Announcements

• Problem Set 1 (PS 1) was due 9/13, but will accept until 11:59pm, 9/20

• See the updated tutoring schedule and see if it would work for you

• Reading assignment for this lecture: Chapter 3 of Liang
The if statement

• The if statement has the following syntax:
  
  ```java
  if (<condition>) {
    statement1;
    statement2;
    ...
  }
  ```

  Statement block executed if `<condition>` is true

• `<condition>` must be a boolean expression – an expression that evaluates to either true or false

• If `<condition>` evaluates to true, then the statement block is executed; otherwise, it’s skipped
Example: if statement

```java
public static void main(String[] args) {
    int a = <some_integer_expression>;
    int b = <another_integer_expression>;

    if (a > b) {
        System.out.println(a + " is greater than " + b);
    }
}
```

- This example just prints one statement if `a` is bigger than `b`
- But, what if `a` is not greater than `b`? Nothing gets printed!
The **if**...**else** statement

- An else branch can be optionally “added on” to an if statement:

```java
if (<condition>) {
    // true branch
    statements;
    ...
} else {
    // false branch
    statements;
    ...
}
```

- Evaluate `<condition>` first.
- If the value is true, then do the true branch.
- If the value is false, then do the false branch.
- So, it will end up taking one or the other branch, but never both!
Relational expressions

- if statements use logical tests
  
  ```
  if (a <= 23) { . . . }
  if (a > b) { . . . }
  ```

- These are boolean expressions

- Tests use relational operators:

<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>3 – 2 == 1</td>
<td>true</td>
</tr>
<tr>
<td>!=</td>
<td>not equal</td>
<td>4.5 != 3.2</td>
<td>true</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>4 &lt; 5</td>
<td>true</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>4 &gt; 5</td>
<td>false</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>12 &lt;= 10</td>
<td>false</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than equal to</td>
<td>4 &gt;= 4</td>
<td>true</td>
</tr>
</tbody>
</table>
Example: *if...else* statement

```java
public static void main(String[] args) {
    int a = some_integer_expression;
    int b = another_integer_expression;
    if (a > b) {
        System.out.println(a + " is greater than " + b);
    }
    else {
        System.out.println(a + " is less than or equal to " + b);
    }
}
```

- Now we print out something no matter what a and b are!
Misuse of `if`

- What is wrong with the following code?

```java
Scanner input = new Scanner(System.in);
System.out.print("What was your score? ");
int score= input.nextInt();
if (score >= 90) {
    System.out.println("A");
}
if (score >= 80) {
    System.out.println("B");
}
if (score >= 70) {
    System.out.println("C");
}
if (score >= 60) {
    System.out.println("D");
}
if (score < 60) {
    System.out.println("F");
}
...
The `else...if` form

- There is also an `else...if` form to handle more “cases” or “branches”

  ```java
  int a, b;
  ...
  if (a > b) {
      System.out.println(a + " is greater than " + b);
  }
  else if (a < b) {
      System.out.println(a + " is less than " + b);
  }
  else {
      System.out.println(a + " is equal to " + b);
  }
  ```

- You can use multiple `else...if` branches as needed
Misuse of `if` (corrected)

```java
Scanner input = new Scanner(System.in);
System.out.print("What was your score? ");
int score = input.nextInt();
if (score >= 90) {
    System.out.println("A");
}
else if (score >= 80) {
    System.out.println("B");
}
else if (score >= 70) {
    System.out.println("C");
}
else if (score >= 60) {
    System.out.println("D");
}
else { // (score < 60)
    System.out.println("F");
}
...
else and else...if structures

- **Exactly 1 path (mutually exclusive)**
  ```java
  if (test1) {
    statement(s);
  }
  else if (test2) {
    statement(s);
  }
  else {
    statement(s);
  }
  ```

- **0 or 1 path (mutually exclusive)**
  ```java
  if (test1) {
    statement(s);
  }
  else if (test2) {
    statement(s);
  }
  else if (test3) {
    statement(s);
  }
  ```

- **0, 1, or many paths (independent tests, not mutually exclusive)**
  ```java
  if (test1) {
    statement(s);
  }
  if (test2) {
    statement(s);
  }
  if (test3) {
    statement(s);
  }
  ```
Testing for equal or not

- To test for equal use the `==` operator (don’t use `=`)
  ```java
  if (a == b) {
      System.out.println(a + " is equal to " + b);
  }
  - A common error is to mix up `==` and `=

- You can also test for inequality using the `!=` operator
  ```java
  if (a != b) {
      System.out.println(a + " is not equal to " + b);
  }
  ```
Nesting if statements

You can nest if statements as deep as needed. Example: find the max of 3 variables:

```c
int a, b, c, max;   // assume a, b, c are assigned values
if (a > b) {
    if (a > c) {
        max = a;
    }
    else {
        max = c;
    }
} else {
    if (b > c) {
        max = b;
    } else {
        max = c;
    }
}
```
Testing multiple conditions

- What if you wanted to test for 2 or more conditions being true?
  
  ```java
  if ( condition1 AND condition2 ) { . . . }
  ```

- You could use nested if statement but that can be clumsy and not very clear or easy to read

- Use logical operators instead

- if (a && b) { . . . }
  - Overall expression is true if a and b are both true

- if (a || b) { . . . }
  - Overall expression is true if a or b is true
Logical operator: AND (&&)

- `&&` represents the AND operation
  - In order for the expression `(A && B)` to be true, both `A` and `B` have to be true
  - Otherwise, the expression evaluates to false

- Truth table:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A &amp;&amp; B</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>-------</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Example: &&

- Example: testing for 1 <= value <= 7:

```java
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Pick a number between 1 and 7: ");
    int val = input.nextInt();
    if ((val >= 1) && (val <= 7)) {
        System.out.println("Good! You chose: " + val);
    } else {
        System.out.println("You chose poorly!");
    }
}
```
Logical operator: OR (||)

- || represents the OR operation
  - In order for the expression (A || B) to be true, either A or B or both have to be true
  - Otherwise, the expression evaluates to false

- Truth table:

| A   | B   | A || B |
|-----|-----|-------|
| false | false | false |
| false | true  | true  |
| true  | false | true  |
| true  | true  | true  |
Example: ||

- Example: testing for val < 1 or val > 7:

```java
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Pick a number between 1 and 7: ");
    int val = input.nextInt();
    if ((val < 1) || (val > 7)) {
        System.out.println("Good! You chose: " + val);
    }
    else {
        System.out.println("You chose poorly!");
    }
}
```
Evaluating logical expressions

- Relational operators have lower precedence than math
  
  \[ 4 \times 3 \leq 4 + 7 \times (4 - 2) \]
  
  \[ 12 \leq 4 + 7 \times 2 \]
  
  \[ 12 \leq 18 \]
  
  true

- Relational operators cannot be “chained” as in algebra
  
  \[ 4 \leq a \leq 20 \]
  
  true \leq 20     (assume x is 12)
  
  error!

  - Instead combine multiple tests with && or ||
    
    \[ 4 \leq a \land a \leq 20 \]
    
    true     &&  true
    
    true

Logical operator: NOT (\(!\))

- \(!\) represents the NOT operation
  - In order for the expression \(!A\) to be true, A has to be false
  - If A is true, \(!A\) is false

- Truth table:

<table>
<thead>
<tr>
<th>A</th>
<th>(!A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>
Logical questions

- Logical operators have lower precedence than relational operators.
- What is the result of each of the following expressions?

```c
int x = 40;
int y = 10;
int z = 14;
```

1. `y <= x && x >= z`
2. `x % 2 == y % 3 || x % 2 == z % 3`
3. `y <= x + z && x >= y + z`
4. `!(x > y && x > z || x == z)`
5. `(x + y) % 3 == 0 || !((x - y) % 2 == 0)`
Factoring if...else code

- factoring: extracting common/redundant code
  - Can reduce or eliminate redundancy from if...else code

- Example:

```java
if (a == 1) {
    System.out.println(a);
    x = 3;
    b = b + x;
}
else if (a == 2) {
    System.out.println(a);
    x = 6;
    y = y + 10;
    b = b + x;
}
else { // a == 3
    System.out.println(a);
    x = 9;
    b = b + x;
}
```

```java
System.out.println(a);
x = 3 * a;
if (a == 2) {
    y = y + 10;
}
b = b + x;
```
Boolean variables as conditions

Since boolean variables are (simple) boolean expressions, we can also use a boolean variable wherever we can use a boolean expression

```java
// assume a and b are int's with a value assigned
boolean cond1 = a > b;
boolean cond2 = a < b;
boolean cond3 = a == b;
if (cond1) {
    System.out.println(a + " is greater than " + b);
}
else if (cond2) {
    System.out.println(a + " is less than " + b);
}
else if (cond3) {
    System.out.println(a + " is equal to " + b);
}
else {
    System.out.println(“Something weird has happened.”);
}
```
Random example

- Any set of possible values or "meaning" can be mapped to integers
- A program to randomly play Rock-Paper-Sissors:

```java
import java.util.Random;
public class Rand {
    public static void main(String[] args) {
        Random rand = new Random();
        int r = rand.nextInt(3); // to get 0, 1, or 2
        if (r == 0) {
            System.out.println("Rock");
        } else if (r == 1) {
            System.out.println("Paper");
        } else { // r == 2
            System.out.println("Sissors");
        }
    }
}
```

**Note:** Section 3.7 of text uses `Math.random()` instead of using `java.util.Random`. You may use either one, but `java.util.Random` is a little easier to use.

See Rand.java
See Rand2.java
Practice: if...else

- Write a program with a main method that prompts the user to enter two real numbers representing x- and y-coordinates of a point on a 2D plane. Then it handles the following cases:
  - If the point falls on either axis, it prints 0
  - If the point falls in the first quadrant, it prints 1
  - If the point falls in the second quadrant, it prints 2
  - If the point falls in the third quadrant, it prints 3
  - If the point falls in the fourth quadrant, it prints 4

- For the latter four cases given above write your if...else statement twice: once without using nested if...else’s and once more using nested if...else’s.
switch statements

• Multi-way conditionals can be written in switch form

```java
int code = <some_integer_value>;
switch (code) {
    case 0: System.out.println("zero");
            break;
    case 1: System.out.println("hana");
            break;
    case 2: System.out.println("two");
            break;
    case 3: System.out.println("tres");
            break;
    default: System.out.println("Error: invalid code");
             System.exit(1);
}
```

**break**: breaks out of the switch statement

Try leaving out one of the break statements and see what happens

*Syntactic sugar for multi-way if...else’s*

You may note the form so that you can read others’ code.