ALGORITHM DEVELOPMENT; DATA TYPES
Outline

• Algorithm Development
• Data Types
  • Type Conversion
• Comments
Algorithms

- By designing methods, programmers provide actions for objects to perform.
- An algorithm describes a means of performing an action.
- Once an algorithm is defined, expressing it in Java (or in another programming language) usually is easy.
Algorithms

• An algorithm is a set of instructions for solving a problem.
• An algorithm must be expressed completely and precisely.
• Algorithms usually are expressed in English or in pseudocode.
Example: Total Cost of All Items

- Write the number 0 on the whiteboard.
- For each item on the list
  - Add the cost of the item to the number on the whiteboard
  - Replace the number on the whiteboard with the result of this addition.
- Announce that the answer is the number written on the whiteboard.
Type Casting

A *type cast* temporarily changes the value of a variable from the declared type to some other type.

For example,

```java
double distance;
distance = 9.0;
int points;
points = (int)distance;
```

- Illegal without `(int)`
- The value of `(int)distance` is 9,
- The value of `distance`, both before and after the cast, is 9.0.
- Any nonzero value to the right of the decimal point is *truncated* rather than *rounded*. 
Type Conversion

- **Java is strongly typed**
  - Helps protect you from mistakes (aka "bugs")

```java
public class TypeExample0 {
    public static void main(String[] args) {
        int orderTotal = 0;
        double costItem = 29.95;

        orderTotal = costItem * 1.06;
        System.out.println("total=\n" + orderTotal);
    }
}
```

% javac TypeExample0.java
TypeExample0.java:7: possible loss of precision found : double required: int
    orderTotal = costItem * 1.06;
    ^
Type Conversion

- Converting from one type to another:
  - Manually → using a cast
    - A cast is accomplished by putting a type inside ()'s
  - Casting to int drops fractional part
    - Does not round!

```java
public class TypeExample1 {
    public static void main(String [] args) {
        int orderTotal = 0;
        double costItem = 29.95;

        orderTotal = (int) (costItem * 1.06);

        System.out.println("total=\" + orderTotal);
    }
}
```
% java TypeExample1
total=31
Type Conversion

• **Automatic conversion**

  • Numeric types:
    • If no loss of precision → automatic promotion

```java
public class TypeExample2 {
    public static void main(String[] args) {
        double orderTotal = 0.0;
        int costItem = 30;

        orderTotal = costItem * 1.06;

        System.out.println("total=" + orderTotal);
    }
}
```

% java TypeExample2
total=31.8
Type Conversion

- Automatic conversion

- **String** concatenation using the + operator converts numeric types to also be a **String**

```java
public class TypeExample3 {
    public static void main(String[] args) {
        double costItem = 29.95;

        String message = "The widget costs ";
        message = message + costItem;
        message = message + "!";

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

% java TypeExample3
The widget costs 29.95!
Static Methods

• Java has lots of **helper methods** (static methods)
  • Things that take value(s) and return a result
    • e.g. Math functions: `Math.abs(-3.5) → 3.5`
    • e.g. Type conversion: `Integer.parseInt("42") → 42`
      `Double.parseDouble() → double`
    • e.g. Random number generation: `Math.rand()`
  
• Live in some particular Java library class
  • e.g. Math, Integer or Double
• Call using class name followed by dot
Type Conversion Quiz

- Automatic: no loss of precision
  - int will convert to a double if need be
  - double cannot automatically convert to int
- Manual: cast or using a static method

<table>
<thead>
<tr>
<th>expression</th>
<th>resulting type</th>
<th>resulting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(int) 3.14159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Math.round(3.6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 * 3.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 * (int) 3.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(int) 2 * 3.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Type Conversion Quiz

• Automatic: no loss of precision
  • `int` will convert to a `double` if need be
  • `double` cannot automatically convert to `int`
• Manual: cast or using a method

<table>
<thead>
<tr>
<th>expression</th>
<th>resulting type</th>
<th>resulting value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(int) 3.14159</code></td>
<td>int</td>
<td>3</td>
</tr>
<tr>
<td><code>Math.round(3.6)</code></td>
<td>long</td>
<td>4</td>
</tr>
<tr>
<td><code>2 * 3.0</code></td>
<td>double</td>
<td>6.0</td>
</tr>
<tr>
<td><code>2 * (int) 3.0</code></td>
<td>int</td>
<td>6</td>
</tr>
<tr>
<td><code>(int) 2 * 3.0</code></td>
<td>double</td>
<td>6.0</td>
</tr>
</tbody>
</table>
String Conversion Quiz

- **String** conversion, using:
  - `Integer.parseInt()`
  - `Double.parseDouble()`

<table>
<thead>
<tr>
<th>expression</th>
<th>resulting type</th>
<th>resulting value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Integer.parseInt(&quot;30&quot;)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Double.parseDouble(&quot;30&quot;)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Integer.parseInt(&quot;30.1&quot;)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Double.parseDouble(&quot;30.1&quot;)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Integer.parseInt(&quot;$30&quot;)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Double.parseDouble(3.14)</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
String Conversion Quiz

• String conversion, using:
  • `Integer.parseInt()`
  • `Double.parseDouble()`

<table>
<thead>
<tr>
<th>expression</th>
<th>resulting type</th>
<th>resulting value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Integer.parseInt(&quot;30&quot;)</code></td>
<td>int</td>
<td>30</td>
</tr>
<tr>
<td><code>Double.parseDouble(&quot;30&quot;)</code></td>
<td>double</td>
<td>30.0</td>
</tr>
<tr>
<td><code>Integer.parseInt(&quot;30.1&quot;)</code></td>
<td>(runtime error, can't parse as int)</td>
<td></td>
</tr>
<tr>
<td><code>Double.parseDouble(&quot;30.1&quot;)</code></td>
<td>double</td>
<td>30.1</td>
</tr>
<tr>
<td><code>Integer.parseInt(&quot;$30&quot;)</code></td>
<td>(runtime error, can't parse as int)</td>
<td></td>
</tr>
<tr>
<td><code>Double.parseDouble(3.14)</code></td>
<td>(compile error, 3.14 not a String)</td>
<td></td>
</tr>
</tbody>
</table>
String Concatenation Quiz

- + is addition for numeric types
- + is concatenation for `String` type
- numeric types convert to `String` if needed
  - Strings never (automatically) go back to number

<table>
<thead>
<tr>
<th>expression</th>
<th>resulting type</th>
<th>resulting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;testing &quot; + 1 + 2 + 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;3.1&quot; + 4159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;2&quot; + &quot; + &quot; + &quot;3&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 + 2 + 3 + &quot;66&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## String Concatenation Quiz

- `+` is addition for numeric types
- `+` is concatenation for `String` type
- Numeric types convert to `String` if needed
  - Strings never (automatically) go back to number

<table>
<thead>
<tr>
<th>expression</th>
<th>resulting type</th>
<th>resulting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;testing &quot; + 1 + 2 + 3</td>
<td>String</td>
<td>&quot;testing 123&quot;</td>
</tr>
<tr>
<td>&quot;3.1&quot; + 4159</td>
<td>String</td>
<td>&quot;3.14159&quot;</td>
</tr>
<tr>
<td>&quot;2&quot; + &quot; + &quot; + &quot;3&quot;</td>
<td>String</td>
<td>&quot;2 + 3&quot;</td>
</tr>
<tr>
<td>1 + 2 + 3 + &quot;66&quot;</td>
<td>String</td>
<td>&quot;666&quot;</td>
</tr>
</tbody>
</table>
Comments

- The best programs are self-documenting.
  - Clean style
  - Well-chosen names
- Comments are written into a program as needed to explain the program.
  - They are useful to the programmer, but they are ignored by the compiler.
- // comment to end of line
- /*
     multi-line comment
- */
- /**
- * javadoc comment
- */
Summary

- Algorithm Development
- Data Types
  - Type Conversion
- Comments