Boolean Expressions and Conditionals (If Statements)

CSE 1310 – Introduction to Computers and Programming
Vassilis Athitsos
University of Texas at Arlington
The boolean Type

- Expressions of type **boolean** can only have two values: **true**, or **false**.
  - **true** and **false** are reserved keywords in Java.

```java
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;
        boolean v1 = (a < 4.3);
        System.out.printf("v1 = %b\n", v1);

        boolean v2 = (a == b);
        System.out.printf("v2 = %b\n", v2);

        boolean v3 = (a != b);
        System.out.printf("v3 = %b\n", v3);
    }
}
```

Output:

```
v1 = true
v2 = false
v3 = true
```
Comparisons of Numbers

- The following operators compare numerical values (of type `double` or `int`), and generate `boolean` results:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>equals</td>
<td><code>!=</code></td>
<td>not equal to</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
<td><code>&gt;=</code></td>
<td>greater than or equal to</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
<td><code>&lt;=</code></td>
<td>less than or equal to</td>
</tr>
</tbody>
</table>

```java
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;
        System.out.printf("a = %.1f, b = %.1f
", a, b);
        System.out.printf("a == b: %b
", a == b);
        System.out.printf("a != b: %b
", a != b);
        System.out.printf("a > b: %b
", a > b);
        System.out.printf("a >= b: %b
", a >= b);
        System.out.printf("a < b: %b
", a < b);
        System.out.printf("a <= b: %b
", a <= b);
    }
}
```

Output:

```
a = 3.2, b = 4.0
a == b: false
a != b: true
a > b: false
a >= b: false
a < b: true
a <= b: true
```
Using Parentheses

• When you assign a boolean variable, use parentheses to make it easy to read your code.
• Even if your code runs correctly without parentheses, parentheses are still recommended to make sure you don't get confused.
• Example: setting c equal to the value of "a equals b".

```java
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean c = a == b; // Correct, but confusing (not recommended!)
        boolean d = (a == b); // Preferred style (parenthesize)
    }
}
```
Using Parentheses

• When you assign a boolean variable, use parentheses to make it easy to read your code.
• Even if your code runs correctly without parentheses, parentheses are still recommended to make sure you don't get confused.
• Example: setting c equal to the value of "a equals b".

```java
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean c = a == b;  // Correct, but confusing (not recommended!)
        boolean d = (a == b);  // Preferred style (parenthesize)
    }
}
```

What is the value of c in this example?
What is the value of d in this example?
Using Parentheses

- When you assign a boolean variable, use parentheses to make it easy to read your code.
- Even if your code runs correctly without parentheses, parentheses are still recommended to make sure you don't get confused.
- Example: setting c equal to the value of "a equals b".

```java
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean c = a == b;  // Correct, but confusing (not recommended!)
        boolean d = (a == b);  // Preferred style (parenthesize)

        System.out.println(c);  // false
        System.out.println(d);  // false
    }
}
```

What is the value of c in this example?

What is the value of d in this example?

They are both equal to false

3.2 is NOT equal to 4.0.
Comparing Numbers: Examples

• Four ways of doing the same comparison (3.2 < 4.0)
  – First way:

```java
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a < b);
        System.out.printf("v1 = %b\n", v1);
    }
}
```

Output:

`v1 = ???`
Comparing Numbers: Examples

• Four ways of doing the same comparison (3.2 < 4.0)
  – First way:

```java
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a < b);
        System.out.printf("v1 = %b\n", v1);
    }
}
```

Output:

```
v1 = true
```
Comparing Numbers: Examples

• Four ways of doing the same comparison (3.2 < 4.0)
  – Second way:

```java
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        boolean v1 = (a < 4.0);
        System.out.printf("v1 = %b\n", v1);
    }
}
```

Output:

```
v1 = true
```
Comparing Numbers: Examples

• Four ways of doing the same comparison (3.2 < 4.0)
  – Third way:

```java
public class example1 {
    public static void main(String[] args) {
        boolean v1 = (3.2 < 4.0);
        System.out.printf("v1 = %b\n", v1);
    }
}
```

Output:
```
v1 = true
```
Comparing Numbers: Examples

• Four ways of doing the same comparison (3.2 < 4.0)
  – Fourth way:

```java
public class example1 {
    public static void main(String[] args) {
        System.out.printf("v1 = %b\n", 3.2 < 4.0);
    }
}
```

Output:

```
v1 = true
```
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a < 4.3 - 2.6);
        System.out.printf("v1 = %b\n", v1);
    }
}

Output:

v1 = ???
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a < 4.3 - 2.6);
        System.out.printf("v1 = %b\n", v1);
    }
}
Comparing Numbers: Examples

```java
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a > 3.2);
        System.out.printf("v1 = %b\n", v1);
    }
}
```

Output:

v1 = ???
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a > 3.2);
        System.out.printf("v1 = %b\n", v1);
    }
}

Output:

v1 = false
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a >= 3.2);
        System.out.printf("v1 = %b\n", v1);
    }
}

Output:

v1 = ???
Comparing Numbers: Examples

```java
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a >= 3.2);
        System.out.printf("v1 = %b\n", v1);
    }
}
```

Output:

v1 = true
Comparing Numbers: Examples

```java
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a < 3.2);
        System.out.printf("v1 = %b\n", v1);
    }
}
```

Output:

```
v1 = ???
```
Comparing Numbers: Examples

```java
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a < 3.2);
        System.out.printf("v1 = %b\n", v1);
    }
}
```

Output:

`v1 = false`
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a <= 3.2);
        System.out.printf("v1 = %b\n", v1);
    }
}

Output:

v1 = ???
Comparing Numbers: Examples

public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a <= 3.2);
        System.out.printf("v1 = %b\n", v1);
    }
}

Output:

v1 = true
Comparing Numbers: Examples

```java
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a != 3.2);
        System.out.printf("v1 = %b\n", v1);
    }
}
```

Output:

```
v1 = ???
```
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a != 3.2);
        System.out.printf("v1 = %b\n", v1);
    }
}

Output:

v1 = false
public class example1 { 
    public static void main(String[] args) { 
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a == 3.2);
        System.out.printf("v1 = \%b\n", v1);
    } 
} 

Output:

v1 = ???
Comparing Numbers: Examples

```java
class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a == 3.2);
        System.out.printf("v1 = %b\n", v1);
    }
}
```

Output:

```
v1 = true
```
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a = 3.2);
        System.out.printf("v1 = %b\n", v1);
    }
}

Output:
???
Comparing Numbers: Examples

```java
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        boolean v1 = (a = 3.2);
        System.out.printf("v1 = %b\n", v1);
    }
}
```

Output:

Error (does not run), we need == sign instead of = sign.

Very common error!!!
Comparisons of Strings

- The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

- Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
public class example1 {
    public static void main(String[] args) {
        String a = "hello";
        String b = "hello";
        boolean r = a.equals(b);
        System.out.printf("r = %b\n", r);
    }
}
```

Output: ???
Comparisons of Strings

• The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

• Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
public class example1 {
    public static void main(String[] args) {
        String a = "hello";
        String b = "hello"
        boolean r = a.equals(b);
        System.out.printf("r = %b
", r);
    }
}
```

Output:

```
r = true
```
Comparisons of Strings

• The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

• Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
public class example1 {
    public static void main(String[] args) {
        String a = "Robert";
        String b = a.substring(0, 3);
        boolean r = (b == "Rob");
        System.out.printf("r = %b\n", r);
    }
}
```

Output:

???
Comparisons of Strings

- The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

- Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
public class example1 {
    public static void main(String[] args) {
        String a = "Robert";
        String b = a.substring(0, 3);
        boolean r = (b == "Rob");
        System.out.printf("r = %b\n", r);
    }
}
```

Output:

```
r = false
```

Very common bug!!!
How do we fix it?
Comparisons of Strings

• The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

• Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
public class example1 {
    public static void main(String[] args) {
        String a = "Robert";
        String b = a.substring(0, 3);
        boolean r = b.equals("Rob");
        System.out.printf("r = %b\n", r);
    }
}
```

Output:
```
r = true
```

Very common bug!!!
How do we fix it?
Use b.equals("Rob")
Comparisons of Strings

• The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

• Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
class example1 {
    public static void main(String[] args) {
        String a = "hello";
        String b = "Hello";
        boolean r = a.equals(b);
        System.out.printf("r = %b\n", r);
    }
}
```

Output:

???
Comparisons of Strings

- The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

- Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
public class example1 {
    public static void main(String[] args) { 
        String a = "hello";
        String b = "Hello";
        boolean r = a.equals(b);
        System.out.printf("r = %b\n", r);
    }
}
```

Output:
```
r = false
```

String comparisons are case sensitive!!!
Comparisons of Strings

- The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

- Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
public class example1 {
    public static void main(String[] args) {
        String a = "hello";
        String b = "world";
        boolean r = (a.compareTo(b) < 0);
        System.out.printf("r = %b\n", r);
    }
}
```

Output: ???
Comparisons of Strings

• The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

• Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
public class example1 {
    public static void main(String[] args) {
        String a = "hello";
        String b = "world";
        boolean r = (a.compareTo(b) < 0);
        System.out.printf("r = %b\n", r);
    }
}
```

Output:

```
r = true
```

hello comes before world
Comparisons of Strings

• The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

• Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
cpyublic class example1 {
    public static void main(String[] args) {
        String a = "hello";
        String b = "World";
        boolean r = (a.compareTo(b) < 0);
        System.out.printf("r = %b\n", r);
    }
}
```

Output:

???
Comparisons of Strings

• The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

• Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
public class example1 {
    public static void main(String[] args) {
        String a = "hello";
        String b = "World";
        boolean r = (a.compareTo(b) < 0);
        System.out.printf("r = %b\n", r);
    }
}
```

Output: r = false

Upper case letters come before lower case letters
Comparisons of Strings

• The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

• Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
public class example1 {
    public static void main(String[] args) {
        String a = "hello";
        String b = "hello";
        boolean r = (a.compareTo(b) == 0);
        System.out.printf("r = %b\n", r);
    }
}
```

Output:

```java
???
```
Comparisons of Strings

• The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

• Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
public class example1 {
    public static void main(String[] args) {
        String a = "hello";
        String b = "hello";
        boolean r = (a.compareTo(b) == 0);
        System.out.printf("r = %b\n", r);
    }
}
```

Output:  

```
r = true
```

This is the same as doing:  

```java
a.equals(b)
```
Comparisons of Strings

- The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

- Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
public class example1 {
    public static void main(String[] args) {
        String a = "hello";
        String b = "Hello";
        boolean r = (a.compareTo(b) == 0);
        System.out.printf("r = %b\n", r);
    }
}
```

Output:

```java
???
```
Comparisons of Strings

• The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

• Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
class example1 {
    public static void main(String[] args) {
        String a = "hello";
        String b = "Hello";
        boolean r = (a.compareTo(b) == 0);
        System.out.printf("r = %b\n", r);
    }
}
```

Output:

```
r = false
```

Again: comparisons are case sensitive, and upper case letters come before lower case letters.
Comparisons of Strings

- The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

- Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
public class example1 {
    public static void main(String[] args) {
        String a = "hello";
        String b = "Hello";
        boolean r = (a.compareTo(b) < 0);
        System.out.printf("r = %b\n", r);
    }
}
```

Output:

```java
???
```

43
Comparisons of Strings

- The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

- Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
public class example1 {
    public static void main(String[] args) {
        String a = "hello";
        String b = "Hello";
        boolean r = (a.compareTo(b) < 0);
        System.out.printf("r = %b\n", r);
    }
}
```

Output:

```
r = false
```

Upper case letters come before lower case letters
Comparisons of Strings

- The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

- Java's version of alphabetical order: upper case letters *come before* lower case letters.

```java
class example1 {
    public static void main(String[] args) {
        String a = "hello";
        String b = "Hello";
        boolean r = (a.compareTo(b) > 0);
        System.out.printf("r = %b\n", r);
    }
}
```
Comparisons of Strings

- The following operators compare strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.equals(b)</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &lt; 0</td>
<td>a comes before b in Java's version of alphabetical order</td>
</tr>
<tr>
<td>a.compareTo(b) == 0</td>
<td>a equals b</td>
</tr>
<tr>
<td>a.compareTo(b) &gt; 0</td>
<td>a comes after b in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

- Java's version of alphabetical order: upper case letters **come before** lower case letters.

```java
public class example1 {
    public static void main(String[] args) {
        String a = "hello";
        String b = "Hello";
        boolean r = (a.compareTo(b) > 0);
        System.out.printf("r = %b\n", r);
    }
}
```

**Output:**

```
r = true
```

**Upper case letters come before lower case letters**
Logical Operators

• The following logical operators can be used to produce boolean results:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
</tr>
<tr>
<td>a &amp;&amp; b</td>
<td>a AND b</td>
</tr>
<tr>
<td>!a</td>
<td>NOT a</td>
</tr>
</tbody>
</table>
## Truth Tables for or, and, not

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>OR</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>
# Truth Tables for or, and, not

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>OR</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>!A</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

```java
class example1 {
    public static void main(String[] args) {
        int x = 3;
        int y = 5;

        boolean m = (x == 3) && (y < 10);
        System.out.printf("m = %b\n", m);
        boolean n = (x == 3) && (y > 10);
        System.out.printf("n = %b\n", n);
    }
}
```

Output:
```plaintext
???
```
Truth Tables for or, and, not

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>OR</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>!A</td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td></td>
</tr>
</tbody>
</table>

```java
public class example1 {
    public static void main(String[] args) {
        int x = 3;
        int y = 5;

        boolean m = (x == 3) && (y < 10);
        System.out.printf("m = %b\n", m);
        boolean n = (x == 3) && (y > 10);
        System.out.printf("n = %b\n", n);
    }
}
```

Output:

m = true
n = false
### Truth Tables for or, and, not

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>OR</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

```java
public class example1 {
    public static void main(String[] args) {
        int x = 3;
        int y = 5;

        boolean m = (x == 3) || (y < 10);
        System.out.printf("m = %b\n", m);
        boolean n = (x == 3) || (y > 10);
        System.out.printf("n = %b\n", n);
    }
}
```

Output:

???
Truth Tables for or, and, not

| a    | b    | a || b | a && b |
|------|------|-------|--------|
| True | True | True  | True   |
| True | False| True  | False  |
| False| True | True  | False  |
| False| False| False | False  |

<table>
<thead>
<tr>
<th>A</th>
<th>!A</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

```java
public class example1 {
    public static void main(String[] args) {
        int x = 3;
        int y = 5;

        boolean m = (x == 3) || (y < 10);
        System.out.printf("m = %b\n", m);
        boolean n = (x == 3) || (y > 10);
        System.out.printf("n = %b\n", n);
    }
}
```

Output:

m = true
n = true
Truth Tables for or, and, not

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>OR</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>!A</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

```java
public class example1 {
    public static void main(String[] args) {
        int x = 3;
        int y = 5;

        boolean m = !(x == 3);
        System.out.printf("m = %b\n", m);
        boolean n = !(x == 4);
        System.out.printf("n = %b\n", n);
    }
}
```

Output:

???
Truth Tables for or, and, not

| a   | b   | a || b | a && b |
|-----|-----|-------|--------|
| True| True| True  | True   |
| True| False| True  | False  |
| False| True| True  | False  |
| False| False| False | False  |

<table>
<thead>
<tr>
<th>A</th>
<th>!A</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

```java
public class example1 {
    public static void main(String[] args) {
        int x = 3;
        int y = 5;

        boolean m = !(x == 3);
        System.out.printf("m = %b\n", m);
        boolean n = !(x == 4);
        System.out.printf("n = %b\n", n);
    }
}
```

Output:
m = false
n = true
## Truth Tables for or, and, not

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>OR</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

```java
class example1 {
    public static void main(String[] args) {
        int x = 3;
        int y = 5;

        boolean m = ((x == y) && (x + y == 8));
        System.out.printf("m = %b\n", m);
        boolean n = ((x == y) || (x + y == 8));
        System.out.printf("n = %b\n", n);
    }
}
```

Output:

???
# Truth Tables for or, and, not

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>OR</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>!A</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

```java
public class example1 {
    public static void main(String[] args) {
        int x = 3;
        int y = 5;

        boolean m = ((x == y) && (x + y == 8));
        System.out.printf("m = %b\n", m);
        boolean n = ((x == y) || (x + y == 8));
        System.out.printf("n = %b\n", n);
    }
}
```

Output:

m = false
n = true
Complicated Use of Operators

• What does this code print?

```java
public class example1 {
    public static void main(String[] args) {
        boolean m = (3 == 5) && (2 < 3) || (3 >= 0);
        System.out.printf("m = %b\n", m);
    }
}
```
Complicated Use of Operators

• What does this code print?

```java
public class example1 {
    public static void main(String[] args) {
        boolean m = (3 == 5) && (2 < 3) || (3 >= 0);
        System.out.printf("m = %b\n", m);
    }
}
```

• I don't know, and I don't want to know.
  – No need to memorize complex rules to predict this type of behavior.
  – Use parentheses to make the meaning clear.

\[
((3 == 5) && (2 < 3)) || (3 >= 0) \Rightarrow \text{true}
\]
\[
(3 == 5) && ((2 < 3) || (3 >= 0)) \Rightarrow \text{false}
\]
Conditionals - `if` statements

- An `if` statement looks like this:

```java
if (boolean)
{
    if-line 1;
    if-line 2;
    ...
    if-line m;
}
else
{
    else-line 1;
    else-line 2;
    ...
    else-line n;
}
```

- Meaning of an `if` statement:
  - if `boolean` is true, execute:
    ```java
    if-line 1;
    if-line 2;
    ...
    if-line n;
    ```
    and skip the else-lines.
  - Otherwise, skip the if-lines, and execute:
    ```java
    else-line 1;
    else-line 2;
    ...
    else-line n;
    ```
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("How old are you? ");
        int age = in.nextInt();

        if (age < 21)
        {
            System.out.printf("How about some milk?\n");
        }
        else
        {
            System.out.printf("How about some beer?\n");
        }
    }
}
An example of an if statement

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("How old are you? ");
        int age = in.nextInt();

        if (age < 21) {
            System.out.printf("How about some milk?\n");
        } else {
            System.out.printf("How about some beer?\n");
        }
    }
}
```

Example Output 1:
How old are you? 18
How about some milk?

Example Output 2:
How old are you? 21
How about some beer?

Example Output 3:
How old are you? 24
How about some beer?
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("How old are you? ");
        int age = in.nextInt();

        if ((age >= 40) && (age <= 60)) {
            System.out.printf("You are middle aged.\n");
            System.out.printf("You are not young.\n");
            System.out.printf("You are not old.\n");
        }
    }
}

Example Output 1:
How old are you? 18

Example Output 2:
How old are you? 45
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("How old are you? ");
        int age = in.nextInt();

        if ((age >= 40) && (age <= 60))
        {
            System.out.printf("You are middle aged.\n");
            System.out.printf("You are not young.\n");
            System.out.printf("You are not old.\n");
        }
    }
}

Example Output 1:
Example Output 2:

Note: the else part of an if statement IS OPTIONAL. No else in this example.
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("How old are you? ");
        int age = in.nextInt();

        if ((age >= 40) && (age <= 60)) {
            System.out.printf("You are middle aged.\n");
            System.out.printf("You are not young.\n");
            System.out.printf("You are not old.\n");
        }
        else {
            System.out.printf("You are not middle aged.\n");
            System.out.printf("You are either young or old.\n");
        }
    }
}

Example Output 1:
How old are you? 18

Example Output 2:
How old are you? 45
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("How old are you? ");
        int age = in.nextInt();

        if ((age >= 40) && (age <= 60))
        {
            System.out.printf("You are middle aged.\n");
            System.out.printf("You are not young.\n");
            System.out.printf("You are not old.\n");
        }
        else
        {
            System.out.printf("You are not middle aged.\n");
            System.out.printf("You are either young or old.\n");
        }
    }
}

Example Output 1:
How old are you? 18
You are not middle aged.
You are either young or old.

Example Output 2:
How old are you? 45
You are middle aged.
You are not young.
You are not old.
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.printf("How old are you? ");
    int age = in.nextInt();

    if (age < 18)
    {
        System.out.printf("You are not an adult.\n");
    }
    else if (age < 40)
    {
        System.out.printf("You are a young adult.\n");
    }
    else if (age < 60)
    {
        System.out.printf("You are middle aged.\n");
    }
    else
    {
        System.out.printf("You are a senior citizen.\n");
    }
}

Example Output 1:
How old are you? 18

Example Output 2:
How old are you? 45

Example Output 3:
How old are you? 65
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.printf("How old are you? ");
    int age = in.nextInt();

    if (age < 18) {
        System.out.printf("You are not an adult.\n");
    }
    else if (age < 40) {
        System.out.printf("You are a young adult.\n");
    }
    else if (age < 60) {
        System.out.printf("You are middle aged.\n");
    }
    else {
        System.out.printf("You are a senior citizen.\n");
    }
}
Successive if conditions vs. if-else if

import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("How old are you? ");
        int age = in.nextInt();

        if (age < 40)
            System.out.printf("You are young.\n");
        else if (age < 60)
            System.out.printf("You are middle aged.\n");
        else
            System.out.printf("You are old.\n");
    }
}

How will the program behavior change if we remove the else that is highlighted in red?
Successive if{}s, vs. if-else if

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("How old are you? ");
        int age = in.nextInt();

        if (age < 40)
        {
            System.out.printf("You are young.\n");
        }
        else if (age < 60)
        {
            System.out.printf("You are middle aged.\n");
        }
        else
        {
            System.out.printf("You are old.\n");
        }
    }
}
```

How will the program behavior change if we remove the `else` that is highlighted in red?

Consider an age of 30.

Output with else if:

How old are you? 30

```
Successive ifs, vs. if-else if

import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("How old are you? ");
        int age = in.nextInt();

        if (age < 40)
            System.out.printf("You are young.\n");
        else if (age < 60)
            System.out.printf("You are middle aged.\n");
        else
            System.out.printf("You are old.\n");
    }
}

How will the program behavior change if we remove the else that is highlighted in red?

Consider an age of 30.

Output with else if:
How old are you? 30
You are young.
Successive ifs, vs. if-else if

import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("How old are you? ");
        int age = in.nextInt();

        if (age < 40) {
            System.out.printf("You are young.\n");
        } 
        else {
            System.out.printf("You are old.\n");
        }
    }
}

How will the program behavior change if we remove the else that is highlighted in red?

Consider an age of 30.

Output with else if

How old are you? 30
You are young.

Output with two successive if statements:

How old are you? 30
???
Successive ifs, vs. if-else if

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("How old are you? ");
        int age = in.nextInt();

        if (age < 40)
            System.out.printf("You are young.\n");
        if (age < 60)
            System.out.printf("You are middle aged.\n");
        else
            System.out.printf("You are old.\n");
    }
}
```

How will the program behavior change if we remove the `else` that is highlighted in red?

Consider an age of 30.

Output with else if

How old are you? 30
You are young.

Output with two successive if statements:

How old are you? 30
You are young.
You are middle aged.
Successive ifs, vs. if-else if

This is an example where using successive if statements, instead of an else if, leads to incorrect behavior.

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("How old are you? ");
        int age = in.nextInt();

        if (age < 40)
            System.out.printf("You are young. \n");
        if (age < 60)
            System.out.printf("You are middle aged. \n");
        else
            System.out.printf("You are old. \n");
    }
}
```

Output with else if

How old are you? 30
You are young.

Output with two successive if statements:

How old are you? 30
You are young.
You are middle aged.
The Importance of Indentation

import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("How old are you? ");
        int age = in.nextInt();

        if (age < 40)
            System.out.printf("You are young.\n");
        else if (age < 60)
            System.out.printf("You are middle aged.\n");
        else
            System.out.printf("You are old.\n");
    }
}

This program is indented appropriately.

Every time we open a brace, we increase indentation.

Every time we close a brace, we decrease indentation.

Netbeans does this for you automatically, but may get confused every now and then, and then you need to fix the indentations manually.
The Importance of Indentation

This program is indented inappropriately.

Indentation does not change program behavior, but makes program harder to read, and mistakes harder to find.
Indentation on NetBeans

• NetBeans can automatically indent your code.
• Select Source->Format.
• This will work only if your code is valid Java code. If your code cannot run because of syntax errors, NetBeans may get confused about the correct indentation.
Placement of Braces

First way:
{ placed in a line of its own.
That is what I usually do.

Second way:
{ placed at the end of the if line.
This is also fine, if you want to do it that way.

```java
public class example1{
    public static void main(String[] args){
        int a = 7;
        if (a > 5){
            System.out.printf("a > 5.\n");
        }
    }
}
```
Braces on NetBeans

• Source->Format will automatically set the position of braces for you.

• Again, this will work only if your code is valid Java code, that can run.

• You can set some preferences for automatic formatting using Tools->Options.
  – Select Editor at the top.
  – Select Formatting.
  – In "Category" select braces. You can specify if you want braces on their own in a new line, or at the end of the current line.
These two examples do not use braces under if.

This is legal, but it can lead to bugs when you add more lines.

STRONGLY NOT RECOMMENDED
Not Using Braces - Example of Bug

```java
public class example1 {
    public static void main(String[] args) {
        int a = 3;
        if (a > 5)
            System.out.printf("a = %d.\n", a);
        System.out.printf("a > 5.\n");
    }
}
```

Not using braces under if: it is legal, but it can lead to bugs when you add more lines.

STRONGLY NOT RECOMMENDED

What will this example print?

How many if-lines are there?
Not Using Braces - Example of Bug

public class example1 {
    public static void main(String[] args) {
        int a = 3;
        if (a > 5)
            System.out.printf("a = %d.\n", a);
        System.out.printf("a > 5.\n");
    }
}

Not using braces under if: it is legal, but it can lead to bugs when you add more lines.

STRONGLY NOT RECOMMENDED

What will this example print?
a > 5

How many if-lines are there?
Just one (if you do not use braces under if, there can only be one if-line).

The top example does the same thing as the bottom example.
Not Using Braces - Example of Bug

```java
public class example1 {
    public static void main(String[] args) {
        int a = 3;
        if (a > 5)
            System.out.printf("a = %d.\n", a);
        System.out.printf("a > 5.\n");
    }
}
```

Not using braces under if: it is legal, but it can lead to bugs when you add more lines.

STRONGLY NOT RECOMMENDED

```java
public class example1 {
    public static void main(String[] args) {
        int a = 3;
        if (a > 5)
        {
            System.out.printf("a = %d.\n", a);
            System.out.printf("a > 5.\n");
        }
    }
}
```

If you wanted two if-lines, you should have used braces, as shown on the bottom example on this slide.
Another Common Bug

What will this print?

```java
public class example1 {
    public static void main(String[] args) {
        int a = 3;
        if (a > 5);
        {
            System.out.printf("a = %d.\n", a);
            System.out.printf("a > 5.\n");
        }
    }
}
```
Another Common Bug

What will this print?

```java
public class example1 {
    public static void main(String[] args) {
        int a = 3;
        if (a > 5);
        {
            System.out.printf("a = %d.\n", a);
            System.out.printf("a > 5.\n");
        }
    }
}
```

a = 3.
a > 5.

What is the problem?
Another Common Bug

public class example1 {
    public static void main(String[] args) {
        int a = 3;
        if (a > 5);
        {
            System.out.printf("a = %d.\n", a);
            System.out.printf("a > 5.\n");
        }
    }
}

public class example1 {
    public static void main(String[] args) {
        int a = 3;
        if (a > 5)
        {
            System.out.printf("a = %d.\n", a);
            System.out.printf("a > 5.\n");
        }
    }
}

What will this print?

a = 3.
a > 5.

What is the problem?

Semicolon on the if line.

The bottom example shows the fixed version.
Conditionals with Strings: An Example

- Write a program that:
  - Asks the user to enter the name of the month.
  - Prints "M has X days" where M is the month and X is the correct number of days.
  - If the user did not enter a valid month name, the program prints "M is not a valid month"
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Enter the name of a month: ");
        String m = in.next();

        if (m.equals("January") ||
            m.equals("March") ||
            m.equals("May") ||
            m.equals("July") ||
            m.equals("August") ||
            m.equals("October") ||
            m.equals("December"))
        {
            System.out.printf("%s has 31 days.\n", m);
        }
        else if (m.equals("April") ||
                 m.equals("June") ||
                 m.equals("September") ||
                 m.equals("November"))
        {
            System.out.printf("%s has 30 days.\n", m);
        }
        else if (m.equals("February"))
        {
            System.out.printf("%s has 28 or 29 days.\n", m);
        }
        else
        {
            System.out.printf("%s is not a valid month.\n", m);
        }
    }
}
The String `indexOf` Method

- Suppose that variables `str1` and `str2` are strings.
- Suppose you want to see if `str1` contains `str2`.
- You can call `str1.indexOf(str2)`.
- If `str1` contains `str2`, `indexOf` returns the FIRST position where `str2` appears in `str1`.
- If `str1` does NOT contain `str2`, `indexOf` returns -1.
indexOf Example

- Write a program that:
  - Asks the user to enter a single letter.
  - If the user enters a string with more than one letter, (or less than one letter) exit the program.
  - If the letter is a vowel, print that it is a vowel
  - Else, print that the letter is not a vowel.
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Enter a single letter: ");

        String c = in.next();
        if (c.length() != 1) {
            System.out.printf("invalid input.\n");
            System.exit(0);
        }

        String vowels = "aeiouAEIOU";
        int result = vowels.indexOf(c);
        if (result != -1) {
            System.out.printf("%s is a vowel.\n", c);
        } else {
            System.out.printf("%s is not a vowel.\n", c);
        }
    }
}
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Enter a single letter: ");

        String c = in.next();
        if (c.length() != 1) {
            System.out.printf("invalid input.\n");
            System.exit(0);
        }

        String vowels = "aeiouAEIOU";
        int result = vowels.indexOf(c);
        if (result != -1) {
            System.out.printf("%s is a vowel.\n", c);
        } else {
            System.out.printf("%s is not a vowel.\n", c);
        }
    }
}
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Enter a single letter: ");

        String c = in.next();
        if (c.length() != 1) {
            System.out.printf("invalid input.\n");
            System.exit(0);
        }

        if (c.equals("a") || c.equals("e") || c.equals("i") || c.equals("o") || c.equals("u") || c.equals("A") || c.equals("E") || c.equals("I") || c.equals("O") || c.equals("U")) {
            System.out.printf("%s is a vowel.\n", c);
        } else {
            System.out.printf("%s is not a vowel.\n", c);
        }
    }
}
Variable Scope

- Variables do not live forever.
- Failing to take that into account leads to problems.
- Let's look at an example.
- Let's write a program that:
  - Asks the user to enter an integer.
  - If the integer is $\geq 0$, it creates a variable `result` and sets it equal to the square of the integer.
  - If the integer is $< 0$, it creates a variable `result` and sets it equal to $10 \times$ the integer.
  - It prints out the value of variable `result`. 
Variable Scope

- This version will not run.
- Why?

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
        {
            int result = number * number;
        }
        else
        {
            result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Variable Scope

- This version will not run.
- Why? Java will complain that "it cannot find symbol" **result**, on the line shown in red.
- How do we fix this?
Variable Scope

• This version will also not run.

• Why?

import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
        {
            int result = number * number;
        }
        else
        {
            int result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
Variable Scope

- This version will also not run.
- Why? Java will complain that "it cannot find symbol" result, on the line shown in red.

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
        {
            int result = number * number;
        }
        else
        {
            int result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Lifetime of a Variable

• What is the reason for these errors?
• Each variable has a lifetime.
  – It is born in a line of code.
  – It dies at some point.
• To understand the lifetime of a variable, we must understand the concept of a block of code.

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
        {
            int result = number * number;
        }
        else
        {
            int result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Blocks of Code and Braces

- A **block** of code is a chunk of code that:
  - Starts with an opening brace `{.`.
  - Ends with the corresponding closing brace `}`.

- **Example:**
  - What block of code does the highlighted line belong to?
Blocks of Code and Braces

• A **block** of code is a chunk of code that:
  – Starts with an opening brace `{.
  – Ends with the corresponding closing brace `}.

• Example:
  – What block of code does the highlighted line belong to?
  – The answer is shown in red.

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
        {
            int result = number * number;
        }
        else
        {
            int result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Blocks of Code and Braces

• A **block** of code is a chunk of code that:
  – Starts with an opening brace `{`.
  – Ends with the corresponding closing brace `}`.

• Example:
  – What block of code does the highlighted line belong to?

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
        {
            int result = number * number;
        }
        else
        {
            int result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Blocks of Code and Braces

• A **block** of code is a chunk of code that:
  – Starts with an opening brace `{.
  – Ends with the corresponding closing brace `}.

• Example:
  – What block of code does the highlighted line belong to?
  – The answer is shown in red.

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
            int result = number * number;
        else
            int result = number * 10;
        System.out.printf("result = %d\n", result);
    }
}
```
Birth of a Variable

• Each variable has a lifetime.
  – It is born in a line of code.
  – It dies at some point.

• Where is a variable born?
  – At the line where it is declared.

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
        {
            int result = number * number;
        }
        else
        {
            int result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Death of a Variable

• Where is a variable born?
  – At the line where it is declared.

• Where does a variable die?
  – Find the **innermost block** containing the variable declaration.
  – The variable dies at the end of that block.

```java
import java.util.Scanner; // incorrect code

class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0) {
            int result = number * number;
        } else {
            int result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Scope of a Variable

• Where is a variable born?
  – At the line where it is declared.

• Where does a variable die?
  – Find the **innermost block** containing the variable declaration.
  – The variable dies at the end of that block.

• The lines of code where a variable is alive are called the **scope** of that variable.

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
        {
            int result = number * number;
        }
        else
        {
            int result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Variable Scope

- Where is variable **number** born?

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
        {
            int result = number * number;
        } else
        {
            int result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Variable Scope

• Where is variable \texttt{number} born?
  – At the line where it is declared.

```java
import java.util.Scanner; // incorrect code

global class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
        {
            int result = number * number;
        }
        else
        {
            int result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Variable Scope

• Where is variable **number** born?
  – At the line where it is declared.

• Where does variable **number** die?

```java
import java.util.Scanner; // incorrect code

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0) {
            int result = number * number;
        } else {
            int result = number * 10;
        }
        System.out.printf("result = %d
", result);
    }
}
```
Variable Scope

• Where is variable **number** born?
  – At the line where it is declared.

• Where does variable **number** die?
  – The innermost block where containing the declaration of **number** is shown in red.

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        public static void main(String[] args)
        {
            Scanner in = new Scanner(System.in);
            System.out.printf("Please enter an integer: ");
            int number = in.nextInt();

            if (number >= 0)
            {
                int result = number * number;
            }
            else
            {
                int result = number * 10;
            }
            System.out.printf("result = %d\n", result);
        }
    }
}
```
Variable Scope

• Where is variable number born?
  – At the line where it is declared.

• Where does variable number die?
  – The innermost block where containing the declaration of number is shown in red.
  – So, number dies when we get outside that block.
  – In short, number dies at the end of the program.

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
        {
            int result = number * number;
        }
        else
        {
            int result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Variable Scope

• So, what is the scope of variable `number`?

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
        {
            int result = number * number;
        }
        else
        {
            int result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Variable Scope

• So, what is the scope of variable `number`?
• It is the lines between its declaration and its death.
  – Shown in red.
• It cannot be used before or after those lines.
• This is fine, the scope of `number` is what it should be.

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
        {
            int result = number * number;
        }
        else
        {
            int result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Variable Scope

- What is the scope of variable `result`?

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
        {
            int result = number * number;
        }
        else
        {
            int result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Variable Scope

- What is the scope of variable `result`?
- In this code, **there are two independent variables called result.**
  - The first one has its scope shown in red.
  - The second one has its scope shown in green.
- Obviously, none of them is alive at the `printf` line.
  - That is why Java complains.

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
        {
            int result = number * number;
        }
        else
        {
            int result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Variable Scope

• What is the scope of variable `result` in this example?

```java
import java.util.Scanner; // incorrect code

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0) {
            int result = number * number;
        } else {
            result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}```
Variable Scope

- What is the scope of variable `result` in this example?
- It is shown in red on this slide.
- Obviously, in this example, `result` is not alive either at the else part or at the `printf` line at the end.
  - That is why Java complains.

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0)
        {
            int result = number * number;
        }
        else
        {
            result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Variable Scope

• How do we make this code correct?

```java
import java.util.Scanner; // incorrect code

public class example1 {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        if (number >= 0) {
            int result = number * number;
        } else {
            result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Variable Scope

- How do we make this code correct?
- We need to make sure that we have a single variable, called `result`, that is alive:
  - at the if part.
  - at the else part.
  - at the `printf` statement at the end.

```java
import java.util.Scanner; // incorrect code

public class example1 {
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.printf("Please enter an integer: ");
    int number = in.nextInt();

    if (number >= 0) {
      int result = number * number;
    } else {
      result = number * 10;
    }
    System.out.printf("result = %d\n", result);
  }
}
```
Variable Scope

• The solution is to declare **result** before the if part.
• What is the scope of **result** now?

```java
import java.util.Scanner; // correct code

class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        int result;
        if (number >= 0) {
            result = number * number;
        } else {
            result = number * 10;
        }
        System.out.printf("result = %d
", result);
    }
}
```
Variable Scope

- The solution is to declare `result` before the if part.
- What is the scope of `result` now?
- It is shown in red on this slide.

```java
import java.util.Scanner; // correct code

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        int result;
        if (number >= 0) {
            result = number * number;
        } else {
            result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Variable Scope

- We do not have to assign a value to `result` when we declare it.
- Java is sure that, when it is time to print the value at the end, `result` has received a value.
- How can it be sure?

```java
import java.util.Scanner; // correct code

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        int result;
        if (number >= 0) {
            result = number * number;
        } else {
            result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Variable Scope

• We do not have to assign a value to `result` when we declare it.
• Java is sure that, when it is time to print the value at the end, `result` has received a value.
• How can it be sure?
  – Because result is assigned a value both by the if part and by the else part.

```java
import java.util.Scanner; // correct code

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        int result;
        if (number >= 0) {
            result = number * number;
        } else {
            result = number * 10;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Variable Scope

• What is going to happen here?
  – We removed the else part.

```java
import java.util.Scanner; // incorrect code

public class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        int result;
        if (number >= 0)
        {
            result = number * number;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
Variable Scope

• What is going to happen here?
  – We removed the else part.
• Java will refuse to run this code.
• Reason: if number < 0, then result never receives a value.
• Before running a program, Java must prove to itself that all variables receive values before they are used.

```java
import java.util.Scanner; // incorrect code

class example1
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int number = in.nextInt();

        int result;
        if (number >= 0)
        {
            result = number * number;
        }
        System.out.printf("result = %d\n", result);
    }
}
```
More Examples of Conditionals

• Determining if integer K is a divisor of integer N.
• Determining if a day is a weekend.
• Determining if a day is a weekday or a weekend.
• Determining if a month is a summer month.
• Determining the season of a month.
• Determining if a year is a leap year.
• Calculating tax.
• Translating English to Spanish.
  – More accurately: translating a few English words to Spanish.
• Determining the weekday for a date in February 2015.