College of Information Studies

University of Maryland Hornbake Library Building College Park, MD 20742-4345

Object Oriented Programming

Week 5

INFM 603

The Key Ideas

- Structured Programming
- Modular Programming
- Data Structures
- ➢ Object-Oriented Programming

Key Ideas

- Protect the programmer from themselves
  - Model actions and attributes together
- Object
  - Encapsulation of methods and data structures
- Class
  - “Blueprint” for an object
  - Must be “instantiated” using a “constructor”

Some Conventions

- CapsInitial skyline is used for a a **class**
- lowerCaseInitial skyline is used for a
  - Variable (if not followed by parameters)
  - Method (if followed by parameters)
- An **object** can be assigned to a variable

Object Instantiation

- **var n = new Array(5);**
  - Creates an Array object using the Array class constructor (and specifying 5 elements)
- **var student = new Student(13205, “George”);**
  - Creates a Student object using the Student class constructor (and specifying the student id and name)
  - Note that the variable name need not be different from (or the same as) the class name

Formalizing Object Interfaces

- **status = student.setHeightInches(74);**
  - Invokes the **setHeightInches()** method for the object that is stored in the variable **student** and passes it **74** as a parameter; **status**=true indicates success
- **feet = student.getHeightFeet();**
  - Invokes the **getHeightFeet()** method for the object that is stored in the variable **student** and then sets the variable **feet** to hold that result (in this case, 6); feet<0 indicates failure

Class Definition (private variable)

```
function Student(studentID, name) {
  var totalInches = -1;   // private variable

  // private method
  function inchesToFeet(i) {
    return Math.floor(i/12);
  }

  // public methods
  this.setHeightInches = function(n) {
    if ((n>0) && (n<100)) {
      totalInches = n;
      return true;
    } else {
      return false;
    }
  }
  this.getHeightFeet = function() {
    if (totalInches>0) {
      return inchesToFeet(totalInches);
    } else {
      return -1;
    }
  }
}
```

```
var student = new Student(13205, "George");
alert(student.setHeightInches(74));
alert(student.getHeightFeet());
alert(student.totalInches);
```

Class Definition (public variable)

```
function Student(studentID, name) {
  this.totalInches = -1;  // public variable

  // private method
  function inchesToFeet(i) {
    return Math.floor(i/12);
  }

  // public methods
  this.setHeightInches = function(n) {
    if ((n>0) && (n<100)) {
      this.totalInches = n;
      return true;
    } else {
      return false;
    }
  }
  this.getHeightFeet = function() {
    if (this.totalInches>0) {
      return inchesToFeet(this.totalInches);
    } else {
      return -1;
    }
  }
}
```

```
var student = new Student(13205, "George");
alert(student.setHeightInches(74));
alert(student.getHeightFeet());
alert(student.totalInches);
```

Alternate Method Definition (private variables)

```
var student = new Student(13205, "George");
alert(student.setHeightInches(74));
alert(student.getHeightFeet());
alert(student.feet);
```

```
function Student(studentID, name) {
  var inches = -1;  // private variable
  var feet   = -1;    // private variable

  // private method
  function inchesToFeet(i) {
    return Math.floor(i/12);
  }

  // public methods
  this.setHeightInches = function(n) {
    if ((n>0) && (n<100)) {
      feet = inchesToFeet(n);
      inches = n-(feet*12);
      return true;
    } else {
      return false;
    }
  }
  this.getHeightFeet = function() {
    if ((feet>0) || (inches>0)) {
      return feet;
    } else {
      return -1;
    }
  }
}
```

String Objects

- (Conceptually) an array of Unicode characters with some interfaces
  - var s = “Mr. Spock”
  - s.toLowerCase is “mr. spock”
  - s.substr(3,4) is “ Spo”
  - s.indexOf(“k”) is 8
  - s.split(“ ”) is [“Mr.”, “Spock”]
  - s.link(http://bit.ly.CUjV) is Mr. Spock
  - s + “Captain Kirk” is “Mr. SpockCaptainKirk”

Everything is an Object

- var b = new Boolean(true);
- var n = new Number(3.15);
- var n = new Number(3); // same as 3.00
- var a = new Array(5);

Some Handy Methods

- document
  - document.writeln(“Test!”);
  - var e=document.getElementById(“goButton”);
  - document.cookie=“message=saveme”;
  - var c=document.cookie.split(“=“)[1];

- window
  - window.prompt(“Input please”);
  - var w=window.open(“”, “New Window”, “”);

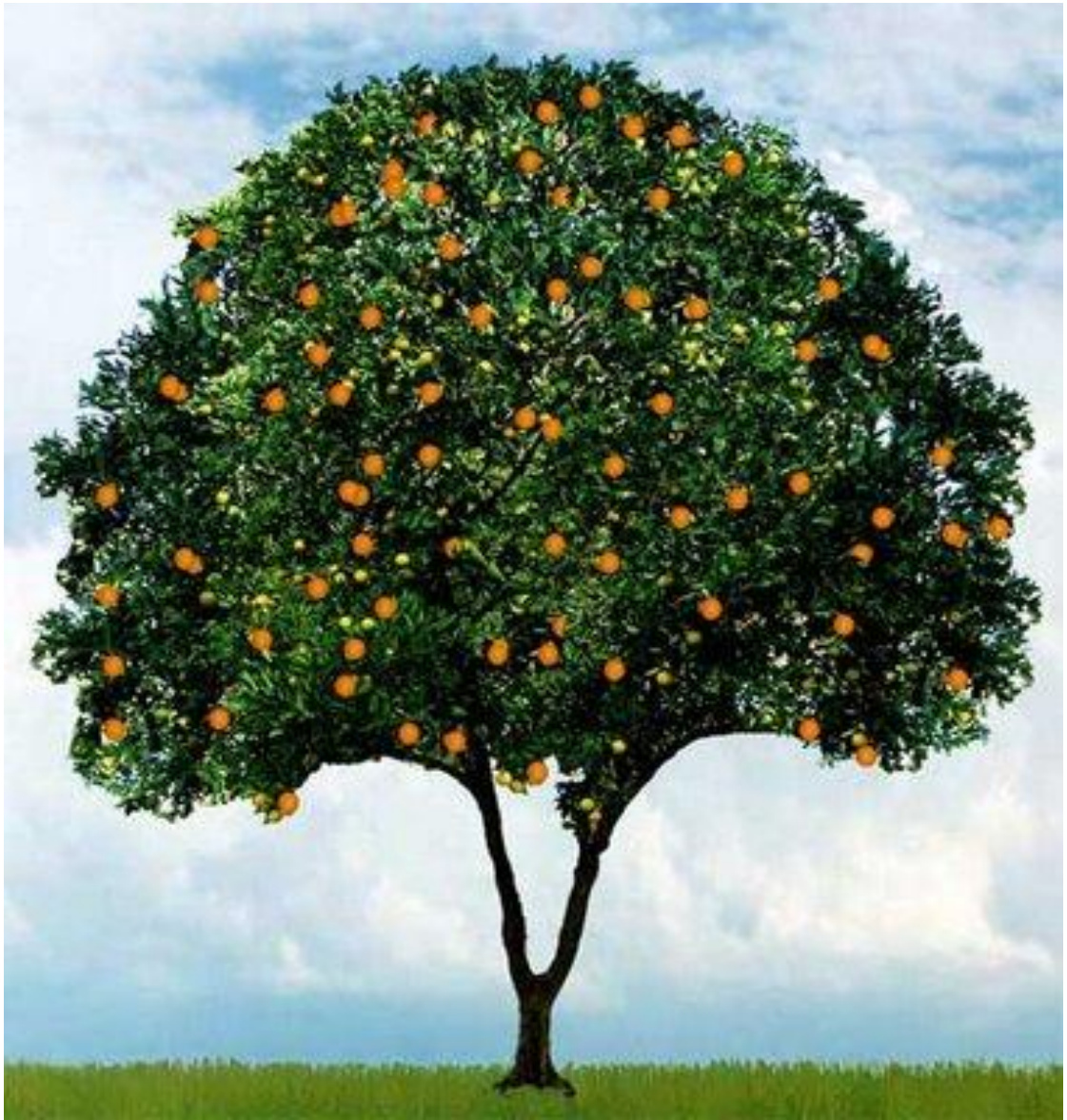
Why Use Objects?

- A way of thinking about programming
  - Objects are nouns, methods are verbs
- A form of defensive programming
  - Hides private variables and methods
  - Allows you to make behaviors explicit
- Represent complex data structures
  - Airplanes have pilots, fuel, and destinations

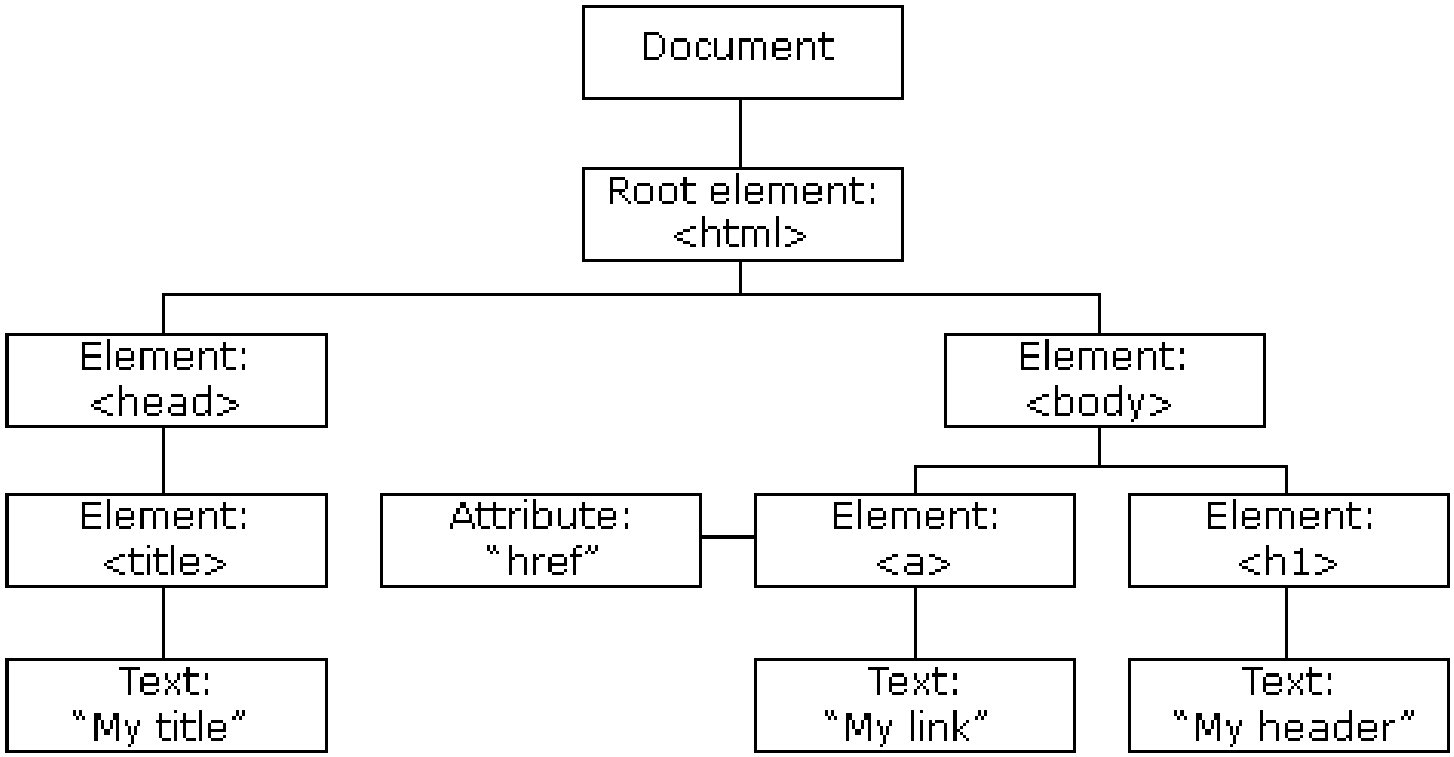
Design Exercise

- Design a **class** for email messages
- **Private** internal representation should include:
  - Header (date, time, sender, recipients, subject, …)
  - Body
  - Attachments (which may be other emails!)
- **Public** interfaces should include
  - Message creation
  - Access to specific header fields, the body, and specific attachments

A Tree



A DOM Tree


Document
Root element:
<html>
Element:
<head>
Element:
<body>
Element:
<title>
Attribute:
"href"
Element:
<a>
Element:
<h1>
Text:
"My title"
Text:
"My link"
Text:
"My header"

A DOM Tree

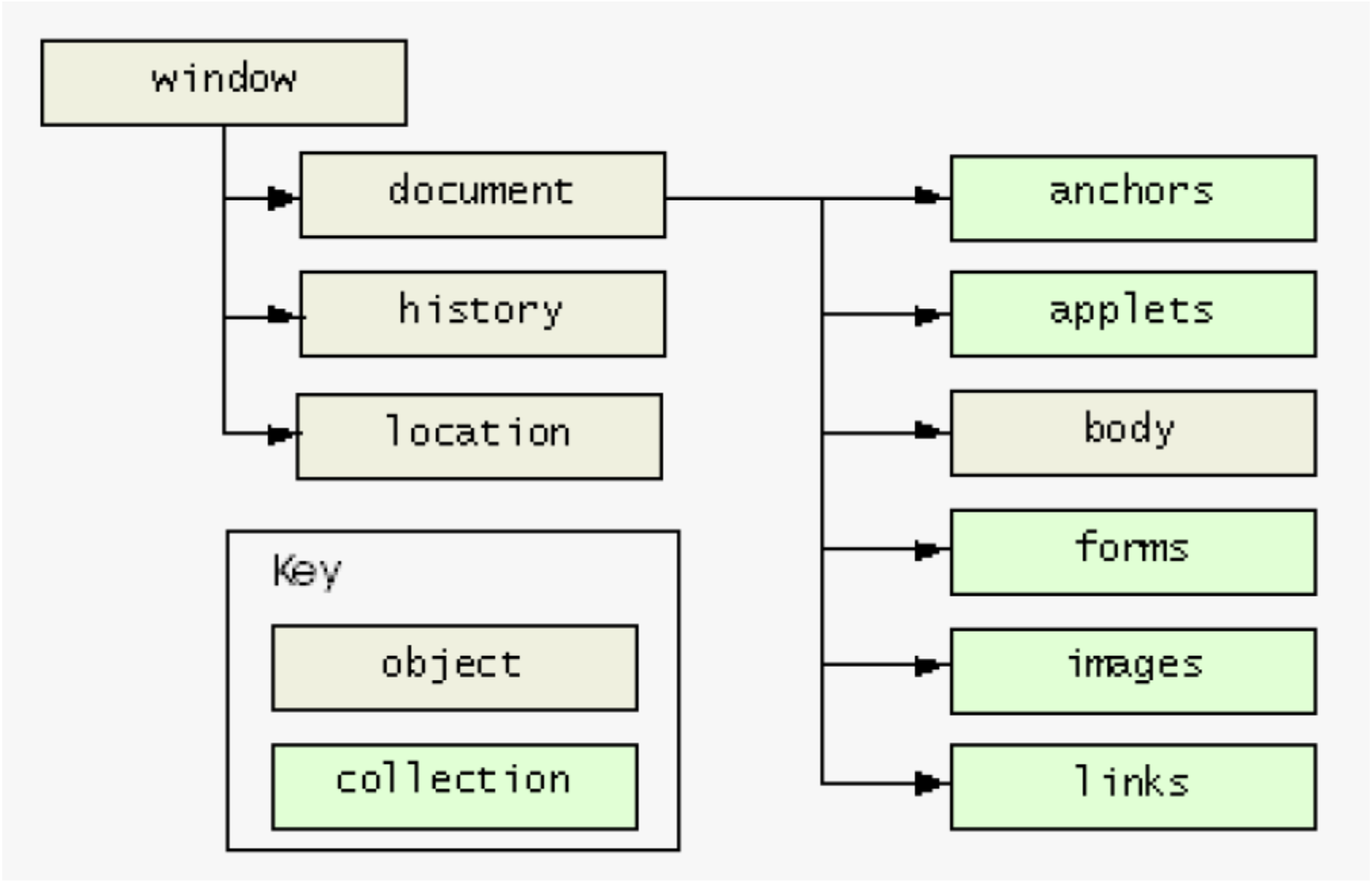
Document
Root element: <html>
Element: <head>
Element: <body>
Element: <title>
Text: "My title"
Element: <a>
Attribute: "href"
Text: "My link"
Element: <h1>
Text: "My header"

Document Object Model

- Tree representation of HTML structure
- Install firebug as a Firefox add-on
- Activate firebug on http://ischool.umd.edu
  - Then browse the DOM

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang="en" dir="ltr" xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
  <body id="homePage">
    <div id="page-wrapper">
      <div id="page">
        <div id="header" class="clearfix">
          <p id="umLogo">
          <p id="iSchoolLogo">
          <div id="headerExtras">
          <div id="menu">
            <div class="region region-sidebar-first column sidebar">
              <div class="section">
                <div id="block-user-1" class="block block-user region-odd odd region-count-1 count-1">
                  <h2 class="title">Navigation</h2>
                  <div class="content">
                    <ul class="menu">
                      <li class="collapsed first">
                        <a title="Prospective Students" href="/content/prospective-students">Prospective Students</a>
                      </li>
                      <li class="collapsed">
                      <li class="collapsed">
                      <li class="collapsed">
                      <li class="collapsed">
                      <li class="collapsed">
                      <li class="collapsed last">
                    </ul>
                  </div>
                </div>
              </div>
            </div>
          </div>
        </div>
        <div id="content">
        <div id="footer">
      </div>
    </div>
  </body>
</html>
```

Getting to DOM Elements



Access to DOM Elements

- Find a single element

  element = document.getElementById(“input2”);
- Find multiple elements

  list = document.getElementsByTagName(input);

  list = document.getElementsByName(“myInput”);
- Move up in the DOM tree
  - element1 = element2.parentNode;
- Move down in the DOM tree
  - list = element1.childNodes

Why Program?

- Data manipulation
- Simulation
- Control
  - Interaction
  - Embedded

Term Project

- Each team will have a Virtual Machine
  - Probably running Red Hat Linux
- Teams can request Drupal or Ruby on Rails
  - Server side or Ajax programming environment
  - 3rd option: JavaScript client side (TerpConnect)
- H6 includes teaming preferences
- P1 is project plan
- P2 is project design
- P3 is presentation slides