Software Development (cs2500)

Lecture 9: Making Decisions (Continued)

M. R. C. van Dongen

October 11, 2013

Boolean Expressions

! ⟨`expr`⟩: `true` iff ⟨`expr`⟩ is `false`.

⟨`fst`⟩ `&&` ⟨`snd`⟩: `true` iff ⟨`fst`⟩ and ⟨`snd`⟩ are `true`.

⟨`fst`⟩ `||` ⟨`snd`⟩: `true` iff at least one of ⟨`fst`⟩ and ⟨`snd`⟩ are `true`.

Cayley Table: !

!
false
true

Cayley Table: !

!	
false	true
true	

Cayley Table: !

!	
false	true
true	

Cayley Table: !

!	
false	true
true	false

Cayley Table: !

!	
`false`	`true`
`true`	`false`

- Pretend `false` is represented as `0` and `true` as `1`.
- Then `!bool == 1 - bool` for any `boolean bool`.

Cayley Table: &&

&&	false	true
false		
true		

Cayley Table: &&

&&	false	true
false		
true		

Cayley Table: &&

&&	false	true
false	false	
true		

Cayley Table: &&

&&	false	true
false	false	
true		

Cayley Table: &&

&&	false	true
false	false	false
true		

Cayley Table: &&

&&	false	true
false	false	false
true		

Cayley Table: &&

&&	false	true
false	false	false
true	false	

Cayley Table: &&

&&	false	true
false	false	false
true	false	

Cayley Table: &&

&&	false	true
false	false	false
true	false	true

Cayley Table: &&

Behaves as "Minumum"

&&	false	true
false	false	false
true	false	true

Cayley Table: ||

`||`	`false`	`true`
`false`		
`true`		

Cayley Table: `||`

`||`	`false`	`true`
`false`		
`true`		

Cayley Table: ||

`||`	`false`	`true`
`false`	`false`	
`true`		

Cayley Table: ||

`||`	`false`	`true`
`false`	`false`	
`true`		

Cayley Table: ||

`||`	`false`	`true`
`false`	`false`	`true`
`true`		

Cayley Table: `||`

`||`	`false`	`true`
`false`	`false`	`true`
`true`		

Cayley Table: `||`

`||`	`false`	`true`
`false`	`false`	`true`
`true`	`true`	

Cayley Table: ||

`||`	`false`	`true`
`false`	`false`	`true`
`true`	`true`	

Cayley Table: ||

`||`	`false`	`true`
`false`	`false`	`true`
`true`	`true`	`true`

Cayley Table: ||

Behaves as “Maximum”

`||`	`false`	`true`
`false`	`false`	`true`
`true`	`true`	`true`

Example (!)

Java

```
if (!(temperatureIndegrees >= FREEZING_TEMPERATURE_OF_WATER)) {
    System.out.println( "It's freezing." );
}
```

Example (!)

Java

```
if (!(temperatureIndegrees >= FREEZING_TEMPERATURE_OF_WATER)) {
    System.out.println( "It's freezing." );
}
```

Example (||)

Java

```
private static final double KILOGRAM = 1.0;
private static final double EPSILON = 1OE-3 * KILOGRAM;
⋮
final double firstWeight = ...;
final double secondWeight = ...;
final double difference = firstWeight - secondWeight;

if ((difference < -EPSILON) || (EPSILON < difference)) {
    System.out.println( "The weights are not in the same range." );
}
```

Example (&&)

Java

```java
final Person person = new Person( );

if ((temperatureInDegrees < FREEZING_TEMPERATURE_OF_WATER)
        && (person.isOutSide( ))) {
    person.shiver( );
}
```

Example (| | and &&)

Java

```
final Person person = new Person( );

if ((person.isOutside( ))
        && ((temperatureInDegrees < 0.0)
                || ((person.isIrish( )) && (temperatureInDegrees < 15.0)))) {
    person.complain( "It's freezing." );
}
```

Distributivite Laws

- Let D be a domain (of values/booleans).
- Let $\oplus_1$ and $\oplus_2$ be operations on D.
- If $a \oplus_1 (b \oplus_2 c) = (a \oplus_1 b) \oplus_2 (a \oplus_1 c)$ for any a, b, and c in D.
- Then $\oplus_1$ is said to *distribute over* $\oplus_2$.

Distributivite Laws

- Let D be a domain (of values/booleans).
- Let $\oplus_1$ and $\oplus_2$ be operations on D.
- If $a \oplus_1 (b \oplus_2 c) = (a \oplus_1 b) \oplus_2 (a \oplus_1 c)$ for any a, b, and c in D.
- Then $\oplus_1$ is said to *distribute over* $\oplus_2$.
- Multiplication distributes over addition:
  - `a * (b + c) == (a * b) + (a * c)`.

Distributivite Laws

- Let D be a domain (of values/booleans).
- Let $\oplus_1$ and $\oplus_2$ be operations on D.
- If $a \oplus_1 (b \oplus_2 c) = (a \oplus_1 b) \oplus_2 (a \oplus_1 c)$ for any a, b, and c in D.
- Then $\oplus_1$ is said to *distribute over* $\oplus_2$.
- Multiplication distributes over addition:
  - `a * (b + c) == (a * b) + (a * c)`.
- Multiplication distributes over subtraction:
  - `a * (b - c) == (a * b) - (a * c)`.

Distributivite Laws

- Let D be a domain (of values/booleans).
- Let $\oplus_1$ and $\oplus_2$ be operations on D.
- If $a \oplus_1 (b \oplus_2 c) = (a \oplus_1 b) \oplus_2 (a \oplus_1 c)$ for any a, b, and c in D.
- Then $\oplus_1$ is said to *distribute over* $\oplus_2$.
- Multiplication distributes over addition:
  - `a * (b + c) == (a * b) + (a * c).`
- Multiplication distributes over subtraction:
  - `a * (b - c) == (a * b) - (a * c).`
- Conjuction (&&) distributes over itself:
  - `a && (b && c) == (a && b) && (a && c).`

Distributivite Laws

- Let D be a domain (of values/booleans).
- Let $\oplus_1$ and $\oplus_2$ be operations on D.
- If $a \oplus_1 (b \oplus_2 c) = (a \oplus_1 b) \oplus_2 (a \oplus_1 c)$ for any a, b, and c in D.
- Then $\oplus_1$ is said to *distribute over* $\oplus_2$.
- Multiplication distributes over addition:
  - `a * (b + c) == (a * b) + (a * c).`
- Multiplication distributes over subtraction:
  - `a * (b - c) == (a * b) - (a * c).`
- Conjuction (&&) distributes over itself:
  - `a && (b && c) == (a && b) && (a && c).`
- Conjuction distributes over disjunction (||):
  - `a && (b || c) == (a && b) || (a && c).`

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false		
false	false	true		
false	true	false		
false	true	true		
true	false	false		
true	false	true		
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	
false	false	true		
false	true	false		
false	true	true		
true	false	false		
true	false	true		
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true		
false	true	false		
false	true	true		
true	false	false		
true	false	true		
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true		
false	true	false		
false	true	true		
true	false	false		
true	false	true		
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	
false	true	false		
false	true	true		
true	false	false		
true	false	true		
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false		
false	true	true		
true	false	false		
true	false	true		
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false		
false	true	true		
true	false	false		
true	false	true		
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false	false	
false	true	true		
true	false	false		
true	false	true		
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true		
true	false	false		
true	false	true		
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true		
true	false	false		
true	false	true		
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true	false	
true	false	false		
true	false	true		
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	`a && (b || c)`	`(a && b) || (a && c)`
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true	false	false
true	false	false		
true	false	true		
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true	false	false
true	false	false		
true	false	true		
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true	false	false
true	false	false	false	
true	false	true		
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true	false	false
true	false	false	false	false
true	false	true		
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true	false	false
true	false	false	false	false
true	false	true		
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true	false	false
true	false	false	false	false
true	false	true	true	
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true	false	false
true	false	false	false	false
true	false	true	true	true
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true	false	false
true	false	false	false	false
true	false	true	true	true
true	true	false		
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true	false	false
true	false	false	false	false
true	false	true	true	true
true	true	false	true	
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true	false	false
true	false	false	false	false
true	false	true	true	true
true	true	false	true	true
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true	false	false
true	false	false	false	false
true	false	true	true	true
true	true	false	true	true
true	true	true		

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true	false	false
true	false	false	false	false
true	false	true	true	true
true	true	false	true	true
true	true	true	true	

Proof of Claim of Distributivity

a	b	c	a && (b || c)	(a && b) || (a && c)
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true	false	false
true	false	false	false	false
true	false	true	true	true
true	true	false	true	true
true	true	true	true	true

Short-Circuit Evaluation

- When one of the operands of `&&` is `false`, the result is `false`.
- When one of the operands of `||` is `true`, the result is `true`.
- When this happens, we say that the operand *forecasts* the result.[1]
- `Java` exploits result forecasting to save time.
- When the first operand forecasts the result, `Java` doesn't evaluate the second operand.
- This is called *short-circuit* evaluation.
- Using short-circuit evaluation doesn't make any difference.

[1] The notion of forecasting the result is not standard.

Short-Circuit Evaluation

- When one of the operands of `&&` is `false`, the result is `false`.
- When one of the operands of `||` is `true`, the result is `true`.
- When this happens, we say that the operand *forecasts* the result.[1]
- `Java` exploits result forecasting to save time.
- When the first operand forecasts the result, `Java` doesn't evaluate the second operand.
- This is called *short-circuit* evaluation.
- Using short-circuit evaluation doesn't make any difference.

[1]The notion of forecasting the result is not standard.

Short-Circuit Evaluation

- When one of the operands of `&&` is `false`, the result is `false`.
- When one of the operands of `||` is `true`, the result is `true`.
- When this happens, we say that the operand *forecasts* the result.[1]
- `Java` exploits result forecasting to save time.
- When the first operand forecasts the result, `Java` doesn't evaluate the second operand.
- This is called *short-circuit* evaluation.
- Using short-circuit evaluation doesn't make any difference.

[1] The notion of forecasting the result is not standard.

Short-Circuit Evaluation

- When one of the operands of `&&` is `false`, the result is `false`.
- When one of the operands of `||` is `true`, the result is `true`.
- When this happens, we say that the operand *forecasts* the result.[1]
- `Java` exploits result forecasting to save time.
- When the first operand forecasts the result, `Java` doesn't evaluate the second operand.
- This is called *short-circuit* evaluation.
- Using short-circuit evaluation doesn't make any difference.

[1] The notion of forecasting the result is not standard.

Short-Circuit Evaluation

- When one of the operands of `&&` is `false`, the result is `false`.
- When one of the operands of `||` is `true`, the result is `true`.
- When this happens, we say that the operand *forecasts* the result.[1]
- `Java` exploits result forecasting to save time.
- When the first operand forecasts the result, `Java` doesn't evaluate the second operand.
- This is called *short-circuit* evaluation.
- Using short-circuit evaluation doesn't make any difference.

[1] The notion of forecasting the result is not standard.

Short-Circuit Evaluation

- When one of the operands of `&&` is `false`, the result is `false`.
- When one of the operands of `||` is `true`, the result is `true`.
- When this happens, we say that the operand *forecasts* the result.[1]
- `Java` exploits result forecasting to save time.
- When the first operand forecasts the result, `Java` doesn't evaluate the second operand.
- This is called *short-circuit* evaluation.
- Using short-circuit evaluation doesn't make any difference,

Except when there are side effects.

[1]The notion of forecasting the result is not standard.

Example

Prints ?

Java

```
public class ShortCircuitEvaluation {
    public static void main( String[] args ) {
        final boolean tt = booleanCall( true, 1 )  && booleanCall( true, 2 );
        final boolean tf = booleanCall( true, 3 )  && booleanCall( false, 4 );
        final boolean ft = booleanCall( false, 5 ) && booleanCall( true, 6 );
        final boolean ff = booleanCall( false, 7 ) && booleanCall( false, 8 );
    }

    private static boolean booleanCall( final boolean result, final int number )  {
        System.out.print( number );
        return result;
    }
}
```

Example

Prints 1

Java

```
public class ShortCircuitEvaluation {
    public static void main( String[] args ) {
        final boolean tt = booleanCall( true, 1 )  && booleanCall( true, 2 );
        final boolean tf = booleanCall( true, 3 )  && booleanCall( false, 4 );
        final boolean ft = booleanCall( false, 5 ) && booleanCall( true, 6 );
        final boolean ff = booleanCall( false, 7 ) && booleanCall( false, 8 );
    }

    private static boolean booleanCall( final boolean result, final int number )  {
        System.out.print( number );
        return result;
    }
}
```

Example

Prints 1

Java

```
public class ShortCircuitEvaluation {
    public static void main( String[] args ) {
        final boolean tt = booleanCall( true, 1 )  && booleanCall( true, 2 );
        final boolean tf = booleanCall( true, 3 )  && booleanCall( false, 4 );
        final boolean ft = booleanCall( false, 5 ) && booleanCall( true, 6 );
        final boolean ff = booleanCall( false, 7 ) && booleanCall( false, 8 );
    }

    private static boolean booleanCall( final boolean result, final int number )  {
        System.out.print( number );
        return result;
    }
}
```

Example

Prints 12

Java

```java
public class ShortCircuitEvaluation {
    public static void main( String[] args ) {
        final boolean tt = booleanCall( true, 1 )  && booleanCall( true, 2 );
        final boolean tf = booleanCall( true, 3 )  && booleanCall( false, 4 );
        final boolean ft = booleanCall( false, 5 ) && booleanCall( true, 6 );
        final boolean ff = booleanCall( false, 7 ) && booleanCall( false, 8 );
    }

    private static boolean booleanCall( final boolean result, final int number )  {
        System.out.print( number );
        return result;
    }
}
```

Example

Prints 12

Java

```
public class ShortCircuitEvaluation {
    public static void main( String[] args ) {
        final boolean tt = booleanCall( true, 1 )  && booleanCall( true, 2 );
        final boolean tf = booleanCall( true, 3 )  && booleanCall( false, 4 );
        final boolean ft = booleanCall( false, 5 ) && booleanCall( true, 6 );
        final boolean ff = booleanCall( false, 7 ) && booleanCall( false, 8 );
    }

    private static boolean booleanCall( final boolean result, final int number )  {
        System.out.print( number );
        return result;
    }
}
```

Example

Prints 123

Java

```
public class ShortCircuitEvaluation {
    public static void main( String[] args ) {
        final boolean tt = booleanCall( true, 1 )  && booleanCall( true, 2 );
        final boolean tf = booleanCall( true, 3 )  && booleanCall( false, 4 );
        final boolean ft = booleanCall( false, 5 ) && booleanCall( true, 6 );
        final boolean ff = booleanCall( false, 7 ) && booleanCall( false, 8 );
    }

    private static boolean booleanCall( final boolean result, final int number )  {
        System.out.print( number );
        return result;
    }
}
```

Example

Prints 123

Java

```
public class ShortCircuitEvaluation {
    public static void main( String[] args ) {
        final boolean tt = booleanCall( true, 1 )  && booleanCall( true, 2 );
        final boolean tf = booleanCall( true, 3 )  && booleanCall( false, 4 );
        final boolean ft = booleanCall( false, 5 ) && booleanCall( true, 6 );
        final boolean ff = booleanCall( false, 7 ) && booleanCall( false, 8 );
    }

    private static boolean booleanCall( final boolean result, final int number )  {
        System.out.print( number );
        return result;
    }
}
```

Example

Prints 1234

Java

```
public class ShortCircuitEvaluation {
    public static void main( String[] args ) {
        final boolean tt = booleanCall( true, 1 )  && booleanCall( true, 2 );
        final boolean tf = booleanCall( true, 3 )  && booleanCall( false, 4 );
        final boolean ft = booleanCall( false, 5 ) && booleanCall( true, 6 );
        final boolean ff = booleanCall( false, 7 ) && booleanCall( false, 8 );
    }

    private static boolean booleanCall( final boolean result, final int number )  {
        System.out.print( number );
        return result;
    }
}
```

Example

Prints 1234

Java

```
public class ShortCircuitEvaluation {
    public static void main( String[] args ) {
        final boolean tt = booleanCall( true, 1 )  && booleanCall( true, 2 );
        final boolean tf = booleanCall( true, 3 )  && booleanCall( false, 4 );
        final boolean ft = booleanCall( false, 5 ) && booleanCall( true, 6 );
        final boolean ff = booleanCall( false, 7 ) && booleanCall( false, 8 );
    }

    private static boolean booleanCall( final boolean result, final int number )  {
        System.out.print( number );
        return result;
    }
}
```

Example

Prints 12345

Java

```java
public class ShortCircuitEvaluation {
    public static void main( String[] args ) {
        final boolean tt = booleanCall( true, 1 )  && booleanCall( true, 2 );
        final boolean tf = booleanCall( true, 3 )  && booleanCall( false, 4 );
        final boolean ft = booleanCall( false, 5 ) && booleanCall( true, 6 );
        final boolean ff = booleanCall( false, 7 ) && booleanCall( false, 8 );
    }

    private static boolean booleanCall( final boolean result, final int number )  {
        System.out.print( number );
        return result;
    }
}
```

Example

Prints 12345

Java

```
public class ShortCircuitEvaluation {
    public static void main( String[] args ) {
        final boolean tt = booleanCall( true, 1 )  && booleanCall( true, 2 );
        final boolean tf = booleanCall( true, 3 )  && booleanCall( false, 4 );
        final boolean ft = booleanCall( false, 5 ) && booleanCall( true, 6 );
        final boolean ff = booleanCall( false, 7 ) && booleanCall( false, 8 );
    }

    private static boolean booleanCall( final boolean result, final int number )  {
        System.out.print( number );
        return result;
    }
}
```

Example

Prints 123457

Java

```
public class ShortCircuitEvaluation {
    public static void main( String[] args ) {
        final boolean tt = booleanCall( true, 1 )  && booleanCall( true, 2 );
        final boolean tf = booleanCall( true, 3 )  && booleanCall( false, 4 );
        final boolean ft = booleanCall( false, 5 ) && booleanCall( true, 6 );
        final boolean ff = booleanCall( false, 7 ) && booleanCall( false, 8 );
    }

    private static boolean booleanCall( final boolean result, final int number ) {
        System.out.print( number );
        return result;
    }
}
```

Example

Prints 123457

Java

```
public class ShortCircuitEvaluation {
    public static void main( String[] args ) {
        final boolean tt = booleanCall( true, 1 )  && booleanCall( true, 2 );
        final boolean tf = booleanCall( true, 3 )  && booleanCall( false, 4 );
        final boolean ft = booleanCall( false, 5 ) && booleanCall( true, 6 );
        final boolean ff = booleanCall( false, 7 ) && booleanCall( false, 8 );
    }

    private static boolean booleanCall( final boolean result, final int number )  {
        System.out.print( number );
        return result;
    }
}
```

De Morgan's Laws

- Many expressions combine negation (`!`) with conjunction (`&&`) or disjunction (`||`).
- *De Morgan's Law* explains how to simplifies these expressions.
- If `A` and `B` are `boolean` values, then:
  1. `!(A && B)` is equal to `(!A) || (!B)`; and
  2. `!(A || B)` is equal to `(!A) && (!B)`.

De Morgan's Laws

If it's not `true` that both `A` and `B` are `true`, at least one of them must be `false`

- Many expressions combine negation (`!`) with conjunction (&&) or disjunction (`||`).
- *De Morgan's Law* explains how to simplifies these expressions.
- If `A` and `B` are `boolean` values, then:
  1. `!(A && B)` is equal to `(!A) || (!B)`; and
  2. `!(A || B)` is equal to `(!A) && (!B)`.

De Morgan's Laws

If it's not `true` that at least one of `A` and `B` is `true`, both must be `false`

- Many expressions combine negation (`!`) with conjunction (`&&`) or disjunction (`||`).
- *De Morgan's Law* explains how to simplifies these expressions.
- If `A` and `B` are `boolean` values, then:
  1. `!(A && B)` is equal to `(!A) || (!B)`; and
  2. `!(A || B)` is equal to `(!A) && (!B)`.

Confusing

Assume that ⟨condition⟩ is Side-effect Free

Java

```
if (⟨condition⟩1) {
    // ⟨condition⟩1 is true.
    ⟨statements⟩1
} else if (!⟨condition⟩1 && ⟨condition⟩2) {
    // ⟨condition⟩1 is false and
    // ⟨condition⟩2 is true.
    ⟨statements⟩2
} else if (!⟨condition⟩1 && !⟨condition⟩2) {
    // ⟨condition⟩1 is false and
    // ⟨condition⟩2 is false.
    ⟨statements⟩3
}
```

Confusing

Assume that ⟨condition⟩ is Side-effect Free

Java

```
if (⟨condition⟩₁) {
    // ⟨condition⟩₁ is true.
    ⟨statements⟩₁
} else if (⟨condition⟩₂) {
    // ⟨condition⟩₁ is false and
    // ⟨condition⟩₂ is true.
    ⟨statements⟩₂
} else {
    // ⟨condition⟩₁ is false and
    // ⟨condition⟩₂ is false.
    ⟨statements⟩₃
}
```

Partial Operator Precedence Table

Description	Operator	Associativity
post-*crement	⟨lvalue⟩++ and ⟨lvalue⟩--	left
pre-*crement, unary	++⟨lvalue⟩, --⟨lvalue⟩, +⟨expr⟩, -⟨expr⟩, and !⟨expr⟩	right
object creation	new	right
multiplicative	*, /, and %	left
additive	+ and -	right
relational	<, >, <=, and >=	left
equality	== and !=	left
logical and	&&	left
logical or	||	left
ternary	⟨condition⟩ ? ⟨expr⟩ : ⟨expr⟩	right
assignment	=, +=, -=, *=, /=, and %=	right

Test for Oddness

ODD-SHAPED EYEGLASSES EXPRESS PERSONALITY

"INDIVIDUALIZED" eyeglasses are becoming a fad in England, and makers, departing from the convention that lenses and frames must be round or oval, are producing them in bizarre patterns. A heart-shaped pair, for feminine wearer, is illustrated.

Puzzler: Implementing a Test for Oddness

Is this Correct?

Don't Try This at Home

```
public static boolean isOdd( int number ) {
    return number % 2 == 1;
}
```

Puzzler: Implementing a Test for Oddness

Solution

Java

```
public static boolean isOdd( int number ) {
    return number % 2 != 0;
}
```

For Monday

- Study Sections 4.7, and 4.8.
- Optional: study Section 4.4 and 4.6.
- Carry out Programming Exercise 4.28.

Acknowledgements

- This lecture corresponds to [*Big Java, Early Objects*, 3.1–3.2].
- The partial operator precedence table is based on `http://download.oracle.com/javase/tutorial/java/nutsandbolts/operators.html`.
- The puzzler is based on Bloch, and Gafter 2005, Puzzles 1.

Bibliography

Bloch, Joshua, and Neal Gafter [2005]. *Java Puzzlers Traps, Pitfalls, and Corner Cases.* Addison–Wesley. ISBN: 0-321-33678-x.

Horstfmann, Cay S. *Big Java, Early Objects.* International Student Version. Wiley. ISBN: 978-1-118-31877-5.

About this Document

- This document was created with `pdflatex`.
- The $\LaTeX$ document class is `beamer`.