CSSE 220 Day 4

Fundamental Data Types, Constants
Console Input, Text Formatting,
Decision Statements and Expressions

Questions?

Data Type Smorgasbord

- Basic Types and Casts
- Big Integers
- Constants
- Strings and Conversions
- Understanding Error Messages
- String Input and Output

- Check out *FundamentalDataTypes* from SVN
- Also check out *TypesAndDecisions* from SVN

Basic Types (again)

Table 1 Primitive Types

Type	Description	Size
`int`	The integer type, with range –2,147,483,648 . . . 2,147,483,647 (about 2 billion)	4 bytes
`byte`	The type describing a single byte, with range –128 . . . 127	1 byte
`short`	The short integer type, with range –32768 . . . 32767	2 bytes
`long`	The long integer type, with range –9,223,372,036,854,775,808 . . . 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 `false` and `true` (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)

Q1-Q2

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`

Q3

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)

Q4 – Q5

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

Q6

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?

Q7

Converting Strings to Numbers

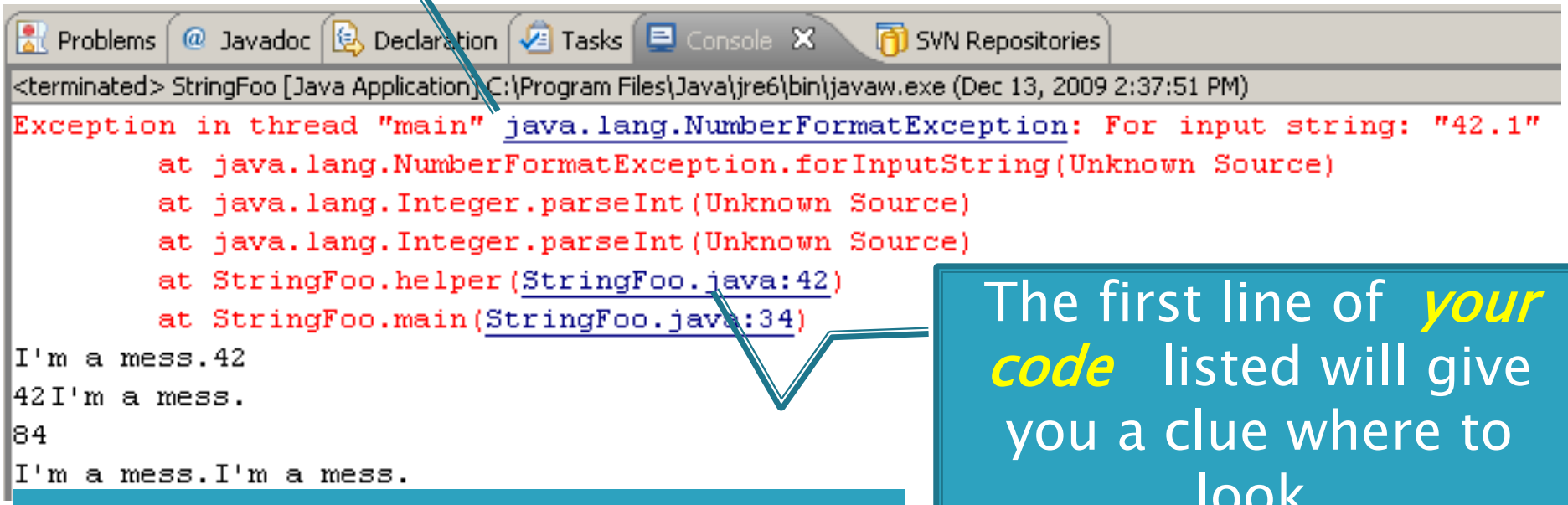
- You can convert strings to numbers:
  - `double Double.parseDouble(String n)`
  - `int Integer.parseInt (String n)`
- Can also convert numbers to strings:
  - `String Double.toString(double d)`
  - `String Integer.toString(int i)`
- Or maybe easier:
  - `"" + d`
  - `"" + i`

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.



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.

Q8 – Q10

`char` Type in Java is Like C's

- In Python:
  - `"This is a string"`
  - `'and so is this'`
- In Java:
  - `"This is a string"`
  - This is a character: `'R'`
  - So is this: `'\n'`
  - `'This is an error'`
  - `'a'` and `"a"` are fundamentally different in Java

Iterating Over Strings in Java

- Can use `charAt(index)`
- Example:

```
String message = "Rose-Hulman";
for (int i=0; i < message.length(); i++) {
    System.out.println(message.charAt(i));
}
```

- `charAt( )` returns a 16-bit `char` value
- Exercise: Work on TODO items in StringsAndChars.java When done, read next slide and do that exercise also.

Reading Console Input with `java.util.Scanner`

- Creating a Scanner object:
  - ```
    Scanner inputScanner =
                        new Scanner(System.in);
    ```
- Defines methods to read from keyboard:
  - `inputScanner.nextInt()`
  - `inputScanner.nextDouble()`
  - `inputScanner.nextLine()`
  - `inputScanner.next()`
- Exercise: Look at ScannerExample.java
  - Add `print`'s to the code to prompt the user for the values to be entered
    ```

Formatting with printf and format

Table 3 Format Types

Code	Type
d	Decimal integer
x	Hexadecimal integer
o	Octal integer
f	Fixed floating-point
e	Exponential floating-point
g	General floating-point (exponential notation used for very large or very small values)
s	String
n	Platform-independent line end

Table 4 Format Flags

Flag	Meaning	Example
-	Left alignment	1.23 followed by spaces
0	Show leading zeroes	001.23
+	Show a plus sign for positive numbers	+1.23
(	Enclose negative numbers in parentheses	(1.23)
,	Show decimal separators	12,300
^	Convert letters to uppercase	1.23E+1

More options than in C. We used a couple in recent examples. Can you find them?

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

Q9 – Q10

Formatting with printf and format

- Printing:
  - `System.out.printf("%5.2f%n", Math.PI);`
- Formatting strings:
  - `String message = String.format("%5.2f%n", Math.PI);`
- Display dialog box messages
  - `JOptionPane.showMessageDialog(null, message);`

If Statements in a Nutshell

```
int letterCount = 0;
int upperCaseCount = 0;
String switchedCase = "";

for (int i = 0; i < message.length(); i++) {
    char nextChar = message.charAt(i);

    if (Character.isLetter(nextChar)) {
        letterCount++;
    }

    if (Character.isUpperCase(nextChar)) {
        upperCaseCount++;
        switchedCase += Character.toLowerCase(nextChar);
    } else if (Character.isLowerCase(nextChar)){
        switchedCase += Character.toUpperCase(nextChar);
    } else {
        switchedCase += nextChar;
    }
}
```

Q11

Comparing Objects

- Exercise: **EmailValidator**
  - Use a **Scanner** object
  - Prompt for user's email address
  - Prompt for it again
  - Compare the two entries and report whether or not they match

- Notice anything strange?

Comparing Objects

The *equals* method is intended to dig inside objects and compare their data in a "sensible" way.

- In Java:
  - `o1 == o2` compares *values*
    - It evaluates to *true* only if their *bits* are the same
      - So for variables of class type, which store *references*, they are == only if they refer to the ***same object*** (same place in memory)
  - There is an `equals` method defined in the `Object` class, that all objects inherit.
    - It behaves the same as == does.
    - But subclasses can, and often do, override the `equals` method to give their own semantics to "equality", using their internal state (their fields). For example:
      - For Strings: `s1.equals(s2)` iff their characters are all ==.
      - `new Integer("0").equals(new Integer("-0"))`

How should you compare the email addresses in the exercise?

Q12 – Q13

If-else statements that choose a value for a variable are common

- ```
  if (amount <= balance) {
    balance -= amount;
  } else {
    balance -= OVERDRAFT_FEE;
  }
  ```

- ```
  if (totalSpent >= 100) {
    discount = 0.15;
  } else {
    discount = 0.0;
  }
  ```

Conditional Operator

- Let us choose between two possible values for an expression
- For example,
  - `balance -= (amount <= balance ? amount : OVERDRAFT_FEE);`
- is equivalent to:

```
if (amount <= balance) {
    balance -= amount;
} else {
    balance -= OVERDRAFT_FEE;
}
```

- Also called **ternary** or **selection** operator (Why?)

Q14

Boolean Essentials—Like C

- Comparison operators: `<`, `<=`, `>`, `>=`, `!=`, `==`
- Comparing objects: `equals()`, `compareTo()`
- Boolean operators:
  - and: `&&`
  - or: `||`
  - not: `!`

Q15

Predicate Methods

- A common pattern in Java:

```
public boolean isFoo() {
    … // return true or false depending on
      // the Foo-ness of this object
}
```

Q16

Test Coverage

- *Black box testing*: testing without regard to internal structure of program
  - For example, user testing
- *White box testing*: writing tests based on knowledge of how code is implemented
  - For example, unit testing
- *Test coverage*: the percentage of the source code executed by all the tests taken together
  - Want high test coverage
  - Low test coverage can happen when we miss branches of switch or if statements

Q17

Switch and Enum

- The next five slides on switch and enumerations are optional. Do the Bid exercise if you're interested. See the book or Google for more info. on switch and enum.

Switch Statements: Choosing Between Several Alternatives

```
char grade = …
int points;
switch (grade) {
case 'A':
   points = 95;
   break;
case 'B':
   points = 85;
   break;
…
default:
   points = 0;
}
```

Can switch on integer, character, or “enumerated constant”

Don’t forget the breaks!

Enumerated Constants

- Specify named sets:

```
public enum Suit {
    CLUBS, SPADES, DIAMONDS, HEARTS
}
```

- Store values from set:

```
Card c = new Card(2, CLUBS);’
```

- Then switch on them:

```
switch (this.suit) {
    case CLUBS:
    case SPADES:
       return "black";
    default:
       return "red";
}
```

Why no break here?

Why no break here?

Optional Exercise: Bids for the Card Game "500"

```
switch (bidSuit) {
    case CLUBS:
    case SPADES:
        return "black";
    default:
        return "red";
}
```

- Implement a class Bid
  - Constructor should take a "trump" Suit and an integer representing a number of "tricks"
  - Test and implement a method, getValue(), that returns the point value of the bid, or 0 if the bid isn't legal. See table for values of the legal bids.

	Spades	Clubs	Diamonds	Hearts	No Trump
6 tricks	40	60	80	100	120
7 tricks	140	160	180	200	220
8 tricks	240	260	280	300	320
9 tricks	340	360	380	400	420
10 tricks	440	460	480	500	520

Suit enum is provided in the repository!

Optional: Predicate Methods

- Live-coding:
  - Test and implement `isValid()` method for Bid
    - JUnit has test methods `assertTrue()` and `assertFalse()` that will be handy
  - Change `getValue()`: return 0 if `isValid()` is false

Optional Exercise

- Study your code for `Bid` and `BidTests`
- Do you have 100% test coverage of the methods?
  - `getValue()`
  - `isValid()`

- Add tests until you have 100% test coverage

For Wednesday…

- The project assigned on Wednesday is a *pair programming* assignment. You MUST find one partner.
  - If you can't find one, we can pair you with someone on Wednesday.
  - Only two people per group

Making Faces

- Faces HW Work Time and HW4

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

Q18 – Q19