Chapter 4:  
Conditionals and Recursion  

Think Java:  
How to Think Like a Computer Scientist  

5.1.2  

by Allen B. Downey
Agenda

- The modulus operator
- Random Number Generation
- Conditional Execution
- Alternative Execution
- Chained Conditionals
- Nested Conditionals
- Scanner Class for Input from Keyboard
- Methods with conditionals:
  - the return statement
- Recursion
Calculate leftover inches

• In assignment 3 we wanted to convert:
  – inches $\rightarrow$ feet and leftover inches.
  – For example, 40 inches = 3 feet, 4 inches

• We used these formulae:
  – feet = inches/12 (using integer division)
  – leftover_inches = inches – feet*12

• There's a better way to calculate leftover_inches:
  – Modulus operator (%) left
  – leftover_inches = inches % 12 ;
Modulus Operator

- The modulus operator ONLY works on integers
  - yields the remainder of first divided by second.
  - In Java, the modulus operator is %

```
int quotient = 7 / 3;  // 2
int remainder = 7 % 3;  // 1
```

- Thus, 7 divided by 3 is 2 with 1 left over.
- Use to check if one number divisible by another
  - if x % y is zero, then x is divisible by y.
  - Solve these on your lab4 handout:

```
25 % 8 =    37 % 10 =    4 % 7 =
125 % 25 =  8 % 4 =  7 % 4 =
```
Modulus Operator

• The modulus operator ONLY works on integers
  – yields the remainder of first divided by second.
  – In Java, the modulus operator is %

```java
int quotient = 7 / 3;   // 2
int remainder = 7 % 3;  // 1
```

• Thus, 7 divided by 3 is 2 with 1 left over.

• Use to check if one number divisible by another
  – if x % y is zero, then x is divisible by y.
  – For example:

```
25 % 8 = 1   37 % 10 = 7   4 % 7 = 4
125 % 25 = 0  8 % 4 = 0   7 % 4 = 3
```
Agenda

• The modulus operator
• Random Number Generation
• Conditional Execution
• Alternative Execution
• Chained Conditionals
• Nested Conditionals
• Scanner Class for Input from Keyboard
• Methods with conditionals
• Recursion
Random Number Generation

• Java's Math library has a method `random` generates a random number between 0 and 1
  \[ 0 \leq x < 1 \]

• Here's how to use it:
  ```java
  double x = Math.random();
  System.out.println( x )
  ```

• Each time you run the program, different x:
  0.4523   0.0001   0.8214   0.5149

• Useful for computer games, coin toss, etc.
Modeling a Coin Toss

• Pick a random number, myNumber
  – If myNumber is less than 0.5, print "heads"
  – If myNumber is larger than 0.5, print "tails"

• We need a new programming structure that lets us print a different message depending on the value of myNumber!
Agenda

• The modulus operator
• Random Number Generation
• Conditional Execution
• Alternative Execution
• Chained Conditionals
• Nested Conditionals
• Scanner Class for Input from Keyboard
• Methods with conditionals
• Recursion
Which way to go?

- Conditional statements allow execution of certain statements only if a particular condition is satisfied.

- Consider these statements as some kind of a gate keeper that allows entry only if a condition is satisfied.
Conditional Execution

• The `if` statement decides whether a section of code executes or not.
  – changes the behavior of the program accordingly.

  \[
  \text{if (expression is true)} \smallbreak
  \text{execute statement block}
  \]

• if the expression is false, nothing happens
Flowcharts

• If statements can be modeled as a flow chart.

```java
if (temp < 32){
    wearCoat();
}
```

```
Is it cold outside?
```

```
Wear a coat.
```
Relational Operators

In most cases, the condition of the if statement is written using *relational operators*.

<table>
<thead>
<tr>
<th>Relational Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>is greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>is less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>is greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>is less than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>is equal to</td>
</tr>
<tr>
<td>!=</td>
<td>is not equal to</td>
</tr>
</tbody>
</table>
Relational Operators

- A condition is a relational expression formed from one of the 6 relational operators.
  - results in a true or false value.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>x &gt; y</td>
<td>Is x greater than y?</td>
</tr>
<tr>
<td>x &lt; y</td>
<td>Is x less than y?</td>
</tr>
<tr>
<td>x &gt;= y</td>
<td>Is x greater than or equal to y?</td>
</tr>
<tr>
<td>x &lt;= y</td>
<td>Is x less than or equal to y.</td>
</tr>
<tr>
<td>x == y</td>
<td>Is x equal to y?</td>
</tr>
<tr>
<td>x != y</td>
<td>Is x not equal to y?</td>
</tr>
</tbody>
</table>
Conditional examples:

- This if statement assigns 0 to x when y is equal to 20
  
  ```java
  if (y == 20) {
    x = 0;
  }
  ```

- If hours is larger than 40, then pay gets the value payRate times 1.5 times hours
  
  ```java
  if (hours > 40) {
    pay = payRate*1.5*hours
  }
  ```
Example: Modeling a Coin Toss, Version 1

double myNumber = Math.random();
//Decide the outcome of a coin flip
if (myNumber < .5) {
    System.out.println("heads");
}
if (myNumber >= .5) {
    System.out.println("tails");
}

- Every time we run the program, we get a different myNumber (between 0 and 1)
  - The first if prints "heads" if myNumber < 0.5
  - The second if prints "tails" if myNumber >= 0.5
Caution with Conditionals

• Syntax differs from math symbols: =, ≠ and ≤.
  – common error: single = instead of a double ==
    = is the assignment operator
    == is a comparison operator.
  – no such thing as =< or =>.

• Two sides of a condition must be same type.
  – Only compare ints to ints and doubles to doubles.
  – == and != don't work with Strings like you expect.
  – Comparing strings is in chapter 8
Agenda

- The modulus operator
- Random Number Generation
- Conditional Execution
- Alternative Execution
- Chained Conditionals
- Nested Conditionals
- Scanner Class for Input from Keyboard
- Methods with conditionals
- Recursion
Alternative Execution

• The \texttt{if-else} statement adds the ability to conditionally execute code when the \texttt{if} condition is false.

\begin{verbatim}
if (expression)
  statementBlockIfTrue;
else
  statementBlockIfFalse;
\end{verbatim}
if-else Statement Flowcharts

```java
if (temp < 32){
    wearCoat();
} else {
    wearShorts();
}
```
Example: Modeling a Coin Toss, Version 2

double x = Math.random();
if (x < 0.5) {
    System.out.println("heads");
} else {
    System.out.println("tails");
}

- Every time we run the program, we get a different x (between 0 and 1)
- The if statement prints "heads" if x < 0.5
- Note: else handles the case that x is NOT < 0.5
  - do not say  
    
    else (x >= 0.5) {   ERROR
Agenda

- The modulus operator
- Random Number Generation
- Conditional Execution
- Alternative Execution
- Chained Conditionals
- Nested Conditionals
- Scanner Class for Input from Keyboard
- Methods with conditionals
- Recursion
Chained Conditionals

if (expression_1)
{
    statement;
    statement;
    etc.  
}

else if (expression_2)
{
    statement;
    statement;
    etc.  
}

Insert as many else if clauses as necessary

else
{
    statement;
    statement;
    etc.  
}

If expression_1 is true these statements are executed, and the rest of the structure is ignored.

Otherwise, if expression_2 is true these statements are executed, and the rest of the structure is ignored.

These statements are executed if none of the expressions above are true.
Chained Conditionals (if-else-if)

- Chained conditionals using the `if-else-if` statement makes it possible to handle more than two possible outcomes – handle as many as you like
  - Continue to add `else-if` blocks until you run out of possibilities
  - Last `else` means "none of the above" were true

- Care must be used since `else` statements match up with the immediately preceding unmatched `if` statement.
Chained Conditional (*if-else-if*) Flowchart
Chained Conditional (if-else-if) Example

```java
if (x > 0) {
    System.out.println("x is positive");
} else if (x < 0) {
    System.out.println("x is negative");
} else {
    System.out.println("x is zero");
}
```

- keep all the statements and squiggly-brackets lined up
- less likely to make syntax errors
- more likely to find them if you do
What if our Coin could fall off the table?

double x = Math.random();
if (x < 0.45) {
    System.out.println("heads");
} else if (x < 0.9) {
    System.out.println("tails");
} else {
    System.out.println("fell off table!");

• Prints "heads": 0.0 <= x < 0.45
• Prints "tails": 0.45 <= x < 0.9
• Prints "fell off table": 0.90 <= x < 1.0
Agenda

- The modulus operator
- Random Number Generation
- Conditional Execution
- Alternative Execution
- Chained Conditionals
- Nested Conditionals
- Scanner Class for Input from Keyboard
- Methods with conditionals
- Recursion
Nested Conditionals

- If an if statement appears inside another if statement block it is called a nested conditional.
- The nested if is executed only if the outer if statement results in a true condition.
- Nesting allows for unlimited variation of structure in selecting possible outcomes.
- Can also be harder to read.
Nested if Statement Flowcharts

- Is it cold outside?
  - Yes: Wear a parka.
  - No: Is it snowing?
    - Yes: Wear a parka.
    - No: Wear a jacket.
- Wear shorts.
Nested Conditional Example

```java
if (coldOutside) {
    if (snowing) {
        wearParka();
    }
    else {
        wearJacket();
    }
} else {
    wearShorts();
}
```
Style helps clarify logic

- proper indentation makes it much easier to match up else statements with their corresponding if statement.

- Bad example:

```java
if (coldOutside) {
    if (snowing) {
        wearParka();
    } else {
        wearJacket();
    }
} else {
    wearShorts();
}
```
Alignment and Nested if Statements

This if and else go together.

```c
if (coldOutside)
{
    if (snowing)
    {
        wearParka();
    }
    else
    {
        wearJacket();
    }
}
else
{
    wearShorts();
}
```
Do Lab 4 Part A

- finish PennyChanger
- debug AgeInsult
  - we'll cover Scanner in next lecture
  - used to input data from keyboard
- write CoinFlip program

- If you finish early, start doing drills in Assignment 4
  - Please do not do Part B until after next lecture
Agenda

- The modulus operator
- Random Number Generation
- Conditional Execution
- Alternative Execution
- Chained Conditionals
- Nested Conditionals
- Scanner Class for Input from Keyboard
- Methods with conditionals
- Recursion
The **Scanner Class**

- To read input from the keyboard we can use the **Scanner class**.
- The **Scanner class** is defined in `java.util`, so we will use the following statement at the top of our programs:

```java
import java.util.Scanner;
```
The Scanner Class

- Scanner objects work with `System.in`

- To create a `Scanner` object:
  ```java
  Scanner keyboard = new Scanner (System.in);
  ```

- The following `Scanner` class methods are useful
  - `next` read the next word (til whitespace) as `String`
  - `nextLine` read to the end of line as a `String`
  - `nextInt` read the next number as an `int`
  - `nextDouble` read the next number as a `double`

- See example on next slide
import java.util.Scanner;  // Needed for the Scanner class

/**
 * This program demonstrates the Scanner class.
 */

public class Payroll
{
    public static void main(String[] args)
    {
        String name;         // To hold a name
        int hours;           // Hours worked
        double payRate;      // Hourly pay rate
        double grossPay;     // Gross pay

        // Create a Scanner object to read input.
        Scanner keyboard = new Scanner(System.in);

        // Get the user's name.
        System.out.print("What is your name? ");
        name = keyboard.nextLine();

        // reads one word of text
        name = keyboard.next();
    }
}
// Get the number of hours worked this week.
System.out.print("Number of hours worked this week? ");
hours = keyboard.nextInt();

// Get the user's hourly pay rate.
System.out.print("What is your hourly pay rate? ");
payRate = keyboard.nextDouble();

// Calculate the gross pay.
grossPay = hours * payRate;

// Display the resulting information.
System.out.println("Hello " + name);
System.out.println("Your gross pay is $" + grossPay);
}
Adding Conditionals to other methods

- This code checks whether x is divisible by 2

```java
if (x%2 == 0) {
    System.out.println("x is even");
} else {
    System.out.println("x is odd");
}
```
If you want to use it a lot, write a method

• This method checks whether x is divisible by 2

```java
public static void printParity(int x){
    if (x%2 == 0) {
        System.out.println("x is even");
    } else {
        System.out.println("x is odd");
    }
}
```

• This code invokes (calls) the method above

```java
printParity(15);
printParity(24);
```
Reminder: do not declare variables in method calls

```java
printParity(7);
int number = 35;
printParity(number);
printParity(int number);  // WRONG
```
Return Statement

- lets you terminate the execution of a method early
  - such as if you detect an error condition:

```java
public static void printSquareRoot(double x) {
    if (x < 0.0) {
        System.out.println("Positive numbers only, please.");
        return;
    }

    double result = Math.sqrt(x);
    System.out.println("The square root of x is " + result);
}
```

- checks whether x is less than zero,
- if so it prints an error message and then exits method.
  - flow of execution returns to the caller
  - remaining lines are not executed.
Automatic Type Conversion

• How does this work?

"The square root of x is " + result
  – one of the operands is a String, the other is a double.
  – Java automatically converts the double to a String
  – then joins them using string concatenation.

• What do you think would happen when adding an integer and a floating-point value?
  3 + 4.5 = ???
Recursion

- it is legal for one method to invoke another
  - it is also legal for a method to invoke itself
  - one of the most magical and interesting things program can do.

```java
public static void countdown(int n) {
    if (n == 0) {
        System.out.println("Blastoff!");
    } else {
        System.out.println(n);
        countdown(n-1);
    }
}
```

- countdown takes a single integer as a parameter.
  - If the parameter is zero, prints “Blastoff.”
  - Otherwise, prints the number, & invokes countdown again, passing n-1 as an argument
Running a recursive countdown

• if we invoke this method, in main:

    countdown(3);

    it prints 3, then invokes itself on 2...
    it prints 2, then invokes itself on 1...
    it prints 1, then invokes itself on 0...
    it prints Blastoff!, then returns
    The countdown that got n=1 returns.
    The countdown that got n=2 returns.
    The countdown that got n=3 returns.
    And then you’re back in main.

    Displays:

3
2
1
Blastoff!
Another Recursive example

- recall newLine and threeLines from chapter 3
  - not very helpful if we want to print 2 lines or 106
  - a better alternative would be

```java
public static void nLines(int n) {
    if (n > 0) {
        System.out.println(""-number of spaces");
        nLines(n-1);
    }
}
```

- if n is greater than zero, prints 1 newline
  - then invokes itself to print n-1 newlines.
  - The total number of newlines that get printed is 1 +(n-1), which usually comes out to roughly n!
- Recursion: when a method invokes itself
Stack Diagram for Recursive Methods

- stack diagrams can help interpret a recursive method.
- every time a method is called, get a new frame
- frame contains new method parameters and vars
- stack diagram for countdown, called with n = 3

```java
public static void countdown(int n) {
    if (n == 0) {
        System.out.println("Blastoff!");
    } else {
        System.out.println(n);
        countdown(n - 1);
    }
}
```
Do Lab 4 Part B