SSEA Computer Science: CS106A

Dr. Cynthia Lee
Lecturer in Computer Science
Stanford
Goals for this Course

- Learn how to harness computing power to solve problems.
- To that end:
  - Explore fundamental techniques in computer programming
  - Develop good software engineering techniques
  - Gain familiarity with the Java programming language
Goals for today

- Learn how to harness computing power to solve problems.

- To that end:
  - Look at a very simple model of computation (Karel), and see how we build from these basic building blocks to surprisingly elaborate results.
  - Observe that in doing that building, we need to be organized in our design and encapsulate ideas into layers of units.
Meet Karel the Robot
Karel's World
Karel's World
Karel's World

Each row is called a street.
Karel's World
Each column is called an avenue.
The intersection of a street and an avenue is a corner.
Karel's World
Karel's World

Karel cannot move through walls.
Karel's World
Karel's World
Beepers mark locations in Karel's world.
Karel's World

Karel Commands
Karel's World

Karel Commands

move
Karel's World

Karel Commands

move
Karel's World

Karel Commands

move
Karel's World

Karel Commands

move
Karel's World

Karel Commands

move
Karel's World

Karel Commands

- move
- pickBeeper
Karel's World

Karel Commands

- move
- pickBeeper
Karel's World

Karel Commands

- move
- pickBeeper
- turnLeft
Karel's World

Karel Commands

- move
- pickBeeper
- turnLeft
Karel's World

Karel Commands
- move
- pickBeeper
- turnLeft
- putBeeper
Karel's World

Karel Commands

- move
- pickBeeper
- turnLeft
- putBeeper
Wait, why are we learning something with such limited rules?
Kenyan Women Create Their Own Geek Culture
import stanford.karel.*;

public class OurKarelProgram extends Karel {
    public void run() {
        move();
        pickBeeper();
        move();
        turnLeft();
        move();
        turnLeft();
        turnLeft();
        turnLeft();
        move();
        putBeeper();
        move();
    }
}

This piece of the program's source code is called a method.
import stanford.karel.*;

public class OurKarelProgram extends Karel {
    public void run() {
        move();
        pickBeeper();
        move();
        move();
        turnLeft();
        move();
        turnLeft();
        turnLeft();
        turnLeft();
        turnLeft();
        move();
        move();
        putBeeper();
        move();
    }
}

This line of code gives the **name** of the method (here, run)
import stanford.karel.*;

public class OurKarelProgram extends Karel {
    public void run() {
        move();
pickBeeper();
move();
move();
turnLeft();
move();
turnLeft();
turnLeft();
turnLeft();
move();
putBeeper();
move();
move();
    }
}

The inside of the method is called the \textit{body of the method} and tells Karel how to execute the method.
import stanford.karel.*;

public class OurKarelProgram extends Karel {
    public void run() {
        move();
        pickBeeper();
        move();
        turnLeft();
        move();
        turnLeft();
        turnLeft();
        turnLeft();
        move();
        putBeeper();
        move();
    }
}

This part of the program is called a class definition. You'll discuss classes later in CS106A.
import stanford.karel.*;

public class OurKarelProgram extends Karel {
    public void run() {
        move();
pickBeeper();
        move();
turnLeft();
        move();
turnLeft();
turnLeft();
turnLeft();
        move();
putBeeper();
        move();
    }
}

This is called an import statement. It tells Java what Karel is.
Improving our Program
The for loop
for (int i = 0; i < \textbf{N}; i++) {
    … statements to repeat \textbf{N} times …
}
The while loop
while \textit{(condition)} {
  ... \textit{statements to repeat when condition holds} ... 
}

Some of Karel's Conditions:

- frontIsClear()
- frontIsBlocked()
- beepersPresent()
- beepersInBag()
- facingNorth()
- facingSouth()

See the Karel reader (Page 18) for more details.
while (**condition**) {
    ... statements to repeat when **condition** holds ... 
}

Some of Karel's Conditions:

- `frontIsClear()`
- `frontIsBlocked()`
- `beepersPresent()`
- `beepersInBag()`
- `facingNorth()`
- `facingSouth()`

See the Karel reader (Page 18) for more details.
NOT SURE IF I SHOULD USE A FOR LOOP

OR A WHILE LOOP
Karel condition methods

- Karel has some commands that are not meant to be complete statements, but rather are used to ask questions:

<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 course reader.*
The while loop 
and the Fencepost Bug
Exercise: Sweeper

- Recall we wrote a "RoombaKarel" that picks up all beepers in front of Karel in a straight line.

- We started with a version with a slight bug...
A tricky bug

- Our code fails when there is a beeper at the final corner.
  - Changing the order of statements is likely to make it fail when there is a beeper at the *first* corner. It must work for both.

Karel has to take six steps...

...but has to check/sweep seven corners.
Fencepost problem

- **fencepost problem**: One with repeated statements, where some statements should be repeated $n$ times and some $n-1$ times.
- Like creating a fence with 5 posts and 4 wires between the posts.

Incorrect:
```
loop {
  place a post.
  place some wire.
}
```

Correct:
```
place a post.
loop {
  place some wire.
  place a post.
}
```
Fencepost loop

To solve a fencepost problem, perform the task that needs to happen \( n \) times (the "post") once outside the loop.

- If necessary, invert the steps inside the loop ("wire", then "post").

```java
public void run() {
    safePickup(); // post
    while (frontIsClear()) {
        move(); // wire
        safePickup(); // post
    }
}

public void safePickup() {
    if (beepersPresent()) {
        pickBeeper();
    }
}
```
Extra Topics
Private methods

- **private method**: An advanced concept. Public methods can be called by other classes/programs. Private ones cannot.
  - Not really relevant for our Karel programs, which always have 1 class.
  - The book examples always use private methods. (except run)
  - Your methods on HW1 can be either public or private. (Up to you)

```java
public void run() {
    safePickup();
    while (frontIsClear()) {
        move();
        safePickup();
    }
}

private void safePickup() {
    if (beepersPresent()) {
        pickBeeper();
    }
}
```
The SuperKarel class has an additional method `paintCorner` that sets Karel's current position to be a given color.

Valid colors are `BLACK`, `BLUE`, `CYAN`, `DARK_GRAY`, `GRAY`, `GREEN`, `LIGHT_GRAY`, `MAGENTA`, `ORANGE`, `PINK`, `RED`, `WHITE`, `YELLOW`, and `null` (no color)

```java
import stanford.karel.*;

public class RedStripe extends SuperKarel {
    public void run() {
        while (frontIsClear()) {
            paintCorner(RED);
            move();
        }
    }
}
```
The following code examples in the Karel reader may prove useful:

- Ch. 4: BeeperCollectingKarel; collects all beepers from towers of arbitrary heights
- Ch. 5: MazeRunningKarel, finds a beeper in an arbitrary maze by following the "right-hand rule"
- Ch. 5: DoubleBeepers, doubles the number of beepers in a corner (harder than it sounds!)