CONDITIONAL EXECUTION:
PART 2

x > y?

max = x;
max = y;

logical AND  logical OR  logical NOT
&&  ||  !
Outline

• Review: The \texttt{if-else} statement
• The \texttt{switch} statement
• A look at enumerations
Review

• Conditional Execution
  • if … then
  • if … then … else
  • Nested if … then statements
Flow of Control

- *Flow of control* is the order in which a program performs actions.
  - Prior to conditionals, the order has been sequential.

- Conditional execution: A *branching statement* chooses between two or more possible actions.

- Iteration: A *loop statement* repeats an action until a stopping condition occurs.
Multibranch `if-else` Statements

• Syntax

```plaintext
if (Boolean_Expression_1)
    Statement_1
else if (Boolean_Expression_2)
    Statement_2
else if (Boolean_Expression_3)
    Statement_3
else if ...
else
    Default_Statement
```
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**.
• A shortcut for the full if...else statement
• Should only be used for very short pieces of code
The Conditional Operator

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

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

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

• *Complete evaluation* can be achieved by substituting `&` for `&&` or `|` for `||`.
Expanded Precedence Rules

*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?

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

score < (min/2 - 10 || score > 90)
Input Validation

• You should check your input to ensure that it is within a valid or reasonable range. For example, consider a program that converts feet to inches. You might write the following:

```java
int feet = keyboard.nextInt();
int inches = feet * 12;
```

• What if:
  • The user types a negative number for feet?
  • The user enters an unreasonable value like 100? Or a number larger than can be stored in an int? (2,147,483,647)
Input Validation

• Address these problems by ensuring that the entered values are reasonable:

```java
int feet = keyboard.nextInt();
if ((feet >= 0) && (feet < 10))
{
    int inches = feet * 12;
    ...
}
```
The `switch` Statement

- **Syntax**

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

- The `switch` statement is a multiway branch that makes a decision based on an integral (integer or character) expression and in newer versions of Java, also the String type.
- 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

• 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.
int month = Integer.parseInt(args[0]);

switch (month) {
    case 1: System.out.println("January");
            break;
    case 2: System.out.println("February");
            break;
    case 3: System.out.println("March");
            break;
    case 4: System.out.println("April");
            break;
    case 5: System.out.println("May");
            break;
    case 6: System.out.println("June");
            break;
    ...
    case 12: System.out.println("December");
             break;
    default: System.out.println("Invalid month");
             break;
}
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

```java
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: ...
```
enum Month {JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER}

public static void main(String[] args)
{
    Month month = Month.SEPTEMBER;
    int numDays = 0;
    switch (month)
    {
        case JANUARY:
        case MARCH:
        case MAY:
        case JULY:
        case AUGUST:
        case OCTOBER:
        case DECEMBER:
            numDays = 31;
            break;
        case APRIL:
        case JUNE:
        case SEPTEMBER:
        case NOVEMBER:
            numDays = 30;
            break;
        case FEBRUARY:
            numDays = 28;
            break;
        default: System.out.println("Invalid month");
            break;
    }
}
Summary

• Review: The if-else statement
• The switch statement
• A look at enumerations
You Try It

• Write a program with a switch statement that reports whether you passed or failed based on the character entered at the command line. A, B, and C are passing, D and F are not.

• Submit your program, named Grades.java, to the Activity01 dropbox on Moodle. You get 1 extra credit point for turning something in, 2 points if it is correct. Hmmm. Let’s say you get 1 additional point if you use an enumeration and get the code right. So that means up to 3 points for this one.

• Don’t forget! Always put your (full) name and a description in a header comment!