Class Design

Lecture 9, Mon Jan 24 2010

borrowing from slides by Paul Carter and Wolfgang Heidrich

http://www.cs.ubc.ca/~tmm/courses/111-10
News

- If you have a midterm conflict with first midterm, let me know by end of day **today** at the latest
- Mon 2/8 6:30-8pm
Reading Assignments

- Chapter 3
Recap: References vs Values

- You copy a CD for your friend. Her dog chews it up. Does that affect your CD?
  - no: different values
  - like primitive types

- You and your friend start eating a slice of cake on one shared plate. You get up to make a cup of tea. Her dog jumps on the table and eats the cake. Does that affect your half of the dessert?
  - yes: both forks reference the same plate
  - like objects
Recap: Abstraction

- **Abstraction**: process whereby we
  - hide non-essential details
  - provide a view that is relevant
- Often want different layers of abstraction depending on what is relevant
Recap: Encapsulation

- **Encapsulation**: process whereby
  - inner workings made inaccessible to protect them and maintain their integrity
  - operations can be performed by user only through well-defined interface.
- aka **information hiding**

- **Cell phone example**
  - inner workings encapsulated in handset
    - cell phone users can’t get at them
  - intuitive interface makes using them easy
    - without understanding how they actually work
Recap: Designing Die Class

- Blueprint for constructing objects of type Die
- Think of manufacturing airplanes or dresses or whatever
  - design one blueprint or pattern
  - manufacture many instances from it
- Consider two viewpoints
  - client programmer: wants to use Die object in a program
  - designer: creator of Die class
Recap: Designer

- Decide on inner workings
  - implementation of class

- Objects need state
  - attributes that distinguish one instance from another
  - many names for these
    - state variables
    - fields
    - attributes
    - data members
  - what fields should we create for Die?
Implementing Die

/**
 * Provides a simple model of a die
 * (as in pair of dice).
 */

public class Die
{
}

Random Numbers

- **Random** class in `java.util` package
  - `public Random()`
    - Constructor
  - `public float nextFloat()`
    - Returns random number between 0.0 (inclusive) and 1.0 (exclusive)
  - `public int nextInt()`
    - Returns random integer ranging over all possible int values
  - `public int nextInt( int num )`
    - Returns random integer in range 0 to (num-1)
Implementing Die

/**
 * Provides a simple model of a die
 * (as in pair of dice).
 */

public class Die
{

}
**return Statement**

- Use the `return` statement to specify the return value when implementing a method:

  ```java
  int addTwoInts (int a, int b) {
    return a+b;
  }
  ```

- Syntax: `return expression`;

- The method stops executing at that point and “returns” to caller.
/**
 * Provides a simple model of a die
 * (as in pair of dice).
 */

public class Die {
}
Information Hiding

- Hide fields from client programmer
  - maintain their integrity
  - allow us flexibility to change them without affecting code written by client programmer

- Parnas' Law:
  - "Only what is hidden can by changed without risk."
Public vs Private

- **public** keyword indicates that something *can* be referenced from outside object
  - can be seen/used by client programmer

- **private** keyword indicates that something *cannot* be referenced from outside object
  - cannot be seen/used by client programmer

Let’s fill in public/private for **Die** class
Public vs. Private Example

```java
public class Die {
    ...
    public int roll()
        ...
        ...
    private void cheat(int nextRoll)
        ...
}
```
Public vs. Private Example

Die myDie = new Die();

int result = myDie.roll(); // OK
myDie.cheat(6); //not allowed!
Implementing Die

/**
   * Provides a simple model of a die
   * (as in pair of dice).
   */
public class Die
{

}
Trying It Out!

- **Die** class has no main method.
- Best is to write another class that instantiates some objects of your new class and tries them out.
  - Sometimes called a “tester” or “testbench”
Implementing RollDice

```java
public class RollDice {

    public static void main ( String [] args) {
    }
}
```