Lesson: Methods

The key to learning to program is:

Writing lots of code!

We will give you opportunities to write interesting programs.

If you have never written a program before today then you are in the right place.

This course requires absolutely no background in programming.

If you are an experienced Java programmer, you will learn how we teach Java programming.

Suppose you want to learn to play a new game.

What would you need to know?

the rules
the goal
controls
how many players

strategies

Which is easier: learning the rules, or the strategy?

rules are easier

We'll be learning to program in
the Java programming language.

You will need to know two kinds of rules to program
in Java:

* grammar rules (what can I type where)
* evaluation rules (what will this instruction do)

These are things you can memorize.

You will also need to learn to use Java to write programs to solve problems.

This is not something you can memorize.

Java has a complex set of rules.

We will start with a very tiny subset of Java, and expand gradually.

The grammar rules for an extremely small
and misleading subset of Java

```
INSTRUCTION:  Robot.move();
           :  Robot.turnLeft();
           :  Robot.makeDark();
           :  Robot.makeLight();
           :  Robot.load("MAPFILENAME");
```

(demo)

Each of these instructions changes the state of
the robot's world.

What state information does the computer need to remember?

where the robot is
which direction robot is facing
the map: where the walls are
colors of the squares

What is the evaluation rule for each instruction?

`Robot.move();`	advance 1 space
`Robot.turnLeft();`	rotate 90 degrees CCW
`Robot.makeDark();`	change current cell color to dark
`Robot.makeLight();`	change current cell color to light

What must be true before we execute these instructions?

`Robot.move();`	front must be clear
`Robot.turnLeft();`	(no precondition)
`Robot.makeDark();`	must be on light
`Robot.makeLight();`	must be on dark

Our first program: solving the maze

```
public class Lesson
{
    public static void run()
    {
        Robot.load("maze.txt");
        Robot.move();
        Robot.move();
        ...
    }
}
```

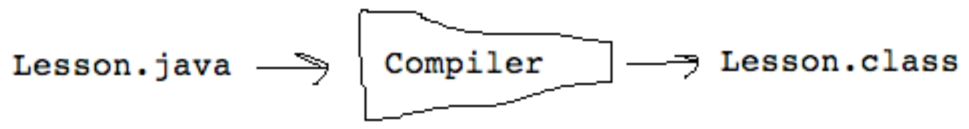
More Grammar Rules

```
FILE:  public class JAVAFILENAME
       {
         public static void run()
         {
           INSTRUCTION
           INSTRUCTION
           ...
           INSTRUCTION
         }
       }

INSTRUCTION:  Robot.move();
           :  Robot.turnLeft();
           :  Robot.makeDark();
           :  Robot.makeLight();
           :  Robot.load("MAPFILENAME");
```

Compiling

The COMPILE button to translates our human-readable program into an equalivalent computer-readable program.



```
public class Lesson
{
    ...
}
```

```
010010000100
101000001001
100111011001
101100011110
...
```

```
Robot.move();
Robot.move();
Robot.move();
Robot.move();
Robot.turnLeft();
Robot.move();
Robot.move();
Robot.turnLeft();
Robot.turnLeft();
Robot.turnLeft();
Robot.move();
Robot.move();
Robot.turnLeft();
Robot.turnLeft();
Robot.turnLeft();
Robot.move();
Robot.move();
...
```

We don't like this program because:

tedious
hard to read
easy to lose track
hard to find mistakes

Any time I copy/paste, alarms should go off in my head.

There must be a better way!

Methods

move, turnLeft, makeDark, makeLight, and load
are all names of methods.

When we execute the instruction

```
Robot.move();
```

we are "calling" the move method.

(often called "functions" in other languages)

What sequence of instructions did I find myself writing over and over again to solve the maze?

```
Robot.turnLeft();
Robot.turnLeft();
Robot.turnLeft();
```

What would be a good name for this sequence of instructions?

```
turnRight
```

Let's tell Java what turnRight means, by defining our own turnRight method.

```
public static void turnRight()
{
    Robot.turnLeft();
    Robot.turnLeft();
    Robot.turnLeft();
}
```

How do we call the turnRight method?

$$\underbrace{\texttt{Lesson}}_{\text{file name}}\texttt{.}\underbrace{\texttt{turnRight}}_{\text{method name}}\texttt{();}$$

In general:

$$\underset{\text{file name}}{\underline{\qquad\qquad}}\ .\ \underset{\text{method name}}{\underline{\qquad\qquad\quad}}\ \texttt{();}$$

This is one of those things you can memorize!

Define turnAround: (DrJava)

```
public static void turnAround()
{
    Robot.turnLeft();
    Robot.turnLeft();
}
```

Whenever you write a method, write before/after comments above it. (ignored by the compiler)

```
//before: back must be clear
//after:  moved 1 space back, facing original direction
public static void backUp()
{
    Lesson.turnAround();
    Robot.move();
    Lesson.turnAround();
}
```

The BEFORE comment tells us what must be true before we call this method (what the method assumes)

The AFTER comment tells us what will be true after the method returns (what the method does)

```
Java Grammar Rules

FILE:  public class FILENAME
       {
         METHOD-DECLARATION
         METHOD-DECLARATION
         ...
         METHOD-DECLARATION
       }

METHOD-DECLARATION:  public static void METHODNAME ()
                     {
                       STATEMENT
                       STATEMENT
                       ...
                       STATEMENT
                     }

STATEMENT:  FILENAME . METHODNAME ();
```

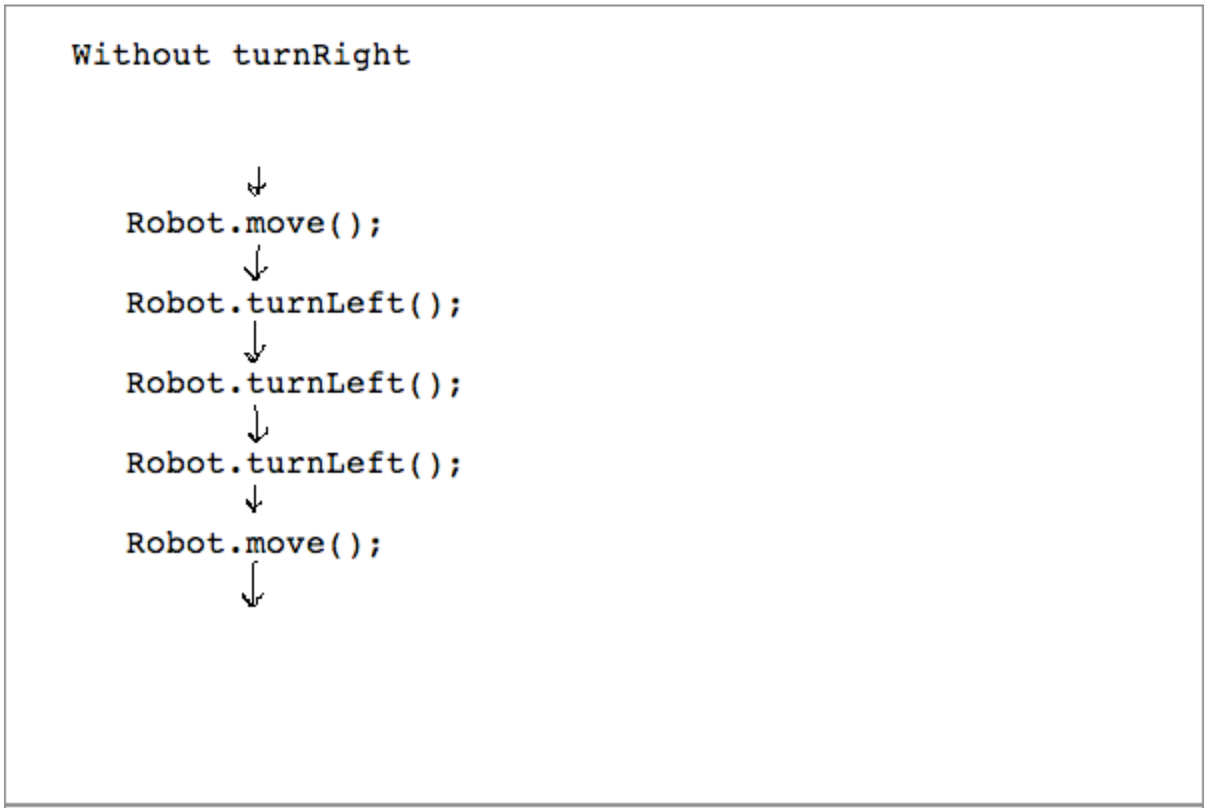
Grammar rule for method calls:

```
   _________ . ___________ ();
   file name   method name
```

Evaluation rule for method calls:

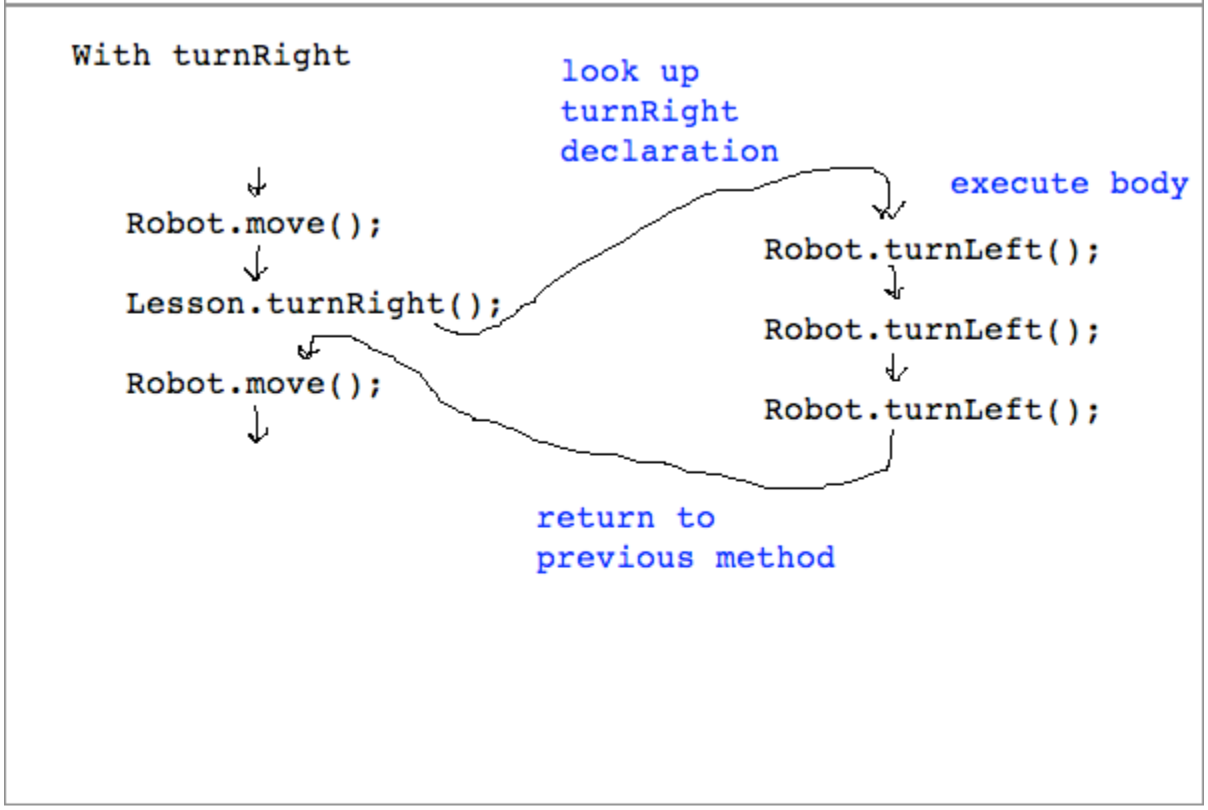
look up method declaration in specified file
execute body of method

Without turnRight



```
Robot.move();
Robot.turnLeft();
Robot.turnLeft();
Robot.turnLeft();
Robot.move();
```

With turnRight



```
Robot.move();
Lesson.turnRight();
Robot.move();
```

```
Robot.turnLeft();
Robot.turnLeft();
Robot.turnLeft();
```

Do methods make more work for us, or less work?

less work!

We can define them once and then re-use.
They help us break down complex problems.

Do methods make more work for the computer, or less work?

more work

Any tradeoff that makes

less work for us
and

more work for the computer

is usually an excellent tradeoff!

The stair-cleaning problem. Robot is at the bottom of
the stairs, and must clean dark spot on each stair.

```
   :
  :X
 :XX
NXXX
```

(N=robot facing north, X=wall, and :=dark)

The key to solving problems in this course:

DON'T THINK IN JAVA.

Think about how you would solve the problem.

Identify high-level tasks,
then break those down into smaller tasks.

Think about how you would explain your solution.

```
I need to clean the stairs,
which involves cleaning one stair 3 times,
which means ...

       public static void cleanStairs()
       {
           Lesson.cleanStair();
           Lesson.cleanStair();
           Lesson.cleanStair();
       }
```

```
//before:  robot is facing north,
//         with no wall in front,
//         and with a dark cell to the northeast
//after:   cell is now light
//         robot is in that cell,
//         facing north
public static void cleanStair()
{
    Robot.move();
    Lesson.turnRight();
    Robot.move();
    Robot.makeLight();
    Robot.turnLeft();
}
```

(see Errors page on web site)

Compile-Time Errors (Syntax Errors)

compiler checks if you violate a grammatical rule

Run-Time Errors (Crashes)

your program crashes if you
violate an evaluation rule

Logical Errors

your program runs without crashing,
but it doesn't do what you wanted

About Errors:

1. Don't take errors personally.

2. Don't attempt to fix the code
until you understand the error message.

You should never be guessing
how to fix your code in this course because

* if you guess right, you won't know why it works

* if you guess wrong, you'll have dug yourself
into a deeper hole

By the way, there is one more useful instruction:

```
Robot.setDelay(___________________________);
               number of seconds to pause
```

For example:

```
Robot.setDelay(0.1);
```

will run 10 operations per second.

The rectangle clearing problem.
Robot is facing the bottom of a 6x5 rectangle of dark cells.

```
  :::::
  :::::           E = robot facing east
  :::::           : = dark cell
  :::::
  :::::
 E:::::
```

```
 //before:  robot facing bottom row to the east
 //after:   rectangle has been cleared
 public static void clearRectangle()
 {
     Lesson.clear2Rows();
     Lesson.clear2Rows();
     Lesson.clear2Rows();
 }

 //before:  robot facing row to the east
 //after:   robot is 2 spaces north of start,
 //         and 2 rows have been cleared
 public static void clear2Rows()
 {
     Lesson.clearRow();
     Lesson.loopLeft();
     Lesson.clearRow();
     Lesson.loopRight();
 }
```

```
//before:  robot is facing row of 5 dark squares
//after:   robot is on last of those squares,
//         and all are light now
public static void clearRow()
{
    Lesson.clearNextSquare();
    Lesson.clearNextSquare();
    Lesson.clearNextSquare();
    Lesson.clearNextSquare();
    Lesson.clearNextSquare();
    Lesson.clearNextSquare();
}

//before:  robot facing dark square
//after:   robot on that square, and it is light now
public static void clearNextSquare()
{
    Robot.move();
    Robot.makeLight();
}
```

```
//before:  robot facing east, right below east edge of row
//after:   robot facing row of dark cells to the west
public static void loopLeft()
{
    Robot.move();
    Robot.turnLeft();
    Robot.move();
    Robot.turnLeft();
}

//before:  robot facing west, right below west edge of row
//after:   robot facing row of dark cells to the east
public static void loopRight()
{
    Robot.move();
    Lesson.turnRight();
    Robot.move();
    Lesson.turnRight();
}
```