Programming Karel the Robot

Announcements

- Five Handouts Today:
  - Honor Code
  - Downloading Eclipse
  - Running Karel Programs in Eclipse
  - **Programming Assignment #1**
  - Submitting Programming Assignments
- Programming Assignment #1 Out:
  - Karel the Robot: Due Friday, January 17 at 3:15 PM
  - Email: Due Sunday, January 19 at 11:59PM

The CS106A Grading Scale

++

+

✓+

✓

✓-

-

--

0

Assignment Grading

- You will receive two scores: a functionality score and a style score.
- The **functionality score** is based on how well your program works.
  - Does it work correctly in the sample worlds?
  - Does it work correctly in custom test worlds?
- The **style score** is based on how well your program is written.
  - We'll cover elements of good style throughout this course.

Late Days

- Everyone has **two** free “late days” to use as you see fit.
- A “late day” is an automatic extension for one **class period** (Monday to Wednesday, Wednesday to Friday, or Friday to Monday). You do get extra time for national holidays.
- If you need an extension beyond late days, please talk to Vikas.

Section Signups

- Section signups open tomorrow at 5PM and close Sunday at 5PM.
- Sign up for section at

  http://cs198.stanford.edu/section

- Link available on the CS106A course website.

A Word on the Honor Code

Our Very First Karel Program Revisited

```
import stanford.karel.*;

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

```
import stanford.karel.*;

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

```
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();
      turnLeft();
      move();
      turnLeft();
      turnLeft();
      turnLeft();
      move();
      putBeeper();
      move();
   }
}
```

This line of code gives the **name** of the method (here, **run**)

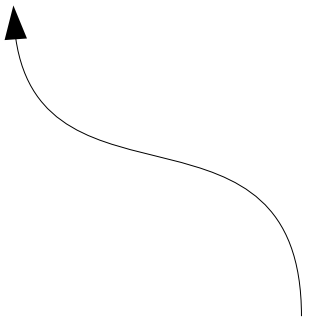
```
import stanford.karel.*;

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

The inside of the method is is called the **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**. We'll discuss classes later this quarter.

```
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**. Again, we will discuss this later in the quarter.

Improving our Program

The `for` loop

```
for (int i = 0; i < N; i++) {
   ... statements to repeat N times ...
}
```

The `while` loop

```
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.

```
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.