Conditional Programming
Chapter Outline

• Control Flow of a Program
• The if statement
  • The if-else statement
• Logical Operators
• The switch statement
• The conditional operator
Introduction

• So far, all the programs we have written executed all the statements they contained.
• Suppose we want to write a program which asks the user to enter two numbers and then displays only the larger of the two.
• This involves executing certain statements in some circumstances, and different statements in other circumstances.
Flow of Control

• By default, the order of statement execution through a method is linear
  • one statement after the other is executed, in textual order (top of page, downwards to end of page)
• Some programming statements modify that order, allowing us to:
  • decide whether or not to execute a particular statement
  • perform a statement over and over repetitively (while)
• The order of statement execution is called the flow of control
Conditional Statement

- A *conditional statement* lets us choose which statement will be executed next
- Therefore they are sometimes called *selection statements*
- Conditional statements give us the power to make basic decisions
- Java's conditional statements are the *if* statement, the *if-else* statement, and the *switch* statement
Conditional Programming Part I
The if Statement
The if Statement

- The if statement has the following syntax:

```java
if (condition)
statement;
```

- if is a Java reserved word
- The condition must be a boolean expression. It must evaluate to either true or false.
- If the condition is true, the statement is executed. If it is false, the statement is skipped.
If Statement Flow Diagram
Comparison Operators

- A condition often uses one of Java's equality operators or relational operators, which all return boolean results:
  
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>equal to</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>not equal to</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal to</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal to</td>
</tr>
</tbody>
</table>

- Note the difference between the equality operator (==) and the assignment operator (=)
More on Comparison Operators

- Equality (==) and inequality (!=) operators apply to values that have any type.
- The other comparison operators (<, <=, >, >=) only apply to values which have a numeric type (byte, short, int, long, float, double) or that have type char.
- They do not apply to values that have type boolean.
- Even though the operands of a comparison operator may have various types, the type of the result of the comparison is always the same: boolean.
- This implies that the result of a comparison is always true or false.
Comparison Operator Examples (1)

- `(denominator == 0)`
  - Evaluates to `true` if `denominator` is equal to 0, evaluates to `false` otherwise
- `(denominator != 0)`
  - Evaluates to `true` if `denominator` is not equal to 0, evaluates to `false` otherwise
- `(balance > amount)`
  - Evaluates to `true` if the value of `balance` is strictly greater than the value of `amount`, evaluates to `false` otherwise
- `(balance < amount)`
  - Evaluates to `true` if the value of `balance` is strictly less than the value of `amount`, evaluates to `false` otherwise
Comparison Operator Examples (2)

• \((\text{balance} \geq \text{amount})\)
  • Evaluates to \text{true} if the value of \text{balance} is greater than or equal to the value of \text{amount}, evaluates to \text{false} otherwise
  • Note that using \(=>\) will not work

• \((\text{balance} \leq \text{amount})\)
  • Evaluates to \text{true} if the value of \text{balance} is less than or equal to the value of \text{amount}, evaluates to \text{false} otherwise
  • Again, note that using \(=\) will not work
if Statement Exercise

• Complete the `main()` method of the `BusRide` class by adding code to check whether the number of passengers is greater than the capacity of the bus

• If it is, then you should display a message asking for $x$ (where $x$ is the number of passengers in excess of the capacity of the bus) volunteers to travel in "economy class": on the roof

• Regardless of whether the number of passengers exceeds the capacity of the bus, you should display "Let's go!" after you have displayed whether or not volunteers are needed for "economy class"
import java.util.Scanner;

public class BusRide {
    public static void main (String[] args) {
        final int CAPACITY = 56;
        int passengers;

        Scanner keyboard = new Scanner(System.in);
        System.out.print("Enter the number of people" +
                        " that want to get on the bus: ");
        passengers = keyboard.nextInt();
        // Add your code here
    }
}

Comp 202 - Conditionals, © 2011 Jörg Kienzle and others

---

McGill
The if–else Statement

• An *else clause* can be added to an *if* statement to make it an *if–else* statement:
  
  ```java
  if (condition)
      statement1;
  else
      statement2;
  ```

• If the condition is *true*, *statement1* is executed; if the condition is *false*, *statement2* is executed

• One or the other will be executed, but not both
if-else Statement Flow Diagram

class diagram
  condition?
  true
  statement1
  false
  statement2
if-else Statement Exercise

• Complete the main() method of the Wages class by adding code to compute the gross earnings of an employee
  • If the employee has worked more than 40 hours during his / her work week, he / she should be paid 1.5 times his / her hourly wage for all hours worked in excess of 40
  • Can you rewrite your code using only a regular if-statement (that is, one that does not have an else clause)?
public static void main (String[] args) {

    final double RATE = 8.25;  // regular pay rate
    final int STANDARD = 40;   // standard hours / week
    double pay = 0.0;
    int hours;
    Scanner keyboard = new Scanner(System.in);
    System.out.print("Enter number of hours worked:");
    hours = keyboard.nextInt();

    // Add your code here

    System.out.println("Your pay is: "+pay);
}
Block Statements

• Several statements can be grouped together into a _block statement_
• A block is delimited by braces ( `{ . . . }` )
• A block statement can be used wherever a statement is called for in the Java syntax
• For example, in an _if-else_ statement, the _if_ portion, or the _else_ portion, or both, could be block statements
Block Statements Example

```java
if (condition) {
    Statements executed if condition evaluates to true
}
else {
    Statements executed if condition evaluates to false
}
```
Block Statements vs. Indentation

• What will happen if the following code fragment is executed:
  ```java
  if (a < b)
      System.out.println(a);
      System.out.println(b);
  ```
• The second `println()` call will be executed regardless of whether the condition evaluates to true or false
• Syntax determines which clause a statement belongs to
  • Indentation has nothing to do with this
  • Tip #1: Always use block statements with `if` and `if-else` statements, even when the block statement contains only one statement
  • Tip #2: Always have consistent indentation
Block Statements Example

- For example, the following code fragment will cause a compilation error:

```c
if (condition)
    statement1;
    statement2;
else
    statement3;
```

- When the compiler reaches `statement2`, it will assume that the `if` is part of an `if` statement (not an `if-else` statement), and that `statement2` should be executed regardless of whether `condition` evaluates to true or false.

- Then, when the compiler reaches the `else`, it will not be able to match it with any `if` statement, and thus will generate an error.
Block Statement Exercise

• Complete the `main()` method of the `GuessGame` class by adding code to determine whether the user won or not
  • The player wins if he / she is able to guess the number that the program chose at random
  • If the player wins, you should display a message stating that he / she has won, and the amount of money he / she has won
  • If the player loses, you should display a message stating that he / she has lost, what the number chosen by the program was, and the amount the player has lost
  • Whether the player wins or loses, you should display the amount of money he / she has after playing the game
import java.util.Scanner; import java.util.Random;

public class GuessGame {
    public static void main(String[] args) {
        final int UPPER_BOUND = 10;
        Scanner keyboard = new Scanner(System.in);
        Random randomSource = new Random();
        double money;
        double betAmount;
        int myNumber;
        int yourNumber;
        System.out.print("How much money to you have? ");
        money = keyboard.nextDouble();

        // Continued on next slide
// Continued from previous slide
betAmount = keyboard.nextDouble();
myNumber = randomSource.nextInt(UPPER_BOUND) + 1;
System.out.print("I've chosen a number between 1" +
    " and " + UPPER_BOUND + ". Try to guess it: ");
yourNumber = keyboard.nextInt();

// Add your code here
}
Nested if Statements

• The statement executed as a result of an if statement or else clause could be another if statement
• These are called nested if statements
• Indentation does not determine which if and else matches with. It is determined by syntax (i.e. by the order of the clauses or {})
  • An else clause matches with the nearest if
• Note: Dr.Java does not automatically perform proper indentation for nested statements
  • Solution: use {}
Nested if Statement Example

- One can write nested if-else statements like this:

```java
if (condition1)
  if (condition2)
    statement1;
  else
    statement2;
else
  if (condition3)
    statement3;
  else
    statement4;
```
Control Flow of Nested \textbf{if} Statement

- \texttt{condition?} \\
  \hspace{1cm} \texttt{true} \\
  \hspace{1cm} \texttt{false} \\
  \hspace{1cm} \texttt{condition?} \\
  \hspace{1cm} \texttt{false} \\
  \hspace{1cm} \texttt{true} \\
  \hspace{1cm} \texttt{false} \\
  \hspace{1cm} \texttt{statement1} \\
  \hspace{1cm} \texttt{statement2} \\
  \hspace{1cm} \texttt{statement3} \\
  \hspace{1cm} \texttt{statement4}
Nested if Statement Exercise

• Complete the `main()` method of the `MinOfThree` class by adding code which determines which of the three numbers entered by the user is the smallest number, and displays that number.

• Can you write this code both with and without using block statements?
public static void main(String[] args) {
    int num1, num2, num3, min = 0;
    Scanner scan = new Scanner(System.in);
    System.out.println("Enter three integers: ");
    num1 = scan.nextInt();
    num2 = scan.nextInt();
    num3 = scan.nextInt();

    // add code here

    System.out.println("Minimum value: " + min);
}
More Than Two Execution Branches

• Nested statements are needed when there are more than two branches / conditions
• An `if-(else-if)-else` statement allows several execution branches.

```c
if (condition) {
    statement1;
} else if (condition) {
    statement2;
} else {
    statement3;
}
```

```c
if (condition) {
    statement1;
} else {
    if (condition) {
        statement2;
    } else {
        statement3;
    }
}
```
Boolean Expressions

• Like an arithmetic expression, a boolean expression is a combinations of operators and operands, and it evaluates to a boolean value (i.e. true or false)

• A boolean expression can be:
  • The comparison of two values using a comparison operator
  • A variable which has type boolean
  • A literal which has type boolean (true or false)
  • The negation of another boolean expression using the ! operator
  • The combination of two or more other boolean expressions using the && or || operators
Comparison Operator Precedence

- Comparison operators have lower precedence than arithmetic operators, but higher precedence than the assignment operator.
- Therefore, the order of evaluation for this code fragment is the following (assume that a, c, d, and e have a numeric type):

```java
boolean b = a > c * d + e;
```

![Diagram showing the order of evaluation](image)
Comparing Characters

• We can use the logical operators on character data
• The results are based on the Unicode character set
• The following condition is true because the character '+' comes before the character 'J' in Unicode:

```java
if ('+' < 'J')
    System.out.println("+ is less than J");
```

• The uppercase alphabet (A-Z) and the lowercase alphabet (a-z) both appear in alphabetical order in Unicode
Comparing Characters (2)

- In the Unicode character set, the numbers assigned to upper-case alphabetic characters ('A' - 'Z'), lower-case alphabetic characters ('a' - 'z') and digits ('0' - '9') not only follow the expected order, but are consecutive.
  - If 'A' is assigned the number $x$, then 'B' is assigned the number $x + 1$, 'C' is assigned the number $x + 2$, ...
  - If 'a' is assigned the number $y$, then 'b' is assigned the number $y + 1$, 'c' is assigned the number $y + 2$, ...
  - If '0' is assigned the number $z$, then '1' is assigned the number $z + 1$, '2' is assigned the number $z + 2$, ...
- Do not hesitate to use this property of characters in your programs.
Comparing Strings

- In Java, the `String` class represents a sequence of characters
- A character string in Java is an object
- *We cannot use the logical operators to compare objects*
- The `equals` method can be called on a `String` to determine if two strings contain exactly the same characters in the same order (even constants)
- The `String` class also contains a method called `compareTo` to determine if one string comes before another alphabetically (as determined by the Unicode character set)
Comparing Floating Point Values

• We also have to be careful when comparing two floating point values (float or double) for equality
• You should rarely use the equality operator (==) when comparing two floats
• In many situations, you might consider two floating point numbers to be "close enough" even if they aren't exactly equal
• Therefore, to determine the equality of two floats, you may want to check if their difference is below a certain threshold:
  \[
  \text{if} \ (\text{Math.abs}(f1 - f2) < 0.00001) \\
  \text{System.out.println("Essentially equal");}
  \]
Logical Operators

• Boolean expressions can also use the following logical operators:

\[
\begin{align*}
! & \quad \text{logical not} \\
\&\& & \quad \text{logical and} \\
|\ | & \quad \text{logical or}
\end{align*}
\]

• All three operators take operands of type `boolean` and produce results of type `boolean`.

• Logical `not` is a unary operator (it has one operand), but logical `and` and logical `or` are binary operators (they each have two operands).
Logical Operator Examples

```java
boolean choice = false;
if (!choice)
    System.out.println("Go");
else
    System.out.println("Stop");

if (!(x>5)) ...

if ( (x>5) && (y<10) )
    choice = true;
else
    choice = false;
```
Logical not

- The logical not operation is also called logical negation or logical complement.
- If some boolean condition $a$ is true, then $\neg a$ is false; if $a$ is false, then $\neg a$ is true.
- Logical expressions can be shown using truth tables.

<table>
<thead>
<tr>
<th>$a$</th>
<th>$\neg a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
Logical *and* and *or* and *xor*

- The logical *and* expression
  \[ a \land b \]
  is true if both \( a \) and \( b \) are true, and false otherwise

- The logical *or* expression
  \[ a \lor b \]
  is true if \( a \) or \( b \) or both are true, and false otherwise

- The logical *xor* expression
  \[ a \oplus b \]
  is *true* if \( a \) or \( b \) are different, and false if \( a \) and \( b \) are equal
Logical *and* and *or* and *xor* Truth Table

- Since `&&` and `||` and `^` each have two operands, there are four possible combinations of *true* and *false*

|    |    | `a && b` | `a || b` | `a ^ b` |
|----|----|---------|---------|--------|
| true | true | true    | true    | false  |
| true | false | false   | true    | true   |
| false| true | false   | true    | true   |
| false| false| false   | false   | false  |
Logical Operator Precedence (1)

- Like arithmetic operators, logical operators have precedence rules among themselves.
- Logical operator `!` has higher precedence than logical operator `&&`, and logical operator `&&` has higher precedence than logical operator `|`
- Consider the following expression (assume that `a`, `b`, and `c` are all of type `boolean`):

\[
a \; || \; b \; && \; !c
\]
Logical Operator Precedence (2)

• Logical operators also have precedence rules relative to other kinds of operators.
• Logical operators have lower precedence than comparison operators, but higher precedence than the assignment operator.
• Therefore, the order of evaluation for this code fragment is the following (assume that `a` has type `boolean`):

```java
boolean b = a && c < d;
```
Short-Circuit Evaluation

• Logical operators `&&` and `||` are evaluated in short-circuit.
• If the first operand of a `&&` operator evaluates to false, the second operand is not evaluated.
  • There is no need to do so considering that a logical `and` expression always evaluates to `false` as soon as one of its operands evaluates to `false`.
• Likewise, if the first operand of a `||` operator evaluates to true, the second operand is not evaluated.
  • Again, there is no need to do so considering that a logical `or` expression always evaluates to `true` as soon as one of its operands evaluates to `true`.
Boolean Expression Exercises (1)

Write boolean expressions that evaluates to true if and only if the given condition is true

• The absolute value of variable \(a\) (of type int) is greater than 100

• The values of variables \(a\), \(b\), and \(c\) are all different

• The character stored in either variable \(c1\), or variable \(c2\) (both of type char), or both, is a digit
Boolean Expression Exercises (2)

Write boolean expressions that evaluates to true if and only if the given condition is true

• The value stored in *exactly one* of the two variables `a` and `b` (both of type `int`) is equal to 0
Conditional Programming Part III
The switch Statement
The switch Statement

- The switch statement provides another means to decide which statement to execute next
- The switch statement evaluates an expression, then attempts to match the result to one of several possible cases
- Each case contains a value and a list of statements
- The flow of control transfers to the case associated with the first value that it matches with (first come, first serve)
### switch Statement Syntax

- The general syntax of a switch statement is the following:

```
switch (expression) {
    case value1:
        statement-list1;
    case value2:
        statement-list2;
    case value3:
        statement-list3;
    // As many cases as needed
}
```

- *switch* and *case* are reserved words.
- *expression* is evaluated and compared to *value1*, then *value2*, etc...
- If *expression* evaluates to *value2*, control jumps to *statement-list2*.
The **break** Statement

- Often a **break** statement is used as the last statement in each case's statement list
- **break** is also a reserved word in Java
- A **break** statement causes control to transfer to the end of the switch statement
- *If a **break** statement is not used, the flow of control will continue into the next case, regardless of whether the value of the expression in the switch matches that case*
- Sometimes this can be helpful, but usually we only want to execute the statements associated with one case
Control Flow of switch Statement

evaluate expression

value1 → statement-list1
value2 → statement-list2
value... → ...
valuen → statement-list3
Control Flow of \textit{switch} with \textit{break}

evaluate expression

\begin{align*}
\text{value}_1 & \rightarrow \text{statement-list}_1 \\
\text{value}_2 & \rightarrow \text{statement-list}_2 + \text{break} \\
\text{value}_n & \rightarrow \text{statement-list}_3
\end{align*}

\textit{...}
**switch Statement Example**

```java
int age;
System.out.print("Enter your age: ");
age = keyboard.nextInt();

switch (age) {
    case 5:
        System.out.println("Five years old");
        break;
    case 10:
        age++;
        System.out.println(age);
    case 20:
        age--;
}
System.out.println(age);
```

What happens when the value of age is:
- 5?
- 10?
- 20?
- 3 or any other number?
switch Statement default case

• A switch statement can have an optional default case
• The default case has no associated value and simply uses the reserved word `default`
• If the default case is present, *control will transfer to it for all values which do not match a case*
• Because of this, though the default case can be positioned anywhere in the switch statement, it is usually placed at the end
• If there is no default case, and no other value matches, control falls through to the statement after the `switch`
switch with default Control Flow

- Evaluate expression
- Value1: statement-
  lst1+break
- Value2: statement-
  lst2+break
- Any other value: def-
  statement-
  lst
- Value...

COMP-202 - Conditionals, © 2011 Jörg Kienzle and others
switch without default

evaluate expression

- value1
  - statement-lst1+break

- value2
  - statement-lst2+break

- value...

- any other value
int section;
System.out.print("Enter your COMP-202 section: ");
section = keyboard.nextInt();

switch(section) {
  case 1:
  case 3:
    System.out.println("Your instructor’s name is Dan.");
    break;
  case 2:
    System.out.println("Your instructor’s name is Jörg.");
    break;
  default:
    System.out.println("How can that be?");
}
```java
int section;
System.out.print("Enter your COMP-202 section: ");
section = keyboard.nextInt();

switch(section) {
    default:
        System.out.println("How can that be?");
    case 1:
    case 3:
        System.out.println("Your instructor’s name is Dan.");
        break;
    case 2:
        System.out.println("Your instructor’s name is Jörg.");
        break;
}
```

What happens now?
More on switch Statements

• The expression of a switch statement must evaluate to a value of type char, byte, short or int; it cannot be a floating point value, a long, a boolean or a String
• Note that the implicit boolean expression in a switch statement is equality (it is never <, <=, >, nor >=)
• You cannot perform relational checks with a switch statement
• The value of each case must be a constant
• It cannot be a variable
switch Statement Exercise (1)

• Complete the `main()` method of the `Drinks` class by adding code that does the following:
  • If the user indicates that he / she wants to drink orange juice, the program should display two messages: "Vitamin C!" and "Your bones will thank you."
  • If the user indicates that he / she wants to drink milk, the program should display "Your bones will thank you."
  • If the user indicates that he / she wants to drink water, the program should display "The classics never die."
**switch Statement Exercise (2)**

- If the user indicates that he / she wants to drink wine, the program display a prompt asking him / her whether he wants red or white wine, and read the answer
  - If the user chooses red wine, the program should display "Good for your heart."
  - If the user chooses white wine, the program should display "Good for your lungs."
- If the user indicates that he / she wants beer, the program should display "Watch that belly!"
- If the user enters an invalid option, the program should display "That's not going to quench your thirst..."
public static void main(String[] args) {
    Scanner keyboard;
    int choice;

    keyboard = new Scanner(System.in);
    System.out.println("Here is the drinks menu : ");
    System.out.println("1. Orange juice");
    System.out.println("2. Milk");
    System.out.println("3. Water");
    System.out.println("4. Wine");
    System.out.println("5. Beer");
    System.out.print("What will it be ? ");
    choice = keyboard.nextInt();

    // Add your code here
}

Conditional Programming Part IV
The Conditional Operator
The Conditional Operator

- Java has a conditional operator that evaluates a boolean expression
- One of two other expressions is evaluated based on the result of the evaluation of that boolean expression
- The result of the chosen expression is the result of the entire conditional operator
- Its syntax is the following:
  
  condition ? expression1 : expression2

- If condition evaluates to true, then expression1 is evaluated; if it evaluates to false, then expression2 is evaluated
Conditional Operator Examples (1)

• The conditional operator is similar to an if–else statement, except that it is an expression that returns a value

• For example:

\[
larger = (\text{num}1 > \text{num}2) \ ? \ \text{num}1 : \text{num}2;
\]

• If num1 is greater than num2, then num1 is assigned to larger; otherwise, num2 is assigned to larger

• The conditional operator is ternary, meaning that it requires three operands
Conditional Operator Examples (2)

- Another example:

```java
System.out.println("Your change is " + count + " dime" + 
   ((count == 1) ? "" : "s");
```

- If `count` evaluates to 1, then "dime" is printed
- If `count` evaluates to any value other than 1, then an "s" is added at the end of "dime"
Conditional Operator Exercise (1)

• Complete the `main()` method of the `MoreWages` class by adding code to compute the gross earnings of an employee
  • If the employee has worked more than 40 hours during his / her work week, he / she should be paid 1.5 times his / her hourly wage for all hours worked in excess of 40
  • Use the conditional operator to determine whether the employee has worked more than 40 hours in his / her work week
public static void main (String[] args) {
    final double RATE = 8.25;  // regular pay rate
    final int STANDARD = 40;   // standard hours / week
    double pay = 0.0;
    int hours;
    Scanner keyboard = new Scanner(System.in);
    System.out.print("Enter number of hours worked:");
    hours = keyboard.nextInt();

    // Add your code here

    System.out.println("Your pay is: " + pay);
}
Conditional Programming
Exercises
Conditional Programming Exercises (1)

• Write a program which consists of a single class called OldEnough that asks the user for their age and displays "You can enter" if he/she is over 18, but displays "Sorry, you are not allowed to enter" otherwise.

• Write a program which consists of a single class called BuyStuff that asks the user for two amounts, adds them, calculates tax at 15%, shows the result to the user, and asks for money. It then compares if the person gave enough money. If so, it displays the amount of change to return otherwise it displays a message asking for more money.
Conditional Programming Exercises (2)

• Write a program which consists of a single class called `Calculator`. This program will display a menu with the following options: (1) add, (2) subtract, (3) multiply, (4) divide, (5) mod, and (6) do nothing. If the user enters something else, the program should display an error message. Otherwise, it should ask the user for two numbers, perform the calculation, and display the result.

• Write a program which consists of a single class called `SortThree`. This program should ask the user to enter 3 integers. The program will then display these in increasing order.