1.1 Your First Program
Why Programming?

Why programming? Need to tell computer what you want it to do.

Naive ideal. Natural language instructions.

“Please simulate the motion of N heavenly bodies, subject to Newton’s laws of motion and gravity.”

Prepackaged solutions (apps)? Great, when what they do is what you want.

Programming. Enables you to make a computer do anything you want.

well, almost anything [stay tuned]
Languages

**Machine languages.** Tedious and error-prone.

**Natural languages.** Ambiguous; can be difficult to parse.

- Kids Make Nutritious Snacks.
- Red Tape Holds Up New Bridge.
- Police Squad Helps Dog Bite Victim.
- Local High School Dropouts Cut in Half.

[real newspaper headlines, compiled by Rich Pattis]

**High-level programming languages.** Acceptable tradeoff.

“Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.” – Donald Knuth
Why Program?

Why program?

• A natural, satisfying and creative experience.
• Enables accomplishments not otherwise possible.
• Opens new world of intellectual endeavor.

First challenge. Learn a programming language.

Next question. Which one?

Naive ideal. A single programming language.
Our Choice: Java

Java features.
• Widely used.
• Widely available.
• Embraces full set of modern abstractions.
• Variety of automatic checks for mistakes in programs.

Java economy.
• Mars rover.
• Cell phones.
• Blu-ray Disc.
• Web servers.
• Medical devices.
• Supercomputing.
• ...

James Gosling
http://java.net/jag

$100 billion, 5 million developers
Why Java?

Java features.
• Widely used.
• Widely available.
• Embraces full set of modern abstractions.
• Variety of automatic checks for mistakes in programs.

Facts of life.
• No language is perfect.
• We need to choose some language.

Our approach.
• Minimal subset of Java.
• Develop general programming skills that are applicable to many languages

It’s not about the language!

“There are only two kinds of programming languages: those people always [gripe] about and those nobody uses.”
– Bjarne Stroustrup
### A Rich Subset of the Java Language

#### Built-In Types
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>double</td>
</tr>
<tr>
<td>long</td>
<td>String</td>
</tr>
<tr>
<td>char</td>
<td>boolean</td>
</tr>
</tbody>
</table>

#### Flow Control
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>if</td>
<td>else</td>
</tr>
<tr>
<td>for</td>
<td>while</td>
</tr>
</tbody>
</table>

#### Boolean
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td></td>
</tr>
</tbody>
</table>

#### Punctuation
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
<td>}</td>
</tr>
<tr>
<td>(</td>
<td>)</td>
</tr>
<tr>
<td>,</td>
<td>;</td>
</tr>
</tbody>
</table>

#### Assignment
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td></td>
</tr>
</tbody>
</table>

#### String
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>length()</td>
<td>compareTo()</td>
</tr>
<tr>
<td>charAt()</td>
<td>matches()</td>
</tr>
</tbody>
</table>

#### Arrays
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a[i]</td>
<td></td>
</tr>
<tr>
<td>new</td>
<td></td>
</tr>
<tr>
<td>a.length</td>
<td></td>
</tr>
</tbody>
</table>

#### Math Library
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.sin()</td>
<td>Math.cos()</td>
</tr>
<tr>
<td>Math.log()</td>
<td>Math.exp()</td>
</tr>
<tr>
<td>Math.sqrt()</td>
<td>Math.pow()</td>
</tr>
<tr>
<td>Math.min()</td>
<td>Math.max()</td>
</tr>
<tr>
<td>Math.abs()</td>
<td>Math.PI</td>
</tr>
</tbody>
</table>

#### System
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>System.out.println()</td>
<td></td>
</tr>
<tr>
<td>System.out.print()</td>
<td></td>
</tr>
<tr>
<td>System.out.printf()</td>
<td></td>
</tr>
</tbody>
</table>

#### Math Library
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.sin()</td>
<td>Math.cos()</td>
</tr>
<tr>
<td>Math.log()</td>
<td>Math.exp()</td>
</tr>
<tr>
<td>Math.sqrt()</td>
<td>Math.pow()</td>
</tr>
<tr>
<td>Math.min()</td>
<td>Math.max()</td>
</tr>
<tr>
<td>Math.abs()</td>
<td>Math.PI</td>
</tr>
</tbody>
</table>

#### Primitive Numeric Types
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>*</td>
<td>/</td>
</tr>
<tr>
<td>%</td>
<td>++</td>
</tr>
<tr>
<td>--</td>
<td>&gt;</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&gt;=</td>
</tr>
<tr>
<td>==</td>
<td>!=</td>
</tr>
</tbody>
</table>

#### Objects
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>static</td>
</tr>
<tr>
<td>public</td>
<td>private</td>
</tr>
<tr>
<td>final</td>
<td>toString()</td>
</tr>
<tr>
<td>new</td>
<td>main()</td>
</tr>
</tbody>
</table>
Program Development
Create the program by typing it into a text editor, and save it as HelloWorld.java.

```java
/**
 * Prints "Hello, World"
 * Everyone's first Java program.
 */

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}

HelloWorld.java
Create the program by typing it into a text editor, and save it as `HelloWorld.java`.

Compile it by typing at the command-line:

```
% javac HelloWorld.java
```

(or click the Compile button in DrJava)

This creates a Java bytecode file named: `HelloWorld.class`. 
Programming in Java

- Create the program by typing it into a text editor, and save it as `HelloWorld.java`.

- Compile it by typing at the command-line:
  `javac HelloWorld.java`.

- Execute it by typing at the command-line:
  `java HelloWorld`.

```
% javac HelloWorld.java
% java HelloWorld
Hello, World
```
Program Development (using DrJava)

Program development in Java (using DrJava).

1. **Edit** your program using the built-in text editor.
2. Compile it to create an executable file.
3. Run your program.
Program Development (using DrJava)

Program development in Java (using DrJava).

1. Edit your program.
2. **Compile** it by clicking the “compile” button.
3. Run your program.
Program Development (using DrJava)

Program development in Java (using DrJava).

1. Edit your program.
2. Compile it to create an executable file.
3. Run your program by clicking the “run” button or using the command line.

Alternative 1: run button (OK if no args)
Alternative 2: command line (to provide args)
Note: Program Style

Three versions of the same program.

```java
// java HelloWorld
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

Fonts, color, comments, and extra space are not relevant to Java.

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

/**************************************************************************
* Compilation:  javac HelloWorld.java
* Execution:    java HelloWorld
* Prints "Hello, World". By tradition, this is everyone's first program.
* % java HelloWorld
* Hello, World
***************************************************************************/

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```
Note: Program Style

Different styles are appropriate in different contexts.
• DrJava
• Booksite
• Book
• COS 126 assignment

Enforcing consistent style can
• Stifle creativity.
• Confuse style rules with language rules.

Emphasizing consistent style can
• Make it easier to spot errors.
• Make it easier for others to read and use code.
• Enable development environment to provide useful visual cues.

Bottom line for COS 126: Life is easiest if you use DrJava style.
1.2 Built-in Types of Data
**Built-in Data Types**

Data type. A set of values and operations defined on those values.

<table>
<thead>
<tr>
<th>type</th>
<th>set of values</th>
<th>literal values</th>
<th>operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>characters</td>
<td>'A'</td>
<td>compare</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'a'</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>sequences of characters</td>
<td>&quot;Hello World&quot;</td>
<td>concatenate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;CS is fun&quot;</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>integers</td>
<td>17</td>
<td>add, subtract,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12345</td>
<td>multiply, divide</td>
</tr>
<tr>
<td>double</td>
<td>floating-point numbers</td>
<td>3.1415</td>
<td>add, subtract,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.022e23</td>
<td>multiply, divide</td>
</tr>
<tr>
<td>boolean</td>
<td>truth values</td>
<td>true</td>
<td>and, or, not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>
**Basic Definitions**

**Variable.** A name that refers to a value.

**Literal.** Programming-language representation of a value.

**Assignment statement.** Associates a value with a variable.

**Program.** Sequence of statements.

```c
int a, b;
a = 1234;
b = 99;
int c = a + b;
```
### Trace

Table of variable values after each statement.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>int a, b;</td>
<td>undefined</td>
<td>undefined</td>
<td>undefined</td>
</tr>
<tr>
<td>a = 1234;</td>
<td>1234</td>
<td>undefined</td>
<td>undefined</td>
</tr>
<tr>
<td>b = 99;</td>
<td>1234</td>
<td>99</td>
<td>undefined</td>
</tr>
<tr>
<td>int t = a;</td>
<td>1234</td>
<td>99</td>
<td>1234</td>
</tr>
<tr>
<td>a = b;</td>
<td>99</td>
<td>99</td>
<td>1234</td>
</tr>
<tr>
<td>b = t;</td>
<td>99</td>
<td>1234</td>
<td>1234</td>
</tr>
</tbody>
</table>
String data type. Useful for program input and output.

<table>
<thead>
<tr>
<th>values</th>
<th>sequences of characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>typical literals</td>
<td>&quot;Hello, &quot; &quot;1 &quot; &quot; * &quot;</td>
</tr>
<tr>
<td>operation</td>
<td>concatenate</td>
</tr>
<tr>
<td>operator</td>
<td>+</td>
</tr>
</tbody>
</table>

String data type examples

expression          value

"Hi, " + "Bob"      "Hi, Bob"
"1" + " 2 " + "1"   "1 2 1"
"1234" + " + " + "99" "1234 + 99"
"1234" + "99"       "123499"

Important note: meaning of characters depends on context!

"1234" + " + " + "99"

String concatenation examples

expression          value

"Hi, " + "Bob"      "Hi, Bob"
"1" + " 2 " + "1"   "1 2 1"
"1234" + " + " + "99" "1234 + 99"
"1234" + "99"       "123499"
Example: Subdivisions of a Ruler

```java
public class Ruler {
    public static void main(String[] args) {
        String ruler1 = "1";
        String ruler2 = ruler1 + " 2 " + ruler1;
        String ruler3 = ruler2 + " 3 " + ruler2;
        String ruler4 = ruler3 + " 4 " + ruler3;
        System.out.println(ruler4);
    }
}
```

% java Ruler
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
**Integers**

**int data type.** Useful for calculations, expressing algorithms.

<table>
<thead>
<tr>
<th>values</th>
<th>integers between $-2^{31}$ and $+2^{31} - 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>typical literals</td>
<td>1234  99  -99  0  1000000</td>
</tr>
<tr>
<td>operations</td>
<td>add  subtract  multiply  divide  remainder</td>
</tr>
<tr>
<td>operators</td>
<td>+    -      *        /        %</td>
</tr>
</tbody>
</table>

- **expression** | **value** | **comment**                    |
- 5 + 3          | 8         |                               |
- 5 - 3          | 2         |                               |
- 5 * 3          | 15        |                               |
- 5 / 3          | 1         | no fractional part           |
- 5 % 3          | 2         | remainder                     |
- 1 / 0          | 2         | run-time error                |
- 3 * 5 - 2      | 13        | * has precedence              |
- 3 + 5 / 2      | 5         | / has precedence              |
- 3 - 5 - 2      | -4        | left associative              |
- (3 - 5) - 2    | -4        | better style                  |

there is a largest int and a smallest int
public class IntOps
{
    public static void main(String[] args)
    {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int sum  = a + b;
        int prod = a * b;
        int quot = a / b;
        int rem  = a % b;
        System.out.println(a + " + " + b + " = " + sum);
        System.out.println(a + " * " + b + " = " + prod);
        System.out.println(a + " / " + b + " = " + quot);
        System.out.println(a + " % " + b + " = " + rem);
    }
}

% javac IntOps.java
% java IntOps 1234 99
1234 + 99 = 1333
1234 * 99 = 122166
1234 / 99 = 12
1234 % 99 = 46

1234 = 12*99 + 46
Floating-Point Numbers

**double data type.** Useful in scientific applications.

<table>
<thead>
<tr>
<th>values</th>
<th>approximations to real numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>typical literals</td>
<td>3.14159  6.022e23  -3.0  2.0  1.4142135623730951</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>operations</th>
<th>add</th>
<th>subtract</th>
<th>multiply</th>
<th>divide</th>
<th>remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>operators</td>
<td>+</td>
<td>-</td>
<td>*</td>
<td>/</td>
<td>%</td>
</tr>
</tbody>
</table>

**double data type**

<table>
<thead>
<tr>
<th>expression</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.141 + .03</td>
<td>3.171</td>
</tr>
<tr>
<td>3.141 - .03</td>
<td>3.111</td>
</tr>
<tr>
<td>6.02e23/2</td>
<td>3.01E+23</td>
</tr>
<tr>
<td>5.0 / 3.0</td>
<td>1.66666666666666700</td>
</tr>
<tr>
<td>10.0 % 3.141</td>
<td>0.577</td>
</tr>
<tr>
<td>1.0 / 0.0</td>
<td>Infinity</td>
</tr>
<tr>
<td>Math.sqrt(2.0)</td>
<td>1.4142135623731000</td>
</tr>
<tr>
<td>Math.sqrt(-1.0)</td>
<td>NaN</td>
</tr>
</tbody>
</table>

examples of double operations

there is a largest double and a smallest double

special value

special value "not a number"
**Excerpts from Java’s Math Library**

```java
public class Math {
    double abs(double a) // absolute value of a
    double max(double a, double b) // maximum of a and b
    double min(double a, double b) // minimum of a and b
    double sin(double theta) // sine function
    double cos(double theta) // cosine function
    double tan(double theta) // tangent function
    double exp(double a) // exponential (e^a)
    double log(double a) // natural log (log_e a, or ln a)
    double pow(double a, double b) // raise a to the bth power (a^b)
    long round(double a) // found to the nearest integer
    double random() // random number in [0, 1)
    double sqrt(double a) // square root of a
    double E // value of e (constant)
    double PI // value of p (constant)
}
```

In radians. Use `toDegrees()` and `toRadians()` to convert.

Inverse functions `asin()`, `acos()`, and `atan()` also available.

Also defined for `int`, `long`, and `float`.
Ex. Solve quadratic equation $x^2 + bx + c = 0$.

\[
\text{roots} = \frac{-b \pm \sqrt{b^2 - 4c}}{2}
\]

public class Quadratic
{
    public static void main(String[] args)
    {
        // Parse coefficients from command-line.
        double b = Double.parseDouble(args[0]);
        double c = Double.parseDouble(args[1]);

        // Calculate roots.
        double discriminant = b*b - 4.0*c;
        double d = Math.sqrt(discriminant);
        double root1 = (-b + d) / 2.0;
        double root2 = (-b - d) / 2.0;

        // Print them out.
        System.out.println(root1);
        System.out.println(root2);
    }
}
Testing. Some valid and invalid inputs.

% java Quadratic -3.0 2.0
2.0
1.0

% java Quadratic -1.0 -1.0
1.618033988749895
-0.6180339887498949

% java Quadratic 1.0 1.0
NaN
NaN

% java Quadratic 1.0 hello
java.lang.NumberFormatException: hello

% java Quadratic 1.0
java.lang.ArrayIndexOutOfBoundsException

\[ x^2 - 3x + 2 \] command-line arguments

\[ x^2 - x - 1 \] golden ratio

\[ x^2 + x + 1 \] “not a number”
**Boolean data type.** Useful to control logic and flow of a program.

<table>
<thead>
<tr>
<th>values literals operations operators</th>
<th>true or false</th>
<th>and</th>
<th>or</th>
<th>not</th>
</tr>
</thead>
<tbody>
<tr>
<td>true false</td>
<td>&amp;&amp;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| a | !a | a | b | a && b | a || b |
|---|----|---|---|--------|--------|
| true | false | false | false | false | false |
| false | true | false | true | false | true |
| true | false | true | false | true | true |
| true | true | true | true | true | true |

**Truth-table definitions of boolean operations**
Comparison Operators

Comparison operators.

- Two operands of the same type.
- Result: a value of type boolean.

<table>
<thead>
<tr>
<th>op</th>
<th>meaning</th>
<th>true</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equal</td>
<td>2 == 2</td>
<td>2 == 3</td>
</tr>
<tr>
<td>!=</td>
<td>not equal</td>
<td>3 != 2</td>
<td>2 != 2</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>2 &lt; 13</td>
<td>2 &lt; 2</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal</td>
<td>2 &lt;= 2</td>
<td>3 &lt;= 2</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>13 &gt; 2</td>
<td>2 &gt; 13</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal</td>
<td>3 &gt;= 2</td>
<td>2 &gt;= 3</td>
</tr>
</tbody>
</table>

comparison operators

non-negative discriminant? \(( b*b - 4.0*a*c ) >= 0.0\)
beginning of a century? \(( year % 100 ) == 0\)
legal month? \(( month >= 1 ) && ( month <= 12 )\)

comparison examples
Q. Is a given year a leap year?
A. Yes if either (i) divisible by 400 or (ii) divisible by 4 but not 100.

```java
public class LeapYear {
  public static void main(String[] args) {
    int year = Integer.parseInt(args[0]);
    boolean isLeapYear;

    // divisible by 4 but not 100
    isLeapYear = (year % 4 == 0) && (year % 100 != 0);

    // or divisible by 400
    isLeapYear = isLeapYear || (year % 400 == 0);

    System.out.println(isLeapYear);
  }
}
```

% java LeapYear 2004 true
% java LeapYear 1900 false
% java LeapYear 2000 true
Type Conversion

**Type conversion.** Convert from one type of data to another.

- **Automatic** (done by Java when no loss of precision; or with strings).
- Explicitly defined by function call.
- **Cast** (write desired type within parens).

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
<th>value</th>
<th>type style</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;1234&quot; + 99</td>
<td>String</td>
<td>&quot;123499&quot;</td>
<td>automatic</td>
</tr>
<tr>
<td>Integer.parseInt(&quot;123&quot;)</td>
<td>int</td>
<td>123</td>
<td>explicit</td>
</tr>
<tr>
<td>(int) 2.71828</td>
<td>int</td>
<td>2</td>
<td>cast</td>
</tr>
<tr>
<td>Math.round(2.71828)</td>
<td>long</td>
<td>3</td>
<td>explicit</td>
</tr>
<tr>
<td>(int) Math.round(2.71828)</td>
<td>int</td>
<td>3</td>
<td>cast</td>
</tr>
<tr>
<td>(int) Math.round(3.14159)</td>
<td>int</td>
<td>3</td>
<td>cast</td>
</tr>
<tr>
<td>11 * 0.3</td>
<td>double</td>
<td>3.3</td>
<td>automatic</td>
</tr>
<tr>
<td>(int) 11 * 0.3</td>
<td>double</td>
<td>3.3</td>
<td>cast, automatic</td>
</tr>
<tr>
<td>11 * (int) 0.3</td>
<td>int</td>
<td>0</td>
<td>cast</td>
</tr>
<tr>
<td>(int) (11 * 0.3)</td>
<td>int</td>
<td>3</td>
<td>cast, automatic</td>
</tr>
</tbody>
</table>

Pay attention to the type of your data. Type conversion can give counterintuitive results but gets easier to understand with practice.
Type Conversion Example: Random Integer

Ex. Generate a pseudo-random number between 0 and N-1.

```java
class RandomInt {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        double r = Math.random();
        int n = (int) (r * N);
        System.out.println("random integer is "+n);
    }
}
```

% java RandomInt 6
random integer is 3

% java RandomInt 6
random integer is 0

% java RandomInt 10000
random integer is 3184
Summary

A data type is a set of values and operations on those values.

- String: text processing, input and output.
- double, int: mathematical calculation.
- boolean: decision making.

Be aware. In Java you must:

- Declare type of values.
- Convert between types when necessary.

Why do we need types?

- Type conversion must be done at some level.
- Compiler can help do it correctly.
- Example: In 1996, Ariane 5 rocket exploded after takeoff because of bad type conversion.
1.3 Conditionals and Loops
Control Flow

Control flow.

- Sequence of statements that are actually executed in a program.
- Conditionals and loops: enable us to choreograph control flow.

straight-line control flow

control flow with conditionals and loops
Conditionals
The *if* statement. A common branching structure.

- Evaluate a boolean expression.
- If true, execute some statements.
- If false, execute other statements.

```
if (boolean expression) {
    statement T;
}
else {
    statement F;
}
```
The **if** statement. A common branching structure.

- Evaluate a **boolean** expression.
- If **true**, execute some statements.
- If **false**, execute other statements.
If Statement

Ex. Take different action depending on value of variable.

```java
public class Flip {
    public static void main(String[] args) {
        if (Math.random() < 0.5) System.out.println("Heads");
        else System.out.println("Tails");
    }
}
```

% java Flip
Heads
% java Flip
Heads
% java Flip
Tails
% java Flip
Heads
If-Else: Leap Year revisited

If-else. Take different action depending on value of variable.

- If isLeapYear is true, then print "is a".
- Otherwise, print "isn't a ".

```java
System.out.print(year + " ");

if (isLeapYear) {
    System.out.print("is a");
} else {
    System.out.print("isn't a");
}

System.out.println(" leap year");
```
public class LeapYear
{
    public static void main(String[] args)
    {
        int year = Integer.parseInt(args[0]);
        boolean isLeapYear;

        // divisible by 4 but not 100
        isLeapYear = (year % 4 == 0) && (year % 100 != 0);

        // or divisible by 400
        isLeapYear = isLeapYear || (year % 400 == 0);

        System.out.print(year + " ");

        if (isLeapYear) {
            System.out.print("is a");
        }
        else {
            System.out.print("isn't a");
        }

        System.out.println(" leap year");
    }
}
Oblivious Sorting

**Sort.** Read in 3 integers and rearrange them in ascending order.

```java
public class Sort3 {
    public static void main(String[] args) {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int c = Integer.parseInt(args[2]);

        if (b > c) { int t = b; b = c; c = t; }
        if (a > b) { int t = a; a = b; b = t; }
        if (b > c) { int t = b; b = c; c = t; }

        System.out.println(a + " " + b + " " + c);
    }
}
```

**Puzzle 1.** Sort 4 integers with 5 compare-exchanges.

**Puzzle 2.** Sort 6 integers with 12.