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

**Methods**

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

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

For example:

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

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

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

I want Karel to repeat some commands!

Know how many times

Don’t know how many times

for loop

while loop
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.

```java
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
• 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:

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

Runs the statements in the body once if `condition` is true.
If/Else Statements

You can also add an \textbf{else} statement:

\begin{verbatim}
if (condition) {
    statement;
    statement;
    statement;
    ...
} else {
    statement;
    statement;
    statement;
    ...
}
\end{verbatim}

Runs the first group of statements if \textit{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
private void turnToWall() {
    while (leftIsClear()) {
        turnLeft();
    }
}
private void turnToWall() {
    while (leftIsClear()) {
        turnLeft();
    }
}
private void turnToWall() {
    while(leftIsClear()) {
        turnLeft();
    }
}
Infinite Loops

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

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

```java
/*
 * 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
• 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.
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

Diagram showing a possible algorithm with a square grid and arrows indicating the path.
Possible algorithm 2
Debugging

1000
stopped - andan
13°uc (032) MP - MC
(033) PRO -
cover
1082350000
9.03

130476415
2.13076415
Relays 6-2 in 033 failed special
in 11,000

Relays changed

1100
Started cosine tape (Sine chord)

1525
Started multiplier and adder test.

1545
First actual use of bug

1650 engaged started.
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.
Demo
Possible algorithm 3
Possible algorithm 4
Demo
Recap

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

Next time: An introduction to Java