

Object-Oriented Design

CS 110

Review

- Objects
- Object Oriented Programming
- Fields
- Methods
- Dot Notation
- PImage Object
- String Object
- Defining Custom Objects using the class keyword
- The Ball Class
- Bounce with the Ball Class

Signature Polymorphism

poly = many, *morph* = form

- It is possible to define multiple functions with the same name, but different signatures.

- A *function signature* is defined as

- The function name, and
 - The order of variable types passed to the function

- Consider the built-in `color()` function ...

```
color(gray)
```

```
color(gray, alpha)
```

```
color(value1, value2, value3)
```

```
color(value1, value2, value3, alpha)
```

```
...
```

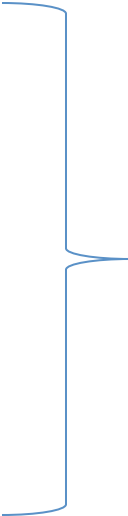
Signature Polymorphism

```
void draw() { }

void mousePressed() {
    int i;
    i = 10;
    i = increment(i, 2);
    //i = increment(i);
    println(i);
}

// increment a variable
int increment(int j, int delta) {
    j = j + delta;
    return j;
}

int increment(int k) {
    k = increment(k, 1);
    return k;
}
```



In this case it is
said that the
increment function
is *overloaded*

Our Toolkit

- Graphics
 - lines, shapes, images, text, color, ...
- Data of Various Types
 - Numbers (with and without decimal places)
 - Booleans (true, false)
 - Color (two color models)
 - Characters and Strings
- Variables
 - Hold/name any type of data values
- Arrays
- Operators
 - Mathematical (+, *, ++, %, ...)
 - Relational (<, >=, !=, ==, ...)
 - Logical (&&, ||, !)

Our Toolkit (Continued)

- Functions
 - Mathematical, Graphical, Utility, ...
 - Of our own design
- Expressions
 - Combine of data, variables, operators, functions
- Conditionals
 - if-statements
 - switch-statement
- Iterations
 - while-loop
 - for-loop
- Data Structures
 - Arrays
 - Functions that manipulate arrays
- Objects

Top-Down Design

- At first blush, solving a hard problem can seem daunting
 - Create a clone of Adobe Photoshop
 - Create a new web browser
- A common technique for solving complex problems is called **Top-Down Design**
 - a.k.a. "Step-wise Refinement"
 1. Define a sequence of steps to solve a given problem at the highest, most abstract level.
 2. Recursively, list a sequence of sub-steps to solve each higher-level step
 3. Repeat until the sub-problem is "easy enough" to solve directly

Top-Down Design - Advantages

- Promotes Organization
 - Your code is naturally organized, and easy to understand
 - Avoids the "spaghetti code" syndrome
- Simplifies the Problem
 - The larger complex problem reduces to several smaller, more simple problems
- Promotes Reuse
 - Several sub-problem solutions may be reusable by multiple parts of your program
 - Some sub-problems have existing solutions implemented
- Enables Shared Development
 - Multiple people can work on different parts of the problem at the same time

Top-Down Design - Example

Have Dinner

1. Cook Food
2. Set Table
3. Serve Food
4. Eat Food
5. Clean Up

Top-Down Design - Example

Have Dinner

1. Cook Food
 1. Boil Noodles
 2. Stir-fry Veggies
 3. Mix together
2. Set Table
3. Serve Food
4. Eat Food
5. Clean Up

Top-Down Design - Example

Have Dinner

1. Cook Food

1. Boil Noodles

1. Boil water
2. Pour in dry noodles
3. Let cook
4. Strain noodles

2. Stir-fry Veggies

3. Mix

2. Set Table

3. Serve Food

4. Eat Food

5. Clean Up