Module interface design example: 
Hiding dependence on face.com

face.com is pretty good, but ...

- a better service might come along later, or
- face detection might be supported in hardware, or
- face.com might go out of business, or
- face.com might start charging too much, or
- face.com might not support a feature we need later

so ... we might want to limit our dependence on that particular service

... like a device-hiding module

but face4j is pretty good!

- We could use the face4j API directly

  - It’s already written
  - It’s (probably) pretty well tested, and works

  - It is somewhat more complex than we need
  - It does not protect us from changes

A thin veneer ...

A simple API that would work for any face recognition service

The services provided by face.com, with a Java API
What should we hide?
What are some things that might differ between face recognition services and devices?

(Some possibilities)
- Accepted photograph formats & sources
- Identifiers (unique face tags)
- Authentication and rate limiting
- Detect/Recognize protocol
- Confidence filtering
- Options (e.g., “aggressive” mode)

We’d like it to be simple ...

Recognize:
- Input: A photo (jpg format)
- Output: Identified person

It’s not going to be quite that simple, though ... what else will the interface need to include?

Complications of the API

Authentication
- How can we authenticate, without making other parts of our application dependent on face.com?

Exceptional conditions
- Not exactly the face4j exceptions (?), but we need to consider the exceptional conditions
- What’s a “user id”? What’s a “namespace”?
- Might differ from service to service
Namespaces and IDs

With face.com, user IDs are associated with tags in facebook, twitter, or private namespaces.

Assume we want private namespaces ...

How do we abstract to something that could use a different photo identification service?

Abstracting Namespace

A namespace is a collection of user records

- Conceptually similar to a “map” or “dictionary” structure, like a Java hashmap or treemap.
- Difference: keys are not unique objects

Face.com UIDs are strings (“name@space”)

- Other services might use a different kind of key. What should we do?
- Suggestion: We could map other key types to strings, if needed for another service.

What can we assume?

Ideally we should look at 3-4 face identification services ... what do they have in common?
But we don’t have other examples yet.

What to assume

Any useful face recognition service will look up faces in a “logical database” (like a name space)
But ...
The database might be local or remote
Keys might not be uid@namespace
UIDs and name spaces might not be strings
FaceClient faceClient =
new DefaultFaceClient(API_KEY, API_SEC);

Photo photo = faceClient.detect(URL_WITH_FACES).get(0);
Face f = photo.getFace();
faceClient.saveTags(f.getTID(), "id@namespace", "label");
faceClient.train(USER_ID);

photo = faceClient.recognize(URL_WITH_FACES, "all@yournamespace").get(0);

Since authorization could vary, how it occurs should be a secret of the module. We’ll either have a no-argument constructor or a constructor that does NOT take face.com-specific API keys and secrets.

FaceClient faceClient =
new DefaultFaceClient(API_KEY, API_SEC);

Photo photo = faceClient.detect(URL_WITH_FACES).get(0);
Face f = photo.getFace();
faceClient.saveTags(f.getTID(), "id@namespace", "label");
faceClient.train(USER_ID);

photo = faceClient.recognize(URL_WITH_FACES, "all@yournamespace").get(0);

One method for detect/getFace/saveTags/train sequence (maybe for a set of photos?)
face4j client code

```java
FaceClient faceClient =
   new DefaultFaceClient(API_KEY, API_SEC);

Photo photo = faceClient.detect(URL_WITH_FACES).get(0);
faceClient.saveTags(f.getTID(), "id@namespace", "label");
faceClient.train(USER_ID);

photo = faceClient.recognize(URL_WITH_FACES,
   "all@yournamespace").get(0);
```

One method for recognition, abstracting the namespace identifier and possibly the image specification

from face4j FaceClient.java

```java
/**
 * Convenience method for detecting faces in an image file
 * @param imageUrl @link File of the image to be uploaded to face.com for detection
 * @param uids comma delimited @code String of user IDs to search for in the photos passed in the request
 * @return @link Photo
 * @throws FaceServerException
 * @throws FaceClientException
 * @see {@link recognize(String, String)}
 */
public Photo recognize(final File imageFile, final String uids) throws FaceClientException, FaceServerException;
```