**CS 112 Introduction to Programming**

**Conditional Statements**

**Boolean Expressions and Methods**

Yang (Richard) Yang  
Computer Science Department  
Yale University  
308A Watson, Phone: 432-6400  
Email: yry@cs.yale.edu

---

**Recap: Scanner**

- **Some Scanner methods:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nextInt()</code></td>
<td>Returns an int from source</td>
</tr>
<tr>
<td><code>nextDouble()</code></td>
<td>Returns a double from source</td>
</tr>
<tr>
<td><code>next()</code></td>
<td>Returns a one-word string from source</td>
</tr>
<tr>
<td><code>nextLine()</code></td>
<td>Returns a one-line string from source</td>
</tr>
</tbody>
</table>

- Java uses objects to remember the source of a scanner

```java
Scanner console = new Scanner(System.in);  
console.nextInt();
```

---

**Recap: Scanning Details**

- `nextInt()`, `nextDouble()` and `next()` are token based scanning methods
- `nextLine()` collects any input character into a string until the first new line and discards the new line
- When `nextInt()` or `nextDouble()` parses a token, but cannot parse it correctly, the method throws an exception
- A robust program will check condition before invoking `nextInt()` or `nextDouble()`

---

**Java Conditional Statements**

- Conditional statements, also called decision statements, decide whether or not to execute a particular sequence of statements

---

**The if Statement**

```java
if (boolean expression) {  
    statement T;  
}  
else {  
    statement F;
}
```

- Example:

  ```java
  int nA = 0;  
  for (int i = 0; i < 100; i++) {  
      double grade = console.nextDouble();  
      if (grade >= 90) {  
          System.out.println("Welcome to Yale!");  
          nA ++; // This is called accumulative sum
      }
  }
  ```
Recall: Basic Boolean Expression

- A basic Boolean expression is to compare two values using a relational operator:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equals</td>
<td>1 + 1 == 2</td>
<td>true</td>
</tr>
<tr>
<td>!=</td>
<td>does not equal</td>
<td>2.0 != 2.5</td>
<td>true</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>10 &lt; 5</td>
<td>false</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>10 &gt; 5</td>
<td>true</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>12 &lt;= 100</td>
<td>false</td>
</tr>
<tr>
<td>=&gt;</td>
<td>greater than or equal to</td>
<td>5.0 =&gt; 5.0</td>
<td>true</td>
</tr>
</tbody>
</table>

- Note the difference between the equality operator (==) and the assignment operator (=).

Outline: Conditional Program Flow Control

- Our "journey" of introducing conditionals:
  
  - simple if (a <comp> b)
  - nested if/else

Example: Basic Boolean Expression

- Example: Basic Boolean Expression

```
public class Flip {
    public static void main(String[] args) {
        System.out.println("Heads");
    }
}
```

Motivation: Chaos Game

- Motivation: Chaos Game

```
public class Chaos {
    public static void main(String[] args) {
        for (int i = 0; i < 1000; i++) {
            int rand = getRand(0, 2);
            if (rand == 0) {
                // …
            } else if (rand == 1) {
                // …
            } else if (rand == 2) {
                // …
            }
        }
    }
}
```

```
// java Flip
// java Flip Heads
// java Flip Tails
```

Q: How many comparisons in each round (iteration)?
Solution II: Nested Comparison

```java
public static void main (String[] args) {
    for (int i = 0; i < 1000; i++) {
        int rand = getRand(0, 2);
        if (rand == 0) {
            // ...
        } else if (rand == 1) {
            // ...
        } else {
            // ...
        }
    }
}
```

Q: Average # of comparisons per iteration?
Benefit of nested comparison: reduce # comparisons

Summary: Why nested if/else

- Mutually exclusive test conditions
  - Reduces # of comparisons
- Non-mutually exclusive test conditions
  - Achieves mutual exclusion

Grading Curve

```
Grading Curve

Q: How is this code for grading based on percentiles?
Scanner console = new Scanner(System.in);
System.out.print("What percentile? ");
int percent = console.nextInt();
if (percent >= 90) {
    System.out.println("You got an A!");
} else if (percent >= 80) {
    System.out.println("You got a B!");
} else if (percent >= 70) {
    System.out.println("You got a C!");
} else if (percent >= 60) {
    System.out.println("You got a D!");
} else {
    System.out.println("You got an F!");
}
```

Exercise: Barnsley Game

```
public static void main(String[] args) {
    StdDraw.setPenRadius(0.002);
    StdDraw.setPenColor(Color.GREEN);
    int T = 40000;
    double x = 0.0, y = 0.0;
    for (int t = 0; t < T; t++) {
        double rand = Math.random();           // 0 - 1.0
        if (rand <= 0.7) {
            x = .78 * x + .03 * y + .11;
            y = -.03 * x + .74 * y + .27;
        } else if (rand <= 0.7 + 0.15) {
            x = -.14 * x + .26 * y + .57;
            y = .25 * x + .22 * y - .04;
        } else if (rand < 0.7 + 0.15 + 0.13) {
            x = .17 * x - .21 * y + .41;
            y = .22 * x + .18 * y + .09;
        } else {
            x = 0.5;
            y = .27 * y;
        }
        StdDraw.point(x, y);
    }
}
```

Exercise: Barnsley Game

Questions to think about:
- What does computation tell us about nature?
- What does nature tell us about computation?
Outline: Conditional Program Flow Control

- Our “journey” of introducing conditionals
  - simple if (a < b)
  - nested if/else

Complexity of nested if/else: all path must return; mismatched else

Matching Nested if Statements

- Nested if statements may have a matching problem

if (temperature < 50)
  System.out.println("Cool!");
else
  System.out.println("Hot!");

Nested if w/ Ambiguity

Give a value of temperature to produce different results

if (temperature < 50)
  System.out.println("Cool!");
else if (temperature < 100)
  System.out.println("Cool!");
else
  System.out.println("Hot!");

Nested if Statements

- Rule: an else clause is matched to the last unmatched if (no matter what the indentation implies)
  - If you find such statements confusing, avoid writing such structures and always use block statements to make the structure clear.

if/else with return

- A method with a return requires that all paths through the code must reach a return statement.
- The compiler analyzes this by considering the syntax only!
### Outline: Conditional Program Flow Control

- **Our “journey” of introducing conditionals**
  - Simple if (a <comp> b)
  - Nested if/else:
    - Complexity of nested if/else: all path must return; mismatched else
    - Combining multiple <comp>:

### Logical Operators

- Tests can be combined using **logical operators**:
  - Operator: Description
    - `&&` or `and`: True if both conditions are true.
    - `||` or `or`: True if at least one condition is true.
    - `!` or `not`: True if condition is false.

### Motivation: Testing Containment

- **We may need to check multiple conditions**

### Extension: Testing Containment

- **We may need to check multiple conditions**

Exercise

Write a method to compute the number of days in a month

public static int daysInMonth(int year, int month)

Applying Boolean Exp. in daysInMonth

Use a logical OR to combine the cases

if (m == 1 || m == 3 || m == 5 ||
   m == 7 || m == 8 || m == 10 || m == 12)
   numDays = 31;

Implement leap year condition:

"... most years that are evenly divisible by 4 are leap years; However, there are some exceptions to this rule: Years that are evenly divisible by 100 are not leap years, unless they are also evenly divisible by 400"

leap year

or

y % 400 == 0 || (y % 100 != 0 && y % 4 == 0)

Testing Leap Year

Outline: Conditional Program Flow Control

Our “journey” of introducing conditionals

simple if (a <comp> b)

nested if/else

Complexity of nested if/else:
all path must return; mismatched else

<condition> combining multiple <comp>:

Boolean variables(expressions/methods)

Reuse Testing for Leap Year

How do we define a method to reuse the ability to test if a year is a leap year?

isLeapYear(int year);
Recall: boolean Type

- boolean: A primitive type whose values are true and false.
  - Like other types, it is legal to:
    - create a boolean variable and assign it values
    - pass a boolean value as a parameter
    - return boolean value as a method
    - call a method that returns a boolean and use it as a test

boolean Expressions

- Similar to arithmetic expressions, except that:
  - the operands are boolean values, boolean variables, or a test using a relational operator (e.g., <, >).
  - the operators are ||, &&, !

- Example
  boolean lovesCS = true;
  boolean student = age < 21;
  // allow only CS-loving students over 21
  if (student && lovesCS)
    System.out.println("Pass");
  // an alternative
  if (age < 21 && lovesCS)
    System.out.println("Pass");

boolean Expressions: Example

```java
boolean goodAge = age >= 18 && age < 29;
boolean goodHeight = height >= 78 && height < 84;
boolean rich = salary >= 100000.0;
if ((goodAge && goodHeight) || rich) {
    System.out.println("Okay, let's go out!");
} else {
    System.out.println("It's not you, it's me...");
}
```

Mixed Arithmetic, Relation, Logical, and Assignment Expression

- Example:
  boolean mystery = 5 * 7 >= 3 + 5 * (7 – 1) && 7 <= 11;

- Precedence ordering of boolean expression
  - Arithmetic operators
  - Relations operators (==, !=, <, >, <=, >=)
    - Note that equality and relational operators cannot be chained (e.g., 1 < x < 3 is invalid)
  - NOT (!)
  - AND (&&)
  - OR (||)
  - Assignment operators (=)

Example

```java
boolean mystery = 5 * 7 >= 3 + 5 * (7 – 1) && 7 <= 11;
5 * 7 >= 3 + 5 * 6 && 7 <= 11
35 >= 33 && 7 <= 11
true && true
true
```

English vs boolean

- OR vs Exclusive OR
  - I'll either watch TV or go to gym: Exclusive OR
  - watchTV || gotoGym can be both true

- x is between 1 and 10
  - 1 <= x <= 10
  - 1 <= x && x <= 10

- x is either 1 or 2 or 3
  - x == 1 || x == 2 || x == 3
  - x == 1 || x == 2 || x == 3
Logical AND/OR Evaluation

- Java uses shortcircuit when evaluating `&&` and `||`
  - `a && b` shortcircuit:
    - if `a` is false, `b` is not evaluated
    ```java
    if ((count != 0) && (total / count > AVG_THRESHOLD)) {
    // …
    }
    ``
  - `a || b` shortcircuit:
    - if `a` is true, `b` is not evaluated