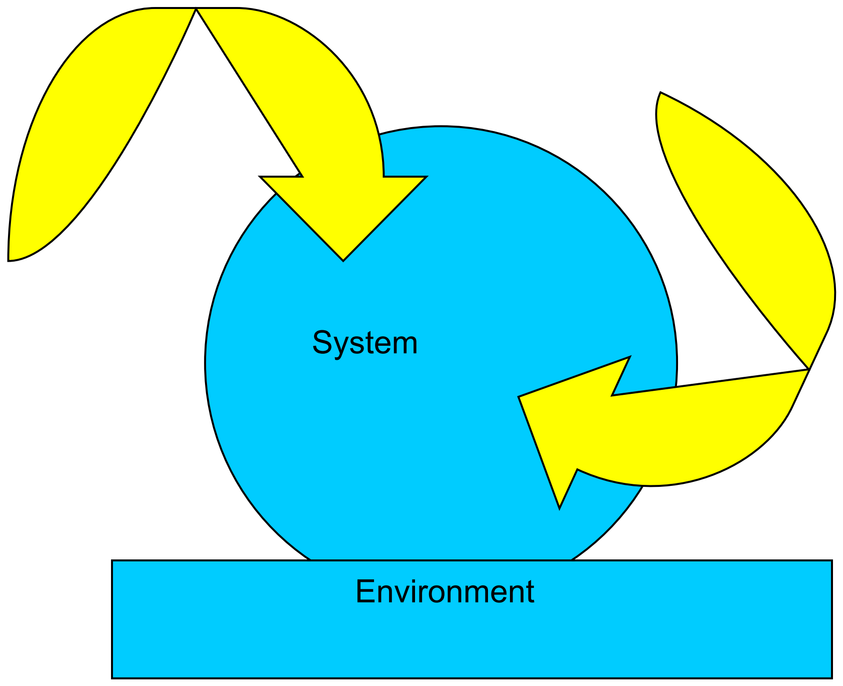
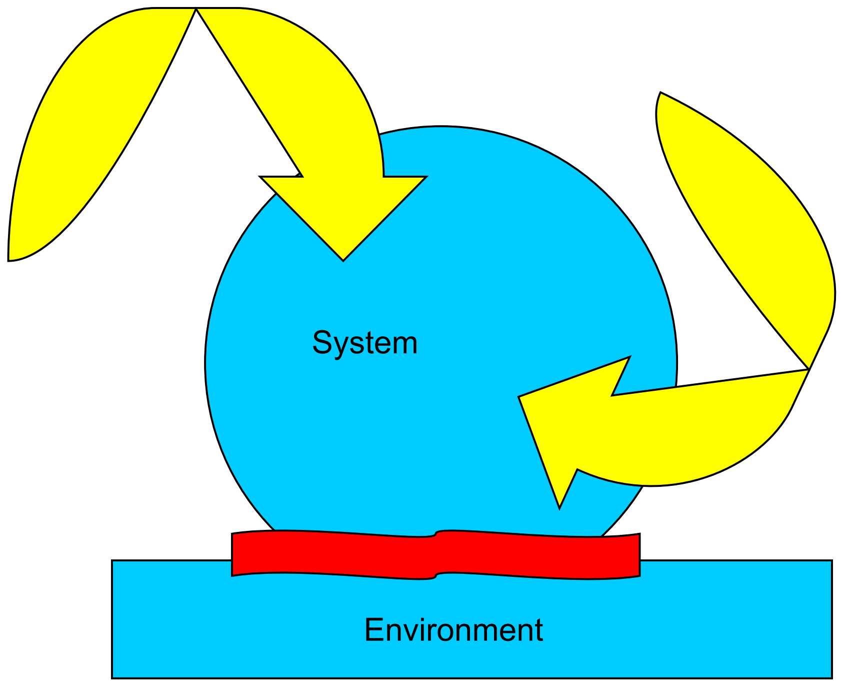
API Design as if Unit Testing Mattered

Michael Feathers
Object Mentor, Inc
Miami, FL
mfeathers@objectmentor.com

- Problem: How do you gain control over code?
- Easy in pure code



- What if the test points aren’t accessible?



Unit Testing meets API Design

- API Designers are among the most talented of developers.
- Why is it so hard to unit test code that **uses** contemporary APIs?

Unit Testing meets API Design

- API Design
  - the art of creating interfaces that are useful to clients and extensible for future needs.
  - Not all code is API code
- Unit testing
  - testing a (small) portion of software independently of everything else.

Unit Test Rules

A test is not a unit test if:

2. **It talks to the database**
3. **It communicates across the network**
4. **It touches the file system**
5. **It can't run correctly at the same time as any of your other unit tests**
6. **You have to do special things to your environment (such as editing config files) to run it.**

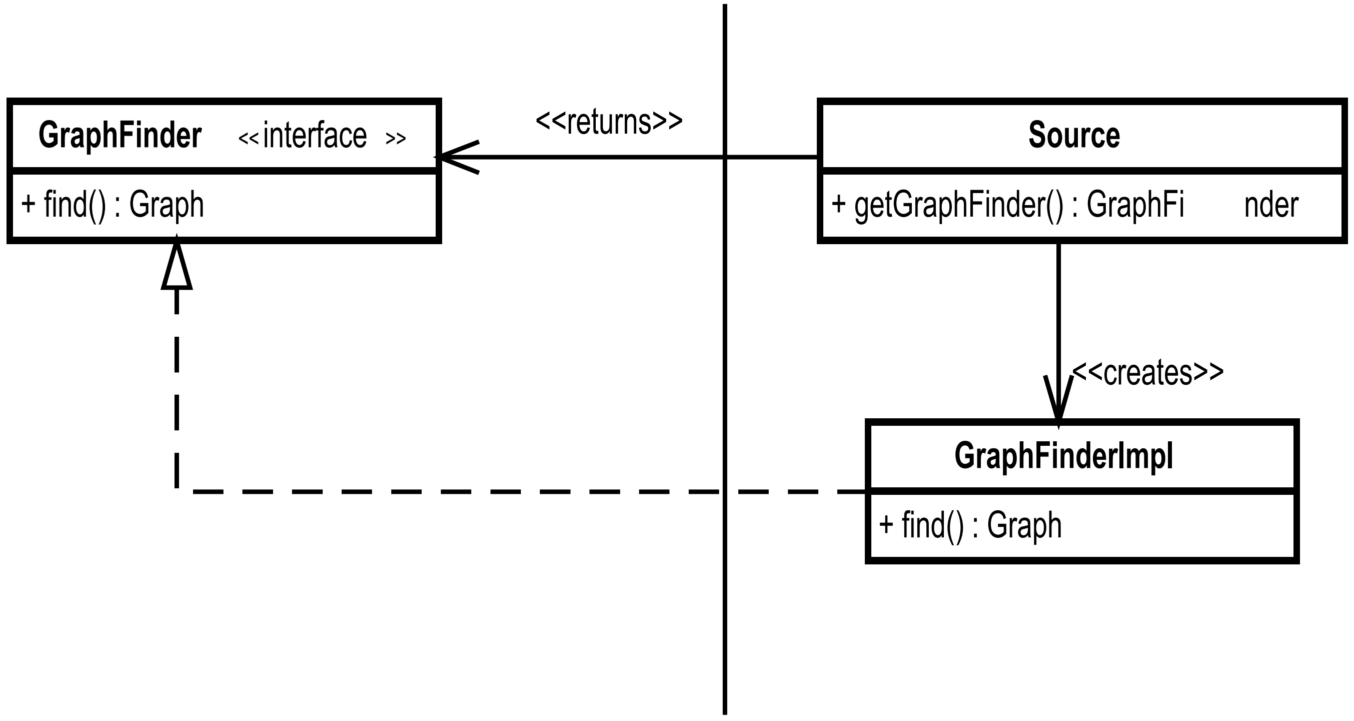
Tests that do these things aren't bad. Often they are worth writing, and they can be written in a unit test harness. However, it is important to be able to separate them from true unit tests so that we can keep a set of tests that we can run fast whenever we make our changes.

Agenda

- API Problems
- Dilemmas of API Development
- Tips and Tricks
- Language Design to the Rescue?
- What Are We Protecting, Really?

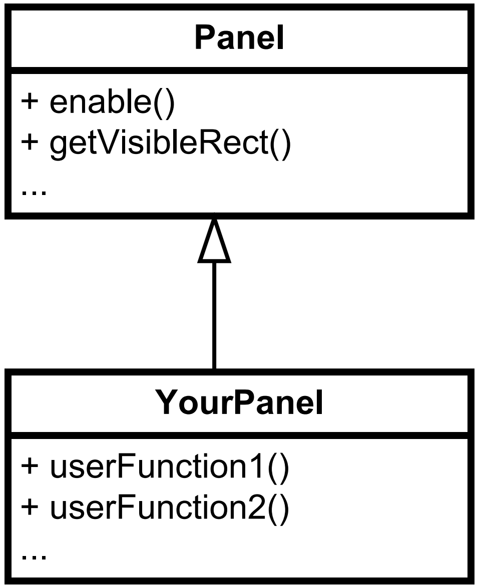
Let's look through some code..

API Anti-pattern:
Private Implementer



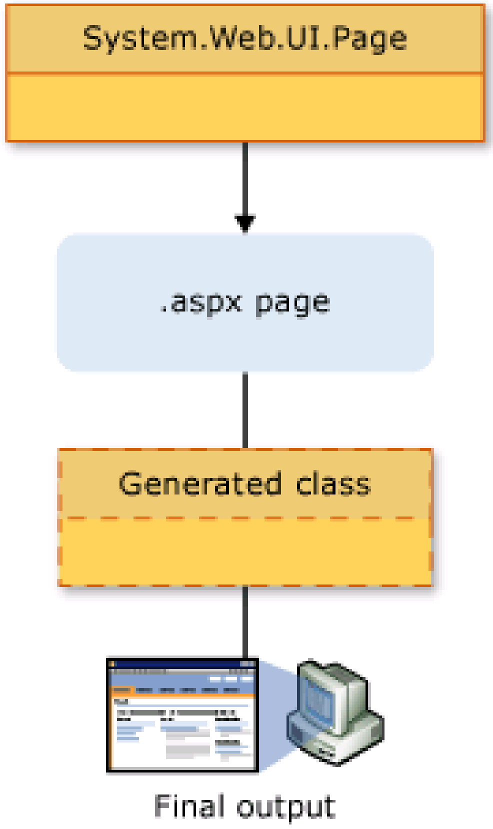
API Anti-pattern:

Partially Implemented Superclass

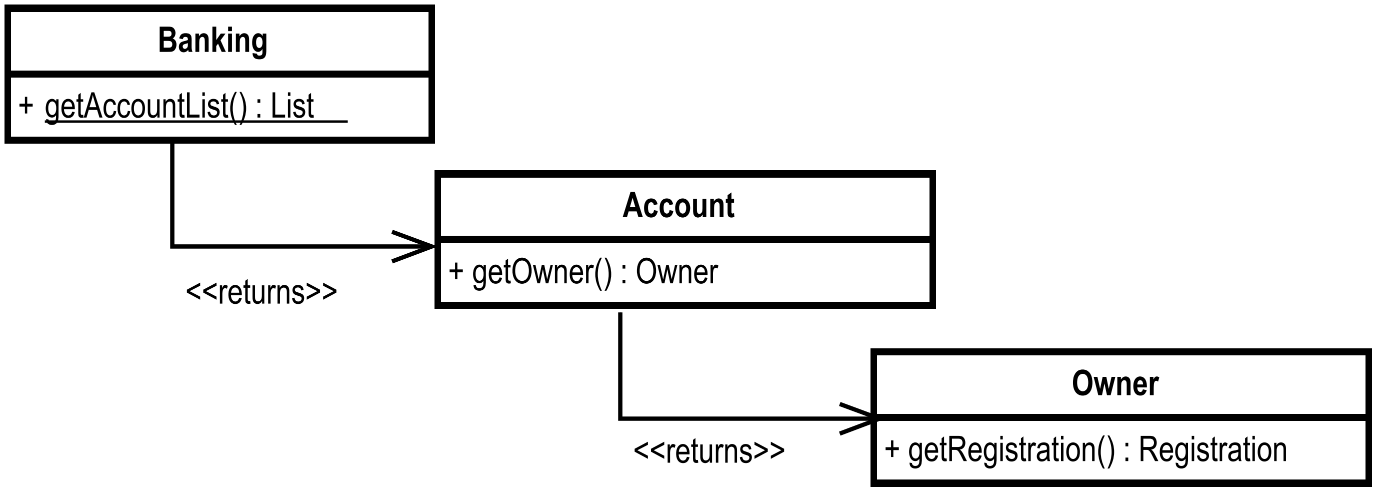


API Problems

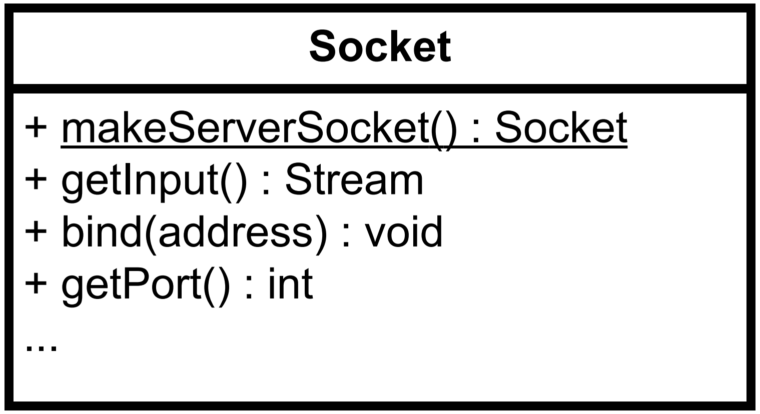
```
protected void Save_Clicked(object sender, EventArgs e)
{
  DataTable table = new DataTable();
  table.Columns.Add(
        new DataColumn("Name",   typeof(string)));
  table.Columns.Add(
        new DataColumn("Comments", typeof(string)));

  DataRow row = table.NewRow();
  row["Name"] = name.Text;
  row["Comments"] = comments.Text;
  table.Rows.Add(row);

  book.DataSource = table;
  book.DataBind();
}
```



API Anti-pattern: Object Chain



API Anti-pattern: Static Factory Method



API Anti-pattern: Irreplaceable and Untestable Behavior

```
void process(EventList& events) {
    for(EventList::iterator it = events.begin(); it != events.end(); ++it) {
        Event *event = *it;
        if (event->desc_tag == RD_TY) {
            ::stepper_write(event->range.next);
        }
        else {
            motion_control_am.sendCommand(event->range.current_action);
        }
    }
}
```

API Development is tough work:

- APIs live forever
  - Mistakes live forever
  - Early choices can make later choices impossible
- Users can misuse APIs (and blame **you**)
- Security
- API development has a high profile

Know your API. Different APIs have different requirements.

Security	Misuse Prevention	Extensibility
		✓
✓	✓	✓

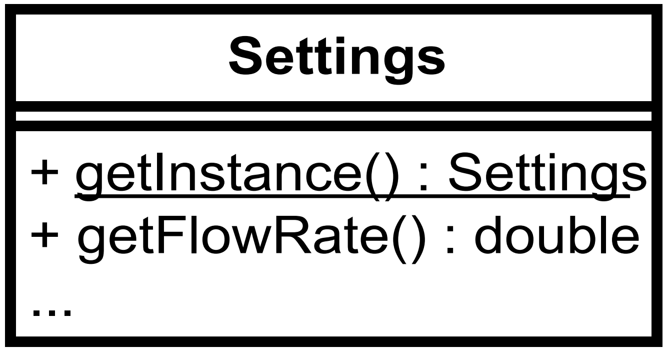
Avoid Static Methods

- Usually problematic but useful in two cases:
  1. When an operation is completely replaceable by other means
  2. When an operation will never need to be replaced

Tips and Tricks

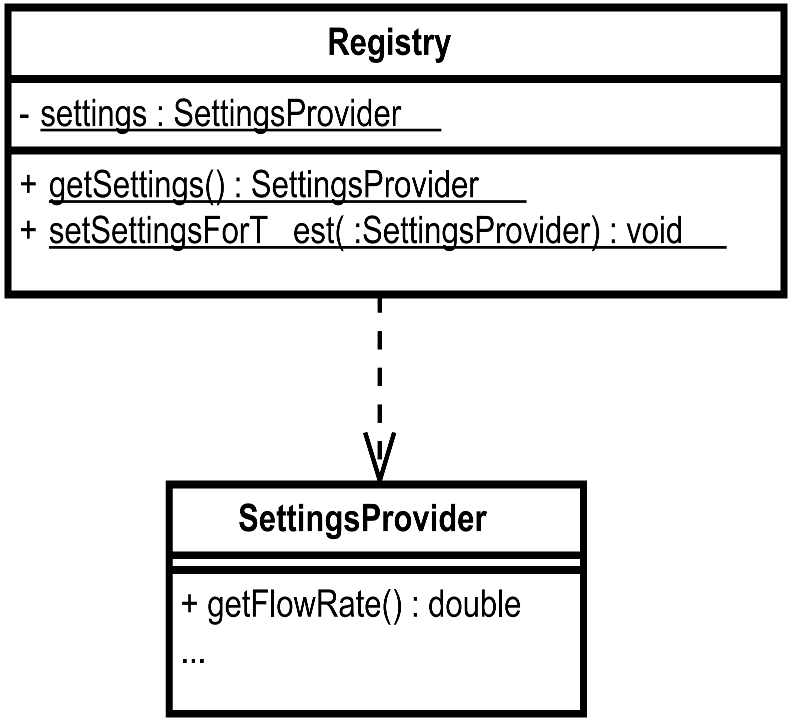
Static methods work better if you pull back one extra level of indirection..

Singleton becomes Registry [Fowler]



Tips and Tricks

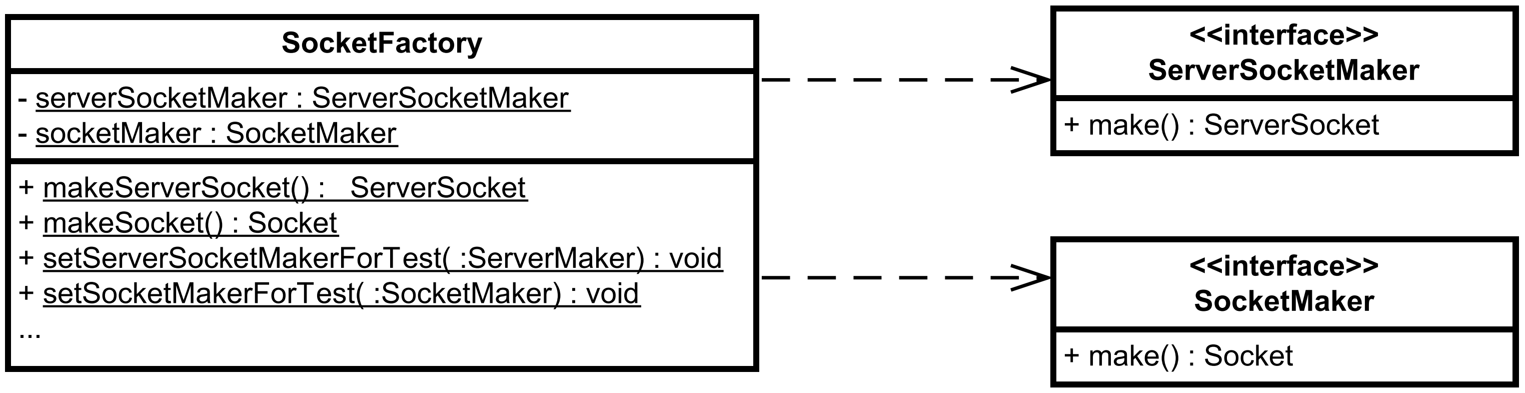
Statics move back to the registry


Registry
- settings : SettingsProvider
+ getSettings() : SettingsProvider
+ setSettingsForTest(:SettingsProvider) : void
SettingsProvider
+ getFlowRate() : double
...

Tips and Tricks

```
public class Registry {
    private static SettingsProvider settingsProvider
                = new ProductionSettingsProvider();

    public static SettingsProvider getSettings() {
        return settingsProvider;
    }

    public static void setSettingsForTest(SettingsProvider provider) {
        settingsProvider = provider;
    }
}
```

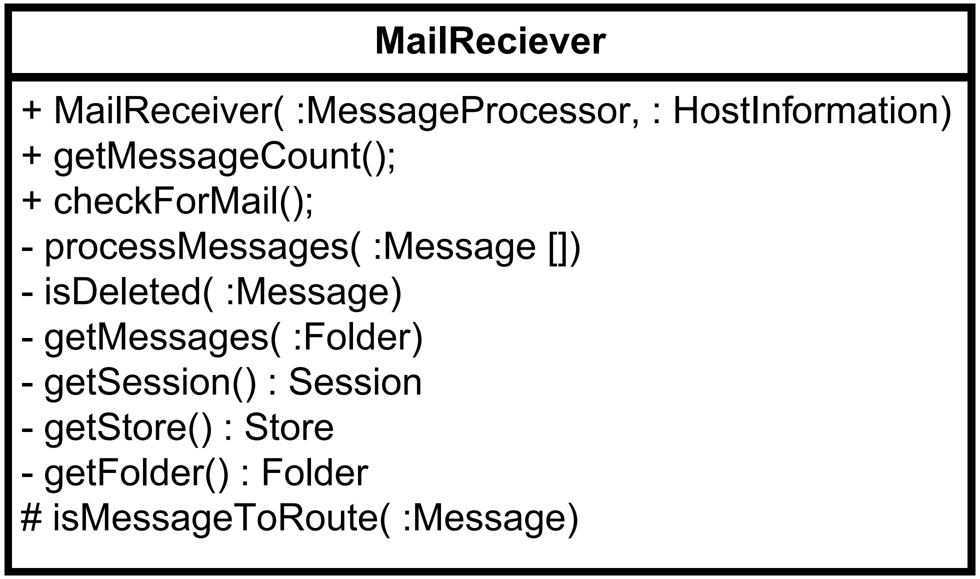
Monostate Factory


SocketFactory
- serverSocketMaker : ServerSocketMaker
- socketMaker : SocketMaker
+ makeServerSocket() : ServerSocket
+ makeSocket() : Socket
+ setServerSocketMakerForTest(:ServerMaker) : void
+ setSocketMakerForTest(:SocketMaker) : void
...
<<interface>> ServerSocketMaker
+ make() : ServerSocket
<<interface>> SocketMaker
+ make() : Socket

The ‘Envelope of Use is the Envelope of Encapsulation’

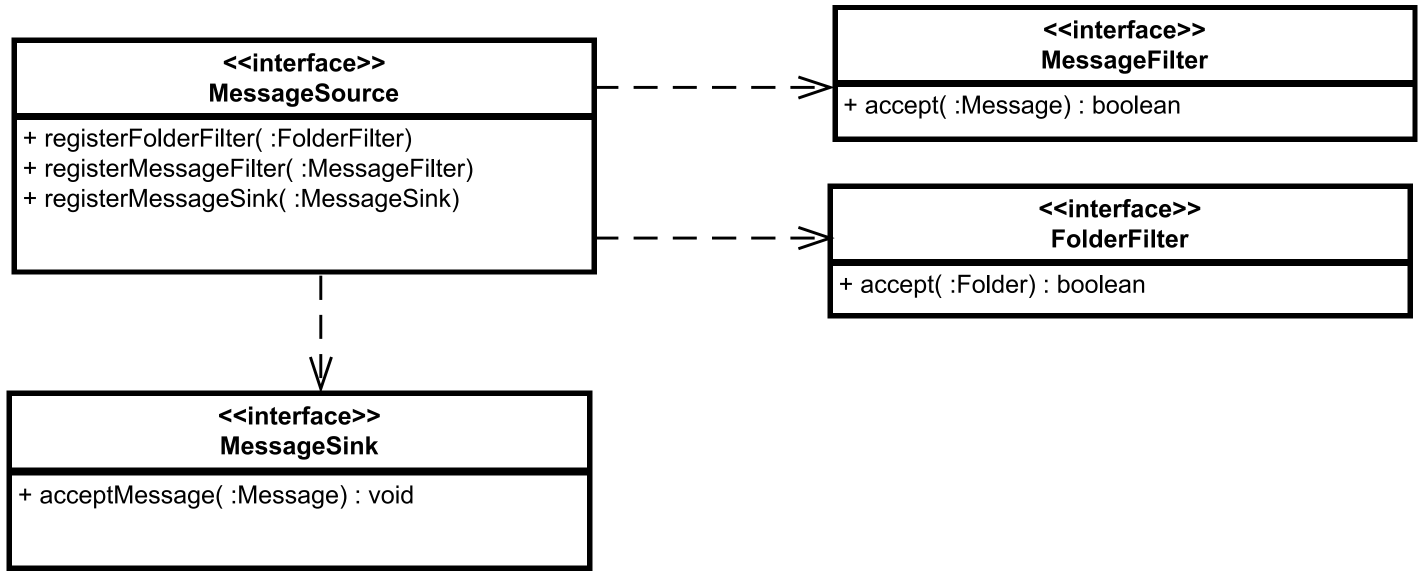
- Look at the typical usage scenarios for your API.
- Recognize that if you can’t/won’t supply mocks, people will wrap and they will wrap at the ‘envelope’ boundary

Handling Mail



```
MailReciever

+ MailReceiver( :MessageProcessor, : HostInformation)
+ getMessageCount();
+ checkForMail();
- processMessages( :Message [])
- isDeleted( :Message)
- getMessages( :Folder)
- getSession() : Session
- getStore() : Store
- getFolder() : Folder
# isMessageToRoute( :Message)
```

An alternative Mail API


<<interface>>
MessageSource
+ registerFolderFilter(:FolderFilter)
+ registerMessageFilter(:MessageFilter)
+ registerMessageSink(:MessageSink)
<<interface>>
MessageFilter
+ accept(:Message) : boolean
<<interface>>
FolderFilter
+ accept(:Folder) : boolean
<<interface>>
MessageSink
+ acceptMessage(:Message) : void

Supply Interfaces

- Interfaces in the broad sense – yes, abstract bases can be interfaces
- The concept of an interface is different in C++, C#, Java, and dynamic languages

Leave your users an “out”

- If users can’t mock your API, they’ll wrap it.
- This could be a valid API choice, but publish it, and avoid object chains.

Avoid making classes and methods *sealed* or *final* or *non-virtual*.

- unless you're sure you've provided all of the access users will need

Wouldn't all of this be solved if API designers just wrote tests for their APIs?

Sadly, no.

The Golden Rule of API Design:

“It's not enough to write tests for an API you develop, you have to write unit tests for code that uses your API.

When you do, you learn first-hand the hurdles that your users will have to overcome when they try to test their code independently.”

Supply your tests and mocks to your users

- Good APIs are tested. If you were testing, chances are you wrote or used mocks. Supply them to your users.
- Supply *your* tests to *your* users also. Why not?

If you have construction, you have everything – you can mock!

```
at acme.invoicingapp.tools.NewShipment.ship(NewShipment.java(121)
  at acme.invoicingapp.utilities.Bundler.newBundle(Bundler.java(5780)
    at acme.services.dispatchers.GroundDispatcher.dispatch(GroundDispatcher.java(56)
      {
        …
        return new RoutingDisplatcher(bundle, packet, Ship.GROUND);
        …
      }
```

Language Rescue

- Various languages have tools for deep mocking:
  - Java – AspectJ, byte-code manipulation libraries, etc
  - .NET- similar
  - C++ - (nothing)
- As an API developer remember:
  - Unit testing is too important to depend upon deep voodoo!
  - The Gulf of Practice

Protection?

You encounter code like this in the middle of an application. Your job is to get the code under control. What do you do?

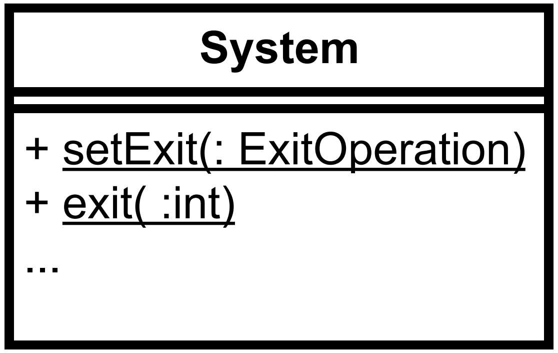
```
…
System.exit(1);
…
```

Protection?

Options

- Test with a security manager (iffy)
- Wrap the call and throw an exception

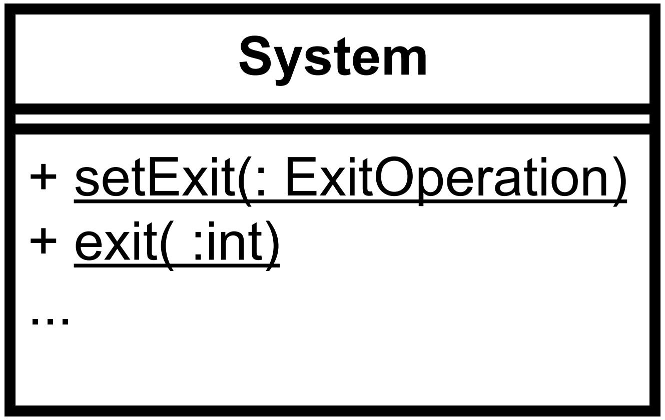
Why doesn't Java supply this?



Protection?

Madness!

- Think of security!
- Safety!
- Malicious attacks!



Protection?

In Ruby..

- Why is this different?

```
class Something
   def do_it
      exit(1)
   end
end

class Something
   def exit(value)
   end
end

# tests ..
```

Protection?

Why are you able to
open your electronics?

Your car engine?

8 TRACK
FM 88 92 96 100 102 104 106 108 MHz
MW 54 60 70 80 100 120 140 160 x10KHz
TUNING
PROGRAM
POWER
VOLUME
BALANCE
TONE
SELECTOR

The Politics of API Design

– Who is responsible when an interface changes?

The Politics of API Design

– Can you really address security with final, sealed and non-virtual functions?

The Delicious Irony of API Development

- You can sweat and toil over the design of your API but if you don't deal with testability, your best users will just wrap your API.

Resources

- Effective Java – Joshua Bloch
- Framework Design Guidelines – Cwalina, Abrams
- Test Driven Development By Example – Kent Beck
- Working Effectively with Legacy Code – Michael Feathers
- The Eclipse API Rules of Engagement
  http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/misc/api-usage-rules.html