Integer Data Types

- `byte`, `short`, `int`, and `long` are all integer data types.
- They can hold whole numbers such as 5, 10, 23, 89, etc.
- Integer data types cannot hold numbers that have a decimal point in them.
- Integers embedded into Java source code are called *integer literals*.

Declaring an Integer Variable

Declaring an integer variable:

```
int counter = 0;
```

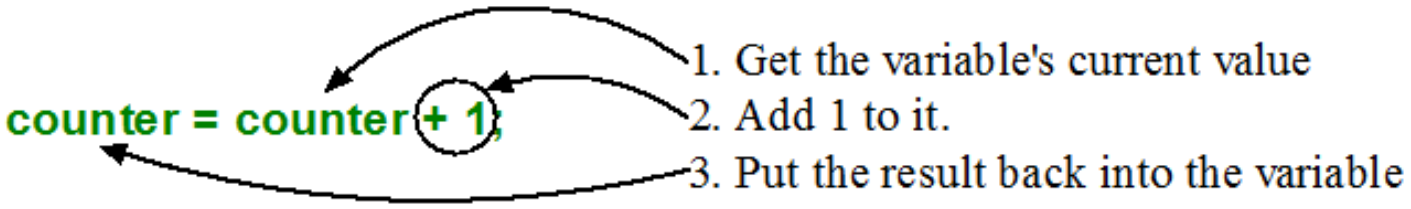
Using a variable's value:

```
if (counter == 0)
{ ...
}
```

```
while (counter > 0)
{ ...
}
```

```
this.myMethod(counter);
```

Changing a variable's value:



Another Way to Show Increment & Decrement

```
numMoves = numMoves + 1;
numThings = numThings - 1;
```

SAME AS

```
numMoves++;
numThings - -;
```

To Repeat:

`X = X + 1` is the same as `X++`

`X = X - 1` is the same as `X--`

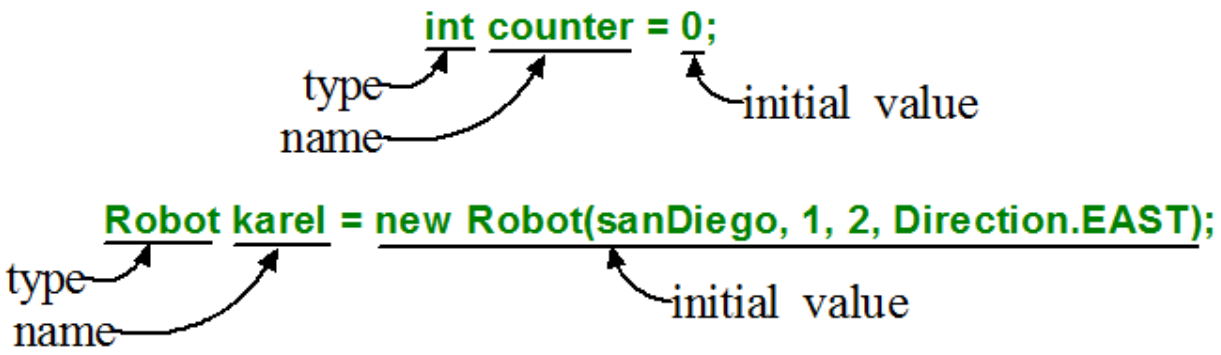
To Summarize:

Temporary Variables (Local Variables)

Temporary variables

- are also called "local variables."
- occur inside a method.
- store a value until the end of the method.
- have a type such as **Robot** or **int** that determines what kind of values may be stored in the variable.
- must be given an initial value.

Two examples:



Counters

- **Declare** the **datatype**, and give it a **name**:

```
int counter;
```

- Then, **initialize** it with a **value**:

```
counter = 0;
```

- So, putting it together it might look like this:

```
int counter;
counter = 0;
```

- You can also do this all on the same line by **combining** the **declaration** and the **initialization**, which saves keystrokes:

```
int counter = 0;
```

Counters CONTINUED

- You can use initialize counters outside of loops and inside of loops, which affects their **scope**, all depending on the logic of the code.

```
int counter = 0;

while (counter < 10) // As long as this is true, loop
{
    move();
    counter++; // Same as counter = counter + 1;
}
```

```
NOTE:  If you forget to increment your counter, you could
end up with an infinite loop, aka, a runtime error.
```

EXAMPLE:

Counting Things on an Intersection

Wanted:

Count things on this intersection

```
if (0 things)
{ this.move();
}
if (1 thing)
{ this.turnLeft();
}
if (2 things)
{ this.turnRight();
}
```

Solution:

```
int numThingsHere = 0;

while (this.canPickThing())
{ this.pickThing();
  numThings = numThings + 1;
}

if (numThingsHere == 0)
{ this.move();
}
if (numThingsHere == 1)
{ this.turnLeft();
}
if (numThingsHere == 2)
{ this.turnRight();
}
```

Storing the Results of a Query

We could do something equivalent with the number of things in the robot's backpack:

One way…

```
if (this.countThingsInBackpack() == 0)
{ this.move();
}
if (this.countThingsInBackpack() == 1)
{ this.turnLeft();
}
if (this.countThingsInBackpack() == 2)
{ this.turnRight();
}
```

Using a temporary variable…

```
int numThings = this.countThingsInBackpack();
if (numThings == 0)
{ this.move();
}
if (numThings == 1)
{ this.turnLeft();
}
if (numThings == 2)
{ this.turnRight();
}
```

Boolean Expressions and Operators

A *Boolean Expression* is an expression that results in a Boolean value, that is, in a value that is either *true* or false.

More complex Boolean expressions can be built out of simpler expressions, using the Boolean *Logical Operators*.

Chapter 5.4

Using **Comparison Operators**, the test in **if** and **while** statements are Boolean expressions that give a true or false answer to a question. So far, our questions have been simple. As our programming skills grow, however, we will want to ask more complex questions, which will need more complex Boolean expressions to answer.

Just like the mathematic operators (**+ - / ***) can be combined to form *more complex expressions* like

```
s = (( x + y) * (w - z)) / 2
```

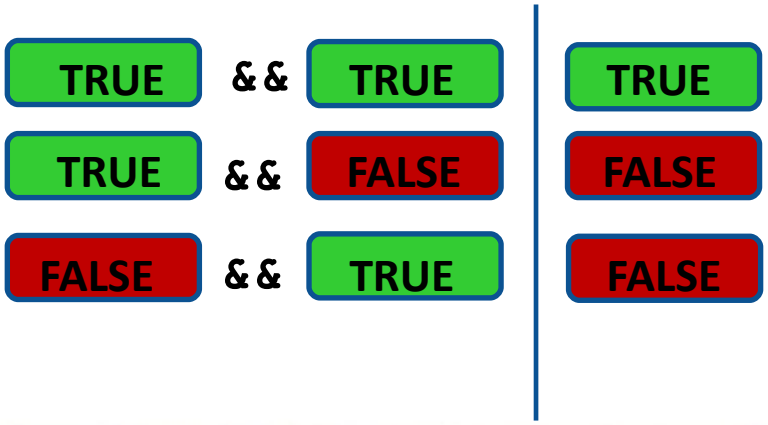
so too can Boolean expressions be combined to create more complex and utility expressions using **`and`** (represented by **&&**) and **`or`** (represented by **||**).

Logical Operators

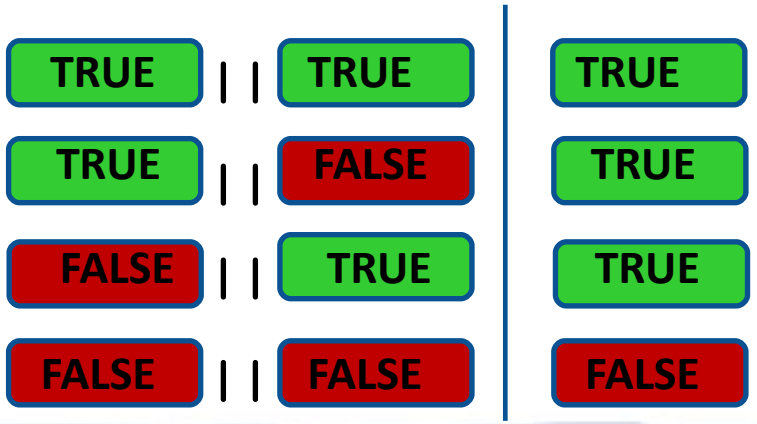
Logical Operators	
AND A && B are true only if both A and B are true	**&&**
OR A || B are true if A is true or B is true or they're both true	**||**
NOT !A is the opposite of A. If A is true, then !A is false. If A is false, !A is true.	**!**

The double ampersand **&&** and double pipe **||** are called "***short circuit" logical operators***

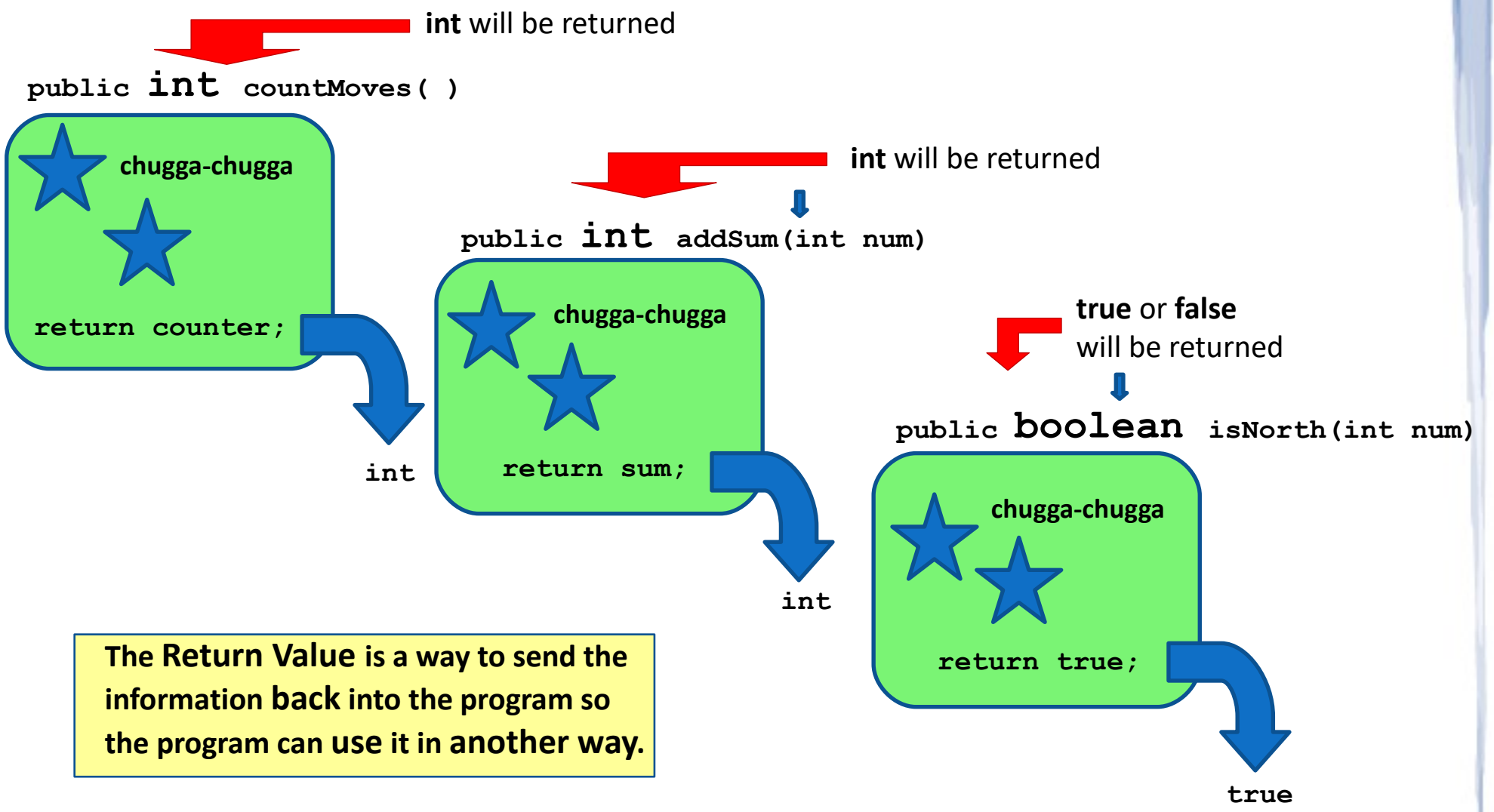
AND Operator



OR Operator



Return a value by declaring the type



The Return Value is a way to send the information back into the program so the program can use it in another way.

Predicate Methods

Predicate methods are methods with a Boolean return value.

```
if(!this.frontIsClear()){

// what to do if this robot's front is
blocked

}

if(this.frontIsBlocked()){

// what to do if this robot's front is
blocked

}
```

Predicate Methods

Write the helper function frontIsBlocked:

```
if(this.frontIsBlocked()){
// what to do if this robot's front is
blocked
}
public boolean frontIsBlocked(){
     return !this.frontIsClear();
}
```

More Predicate Methods

```
public boolean frontIsBlocked(){
   return !this.frontIsClear();
}
protected boolean atRowsEastEnd(){
   return this.getAvenue() == 5;
}
protected boolean isFacingSouth(){
   return this.getDirection()== Direction.SOUTH;
}
```