

# CSSE 220 Day 7

## Decision Statements and Expressions

Check out *Decisions* from SVN

Questions?


# Today

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

# 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

- ▶ In Java:
  - `o1 == o2` compares *values*
  - `o1.equals(o2)` compares *objects*
- ▶ Remember: variables of class type store **reference values**
- ▶ How should you compare the email addresses in the exercise?

# 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

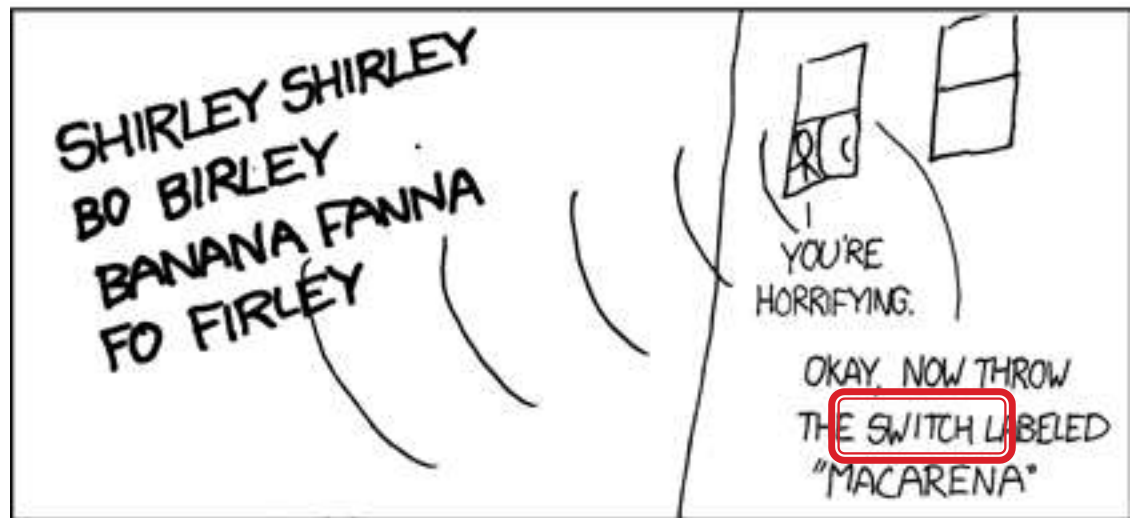
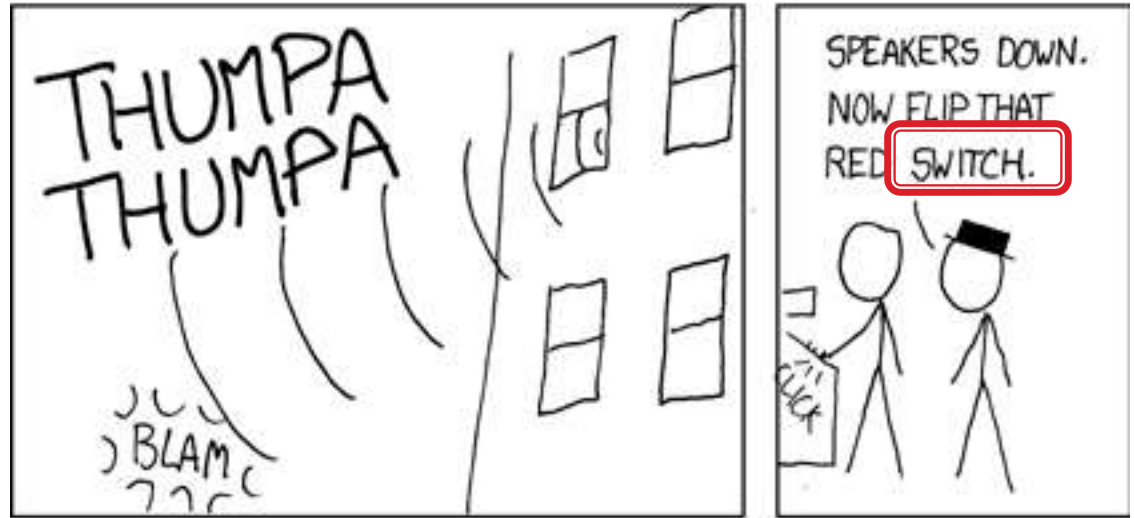
- ▶ Let's us choose between two possible values for an expression
- ▶ Example:
  - **balance = balance - (amount <= balance) ? amount : OVERDRAFT\_FEE**
- ▶ Also called “ternary” operator (Why?)



# Bass (1 / 2)



# Bass (2/2)



# 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 (c.getSuit()) {  
    case CLUBS:  
    case SPADES:  
        return "black";  
    default:  
        return "red";  
}
```

Why no break here?

Why no break here?

# 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!

# 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  
}
```

- ▶ Live-coding:


- Tests and implement `isValid()` method for Bid
  - JUnit has test methods `assertTrue()` and `assertFalse()` that will be handy
- Change `value()` to return 0 if `isValid()` is false

# 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



# 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

- »» Finish CubicPlot from last time
- Other homework problems if time permits