

Lesson: Ifs

Grammar rule for method calls:

```
filename . methodname ();
```

Evaluation rule for method calls:

looks in file for that method declaration

runs code inside that method

## 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 ();
```

---

The key to solving problems in this course:

DON'T THINK IN Java!

The key to solving a complex problem is to

1. break a complex problem into bite-sized pieces,
2. give names to those pieces, and
3. think in terms of those names.

In computer science,  
this is our most important  
problem-solving strategy!

For example, when we wrote

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

we were giving a name ("turnRight") to a sequence of instructions, and now we can think in terms of that name.

This idea of naming the pieces of a problem  
and then thinking in terms of those pieces  
is SO important in computer science  
that computer scientists have even given a name  
to this idea!

It's called ABSTRACTION.

When we name something,  
we say we're "defining an abstraction."

## STATEMENTS

We find statements in the body of a method.  
So far, all our statements are method calls, such as:

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

A statement is a command.  
It tells the computer to do something.

## STATEMENTS

We execute a statement to change the computer's state  
(e.g. which way the robot is facing).

We say that the robot's position or direction changes  
as a "side effect" of executing the statement.

clearRectangle, revisited ...

Our code can clear  
this rectangle:

```
:::::  
:::::  
:::::  
:::::  
:::::  
:::::  
:::::
```

But not this rectangle:

```
:::::  
 : :  
 : :  
:::::  
 : :  
 : :  
 : :
```

But we'd like to write programs that work in  
a variety of situations, instead of hard-wiring  
them to handle one particular situation.

What conditions do you wish you could test for?

whether it's on light or dark

clear in front or blocked in front

Testing robot conditions:

```
Robot.onDark()
```

```
Robot.frontIsClear()
```

Each condition returns true/false

## EXPRESSIONS

Some expressions:

```
Robot.onDark()
```

```
Robot.frontIsClear()
```

(Notice: no semicolons)

An expression is something that can give us a value.

The computer evaluates an expression to find its value.

## VALUES

We only know 2 values so far:

```
true  
false
```

These are called "boolean" values  
(as opposed to numbers, words, etc.)

In DrJava's interactions pane,

you can execute a statement

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

or you can evaluate an expression

```
Robot.onDark()
```

How can we use expressions in our program?



Introducing the "if" statement:

```
if (Robot.frontIsClear())
{
    Robot.move();
}
else
{
    Robot.turnLeft();
}
```

In general, the grammatical rule is:

```
if ( _____ )
    boolean expression
{
    STATEMENT
    STATEMENT
    ...
    STATEMENT
}
else
{
    STATEMENT
    STATEMENT
    ...
    STATEMENT
}
```

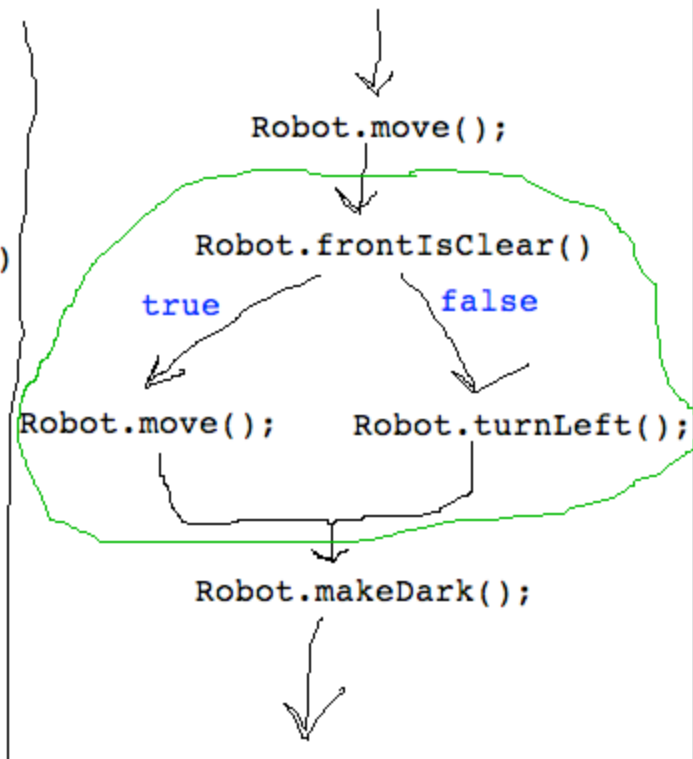
The evaluation rule for ifs:

1. Find the value of the boolean expression.
2. If the value is true, execute the first block of statements. Otherwise, if the value is false, execute the second block of statements.

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

```
if (Robot.frontIsClear())  
{  
    Robot.move();  
}  
else  
{  
    Robot.turnLeft();  
}
```

```
Robot.makeDark();
```



Now we have 2 kinds of statements:

```
FILENAME . METHODNAME ();
```

```
if (BOOLEXP)
{
    STATEMENTS
}
else
{
    STATEMENTS
}
```

```
//before: clear in front
//after:  robot moved one square, which is now light
public static void clearNextSquare()
{
    Robot.move();
    if (Robot.onDark())
    {
        Robot.makeLight();
    }
    else
    {
    }
}
```

We wrote this to fix clearRectangle  
to handle squares that were already  
light.

```
//before: clear in front
//after:  robot moved one square, which is now light
public static void clearNextSquare()
{
    Robot.move();
    if (Robot.onDark())
    {
        Robot.makeLight();
    }
}
```

We can omit the "else" part if there's nothing to do when the condition is false.

When you decide to write an "if" in your code, always write out:

```
if (    )
{
}
else
{
}
```


This way, you won't forget to decide what your program should do if the condition is false.

If you decide there's nothing for the else to do, then go ahead and remove it.

From now on, when I ask you to write a program to solve a problem, your program will need to handle a variety of possible cases.

I will often show you a specific example, but your code should handle any such situation.

```
//turn around if there's a wall in front
public static void turnAroundIfWall()
{
    if (Robot.frontIsClear())
    {
    }
    else
    {
        Lesson.turnAround();
    }
}
if ( ! Robot.frontIsClear())
{
    Lesson.turnAround();
}
```



kind of ugly

```
//before turn around if there's a wall in front
public static void turnAroundIfWall()
{
    if ( ! Robot.frontIsClear())
    {
        Lesson.turnAround();
    }
    else
    {
    }
}
```

Note the exclamation point.

This is the NOT operator.

Read as "if not robot front is clear"

```
//before turn around if there's a wall in front
public static void turnAroundIfWall()
{
    if ( ! Robot.frontIsClear())
    {
        Lesson.turnAround();
    }
}
```

Since the else did nothing, we can remove it.

The NOT operator: !

If the value of  
    Robot.onDark()

is true,

then the value of

    !Robot.onDark()

is false.

If the value of

    Robot.onDark()

is false,

then the value of

    !Robot.onDark()

is true.

Now we have 3 kinds of boolean expressions:

```
BOOLEXP:  Robot.onDark()  
          :  Robot.frontIsClear()  
          :  ! BOOLEXP
```

### Simplifying ifs...

```
if (Robot.onDark())
{
    Robot.move();
}
else
{
    Robot.makeDark();
    Robot.move();
}
```

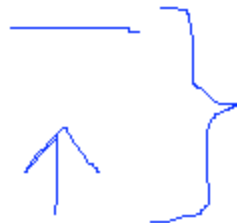
```
if ( ! Robot.onDark())
{
    Robot.makeDark();
}
Robot.move();
```

Do these two code segments behave the same? **No!**

```
if (Robot.frontIsClear())
{
    Robot.move();
}
```

```
if ( ! Robot.frontIsClear())
{
    Robot.turnLeft();
}
```

```
if (Robot.frontIsClear())
{
    Robot.move();
}
else
{
    Robot.turnLeft();
}
```



If robot is one space away from a wall, the code on the right will just advance one space, but the code on the left will advance one space and then turn left.



Very important:

Think about whether you're testing for

2 alternatives (if/else) or

2 independent conditions (if/if)

You're less likely to make this mistake  
if you always start by writing an "else".

true and false are also expressions!

We can evaluate them in DrJava's interactions pane.

What do you think the value of true is?

true

What does this code do?

```
if (true)
{
    Robot.move();           moves
}
```

What does this code do?

```
if (false)
{
    Robot.move();           does nothing
}
```

The carpetRooms problem:

```
 1 2 3 4 5 6 7 8
X.X...X.X...X.X.X
E.....
```

8 possible "rooms". Want to "carpet" (darken) each room with 2 walls: X.X

```
 1 2 3 4 5 6 7 8
X:X...X:X...X:X:X
.....E
```

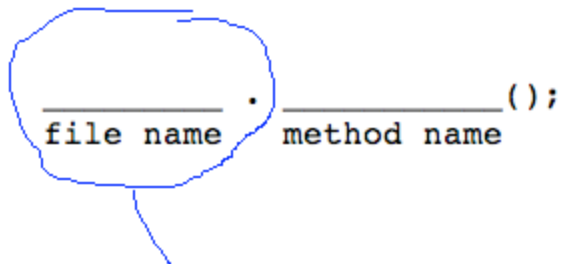
(Rooms 1, 4, 7, and 8 have been carpeted.)

```
public static void carpetRooms()  
{  
    carpetRoom();  
    carpetRoom();  
    carpetRoom();  
    carpetRoom();  
    carpetRoom();  
    carpetRoom();  
    carpetRoom();  
    carpetRoom();  
}
```

What do you notice about this method?

no file names

A Shortcut:



```
_____ . _____();  
file name  method name
```

Can omit if method is in this file.

Must still think about which file the method is in,  
before deciding to leave off the file name.

```
//before: below left edge of possible room, facing east.
//after:  below right edge of possible room, facing east.
//        if room, center square has been darkened.
public static void carpetRoom()
{
    Robot.move(); Robot.turnLeft();
    Robot.move(); Robot.turnLeft();
    if (Robot.frontIsClear())
    { Robot.turnLeft(); }
    else
    {
        turnAround();
        if (!Robot.frontIsClear()) { Robot.makeDark(); }
        turnRight();
    }
    Robot.move(); Robot.turnLeft(); Robot.move();
}
```

We were thinking in Java!

How would we have thought about it in English?

1. enter room
2. if surrounded by walls, make dark
3. exit room

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

```
public static void exitRoom()
{
    backUp();
    turnRight();
}
```

```
public static void carpetRoom()
{
    enterRoom();

    if ( surrounded by walls )
    {
        Robot.makeDark();
    }

    exitRoom();
}
```

How do we do this?

Boolean Methods (methods that return true/false)

Only 2 boolean methods are provided in Robot.java:

```
onDark  
frontIsClear
```

But we can make our own!

Let's define frontIsBlocked in DrJava ...

```
public static boolean frontIsBlocked()  
{  
    if (Robot.frontIsClear())  
    {  
        return false;  
    }  
    else  
    {  
        return true;  
    }  
}
```

The return statement:

Grammatical Rule:

```
return _____;  
    boolean expression
```

Evaluation Rule:

1. Find value of boolean expression.
2. Return that value IMMEDIATELY.

"void" means "doesn't return a value"

You call a void method only for its side effects.

You call a boolean method for its return value.

A good boolean method should not have side effects.

## Java Grammar Rules

FILE:     public class FILENAME { METHODS }

METHOD: public static TYPE METHODNAME () { STMTs }

TYPE:     void  
          :     boolean

STMT:     FILENAME . METHODNAME ();     *call to void method*  
          :     if ( BOOLEXP ) { STMTs } else { STMTs }  
          :     return BOOLEXP ;     *only in boolean methods*

Even prettier:

```
public static boolean frontIsBlocked()
{
    return !Robot.frontIsClear();
}
```



## Naming methods:

Pick command/action verbs for void methods.

"move" "turnLeft"

Pick conditions for boolean methods,  
that sound like they could be true or false.

"onDark" "frontIsClear"

```
public static boolean leftIsBlocked()
{
    Robot.turnLeft();
    if (Robot.frontIsClear())
    {
        turnRight();
        return false;
    }
    else
    {
        turnRight();
        return true;
    }
}
```

should turn back to original direction before returning, to avoid surprising side effects