

LABORATORY **2**
Leap Years

In the previous laboratory we examined programs that did simple computation. The programs were sequential, i.e., they performed all the instructions in the order they were listed in the main method. In this lab we will use a facility provided by all programming languages—the selection or choice of which instructions to execute based on conditions that are true or false. For example, if it were true that a value x was not 0, then we could execute an instruction that involved division by x . However, if x were 0, then a different set of instructions would have to be executed or the program would maybe even terminate. In Java these choices or selections are implemented using if statements.

Pre-Laboratory Reading

Boolean Expressions

Java has two special values: `true` and `false`. These are considered constants. They are the only values possible for boolean expressions. **By definition, a boolean expression is an expression that has either the value `true` or the value `false`.**

The usual comparison operators used for numbers in algebra are available:

<code>x < y</code>	true if x is less than y
<code>x == y</code>	true if x equals y
<code>x != y</code>	true if x is not equal to y
<code>x <= y</code>	true if x less than or equal to y
<code>x > y</code>	true if x is greater than y
<code>x >= y</code>	true if x is greater than or equal to y

In the examples above, we assume `x` and `y` are numeric values. If an expression is not true, then it must be false. Take note of how we ask about equality—two equal signs! This avoids confusion with assignment expressions, which have a single equal sign.

Boolean Operators—“and,” “or,” and “not”

Boolean expressions can be combined by using the operators `&&` (meaning “and”), `||` (meaning “or”), and `!` (meaning “not”). The `&&` has higher precedence than the `||`. The `!` operation has the highest precedence (without parentheses).

The “and” of two boolean expressions is true only if *both* expressions are true. The “or” of two boolean expressions is true provided *either* one is true *or* both are true. “Not” simply changes the truth value to its opposite.

In Java, the mathematical inequality $3 < x \leq 10$ is true for a value of `x` if the boolean expression

```
3 < x && x <= 10
```

is true. Notice that we must repeat the `x`. The mathematical expression is shorthand for saying `x` is greater than 3 *and* `x` is less than or equal to 10.

Selection Statements in Java—“if” and “if-else”

The “if” statement uses a boolean condition to determine whether or not to perform certain statements. It comes in two forms.

Simple “if”

```
if ( boolean condition )
    statement
```

The boolean condition is evaluated. If it is true, then the statement is executed. Otherwise, the statement is skipped and ignored.

“if-else”

```
if ( boolean condition )
    statement1
else
    statement2
```

The boolean condition is evaluated. If it is true, then `statement1` is executed. If it is false, `statement2` is executed. Only one of the statements is executed. The other is skipped.

Example 1

```
if (x < 0)
    System.out.println("Sorry, negative value");
else
    System.out.println("Square root is " + Math.sqrt(x));
```

This is a very simple example using the if-else form. Since we cannot take the square root of a negative value, it makes sense to check for that condition. Notice that if $x < 0$ is false, then x is greater than or equal to 0 and we can take its square root. Only one of the two statements will be done, and it depends entirely on the condition (and hence the value of x) as to which is executed.

Example 2

```
if (n == 0)
{
    System.out.println("Zero encountered. Terminating program!");
    System.exit(1);    // stop program immediately
}
```

This is an example of a *simple if*, but we have used a *compound statement*. We wanted to do two statements if $n == 0$ was true. The syntax only allows one statement! Java (and most other languages) has a simple fix—it allows multiple statements to be grouped together and count as a unit. Java uses the *curly braces* for this.

```
{
    statement
    statement
    ...
    statement
}
```

Compound statements have another special feature—unlike assignment statements they are never followed by a semicolon. The closing brace makes it unnecessary.

Finally, we need to make special mention of the method `exit()` in the `System` class. `System.exit(1)` and `System.exit(0)` both terminate the program immediately. By convention, the 0 value means nothing was wrong, while the 1 value means an error of some kind occurred. This is a rather severe action and often the displayed message is not seen if it is part of a visual component created by the program!

Review Questions

1. If x is an integer variable that has been declared and given a value, what Java statement will print “Zero” if it has the value 0 and “Not Zero” otherwise? Use `System.out.println("...")` for displaying the message.
2. Suppose x and y are two integer variables that have been declared and given values. Write a Java statement that will display the larger of the two values (without using `Math.max()`). Do not worry about whether they may be equal in value.

Multiple Options—
The if-else-if-else Pattern

Consider this case: a user enters an integer value that represents a test grade. Suppose the value is stored in the variable called `grade`. If the grading system is of the form

```
90 and above is an A
80-89 is a B
70-79 is a C
```

60-69 is a D
below 60 is a F

we could write the following code to display the letter grade:

```

if (grade >= 90)
{
    System.out.println("A");
}
else
{
    if (grade >= 80 && grade < 90)
    {
        System.out.println("B");
    }
    else
    {
        if (grade >= 70 && grade < 80)
        {
            System.out.println("C");
        }
        else
        {
            if (grade >= 60 && grade < 70)
            {
                System.out.println("D");
            }
            else
            {
                System.out.println("F");
            }
        }
    }
}
}

```

This shows that a statement that is the object of an if condition can itself be another if or if-else statement. Carefully connect the corresponding opening and closing braces to see the structure.

This is correct, but the indentation is somewhat annoying. We essentially are handling multiple options, where only one is true. In such a case we can shorten the code using the style shown below.

```

if (grade >= 90)
{
    System.out.println("A");
}
else if (grade >= 80 && grade < 90)
{
    System.out.println("B");
}
else if (grade >= 70 && grade < 80)
{
    System.out.println("C");
}
else if (grade >= 60 && grade < 70)
{
    System.out.println("D");
}
else
{
    System.out.println("F");
}

```

Notice how much more readable this form is. This is our choice when we have more than two options. Notice that the last option is simply an else with no condition. This option is automatically done if no previous conditions are true.

In this form only the *first* true condition has its statement(s) executed.

Leap Year Defined

The following was a feature of the University of Kansas' online news.¹ It was taken from the Web page located at <http://www.ur.ku.edu/News/96N/FebNews/Feb20/leapyear.html>. It is dated February 20, 1996.

SO YOU THOUGHT LEAP YEAR CAME EVERY FOUR YEARS!

Lawrence, Kansas - Leap year doesn't come every four years—not always.

True, 1996 is a leap year, and so we add a day to February. The year 2000, a century year, will be a leap year, too. But if every fourth year were a leap year, our calendars and our seasons would gradually grow apart, said Barbara Anthony-Twarog, professor of physics and astronomy at the University of Kansas.

So some years aren't leap years although they're divisible by four. Anthony-Twarog explains why.

Leap years are needed to bring together our two methods of calculating the length of a year. One measure is the time the earth takes to complete an orbit around the sun. The other is the number of days, or Earth rotations, in that period, Anthony-Twarog said.

More than 2,000 years ago, astronomers knew that the difference between the two methods of calculating is about a quarter of a day. They figured the orbital year at 365.2422 days by noting the interval between vernal equinoxes. The vernal equinox occurs in the spring, around March 21.

The Julian calendar, produced by Julius Caesar, added a day every four years to account for that leftover fraction of a day, Anthony-Twarog said. This calendar worked well for a while, but by the late 16th century, when the calendar read March 21, spring was lagging two weeks behind, according to the sun's position and the weather.

In the late 16th century Pope Gregory XIII commanded radical calendar reform to avert further embarrassment. The Gregorian plan went into effect in 1582 in the Roman Catholic countries of Europe, but not without some pain, Anthony-Twarog said.

Ten days were dropped from the calendar—in October of that year—to adjust matters. How would you feel if someone dropped 10 days out of a month but still charged you for a full month's rent?

Protestant governments and countries obedient to the Eastern Rite churches ignored the mandate. England and its colonies held out until 1752, and Russia didn't alter its calendar until 1917, Anthony-Twarog said.

How does the Gregorian calendar work? Knowing that the year must be 365.2422 days long, the first step is to keep the scheme of adding a day every fourth year. That makes the average year 365.25 days long—a bit too long.

Removing one leap-day per century brings the calendar down to 365.2400 days, so century years are generally not leap years. Adding back century years divisible by 400 makes the year length 365.2425 days long, close enough to the Earth's orbital period that the error is less than one day in 3,300 years. So the year 2000 will be a leap year. The next century year that will be a leap year is 2400.

1. Copyright 1996, the University of Kansas Office of University Relations. Used with permission.

Selecting Proper Code—A Paper and Pencil Exercise

Using the if Statement to Check for Leap Years

We might want to write a program that will determine if a year typed in at the keyboard is a leap year based on the following definition:

A year in the current calendar system, i.e., after 1582, is a leap year if it is divisible by 4, with the exception that if it is divisible by 100 but not by 400, it is not a leap year.

So 1933 is not a leap year, but 1960 is a leap year. 1700 is not a leap year, but 1600 is a leap year. Keep the examples in mind as you do the rest of the lab. (Note that if a year can be divided by 100 it can be divided by 4. Similarly, any year that can be divided by 400 can be divided by 100 and by 4.)

In practice, the year should be at least 1582. (Why?) However, we will not check for this in the code that we will be examining.

Divisible By...

When one has two integers, A and B, and B is not 0, then A is *divisible by* B if $A\%B$ is 0. For example, a number is *even* provided it is divisible by 2. In Java we might write

```
if (x % 2 == 0) // is x even?
    System.out.println("Even");
else
    System.out.println("Odd");
```

What Works?

For each of the following options, A, B, C, and D, you will see statements that are part of a Java program. Assume that the variable `year` was given a value that was entered by a user. Assume that it was checked to see that it was positive and greater than 1582. Read each option carefully. There should be no syntax errors.

Determine which options correctly determine if the year is a leap year and print the appropriate message.

You should try some values and check them by *tracing*. Tracing is the method of simulating the actions of the computer with paper and pencil.

At least one is incorrect, and at least one is correct. Which are they? For the one(s) with incorrect results, indicate a year for which the code doesn't work properly.

Option A

```
if (year % 4 == 0 && year % 400 == 0) {
    System.out.println("It's a leap year!");
}
else {
    System.out.println("It's NOT a leap year.");
}
```

Option B

```
if (year % 4 != 0) {
    System.out.println("It's NOT a leap year.");
}
else {
    if (year % 100 == 0){
        System.out.println("It's NOT a leap year.");
    }
    else {
        System.out.println("It's a leap year!");
    }
}
```

Option C

```
if (year % 400 == 0) {
```

```

        System.out.println("It's a leap year!");
    }
    else if (year % 100 == 0) {
        System.out.println("It's NOT a leap year.");
    }
    else if (year % 4 == 0) {
        System.out.println("It's a leap year!");
    }
    else {
        System.out.println("It's NOT a leap year.");
    }
}

```

Option D

```

if (year % 4 != 0) {
    System.out.println("It's NOT a leap year.");
}
else {
    if (year % 100 == 0 && year % 400 != 0) {
        System.out.println("It's NOT a leap year.");
    }
    else {
        System.out.println("It's a leap year!");
    }
}
}

```

The Laboratory

Objectives

- To successfully write simple programs that involve if statements.
- To gain practice in the use of boolean operators like && and ||.

A Guide to the Laboratory

- Task 1 requires that you take the program `RectData.java` that you wrote in the previous lab and modify it to avoid non-positive values. Turn in a printout of the modified program.
- Task 2 requires that you write a complete program `Options.java` that will accept an integer as input and display one of these messages: “positive,” “negative,” or “zero.” When this program is working properly, print out a copy to turn in.
- Task 3 requires that you write a complete program `Big3.java` to read in three integers and display the largest value without using any of the `Math` methods. Turn in a printout of your solution.

Preliminaries

There are no files to download for this laboratory.

**Task 1:
Avoiding Bad
Input**

The program `RectData.java` reads in two numbers that represent the length and width of a rectangle. In the previous lab, we didn’t check if the numbers were 0 or negative values, which wouldn’t be meaningful.

Open `RectData.java` in the IDE and modify it to check for 0 and negative values of the two numbers that are entered. Do the computation only if both values are positive. Otherwise, execute the statement

```
vf.println("Bad input");
```

if you used a `ViewFrame` object called `vf` in your program.

Print out a copy of the revised program to turn in. Be sure to include your name(s) in a comment. If you have changed lab partners, use the current names.

Task 2: Multiple Options

Write a new program called `Options.java` that will request that the user enter an integer and then will display the message “positive,” “negative,” or “zero” if the value that was entered was greater than zero, less than zero, or equal to zero, respectively.

Turn in a printout with your name(s) on it.

Task 3: Finding the Largest Value

It is easy to find the largest of two values and print it out using an if statement. For example, if `a` and `b` are the values, then

```
if (a > b)
    System.out.println(" " + a);
else
    System.out.println(" " + b);
```

will work. (Of course, we may want to display the answer using a visual component such as a `ViewFrame` object and its `println()` method, but that is a simple modification.)

You are to create a new program in the file `Big3.java` that will request three integers from the user and display the largest of the values. Do *not* use the method `Math.max()`. Use a `ViewFrame` object to obtain input and display the answer.

Turn in a printout of your program with your name(s) on it when the program is working properly. Be sure to test it with all possible orders of three numbers.

Post-Laboratory Exercises

Acid or Alkaline?

Write a program called `PH.java` that will accept a value with a decimal point from the user. If the value is 0 or negative or if it is greater than 12.0, display a message indicating bad data. Otherwise, if the value entered is less than 6.0, display the message “acidic.” If it is greater than 6.0, display the message “alkaline.” If it equals 6.0, display the message “neutral.”

Quotients

Write a program in a file called `Quotients.java` that will read in two integers from the user. If either integer is negative, replace it by its absolute value. If either integer is 0, display the message “No quotient.” Otherwise, display the quotient of the larger value divided by the smaller value.

Leap Year

Select the leap year code option that you felt was correct from options A through D and write a complete program `LeapYear.java` to test it.

Warning Messages

LabPkg's `ViewFrame` has a useful method `showWarningMsg` that will display a warning message that will not go away until dismissed by the user. It solves the problem of killing the program before the user can read the message. For example, if `vf` is a `ViewFrame` object, then

```
if (n == 0)
{
    vf.showWarningMsg("Zero value encountered - quitting");
    System.exit(1);
}
```

will allow the user to read the message before the `ViewFrame` window disappears due to the execution of `System.exit(1)`.