CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2012
Doc 25 Review
May 3 2012

What this course is about

Writing quality OO code
Design Patterns
Coupling & Cohesion

Unit Testing
Refactoring

Anti-Quality

```
public void onCreate(Bundle icicle) {
        super.onCreate(icicle);

        this.setContentView(R.layout.main);

        db=new Database(this);


        listView= (ListView)findViewById(R.id.list);

        ArrayList<Instructor> list=new ArrayList<Instructor>();



        LayoutInflater inflater = this.getLayoutInflater();
        View header = inflater.inflate(R.layout.main_header, null);

        listView.addHeaderView(header);
        adapter = new RatingAdapter(list);
        listView.setAdapter(adapter);

        try {
        constantsCursor=db
                        .getReadableDatabase()
                        .rawQuery("SELECT _id, firstName, lastName, office, phone, email, rating, totalRatings "+
                                                "FROM instructors order by lastName",
                                                null);

while (constantsCursor.moveToNext()) {
        //list the instructors in the db
        int id = constantsCursor.getInt(0);
        String firstName = constantsCursor.getString(1);
        String lastName = constantsCursor.getString(2);
        String office = constantsCursor.getString(3);
        String phone = constantsCursor.getString(4);
        String email = constantsCursor.getString(5);
        int rating = constantsCursor.getInt(6);
        int totalRatings = constantsCursor.getInt(7);

        //publish the data
        Instructor instructor = (Instructor) new Instructor(firstName, lastName, office, phone, email, rating, totalRatings, id);
        adapter.add(instructor);


}
```

```
//
// Method: getFirstName
// @return type: String
// This method returns the instructors first name
//
public String getFirstName() {
        return this.m_firstName;
}
```

```
//rew 1 means what???
        if(last_opened.equals("1")) {
                //rew hope these never change
                httpGet = new HttpGet("http://bismarck.sdsu.edu/rateme/list");
            } else {
                Log.i("Foo","not first timer");

                httpGet = new HttpGet("http://bismarck.sdsu.edu/rateme/list/since/" +
last_opened);
            }
```

```
String url3 = "http://bismarck.sdsu.edu/rateme/comments/";
```

```
public class Assignmt_Three_t2Activity extends ListActivity implements
View.OnClickListener {
    /** Called when the activity is first created. */
      ArrayList<String> array_ids;
      ArrayList<String> changedIds;
      ArrayList<String> array_fnames;
      ArrayList<String> array_lnames;
   ArrayAdapter<String> listAdapter;
```

```
protected Object doInBackground(String... params) {
                Object response = null;
                infoToGet = params[INFORMATION_TO_GET_IDX];
                String url = params[URL_IDX];
                HttpClient httpClient = new DefaultHttpClient();
                HttpGet getMethod = new HttpGet(url);

                try {
                        ResponseHandler<String> responseHandler = new BasicResponseHandler();
                        response = httpClient.execute(getMethod, responseHandler);
                } catch (Throwable t) {
                        Log.e("gill", t.toString());
                        response = t;
                        httpClient.getConnectionManager().shutdown();
                        return response;
                }

                httpClient.getConnectionManager().shutdown();

                if (infoToGet.equals(INSTRUCTOR_LIST)) {
                        try {
                                _instructorDb = _dbHelper.getWritableDatabase();
                                JSONArray instructorArrayAsJson = new JSONArray(
                                                (String) response);
                                int numberOfInstructors = instructorArrayAsJson.length();

                                for (int k = 0; k < numberOfInstructors; k++) {
                                        JSONObject instructor = (JSONObject) instructorArrayAsJson
                                                        .get(k);
                                        String firstName = instructor.getString("firstName");
                                        String lastName = instructor.getString("lastName");
                                        int id = instructor.getInt("id");

                                        Cursor cursor = _instructorDb.rawQuery(
                                                        "SELECT _id, firstname, lastname, office, phone, " +
                                                                        "email, average_rating, number_ratings " +
                                                                        "FROM instructors WHERE _id=?",
                                                        new String[]{Integer.toString(id)});

                                        cursor.moveToFirst();
                                        if (cursor.getCount() == 0) {           // Instructor not in db
                                                ContentValues categoryContent = new ContentValues(3);
                                                categoryContent.put("_id", id);
                                                categoryContent.put("firstname", firstName);
                                                categoryContent.put("lastname", lastName);

                                                _instructorDb.insert("instructors", null, categoryContent);
                                        }
                                        else {
                                                ContentValues categoryContent = new ContentValues(7);
                                                categoryContent.put("firstname", firstName);
                                                categoryContent.put("lastname", lastName);
                                                categoryContent.put("office",
                                                                cursor.getString(cursor.getColumnIndex("office")));
                                                categoryContent.put("phone",
                                                                cursor.getString(cursor.getColumnIndex("phone")));
                                                categoryContent.put("email",
                                                                cursor.getString(cursor.getColumnIndex("email")));
                                                categoryContent.put("average_rating",
                                                                cursor.getString(cursor.getColumnIndex("average_rating")));
                                                categoryContent.put("number_ratings",
                                                                cursor.getString(cursor.getColumnIndex("number_ratings")));

                                                _instructorDb.update("instructors", categoryContent, "_id=?",
                                                                new String[]{Integer.toString(id)});
                                        }
                                        cursor.close();
                                }
                        } catch (JSONException e) {
                                e.printStackTrace();
                        }
                        _instructorDb.close();
                }

                else if (infoToGet.equals(INSTRUCTOR_COMMENTS)) {
                        try {
                                _instructorDb = _dbHelper.getWritableDatabase();
                                instructorId = Integer.parseInt(params[INSTRUCTOR_ID_IDX]);

                                JSONArray commentArray = new JSONArray(
                                                (String) response);
                                int numberOfComments = commentArray.length();
```

```
newRating = -1; // flag for rating being omitted
```

```
public class Assign3Activity extends Activity {

int x = 0;



        List<String> list = new ArrayList<String>();
        List<String> list2 = new ArrayList<String>();
        List<String> list3 = new ArrayList<String>();
        List<String> list4 = new ArrayList<String>();
        List<String> newList = new ArrayList<String>();
        List<String> newList2 = new ArrayList<String>();
```

```
private String PostComments(String Arg1,String Arg2) {
```

```
private class SampleTask extends AsyncTask<String, String, String> {


        pr = new DatabaseHelper(this);
        db = new DatabaseHelper(this);
```

Prime Directive

Data + Operations

Heuristics

Keep related data and behavior in one place

A class should capture one and only one key abstraction

Heuristics

Beware of classes that have many accessor methods defined in their public interface

Do not create god classes/objects in your system

Beware of classes that have too much noncommunicating behavior

Code Smells

Duplicate Code

Duplicate Code

Duplicate Code

Duplicate Code

Duplicate Code

Duplicate Code

Duplicate Code

Duplicate Code

Long Method - Large Class

The average method size should be less than 8 lines of code (LOC) for Smalltalk and 24 LOC for C++

The average number of methods per class should be less than 20

The average number of fields per class should be less than 6.

The class hierarchy nesting level should be less than 6

The average number of comment lines per method should be greater than 1

Mark Lorenz, *Object-Oriented Software Development: A Practical Guide*, 1993, Appendix I Measures and Metrics

Feature Envy

A method seems more interested in a class other than the on it is in.

Data Clumps

Same three or four data items together in lots of places

Primitive Obsession

Using primitive types instead of creating small classes

Switch Statements

How do you program without them?

Lazy Class

Class that is not doing enough to pay for itself

Data Class

Class with just fields and setter/getter methods

Data classes are like children.

They are okay as a starting point, but to participate as a grownup object, they need to take some responsibility

Inappropriate Intimacy

Classes that spend too much time delving into other classes private parts

Message Chains

location = rat.getRoom().getMaze().getLocation()

Patterns

Creational Patterns

Abstract factory	Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
Builder	Separate the construction of a complex object from its representation allowing the same construction process to create various representations.
Factory method	Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses (dependency injection[17]).
Lazy initialization	Tactic of delaying the creation of an object, the calculation of a value, or some other expensive process until the first time it is needed.
Multiton	Ensure a class has only named instances, and provide global point of access to them.
Object pool	Avoid expensive acquisition and release of resources by recycling objects that are no longer in use. Can be considered a generalisation of connection pool and thread poolpatterns.
Prototype	Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.
Resource acquisition is initialization	Ensure that resources are properly released by tying them to the lifespan of suitable objects.
Singleton	Ensure a class has only one instance, and provide a global point of access to it.

From http://en.wikipedia.org/wiki/Software_design_pattern

Structural patterns

Adapter or Wrapper orTranslator.	Convert the interface of a class into another interface clients expect. An adapter lets classes work together that could not otherwise because of incompatible interfaces. The enterprise integration pattern equivalent is the Translator.
Bridge	Decouple an abstraction from its implementation allowing the two to vary independently.
Composite	Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
Decorator	Attach additional responsibilities to an object dynamically keeping the same interface. Decorators provide a flexible alternative to subclassing for extending functionality.
Facade	Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
Front Controller	The pattern relates to the design of web applications. It provides a centralized entry point for handling requests.
Flyweight	Use sharing to support large numbers of similar objects efficiently.
Proxy	Provide a surrogate or placeholder for another object to control access to it.
Module	Group several related elements, such as classes, singletons, methods, globally used, into a single conceptual entity.

From http://en.wikipedia.org/wiki/Software_design_pattern

Behavioral patterns

Blackboard	Generalized observer, which allows multiple readers and writers. Communicates information system-wide.
Chain of responsibility	Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.
Command	Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.
Interpreter	Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.
Iterator	Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
Mediator	Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.
Memento	Without violating encapsulation, capture and externalize an object's internal state allowing the object to be restored to this state later.
Null object	Avoid null references by providing a default object.

From http://en.wikipedia.org/wiki/Software_design_pattern

Behavioral patterns

Observer orPublish/subscribe	Define a one-to-many dependency between objects where a state change in one object results in all its dependents being notified and updated automatically.
Servant	Define common functionality for a group of classes
Specification	Recombinable business logic in a Boolean fashion
State	Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.
Strategy	Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.
Template method	Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.
Visitor	Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

From http://en.wikipedia.org/wiki/Software_design_pattern

Concurrency Patterns

Active Object
Balking
Binding properties
Messaging design pattern (MDP)
Double-checked locking
Event-based asynchronous
Guarded suspension
Lock
Monitor object
Reactor
Read-write lock
Scheduler
Thread pool
Thread-specific storage

How to Select a design pattern

Consider how design patterns solve design patterns

Scan Intent sections

Study how patterns interrelate

Study patterns of like purpose

Examine a cause of redesign

Consider what should vary in your design

Cause of Redesign

Creating an object by specifying a class explicitly

Abstract factory, Factory Method, Prototype

Cause of Redesign

Dependence on specific operation

Chain of Responsibility, Command

Cause of Redesign

Dependence on hardware and software platforms

Abstract Factory, Bridge

Cause of Redesign

Dependence on object representations or implementations

Abstract factory, Bridge, Memento, Proxy

Cause of Redesign

Algorithmic dependencies

Builder, Iterator, Strategy, Template Method, Visitor

Cause of Redesign

Extending functionality by subclassing

Abstract Factory, Bridge, Chain of Responsibility, Command, Facade, Mediator, Observer

Cause of Redesign

Inability to alter classes conveniently

Adapter, Decorator, Visitor

What is the difference between

Structural and behavioral patterns