


CSSE 220 Day 5

More Java Graphics
Shape Classes: Face
Fundamental Data Types, Constants

Check out *BiggestFan* from SVN

Questions?

Outline

- ▶ Quiz questions 1–3 review choosing fields for a class
 - ▶ Sample program using:
 - translate
 - rotate
 - ▶ Design of the Faces project
 - ▶ Work time:
 - Review of fundamental data types
 - Faces
- 

Choosing Fields for Classes

- ▶ Work on Q1–3 alone or in small groups

I'm Your Biggest Fan!

- »» Using Graphics2D's rotate and translate methods.

Making Faces

- » Design and implement a Face class that draws a face of a given size at a given location




Rest of today's class

- ▶ Review of fundamental data types:
 - Work through the slides, quiz, and exercises at your own pace
 - Please ask questions as needed!
 - Start the Faces HW when you are done

Check out *FundamentalDataTypes* from SVN

Data Type Smorgasbord

- ▶ Basic Types and Casts
 - ▶ Big Integers
 - ▶ Constants
 - ▶ Strings and Conversions
 - ▶ Understanding Error Messages
 - ▶ String Input and Output (next class)
- 

Basic Types (again)

Table 1 Primitive Types






Type	Description	Size
 int	The integer type, with range $-2,147,483,648 \dots 2,147,483,647$ (about 2 billion)	4 bytes
byte	The type describing a single byte, with range $-128 \dots 127$	1 byte
short	The short integer type, with range $-32768 \dots 32767$	2 bytes
 long	The long integer type, with range $-9,223,372,036,854,775,808 \dots 9,223,372,036,854,775,807$	8 bytes
 double	The double-precision floating-point type, with a range of about $\pm 10^{308}$ and about 15 significant decimal digits	8 bytes
float	The single-precision floating-point type, with a range of about $\pm 10^{38}$ and about 7 significant decimal digits	4 bytes
 char	The character type, representing code units in the Unicode encoding scheme (see Advanced Topic 4.5)	2 bytes
 boolean	The type with the two truth values <code>false</code> and <code>true</code> (see Chapter 5)	1 bit

Table from Horstmann, Big Java (3e),
John Wiley & Sons, Copyright 2007

Conversions and Casts

- ▶ Consider:

```
int i, j;  
double d, e;  
i = 10;  
d = 20.1;  
e = i; // OK  
j = d; // ERROR!
```

- ▶ Why the difference?
 - Assigning a double to an int can result in information loss (the fractional part)
- ▶ Add a cast to tell Java that we understand there could be a problem here:


```
j = (int) d; // OK
```
- ▶ But what happens to the fractional part of d?
 - It is truncated (lost)

Example

- ▶ Look at `RoundAndRound.java`
 - What does it do?
- ▶ Run it and try some different numbers, like:
 - 1.004
 - 1.005
 - 1.006
 - -1.006
 - 4.35
- ▶ Zoinks! What's up with these, especially the last one?
 - Try changing the `%f` format specifier to `%24.20f`

When Nine Quintillion Isn't Enough

- ▶ **BigInteger** for arbitrary size integer data
 - ▶ **BigDecimal** for arbitrary precision floating point data

 - ▶ We plan to revisit BigInteger later in the course
- 

Constants in Methods

- ▶ Constants let us avoid *Magic Numbers*
 - Hardcoded values within more complex expressions
- ▶ Why bother?
 - ▶ Code becomes more readable, easier to change, and less error-prone!
- ▶ Example:

```
final double relativeEyeOutset = 0.2;  
final double relativeEyeSize = 0.28;  
final double faceRadius = this.diameter / 2.0;  
final double faceCenterX = this.x + faceRadius;  
final double eyeDiameter = relativeEyeSize * this.diameter;
```

...

`final` tells Java to stop us from changing a value (and also gives a “hint” to the compiler that lets it generate more efficient code)

Constants in Classes

- ▶ We've also seen constant fields in classes:
 - `public static final int FRAME_WIDTH = 800;`
- ▶ Why put constants in the class instead of a method?
 1. So they can be used by other classes
 2. So they can be used by multiple methods
 3. So they are easier to find and change

Strings in Java

- ▶ Already looked at some String methods
- ▶ Can also use **+** for string concatenation
- ▶ Quiz question:
 - Look at [StringFoo.java](#)
 - Based on the four uses of **+** in **main()**, can you figure out how Java decides whether to do string concatenation or numeric addition?
 - Decide what the 3 commented-out uses of **+** in **main()** will print, then uncomment them and see if you were right.
 - Do you see why they work as they do?

Converting Strings to Numbers

- ▶ You can convert strings to numbers:
 - **double Double.parseDouble(String n)**
 - **int Integer.parseInt (String n)**
- ▶ Can also convert numbers to strings:
 - **String Double.toString(double d)**
 - **String Integer.toString(int i)**
- ▶ Or maybe easier:
 - **“” + d**
 - **“” + i**

Conversions Gone Awry

- ▶ Go back to `StringFoo.java`
- ▶ Uncomment the last line of `main()`:
 - `StringFoo.helper()` ;
- ▶ Run it
- ▶ What happened?

Reading Exception Stack Traces

The first line will usually give you a hint about what went wrong.

```
<terminated> StringFoo [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Dec 13, 2009 2:37:51 PM)
Exception in thread "main" java.lang.NumberFormatException: For input string: "42.1"
    at java.lang.NumberFormatException.forInputString(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at StringFoo.helper(StringFoo.java:42)
    at StringFoo.main(StringFoo.java:34)
I'm a mess.42
42I'm a mess.
84
I'm a mess.I'm a mess.
```

The first line of *your code* listed will give you a clue where to look.

The error output often appears at the *top* of the Console window (even though the error occurred *after* the output that is displayed). This is because the normal output and the error output are written *concurrently* to two different places, but Eclipse shows them together.

Making Faces

»» Faces HW Work Time

Check out *Faces* from SVN if you haven't already.