

CSSE 220 Day 6

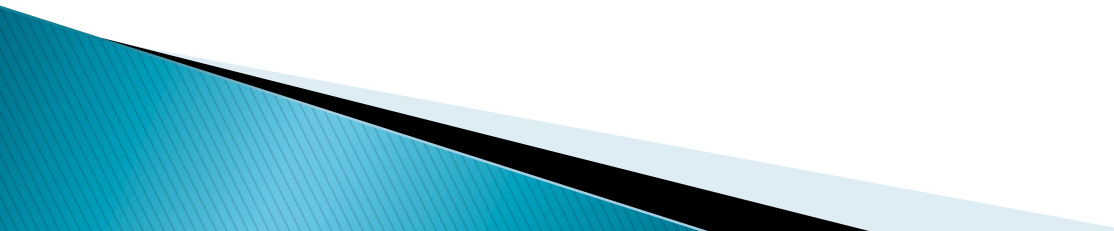
Console Input, Text Formatting,
Decision Statements and Expressions

Check out *Decisions* from SVN

Questions?

Outline

- ▶ String Input and Output
 - ▶ Quick review of **if** statements
 - ▶ **==** vs. **equals()**
 - ▶ Selection operator, **?** **:**

 - ▶ Optional: **switch** and enumerations
- 

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

Table 4 Format Flags

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

More options than in C.
I used a couple in
today's examples.
Can you find them?

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);
```

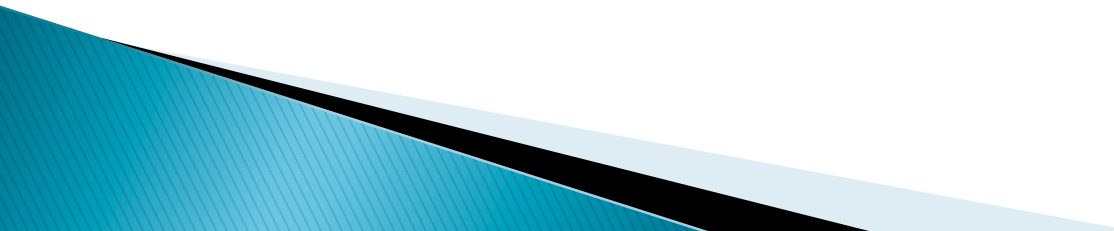
```
    if (Character.isLetter(nextChar)) {  
        letterCount++;  
    }
```

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

```
}
```

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?

Q3 - Q4

Statement vs. Expressions

- ▶ Statements: used only for their *side effects*
 - Changes they make to stored values or control flow
 - Printed output
 - Drawing
- ▶ Expressions: calculate values
- ▶ Many statements contain **expressions**:
 - ```
if (amount <= balance) {
 balance -= amount;
} else {
 balance -= OVERDRAFT_FEE;
}
```

# 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?)

# Boolean Essentials—Like C

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

# Predicate Methods

- ▶ A common pattern in Java:

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

# 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



# 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

# Work Time

»» Hand in quiz.

Work on Homework 6:

Grade and CubicPlot

**These are challenging  
exercises!**

If you do not make a lot of progress during today's class, be sure to work on it some more today! People who put this one off until Friday or Saturday may be in trouble!