Flow of Control

Chapter 3
The Conditional Operator

if (n1 > n2)
    max = n1;
else
    max = n2;

can be written as

max = (n1 > n2) ? n1 : n2;

• The ? and : together are call the conditional operator or ternary operator.
The Conditional Operator

• The conditional operator is useful with print and println statements.

```java
System.out.print("You worked " +
    ((hours > 1) ? "hours" ; "hour")
);```

The `exit` Method

- Sometimes a situation arises that makes continuing the program pointless.
- A program can be terminated normally by
  ```java
  System.exit(0).
  ```
The **exit** Method

**Example**

```java
if (numberOfWinners == 0) {
    System.out.println("Error: Dividing by zero.");
    System.exit (0);
} else {
    oneShare = payoff / numberOfWinners;
    System.out.println("Each winner will receive 
    $" + oneShare);
}
```
A Dialog Box for a Yes-or-No Question

• Used to present the user with a yes/no question
• The window contains
  ▪ The question text
  ▪ Two buttons labeled yes and no.
A Dialog Box for a Yes-or-No Question

• Example

```java
int answer = JOptionPane.showConfirmDialog(null,
"End program?",
"Click Yes or No:",
JOptionPane.YES_NO_OPTION);
if (answer == JOptionPane.YES_OPTION)
    System.exit(0);
else if (answer == JOptionPane.NO_OPTION)
    System.out.println("One more time");
```
A Dialog Box for a Yes-or-No Question

- Figure 3.11 A Yes-or No-Dialog Box
The Type **boolean**

- The type **boolean** is a primitive type with only two values: **true** and **false**.
- Boolean variables can make programs more readable.

```java
if (systemsAreOK)
```

instead of

```java
if((temperature <= 100) && (thrust >= 12000) && (cabinPressure > 30) && ...)
```
Boolean Expressions and Variables

- Variables, constants, and expressions of type `boolean` all evaluate to either `true` or `false`.
- A boolean variable can be given the value of a boolean expression by using an assignment operator.

```java
boolean isPositive = (number > 0);
...
if (isPositive) ...
```
Naming Boolean Variables

• Choose names such as `isPositive` or `systemsAreOk`.
• Avoid names such as `numberSign` or `systemStatus`.
Precedence Rules

- Parentheses should be used to indicate the order of operations.
- When parentheses are omitted, the order of operation is determined by *precedence rules*.
Precedence Rules

• Operations with higher precedence are performed before operations with lower precedence.
• Operations with equal precedence are done left-to-right (except for unary operations which are done right-to-left).
Precedence Rules

• Figure 3.9

**Highest Precedence**

First: the unary operators +, −, ++, −−, and !
Second: the binary arithmetic operators *, /, %
Third: the binary arithmetic operators +, −
Fourth: the boolean operators <, >, <=, >=
Fifth: the boolean operators ==, !=
Sixth: the boolean operator &
Seventh: the boolean operator |
Eighth: the boolean operator &&
Ninth: the boolean operator ||

**Lowest Precedence**
Precedence Rules

• In what order are the operations performed?

\[
\text{score} < \frac{\text{min}}{2} - 10 \text{ || score} > 90
\]
Precedence Rules

• In what order are the operations performed?

score < min/2 - 10 || score > 90
score < (min/2) - 10 || score > 90
score < ((min/2) - 10) || score > 90
(score < ((min/2) - 10)) || score > 90
(score < ((min/2) - 10)) || (score > 90)
Short-circuit Evaluation

• Sometimes only part of a boolean expression needs to be evaluated to determine the value of the entire expression.
  ▪ If the first operand associated with an || is true, the expression is true.
  ▪ If the first operand associated with an && is false, the expression is false.
• This is called short-circuit or lazy evaluation.
Short-circuit Evaluation

• Short-circuit evaluation is not only efficient, sometimes it is essential!

• A run-time error can result, for example, from an attempt to divide by zero.

  ```java
  if ((number != 0) && (sum/number > 5))
  ```

• Complete evaluation can be achieved by substituting `&` for `&&` or `|` for `||`.
The **switch** Statement

- The **switch** statement is a multiway branch that makes a decision based on an *integral* (integer or character) expression.
  - Java 7 allows String expressions
- The **switch** statement begins with the keyword **switch** followed by an integral expression in parentheses and called the *controlling expression*. 
The `switch` Statement

- A list of cases follows, enclosed in braces.
- Each case consists of the keyword `case` followed by
  - A constant called the `case label`
  - A colon
  - A list of statements.
- The list is searched for a case label matching the controlling expression.
The `switch` Statement

• The action associated with a matching case label is executed.

• If no match is found, the case labeled `default` is executed.
  ▪ The `default` case is optional, but recommended, even if it simply prints a message.

• Repeated case labels are not allowed.
The **switch** Statement

- **Syntax**

  ```java
  switch (Controlling_Expression) {
    case Case_Label:  
      Statement(s);
      break;
    case Case_Label:  
      ...  
    default:  
      ...  
  }
  ```
The **switch** Statement

- View sample program Listing 3.5

```java
class MultipleBirths

Enter number of babies: 1
Congratulations.

Enter number of babies: 3
Wow. Triplets.

Enter number of babies: 4
Unbelievable; 4 babies.

Enter number of babies: 6
I don't believe you.
```

Sample screen output
The `switch` Statement

- The action for each case typically ends with the word `break`.
- The optional `break` statement prevents the consideration of other cases.
- The controlling expression can be anything that evaluates to an integral type.
Enumerations

• Consider a need to restrict contents of a variable to certain values
• An enumeration lists the values a variable can have
• Example

```java
enum MovieRating {E, A, B}
MovieRating rating;
rating = MovieRating.A;
```
Enumerations

- Now possible to use in a `switch` statement

```
switch (rating)
{
    case E: //Excellent
        System.out.println("You must see this movie!");
        break;
    case A: //Average
        System.out.println("This movie is OK, but not great.");
        break;
    case B: // Bad
        System.out.println("Skip it!");
        break;
    default:
        System.out.println("Something is wrong.");
}
```
Enumerations

• An even better choice of descriptive identifiers for the constants

```java
enum MovieRating
    {EXCELLENT, AVERAGE, BAD}
rating = MovieRating.AVERAGE;

case EXCELLENT:  ...
```
Summary

• You have learned about Java branching statements.
• You have learned about the type `boolean`.
• You have learned to use the `JOptionPane` yes/no window.