Classes, Objects, and UML
Java Objects
Class: a means of creating new types
• Group data elements that describe some abstract concept
• These data elements can be primitive data or other objects
• This is an important way to organize your data – and hence your coding!
Java Objects

An object is one instance of a class

- Occupies a block of memory in the heap that contains the values of the data elements
- Each instance has its own memory
- The set of values stored in this memory block is called the state of the object
- In code, we refer to object instances using a reference to the memory block
Java Objects

Behavior: class defines the legal ways to change the object’s state

• There may be no methods to do so (e.g. String, Integer, Float classes). These are called *immutable classes*

• There may be many methods that change the object’s state (e.g. StringBuffer class)
Examples

• What is the state of a StringBuffer object?
• How can the state of the StringBuffer object be changed?

(StringBuffer API)
Examples

What is the state for Date?
Instance Methods

Instance methods describe the behavior of objects

• Accessors: Methods used to report the state of objects (including *getters*)
• Mutators: Methods used to change the state of objects (including *setters*)

**Syntax:** `object.method(parameters)`
A Class is a Contract

• Classes can construct objects
• All operations on an object: must always leave the object in a consistent state
  • Enforce through variable visibility and through methods

• Best practice:
  • On entry to a method: assume that the object is in a consistent state
  • On exit, ensure that it is still consistent
Examples

Find examples of accessors and mutators in StringBuffer
• And String
Examples

What would an inconsistent state be for a Triangle object?
• Properties: height, base width, area
A Class as an “Encapsulater”

• A class hides many details from the outside world

• The user of a class only has to worry about the class’ public interface
  • Easier to understand how to use the class
  • The implementation of the underlying class can change without the user knowing
• Lab 1 grading complete soon
• Lab 2 grading is underway
• Lab 3 and project 1 go out this week
• Team assignments (for projects) will be done soon
# Unified Modeling Language (UML)

<table>
<thead>
<tr>
<th>Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>- title: String</td>
</tr>
<tr>
<td>- author: String</td>
</tr>
<tr>
<td>- isbn: String</td>
</tr>
<tr>
<td>+ Book(myAuthor: String, myTitle: String, myISBN: String)</td>
</tr>
<tr>
<td>+ getTitle(): String</td>
</tr>
<tr>
<td>+ getAuthor(): String</td>
</tr>
<tr>
<td>+ getISBN(): String</td>
</tr>
</tbody>
</table>
Unified Modeling Language (UML)

Let’s implement this class

<table>
<thead>
<tr>
<th>Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>- title: String</td>
</tr>
<tr>
<td>- author: String</td>
</tr>
<tr>
<td>- isbn: String</td>
</tr>
</tbody>
</table>

+ Book(myAuthor: String, myTitle: String, myISBN: String)
+ getTitle(): String
+ getAuthor(): String
+ getISBN(): String
UML Class Diagrams

• Name of class at top
• Middle section contains data
  • Name: type
• Bottom section contains methods
  • Name(param1: type, param2: type...): return type
• Plus (+) means public
• Minus (-) means private
Putting it All Together

• TopHat exercise
Unified Modeling Language (UML)

Umlet tool:
http://www.umlet.com/
Classes & Objects (continued)
Public vs Private Data

Can be a tough decision.
• What are the pros & cons?
Public vs Private Data

• Public Pros:
  • Easy access to all data by other classes
  • Don’t have to implement getters and setters

• Public Cons:
  • Can’t protect the data from other classes – easy to get into an inconsistent state
  • Therefore, the class cannot make any guarantees about how it behaves
Public vs Private Data

For this class:
• We want our classes to protect themselves
• All instance variables will be declared as private or protected (more on the latter soon)
• All external access to instance variables will be through public methods
Instance vs Class Data

• Each object gets its own copy of *instance data*
• All objects in a class share one copy of *class data*
  • In UML, class variables are underlined
  • In the class definition, class variables are declared as *static*
Example

Suppose we were going to design a post-it note application

• What is the state of the Note?
• How might the state be changed?
  • Let’s make UML for this...
Example

How are we going to store things like the number of characters that are allowed in the note?

• Why is instance data not appropriate for this?
Class Variables

Only one copy of the variables for all instances in the class

• Declare as static:

private static final int maxCharacters = 100;
private static int numNotes = 0;
Class Methods

• Class-level methods are labeled *static* in Java
• Invocation (execution):
  
  `Class.methodName(parameters)`
Class Methods

Examine Math class on Java API
• How is Math different from String?
Class Methods

• Class methods have no access to instance data
  • There is no object, so no instance data

• Example: examine toString() in Integer class for both instance and class methods
Instance Methods

- Always are called with respect to an object instance
- Can “see” both instance and class variables
Parameter Passing

Primitive data types:
• Value gets copied (pass by value)
• Changes made in method don’t affect the calling method
  • Except when a value is explicitly returned
• A reference is a primitive data type
Parameter Passing

Objects:

• References are passed by value
• But: inside and outside the method, the reference refers to the same memory location
• So: changes to data by the called method are visible to the calling method
  • True for both primitive data and objects inside the object
Method Overloading

Overloading: using the same method name, but different parameters

• Common when we want to assume default parameters
• or when different types convey similar types of information

public void addValue(int val);
public void addValue(double val);
“this"

• The “this” keyword is a reference that refers to the object on which an instance method was called on

• Can also refer to a constructor
“this” Referring to the Called Object

class Person{
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

“this” as a Constructor

class Person{
    private String name;
    private int age;

    public Person(String name, int age){
        this.name = name;
        this.age = age;
    }

    public Person(String name){
        this(name, 20);
    }

    public Person(){
        this("Bob", 42);
    }
}
Classes within Classes

• One of the “big wins” with object-oriented programming is that we can define classes hierarchically

• Now that we have a “Person”, we can create new classes that contain Persons
Classes within Classes

class Course {
    private int courseNumber;
    private Person instructor;
    private ArrayList<Person> teachingAssistants;
    private ArrayList<Person> students;

    :
    :
    :
}

Andrew H. Fagg: CS 2334: Classes and Objects
Classes within Classes

Constructor is responsible for initializing underlying classes...

class Course {
    private int courseNumber;
    private Person instructor;
    private ArrayList<Person> teachingAssistants;
    private ArrayList<Person> students;

    public Course(){
        teachingAssistants = new ArrayList<Person>();
        students = new ArrayList<Person>();
    }
}

Andrew H. Fagg: CS 2334: Classes and Objects
Classes within Classes

Constructors can use the default constructor to handle some initialization

class Course {
    :
    public Course() {
        teachingAssistants = new ArrayList<Person>();
        students = new ArrayList<Person>();
    }

    public Course(int courseNumber, Person instructor) {
        this();
        this.courseNumber = courseNumber;
        this.instructor = instructor;
    }
    :
}