Decisions (If Statements) And Boolean Expressions

CSE 1310 – Introduction to Computers and Programming
Alexandra Stefan
University of Texas at Arlington

Last updated: 9/22/2019
Simple Example: winner song

Scanner in = new Scanner(System.in);
int votes1, votes2;
System.out.printf("Enter total votes for song 1:");
votes1 = in.nextInt(); // assume user enters 5
System.out.printf("Enter total votes for song 2:");
votes2 = in.nextInt(); // assume user enters 17

if (votes1 <= votes2) {
    System.out.print("Song 1 is the winner! \n");
} else {
    System.out.print("Song 2 is the winner! \n");
    System.out.print("It may have been a tie! \n");
}
System.out.printf("Bye \n");
Complex Example: winner song

```java
Scanner in = new Scanner(System.in);
int votes1, votes2;
System.out.printf("Enter total votes for song 1:");
votes1 = in.nextInt(); // assume user enters 5
System.out.printf("Enter total votes for song 2:");
votes2 = in.nextInt(); // assume user enters 17

if (votes1 <= votes2) {
    System.out.print("Song 1 is the winner!\n");
} else {
    System.out.print("Song 2 is the winner!\n");
    System.out.print("But it may have been a tie!\n");
    System.out.print("Do you want to know that? (yes/no) ");
    userAnsw = in.next();
    if ( userAnsw.equals("yes") ) {
        if ( votes1==votes2 ) {
            System.out.print("yes, it was a tie!\n");
        } else {
            System.out.print("It was NOT a tie!\n");
        }
    } else {
        System.out.print("It was NOT a tie!\n");
    }
}
System.out.printf("Bye\n");
```

Note:

- the 2 ways to place curly braces
- use == for comparing numbers
- use equals() for string comparison (NOT ==)
- Any number and any type of instruction can be in any of the branches of an if-statement
- Nested if statements

Draw diagram
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 <= 18) {
            System.out.printf("You are a teenager.\n");
            System.out.printf("You are young.\n");
        }
        System.out.printf("Bye\n");
    }
}

Example Output 1:
How old are you? 29
Bye

Example Output 2:
How old are you? 15
You are a teenager.
You are young.
Bye

Note: the else part of an if statement IS OPTIONAL. No else in this example.
Common Bug: semicolon

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

It will print:

a = 3.
a > 5.

What is the problem?

Semicolon on the if line.

fixed.

public class example1 {
    public static void main(String[] args) {
        int a = 3;
        if (a > 5)
        {
            System.out.printf("a = %d.
", a);
            System.out.printf("a > 5.
");
        }
    }
}
Combining if statements: 4 ways

• Sequential:
  – one after the other

• Nested
  – One inside another

• Multiple choice
  – More than 2 options.
  – E.g. given age, print: baby, toddler, kid, teenager,…

• A combination of the above
Problem description:

Write a program that asks for their age and then prints the category they belong to based on age:

- age < 18: not an adult
- 18 ≤ age < 40: young adult
- 40 ≤ age < 60: middle-aged
- 60 ≤ age: senior citizen

Sample Output 1:
How old are you? 18
You are a young adult.

Sample Output 2:
How old are you? 45
You are middle aged.

Sample Output 3:
How old are you? 65
You are a senior citizen.
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.print("You are not an adult. \\
    ");
    else if (age < 40)
        System.out.print("You are a young adult. \\
    ");
    else if (age < 60)
        System.out.print("You are middle aged. \\
    ");
    else  // the else is still optional
        System.out.print("You are a senior citizen. \\
    ");
}
Multiple Cases / Multiple Choice

```java
if (age<18) {
    S.o.print("Not an adult.");
}
else if (age < 40) {
    S.o.print("Young adult.");
}
else if (age < 60) {
    S.o.print("Middle aged.");
}
else { //with or without this else
    S.o.print("Senior citizen.");
}
```

Important:
- If age<18 is **True**, age<40 and age<60 will NOT even be evaluated!
- The **else** “binds” them together. DO not skip it!

How many different paths of execution are possible here?
A *path of execution* is the sequence of instructions that a program goes through. It results in different program behavior.
(Multiple cases bug:)
Successive ifs, vs. if-else if

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

```
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");
    }
}
```

Desired output
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.
Multiple Case (if-else-if) becomes Sequential if statements if there else is missing

S.o.p("How old are you? ");
age = in.nextInt();
if (age<18) {
    S.o.print("Not an adult.");
}
if (age < 40) { // no else
    S.o.print("Young adult.");
}
if (age < 60) { // no else
    S.o.print("Middle aged.");
}
else { //with or without this else
    S.o.print("Senior citizen.");
}

Follow the program execution using the code and the diagram to see how (why) we got this sample output:

How old are you? 30
You are young.
You are middle aged.
Auto-indent your code

• Most IDEs will have an option for code “auto indent”
• jGRASP does not, but you can trick it to do it, by generating a CSD (Control Structure Diagram). Do: View -> Generate CSD
after this you can keep the CSD or remove it (View -> Remove CSD)
Not Using Braces – Allows bugs

These two examples do not use braces under if.

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

***NOT*** RECOMMENDED

Consider the code below. What do you think it will print?

// example 2
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 - Example of Bug

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

```java
int a = 3;
if (a > 5)
    System.out.printf("a = %d.\n", a);
System.out.printf("a > 5.\n");
```

This code will 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 code above is equivalent (does the same thing) as the one below:

```java
int a = 3;
if (a > 5)
{
    System.out.printf("a = %d.\n", a);
}
System.out.printf("a > 5.\n");
```

CORRECT VERSION: use {}
The Importance of Indentation

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.
Summary

if-else

if (condition) {
...
}
else {
...
}

if with no else

if (condition) {
...
}

Multiple choices (e.g. for menus):
(There is also a dedicated instruction for this: `switch`).

if (condition 1) {
...
}
else if (condition 2) {
...
}
else if (condition 3) {
...
}
else if (condition n) {
...
}
else {  //with or without this else
}
if (boolean_expr)
{
    if-line 1;
    if-line 2;
    ...
    if-line m;
}
else
{
    else-line 1;
    else-line 2;
    ...
    else-line n;
}

It is a “compound” statement: a statement that includes multiple statements.
Sequential if-statements

```java
if (condition 1) {
    ... // block A
} else {
    ... // block B
}

if (condition 2) {
    ... // block C
} else {
    ... // block D
}

if (condition 3) {
    ... // block E
} else {
    ... // block F
}
```
Boolean Variables and Expressions

Comparing Strings
The boolean Type

• Answers to “questions” are data of type boolean.
• Data 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\n", a, b);
        System.out.printf("a == b: %b\n", a == b);
        System.out.printf("a != b: %b\n", a != b);
        System.out.printf("a > b: %b\n", a > b);
        System.out.printf("a >= b: %b\n", a >= b);
        System.out.printf("a < b: %b\n", a < b);
        System.out.printf("a <= b: %b\n", 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)
    }
}
```

What is the value of c 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)
• And combining operators: v5 = (a < 4.3 - 2.6);

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

        boolean v1 = (a < b);  //first way
        System.out.printf("v1 = %b\n", v1);
        boolean v2 = (a < 4.0);  //second way
        boolean v3 = (3.2 < 4.0);  // 3-rd way
        // 4-th way. Note that v4 does not exist.
        System.out.printf("v4 = %b\n", 3.2 < 4.0);
        boolean v5 = (a < 4.3 - 2.6);
        System.out.printf("v5 = %b\n", v5);
    }
}
```

Output:

```
v1 = true
v4 = true
v5 = false
```
public class example1 {
    public static void main(String[] args) {
        double a = 3.2;
        double b = 4.0;

        //boolean v1 = (a = 3.2);
        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!!!
public class example1 {
    public static void main(String[] args) {
        double x = 2;
        double y = Math.sqrt(2);
        if ((y*y) == 2) {
            System.out.println("equal");
        } else {
            System.out.println("not equal");
        }
    }
}
Truth Tables for:  ||, &&, !

The following logical operators can be used to ask more complex questions. They produce boolean results:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td>b</td>
<td>a OR b</td>
</tr>
<tr>
<td>!a</td>
<td>NOT a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **a || b (a OR b)**: One is sufficient: If the answer to either a or to b is yes/true. Also ok if they are both true.
- **a && b (a AND b)**: I want both a and b: I want the answer to both questions to be yes/true.
- **!a (NOT a)**: Take the answer for a and flip it! (Like kids do.)

### Truth Tables

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>OR</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td></td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td></td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td></td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td></td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td></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></td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td></td>
<td>false</td>
</tr>
</tbody>
</table>
Truth Tables for: | |, &&, !

The following logical operators can be used to ask more complex questions. They produce boolean results:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</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>

Note: a and b will typically be boolean expressions such as x<0 or y==x

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</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>
### Boolean Expressions Worksheet

<table>
<thead>
<tr>
<th></th>
<th></th>
<th><strong>OR</strong></th>
<th><strong>AND</strong></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><strong>NOT</strong></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>

// What will the following instructions print?

```java
int x = 3;
int y = 5;
System.out.println("B1: " + ((x == 3) && (y < 10)) );
System.out.println("B2: " + ((x == 3) && (y > 10)) );
System.out.println("B3: " + ((x == 3) || (y < 10)) );
System.out.println("B4: " + ((x == 3) || (y > 10)) );
System.out.println("B5: " + !(x == 3) );
System.out.println("B6: " + !(x == 4) );
System.out.println("B7: " + ((x == y) && (x + y == 8)) );
System.out.println("B8: " + ((x == y) || (x + y == 8)) );
```
Asking complex questions

• Write the boolean expression that makes the program print work day or week day correctly
• (You can also think of other ways to structure the code.)

S.o.p("Enter a number (1-7) for the day of the week (Monday is 1 and Sunday is 7): ");
int day = in.nextInt();
if (_______________________) {
    S.o.print("Work day");
}
else {
    S.o.print("Weekend");
}
Asking complex questions - **ANSWER**

- Write the boolean expression that makes the program print work day if day is a number between 1 and 5 inclusive and weekend otherwise. (We are assuming the user enters only values in the range 1-7)

- (You can also think of other ways to structure the code.)

```java
S.o.p("Enter a number (1-7) for the day of the week (Monday is 1 and Sunday is 7): " );
int day = in.nextInt();
if (_______________________) {
    S.o.print("Work day");
} else {
    S.o.print("Weekend");
}
```

<table>
<thead>
<tr>
<th>Correct</th>
<th>Wrong (think about the types)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(1&lt;=day) &amp;&amp; (day &lt;= 5)</code></td>
<td><code>1&lt;=day&lt;=5 (It does not work like math!)</code></td>
</tr>
<tr>
<td>Also ok: <code>(0 &lt; day) &amp;&amp; (day &lt; 6)</code></td>
<td></td>
</tr>
<tr>
<td>`(day == 1)</td>
<td></td>
</tr>
<tr>
<td>(since they are all connected with `</td>
<td></td>
</tr>
</tbody>
</table>
Writing complex expressions

Assume variable day holds an integer 1-7 and variable goodWeather holds the string “yes” or “no”.

Write a boolean expression that involves day and goodWeather that evaluates to true if it is a week-end and the weather is good:

\[( \text{day}==6 \) || \text{day}==7 \) \) && goodWeather.equals("yes")

Is it weekend? Is the weather good?

Notice the ( ) that tells Java to first do || and then the &&.

These parenthesis are crucial. Remove them. Evaluate the expression for day=6 and goodWeather = "no" assuming the following 2 placements of parenthesis.

Remember! Java will place its own parenthesis if you do not do it, and they may not be in the order you want (it is based on the “operator precedent”)

Run this in Java without parenthesis to see what it does:

```java
int day = 6;
String goodWeather = "no";
S.o.print( (day==6) || (day==7) && goodWeather.equals("yes") );
```
Here is another example:

• What does this code print?

```java
System.out.println( (3 == 5) && (2 < 3) || (3 >= 0) );
```

− Use parentheses to make the meaning clear. If paired different, they produce different results

```
((3 == 5) && (2 < 3)) || (3 >= 0)  \rightarrow true
(3 == 5) && ((2 < 3) || (3 >= 0))  \rightarrow false
```
## Ordering Strings in Java

<table>
<thead>
<tr>
<th>Original collection of strings</th>
<th>Collection sorted (in increasing order) by Java:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;airplane&quot;, &quot;Sam&quot;, &quot;90&quot;, &quot;137&quot;, &quot;HTML&quot;,</td>
<td>&quot;137&quot;, &quot;90&quot;, &quot;HTML&quot;, &quot;Sam&quot;, &quot;airplane&quot;, Note: “137” and “90” are compared as strings. Uppercase letters compare less than lowercase (that is why “HTML” and “Sam” come before “airplane”</td>
</tr>
</tbody>
</table>
## Comparisons of Strings:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s1.equals(s2)</code></td>
<td><code>s1</code> has the same content as <code>s2</code>: same symbols, same capitalization (upper or lower case), in same order. E.g. no two strings in this list are equal: &quot;Sam&quot;, &quot;SAM&quot;, &quot;sam&quot;, &quot;aSm&quot;,</td>
</tr>
<tr>
<td><code>s1.compareTo(s2) &lt; 0</code></td>
<td><code>s1</code> comes before <code>s2</code> in Java's version of alphabetical order</td>
</tr>
<tr>
<td><code>s1.compareTo(s2) == 0</code></td>
<td><code>s1</code> equals <code>s2</code></td>
</tr>
<tr>
<td><code>s1.compareTo(s2) &gt; 0</code></td>
<td><code>s1</code> comes after <code>s2</code> in Java's version of alphabetical order</td>
</tr>
</tbody>
</table>

Hint to remember `compareTo` behavior: think of it as computing `s1-s2`.

To compare strings you do the same process as for a word look-up in a dictionary, but for the order of characters look at an ASCII table.
Character ordering

• 0 < 1 < ... < 9 < ... < A < B < .... < Z < ... < a < b ... < z

• Each character has a ‘code’. This code is used to:
  – Store/represent the character (in binary)
  – Compare characters (based on the value of the code)

• Encoding
  – The mapping between characters and codes
  – There are several encodings: ASCII (old) UTF-8 (popular)
// What will these lines print?
// Discuss WHAT EACH PIECE OF CODE DOES: "6a: " + ("h".compareTo("H") < 0)
System.out.println("1 : " + "hello".equals("hello") );
System.out.println("2 : " + "hello".equals("Hello") );
System.out.println("3a: " + "hello".compareTo("hello") );
System.out.println("3c: " + "world".compareTo("hello") );
System.out.println("3b: " + "hello".compareTo("world") );
System.out.println("4a: " + "h".compareTo("W") );
System.out.println("4b: " + "h".compareTo("World") );
System.out.println("4c: " + "hello".compareTo("World") );
// Add code below so that it prints true for all (e.g. "5a: true")
System.out.println("5a: " + "hello".compareTo("hello") );
System.out.println("5b: " + "world".compareTo("hello") );
// What will these lines print?
System.out.println("6a: " + ("h".compareTo("H") < 0) );
System.out.println("6b: " + ("h".compareTo("W") < 0) );
System.out.println("6c: " + ("hello".compareTo("World") < 0) );
System.out.println("7 : " + ("act".compareTo("actor") < 0) );
System.out.println("8 : " + ("97".compareTo("145") < 0) );
Bridging with other topics: Modulo operator % (together with if-else)

% computes the **remainder of division**. Returns an int.

- \( a \% b \) gives what remains from \( a \) after removing as many of \( b \) as possible.
- \( a \% b \) is \( a \), whenever \( a < b \) (I cannot fit \( b \) in \( a \) at all, therefore all of \( a \) is the remainder.)
  - Example: \( 5 \% 7 \) is 5

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>7%5</td>
<td>2</td>
</tr>
<tr>
<td>5%7</td>
<td>5</td>
</tr>
<tr>
<td>39 % 12</td>
<td>3</td>
</tr>
<tr>
<td>100%25</td>
<td>0</td>
</tr>
<tr>
<td>125%25</td>
<td>0</td>
</tr>
</tbody>
</table>

**Typical Usage**

<table>
<thead>
<tr>
<th>Example applications (% will be used in an if-else)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify odd/even</td>
</tr>
<tr>
<td>No TV on even days.</td>
</tr>
<tr>
<td>Identify white squares on a chess board.</td>
</tr>
<tr>
<td>Identify every b-th event</td>
</tr>
<tr>
<td>Every b days I run my Sprinklers. Use (day%b == 0)</td>
</tr>
<tr>
<td>Produce patterns (using loops)</td>
</tr>
<tr>
<td>Produce: <strong>._</strong>.<em>**.</em>** (user chooses length)</td>
</tr>
<tr>
<td>Produce a chess board of size n</td>
</tr>
</tbody>
</table>
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);
        }
    }
}

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"
Smart use of indexOf()

Write a program that:
- Asks the user to enter a single letter.
- If the user enters a string with more 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.

See:
- Smart usage of indexOf
- using System.exit(0) to force the program to end.

```java
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);  // ends the program
}

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);
}
```

Without indexOf, you would have to use something like:

```java
(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"))
```
The `indexOf()` Method for strings

- 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.
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.
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 <= 18) {
            System.out.printf("You are a teenager.\n");
            System.out.printf("You are young.\n");
        }
        System.out.printf("Bye\n");
    }
}

Note: the else part of an if statement IS OPTIONAL. No else in this example.