Lesson 38: Conditionals #2 (W11D3)

Balboa High School

Michael Ferraro

October 28, 2015

Do Now

In `Lesson38/DoNow.java`, write a method called `isDivisibleByFour()` that

- takes an `int`
- returns the `boolean` values `true` or `false` indicating whether the input value is evenly divisible by `4`.

Once finished, have `main()` call `isDivisibleByFour()` using the following `for()` loop:

```
for(int i = 0 ; i <= 100 ; i++) {
  if ( isDivisibleByFour(i) ) {
    System.out.println(i+" is divisible by 4");
  }
}
```

Aim

Students will learn how to perform basic logic operations (`and`, `or`, & `not`) in Java and how Java's order of operations works with respect to relational (e.g., <, <=, & !=) and logical operators.

Basic Logic: && (and)

How does "`and`" work in logic?

Basic Logic: && (and)

How does "and" work in logic?

- "I work really hard and I get good grades." $\rightarrow$ (p && q)

Basic Logic: && (and)

How does "and" work in logic?

- "I work really hard and I get good grades." → (p && q)
- Evaluate the value of the boolean expression (p && q) when...

Basic Logic: && (and)

How does "and" work in logic?

- "I work really hard and I get good grades." $\rightarrow$ (`p && q`)
- Evaluate the value of the `boolean expression` `( p && q )` when...
  1. $p =$ `true`, $q =$ `true`

Basic Logic: && (and)

How does "and" work in logic?

- "I work really hard and I get good grades." → (p && q)
- Evaluate the value of the boolean expression (p && q) when...
  1. $p =$ true, $q =$ true
  2. $p =$ true, $q =$ false

Basic Logic: && (and)

How does "and" work in logic?

- "I work really hard and I get good grades." → (p && q)
- Evaluate the value of the boolean expression (p && q) when...
  1. $p =$ true, $q =$ true
  2. $p =$ true, $q =$ false
  3. $p =$ false, $q =$ true

Basic Logic: && (and)

How does "and" work in logic?

- "I work really hard and I get good grades." → (p && q)
- Evaluate the value of the boolean expression (p && q) when...
  1. $p =$ true, $q =$ true
  2. $p =$ true, $q =$ false
  3. $p =$ false, $q =$ true
  4. $p =$ false, $q =$ false

Basic Logic: && (and)

How does "and" work in logic?

- "I work really hard and I get good grades." → (p && q)
- Evaluate the value of the boolean expression (p && q) when...
  1. $p =$ true, $q =$ true
  2. $p =$ true, $q =$ false
  3. $p =$ false, $q =$ true
  4. $p =$ false, $q =$ false
- && **is** true **when both operands (inputs) are** true**.**

Basic Logic: || (or)

How does "or" work in logic?

[1] or is also true when *both* operands are true — when using inclusive or

Basic Logic: || (or)

How does "or" work in logic?

- "I'm a slacker or I get good grades." $\rightarrow$ (p || q)

[1] or is also true when *both* operands are true — when using inclusive or

Basic Logic: || (or)

How does "or" work in logic?

- "I'm a slacker or I get good grades." $\rightarrow$ (p || q)
- Evaluate the value of the expression (p || q) when...

[1] or is also true when *both* operands are true — when using inclusive or

Basic Logic: || (or)

How does "or" work in logic?

- "I'm a slacker or I get good grades." $\rightarrow$ (`p` || `q`)
- Evaluate the value of the expression `( p || q )` when...
  1. $p =$ `true`, $q =$ `true`

[1] `or` is also true when *both* operands are true — when using `inclusive or`

Basic Logic: || (or)

How does "or" work in logic?

- "I'm a slacker or I get good grades." $\rightarrow$ (`p` || `q`)
- Evaluate the value of the expression `( p || q )` when...
  1. $p =$ `true`, $q =$ `true`
  2. $p =$ `true`, $q =$ `false`

[1] `or` is also true when *both* operands are true — when using `inclusive or`

Basic Logic: || (or)

How does "or" work in logic?

- "I'm a slacker or I get good grades." $\rightarrow$ (`p` || `q`)
- Evaluate the value of the expression `( p || q )` when...
  1. $p =$ `true`, $q =$ `true`
  2. $p =$ `true`, $q =$ `false`
  3. $p =$ `false`, $q =$ `true`

[1] `or` is also true when *both* operands are true — when using `inclusive or`

Basic Logic: || (or)

How does "or" work in logic?

- "I'm a slacker or I get good grades." $\rightarrow$ (`p` || `q`)
- Evaluate the value of the expression `( p || q )` when...
  1. $p =$ `true`, $q =$ `true`
  2. $p =$ `true`, $q =$ `false`
  3. $p =$ `false`, $q =$ `true`
  4. $p =$ `false`, $q =$ `false`

[1] `or` is also true when *both* operands are true — when using `inclusive or`

Basic Logic: || (or)

How does "or" work in logic?

- "I'm a slacker or I get good grades." → (`p` || `q`)
- Evaluate the value of the expression `( p || q )` when...
  1. $p =$ `true`, $q =$ `true`
  2. $p =$ `true`, $q =$ `false`
  3. $p =$ `false`, $q =$ `true`
  4. $p =$ `false`, $q =$ `false`
- || **is** `true` **when either operand (input) is** `true`.[1]

[1] `or` is also true when *both* operands are true — when using `inclusive or`

Examples

Predict the output of the following:

```
boolean p = true;  //I like to sing
boolean q = false; //I like to dance

if( p || q ) {
    System.out.println("Perform for us!");
}  else {
    System.out.println("Get off the stage!");
}
```

Examples

Predict the output of the following:

```
boolean p = false;  //I like to sing
boolean q = false;  //I like to dance

if( p || q ) {
    System.out.println("Perform for us!");
}  else {
    System.out.println("Get off the stage!");
}
```

Examples

Predict the output of the following:

```
boolean p = false;  //I like to sing
boolean q = true;   //I like to dance

if( p && q ) {
    System.out.println("Perform for us!");
}  else {
    System.out.println("Get off the stage!");
}
```

Examples

Predict the output of the following:

```
boolean p = true;    //I like to sing
boolean q = true;    //I like to dance

if( p && q ) {
    System.out.println("Perform for us!");
}  else {
    System.out.println("Get off the stage!");
}
```

Using `and` and `or`

- Write a `boolean` expression that evaluates to `true` when a number is between 0 and 9, inclusive...

Using `and` and `or`

- Write a `boolean expression` that evaluates to `true` when a number is between 0 and 9, inclusive...
- Restated: `true` when a number is ≥ 0 `and` ≤ 9

Using and and or

- Write a `boolean expression` that evaluates to `true` when a number is between 0 and 9, inclusive...
- Restated: `true` when a number is ≥ 0 and ≤ 9
- Solution: For a number, n,

```
( n >= 0 ) && ( n <= 9 )
```

Using and and or

- Write a `boolean expression` that evaluates to `true` when a number is between 0 and 9, inclusive...
- Restated: `true` when a number is ≥ 0 and ≤ 9
- Solution: For a number, n,

```
( n >= 0 ) && ( n <= 9 )
```

- As used in an `if()` statement:

```
if ( ( n >= 0 ) && ( n <= 9 ) ) {
    ...
}
```

Using `and` and `or`

- Write a `boolean` expression that evaluates to `true` when a number is less than 5 or greater than 6, exclusive[2]...

[2] Excluding the values 5 and 6, i.e., the number is strictly less than 5 or strictly greater than 6, and equal to neither.

Using `and` and `or`

- Write a `boolean` expression that evaluates to `true` when a number is less than 5 or greater than 6, exclusive[2]...
- Restated: `true` when a number is < 5 `or` > 6

[2] Excluding the values 5 and 6, i.e., the number is strictly less than 5 or strictly greater than 6, and equal to neither.

Using `and` and `or`

- Write a `boolean` expression that evaluates to `true` when a number is less than 5 or greater than 6, exclusive[2]...
- Restated: `true` when a number is $<$ 5 `or` $>$ 6
- Solution: For a number, n,

```
( n < 5 ) || ( n > 6 )
```

[2] Excluding the values 5 and 6, i.e., the number is strictly less than 5 or strictly greater than 6, and equal to neither.

Basic Logic: ! (not)

- ! is the *not*, or negation, operator
- It flips a boolean's truth value

```
boolean isEmpty = true; //glass is empty

if ( !  isEmpty ) {
   takeSip();
} else {
   refillGlass();
}
```

Basic Logic: ! (not)

- ! is the *not*, or negation, operator
- It flips a `boolean`'s truth value

```
boolean isEmpty = true; //glass is empty

if ( !  isEmpty ) {
   takeSip();
} else {
   refillGlass();
}
```

- ! is referred to as an *unary* operator

Basic Logic: ! (not)

- ! is the *not*, or negation, operator
- It flips a `boolean`'s truth value

```
boolean isEmpty = true; //glass is empty

if ( !  isEmpty ) {
   takeSip();
} else {
   refillGlass();
}
```

- ! is referred to as an *unary* operator
  - *unary* means that it takes one operand — the values whose truth value is to be flipped

Basic Logic: ! (not)

- ! is the *not*, or negation, operator
- It flips a `boolean`'s truth value

```
boolean isEmpty = true; //glass is empty

if ( !  isEmpty ) {
   takeSip();
} else {
   refillGlass();
}
```

- ! is referred to as an *unary* operator
  - *unary* means that it takes one operand — the values whose truth value is to be flipped
  - cf. *binary* operators like `+`, `-`, `*`, `/`, `%`, $\geq$, etc.

Mixing Logical Operators

Determine the value of c.

```
boolean a = true;
boolean b = false;

boolean c = ( a && !b );
```

Mixing Logical Operators

Determine the value of c.

```
boolean a = true;
boolean b = true;

boolean c = ( !a || !b );
```

Mixing Logical Operators

Are things any different now?

```
boolean a = true;
boolean b = true;

//boolean c = ( !a || !b );
boolean c = !( a && b );
```

Mixing Logical Operators

Are things any different now?

```
boolean a = true;
boolean b = true;

//boolean c = ( !a || !b );
boolean c = !( a && b );
```

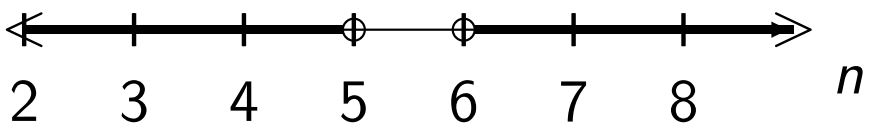
We'll revisit this shortly...

Revisiting the || Example

- Write a `boolean` expression that evaluates to `true` when a number is less than 5 or greater than 6, exclusive: `( n < 5 ) || ( n > 6 )`

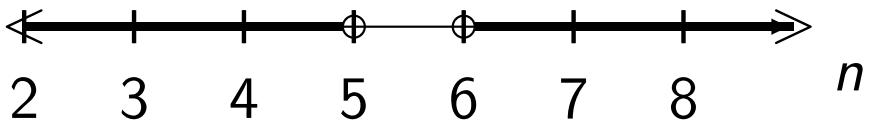
Revisiting the || Example

- Write a `boolean` expression that evaluates to `true` when a number is less than 5 or greater than 6, exclusive: `( n < 5 ) || ( n > 6 )`
- In other words, `true` when n is



Revisiting the || Example

- Write a `boolean` expression that evaluates to `true` when a number is less than 5 or greater than 6, exclusive: `( n < 5 ) || ( n > 6 )`
- In other words, `true` when *n* is



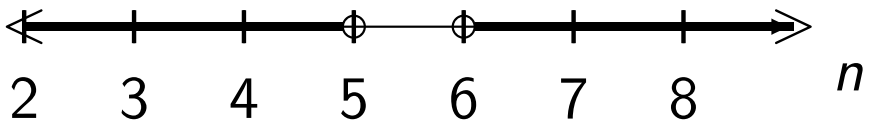
- Isn't it reasonable to say the following instead?

  It is not true that n is between 5 and 6, inclusive

  `! ( ( n >= 5 ) && ( n <= 6 ) )`

Revisiting the || Example

- Write a `boolean` expression that evaluates to `true` when a number is less than 5 or greater than 6, exclusive: `( n < 5 ) || ( n > 6 )`
- In other words, `true` when *n* is



- Isn't it reasonable to say the following instead?

  It is not true that n is between 5 and 6, inclusive

  `! ( ( n >= 5 ) && ( n <= 6 ) )`

- ∴ both `boolean` expressions are logically equivalent

DeMorgan's Rules

- The rules are fully described in PS #6, §3.1.2

DeMorgan's Rules

- The rules are fully described in PS #6, §3.1.2
- Summary: Distribute the negation, swap the `and`/`or`

DeMorgan's Rules

- The rules are fully described in PS #6, §3.1.2
- Summary: Distribute the negation, swap the `and`/`or`
  - `! ( A || B )` $\equiv$ `!A && !B`

DeMorgan's Rules

- The rules are fully described in PS #6, §3.1.2
- Summary: Distribute the negation, swap the `and`/`or`
  - `! ( A || B )` $\equiv$ `!A && !B`
  - `! ( A && B )` $\equiv$ `!A || !B`

DeMorgan's Rules

- The rules are fully described in PS #6, §3.1.2
- Summary: Distribute the negation, swap the `and`/`or`
  - `! ( A || B )` $\equiv$ `!A && !B`
  - `! ( A && B )` $\equiv$ `!A || !B`
- Example:

DeMorgan's Rules

- The rules are fully described in PS #6, §3.1.2
- Summary: Distribute the negation, swap the `and`/`or`
  - `! ( A || B )` $\equiv$ `!A && !B`
  - `! ( A && B )` $\equiv$ `!A || !B`
- Example:

```
!( a && b )
```

DeMorgan's Rules

- The rules are fully described in PS #6, §3.1.2
- Summary: Distribute the negation, swap the `and`/`or`
  - `! ( A || B )` $\equiv$ `!A && !B`
  - `! ( A && B )` $\equiv$ `!A || !B`
- Example:

```
!( a && b )

!a || !b
```

Second DeMorgan's Example

```
!( ( n >= 5 ) && ( n <= 6 ) )
```

Second DeMorgan's Example

```
!( ( n >= 5 ) && ( n <= 6 ) )

  !( n >= 5 ) || !( n <= 6 )
```

Second DeMorgan's Example

```
!( ( n >= 5 ) && ( n <= 6 ) )

  !( n >= 5 ) || !( n <= 6 )

!( n >= 5 ) || !( n <= 6 )
```

Second DeMorgan's Example

```
!( ( n >= 5 ) && ( n <= 6 ) )

  !( n >= 5 ) || !( n <= 6 )

  !( n >= 5 ) || !( n <= 6 )

   ( n < 5 )  ||   ( n > 6 )
```

Java Order of Operations

When do we need `()`s? When don't we?

Java Order of Operations

Priority	Operator Type	Operators
highest	**P**EMDAS	()
	Unary	!, −, *casts*, ++, −−
	P**E**MDAS	x^y
	PE**MD**AS	×, ÷ (*also modulo*, %)
	PEMD**AS**	+, −
	Relational Operators	<, <=, >, >=, ==, ! =
	Logical and	&&
lowest	Logical or	||

Java Order of Operations

Let's see if we can simplify some earlier examples. . .

Java Order of Operations

Let's see if we can simplify some earlier examples...

- `( n >= 0 ) && ( n <= 9 )` ← are ()s needed?

Java Order of Operations

Let's see if we can simplify some earlier examples...

- `( n >= 0 ) && ( n <= 9 )` ← are ()s needed?

```
n >= 0 && n <= 9
```

Java Order of Operations

Let's see if we can simplify some earlier examples...

- `( n >= 0 ) && ( n <= 9 )` ← are ()s needed?

  `n >= 0 && n <= 9`

- `( n < 5 ) || ( n > 6 )` ← are ()s needed?

Java Order of Operations

Let's see if we can simplify some earlier examples...

- `( n >= 0 ) && ( n <= 9 )` ← are ()s needed?

  `n >= 0 && n <= 9`

- `( n < 5 ) || ( n > 6 )` ← are ()s needed?

  `n < 5 || n > 6`

HW

- Complete §§1-3.2 of PS #6, inclusive
- The reading for Litvin Ch. 6 is available here.