Unit 3: Boolean Expressions and If Statements

Mr. Pelletier, Lord Byng Secondary

Unit 3 Overview

In Unit 3, we explore in-depth the if statement, which allows programmers to control the FLOW of a program's execution.

Topics will include:

- The boolean data type and relational operators that make up boolean expressions
- The if statement that uses a boolean expression to decide whether code should run or not.
- The if else statement that can choose between two blocks of code to run
- The if else if statement that can choose between an arbitrary number of choices to run.
- &&, ||, and ! operators to create compound boolean expressions.
- How to make simplify and evaluate equivalent boolean expressions
- How to compare objects.

Unit 3.1: Boolean Expressions

Mr. Pelletier, Lord Byng Secondary

Unit 3.1 Boolean Expressions

Overview

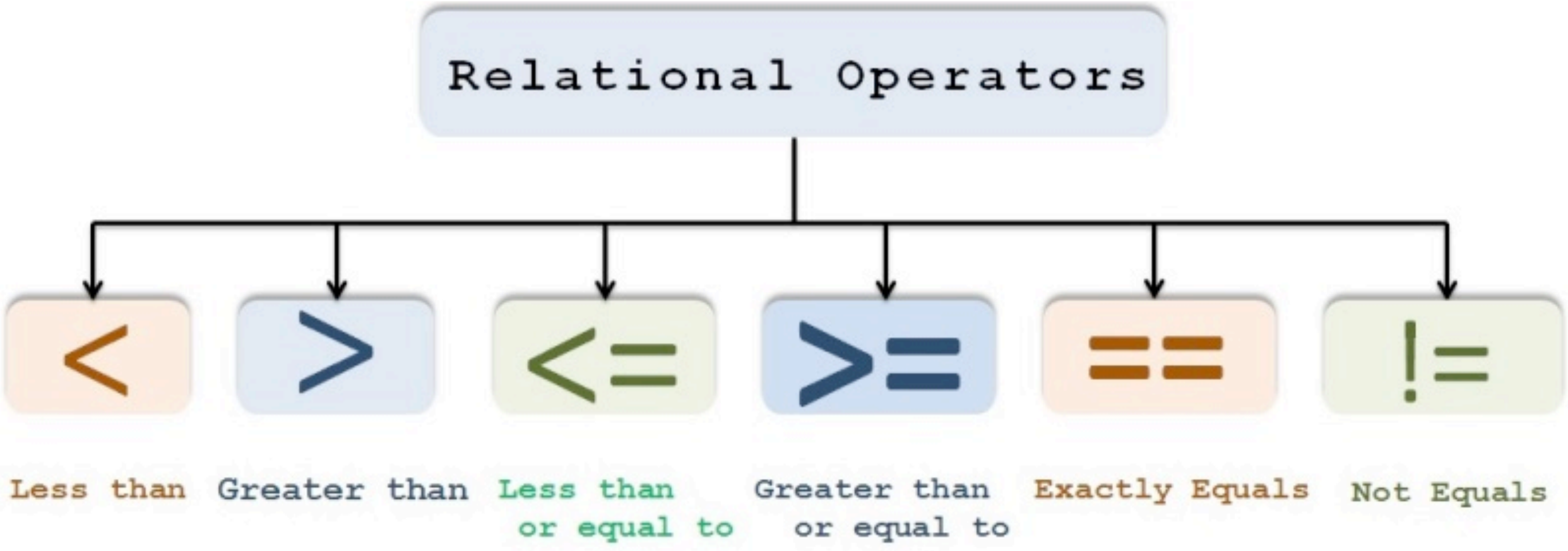
Recall that **Expressions** are combinations of variables, literals, and operators that evaluate to a single value.

We have already looked at **Arithmetic Expressions** that evaluate to numbers and allow computers to do math.

Now we will look at **Boolean Expressions** that evaluate to true/false. They allow computers to **ask questions** and **make decisions**.

Unit 3.1 Boolean Expressions

Relational Operators



Unit 3.1 Boolean Expressions

You Try It

What does this boolean expression evaluate to?

$$(5 \% 3 == 0) == (3 > 5)$$

Assume x and y are int variables. What values of x and y will make this boolean expression true?

```
(x >= 10) == (y < 12)
```

Unit 3.2: If Statements

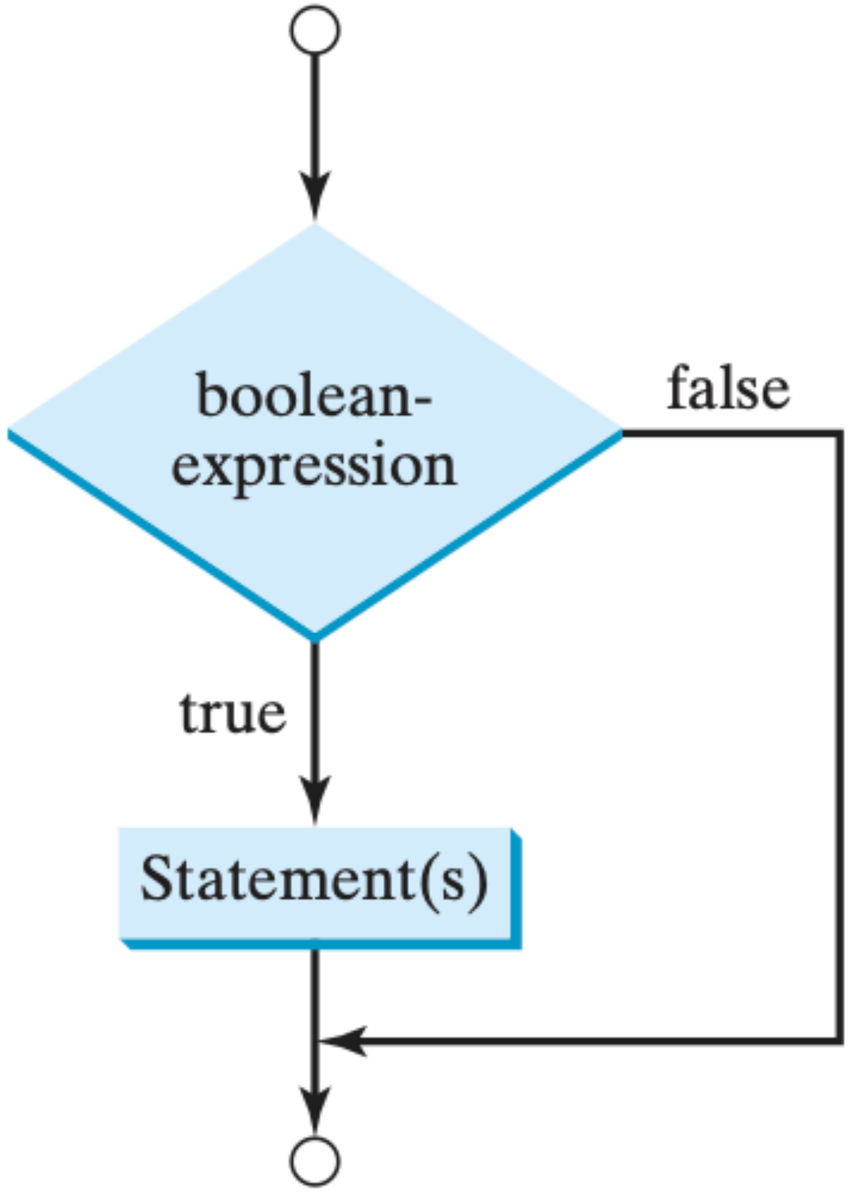
Mr. Pelletier, Lord Byng Secondary

Unit 3.2 If Statements

Overview

If statements allow you to control the flow of a program. They can allow a program to choose to execute a block of code or not based on the value of a boolean expression.

```
if (boolean expression) {
   Statement(s)
}
```



Unit 3.2 If Statements

Example with Braces

```
if (r >= 0) {
    area = r * r * Math.PI;
    System.out.println("Circle area: " + area);
}
System.out.println("Circles are neat");
```

- What is the output if r is positive? What about if r is negative?

Unit 3.2 If Statements

Example with No Braces

What's going on here? What happens if r is positive? Negative?

```
if (r >= 0)
area = r * r * Math.PI;
System.out.println("Circle area: " + area);
System.out.println("Circles are neat");
```

Unit 3.2 If Statements

Example with No Braces

If there are no braces, Java treats only the next statement as being connected to the if statement.

```
if (r >= 0)
    area = r * r * Math.PI;
System.out.println("Circle area: " + area);
System.out.println("Circles are neat");
```

Unit 3.2 If Statements

A Note About Style 1/5

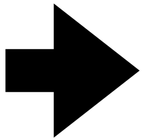
If you have an if statement like this:

```
boolean alive = lives > 0;
if (alive == true) {
    playGame();
}
```

Unit 3.2 If Statements

A Note About Style 2/5

```
boolean alive = lives > 0;
if (alive == true) {
    playGame();
}
```



This is equivalent:

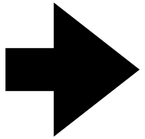
```
boolean alive = lives > 0;
if (alive) {
    playGame();
}
```

Unit 3.2 If Statements

A Note About Style 3/5

Similarly, these if statements are also the same:

```
boolean alive = lives > 0;
if (alive == false) {
    playGame();
}
```



```
boolean alive = lives > 0;
if (!alive) {
    area = r * r * Math.PI;
}
```

The ! is the "not" operator, which inverts the boolean value. We will see more about this and other boolean operators in lesson 3.5

Unit 3.2 If Statements

A Note About Style 4/5

There are also, these if statements are also the same:

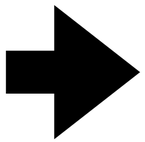
```
boolean over21;
if (age > 21) {
    over21 = true;
} else {
    over21 = false;
}
```

Unit 3.2 If Statements

A Note About Style 5/5

Many statements can be more succinctly expressed.

```
boolean over21;
if (age > 21) {
    over21 = true;
} else {
    over21 = false;
}
```



```
boolean over21 = age > 21;
```

Unit 3.3: If Else Statements

Mr. Pelletier, Lord Byng Secondary

Unit 3.3 If Else Statements

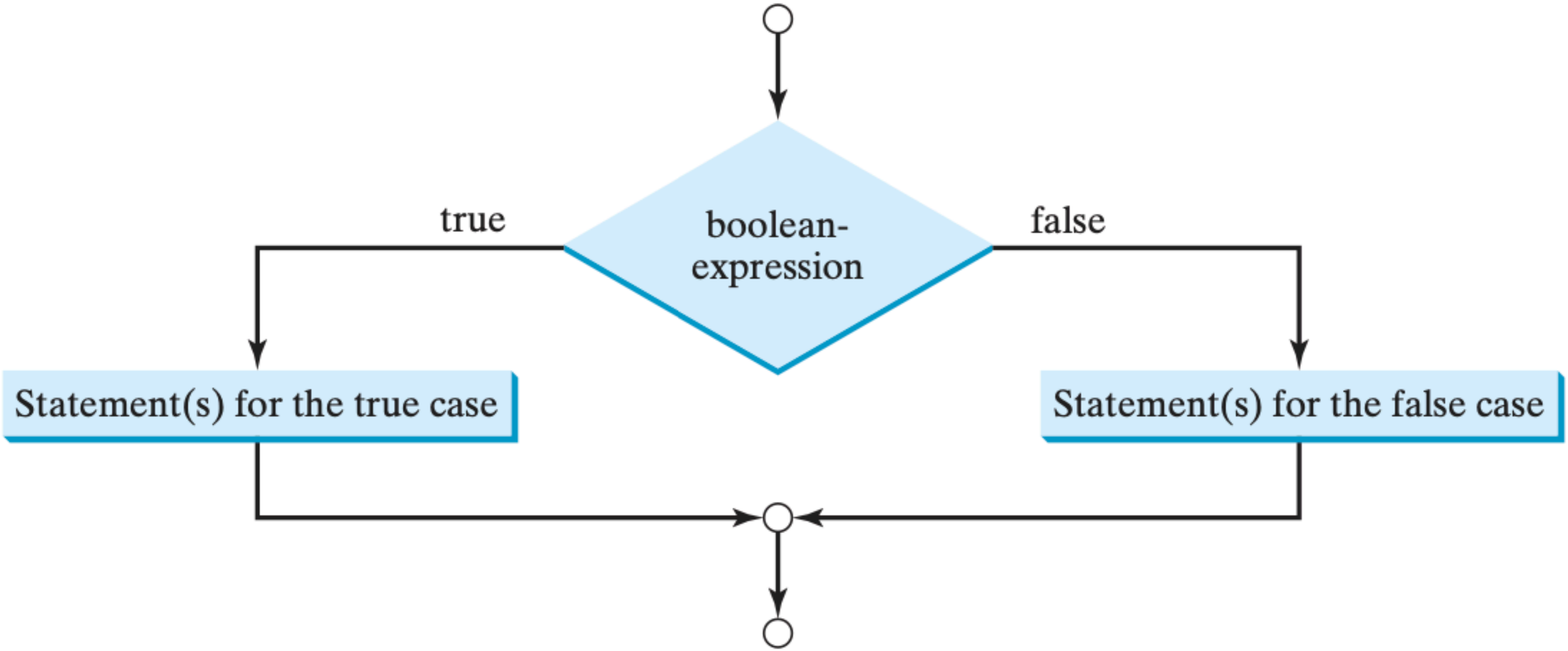
Overview

If else statements allow you to control the flow of a program. They can allow a program to choose one of two blocks of code to execute based on the value of a boolean expression.

```
if (boolean expression) {
   Statement(s) for the true case
} else {
   Statement(s) for the false case
}
```

Unit 3.3 If Else Statements

Overview



Unit 3.3 If Else Statements

Example with Braces

```
if (r >= 0) {
   area = r * r * PI;
   System.out.println("The area for the circle of radius " +
                      r + " is " + area);
} else {
   System.out.println("Error: radius can’t be negative");
}
```

Unit 3.3 If Else Statements

Example with Braces

```
if (n % 2 == 0) {
  System.out.print(“even”);
} else {
  System.out.print("odd”);
}
```

This code prints "odd" or "even" depending on whether n is odd or even.

Unit 3.3 If Else Statements

If Else is a Single, Continuous Statement

```
if (n % 2 == 0) {
  System.out.print(“even”);
}
System.out.println(“Let’s take a break”); ❌
else {
  System.out.print("odd”);
}
```

This will not compile because the else clause must immediately follow the if block.

Unit 3.3 If Else Statements

Dangling Else

What is printed out if n is 7?

```
int n = ...;              //read user input
if (n > 0)
    if (n % 2 == 0)
        System.out.println(n);
else
    System.out.println(n + " is not positive");
```

Unit 3.4: If Else If Statements

Mr. Pelletier, Lord Byng Secondary

Unit 3.4 If Else-If Statements

Overview

If else-if statements allow you to choose between an arbitrary number of options. This example shows three, but you can keep on adding more.

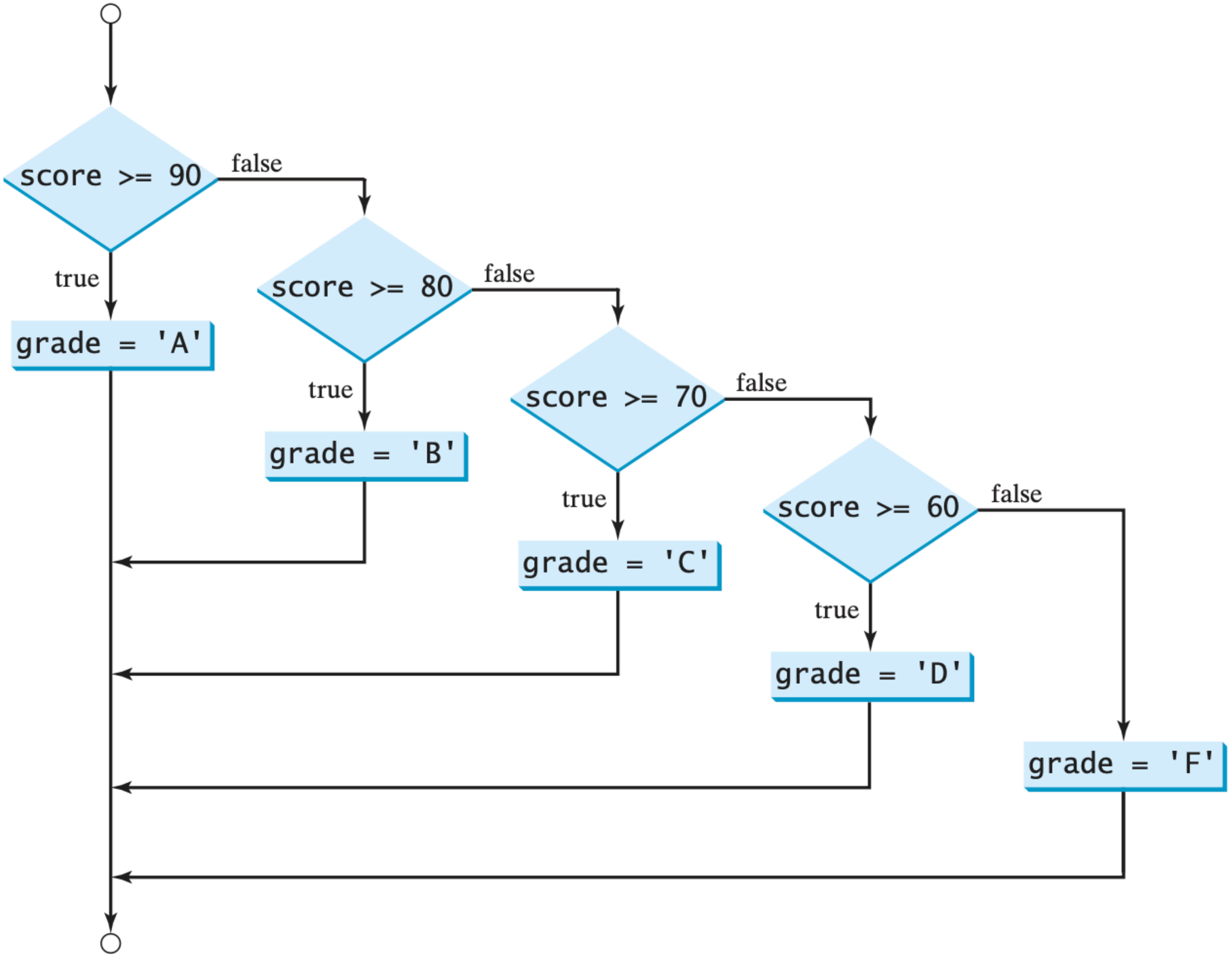
```
if (boolean expression 1) {
  block one!
} else if (boolean expression 2) {
  block two!
} else if (boolean expression 3) {
  block three!
} ...
```

Unit 3.4 If Else Statements

How it works 1/4

Java checks the conditions sequentially from top to bottom. Once a condition has been met, it DOES NOT check the rest of the conditions or run any of the other blocks. For example:

```
if (score >= 90.0) grade = 'A';
else if (score >= 80.0) grade = 'B';
else if (score >= 70.0) grade = 'C';
else if (score >= 60.0) grade = 'D';
else grade = 'F';
```


score >= 90
false
true
grade = 'A'
score >= 80
false
true
grade = 'B'
score >= 70
false
true
grade = 'C'
score >= 60
false
true
grade = 'D'
grade = 'F'

Unit 3.4 If Else Statements

How it works 2/4

Compare these two pieces of code. What is the difference if x is 50?

```
if (x > 10)
  System.out.println(“A”);
else if (x > 20)
  System.out.println(“B”);
else if (x > 30)
  System.out.println(“C”);
```

```
if (x > 10)
  System.out.println(“A”);
if (x > 20)
  System.out.println(“B”);
if (x > 30)
  System.out.println(“C”);
```

Unit 3.4 If Else Statements

How it works 4/4

The else clause at the end is optional.

```
if (grade >= 100)
  System.out.println(“A++”);
else if (grade >= 95)
  System.out.println(“A+”);
else if (grade >= 90)
  System.out.println(“A”);
else
  System.out.println(“¯\_(ツ)_/¯”);
```

Unit 3.5: Compound Boolean Expressions

Mr. Pelletier, Lord Byng Secondary

Unit 3.5 Compound Boolean Expressions

Overview

If else if statements allow choose between an arbitrary number of options. This example shows three, but you can keep on adding more.

Operator	Meaning
!	Not
&&	And
||	Or

Unit 3.5 Compound Boolean Expressions

Overview

Here's a set of Truth Tables that show how they work:

AND

A	B	A&&B
F	F	F
T	F	F
F	T	F
T	T	T

OR

A	B	A||B
F	F	F
T	F	T
F	T	T
T	T	T

NOT

A	!A
F	T
T	F

Unit 3.5 Compound Boolean Expressions

Operator Precedence 1/3

This is also their order of precedence. Like algebraic expressions, boolean expressions are evaluated left to right and you can use parentheses to force operations to happen out of precedence order.

Operator	Meaning
!	Not
&&	And
||	Or

Unit 3.5 Compound Boolean Expressions

Order of Precedence 2/3

Given the following declaration, what do the following compound boolean expressions evaluate to?

```
boolean A = true, B = true, C = false;
```

1. `!A && C`

2. `!(A && C)`

3. `A || B && C`

4. `(A || B) && C`

5. `A && !(B && C)`

Operator Precedence 3/3

Operator	Description	Direction of Evaluation
`()`	Parentheses	→
`++, --`	Increment/Decrement	→
`!`	Logical not	←
`(int)` `(double)`	Casting	→
`* / %`	Multiplicative	→
`+ - +`	Additive, String Concatenation	→
`< <=` `> >=`	Comparison	→
`==` `!=`	Equality	→
`&&`	And	→
`||`	Or	→
`= += -=` `*= /= %=`	Assignment	←

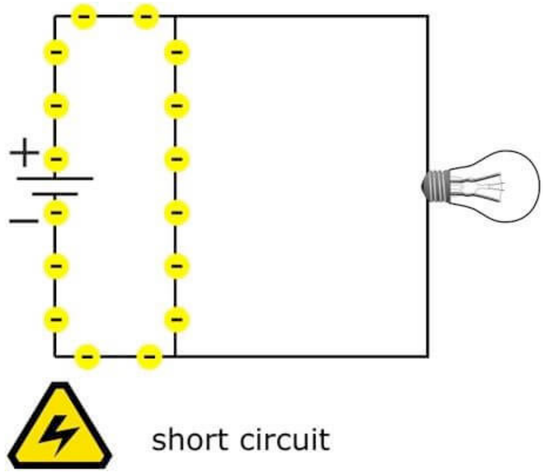
Unit 3.5 Compound Boolean Expressions

Short Circuit Evaluation 1/3

Java stops evaluating logical expressions as soon as the result is known. For &&:

```
condition1 && condition2 && condition3 ...
```

Java evaluates the conditions from left to right. As soon as a condition evaluates as false, it doesn't evaluate the rest of the conditions; the whole expression **must** be false.



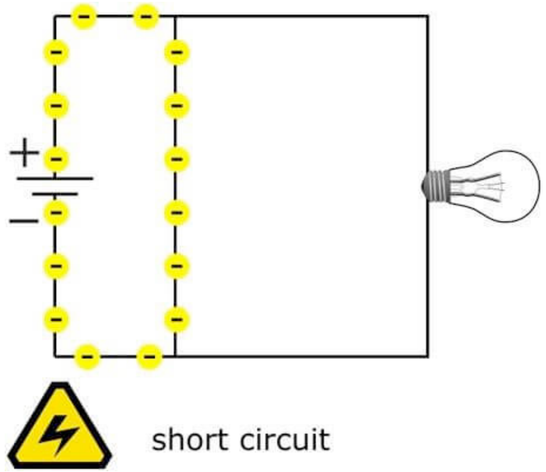
Unit 3.5 Compound Boolean Expressions

Short Circuit Evaluation 1/3

This also happens with ||

```
condition1 || condition2 || condition3 ...
```

Working left to right, if Java finds a condition that evaluates to true, then it stops evaluating the rest of the expressions.



Unit 3.5 Compound Boolean Expressions

Short Circuit Evaluation 3/3

This is useful in an example like this:

```
boolean result = (a != 0) && (b / a > 2);
```

In this example, the programmer cleverly protected the second condition `(b / a > 2)` from divide by zero errors using Short Circuit Evaluation.

Unit 3.6: Equivalent Boolean Expressions

Mr. Pelletier, Lord Byng Secondary

Unit 3.6 Equivalent Boolean Expressions

Overview

In this lesson, we will be comparing whether two boolean expressions are equivalent.

Two boolean expressions are equivalent if they evaluate to the same value in all case.

We will learn two ways to do this: De Morgan’s law and Truth Tables.

Unit 3.6 Equivalent Boolean Expressions

De Morgan's Laws 1/4

August De Morgan was a 19th century British mathematician who read George Boole's ideas about boolean logic.

He codified laws to help us **simplify boolean expressions**, especially regarding the ! operator.

Unit 3.6 Equivalent Boolean Expressions

De Morgan's Laws 2/4

How does ! work with relational operators in boolean expressions?

`!(x > 0)`	is equivalent to	`x <= 0`
`!(x < 0)`	is equivalent to	`x >= 0`
`!(x <= 0)`	is equivalent to	`x > 0`
`!(x >= 0)`	is equivalent to	`x < 0`
`!(x == 0)`	is equivalent to	`x != 0`
`!(x != 0)`	is equivalent to	`x == 0`

Negating a relational operator makes it flip to the opposite operator!

Negating a boolean expression means finding the opposite of that expression

Unit 3.6 Equivalent Boolean Expressions

De Morgan's Laws 3/4

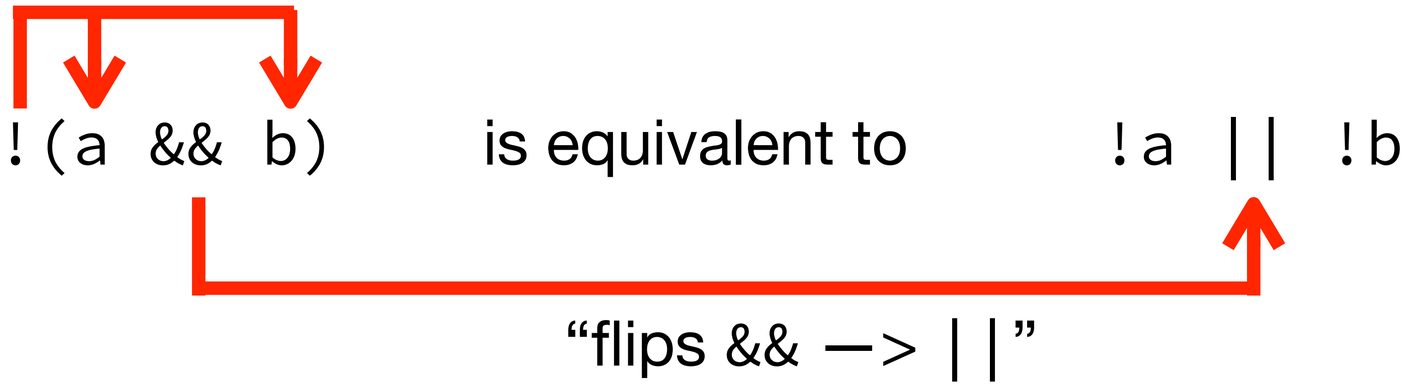
How does ! work with a compound boolean expression? Given 2 boolean variables/expressions, a and b:

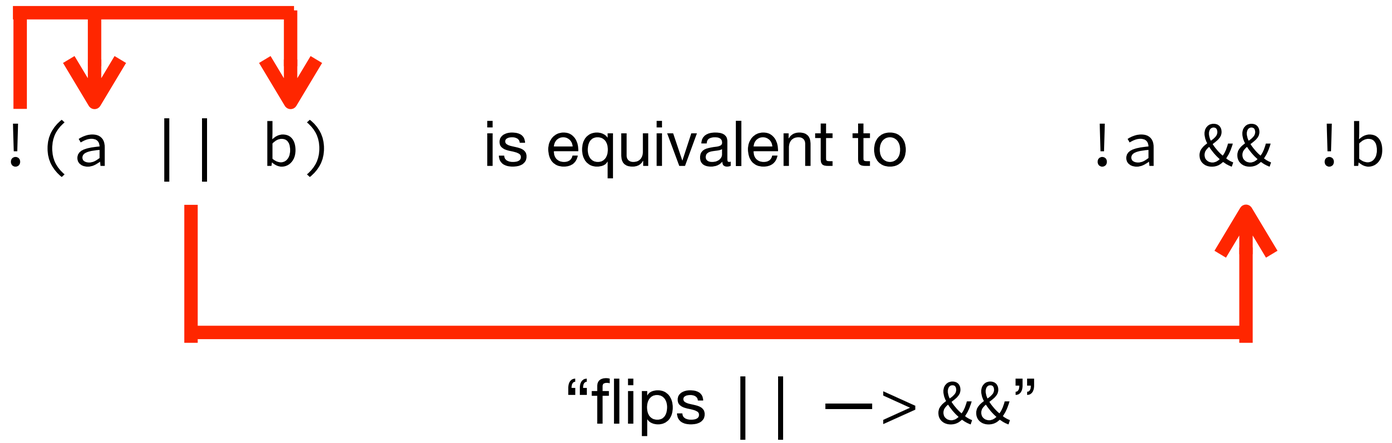
`!(a && b)`	is equivalent to	`!a || !b`
`!(a || b)`	is equivalent to	`!a && !b`

Unit 3.6 Equivalent Boolean Expressions

De Morgan's Laws 3/4

This is effectively the laws of distributing a Not operator





Unit 3.6 Equivalent Boolean Expressions

Visual Example 1/2

Say we have the following boolean expression: `(x < -5 || x > 10)`

How would we negate it?

Unit 3.6 Equivalent Boolean Expressions

Visual Example 2/2

Using De Morgan's laws:

```
!(x < -5 || x > 10)

!(x < -5) && !(x > 10)

x >= -5 && x <= 10
```

Therefore, `!(x < -5 || x > 10)` is equivalent to

```
x >= -5 && x <= 10
```

Unit 3.6 Equivalent Boolean Expressions

Truth Tables 1/6

Truth Tables allow you to test all combinations of boolean expressions.

p	q	p&&q	p||q	!p	!q
T	T				
T	F				
F	T				
F	F				

Unit 3.6 Equivalent Boolean Expressions

Truth Tables 2/6

No column is the same, so none of these boolean expressions are equivalent.

p	q	p&&q	p||q	!p	!q
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

Unit 3.6 Equivalent Boolean Expressions

Truth Tables 3/6

Let's use a Truth Table to confirm De Morgan's Law:

p	q	!p	!q	! (p&&q)	!p||!q
T	T	F	F		
T	F	F	T		
F	T	T	F		
F	F	T	T		

Unit 3.6 Equivalent Boolean Expressions

Truth Tables 4/6

Because their truth tables are the same, the expressions are equivalent.

p	q	!p	!q	! (p&&q)	!p||!q
T	T	F	F	F	F
T	F	F	T	T	T
F	T	T	F	T	T
F	F	T	T	T	T

Unit 3.6 Equivalent Boolean Expressions

Truth Tables 5/6

Practice it yourself! Are any of these expressions equivalent?

p	q	p||q&&p	(p||q)||p	p&&(p||q)
T	T			
T	F			
F	T			
F	F			

Unit 3.6 Equivalent Boolean Expressions

Truth Tables 6/6

Indeed!

p	q	p||q&&p	(p||q)||p	p&&(p||q)
T	T	T	T	T
T	F	T	T	T
F	T	F	T	F
F	F	F	F	F

Unit 3.6 Equivalent Boolean Expressions

Example 1/2

Given two boolean variables, x and y, which of the following expressions is equivalent to `(x && !y)`?

A. `(!x || y)`

B. `(!x && y)`

C. `!(!x || y)`

D. `!(!x && y)`

E. `(x || !y)`

Unit 3.6 Equivalent Boolean Expressions

Example 2/2

Given two boolean variables, x and y, which of the following expressions is equivalent to `(x && !y)`?

A. `(!x || y)`

B. `(!x && y)`

C. `!(!x || y)`

D. `!(!x && y)`

E. `(x || !y)`

Unit 3.7: Comparing Objects

Mr. Pelletier, Lord Byng Secondary

Unit 3.7 Comparing Objects

Overview

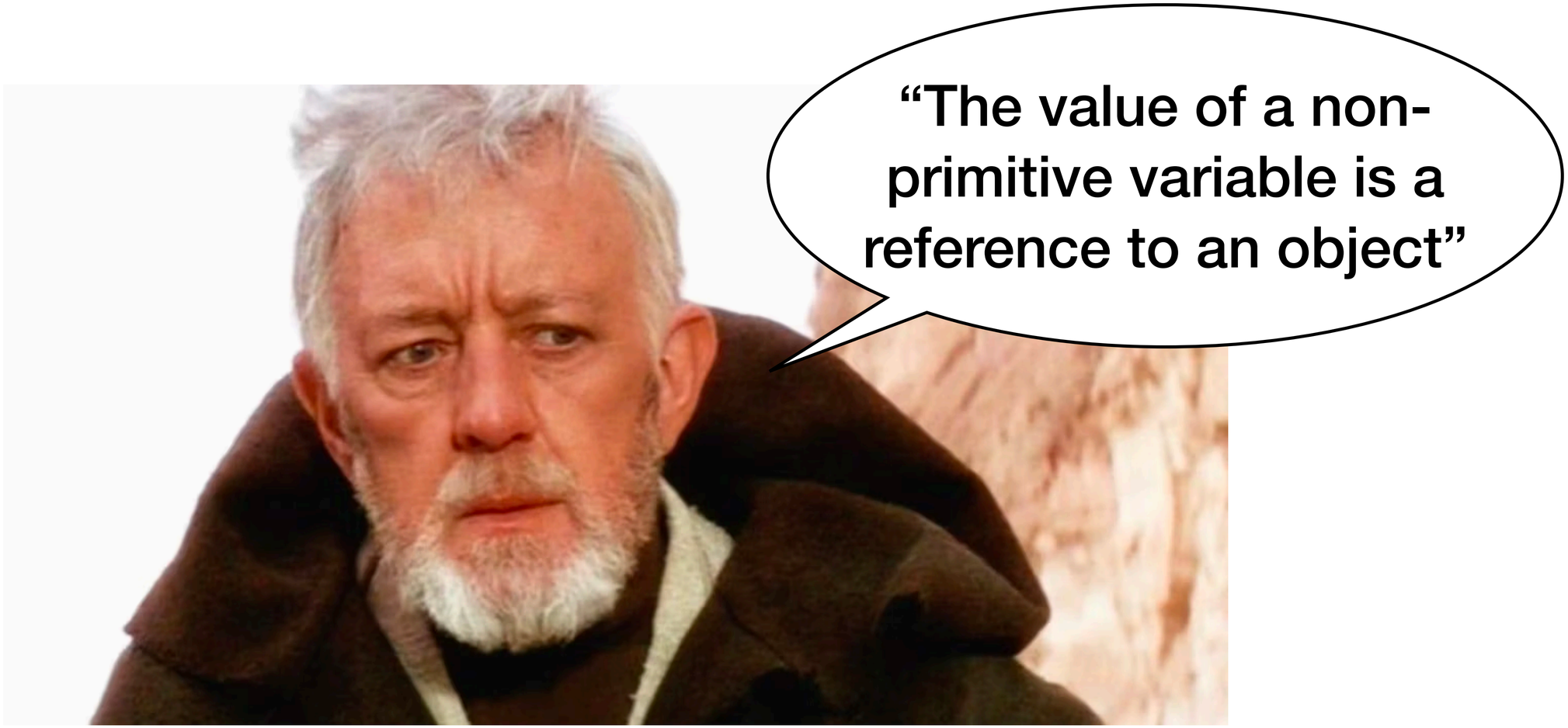
How do you compare the value of two objects?

How do you compare whether two non-primitive variables point to the same object?

How do you check to see if a non-primitive variable is null?

Unit 3.7 Comparing Objects

The Foundational Idea

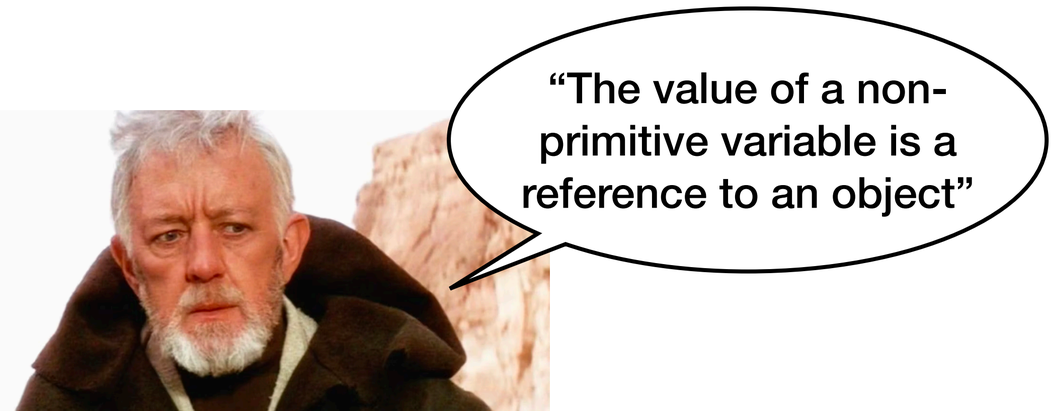


Unit 3.7 Comparing Objects

The Foundational Idea

The == operator compares the value of two variables.

So, for non-primitives, it compares their references, **NOT** whether the object they point are equivalent.



Unit 3.7 Comparing Objects

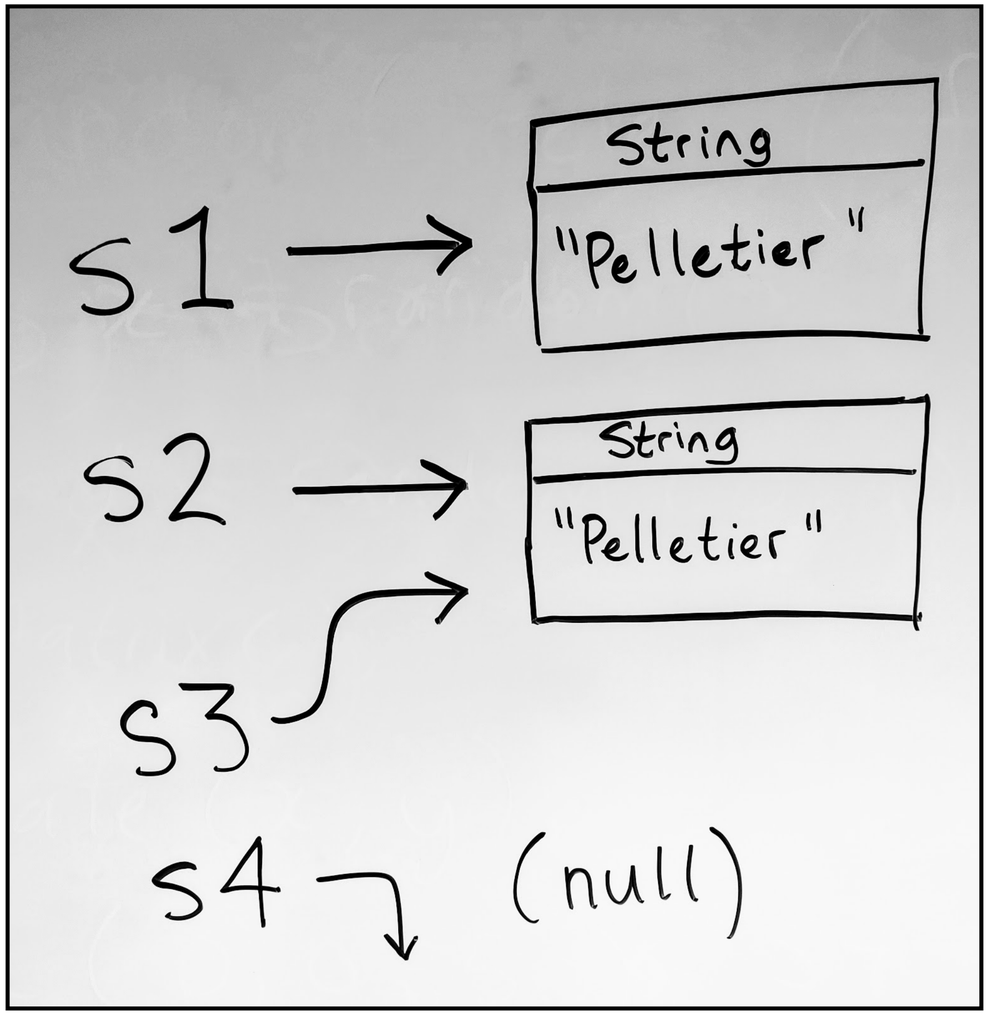
Example

```
String s1 = “Pelletier”;
String s2 = “Pelletier”;
String s3 = s2;
String s4;
```

What is the value of `s1 == s2?` false!

`s2 == s3?` true!

`s4 != null?` false!



Unit 3.7 Comparing Objects

Aliases

If two non-primitives point to the same object, they are called aliases.

```
String s1 = “Pelletier”;
String s2 = “Pelletier”;
String s3 = s2;
String s4;
```

Here, s2 and s3 are aliases

Unit 3.7 Comparing Objects

So How Do You Compare Objects?

Most objects have a built-in equals and/or compareTo method.

```
String s1 = “Pelletier”;
String s2 = “Pelletier”;
String s3 = s2;
String s4;
```

What are the values of the following?

```
s1.equals(s2)? true
s2.equals(s3)? true
s4.equals(s1)? null pointer
               exception!
```