CS 112 Introduction to Programming

Boolean Expressions;
Nested if/else;

Yang (Richard) Yang
Computer Science Department
Yale University
208A Watson, Phone: 432-6400
Email: yry@cs.yale.edu
Admin.

- PS4 questions
- Exam 1 date
Recap: Input using Scanner

- Java uses an object to remember the state (e.g., source) of a scanner

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

- Input can be read as tokens or a line (nextLine)

- Token based input may have a mismatch problem

- Input processing may generate exceptions

- Java introduces try/catch to handle exceptions
Recap: Printf/Format Design

```
System.out.printf("format string", parameters);

"format string" can be easier than string concatenation to see the structure of output text

System.out.println("x: [" + xmin + " " + xmax + "] y: [" + ymin + " " + ymax + "]");

System.out.printf("x: [%f %f] y: [%f %f]", xmin, xmax, ymin, ymax);

System.out.printf("x: [%,.2f %,.2f] y: [%,.2f %,.2f]", xmin, xmax, ymin, ymax);
```
Recap: Program Flow of Control

- Java provides conditional and loop statements as program flow of control statements:
  - decision statements, or **conditional statements**: decide whether or not to execute a particular statement
  - repetition statements, or **loop statements**: perform a statement over and over repetitively

- The foundation of program flow of control is the logical condition, which should be a boolean expression, e.g.,

  ```java
  if ( <boolean expression> ) {
      do something
  } [else {
      so something else
  }]
  ```
A basic Boolean expression is a relational test of 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>3.2 != 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>126 &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 (=).
More Complex Relational Test: Testing Containment

Test if point \((x, y)\) is in the rectangle?

\[(0 \leq x \leq W) \land (0 \leq y \leq H)\]  

SYNTAX ERROR

\[(0 \leq x) \land (x \leq W) \land (0 \leq y) \land (y \leq H)\]
**Logical Operators**

- Tests can be combined using *logical operators*:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
<td>(2 == 3) &amp;&amp; (-1 &lt; 5)</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>or</td>
</tr>
<tr>
<td>!</td>
<td>not</td>
<td>!(2 == 3)</td>
<td>true</td>
</tr>
</tbody>
</table>

- "Truth tables" for each, used with logical values $p$ and $q$:

| $p$ | $q$ | $p$ && $q$ | $p$ || $q$ |
|-----|-----|------------|---------|
| true| true| true       | true    |
| true| false| false     | true    |
| false| true| false     | true    |
| false| false| false     | false   |

<table>
<thead>
<tr>
<th>$p$</th>
<th>!$p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
Exercise: Testing Containment

Test if dark rectangle is in the bigger rectangle?
Exercise

- Write a method to return the number of days in a month

```java
public static int daysInMonth( int year, int month )
```
Applying Logical Operators

- 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

\[
\text{year} \% 400 == 0 \quad \text{or} \quad \text{year} \% 4 == 0 \\
\text{year} \% 100 != 0
\]
Testing Leap Year

leap year

or

year % 400 == 0

and

year % 4 == 0  year % 100 != 0

\[ y \% 400 == 0 \ || \ (y \% 100 \ != 0 \ && \ y \% 4 == 0) \]
Applying Logical Operators

- Use a logical OR to combine the cases

```java
if (m == 1 || m == 3 || m == 5 || m == 7 || m == 8 || m == 10 || m == 12)
    numDays = 31;
```
(Offline) Exercise: Basic Boolean Expressions

- gentle || active
- !active && healthy cuddles
- !(clean && healthy) vamp, yoda
- !clean || !healthy vamp, yoda
- !(clean || gentle) vamp
- !clean &&!gentle vamp
Reuse Testing for Leap Year

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

```java
??? isLeapYear(int year);
```
Outline

- Admin and recap
- Conditionals
  - Logical conditions (Boolean expressions)
    - Basic relational tests
    - Tests using logical operators
    - Boolean type
boolean Type

- boolean: A primitive type whose values are true or 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

- You can use Boolean variables in Boolean expressions
Similar to arithmetic expressions, except that

- the operands are Boolean values, Boolean variables, or relational tests.
- the operators are logical operators \(||\) \&\& !

Example

```java
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: Why**

- **Why is the type boolean useful?**
  - Can capture a complex logical test result and use it later
  - Can write a method that does a complex test and returns it

```java
boolean goodAge = age >= 18 && age < 29;
boolean goodHeight = height >= 78 && height < 84;
boolean rich = salary >= 100000.0 || hasStock;
boolean perfect = school == YALE;

if ((goodAge && goodHeight) || rich || perfect) {
    System.out.println("Okay, let's go out!");
} else {
    System.out.println("It's not you, it's me...");
}
```
English vs Programming

- **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 || 2 || 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
Mixed Arithmetic, Relation, Logical, and Assignment Expression

- **Example:**

```java
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

boolean mystery = 5 * 7 >= 3 + 5 * (7 - 1) && 7 <= 11;
5 * 7 >= 3 + 5 * 6 && 7 <= 11
35 >= 3 + 30 && 7 <= 11
35 >= 33 && 7 <= 11
true && true
true
Define and Use boolean Methods

Define a boolean method

```java
public static boolean isOdd(int n) {
    if (n % 2 == 1) {
        return true;
    } else {
        return false;
    } // see a shorter version shortly
}
```

Calls to a boolean test method:

```java
if (isOdd(57)) {
    ...
}
```
Exercise: Implement and Use isLeapYear
Programmers new to boolean often test if a result is true:

```java
if ( isOdd(57) == true ) { // reads redundant
    ...
}
```

Simplify to:

```java
if ( isOdd(57) ) { // concise
    ...
}
```

A similar pattern can be used for a false test:

```java
if ( isOdd(57) == false ) { // does not read well
if ( !isOdd(57) ) { // concise
    ...
}
```
Programmers new to boolean often have an if/else to return true or false:

```java
public static boolean isOdd(int n) {
    if (n % 2 == 1) {
        return true;
    } else {
        return false;
    }
}
```

```java
public static boolean isOdd(int n) {
    return (n % 2 == 1);
}
```
"Boolean Zen" template

- Replace

  public static boolean <name>(<parameters>) {
    if (<test>) {
      return true;
    } else {
      return false;
    }
  }

- with

  public static boolean <name>(<parameters>) {
    return <test>;
  }
public static boolean isPrime(int n) {
    int factors = 0;
    for (int i = 1; i <= n; i++) {
        if (n % i == 0) {
            // cumulative sum
            factors++;
        }
    }
    if (factors == 2) {
        return true;
    } else {
        return false;
    }
}
Students new to boolean often have an if/else to return true or false:

```java
public static boolean bothOdd(int n1, int n2) {
    if (n1 % 2 == 1 && n2 % 2 == 1) {
        return true;
    } else {
        return false;
    }
}
```
Students new to boolean often have an if/else to return true or false:

```java
public static boolean bothOdd(int n1, int n2) {
    if (n1 % 2 == 1 && n2 % 2 == 1) {
        return true;
    } else {
        return false;
    }
}
```

```java
public static boolean bothOdd(int n1, int n2) {
    return (n1 % 2 == 1 && n2 % 2 == 1);
}
```
Outline

- Admin and recap
- Conditionals
  - Logical conditions (Boolean expressions)
    - Basic relational tests
    - Tests using logical operators
    - Boolean type
  - Nested if/else conditional statements
Motivation: Chaos Game

```java
public static void main (String[] args) {
    for (int i = 0; i < 10000; i++) {
        int rand = (int) (Math.random() * 3);
        if (rand == 0) {
            // ...
        }
        if (rand == 1) {
            // ...
        }
        if (rand == 2) {
            // ...
        }
    }
}
```

Q: How many comparisons in each round (iteration)?
public static void main (String[] args) {
    ...
    for (int i = 0; i < 1000; i++) {
        int rand = (int) (Math.random() * 3);
        if (rand == 0) {
            //
            }
        }
        else if (rand == 1) {
            // ...
        }
        else if (rand == 2) {
            // ...
        }
    }
}

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

Add else to skip tests if already matched. This is called nested comparison (else if)