So far,

We've created a simple persistence project with cloud references.
There were lots of relationships between entities that must be fulfilled.
How do we keep that straight?
How does one efficiently develop a cloud application?
Keys to AppEngine development: **stubbing**, **binding**, and **refinement**

**Stubbing**: creating and naming a placeholder method/object/html item for later control.

**Binding**: insuring that there is an exact match between attributes of several related objects.

**Refinement**: Proceeding to develop code from stubs, using full powers of Eclipse to remember bindings where appropriate!
Stubbing HTML

Create an HTML page that represents project input/output.
Create "stubs" (empty elements) that will contain dynamic content.
Name them via the ID field in HTML.
Second step:

Create java describing widgets that appear on the page
Bind them to ID's.
Fill them with junk (but plausible) data for now.

Widget is an implicit binding between HTML and (GWT) Java.
Has a java exposure as an instance in the class of page.
Has an HTML exposure that is used when stuffing it into an ID'd place.
Kinds of widgets

Some kinds of widgets

**FlexTable**: a table of data, with many formatting options.

**Button**: a pressable button.

**Label**: settable text with no interaction options.

**Panel**: a container for objects.
Creating widgets

Sunday, January 30, 2011
1:48 PM

Creating widgets: in onModuleLoad method:
  Create a Java object for each HTML object to be managed.
  Put objects inside one another, as desired.
  Generate initial callback calls for dynamic content.
Important caveats when stubbing

Don't try to implement the whole stack at once.
Test one stub at a time.
Each stub should generate plausible (but not necessarily accurate) depictions.
Debug clientside interactions first, then write serverside code.
Good news: created by "new project" wizard, maintained by Eclipse (sometimes).
Bad news: very hard to rename things!

A small suggestion:
   Save your project under a different name before trying to rename anything!
Two key binding files for SimplePersistence project:

(src/com.example.Example02/)SimplePersistence.gwt.xml: Describes bindings between java code and HTML.

(war/WEB-INF/)web.xml: describes mapping between servlets and (server) URLs.
HTML/Java binding: SimplePersistence.gwt.xml

```xml
<module rename-to='simplepersistence'>
  <!-- Inherit the core Web Toolkit stuff. -->
  <inherits name='com.google.gwt.user.User'/>

  <!-- Inherit the default GWT style sheet. You can change the theme of your GWT application by uncommenting any one of the following lines. -->
  <inherits name='com.google.gwt.user.theme.standard.Standard'/>
  <!-- <inherits name='com.google.gwt.user.theme.chrome.Chrome'/> -->
  <!-- <inherits name='com.google.gwt.user.theme.dark.Dark'/> -->

  <!-- Other module inherits -->

  <!-- Specify the app entry point class. -->
  <entry-point class='com.example.Example02.client.SimplePersistence'/>

  <!-- Specify the paths for translatable code -->
  <source path='client'/>

</module>
```
web.xml: describes bindings with web server

```xml
<web-app>

  <!-- Servlets: can be more than one -->

  <!-- the servlet that stores and retrieves persistent data -->
  <servlet>
    <servlet-name>thingServlet</servlet-name>
    <servlet-class>
      com.example.Example02.server.PersistentThingServiceImpl
    </servlet-class>
  </servlet>

  <!-- refer to this as http://localhost:8888/simplepersistence/thing -->
  <servlet-mapping>
    <servlet-name>thingServlet</servlet-name>
    <url-pattern>/simplepersistence/thing</url-pattern>
  </servlet-mapping>

  <!-- Default page to serve -->
  <welcome-file-list>
    <welcome-file>SimplePersistence.html</welcome-file>
  </welcome-file-list>

</web-app>
```
Notes on servlet bindings:
Servlet name is **common attribute** to class binding and URL binding
Can have one class binding known by **multiple URLs**. Can have a servlet without a URL, which means it can't be called!
Cannot have a URL without a bound servlet (will cause application failure).
Some bindings describe what the compiler will do. **Example:**

"client" code directory gets compiled with GWT. 
"server" code directory gets compiled with real Java compiler.

Watch out: errors in GWT compilation persist until you run the GWT compiler again; other (real Java) errors disappear immediately when repaired.
Eclipse is particularly suitable to top-down coding via **top-down stepwise refinement**.

Special comment

// TODO: ...

Puts a task onto your TODO list.

Eclipse sets up bindings, and then tracks their correctness.

Example: eclipse knows that every interface that extends RemoteServer needs a corresponding Async interface!
Stubbing

When programming in eclipse, it often helps to "stub" a function with a // TODO comment. A stub should return a reasonable value and have a reasonable effect, but not necessarily the effect you wanted. The global list of TODO comments forms a "to-do list".
How to create a project
   Start with the sample project.
   From interface to cloud, top to bottom
   Add one detail at a time.
   Respond to issues generated by eclipse.
Proceed via a series of refinements:
   Stub complex regions for later development.
   Keep a TODO list of things to be done next.
   Test unrefined code before each next step.
Importing and exporting projects

Exporting a project to a folder:
File/Export.../to filesystem...
Only files that are subject to change are saved.

Importing a project from a folder:
File/Import.../from folder...
Project must not exist beforehand.

Code for project is
/comp/150CPA/examples/SimplePersistence
Copy onto desktop, "Import...".
Refactoring

Often during development, you find that you've named something the wrong name or put it in the wrong place.
Changing the name or location of a thing is called "refactoring".
When you "refactor" a thing, references to it from java are "usually" **repaired**.
But watch out for XML files: their contents are **not repaired**!
A couple of really important eclipse tricks

Control-<space>: bring up options for matching entities at your current position in the file.

Central error list: points you to each syntax error in the file

Syntax errors in server code are detected immediately.

Careful: syntax errors in client code are only detected when you compile it.
Stubbing goes hand-in-hand with top-down design
Decide what the interface looks like.
Build the interface in HTML.
Decide what it should do.
Implement and test one function at a time.
Using external services

So far, we've only studied how to use the distributed datastore service.
There are many others, each with its own API.
Most important: **User service**.
import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;
// ...
Label userLabel = new Label(""); // <td id="userLabelContainerID"></td>
Anchor loginAnchor = new Anchor();
// ...
UserService userService = UserServiceFactory.getUserService();
User user = userService.getCurrentUser();
if (user != null) {
    userLabel.setText("Hello, " + user.getNickname());
    loginAnchor.setHref(userService.createLogoutURL("/"));
    loginAnchor.setText("logout");
} else {
    userLabel.setText("Hello guest!");
    loginAnchor.setText("login");
    loginAnchor.setHref(userService.createLoginURL("/"));
}

// createLoginURL creates a URL with an implicit reference
// to *your* page, so that when you login, you get bounced back
// via a redirect
User identity caveats

Cannot store things in the User object but
Can store things in the datastore indexed by User.
User.getID() field persists over lifetime of account.
User.getEmail() field can be changed by user and rendered inoperative!

The default is to use the email field!
Other services

Wednesday, February 03, 2010
5:04 PM

Other services

Many APIs for various google entities
  Calendar
  Mail
  Youtube

All depend upon identifying the User beforehand
Once an application is running, it can be "deployed"
    Then it runs on google's servers.
    And acquires an application URL.
    That I can test!
    I can't get to your localhost applications!
Thus, the first assignment is to create a deployed application!
    This means that you need a google account, so sign up asap!