Methods

Creating more powerful robots

Methods

- Extending Robot's vocabulary
- Examples: creating new types
  - EightWalker, RightTurner
- Using methods to help solve problems.
  - Field of Beepers

The Basic Language is not enough

- To make a robot turn right, we have to execute 3 *turnLeft()* methods
- In the robot world to travel 8 squares you need 8 *move* statements.

The Solution

- Need to create useful methods and definitions
- Each definition is built from simpler ones that the robot already understands
- The simplest instructions are the primitive ones that the robot knows

Now we can solve problems using instructions natural to our way of thinking.

For Example…

- A new method, *turnRight*, can be defined as three *turnLeft* instructions
- A new *move8* method can be defined as eight *move* instructions
- So a robot can use simpler more natural instructions

The smaller program is much easier to read and understand. This is important!

How do we get a Robot that has new methods?

- We create a new class that *extends* the Robot class
- The new class inherits all of the methods and attributes of the Robot class, but then adds new methods of its own
- Then we use the new class to create instances of robots, which will be able to use the new methods

The EightWalker class and move8()

```
public class EightWalker extends Robot
{
    void move8()
    {
        move();
        move();
        move();
        move();
        move();
        move();
        move();
        move();
    }
}
```

The EightWalker class and move8()

```
public class EightWalker extends Robot
{
    void move8()
    {
        move();
        move();
        move();
        move();
        move();
        move();
        move();
        move();
    }
}
```

reserved words, public class, extends, void

Note the form of the method definition

```
public class EightWalker extends Robot
{
    void move8()
    {
        move();
        move();
        move();
        move();
        move();
        move();
        move();
        move();
    }
}
```

Method Header

braces

The instructions

Note the form of the method header

```
public class EightWalker extends Robot
{
    void move8()
    {
        move();
        move();
        move();
        move();
        move();
        move();
        move();
        move();
    }
}
```

Return type

parentheses

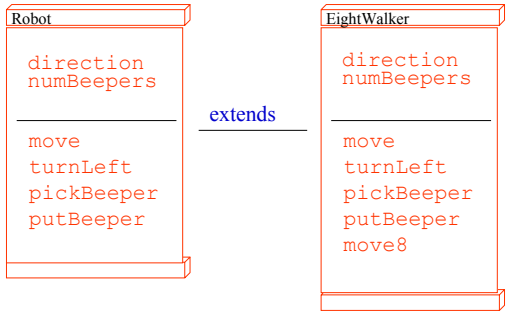
Method name

The Header is itself composed of three parts

The EightWalker class and move8()

```
public class EightWalker extends Robot
{
    void move8()
    {
        move();
        move();
        move();
        move();
        move();
        move();
        move();
        move();
    }
}
```

What Have We Got?



How to use the new method

- We can now make an instance of the EightWalker class, called steve:

```
EightWalker steve = new EightWalker();
```

- We get steve to execute the move8() method:

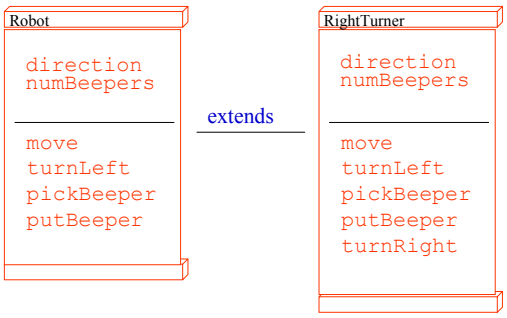
```
steve.move8();
```

- steve executes the method, which means that steve executes 8 “move()” methods

Finally, *turnRight*

```
public class RightTurner extends Robot
{
   void turnRight()
   {
      turnLeft();
      turnLeft();
      turnLeft();
   }
}
```

What Have We Got?



Meaning and Correctness

Robots do not understand what methods are *supposed* to accomplish; robots are quite literal:

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

This is a logic error.

Meaning and Correctness

- This definition of turnRight is perfectly legal, but obviously incorrect.
- The name says *what* the method is intended to do.
- The definition says *how* the method does what the name implies.
- The two should match or the programmer will have problems.

The new methods

- The methods are defined in the class definition.
- The order of method definitions is not important
- The definition of Robot is contained in the file Robot.class (which itself is in the file Robot.jar)

Examples

- These programs use methods
  - Climbing Stairs 
  - Harvest a field of Beepers 

When faced with a task, you should try and imagine useful methods that the robot could have

Summary

- A new class can extend another class and inherit all its methods
- A class can define its own methods
- You can use methods to break a problem into smaller chunks.
- It is quite common for programmers to start using a method even before they have written it.
- E.g. if you need the move8() instruction, go ahead and use it even before you write it! You can always write it later.