Announcements

- 1st Midterm Exam: Wed, Oct 7, 6:30 pm
- Locations:
  - Henning 200, 201, 202
  - Woodward 4, 5
- Room allocation by last name
  - List of names vs rooms is online (WebCT and section web page)
- Material: Chapters 1-4

Reading Assignments

- For this week, read
  - Read rest of Ch. 4
  - Revisit earlier chapters as we talk about class development

- Reading for next week:
  - Edition 2: Ch. 6.1-6.4
  - Edition 3: Ch. 5.1-5.4

Recap: Creating Classes - 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 hand set
  - cell phone users can't get at them
  - intuitive interface makes using them easy
  - without understanding how they actually work

Recap: Information Hiding

- Hide internal details from user of object.
  - maintains integrity of object
  - allow us flexibility to change them without affecting users

- Parnas’ Law:
  - “Only what is hidden can by changed without risk.”

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: Class Implementation Concepts
- Public/private methods:
  - `public` keyword indicates that something can be referenced from outside object
  - `private` keyword indicates that something cannot be referenced from outside object
- Fields:
  - Variables within an object, that are accessible by all the methods the object belongs to
- Return values:
  - `return` keyword used to specify results of methods

Using vs. Designing Classes
<table>
<thead>
<tr>
<th>Using a Class</th>
<th>Designing a Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read API.</td>
<td>Write API.</td>
</tr>
<tr>
<td>Get and use objects.</td>
<td>Provide blueprint/pattern for objects.</td>
</tr>
<tr>
<td>Instantiate objects with new ClassName(...)</td>
<td>Define constructor ClassName(...)</td>
</tr>
<tr>
<td>Call methods to get things done.</td>
<td>Implement methods. Say how to actually do things.</td>
</tr>
<tr>
<td>Access public things only.</td>
<td>Decide what is public and what is private.</td>
</tr>
</tbody>
</table>

Objectives for Today
- Design, implement, and try out another class
- Gain a lower-level, operational view of how method calls work.
- Understand class and local scope of declarations.

Continuing w/ Die Example
Adding a Private Method
- Expand the Die class set the value of the next roll (i.e. cheat):
  ```cpp
def cheat(int nextRoll) {
  
}
```
- Also:
  - All fields should be private as well...

Mileage Computer
- Design a class for a trip computer in a car that computes the gas mileage
  - Periodically, sensors tell the computer how much gas was used and how much distance was covered in the last measurement interval
  - The measurements can be reset
  - We can query the computer for the gas mileage in the last measurement interval, or averaged since the last reset.

Control Flow Between Modules
- Object-Oriented View: Instantiate objects and call their methods.
- Procedural View:
  - There is control flow – order in which statements are executed.
  - In a sequence of statements, march down line-by-line through file.
  - Now consider control flow between modules
  - Sometimes helpful to mix views…
**Control Flow Between Modules**

**Client Code**
```java
double mpg;
MileageComputer foo =
    new MileageComputer();
foo.addData(1.0,0.1);
foo.addData(1.0,0.1);
mpg = foo.avgMileage();
foo.addData(1.0,0.1);
foo.reset();
```

**Class Methods**
```java
public MileageComputer() {
    …
}
public void addData(..) {
    …
}
public double avgMileage(..) {
    …
}
public void reset() {
    …
}
```

**Types of Methods**
- **Accessor:** Method that provides information about (access to) internal state of an object
  - E.g.: `MileageComputer.avgMileage()`
- **Mutator:** Methods that change (mutate) the internal state of an object
  - E.g.: `MileageComputer.addData()`, `MileageComputer.reset()`

**Data Flow Between Modules**

**Client Code**
```java
foo.addData(1.0,0.1);
```

**Class Methods**
```java
public void addData(double miles, double gas) {
    totalMiles = totalMiles+miles;
    totalGas = totalGas+gas;
}
```

**Data Flow Between Modules**

Values are copied into parameters, as if these assignments were there.

**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
  - Java uses call by value:
    - Value of actual parameter copied into formal parameter when method is called
    - For primitive types, changing formal parameter inside method body does not change actual parameter value outside
  - What if parameter is an object?
Introduction to Scope
- Private fields and methods of class have **class scope**: accessible anywhere in class
- 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 { }

Java Shorthand Assignment Operators
- Very often we update the value of a variable using its current value:
  - `totalGas = totalGas + gas;`
  - `totalMiles = totalMiles + miles;`
- Java provides a special shorthand for this:
  - `totalGas += gas;`
  - `totalMiles += miles;`

Other Assignment Operators
- Syntax: `variable op= expression`;
  - is exactly the same as `variable = variable op expression;`
- Example:
  - `int a = 3;`
  - `int b = 5;`
  - `a *= b;`
  - // What’s the value of a? Of b?

Scope Example
```java
public class MileageComputer {
    __
    public void addData(double miles,double gas) {
        totalMiles = totalMiles+miles;
        totalGas = totalGas+gas;
    }
    __
    private double totalMiles;
    private double totalGas;
}
```

The += Operator
- Syntax: `variable += expression`;
  - is exactly the same as `variable = variable + expression;`
- Example:
  - `int a = 3;`
  - `int b = 5;`
  - `a += b;`
  - // What’s the value of a? Of b?

Other Assignment Operators
- What were the Java operators again?
  - See Appendix F.
- They pretty much all work:
  - `tigers += 5;` // like tigers=tigers+5;
  - `lions -= 3;` // like lions=lions-3;
  - `bunnies *= 2;` // like bunnies=bunnies*2;
  - `dinos /= 100;` // like dinos=dinos/100;
  - `bears %= 100;` // like bears=bears%100;`
Increment and Decrement

- Often want to increment or decrement by 1
  - obvious way to increment
    - `count = count + 1;`
  - assignment statement breakdown
    - retrieve value stored with variable `count`
    - add 1 to that value
    - store new sum back into same variable `count`
  - obvious way to decrement
    - `count = count - 1;`

Super Shorthand Increment/Decrement

- You could use assignment operators:
  - `count += 1;`
  - `count -= 1;`
- But Java has an even more compact shorthand:
  - `count++; // like count = count + 1;`
  - `count--; // like count = count - 1;`
  - `++count; // like count = count + 1;`
  - `--count; // like count = count - 1;`

Shorthand Assignment Operators

- what value ends up assigned to `total`?
  
  ```java
  int total = 5;
  int current = 4;
  total *= current + 3;
  ```

- remember that Java evaluates right before left of `=`
  - first right side is evaluated: result is 7
  - `total *= 7;`
  - `total = total * 7;`
  - `total = 5 * 7;`
  - `total = 35;`

Assignments as Expressions

- So far, you’ve learned assignments as statements – they modify a variable.
  - `x = 3;` // sets `x` to the value 3
- So far, you’ve learned operators as expressions – they evaluate to some value.
  - `x + 3` // computes sum of `x` and 3
- But in Java, you can use an assignment as **both** an assignment and an operator!

Assignments as Expressions

- This can get very twisted!
  - Rarely used
  - Generally considered bad style
- But one common usage:
  - `a = b = c = 0;` assigns a value to a bunch of variables.
  - Assignment operators are **right** associative
  - `a = (b = (c = 0));`