CSSE 220 Day 6

Fundamental Data Types, Constants, Console Input, More Text Formatting, Decisions

Check out *TypesAndDecisions* from SVN

Questions?

Today: lots of small topics

- Primitive types, conversions and casts
- Defining constants
- Convert numbers to and from Strings
- Reading input with a Scanner object
- Formatting with format and printf
- Quick review of **`if`** statements
- **`==`** vs. **`equals()`**
- Selection operator, **`? :`**
- **`switch`** and enumerations
- and a partridge in a pear tree

View Grader Comments in Eclipse

- Now posted:
  - HW2: ObjectsAndMethods
    - Comments are in ANGEL
  - HW3: JavadocsAndUnitTesting
    - Right-click and choose Team → Update
    - Look in Task view for:
      - CONSIDER
      - POINTS

Primitive Types (again)

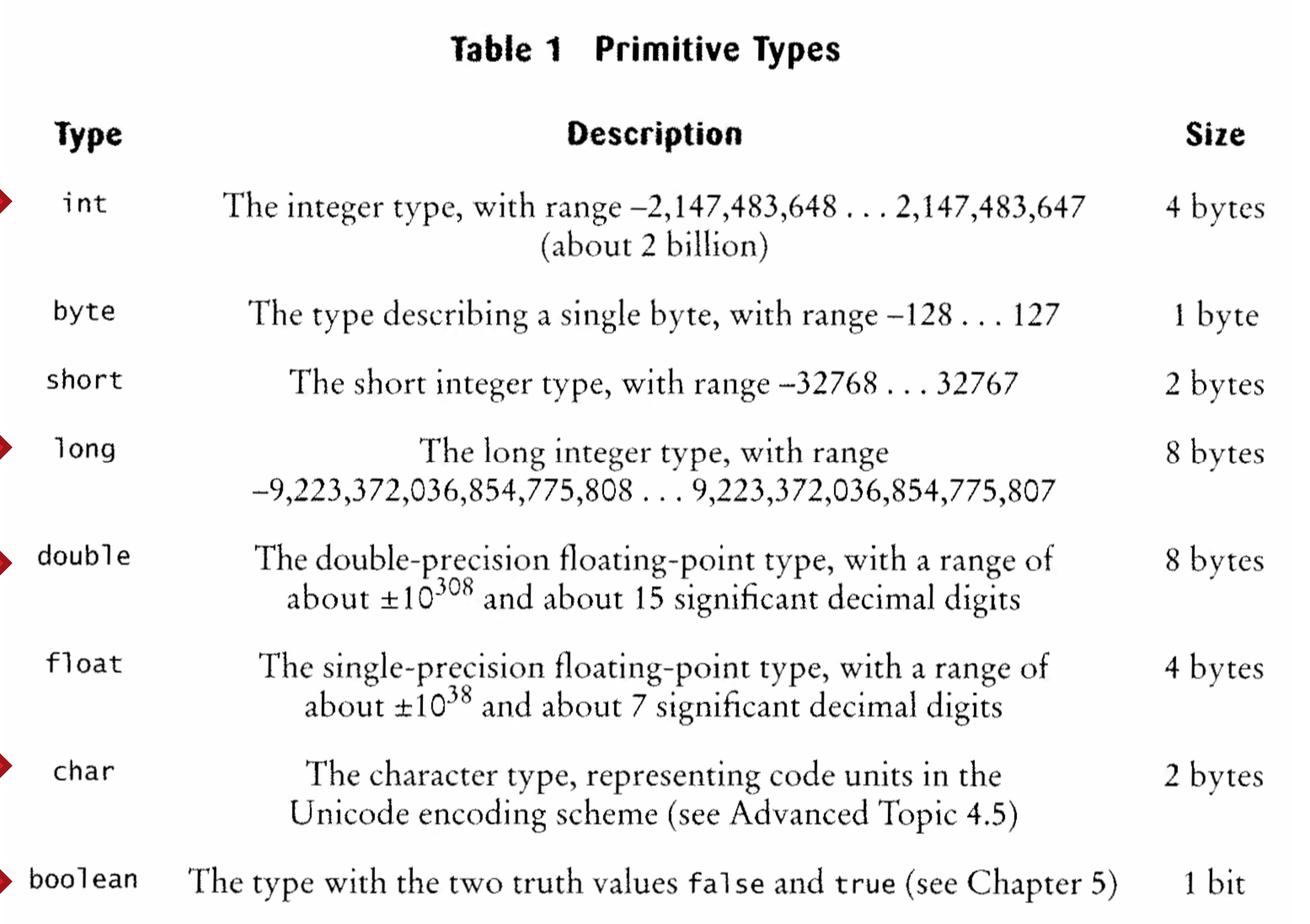

Table 1 Primitive Types

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 = 10;
    double d = 20.1;
    double e = i; // OK
    int j = d; // ERROR!
    ```
- Why the difference?
- Add a cast to tell Java that we understand their could be a problem here:

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

Q1,2

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 the last one?

When Nine Quintillion Isn't Enough

- `BigInteger` for arbitrary size integer data
- `BigDecimal` for arbitrary precision floating point data

- See the Java API documentation for these classes.
- Space and time issues

Constants in Methods

- Constants let us avoid *Magic Numbers*
  - Hardcoded values within more complex expressions
- 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 double eyeRadius = eyeDiameter / 2.0;
double eyeCenterX =
      faceCenterX - relativeEyeOutset * this.diameter;
Ellipse2D.Double eye =
      new Ellipse2D.Double(eyeCenterX - eyeRadius,
                           eyeCenterY - eyeRadius,
                           eyeDiameter, eyeDiameter);
graphics.fill(eye);
```

Q4,5

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?

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?

Q7

Converting Strings to Numbers

- Saw these in Circle of Circles:
  - `double Double.parseDouble(String n)`
  - `int Integer.parseInteger(String n)`
- Can also convert numbers to strings:
  - `String Double.toString(double d)`
  - `String Integer.toString(int i)`
- Or an easier way:
  - `"" + d`
  - `"" + i`

Conversions Gone Awry

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

Reading Exception Traces

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

```
Exception in thread "main"
java.lang.NumberFormatException: For input string:
"42.1"
        at
java.lang.NumberFormatException.forInputString(NumberFor
matException.java:48)
        at java.lang.Integer.parseInt(Integer.java:456)
        at java.lang.Integer.parseInt(Integer.java:497)
        at StringFoo.helper(StringFoo.java:34)
        at StringFoo.main(StringFoo.java:26)
```

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

`char` type in Java is like in C

- In Python:
  - `"This is a string"`
  - `'and so is this'`
- In Java:
  - `"This is a string"`
  - This is a character: `'R'`
  - `'This is an error'`

Iterating Over a String in Java

- Can (usually*) 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

* Unfortunately there are more than 2^{16} (65536) symbols in the known written languages. See Character API docs for the sordid details.

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 `println`’s to the code to prompt the user for the values to be entered

Formatting with printf and format

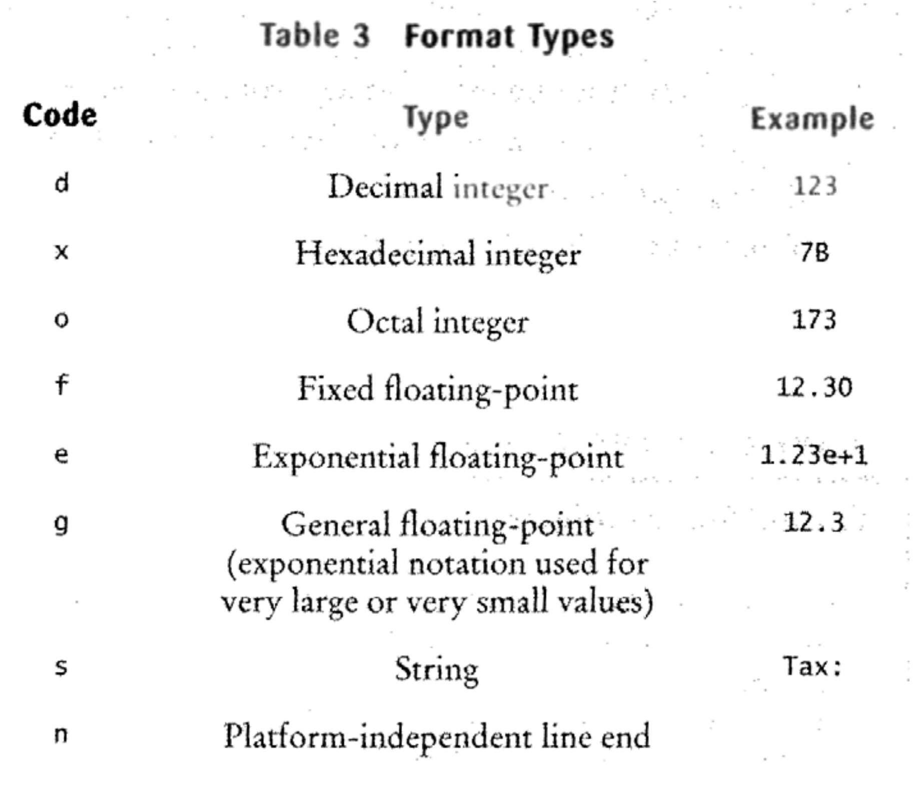

Table 3 Format Types

Code	Type	Example
d	Decimal integer	123
x	Hexadecimal integer	7B
o	Octal integer	173
f	Fixed floating-point	12.30
e	Exponential floating-point	1.23e+1
g	General floating-point (exponential notation used for very large or very small values)	12.3
s	String	Tax:
n	Platform-independent line end	

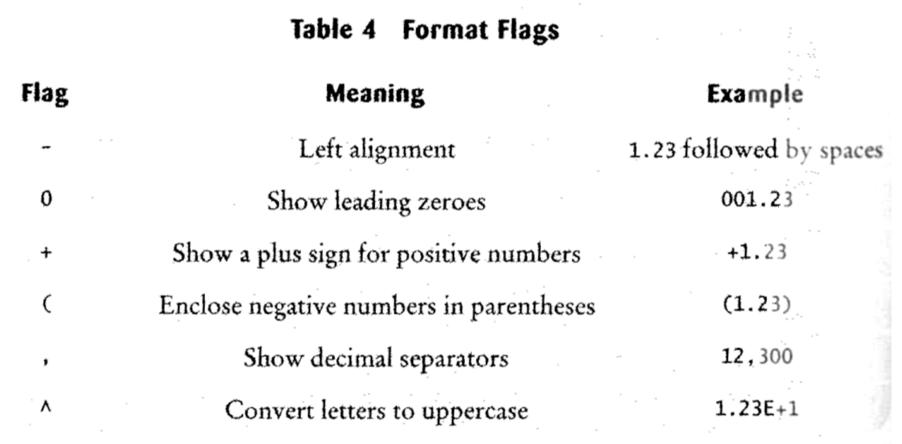

Table 4 Format Flags

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. I used a couple in today's examples. Can you find them?

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

Formatting with `printf` and `format`

- Printing:
  - `System.out.printf("%5.2f%n", Math.PI)`
- Formatting strings without printing:

  -
    ```
    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;
    }
}
```

Comparing Objects

- Exercise: EmailValidator.java
  - 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

- In Java:
  - `oneObject == otherObject` compares *references*
  - `oneObject.equals(oneObject)` compares *objects*

- Remember: variables of object types store **reference** values

- How should you compare the email addresses in the exercise?

Q8

Statement vs. Expressions

- Statements: used only for their *side effects*
  - Changes they make to stored values or control flow
- Expressions: calculate values

- Many statements contain expressions:

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

Selection Operator

- Let’s us choose between two possible values for an expression
- Example:

  -
    ```
    balance = balance -
               (amount <= balance) ?
                   amount : OVERDRAFT_FEE
    ```
- Also called the “ternary” operator (Why?)

Q9

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!

Q10

Enumerated Constants

- Let us specify named sets of values:

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

- Then **switch** on them:

```
public String colorOf(Suit s) {
  switch (s) {
    case CLUBS:
    case SPADES:
      return “black”;
    default:
      return “red”;
  }
}
```

Another Enumeration Example

```
public class TryEnums {

      public enum Day {
            SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
             FRIDAY, SATURDAY
      }

      public static void main(String[] args) {
            Day d = Day.SUNDAY;
            System.out.println(d);
            System.out.println(Day.MONDAY.ordinal());
            for (Day d2 : Day.values())
                  System.out.print(d2 + " ");
            System.out.println();
      }
}
```

Output:

```
SUNDAY
1
SUNDAY MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SATURDAY
```

Boolean Essentials—Like C

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

Q11

Predicate Methods

- A common pattern in Java:

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

- We tested and implemented isWholeNumber in the Fraction class

Q12

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

Q6

Exercise

- If, by some miracle, we still have time left:

  Begin to create a `CubicPlot` class as described in HW6