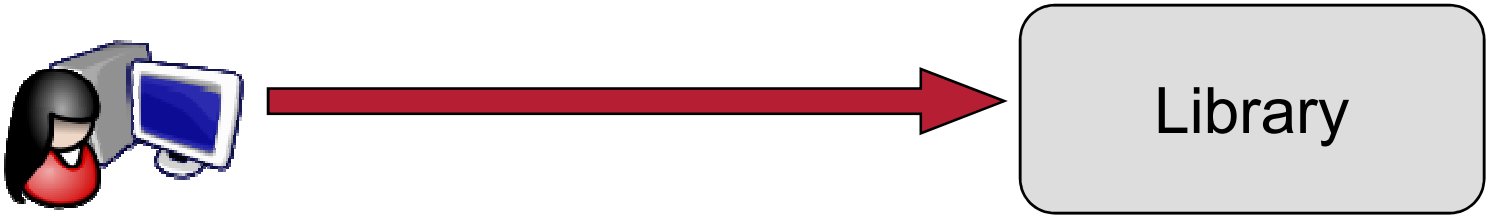
Frameworks

15-214: Principles of Software System Construction

Some material from Ciera Jaspan,
Bill Scherlis, and Erich Gamma

Terminology: Libraries

- Library: A set of classes and methods that provide reusable functionality
- Client calls library to do some task
- Client controls
  - System structure
  - Control flow
- The library executes a function and returns data



Math

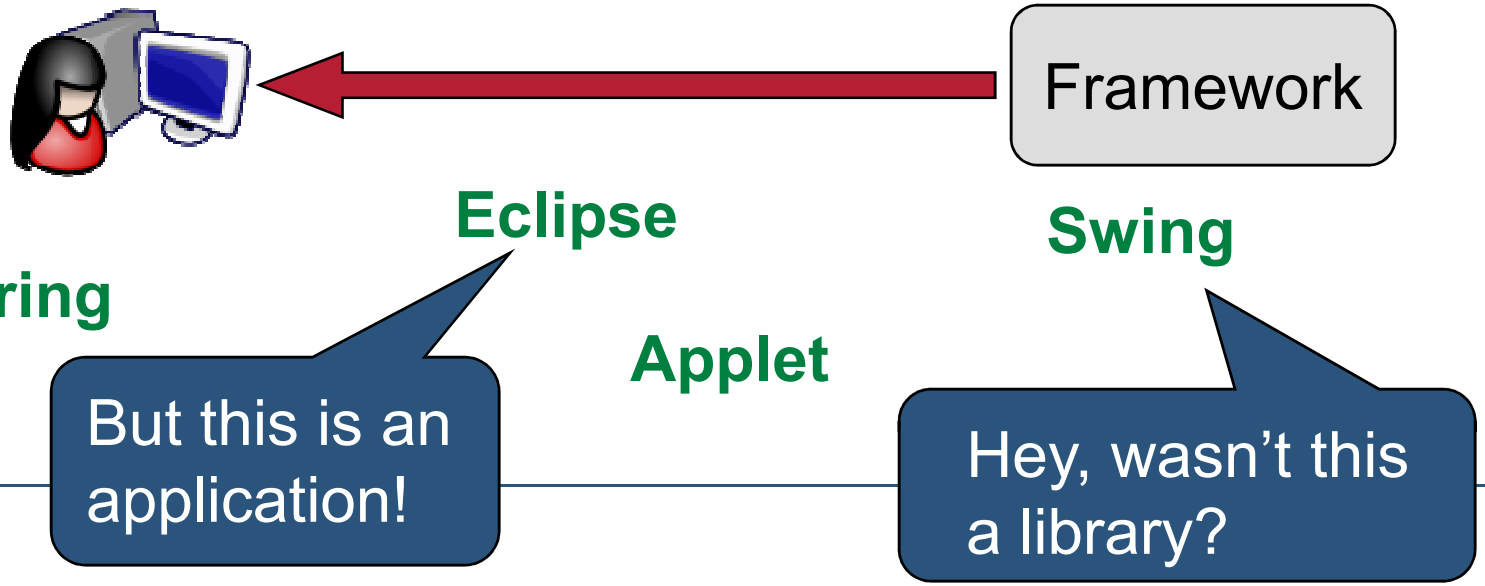
Collections

I/O

Swing

Terminology: Frameworks

- Framework: Reusable skeleton code that can be customized into an application
- Framework controls
  - Program structure
  - Control flow
- Framework calls back into client code
  - The Hollywood principle: "Don't call us; we'll call you."



More terms

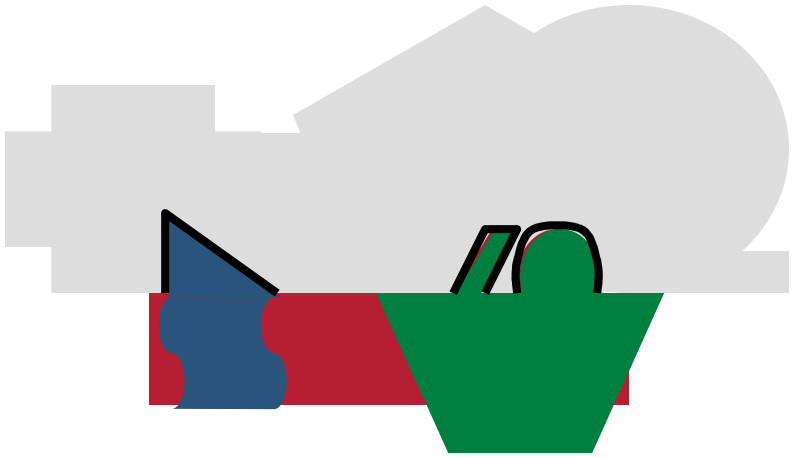
- API: Application Programming Interface, the interface of a library or framework
- Client: The code that uses an API
- Plugin: Client code that customizes a framework
- Extension point: A place where a framework supports extension with a plugin

More terms

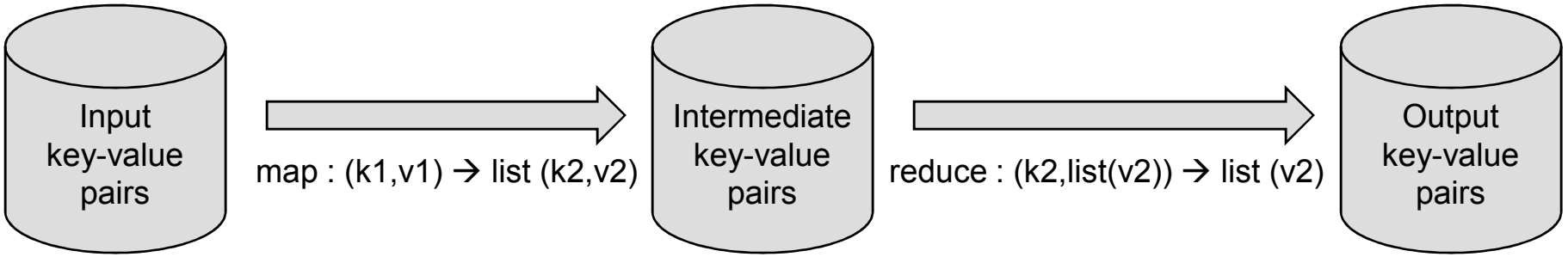
- Protocol: The expected sequence of interactions between the API and the client
- Callback: A plugin method that the framework will call to access customized functionality
- Lifecycle method: A callback method of an object that gets called in a sequence according to the protocol and the state of the plugin

Using an API

- Like a partial design pattern
- Framework provides one part
- Client provides the other part
- Very common for plugin trees to exist
- Also common for two frameworks to work better together

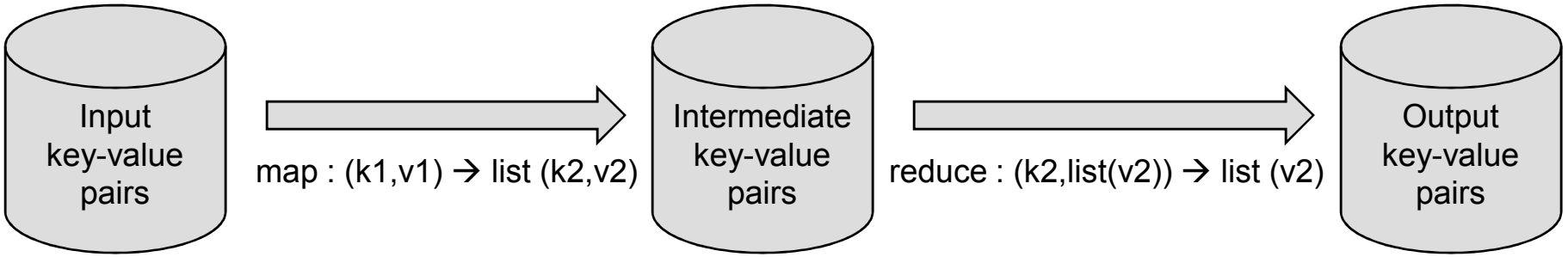


Google's Map-Reduce



- Programming model for processing large data sets
- Example: word count
  - map(document, contents):
    for each word w in document
      emit (w, 1)
  - reduce(word, listOfCounts):
    for each count c in listOfCounts
      result += c
    emit result

Google's Map-Reduce



- Questions
  - Is this a framework? How do you know?
  - What are the benefits?
  - Could those benefits be achieved if it were not?

Some Benefits of Map-Reduce

- Automatically parallelizes and distributes computation
- Scales to 1000s of machines, terabytes of data
- Automatically handles failure via re-execution
- Simple programming model
  - Successful: hundreds of plugins
  - Functional model facilitates correctness

Constraints

- Computation must fit the model
  - Not everything can be phrased in terms of map and reduce
- Map and Reduce must be largely functional
  - Side effects allowed but must be atomic and idempotent
- What benefits does the client get in exchange for accepting these restrictions?

Example: An Eclipse Plugin

- A popular Java IDE
- More generally, a framework for tools that facilitate "building, deploying and managing software across the lifecycle."

- Plug-in framework based on OSGI standard
- Starting point: Manifest file
  - Plugin name
  - Activator class
  - Meta-data

Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: MyEditor Plug-in
Bundle-SymbolicName: MyEditor; *singleton*:=true
Bundle-Version: 1.0.0
Bundle-Activator: myeditor.Activator
Require-Bundle: org.eclipse.ui,
org.eclipse.core.runtime,
org.eclipse.jface.text,
org.eclipse.ui.editors
Bundle-ActivationPolicy: lazy
Bundle-RequiredExecutionEnvironment:
JavaSE-1.6

Example: An Eclipse Plugin

- plugin.xml
  - Main configuration file
  - XML format
  - Lists extension points
- Editor extension
  - extension point: org.eclipse.ui.editors
  - file extension
  - icon used in corner of editor
  - class name
  - unique id
    - refer to this editor
    - other plugins can extend with new menu items, etc.!

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>

  <extension
        point="org.eclipse.ui.editors">
    <editor
          name="Sample XML Editor"
          extensions="xml"
          icon="icons/sample.gif“
          contributorClass="org.eclipse.ui.texteditor.Basic
          TextEditorActionContributor"
        class="myeditor.editors.XMLEditor"
        id="myeditor.editors.XMLEditor">
    </editor>
  </extension>

</plugin>
```

Example: An Eclipse Plugin

- At last, code!
- XMLEditor.java
  - Inherits TextEditor behavior
    - open, close, save, display, select, cut/copy/paste, search/replace, …
    - REALLY NICE not to have to implement this
    - But could have used ITextEditor interface if we wanted to
  - Extends with syntax highlighting
    - XMLDocumentProvider partitions into tags and comments
    - XMLConfiguration shows how to color partitions

```
package myeditor.editors;

import org.eclipse.ui.editors.text.TextEditor;

public class XMLEditor extends TextEditor {
    private ColorManager colorManager;

    public XMLEditor() {
        super();
        colorManager = new ColorManager();
        setSourceViewerConfiguration(
                new XMLConfiguration(colorManager));
        setDocumentProvider(
                new XMLDocumentProvider());
    }

    public void dispose() {
        colorManager.dispose();
        super.dispose();
    }
}
```

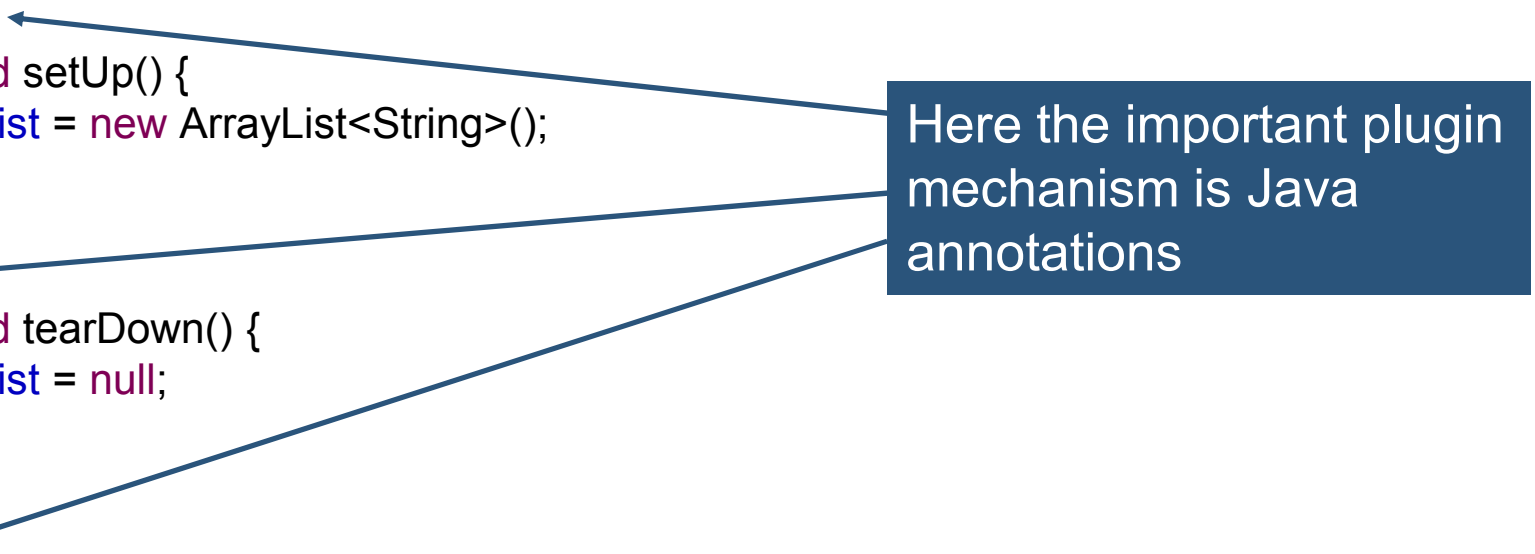
Example: a JUnit Plugin

```
public class SampleTest  {
    private List<String> emptyList;

    @Before
    public void setUp() {
        emptyList = new ArrayList<String>();
    }

    @After
    public void tearDown() {
        emptyList = null;
    }

    @Test
    public void testEmptyList() {
        assertEquals("Empty list should have 0 elements",
                0, emptyList.size());
    }
}
```



The Golden Rule of Framework Design

- Extending the framework should NOT require modifying the framework source code!

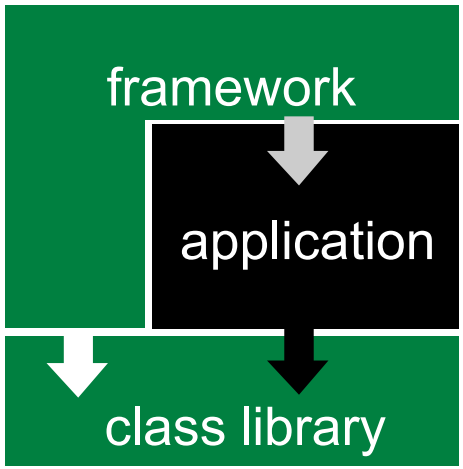
- Discussion: how can we extend without modification?
  - Client writes main(), creates a plugin, and passes it to framework
  - Framework writes main(), client passes name of plugin
    - E.g. using a command line argument or environment variable

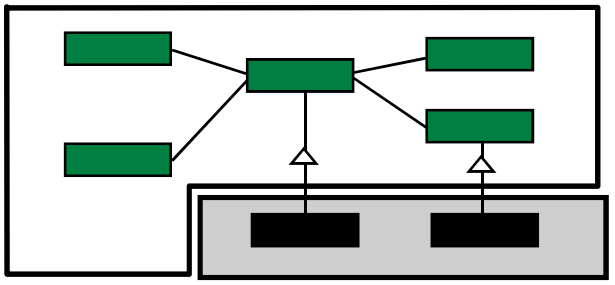
    ```
    Class c = ClassLoader.getSystemClassLoader().loadClass(args[0]);
    Plugin p = c.newInstance();
    ```

  - Framework looks in a magic location
    - Config files or JAR files there are automatically loaded and processed

OO Frameworks (credit: Erich Gamma)

- A customizable set of cooperating classes that defines a reusable solution for a given problem
  - defines key abstractions and their interfaces
  - object interactions
    - invariants
  - flow of control
    - override and be called
  - defaults
- Reuse
  - reuse of design and code
  - reuse of a macro architecture
- Framework provides architectural guidance



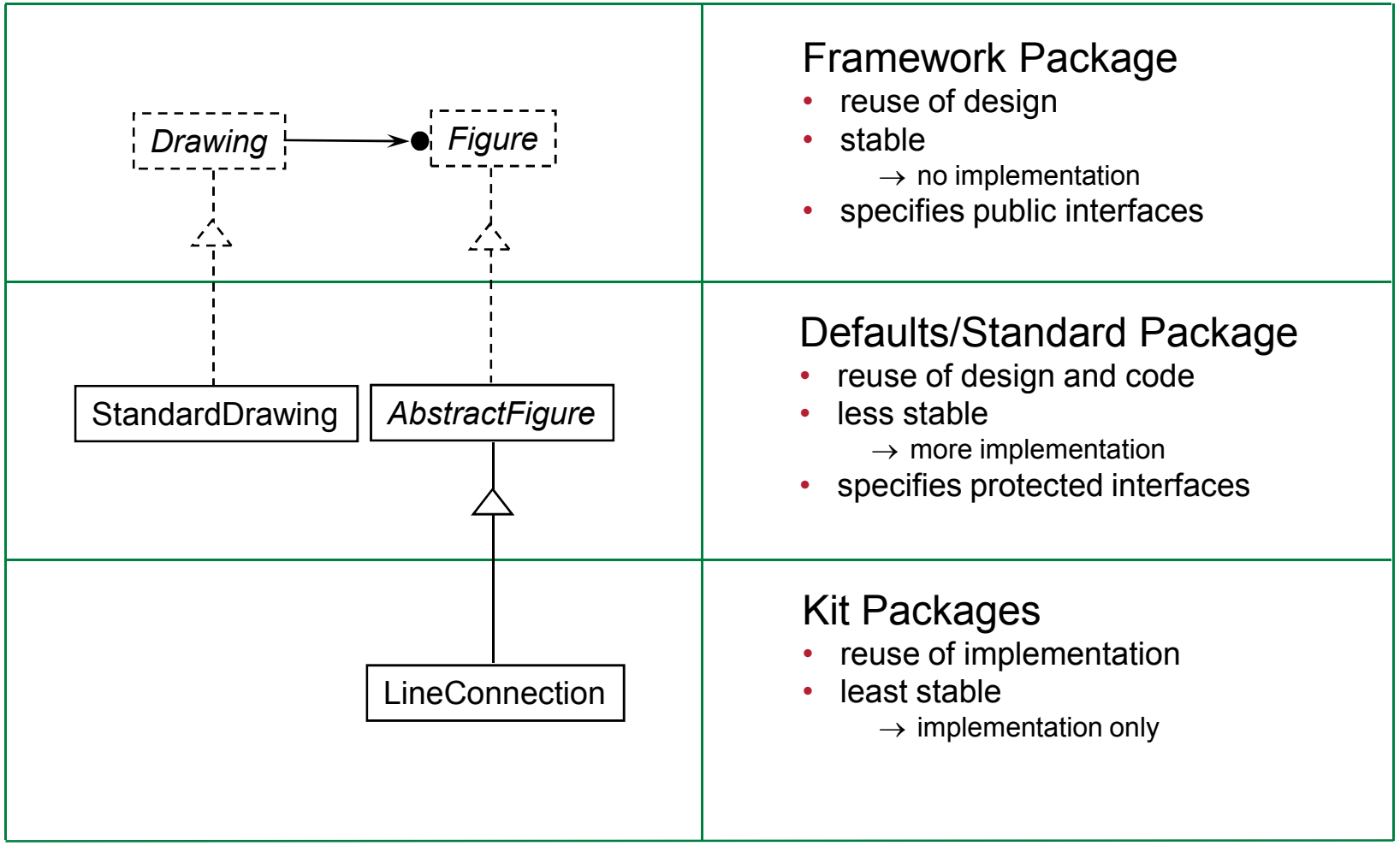


reusing a framework

Framework Challenges (credit: Erich Gamma)

- frameworks are hard to maintain
- framework enables reuse of both design and implementation
  - easy for clients to add implementation dependencies
  - "what is the framework - what is just default implementation"
- therefore:
  - separation of design from implementation
    "we believe that **interface design and functional factoring constitute the key intellectual content of software** and that they are far **more difficult to create** or re-create than code" -- Peter Deutsch
  - late commitment to implementation
    - but, frameworks still have to work out of the box!

Framework Layering (credit: Erich Gamma)



Evolution: Extract Interface from Class

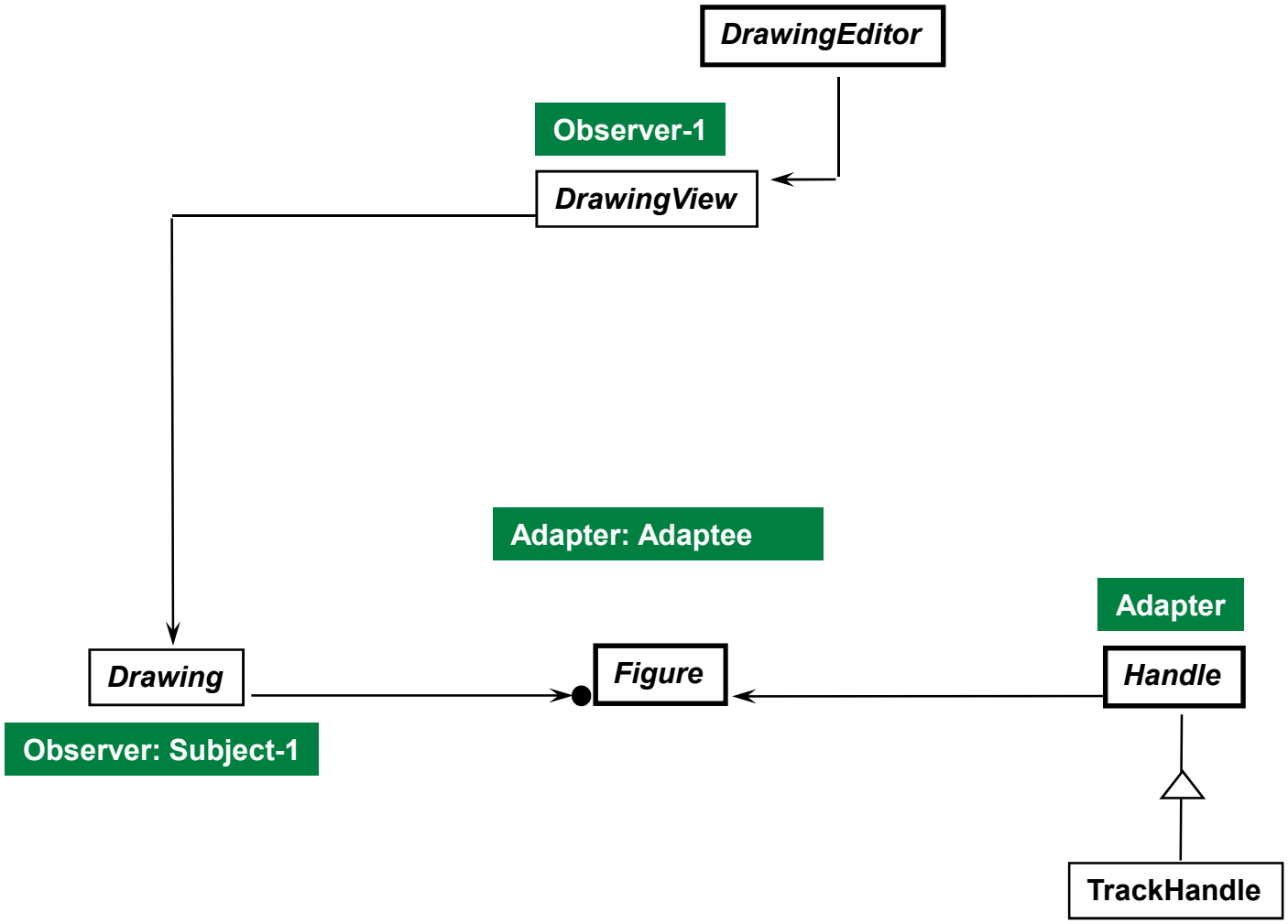
(credit: Erich Gamma)

⇒JHotDraw defines framework abstractions as interfaces

- extracting interfaces is a new step in evolutionary design
  - abstract classes are **discovered** from concrete classes
  - interfaces are **distilled** from abstract classes
- start once the architecture is stable!
- remove non-public methods from class
- move default implementations into an abstract class which implements the interface

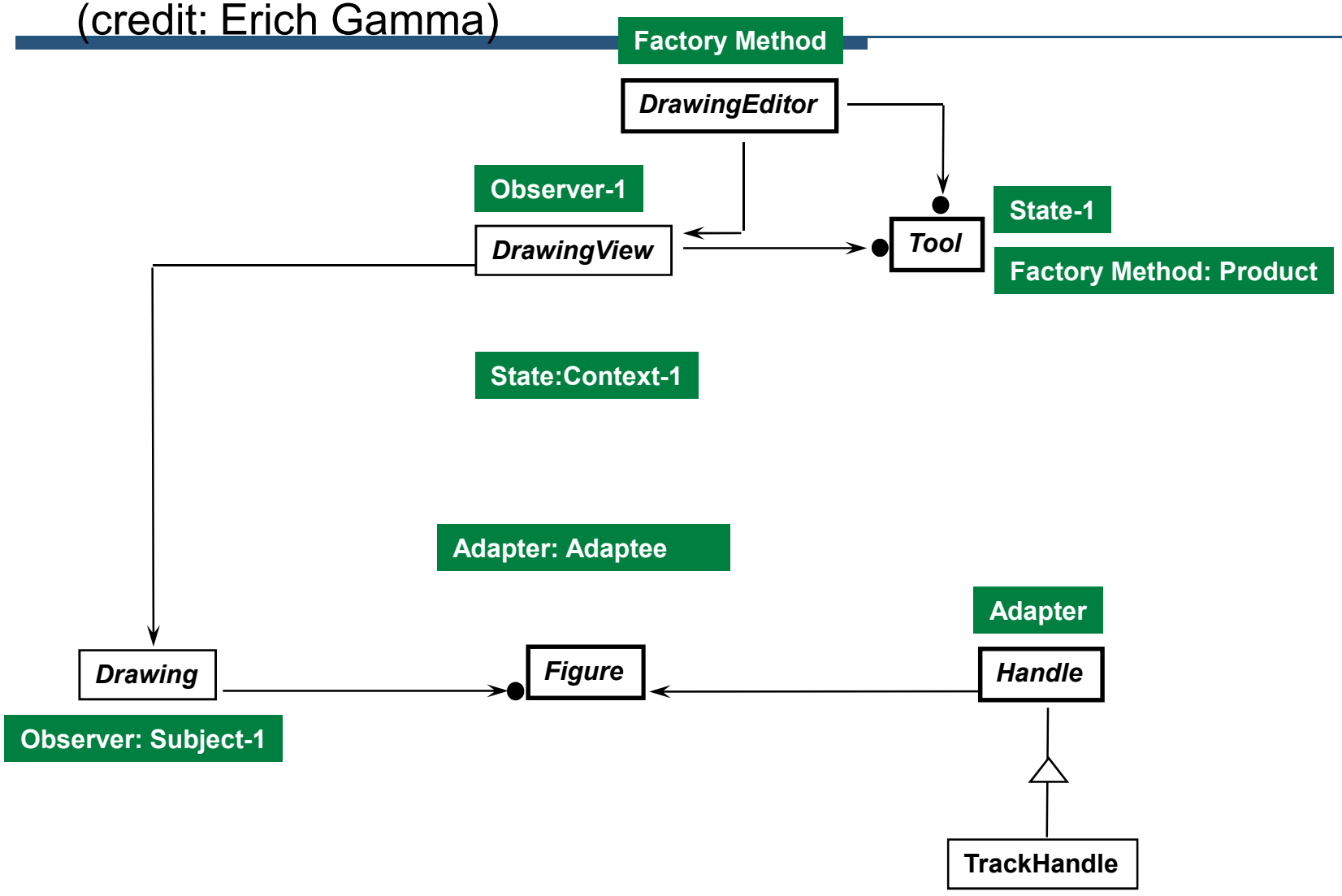
JHotDraw: Design Patterns Summary

(credit: Erich Gamma)



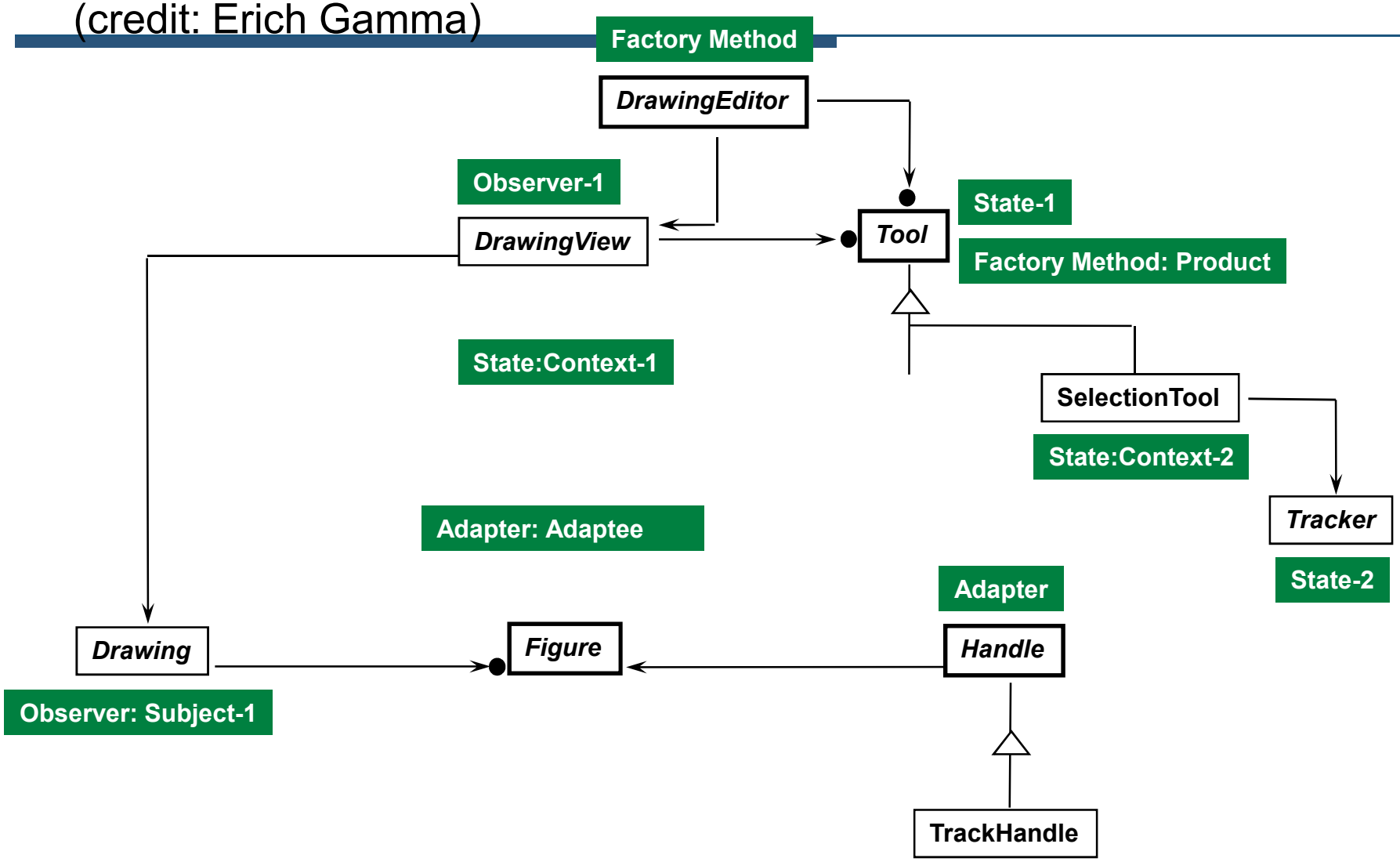
JHotDraw: Design Patterns Summary

(credit: Erich Gamma)



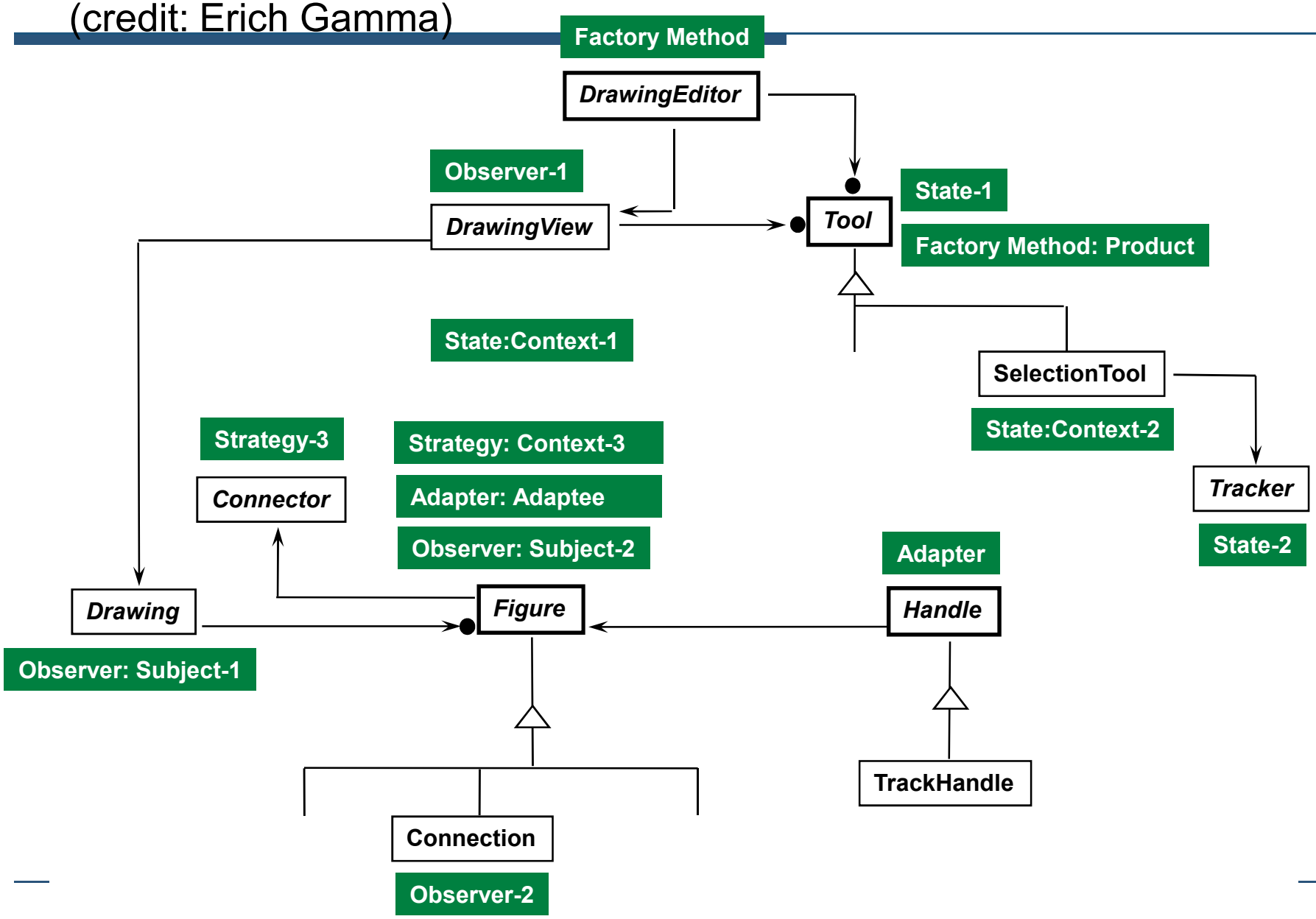
JHotDraw: Design Patterns Summary

(credit: Erich Gamma)



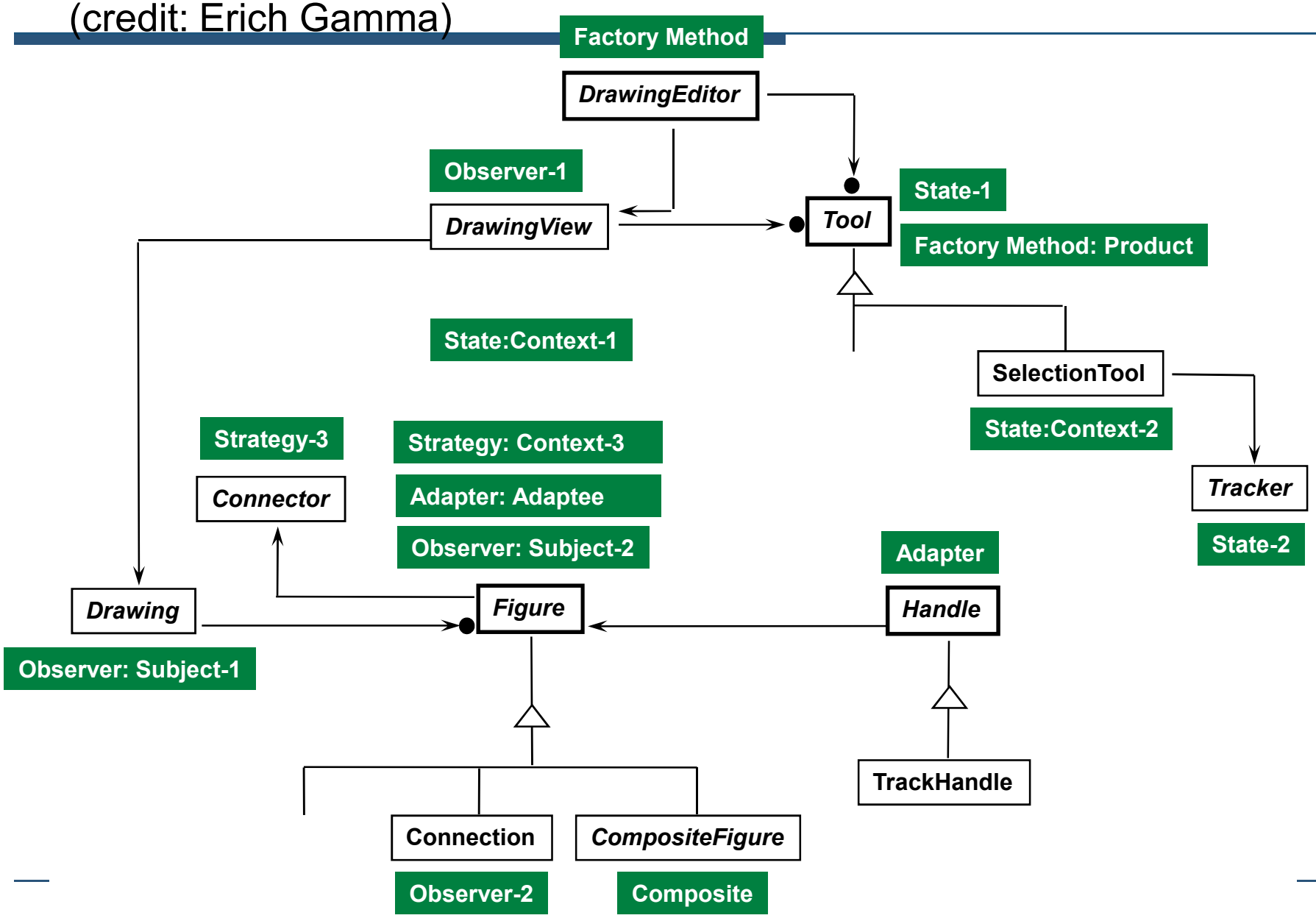
JHotDraw: Design Patterns Summary

(credit: Erich Gamma)



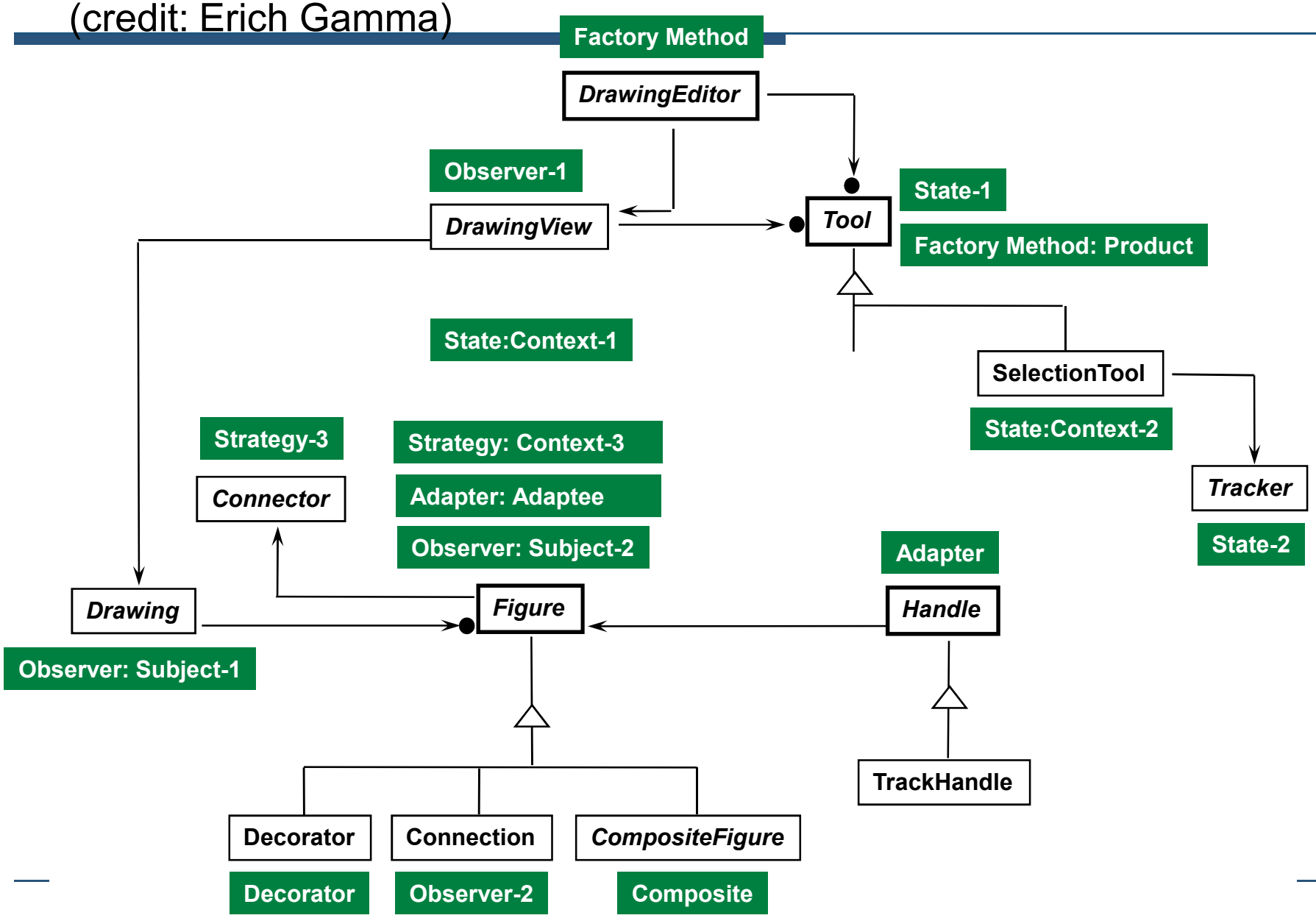
JHotDraw: Design Patterns Summary

(credit: Erich Gamma)



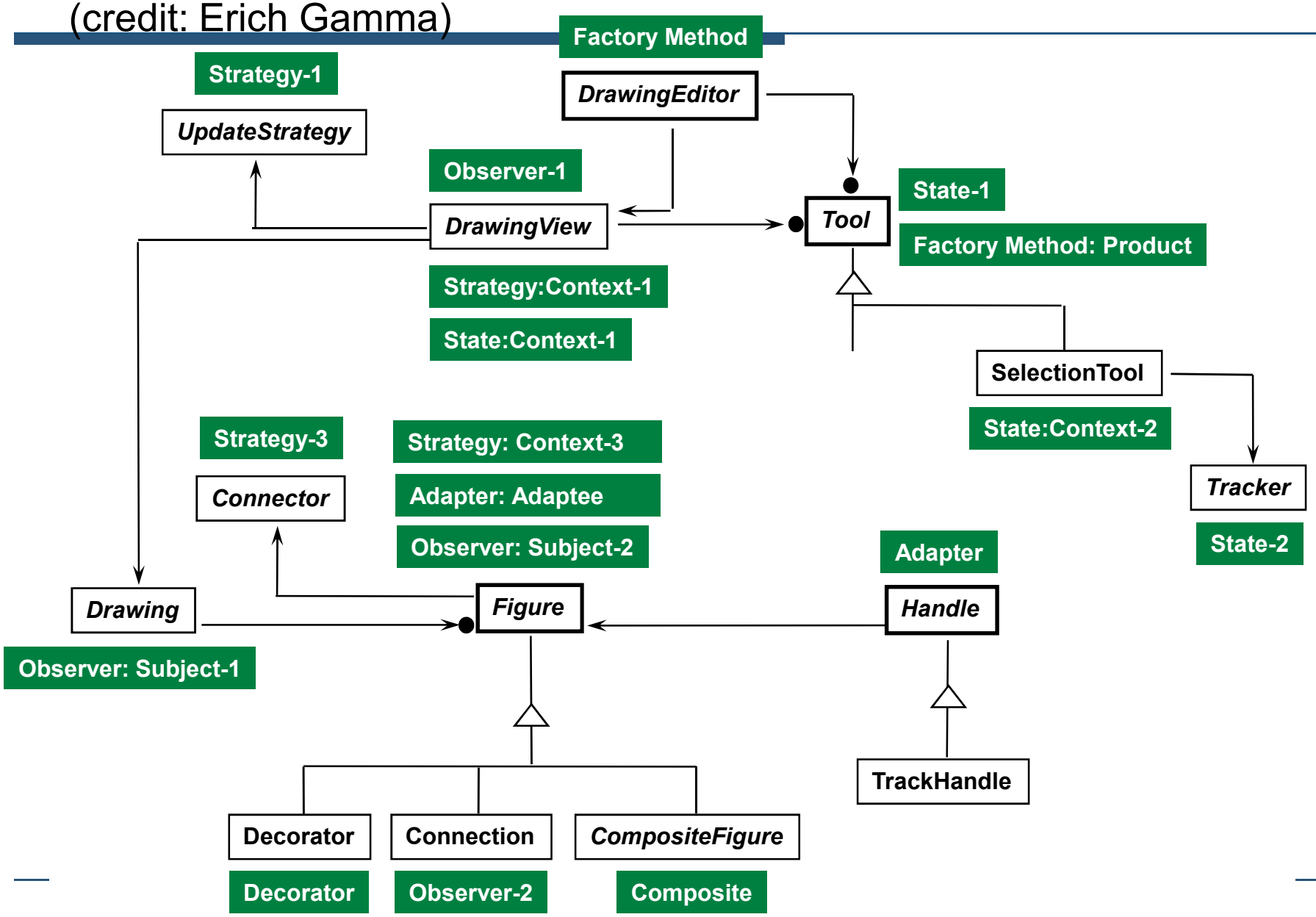
JHotDraw: Design Patterns Summary

(credit: Erich Gamma)



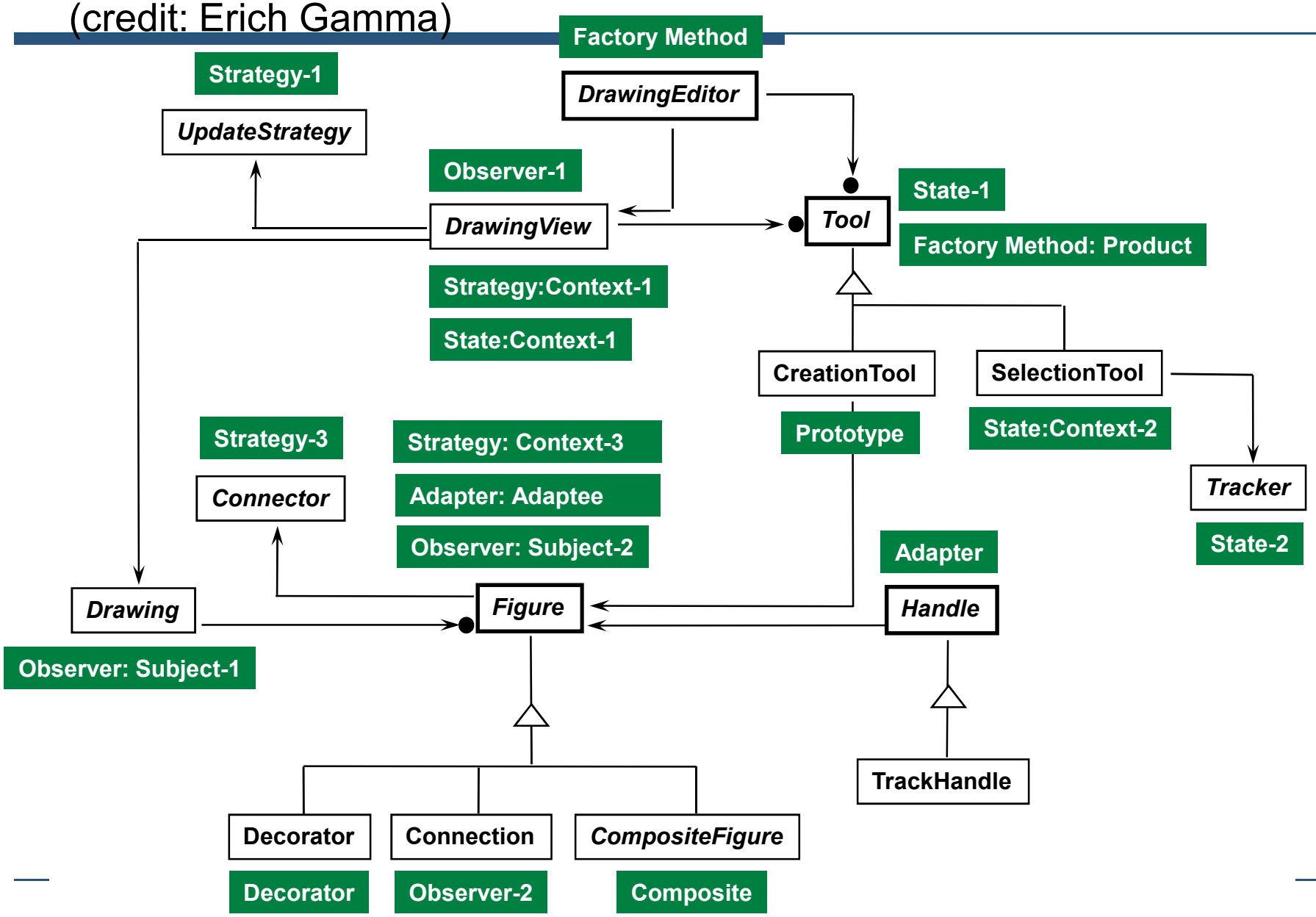
JHotDraw: Design Patterns Summary

(credit: Erich Gamma)



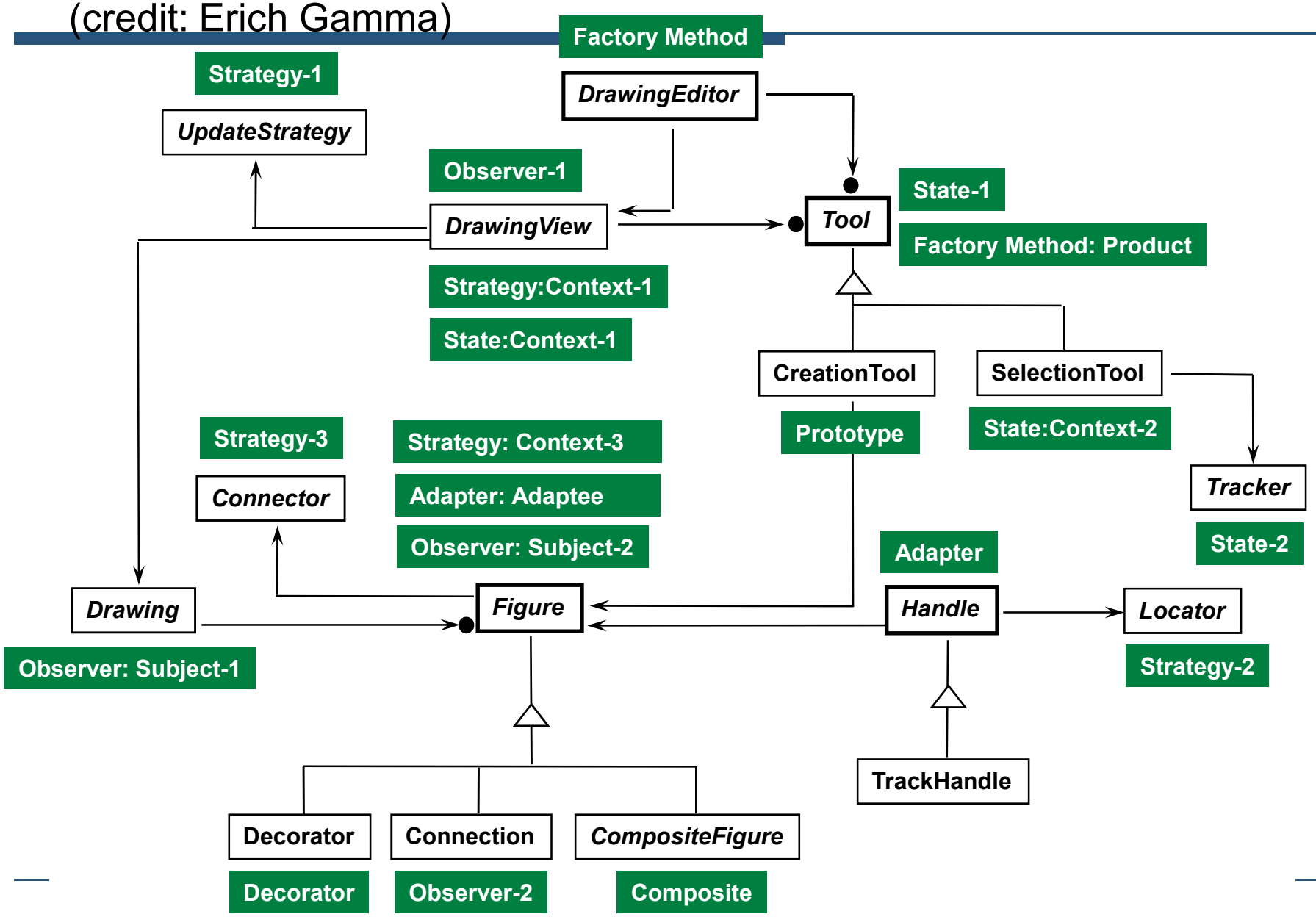
JHotDraw: Design Patterns Summary

(credit: Erich Gamma)



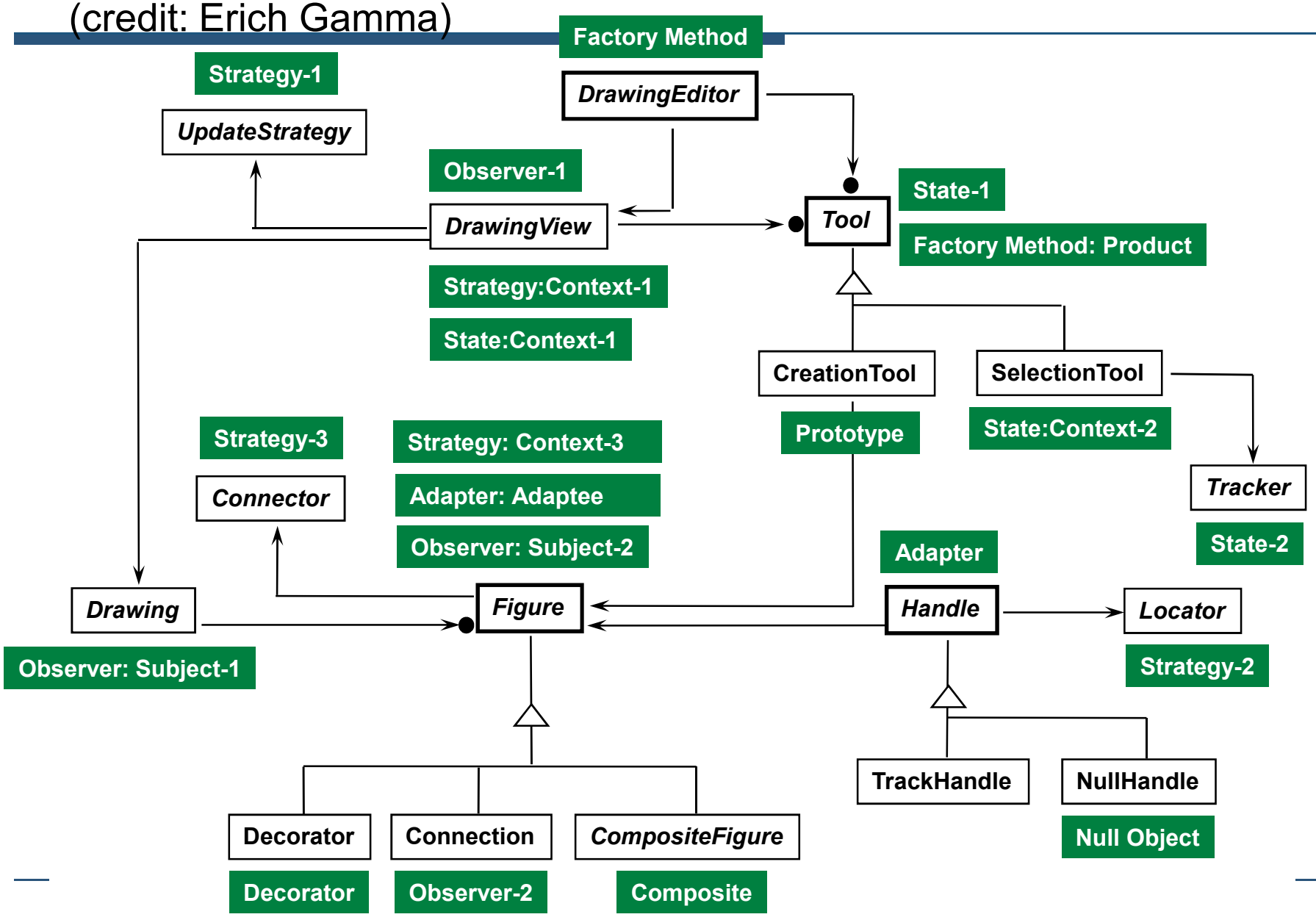
JHotDraw: Design Patterns Summary

(credit: Erich Gamma)



JHotDraw: Design Patterns Summary

(credit: Erich Gamma)



JHotDraw Pattern Experiences

(credit: Erich Gamma)

- Increased design velocity
  - patterns helped us generate the architecture
- It wasn't always clear which pattern to apply
  - patterns can be competitors
  - implementing the patterns is easy
  - difficulty is knowing when and why to use them!
- Framework development remains iterative
  - design patterns are targets for refinements and refactoring
- JavaDoc can be used to document the applied patterns
  - javadoc comments may include URLs
  - URLs refer to a pattern description or *patlet*
- JHotDraw: http://sourceforge.net/projects/jhotdraw

Callback challenges

- Simple ASP.NET Page with a drop down list
  - Derive from Page
  - Add the controls
  - Handle any user actions on controls
- 10 simple Page callbacks
  - Many more complex ones
- Where do we add the controls?
- When can I access data?
- Where does the framework expect it to happen?

- **`PreInit`**
- **`Init`**
- **`InitComplete`**
- **`PreLoad`**
- **`Load`**
- **`Control events…`**
- **`LoadComplete`**
- **`PreRender`**
- **`SaveStateComplete`**
- **`Render`**
- **`Close`**

Dynamically add a control to the page

```
private void Page_Load(object sender, EventArgs e) {

   DropDownList ddl = new DropDownList();
   ddl.DataSource = …; //accesses another control
   ddl.DataBind();
   addControl(ddl);
}
```

Whoops! Resets the initial data every time, so we lose the user's selection.

Dynamically add a control to the page, attempt 2

```
private void Page_Load(object sender, EventArgs e) {

   if (!IsPostBack()) {
      DropDownList ddl = new DropDownList();
      ddl.DataSource = …; //accesses another control
      ddl.DataBind();
      addControl(ddl);
   }
}
```

Ok, now the control entirely disappears when the page refreshes after an action (the postback)….

Dynamically add a control to the page, attempt 3

```
private void Page_Load(object sender, EventArgs e) {

   DropDownList ddl = new DropDownList();
   if (!IsPostBack()) {
      ddl.DataSource = …;
      ddl.DataBind();
   }
   addControl(ddl);
}
```

Ok, the control is there, but there's no data in it after an update/refresh….

Dynamically add a control to the page, attempt 4

```
private void Page_PreInit(object sender, EventArgs e) {

   DropDownList ddl = new DropDownList();
   if (!IsPostBack()) {
      ddl.DataSource = …; //accesses another control
      ddl.DataBind();
   }
   addControl(ddl);
}
```

Now we get a null reference exception when accessing that other control's data…

Dynamically add a control to the page, attempt 5

```
DropDownList ddl;

private void Page_PreInit(object sender, EventArgs e) {
   ddl = new DropDownList();
   addControl(ddl);
}

private void Page_Load(object sender, EventArgs e) {
   if (!IsPostBack()) {
      ddl.DataSource = …; //accesses another control
      ddl.DataBind();
   }
}
```

Finally it works!

Couldn't they design it better?

- Could have fewer callbacks
  - But it would make it less extensible
- In some cases, could give better errors and warnings
  - But it would give up performance
- Some design choices could map to the developer's mind more easily
  - But we might lose other quality attributes, like security

Interaction is not limited to your primary code!

- Many methods of interacting with a framework
- Declarative files, such as XML or properties files
- Annotations within code

- If the functionality is supported by all, which do I choose?
- And what happens if they are conflicting?

Choosing an interaction

- Example 1: Internationalization
  - Properties files or directly in code?
- Example 2: Transactions
  - XML file, annotations, or in code?
- Example 3: Database URL
  - XML file, properties file, annotation, or in code?
- Notice that the choice affects how easy the code is to read, how difficult it is to change later, and who can do the change!

Putting controls in a LoginView

- Can specify different controls to be shown when a user is logged in
  - Ex: username and password fields v. "Welcome, Username!"

```
<asp:LoginView ID="LoginScreen" runat="server">
  <AnonymousTemplate>
    You can only setup accounts when you are logged in.
  </AnonymousTemplate>
  <LoggedInTemplate>
    <h4>Location</h4>
    <asp:DropDownList ID="LocationList" runat="server"/>
    <asp:Button ID="ChangeButton" runat="server" Text="Change"/>
  </LoggedInTemplate>
</asp:LoginView>
```

Retrieve controls and set them up

```
LoginView LoginScreen;

private void Page_Load(object sender, EventArgs e) {

   DropDownList list = (DropDownList)
    LoginScreen.FindControl("LocationList");

   list.DataSource = …;
   list.DataBind();
}
```

NullReferenceException at list.DataSource = …;

Correct code

```
LoginView LoginScreen;

private void Page_Load(object sender, EventArgs e) {
   if (this.getRequest().IsAuthenticated()) {
      DropDownList list = (DropDownList)
       LoginScreen.FindControl(“LocationList”);

      list.DataSource = …;
      list.DataBind();
   }
}
```

These sound tough to use…why bother?

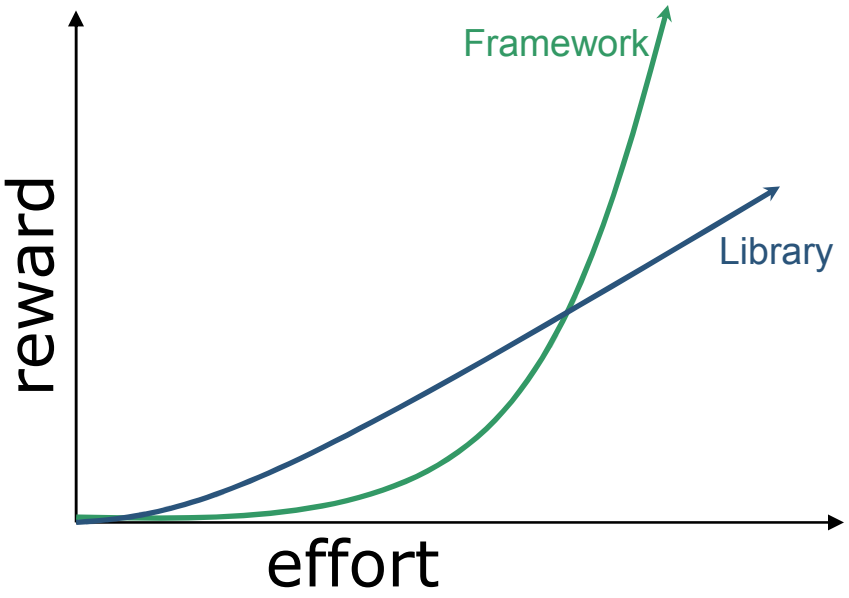
- Code reuse
  - Eclipse framework: ~2,000,000 LoC
  - Eclipse plugin: 12 LoC
  - … of course you need to know which 12 lines to write
- Maintainability
- Existing knowledge of employees
- External community support
- Large-scale (architectural) reuse
- Built-in quality attributes

Frameworks and Quality Attributes

- Quality attributes
  - Performance
  - Security
  - Scalability
  - *-ility
- All QA's have tradeoffs with each other
- Old way: hack quality attributes in after development
- New way: Embed quality attributes into the framework
  - More cost effective, less refactoring
  - Handled at high level, not scattered in program
- Works if you know your QA tradeoffs up front
  - This is why those requirements are so important…

Getting up a framework's learning curve

- Tips on using frameworks
  - Tutorials, Wizards, and Examples
    - SourceForge, Google Code Search
  - Communities – email lists and forums
    - Eclipse.org
  - Group knowledge dispersal
    - Wiki of resources, Problem/solution log



- Common client trick: Follow the leader
  - Appropriate code from examples – find an "imputed pattern"
    - Search source code
    - Infer compatible intent
    - Identify scope (not too much, not too little)
  - Copy it
    - Tear out the app-specific logic, keep the bureaucracy
    - Insert your own logic into the reused bureaucracy
  - But there's a problem
    - Classic copy-and-paste problem – looks just like my own code
    - Design intent is lost – "my intention is to use the framework this way"

- Framework designer's conundrum: complexity vs. capability

Choosing a framework

- Business objectives
- Existing software lock-in
- Ability to match quality attributes and tradeoff decisions
- Costs of learning
- Costs of purchase (or maintenance for homegrown)

Do we build it ourselves?

- Outsourcing the framework
  - Examples: Eclipse, J2EE, ASP.Net, etc
  - Benefits: lower risk, high reuse, community support
  - Costs/risks: compromise of control
- Insourcing the framework
  - Examples: product-line frameworks
  - Benefits: economy of scale, control over system
  - Costs/risks: building and maintenance, requires experts