More Class Design

Midterm Format
- closed book, no notes, no calculators
- must bring ID, put in front of you face up so we can see picture and name
- 6:30 Monday 26, FSC 1005
  - exam starts at 6:30, please arrive before that
  - you will have 60 minutes to do the exam
  - do not turn over or open exam until we say to begin

Midterm Advice
- good to read book, but definitely don't stop there!
- best thing to do: practice programming
  - try exercises in Big Java
  - solutions for some practice problems now posted in new Handy Links folder on WebCT Vista
  - and/or invent your own problems!
  - do a mix of programming on the computer, and on paper
  - you will only have paper for the exam

Midterm Coverage
- reading: chapters 1-4
- lectures: weeks 0-4
- through this Friday 2/5
- topics: intro, hardware background, programming languages, comments, identifiers, whitespace, errors, variables, primitive data types, assignment, casting, constants, objects, classes, strings, input, class design
- assignments: assignment 1

Recap: UML Visual Syntax
- for public, - for private
- fields above, methods below

Recap: UML Design for Point
- preliminary design for 2D point class

Recap: Control Flow Between Modules
- Two weeks ago it was easy to understand control flow: order in which statements are executed
- march down line by line through file
- Now consider control flow between modules

Formal vs. Actual Parameters
- formal parameter: in declaration of class
- actual parameter: passed in when method is called
- if parameter is primitive type
  - call by value: value of actual parameter copied into formal parameter when method is called
  - changes made to formal parameter inside method body will not be reflected in actual parameter value outside of method
- if parameter is object: covered later

Scope
- Fields of class are have class scope: accessible to any class member
- in Die and Point class implementation, fields accessed by all class methods
- Parameters of method and any variables declared within body of method have local scope: accessible only to that method
- not to any other part of your code
- In general, scope of a variable is block of code within which it is declared
  - block of code is defined by braces { }

Comments
- Conventions
  - explain what classes and methods do
  - plus anywhere that you've done something nonobvious
  - often better to say why than what
  - not useful
  - int wishes = 3; // set wishes to 3
  - useful
  - int wishes = 3; // follow fairy tale convention

javadoc Comments
- Specific format for method and class header comments
  - running javadoc program will automatically generate HTML documentation
- Rules
  - /** to start, first sentence used for method summary
  - @param tag for parameter name and explanation
  - @return tag for return value explanation
  - @see tag, @version
  - */ to end
- Running
  - javadoc Die.java
  - javadoc * .java
**javadoc Method Comment Example**

```java
/**
 * Sets the die shape, thus the range of values it can roll.
 * @param numSides the number of sides of the die
 */
public void setSides(int numSides) {
    sides = numSides;
}
```

**javadoc Class Comment Example**

```java
/**
 * Die: simulate rolling a die
 * @author: CPSC 111, Section 206, Spring 05-06
 * @version: Jan 31, 2006
 *
 * This is the final Die code. We started on Jan 24, tested and improved in on Jan 26, and did a final cleanup pass on Jan 31.
 */
```

**Cleanup Pass**

- Would we hand in our code as it stands?
- Good use of whitespace?
- Well commented?
- Every class, method, parameter, return value clear, descriptive variable naming conventions?
- Constants vs. variables or magic numbers?
- Fields initialized?
- Good structure?
- Follows specification?
- Ideal: do as you go
  - Commenting first is a great idea!
- Acceptable: clean up before declaring victory

**Key Topic Summary**

- Generalizing from something concrete
  - Fancy name: abstraction
- Hiding the ugly guts from the outside
  - Fancy name: encapsulation
- Not letting one part ruin the other part
  - Fancy name: modularity
- Breaking down a problem
  - Fancy name: functional decomposition