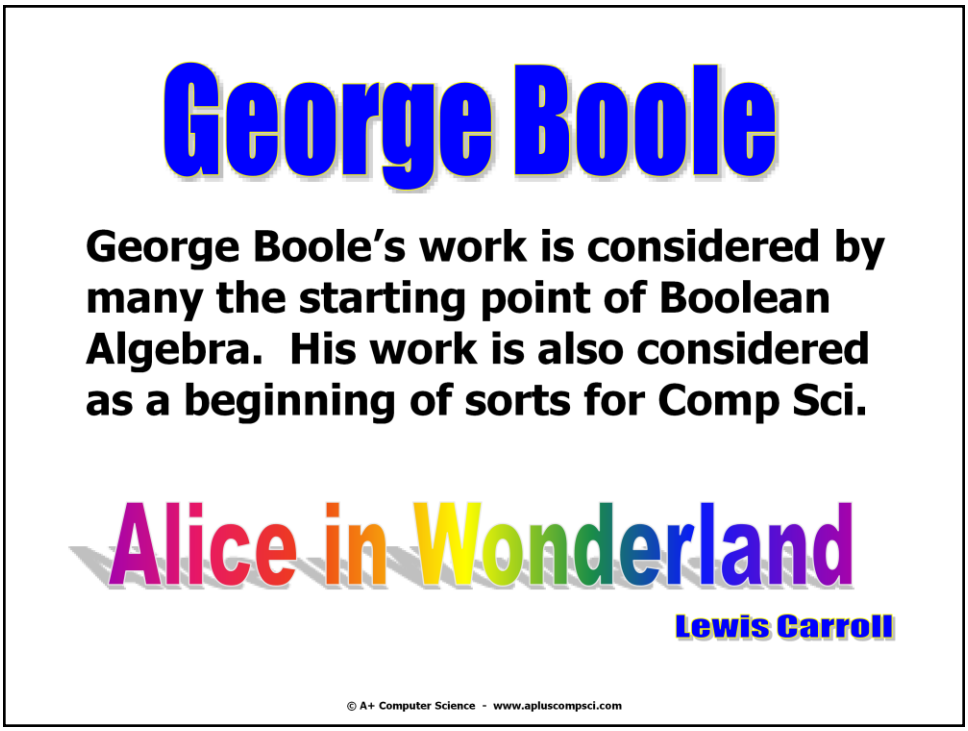
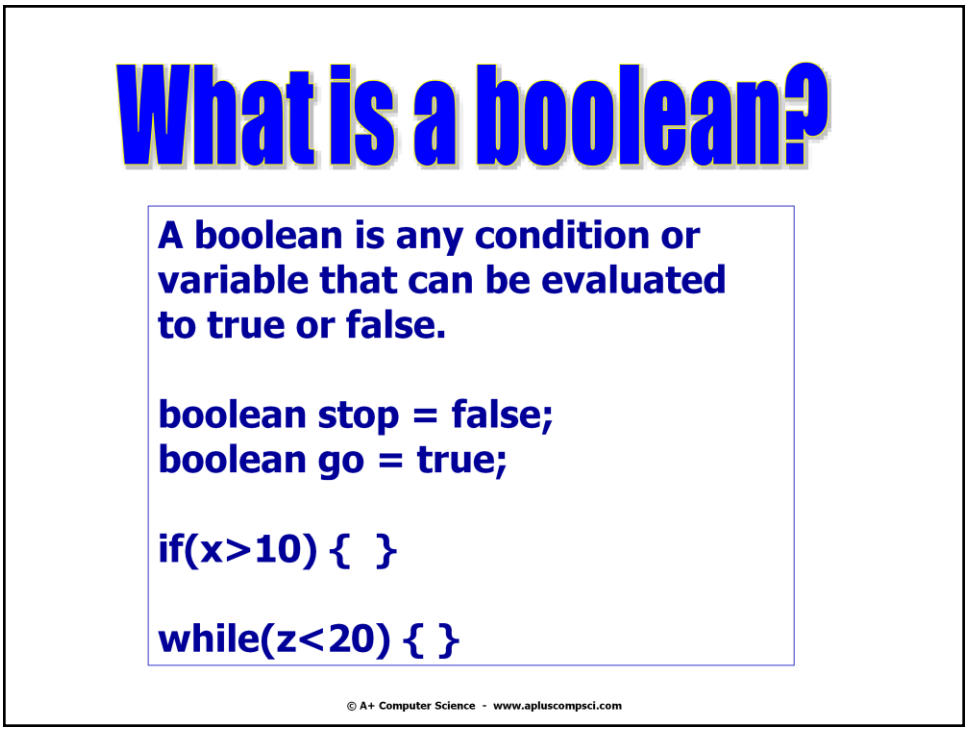
Boolean Algebra



George Boole is credited with laying the foundation for Boolean Algebra. The work of George Boole is still very important today.



A boolean is anything that can be evaluated as true or false. Boolean variables can store the value true or false. Ifs and Loops have boolean conditions that are evaluated to true or false.

Operator Precedence

()	HIGH
! ++ --	
* / %	
+ -	
<< >> (bitwise shifts)	
< <= > >=	
== !=	
& (bitwise and)	
^ (bitwise xor)	
| (bitwise or)	
&& (logical and)	
|| (logical or)	
= += -= *= /= %=	
,	LOW

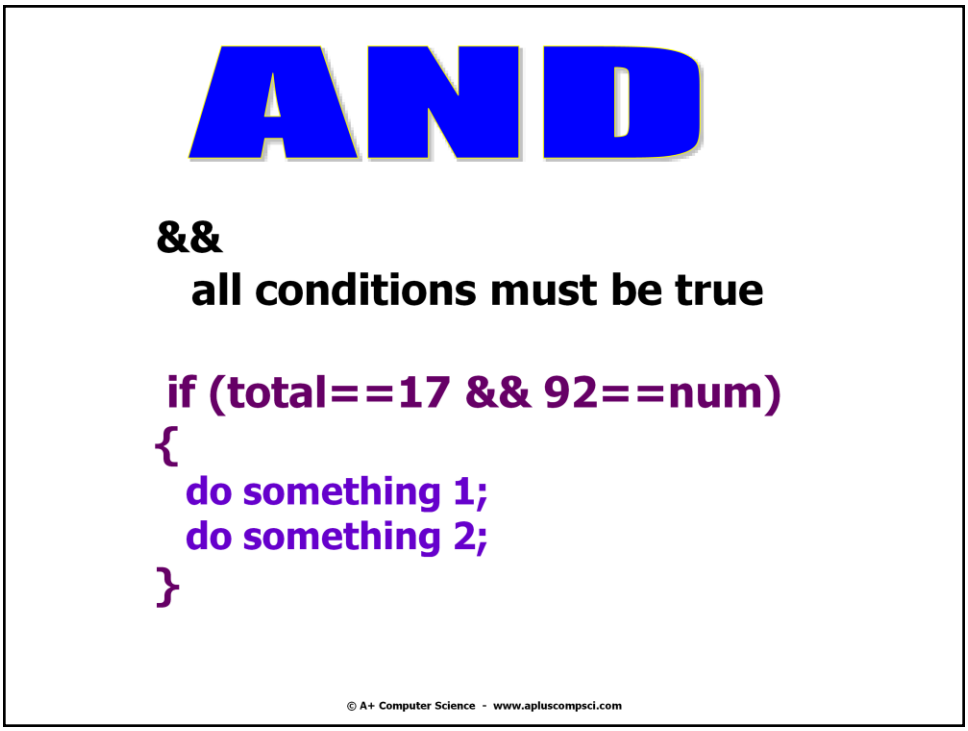
Common Boolean Symbols

Name	Boolean Symbol	Java Counterpart
and	∧ logical and	&&
or	∨ logical or	||
not	¬ logical not	!

And is used to see if all parts are true. In some languages, and is actually written as a word. In other languages, and is written as a symbol, like `&&` or `&`.

Or is used to see if any part is true. In some languages, or is actually written as a word. In other languages, or is written as a symbol, like `||` or `|`.

Not is used to negate a boolean value. In some languages, not is actually written as a word. In other languages, not is written as a symbol, like `!`.



And is used to see if all parts are true. In some languages, and is actually written as a word. In other languages, and is written as a symbol, like `&&` or `&`.

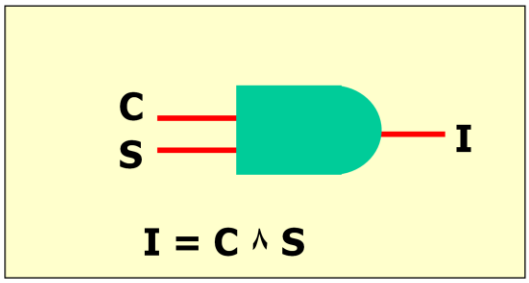
`&&` evaluates as true if all parts connected by `&&`s are true.

```
if(A and B)
```

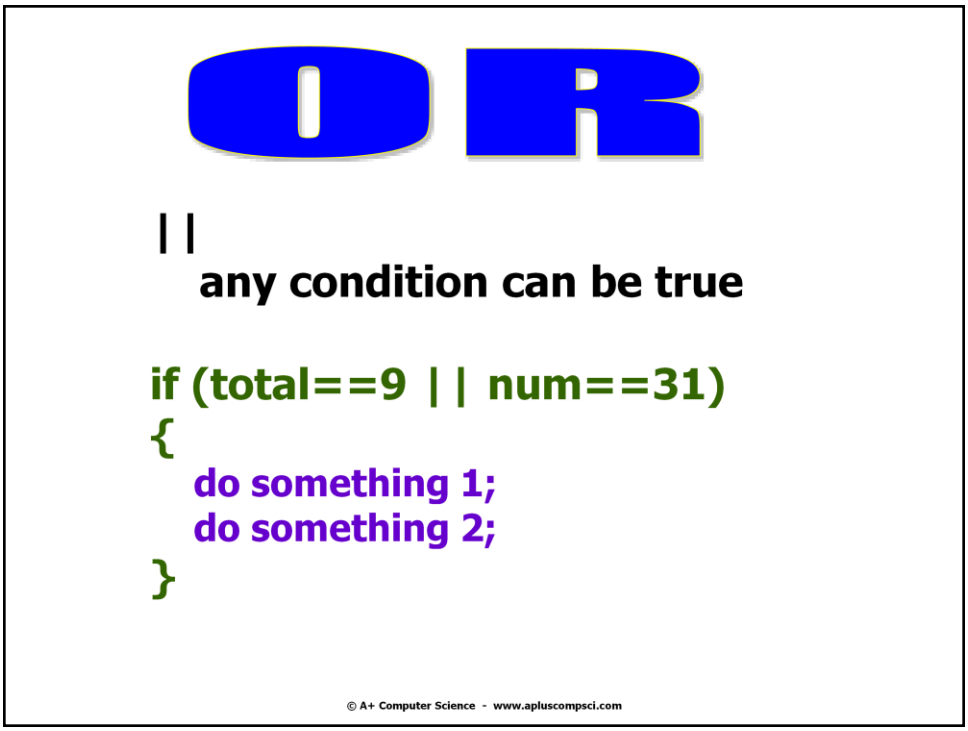
This condition is true if A and B are both true. If either A or B is false, the condition is false as both parts must be true in order for the condition to be true.

AND

Engineering Symbol



C	S	I
0	0	0
0	1	0
1	0	0
1	1	1



Or is used to see if any part is true. In some languages, or is actually written as a word. In other languages, or is written as a symbol, like `||` or `|`.

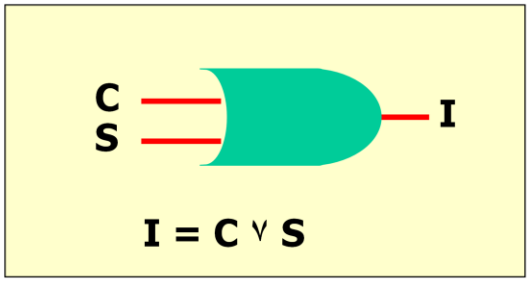
`||` evaluates as true if any part connected by `||`s is true.

```
if(A or B)
```

This condition is true if A or B is true. If A and B are both true, the condition is still true.

OR

Engineering Symbol

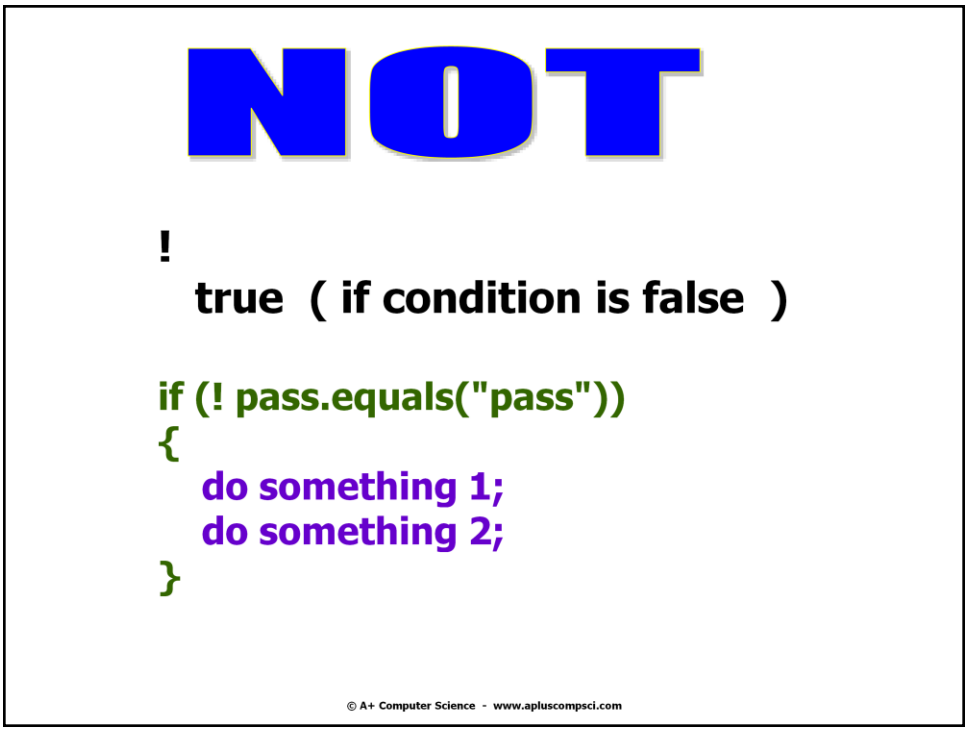


C	S	I
0	0	0
0	1	1
1	0	1
1	1	1

Fundamental Boolean Logic

true and false = false
false and true = false
false and false = false
true and true = true

false or true = true
true or false = true
true or true = true
false or false = false



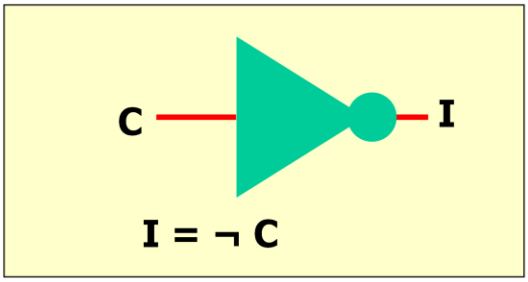
Not is used to negate a boolean value. In some languages, not is actually written as a word. In other languages, not is written as a symbol, like !.

!true is false

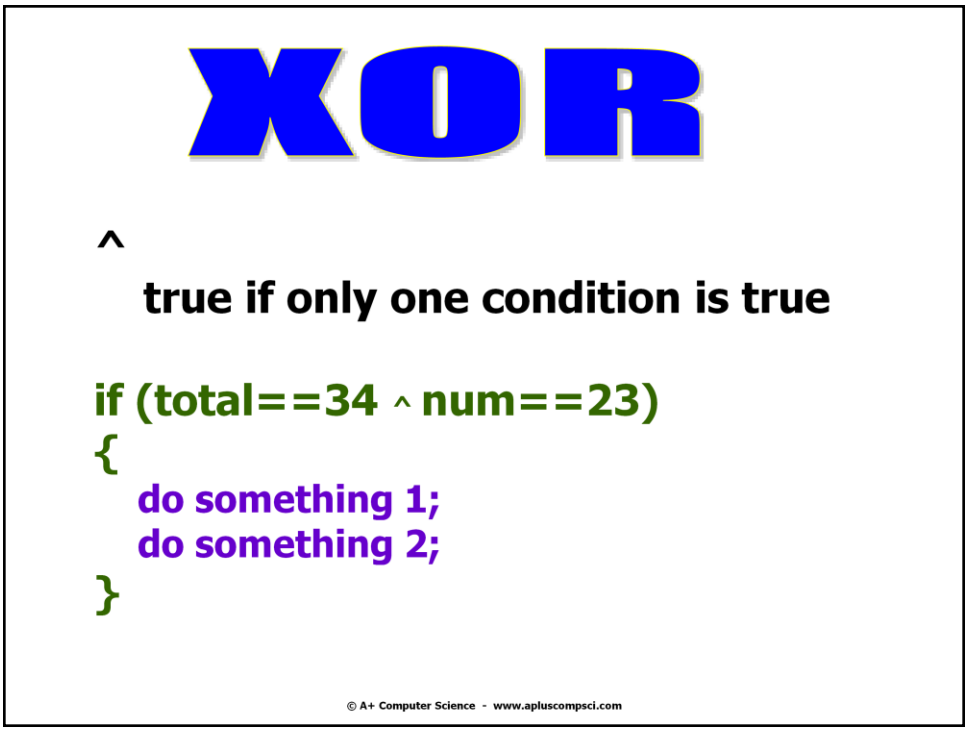
!false is true

NOT

Engineering Symbol



C	I
0	1
1	0



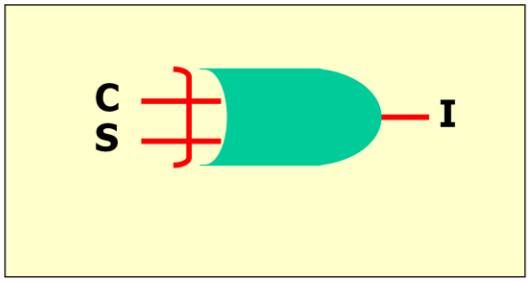
Xor is a very interesting boolean operator. Xor is true if any part of the condition is true, but false is more than one part is true.

```
if(A xor B)
```

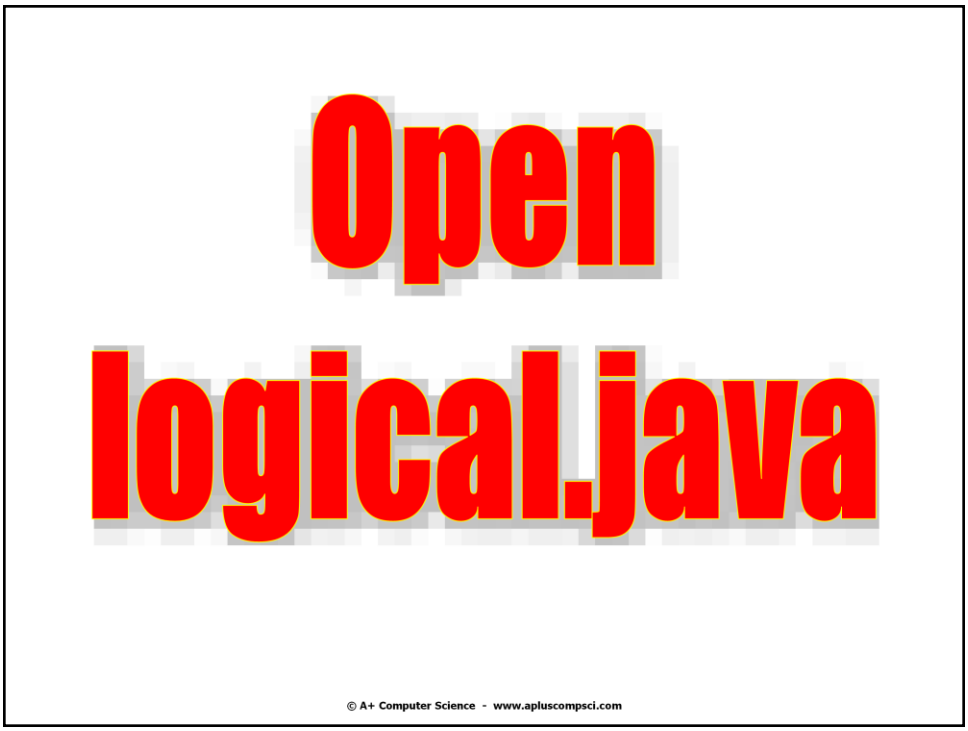
This condition is true if A or B is true. If A and B are both true, the condition is false. If A and B are both false, the condition is false.

XOR

Engineering Symbol

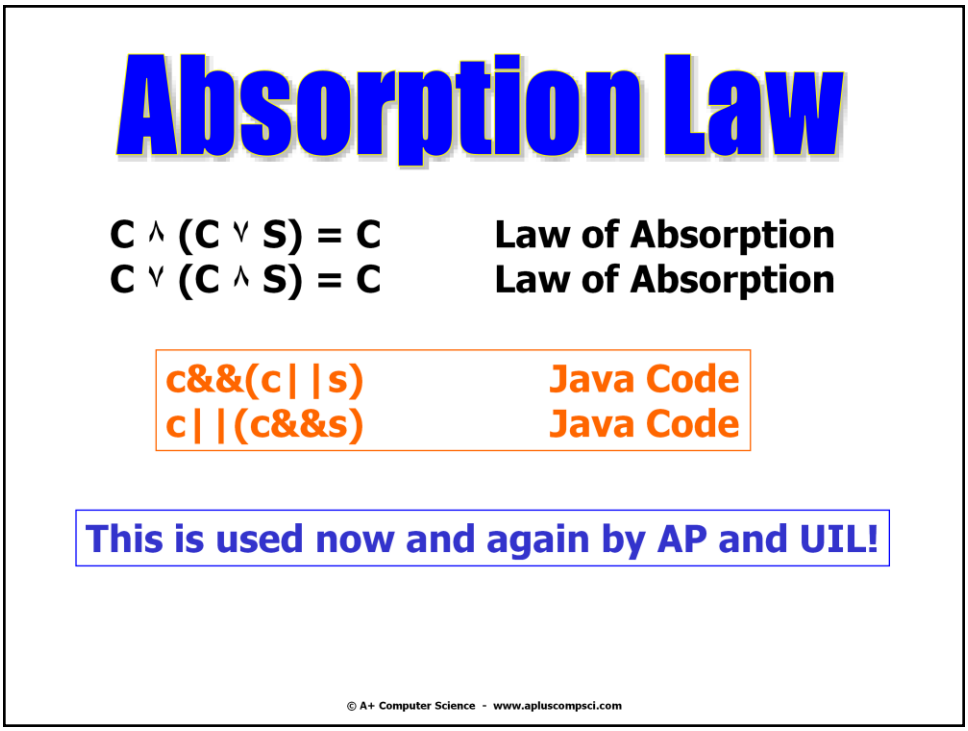


C	S	I
0	0	0
0	1	1
1	0	1
1	1	0


Open
logicalJ.java
© A+ Computer Science - www.apluscompsci.com

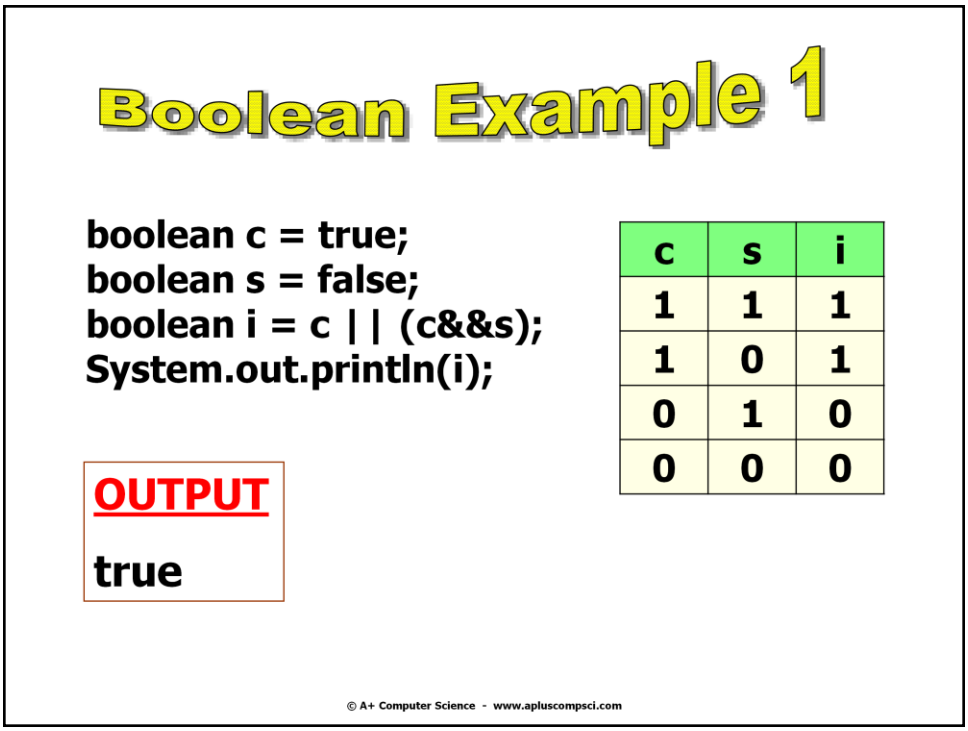
Open dowhile.java password.java

Common Boolean Laws



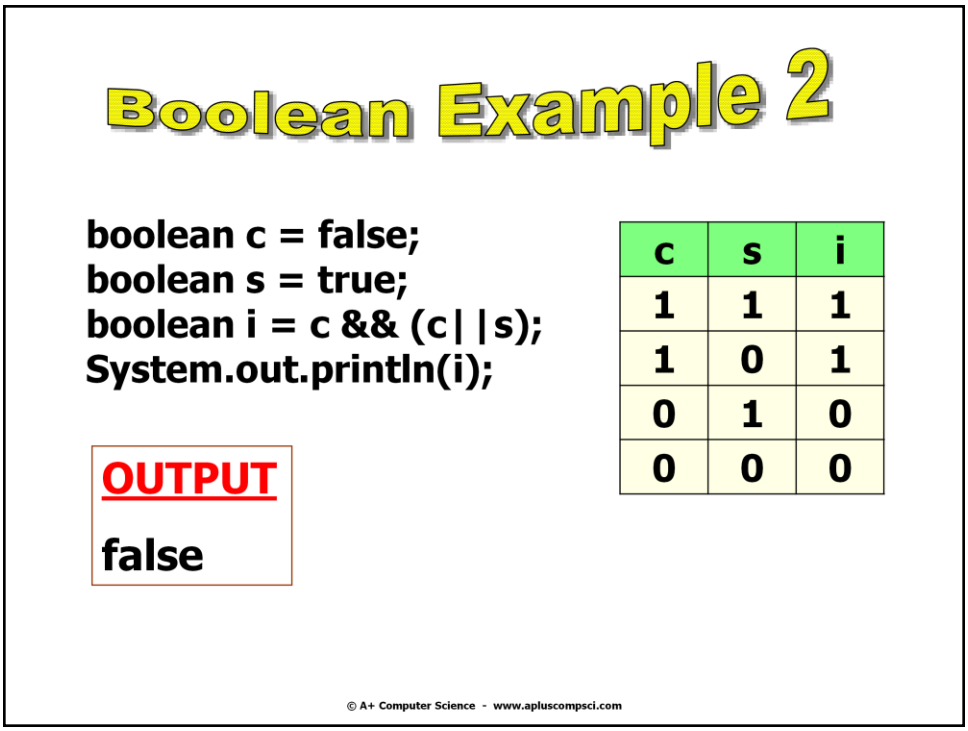
The law is absorption is essentially stating that C is the determinant of the expression.

C and (C or S) is equal to ((C and C) or (C and S)). C's value determines the value of the entire expression. If C is false, the expression is false. If C is true, the expression is true.



The law is absorption is essentially stating that C is the determinant of the expression.

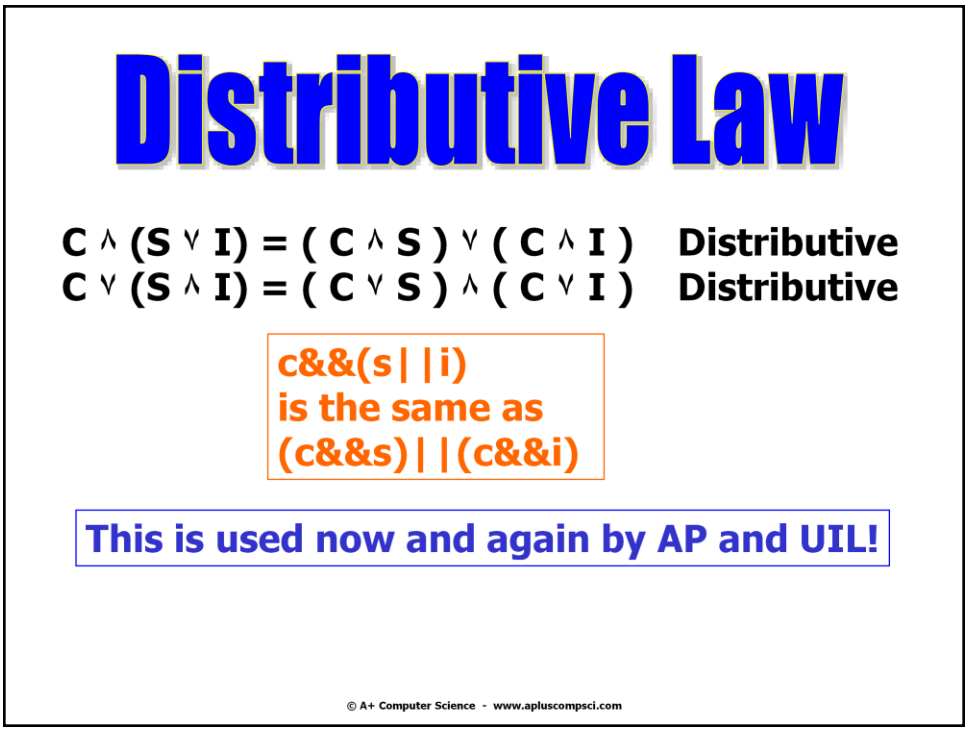
C or (C and S) is equal to ((C or C) and (C or S)). C's value determines the value of the entire expression. If C is false, the expression is false. If C is true, the expression is true.



The law is absorption is essentially stating that C is the determinant of the expression.

C and (C or S) is equal to ((C and C) or (C and S)). C's value determines the value of the entire expression. If C is false, the expression is false. If C is true, the expression is true.

open absorptionlaw.java

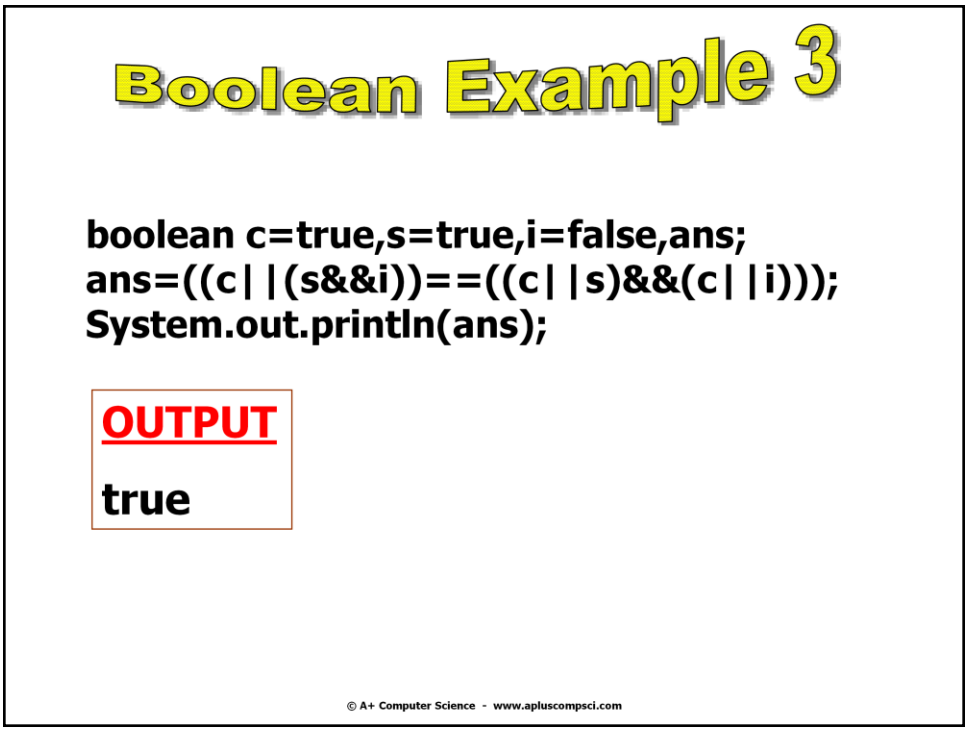


The Distributive Law is a basic algebraic law.

In the expressions above, distributing the term on the outside of parenthesis to each group inside the parenthesis is equivalent to the original expression.

C and (S or I) is equal to (C and S) or (C and I).

C or (S and I) is equal to (C or S) and (C or I).



The Distributive Law is a basic algebraic law.

In the expressions above, distributing the term on the outside of parenthesis to each group inside the parenthesis is equivalent to the original expression.

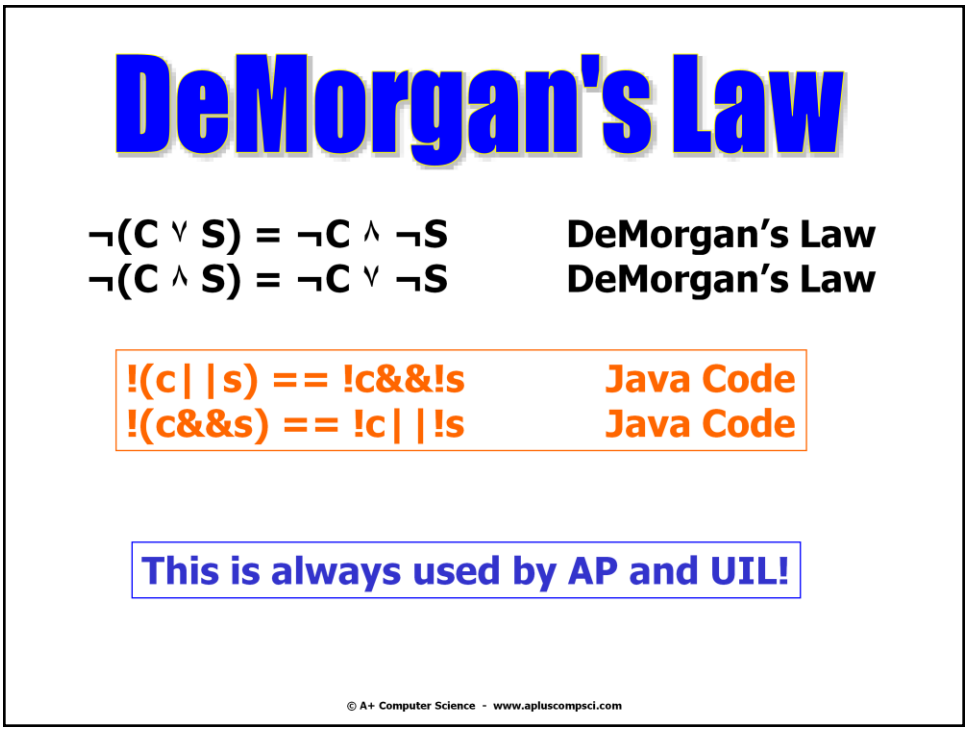
C and (S or I) is equal to (C and S) or (C and I).

C or (S and I) is equal to (C or S) and (C or I).


open
distributivelaw.java
© A+ Computer Science - www.apluscompsci.com


Start work
on the labs
© A+ Computer Science - www.apluscompsci.com

More Boolean Laws

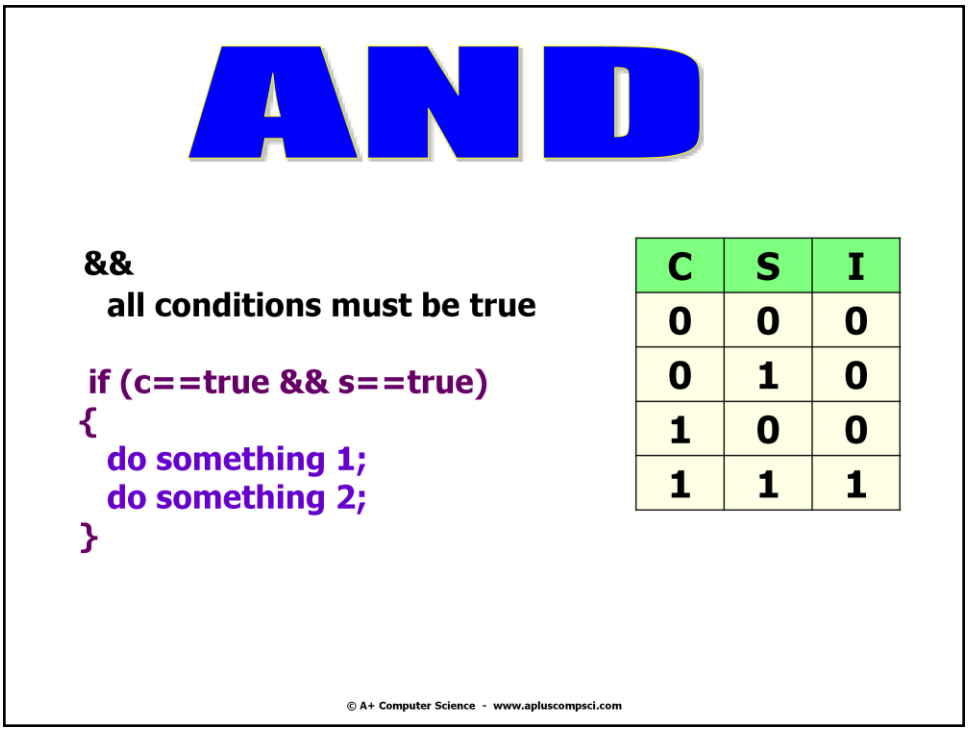


Demorgan's Law is useful to get a better picture of how the not affects the expression.

The easiest way to evaluate a negated expression is to simply evaluate the expression and then apply the negation.

If the expression is true, applying the negation makes the condition false.

If the expression is false, applying the negation makes the condition true.



And is used to see if all parts are true. In some languages, and is actually written as a word. In other languages, and is written as a symbol, like `&&` or `&`.

`&&` evaluates as true if all parts connected by `&&`s are true.

```
if(A and B)
```

This condition is true if A and B are both true. If either A or B is false, the condition is false as both parts must be true in order for the condition to be true.

Boolean Example 4

```
boolean c = true;
boolean s = true;
boolean i = !(c&&s);
System.out.println(i);
```

c	s	i
1	1	0
1	0	1
0	1	1
0	0	1

OUTPUT

false

The expression !(c and s) is equal to !c or !s.

First, evaluate (c and s). (c and s) is only true when both c and s are true. In all other cases, (c and s) is false.

Second, apply the !.

!(true) is false.

!(false) is true.

Evaluating the expression first and then applying the !, makes determining the overall value pretty straightforward.

Boolean Example 5

```
boolean c = false;
boolean s = true;
boolean i = !(c||s);
System.out.println(i);
```

c	s	i
1	1	0
1	0	0
0	1	0
0	0	1

OUTPUT

false

The expression !(c or s) is equal to !c and !s.

First, evaluate (c or s). (c or s) is true when either c or s is true. When c and s are both false, (c or s) is false.

Second, apply the !.

!(true) is false.

!(false) is true.

Evaluating the expression first and then applying the !, makes determining the overall value pretty straightforward.

open demorganslaw.java

Boolean Example 6

Which statement is represented by the truth table at right?

A. `i = !(c&&s)&&(c||s);`
B. `i = c||s&&s;`
C. `i = c&&s;`
D. `i != c&&s;`

c	s	i
0	0	0
0	1	1
1	0	1
1	1	1

b

Boolean Example 7

Which statement is represented by the truth table at right?

A. `i = !(c&&s) && c||s;`
B. `i = c||s&&s;`
C. `i = c&&s;`
D. `i = !(c&&s) &&(c||s);`

c	s	i
0	0	0
0	1	1
1	0	1
1	1	0

d

Short Circuit Evaluation

Short Circuit Evaluation

Java evaluates boolean expressions from left to right in most situations and stops the evaluation process once a condition is found that can complete the expression.

&& - and
|| - or

Short Circuit Evaluation || or

```
int total=9;
boolean flipper = false;

if(flipper || total>4)
{
  out.println("short");
}
out.println("check");
```

OUTPUT

```
short
check
```


If flipper is true, total>4 is never evaluated.

If flipper is false, total>4 is evaluated.

flipper is false, but total>4 so the condition is true. short is printed.

Short Circuit Evaluation || or

```
int total=2;
boolean flipper = true;

if(flipper || total>4)
{
  out.println("short");
}
out.println("check");
```

OUTPUT

```
short
check
```


If flipper is true, total>4 is never evaluated.

If flipper is false, total>4 is evaluated.

flipper is true so the condition is true. short is printed.

Short Circuit Evaluation || or

```
int total=2;
boolean flipper = false;

if(flipper || total>4)
{
  out.println("short");
}
out.println("check");
```

OUTPUT

check

If flipper is true, total>4 is never evaluated.

If flipper is false, total>4 is evaluated.

flipper is false and total>4 is false so the condition is false. short is not printed.

Short Circuit Evaluation || or

```
int total=9, num=13;

if (total<4 || ++num<15)
{
    out.println("short");
}
out.println(num);
```

OUTPUT

```
short
14
```


If total<4 is true, ++num<15 is never evaluated.

If total<4 is false, ++num<15 is evaluated.

total<4 is false so ++num<15 is evaluated. 14 is less than 15 so short is printed.

Short Circuit Evaluation && and

```
int total=9, num=13;

if (total>4 && ++num>15)
{
    out.println("short");
}
out.println(num);
```

OUTPUT

14

If total>4 is false, ++num<15 is never evaluated.

If total>4 is true, ++num<15 is evaluated.

total>4 is true so ++num<15 is evaluated. 14 is less than 15 so short is not printed.

Short Circuit Evaluation && and || or

```
int total=9, num=13;

if (total>4 ||++num>15 && total>0)
{
    out.println("short");
}
out.println(num);
```

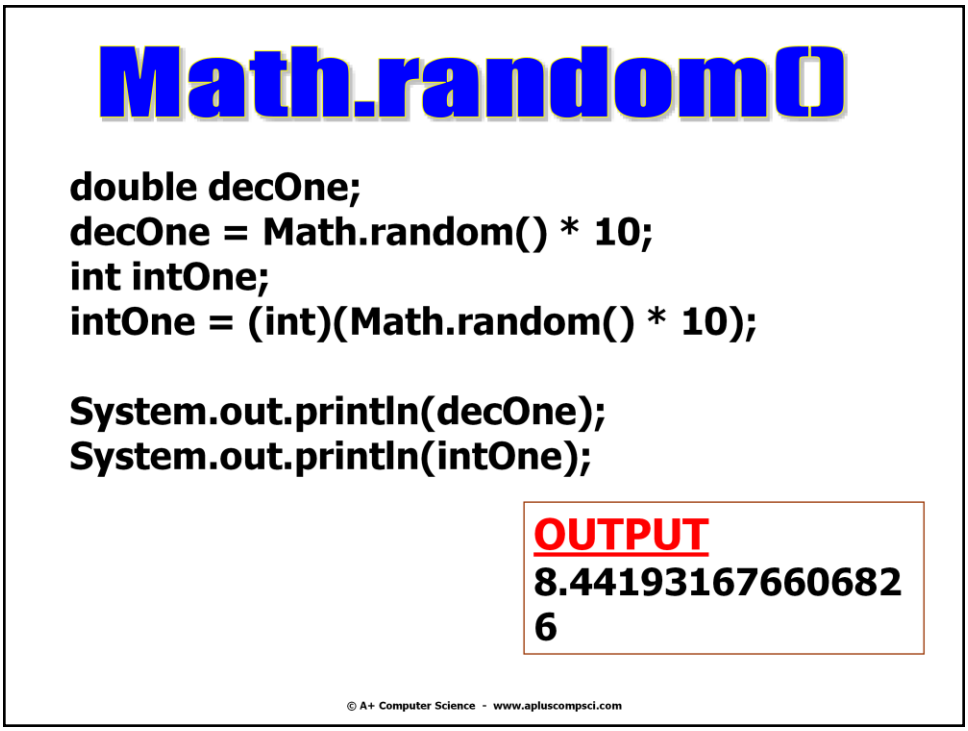
OUTPUT

```
short
13
```

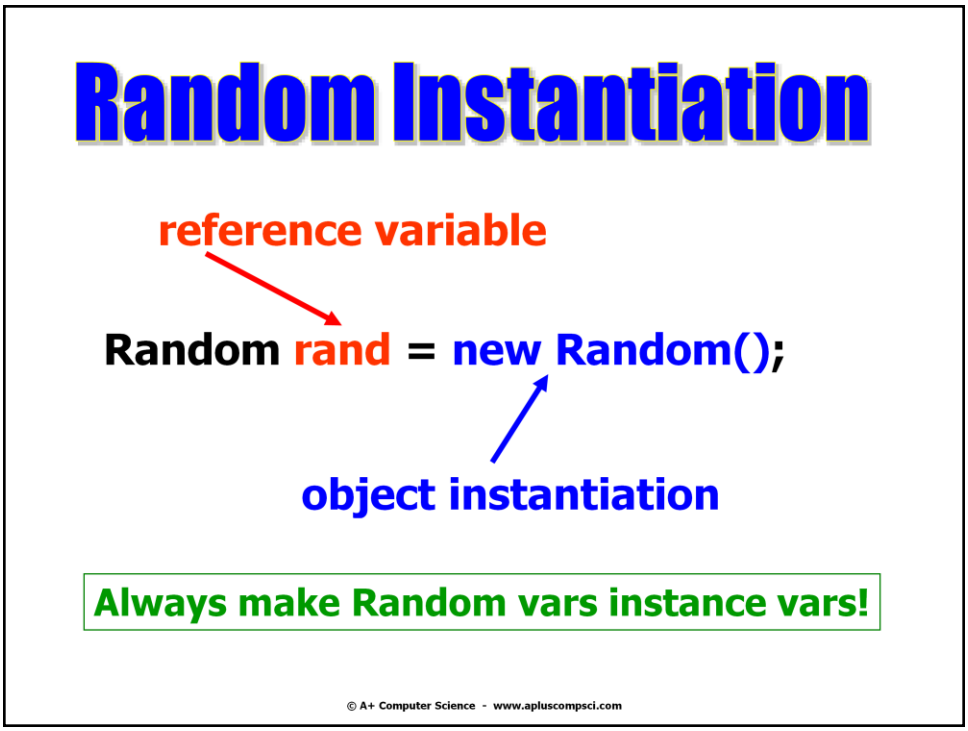
The && never happens!

open
shortone.java
shorttwo.java
shortthree.java
shortfour.java

Random Numbers



Math.random() returns a number between 0.0 and 1.0, not including 1.0.



Class Random contains many useful random methods. Math.random() can be used to generate the same results as the Random class methods.

Random
frequently used methods

Name	Use
nextInt(x)	returns a random int 0 to x(exclusive)
nextInt()	returns a random int MIN to MAX(exclusive)
nextDouble()	returns a random int 0.0 to 1.0(exclusive)

```
import  java.util.Random;
```

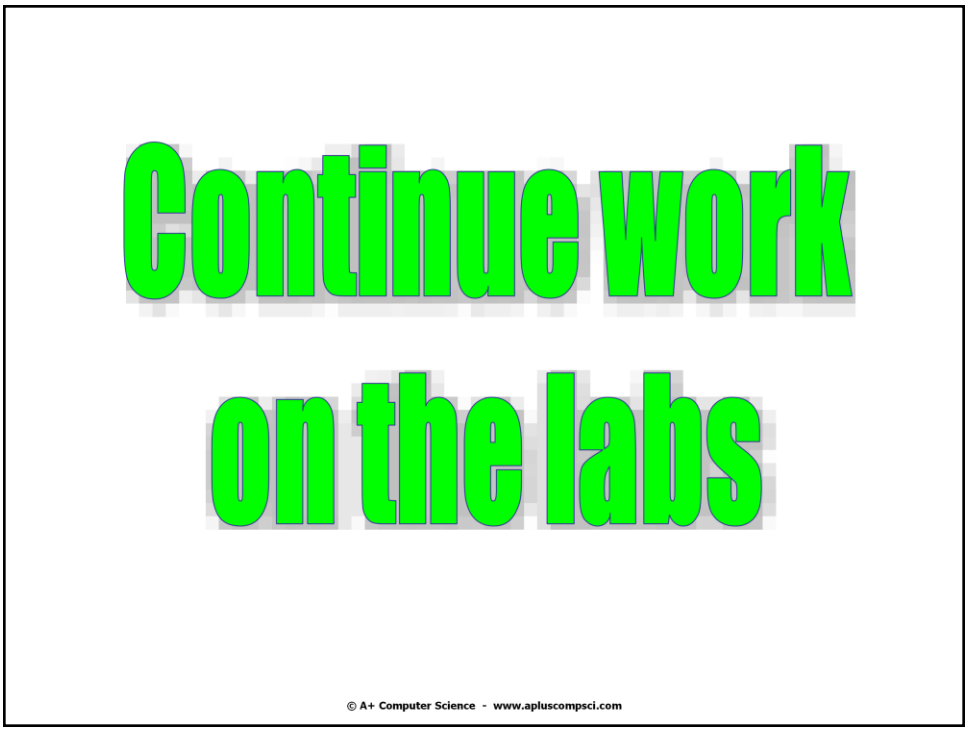
Random

```
Random rand = new Random();
int intOne = rand.nextInt(10);          //0-9
System.out.println(intOne);
intOne = rand.nextInt(50)+1;            //1-50
System.out.println(intOne);
intOne = rand.nextInt(20)+20;           //20-39
System.out.println(intOne);
```

OUTPUT

```
7
29
37
```

open randomone.java


Continue work
on the labs
© A+ Computer Science - www.apluscompsci.com