Boolean Arithmetic
Recall: Java's Primitive Data Types

<table>
<thead>
<tr>
<th>Discrete Types</th>
<th>Continuous Types</th>
<th>Non-numeric Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>float</td>
<td>boolean</td>
</tr>
<tr>
<td>short</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td></td>
<td>char</td>
</tr>
<tr>
<td>long</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The boolean Type

The boolean type has two possible values: true and false.

Boolean variables are declared and initialized just like other primitive data types:

```java
boolean iAmSoSmrt = false;    // just like int i = 2;

boolean minor = (age < 21);   // just like int x = y*2;
```
Relational expressions

- Relational expressions have
  - numeric arguments and
  - boolean values.

- They use one of the following six 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>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>
Evaluating Relational expressions

- Relational operators have lower precedence than math operators.

  \[
  5 \times 7 \geq 3 + 5 \times (7 - 1)
  \]
  \[
  5 \times 7 \geq 3 + 5 \times 6
  \]
  \[
  35 \geq 3 + 30
  \]
  \[
  35 \geq 33
  \]
  \[
  \text{true}
  \]

- Relational operators cannot be chained (unlike math operators)

  \[
  2 \leq x \leq 10
  \]
  \[
  \text{true} \leq 10
  \]
  \[
  \text{error!}
  \]
Logical operators

- Logical operators have
  - boolean arguments and
  - boolean values

<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>(9 != 6) &amp;&amp; (2 &lt; 3)</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>or</td>
</tr>
<tr>
<td>!</td>
<td>not</td>
<td>!(7 &gt; 0)</td>
<td></td>
</tr>
</tbody>
</table>
Boolean expressions

What is the result of each of the following expressions?

```java
int x = 42;
int y = 17;
int z = 25;
```

- `y < x && y <= z`
- `x % 2 == y % 2 || x % 2 == z % 2`
- `x <= y + z && x >= y + z`
- `!(x < y && x < z)`
- `(x + y) % 2 == 0 || !((z - y) % 2 == 0)`

Answers: true, false, true, true, false
if/else statements
Conditionals

- “If you eat your vegetables, then you can have dessert.”

- “If you do your homework, then you may go outside to play, or else you’ll be grounded for life.”
The if statement

- **if statement**: A *control structure* that executes a block of statements only if a certain condition is true.

- **General syntax**:
  ```java
  if (<test>) {
      <statement(s>) ;
  }
  ```

- **Example (with grade inflation)**:
  ```java
  if (gpa >= 2.0) {
      System.out.println("You get an A!");
  }
  ```
if statement flow chart

Is the test true?

- no
  - execute statement after if statement

- yes
  - execute the controlled statement(s)
The if/else statement

- **if/else statement**: A control structure that executes one block of statements if a certain condition is true, and a second block of statements if it is false. We refer to each block as a *branch*.

- **General syntax**:

  ```java
  if (<test>) {
      <statement(s>)
  } else {
      <statement(s>)
  }
  ```

- **Example**:

  ```java
  if (gpa >= 3.0) {
      System.out.println("Welcome to Temple!");
  } else {
      System.out.println("Try applying to Penn.");
  }
  ```
if/else statement flow chart

- Is the test true?
  - no: execute the 'else' controlled statement(s)
  - yes: execute the 'if' controlled statement(s)

execute statement after if/else statement
if/else: Exercise

- Write code to read a number from the user and print whether it is even or odd using an if/else statement.

Example executions:
Type a number: 42
Your number is even

Type a number: 17
Your number is odd
### Chained if/else statements

- **Chained if/else statement**: A chain of `if/else` that can select between many different outcomes based on several tests.

- **General syntax**:
  ```java
  if (<test>) {
      <statement(s)>;
  } else if (<test>) {
      <statement(s)>;
  } else {
      <statement(s)>;
  }
  ```

- **Example**:
  ```java
  if (number > 0) {
      System.out.println("Positive");
  } else if (number < 0) {
      System.out.println("Negative");
  } else {
      System.out.println("Zero");
  }
  ```
### Chained `if/else` variations

- A chained `if/else` can end with an `if` or an `else`.
  - If it ends with `else`, one of the branches must be taken.
  - If it ends with `if`, the program might not execute any branch.

```plaintext
if (<test>) {
    <statement(s)>;
} else if (<test>) {
    <statement(s)>;
} else {
    <statement(s)>;
}
```

```plaintext
if (<test>) {
    <statement(s)>;
} else if (<test>) {
    <statement(s)>;
} else if (<test>) {
    <statement(s)>;
}
```
if (<test>) {
    <statement(s)>;
} else if (<test>) {
    <statement(s)>;
} else {
    <statement(s)>;
}
Chained if/else if flow chart

```java
if (<test>) {
    <statement(s)>;
} else if (<test>) {
    <statement(s)>;
} else if (<test>) {
    <statement(s)>;
}
```
Chained if/else variations

```java
if (place == 1) {
    System.out.println("You win the gold medal! ");
} else if (place == 2) {
    System.out.println("You win a silver medal!");
} else if (place == 3) {
    System.out.println("You earned a bronze medal.");
}
```

- Are there any cases where this code will not print a message?

- How could we modify it to print a message to non-medalists?
if (<test>) {
    <statement(s)>;
}
if (<test>) {
    <statement(s)>;
}
if (<test>) {
    <statement(s)>;
}
Summary: if/else structures

- Choose exactly 1 set of statements
  
  ```java
  if (<test>) {
    <statement(s>);
  } else if (<test>) {
    <statement(s>);
  } else {
    <statement(s>);
  }
  ```

- Choose 0, 1, or more set of statements
  
  ```java
  if (<test>) {
    <statement(s>);
  } else if (<test>) {
    <statement(s>);
  } else if (<test>) {
    <statement(s>);
  }
  ```

- Choose 0 or 1 set of statements
  
  ```java
  if (<test>) {
    <statement(s>);
  } else if (<test>) {
    <statement(s>);
  } else if (<test>) {
    <statement(s>);
  }
  ```
Which \texttt{if/else} construct to use?

- Reading the user's GPA and printing whether the student is on the dean's list (3.8 to 4.0) or honor roll (3.5 to 3.8)
  \hspace{2cm} \texttt{if / else if}

- Printing whether a number is even or odd
  \hspace{2cm} \texttt{if / else}

- Printing whether a user is lower-class, middle-class, or upper-class based on their income
  \hspace{2cm} \texttt{if / else if / else}

- Determining whether a number is divisible by 2, 3, and/or 5
  \hspace{2cm} \texttt{if / if / if}

- Printing a user's grade of A, B, C, D, or F based on their percentage in the course
  \hspace{2cm} \texttt{if / else if / else if / else if / else}
Subtleties of if/else
Scope

- **scope**: The portion of a program where a given variable exists.
  - A variable's scope is from its declaration to the end of the `{}` braces in which it was declared.

```
public class ScopeExample {
    public static void main(String[] args) {
        int x = 3;
        int y = 7;
        if(x > 0 && y > 0) {
            int sumPositives = x + y;
        } else {
            sumPositives = 0; // illegal: sumPositives is out of scope
        }
        // illegal: sumPositives is out of scope
        System.out.println("sum = " + sumPositives);
    }
}
```

- Why not just have the scope of a variable be the whole program?
Variable initialization

String message;
if (gpa >= 3.0) {
    message = "Welcome to Temple!";
}
if (gpa >= 2.0) {
    message = "Have you considered applying to Penn?";
}
if (gpa < 2.0) {
    message = "I hear Harvard still needs students...";
}
System.out.println(message);

- The compiler will complain that "variable message might not have been initialized". Why?
Variable initialization

The solution:

```java
String message;
if (gpa >= 3.0) {
    message = "Welcome to Temple!";
} else if (gpa >= 2.0) {
    message = "Have you considered applying to Penn?";
} else { // gpa < 2.0
    message = "I hear Harvard still needs students...";
}
System.out.println(message);
```
Putting it all together: Exercises

- Write a program named **NumUnique** that reads two integers from the user and displays how many unique values were passed.
  - For example, if the user enters 3 and 7, **NumUnique** displays 2 because 3 and 7 are two unique numbers, but if the user enters 4 and 4, it displays 1 because 4 and 4 only represent one unique number.

- Write a program named **DividesEvenly** that reads two integers from the user and displays “true” if the first evenly divides the second, and false otherwise.
  - For example, if the user enters 7 and 28, **DividesEvenly** displays true because 7 is a factor of 28.
Magic numbers are **BAD NEWS**.

They are numeric constants other than 0, 1, and 2 that appear in the body of a program.
public class Magic {
    public static void main(String [] args) {
        for(int i=0; i<10; i++) {
            if(i % 3 == 0) {
                System.out.println("Divisible by 3");
            }
        }
    }
}
A class constant is a variable
- whose scope is the entire class, and
- whose value can never change after it has been initialized.

To give it the right scope, simply declare it right inside the class:

```java
public class MyClass {
    public static final int myConstant = 4;
}
```

The final keyword means its value can’t be changed.
public class Magic {
    public static final int upperBound = 10;
    public static final int divisor = 3;

    public static void main(String [] args) {
        for(int i=0; i<upperBound; i++) {
            if(i % divisor ==0) {
                System.out.println("Divisible by "+divisor);
            }
        }
    }
}
Factoring if/else
Factoring if/else

- **factoring**: extracting common/redundant code

- Factoring *if/else* code reduces the size of the *if* and *else* statements

- Factoring tips:
  - If the start of each branch is the same, move it *before* the *if/else*.
  - If the end of each branch is the same, move it *after* the *if/else*.
Factoring: Before

```java
if (money < 500) {
    System.out.println("You have, $" + money + " left.");
    System.out.print("Caution! Bet carefully.");
    System.out.print("How much do you want to bet? ");
    bet = console.nextInt();
} else if (money < 1000) {
    System.out.println("You have, $" + money + " left.");
    System.out.print("Consider betting moderately.");
    System.out.print("How much do you want to bet? ");
    bet = console.nextInt();
} else {
    System.out.println("You have, $" + money + " left.");
    System.out.print("You may bet liberally.");
    System.out.print("How much do you want to bet? ");
    bet = console.nextInt();
}
```
Factoring: After

System.out.println("You have, "+ money + " left.");

if (money < 500) {
    System.out.print("Caution! Bet carefully.");
} else if (money < 1000) {
    System.out.print("Consider betting moderately.");
} else {
    System.out.print("You may bet liberally.");
}

System.out.print("How much do you want to bet? ");
bet = console.nextInt();