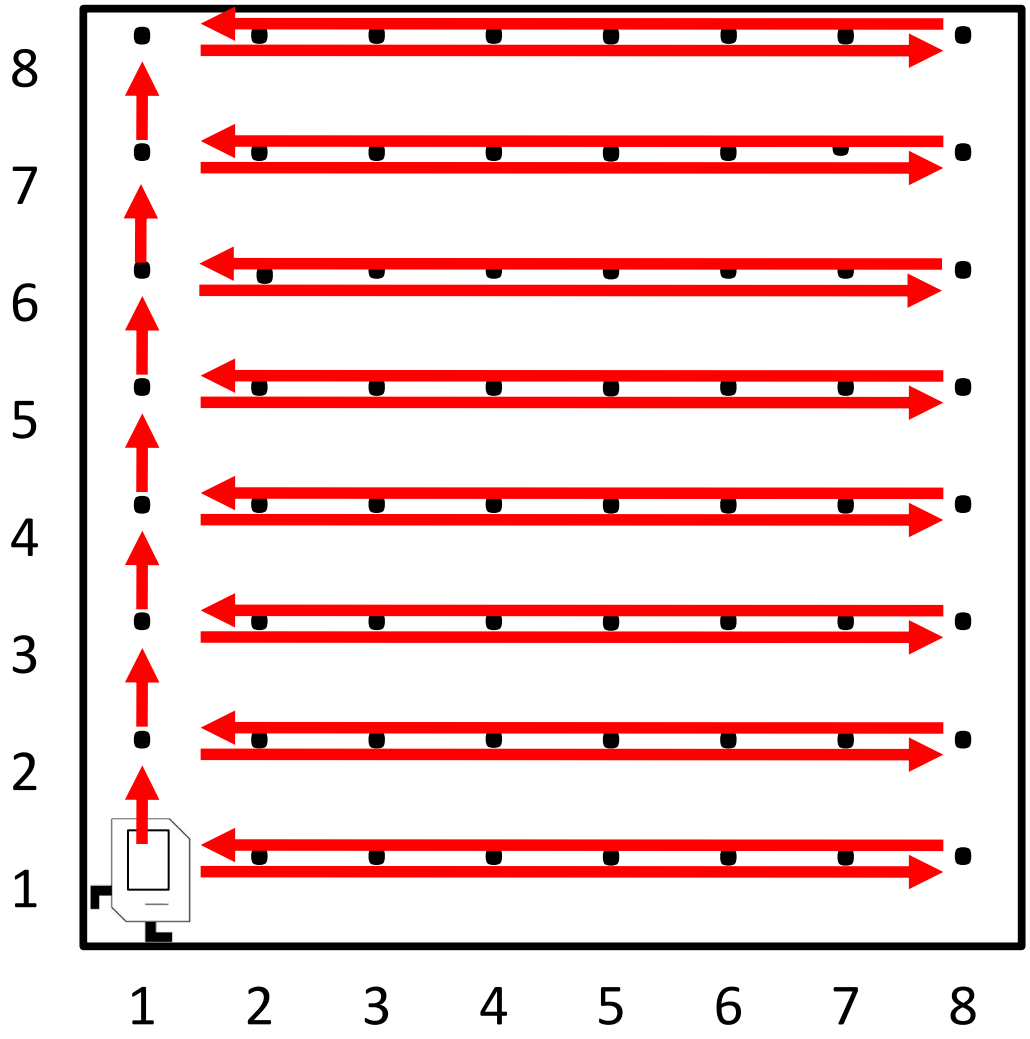
BuggyRoomba

Demo

Possible algorithm 3



Possible algorithm 4

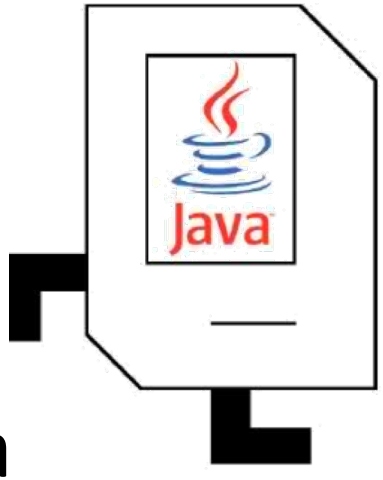


Practice: Roomba

Demo

Recap

- Announcements
- Recap: Control Flow
- Control Flow: If/else
- Decomposition
- Demo: HurdleJumper
- Practice: Debugging and Roomba



Next time: An introduction to Java