Java Chapter 3 Part 1

* Decision Structures and Boolean Expressions
* CIS 255 • Shelby-Hoover Campus

Decision Structures Defined

* The syntax from chapter 2 is sufficient for creating a **sequence structure**
* Additional syntax is required to create alternate flows of execution
* Chapter 3 focuses on **decision structures**
  + Also known as **branching** or **selection**
  + Evaluates a condition to determine whether to execute a certain block of statements or (optionally) an alternative
  + The statements that are performed based on the condition are said to be **conditionally executed**

Boolean Expressions

* Expressions that have a result of true or false are called **Boolean expressions**
* Formed with **relational operators**
  + Greater than: >
  + Less than: <
  + Greater than or equal to: >=
  + Less than or equal to: <=
  + Equal to: ==
  + Not equal to: !=

Using Relational Operators

* The assignment operator = and the equality comparison operator == are different (you will get syntax errors if you confuse them)
* Don’t put a space between the two characters of a two-character relational operator
* Each side of an operator should be a literal, variable, or expression to be compared to the other side
* The result of a Boolean expression can be stored in a boolean variable (sometimes called a flag):  
    
  boolean seniorCitizen = (age >= 65);
* Characters are compared using their underlying Unicode values (uppercase and lowercase letters are in separate groups)

Single-Alternative Decisions

* To execute a particular set of statements only when a Boolean expression is true, use the if statement:  
    
  if (Boolean\_expression)  
  {  
   statements\_to\_execute\_if\_true;  
  }
* There is no semicolon at the end of the header of an if statement, but the statements to be executed should end with semicolons
* The brackets are optional if there is only one statement to execute for a true result; however, if you omit the brackets around a set of statements, Java will only see the first statement as dependent upon the Boolean expression, with the other statements always executing regardless of the outcome of the Boolean expression

Single-Alternative Example

* Providing incentives for meeting a sales goal:  
    
  double sales, bonus = 0, commissionRate = 0;  
  int daysOff = 0;  
  Scanner keyboard = new Scanner(System.in);  
    
  System.out.print("Enter sales amount: $");  
  sales = keyboard.nextDouble();  
    
  if (sales > 50000)  
  {  
   bonus = 500.0;  
   commissionRate = 0.12;  
   daysOff += 1;  
  }
* The placement of the brackets (on separate lines vs. with the header / last line) is up to the programmer

Dual-Alternative Decisions

* If a program should execute an alternative set of statements when the Boolean expression is false, use an if-else statement:  
    
  if (Boolean\_expression)  
  {  
   statements\_to\_execute\_if\_true;  
  }  
  else  
  {  
   statements\_to\_execute\_if\_false;  
  }
* Only one of the two blocks will be executed

Dual-Alternative Example

* Deducting medical expenses if they meet the minimum:  
    
  double gross, expenses, deduction;  
  Scanner kb = new Scanner(System.in);  
    
  System.out.print("Enter your AGI: ");  
  gross = kb.nextDouble();  
  System.out.print("Enter medical expenses: ");  
  expenses = kb.nextDouble();  
    
  if (expenses >= gross \* 0.075) {  
   deduction = expenses - gross \* 0.075;  
   System.out.println("Deduction amount: $" + deduction);  
  }  
  else {  
   deduction = 0;  
   System.out.println("Expenses too low for deduction.");  
  }
* Avoiding division by zero: Division.java (Code Listing 3-2)

Multiple Alternatives with else-if

* To ask a series of questions, stopping to execute a set of statements when reaching the first true result, use an if-else-if statement:  
    
  if (Boolean\_expression\_1)  
  {  
   statements\_executed\_if\_1\_is\_true;  
  }  
  else if (Boolean\_expression\_2)  
  {  
   statements\_executed\_if\_1\_false\_2\_true;  
  }  
  else if (Boolean\_expression\_3)  
  {  
   statements\_if\_1\_and\_2\_false\_3\_true;  
  }  
  else  
  {  
   statements\_if\_1\_2\_and\_3\_are\_false;  
  }
* A plain else statement added to the end can be used to execute statements when none of the Boolean expressions evaluate to true; this is known as a **trailing else**

else-if Structure Example

* Various discount rates dependent upon the quantity ordered:  
    
  double discount;  
  int quantity;  
  Scanner keyboard = new Scanner(System.in);  
    
  System.out.print("Enter quantity desired: ");  
  quantity = keyboard.nextInt();  
    
  if (quantity >= 50)  
   discount = 0.25;  
  else if (quantity >= 25)  
   discount = 0.20;  
  else if (quantity >= 10)  
   discount = 0.10;  
  else  
   discount = 0;
* Converting scores to letter grades: TestResults.java (Code Listing 3-5) and TrailingElse.java (Code Listing 3-6)

Decision Structure Details

* When using an if-else-if statement, only one set of statements will be executed
  + Based on the first true Boolean expression reached
  + Boolean expressions that are most likely to require evaluation should be placed first
  + The trailing else is optional
  + Remember that a plain else never has a Boolean expression; you must use else if when you want to evaluate another expression
* One decision structure may be nested inside of another (LoanQualifier.java, Code Listing 3-3)
  + Brackets are important to make sure that structures are grouped properly
  + A “dangling else” will be matched with the most recent if (the nearest neighbor) regardless of indentation

Logical Operators

* A **logical operator** creates a complex Boolean expression from one or more expressions
* && (and): connects two Boolean expressions; requires both expressions to be true
* || (or): connects two Boolean expressions; requires at least one expression to be true
* ! (not): applied to a single Boolean expression in parentheses; reverses the result
* Each Boolean expression must be complete

Logical Operator Example

* Math placement based on exam scores (assume act and sat are int variables already containing values and placement is a String variable):  
    
  if ((act >= 27) || (sat >= 610))  
   placement = "MTH 125S";  
  else if ((act == 26) || (sat >= 590))  
   placement = "MTH 113";  
  else if ((act >= 24) || (sat >= 550))  
   placement = "MTH 112";  
  else if ((act >= 20) || (sat >= 480))  
   placement = "MTH 100";  
  else if ((act >= 16) || (sat >= 370))  
   placement = "MTH 098";  
  else  
   placement = "MTH 090";
* Loan qualification with &&: LogicalAnd.java (Code Listing 3-7)

Logical Operator Details

* With && and ||, parentheses around each simple expression are optional, but there must be a set around the entire expression
* Java uses **short-circuit evaluation** with && and ||
  + &&: if the left-side expression is false, the result must be false, so Java skips the right-side expression
  + ||: if the left-side expression is true, the result must be true, so Java skips the right-side expression
* When using a mixture of logical operators in a single expression, precedence is involved
  + ! first, then &&, then ||
  + Complete precedence ordering: Table 3-10

Logical Operators and Ranges

* Be careful when testing a value to see if it’s in a particular range:
  + To check a range between endpoints, use &&
  + To check a range outside of endpoints, use ||
  + To include the endpoints, use >= and <=
  + To exclude the endpoints, use > and <
* Example: normal BMI (body mass index):
  + BAD: if (18.5 <= bmi < 25)
  + STILL BAD: if (bmi >= 18.5 && < 25)
  + GOOD: if ((bmi >= 18.5) && (bmi < 25))
  + GOOD: if ((18.5 <= bmi) && (bmi < 25))

Comparing Objects

* The relational operators shown so far work with primitive variables
* String values are objects
  + An object variable stores the address of the object
  + The relational operators compare the addresses, not the contents, of the objects
* You must use methods to perform comparisons between the contents of objects

String Comparison Methods

* Equality: string1.equals(string2) returns true if the contents of string1 and string2 are equal
* Comparisons: string1.compareTo(string)
  + Returns a negative int if string1 is “less than” string2
  + Returns 0 if string1 and string2 contain the same characters
  + Returns a positive int if string1 is “greater than” string2
* Both methods perform a **lexicographical comparison**, comparing the values of each character at the same position (case sensitive) rather than using alphabetical order
* There are case-insensitive versions of each method
  + string1.equalsIgnoreCase(string2)
  + string1.compareToIgnoreCase(string2)

String Comparison Example

* Determining an advisor from a student’s major:  
    
  String major, advisor;  
    
  major = JOptionPane.showInputDialog("Enter your major:");  
    
  if (major.equalsIgnoreCase("Programming"))  
   advisor = "Tommy Battles";  
  else if (major.equalsIgnoreCase("Networking"))  
   advisor = "Linda Dobyns";  
  else if (major.equalsIgnoreCase("Web Technologies"))  
   advisor = "Hal Harris, Jr.";  
  else  
   advisor = "Unknown";  
    
  JOptionPane.showMessageDialog(null, "Your advisor is " + advisor);
* Letting the user guess a secret word: SecretWord.java (Code Listing 3-11)