More Class Design

Lecture 13, Wed Feb 3 2010

borrowing from slides by Paul Carter and Steve Wolfman

http://www.cs.ubc.ca/~tmm/courses/111-10
Reminder: Lab Schedule Change

- no labs next week Feb 8-12
- TAs will hold office hours in labs during Monday lab times to answer pre-midterm questions
  - Mon Feb 8 11am - 3pm ICICS 008
- labs resume after break
  - staggered to ensure that even Monday morning labs have seen material in previous week's lecture
Recap: Refined UML Design for Point

- refined design for 2D point class

<table>
<thead>
<tr>
<th>Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>- x: double</td>
</tr>
<tr>
<td>- y: double</td>
</tr>
<tr>
<td>+ Classname(inX: double, inY: double)</td>
</tr>
<tr>
<td>+ distanceBetween(Point otherPoint): double</td>
</tr>
<tr>
<td>+ getX(): double</td>
</tr>
<tr>
<td>+ getY(): double</td>
</tr>
<tr>
<td>+ distanceToOrigin(): double</td>
</tr>
</tbody>
</table>
Recap: Point Class Ideas

- continued testing after first victory
  - negative vs positive values
  - double vs integer values
  - check distance between same point is zero
- avoided duplication of code
  - for distanceToOrigin we created new Point representing origin, and used distanceBetween
  - versus cut/paste + tweaking
- cannot initialize fields by having parameter names in constructor match field names
Formal vs. Actual Parameters

- **formal** parameter: in declaration of class
- **actual** parameter: passed in when method is called
  - variable names may or may not match
- 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 `{ }`
Point Final Testing/Refinement

- check questions we noted in comments along the way
- clean up and comment
Commenting Code

- Conventions
  - explain what classes and methods do
  - plus anywhere that you've done something nonobvious
    - often better to say why than what
      - not useful
        ```java
        int wishes = 3; // set wishes to 3
        ```
      - useful
        ```java
        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
  - other tags: @author, @version
  - */ to end

- Running
  - % javadoc Die.java
  - % javadoc *.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;
}

/**
Gets the number of sides of the die.
@return the number of sides of the die
*/
public int getSides() {
    return sides;
}
/** 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