CS 106A, Lecture 3
Problem-solving with Karel

suggested reading:
Karel, Ch. 5-6
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
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”
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 \textit{max} times.
while (\texttt{condition}) \{ 
    \texttt{statement;}
    \texttt{statement;}
    \texttt{...}
\}

Repeats the statements in the body until \texttt{condition} is no longer true. Each time, Karel executes \textit{all statements}, and \texttt{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

for loop

Don’t know how many times

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

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

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

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

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

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

```java
private void turnToWall() {
    while (leftIsClear()) {
        if (frontIsBlocked()) {
            turnLeft();
        }
    }
}
```
// 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
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.
• **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.

```java
/*
 * 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
Demo
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.
Demo
Recap

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

Next time: An introduction to Java