Classes, Objects, and UML
Notes

• Lab 1: grading this week
• Lab 2: available soon
• Catme: should have email invitation
  • Use for group formation

• Next Week: lab 3 and project 1
Notes

• Advancing to the Data Structures course (CS 2413) requires an A or B in at least one of 1323 and 2334 (and at least a C in both)
• Programming team
Lab 1 Lessons
Lab 1 Lessons

• Should now have the essential tools running: Java, Eclipse, Web-Cat
• Web-Cat gives important feedback – use it
• Learn to read the documentation
• Learn to read the specification in detail
  • Requirements are very precise
• Start early
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
• Provide methods that operate on these data elements

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
Instance Methods

Some specific types of instance methods
• Accessors: Methods used to report the state of objects
• Mutators: Methods used to change the state of objects

Syntax: reference.method(parameters)
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 a Date object?
Instance Methods: Special Cases

Some specific types of instance methods

• Getters: Accessor methods used to report the low-level state of objects

• Setters: Mutator methods used to change the low-level state of objects
Instance Methods: Special Cases

Mutator methods:

• If there are no methods that change the state of an object. These are called *immutable classes* (e.g. String, Integer, Float classes)

• There may be many methods that change the object’s state (e.g. StringBuffer class). We call these classes *mutable*
Examples

Find examples of accessors and mutators in StringBuffer

• And String
A Class is a Contract

The set of instance methods define the legal ways that an object may be accessed/changed

• All operations on an object: must always leave the object in a consistent state
  • Enforce through variable visibility and through proper method definition

• 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

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
Unified Modeling Language (UML)

UML is a spatial representation that describes:

• The definition of a class
• How the different classes relate to one another
Unified Modeling Language (UML)

```java
class Book {
    String title;
    String author;
    String isbn;

    Book(String author, String title, String isbn) {
        this.title = title;
        this.author = author;
        this.isbn = isbn;
    }

    String getTitle() {
        return title;
    }

    String getAuthor() {
        return author;
    }

    String getISBN() {
        return isbn;
    }
}
```
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>
<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>
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
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
Putting it All Together

• TopHat exercise
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 a 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:

```java
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)
```
• Project 1 is live
• Lab 1 grades have been transferred to Canvas
• Lab 2 grading has started
• Lab 3 is live
Class Methods

Examine Math class on Java API

• How is Math different from String?
Class Methods

• Many class methods have no access to instance data
  • There is no object, so there is no instance data
  • Example: examine toString() in Integer class for both instance and class methods

• But: if a static method in a class has access to an object reference, it can access the private instance variables of that object
  • My opinion: this is poor language design. Avoid doing this!
Instance Methods

• Are always 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
Object 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

```java
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;

    ...
    ...
}

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> ( );
    }
}
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;
    }

    // : public methods

}