DATA TYPES AND EXPRESSIONS
Outline

• Variables
  • Naming Conventions

• Data Types
  • Primitive Data Types
    • Review: int, double
    • New: boolean, char
  • The String Class
  • Type Conversion

• Expressions
  • Assignment
  • Mathematical
  • Boolean
Variables

- **Variables** store data such as numbers and letters.
  - Think of them as places to store data.
  - They are implemented as memory locations.
- The data stored in a variable is called its **value**.
  - The value is stored in the memory location.
- Its value can be changed.
- In Java, we always have to declare (define) a variable before we can use it.
  - We need to tell Java what type of data it is and its name.
Variables: Naming Convention

• A variable's name should suggest its use
  • e.g. **taxRate**, **count**, **sum**, etc.
  • that is, it should be meaningful

• Begin with lowercase, uppercase each new word
  • **int** totalWidgets;
  • Called “lower camel case”
Data Types

- A **primitive type** is used for simple, non-decomposable values such as an individual number or individual character.
  - `int`, `double`, and `char` are primitive types.
  - `3.1415` is a value of type `double`
- A **class type** is used for a class of objects and has both data and methods.
  - "Java is fun" is a value of class type `String`

**DEFINITION**

- A **data type** is a set of values and the legal operations on those values.
Variables and Data Types

• Variables
  • **Stores information** your program needs
  • Each has a **unique name**
  • Each has a specific **type**

<table>
<thead>
<tr>
<th>Java built-in type</th>
<th>what it stores</th>
<th>example values</th>
<th>operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>int</strong></td>
<td>integer values</td>
<td>42</td>
<td>add, subtract, multiply, divide, remainder,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1234</td>
<td>compare, increment, decrement</td>
</tr>
<tr>
<td><strong>double</strong></td>
<td>floating-point values</td>
<td>9.95</td>
<td>add, subtract, multiply, divide, compare</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.0e8</td>
<td></td>
</tr>
<tr>
<td><strong>char</strong></td>
<td>characters</td>
<td>'a', 'b', '!', '2'</td>
<td>compare</td>
</tr>
<tr>
<td><strong>String</strong></td>
<td>sequence of characters</td>
<td>&quot;Hello world!&quot;</td>
<td>concatenate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;I love this!&quot;</td>
<td></td>
</tr>
<tr>
<td><strong>boolean</strong></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>
### Data Types: Integers

<table>
<thead>
<tr>
<th>Java built-in type</th>
<th>what it stores</th>
<th>example values</th>
<th>operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integer values</td>
<td>42</td>
<td>add, subtract, multiply, divide, remainder, compare, increment, decrement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1234</td>
<td></td>
</tr>
</tbody>
</table>

```java
public class Integers {
    public static void main(String[] args) {
        int i = 23;
        int j = 5;

        System.out.println("Addition: " + (i + j));
        System.out.println("Subtraction: " + (i - j));
        System.out.println("Multiplication: " + (i * j));
        System.out.println("Division: " + (i / j));
        System.out.println("Remainder: " + (i % j));
        System.out.println("i greater than j? " + (i > j));
        System.out.println("i less than j? " + (i < j));
        System.out.println("i equal to j? " + (i == j));
        System.out.println("i not equal to j? " + (i != j));
        System.out.println("i + 1 " + (i++));
        System.out.println("j - 1 " + (j--));
    }
}
```
Data Types: Floating Point Numbers

<table>
<thead>
<tr>
<th>Java built-in type</th>
<th>what it stores</th>
<th>example values</th>
<th>operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>floating-point values</td>
<td>9.95</td>
<td>add, subtract, multiply, divide, compare</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.0e8</td>
<td></td>
</tr>
</tbody>
</table>

```java
public class Doubles {
    public static void main(String[] args) {
        double i = 23.5;
        double j = 5.5;

        System.out.println("Addition: "+ (i + j));
        System.out.println("Subtraction: "+ (i - j));
        System.out.println("Multiplication: "+ (i * j));
        System.out.println("Division: "+ (i / j));
        System.out.println("i greater than j? "+ (i > j));
        System.out.println("i less than j? "+ (i < j));
        System.out.println("i equal to j? "+ (i == j));
        System.out.println("i not equal to j? "+ (i != j));
    }
}
```
### Data Types: Characters

<table>
<thead>
<tr>
<th>Java built-in type</th>
<th>what it stores</th>
<th>example values</th>
<th>operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>characters</td>
<td>'a', 'b', '!', '2'</td>
<td>compare</td>
</tr>
</tbody>
</table>

```java
public class Chars {
    public static void main(String[] args) {
        char c = 'h';
        char d = 'i';

        System.out.println("Addition: " + (c + d));
        System.out.println("c greater than d? " + (c > d));
        System.out.println("c less than d? " + (c < d));
        System.out.println("c equal to d? " + (c == d));
        System.out.println("c not equal to d? " + (c != d));
    }
}
```
Data Types: Strings (of Characters)

<table>
<thead>
<tr>
<th>Java built-in type</th>
<th>what it stores</th>
<th>example values</th>
<th>operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>sequence of characters</td>
<td>&quot;Hello world!&quot;</td>
<td>concatenate,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;I love this!&quot;</td>
<td>length()</td>
</tr>
</tbody>
</table>

```java
public class Strings {
    public static void main(String[] args) {
        String c = "Hello ";
        String d = "World!";
        char e = '!';

        System.out.println("Concatenation: " + (c + d));

        String result = c + d + e;
        System.out.println(result);

        System.out.println(result.length());
    }
}
```
Concatenating Strings with Other Types

```java
public class StringConcat {

    public static void main(String[] args) {
        String s = ""; // An empty string
        String r = s; // Another empty string

        s = s + (3 + 4);
        r = s + 2;

        System.out.println(s);
        System.out.println(r);

        System.out.println(r + " Trombones");
        System.out.println(r + 3.41512);
        System.out.println(3.41512 + r);
    }
}
```
public class Booleans {

    public static void main(String[] args) {
        boolean t = true;
        boolean f = false;

        System.out.println("And: " + t + " and " + t + " is " + (t && t));
        System.out.println("And: " + t + " and " + f + " is " + (t && f));
        System.out.println("And: " + f + " and " + t + " is " + (f && t));
        System.out.println("And: " + f + " and " + f + " is " + (f && f));
        System.out.println();

        System.out.println("Or: " + t + " or " + t + " is " + (t || t));
        System.out.println("Or: " + t + " or " + f + " is " + (t || f));
        System.out.println("Or: " + f + " or " + t + " is " + (f || t));
        System.out.println("Or: " + f + " or " + f + " is " + (f || f));
        System.out.println();

        System.out.println("Not: !" + t + " is " + !t);
        System.out.println("Not: !" + f + " is " + !f);
    }
}
Booleans

!a → “Is a set to false?”

a && b → “Are both a and b set to true?”

a || b → “Is either a or b (or both) set to true?”

<table>
<thead>
<tr>
<th>logical AND</th>
<th>logical OR</th>
<th>logical NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>a</th>
<th>!a</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
Data Types: Constants

- Sometimes you have a value that should not change
  - e.g. pi, my favorite number, the speed of light

- Values that shouldn’t change are called *constants*.

- Numeric constants can be preceded by a + or – sign (but never use commas)

- Floating-point constants can be written
  - With digits after a decimal point or
  - Using *e notation*.

- Naming convention
  - All upper case, use _ between words
  - `final double` `SPEED_LIGHT = 3.0e8;`
Creating and Initializing a Primitive Variable

byte \( x = 7; \)
Changing the Value of a Primitive Variable

```
byte x = 7;
x = x + 1;
```
Mathematical Expressions: Parentheses and Precedence

- Parentheses can change the order in which arithmetic operations are performed
  - examples:
    - (cost + tax) * discount
    - (cost + (tax * discount))
- Without parentheses, an expression is evaluated according to the rules of precedence.

```
Highest Precedence
First: the unary operators +, -, !, ++, and --
Second: the binary arithmetic operators *, /, and %
Third: the binary arithmetic operators + and -

Lowest Precedence
```
Boolean Expressions: Comparisons

• Given two numbers → return a boolean

<table>
<thead>
<tr>
<th>operator</th>
<th>meaning</th>
<th>true example</th>
<th>false example</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equal</td>
<td>7 == 7</td>
<td>7 == 8</td>
</tr>
<tr>
<td>!=</td>
<td>not equal</td>
<td>7 != 8</td>
<td>7 != 7</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>7 &lt; 8</td>
<td>8 &lt; 7</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal</td>
<td>7 &lt;= 7</td>
<td>8 &lt;= 7</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>8 &gt; 7</td>
<td>7 &gt; 8</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal</td>
<td>8 &gt;= 2</td>
<td>8 &gt;= 10</td>
</tr>
</tbody>
</table>

Is the sum of a, b and c equal to 0?
(a + b + c) == 0

Is grade in the B range?
(grade >= 80.0) && (grade < 90.0)

Is sumItems an even number?
(sumItems % 2) == 0
Leap Year Example

- Years divisible by 4 but not by 100 → leap year
- Years divisible by 400 → leap year

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

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

        // But also leap year if divisible by 400
        isLeapYear = isLeapYear || (year % 400 == 0);

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

- Java is said to be *strongly typed*.
  - You can't assign a floating point value to a variable declared to store an integer.
- Sometimes conversions between numbers are possible:
  ```java
double doubleVariable = 7;
```
- A value of one type can be assigned to a variable of that uses more memory, but not vice versa
  ```java
byte --> short --> int --> long --> float --> double
```
- This is called “automatic type conversion”
Creating and Initializing a Primitive Variable

```java
byte x = 7;
x = x + 1;
short y = 7;
```
You can't put a big cup into a small one

You may know 7 can fit in a byte, but compiler doesn't!

\[
\begin{align*}
\text{byte} & \quad x = 7; \quad x = x + 1; \\
\text{short} & \quad y = 7; \quad x = y;
\end{align*}
\]
Type Casting

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

- For example,
  ```
  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=");
    }
}
```

% 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><code>(int) 3.14159</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Math.round(3.6)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>2 * 3.0</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>2 * (int) 3.0</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>(int) 2 * 3.0</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></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 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>
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

• Variables
  • Naming Conventions
• Data Types
  • Primitive Data Types
    • Review: int, double
    • New: boolean, char
  • The String Class
• Type Conversion
• Expressions
  • Assignment
  • Mathematical
  • Boolean