Today

- Making decisions in Java
  - If statements
  - If/Else (“Either-or”) statements
  - Logical NOT operator
  -

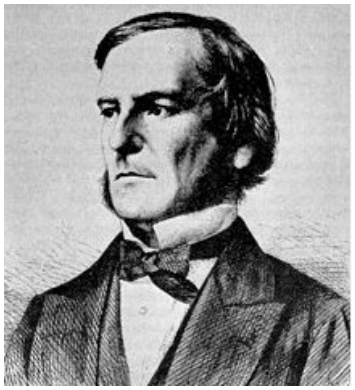
Two Kinds of Decisions

So far, our programs have been composed of a sequence of statements executed in order.

The **if** and **while** statements are different. *As the program is running, they can ask a question*. Based on the answer, they choose the next statement or group of statements to execute.

In a robot program, the question asked might be, "Is the robot's front blocked by a wall?" or "Is there something on this intersection the robot can pick up?"

All of these questions have "yes" or "no" answers. In fact, ***if*** *and* ***while*** *statements can only ask* ***yes*** *and* ***no*** *questions.* Java uses the *keyword* **true** for "yes" and **false** for "no." These keywords represent **Boolean** values, just like the numbers 0 and 23 represent *integer* values.



George Boole (2 November 1815 – 8 December 1864) was an English-born mathematician, philosopher and logician. His work was in the fields of differential equations and algebraic logic, and he is now best known as the author of *The Laws of Thought*. As the inventor of the prototype of what is now called **Boolean logic**, which became the basis of the modern digital computer, Boole is regarded in hindsight as a founder of the field of computer science.

Built-In Queries, Predicates

The **Robot** class has several built-in **queries** that answer questions like

- Can I pick a **Thing** up from this Intersection? boolean **canPickThing()**
- How many **Things** are in my backpack? int **countThingsInBackpack()**
- What am I called (what **string of characters** is labeling me)? String **getLabel()**
- What **Avenue** am I on? int **getAvenue()**
- What is my **speed**? int **getSpeed()**
- What **Street** am I on? int **getStreet()**

Questions with *Boolean* **True** or **False** answers, like **canPickThing**, are called ***predicates***.

Negating a predicate gives it the *opposite* meaning.

Logical Negation Operator

The **Robot** class does not provide a predicate for testing if the **Robot** cannot pick up a **Thing**.

Fortunately, any **Boolean** expression may be negated, or given the opposite value, by using the logical negation operator " **!** ". In English, this is usually written and pronounced as "**not**".

! means "not"

```
if (!karel.canPickThing())
{ karel.putThing();
}
```

Logical Negation Operator

Testing Integer Queries

The **if** and **while** statements always ask **True** or **False** questions.

"*Should I execute the code, true or false?*"

This approach works well for queries that return a **boolean** value, but how can we use queries that return **integers**?

We do it with

comparison operators

```
if (karel.getStreet() == 1)
{  // what to do if karel is on 1st street
}
```

```
while (karel.countThingsInBackpack() < 8)
{ karel.pickThing();
}
```

Operator	Name	Example	Meaning
<	less than	`karel.getAvenue() < 5`	Evaluates to `true` if `karel`'s current avenue is strictly less than 5; otherwise, evaluates to `false`.
<=	less than or equal	`karel.getStreet() <= 3`	Evaluates to `true` if `karel`'s current street is less than or equal to 3; otherwise, evaluates to `false`.
==	equal	`karel.getStreet() == 1`	Evaluates to `true` if `karel` is currently on 1st Street; otherwise, evaluates to `false`.
!=	not equal	`karel.getStreet() != 1`	Evaluates to `true` if `karel` is not currently on 1st Street; if the robot *is* on 1st Street, evaluates to `false`.
>=	greater than or equal	`5 >= karel.getAvenue()`	Evaluates to `true` if 5 is greater than or equal to `karel`'s current avenue. Most people find this easier to understand when written as `karel.getAvenue() <= 5`.
>	greater than	`karel.getAvenue() > 5`	Evaluates to `true` if `karel`'s current street is strictly greater than 5; otherwise, evaluates to `false`.

Two-Sided Queries

The examples in the **comparison operator** table show an integer on only one side, but Java is more flexible than this: it can have a **query** on both sides of the **operator**, as in the following statements --

```
if (karel.getAvenue() == karel.getStreet())
{ ...
}
```

This test determines whether **karel** is on the diagonal line of intersections (0,0), (1,1), (2,2), and so on. It also includes intersections with negative numbers such as (-3, -3).

The following code tests whether the robot's **Avenue** is five more than the **Street**. Locations where this tests **true** include (0,5) and (1,6).

```
if (karel.getAvenue() == karel.getStreet() + 5)
{ ...
}
```

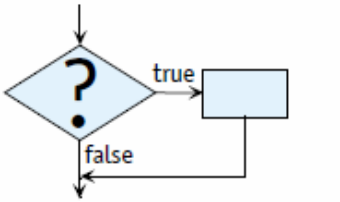
If and **While**

When the simplest form of an **if** statement asks a question and the answer is *true*, it executes a group of statements once and then continues with the rest of the program. If the answer to the question is *false*, that group of statements is not executed.

When a **while** statement asks a question and the answer is *true*, it executes a group of statements (just like the *if* statement). However, instead of continuing down to the rest of the program, the while statement asks the question again. If the answer is *still true*, that same group of statements is executed again. This continues until the answer to the question is *false*.

The ***if*** statement's question is **"Should I execute this group of statements once?"**

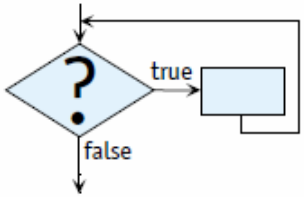
if (*test statement*) {
// list of statements
}



Flowchart for the `if` statement

The ***while*** statement's question is **"Should I execute this group of statements again**

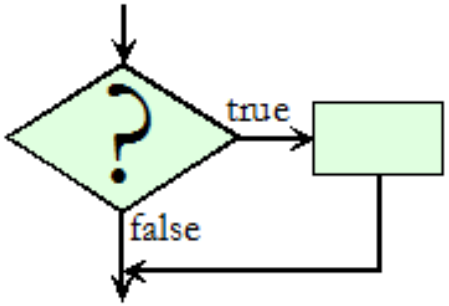
while (*test statement*) {
// list of statements
}
.



Flowchart for the `while` statement

Example: **If** Statement

ASKS THE QUESTION: Should this statement or group of statements be executed once or not at all?




```
if (karel.frontIsClear())
{ karel.move();
}
karel.turnLeft();
```

```
if (karel.frontIsClear())
{ karel.move();
}
karel.turnLeft();
```

```
if (karel.frontIsClear())
{ karel.move();
}
karel.turnLeft();
```

Once

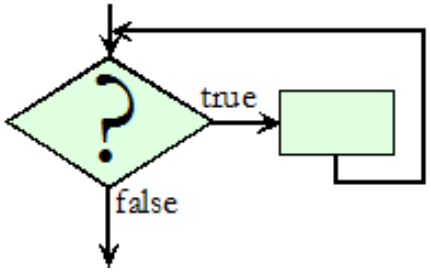
```
if (karel.frontIsClear())
{ karel.move();
}
karel.turnLeft();
```

```
if (karel.frontIsClear())
{ karel.move();
}
karel.turnLeft();
```

Not At All

MsRobotoIf.java

Example: **While** Statement



ASKS THE QUESTION: Should this statement or group of statements be executed again?

```
while (karel.frontIsClear())
{ karel.move();
}
karel.turnLeft();
```

```
while (karel.frontIsClear())
{ karel.move();
}
karel.turnLeft();
```

```
while (karel.frontIsClear())
{ karel.move();
}
karel.turnLeft();
```

```
while (karel.frontIsClear())
{ karel.move();
}
karel.turnLeft();
```

Yes

```
while (karel.frontIsClear())
{ karel.move();
}
karel.turnLeft();
```

```
while (karel.frontIsClear())
{ karel.move();
}
karel.turnLeft();
```

```
while (karel.frontIsClear())
{ karel.move();
}
karel.turnLeft();
```

No

MsRobotWhile.java

The General Form of an **if** Statement:

```
if («test»)
{ «list of statements»
}
```

Examples:

```
if (karel.canPickThing())
{ karel.pickThing();
  karel.turnLeft();
}

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

The General Form of a **while** Statement:

```
while («test»)
{ «list of statements»
}
```

Examples:

```
while (karel.canPickThing())
{ karel.pickThing();
  karel.turnLeft();
}

while (this.frontIsClear())
{ this.move();
}
```

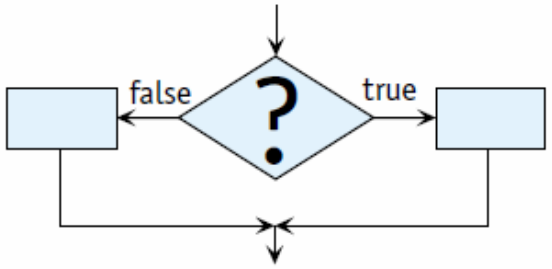
The **If-Else** Statement

The **if** statement performs an action once or not at all. Another version of the **if** statement, the **if-else** statement, choose between two groups of actions. It performs one or it performs the other based on a test.

Unlike the **if** statement, the **if-else** statement always performs an action. The question is, which action?

The general form of the **if-else** is as follows:

```
if («test»)
{ «statementList1»
} else
{ «statementList2»
}
```



MsRobotoIfElse.java · MsRobotoWhileIfElse.java · MsRobotoWhileifNot.java

Brief Intro: **Parameters & Variables**

PLEASE NOTE: We will be going over **Parameters** & **Variables** again in more detail in several upcoming Lectures. This is just a teaser or "taste" of things to come ☺

A **parameter** is a **variable** that you *pass* into a **method** (or function)

A **variable** is a piece of data like a number that can be changed *programmatically* (it doesn't have to be the same number over and over again, it isn't constant or set in stone).

If you think back to math class, you spend a lot of time talking about **f(x)**.

In that case, '**f**' is the function, '**x**' is the variable, and the passing of '**x**' inside the function's *parentheses* is the parameter. In other words, something passed to '**x**' is going to change what the output of '**f**' gives you. Being a variable, '**x**' is a placeholder for any number that might be passed to the parameter.