1.1 Your First Program
Why Programming?

Why programming? Need to tell computer what to do

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

Prepackaged software solutions Great, they do exactly what you want

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

well, almost anything

Ada Lovelace

Analytic Engine
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
- ...
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 perfect language
- 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 [grip] 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>

### System

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

### Math Library

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</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>

### Parsing

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer.parseInt()</td>
</tr>
<tr>
<td>Double.parseDouble()</td>
</tr>
</tbody>
</table>

### Flow Control

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>if</td>
<td>else</td>
</tr>
<tr>
<td>for</td>
<td>while</td>
</tr>
</tbody>
</table>

### Boolean

<table>
<thead>
<tr>
<th>Value</th>
<th>Value</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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Symbol</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
<td>}</td>
<td></td>
</tr>
<tr>
<td>(</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>,</td>
<td>;</td>
<td></td>
</tr>
</tbody>
</table>

### Assignment

<table>
<thead>
<tr>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
</tr>
</tbody>
</table>

### Primitive Numeric Types

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Symbol</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>-</td>
<td>*</td>
</tr>
<tr>
<td>/</td>
<td>%</td>
<td>++</td>
</tr>
<tr>
<td>--</td>
<td>&gt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&gt;=</td>
<td>==</td>
</tr>
<tr>
<td>!=</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### String

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
</tr>
<tr>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>length()</td>
</tr>
<tr>
<td>compareTo()</td>
</tr>
<tr>
<td>charAt()</td>
</tr>
<tr>
<td>matches()</td>
</tr>
</tbody>
</table>

### Arrays

<table>
<thead>
<tr>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[i]</td>
</tr>
<tr>
<td>new</td>
</tr>
<tr>
<td>a.length</td>
</tr>
</tbody>
</table>

### Objects

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Keyword</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>
Create the program by typing it into a text editor, and save it as HelloWorld.java

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

HelloWorld.java
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:

```bash
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
```
Dr. Java

http://drjava.org
```java
import java.util.Scanner;

public class UseArgument {
    public static void main(String[] args) {
        System.out.print("Hi, ");
        System.out.print(args[0]);
        System.out.println(" . How are you?");
    }
}
```
Dr. Java

```java
public class UseArgument {
    public static void main(String[] args) {
        System.out.print("Hi, ");
        System.out.print(args[0]);
        System.out.println(". How are you?");
    }
}
```

command-line argument
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>String</td>
<td>sequences of characters</td>
<td>&quot;Hello World&quot;</td>
<td>concatenate</td>
</tr>
<tr>
<td>int</td>
<td>integers</td>
<td>17, 12345</td>
<td>add, subtract, multiply, divide</td>
</tr>
<tr>
<td>double</td>
<td>floating-point numbers</td>
<td>3.1415, 6.022e23</td>
<td>add, subtract, multiply, divide</td>
</tr>
<tr>
<td>boolean</td>
<td>truth values</td>
<td>true, false</td>
<td>and, or, not</td>
</tr>
</tbody>
</table>
Basic Definitions

**Variable**  A name that refers to a value of declared type

**Literal**  Programming language representation of a value

**Assignment statement**  Associates a value with a variable
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></td>
</tr>
<tr>
<td>a = 1234;</td>
<td>1234</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = 99;</td>
<td>1234</td>
<td>99</td>
<td></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

values | sequences of characters
---|---
typical literals | "Hello, " "1 " " * "
operation | concatenate
operator | +

expression | value
---|---
"Hi, " + "Bob" | "Hi, Bob"
"1" + " 2 " + "1" | "1 2 1"
"1234" + " + " + "99" | "1234 + 99"
"1234" + "99" | "123499"

Caveat Meaning of characters depends on context

"1234" + " + " + "99"

white space

character

operator

operator

space

characters
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

Subdivisions of a Ruler
Integers

..., -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, ...
# Integers

**int data type** Useful for 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>
</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>

<table>
<thead>
<tr>
<th>expression</th>
<th>value</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 + 3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>5 - 3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5 * 3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>5 / 3</td>
<td>1</td>
<td>no fractional part</td>
</tr>
<tr>
<td>5 % 3</td>
<td>2</td>
<td>remainder</td>
</tr>
<tr>
<td>1 / 0</td>
<td></td>
<td>run-time error</td>
</tr>
<tr>
<td>3 * 5 - 2</td>
<td>13</td>
<td>* has precedence</td>
</tr>
<tr>
<td>3 + 5 / 2</td>
<td>5</td>
<td>/ has precedence</td>
</tr>
<tr>
<td>3 - 5 - 2</td>
<td>-4</td>
<td>left associative</td>
</tr>
<tr>
<td>(3 - 5) - 2</td>
<td>-4</td>
<td>better style</td>
</tr>
<tr>
<td>3 - (5 - 2)</td>
<td>0</td>
<td>unambiguous</td>
</tr>
</tbody>
</table>
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

REAL NUMBERS

0

FLOATING-POINT NUMBERS
Floating-Point Numbers

double data type Useful in scientific applications

<table>
<thead>
<tr>
<th>values</th>
<th>real numbers (specified by IEEE 754 standard)</th>
</tr>
</thead>
<tbody>
<tr>
<td>typical literals</td>
<td>3.14159 6.022e23 -3.0 2.0 1.4142135623730951</td>
</tr>
<tr>
<td>operations</td>
<td>add subtract multiply divide</td>
</tr>
<tr>
<td>operators</td>
<td>+ - * /</td>
</tr>
</tbody>
</table>

<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 / 2.0</td>
<td>3.01e23</td>
</tr>
<tr>
<td>5.0 / 3.0</td>
<td>1.66666666666666667</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>1.4142135623730951</td>
</tr>
<tr>
<td>Math.sqrt(-1.0)</td>
<td>NaN</td>
</tr>
</tbody>
</table>
public class Math

    double abs(double a)    \hspace{1em} \text{absolute value of } a
    double max(double a, double b) \hspace{1em} \text{maximum of } a \text{ and } b
    double min(double a, double b) \hspace{1em} \text{minimum of } a \text{ and } b

\textbf{Note 1: abs(), max(), and min() are defined also for int, long, and float.}

    double sin(double theta) \hspace{1em} \text{sine function}
    double cos(double theta) \hspace{1em} \text{cosine function}
    double tan(double theta) \hspace{1em} \text{tangent function}

\textbf{Note 2: Angles are expressed in radians. Use toDegrees() and toRadians() to convert.}
\textbf{Note 3: Use asin(), acos(), and atan() for inverse functions.}

    double exp(double a) \hspace{1em} \text{exponential } (e^a)
    double log(double a) \hspace{1em} \text{natural log } (\log_e \text{, or } \ln a)
    double pow(double a, double b) \hspace{1em} \text{raise } a \text{ to the } b\text{th power } (a^b)

    long round(double a) \hspace{1em} \text{round to the nearest integer}
    double random() \hspace{1em} \text{random number in } [0, 1)
    double sqrt(double a) \hspace{1em} \text{square root of } a

    double E \hspace{1em} \text{value of } e \text{ (constant)}
    double PI \hspace{1em} \text{value of } \pi \text{ (constant)}

http://download.oracle.com/javase/6/docs/api/java/lang/Math.html
Quadratic Equation

Ex. Solve quadratic equation $x^2 + bx + c = 0$

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

```java
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

```sh
% java Quadratic -3.0 2.0
2.0
1.0
```

```
x^2 - 3x + 2
```

```
% java Quadratic -1.0 -1.0
1.618033988749895
-0.6180339887498949
```

```
x^2 - x - 1
```

```
% java Quadratic 1.0 1.0
NaN
NaN
```

```
x^2 + x + 1
```

```
% java Quadratic 1.0 hello
defined.
```

```
java.lang.NumberFormatException: hello
```

```
% java Quadratic 1.0
defined.
```

```
java.lang.ArrayIndexOutOfBoundsException
```

Testing Some valid and invalid inputs

```sh
% java Quadratic -3.0 2.0
2.0
1.0
```

```
x^2 - 3x + 2
```

```
% java Quadratic -1.0 -1.0
1.618033988749895
-0.6180339887498949
```

```
x^2 - x - 1
```

```
% java Quadratic 1.0 1.0
NaN
NaN
```

```
x^2 + x + 1
```

```
% java Quadratic 1.0 hello
defined.
```

```
java.lang.NumberFormatException: hello
```

```
% java Quadratic 1.0
defined.
```

```
java.lang.ArrayIndexOutOfBoundsException
```
Booleans
**Booleans**

**boolean data type** Useful to control logic and flow of a program

<table>
<thead>
<tr>
<th>values</th>
<th>true or false</th>
</tr>
</thead>
<tbody>
<tr>
<td>literals</td>
<td>true false</td>
</tr>
<tr>
<td>operations</td>
<td>and or not</td>
</tr>
<tr>
<td>operators</td>
<td>&amp; &amp;</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 | false | true  | false | true  |
| true  | true  | true  | true  | true   | true  |
**Comparisons**  Take two operands of one type (e.g., `int`) and produce a result of type `boolean`

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

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

- Automatic: no loss of precision; or with strings
- Explicit: cast; or method

<table>
<thead>
<tr>
<th>expression</th>
<th>expression type</th>
<th>expression value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;1234&quot; + 99</td>
<td>String</td>
<td>&quot;123499&quot;</td>
</tr>
<tr>
<td>Integer.parseInt(&quot;123&quot;)</td>
<td>int</td>
<td>123</td>
</tr>
<tr>
<td>(int) 2.71828</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>Math.round(2.71828)</td>
<td>long</td>
<td>3</td>
</tr>
<tr>
<td>(int) Math.round(2.71828)</td>
<td>int</td>
<td>3</td>
</tr>
<tr>
<td>(int) Math.round(3.14159)</td>
<td>int</td>
<td>3</td>
</tr>
<tr>
<td>11 * 0.3</td>
<td>double</td>
<td>3.3</td>
</tr>
<tr>
<td>(int) 11 * 0.3</td>
<td>double</td>
<td>3.3</td>
</tr>
<tr>
<td>11 * (int) 0.3</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>(int) (11 * 0.3)</td>
<td>int</td>
<td>3</td>
</tr>
</tbody>
</table>
Ex. Generate a pseudo-random number between 0 and N-1

```java
public 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
```
A **data type** is a set of values and operations on those values
- **String** text processing
- **double, int** mathematical calculation
- **boolean** decision making

**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
- **Ex 1:** in 1996, Ariane 5 rocket exploded after takeoff because of bad type conversion
- **Ex 2:** $i = 0$ in Matlab redefines $\sqrt{-1}$