Learning goals

- Describe example well-known example frameworks
- Know key terminology related to frameworks
- Know common design patterns in different types of frameworks
- Discuss differences in design trade-offs for libraries vs. frameworks
- Analyze a problem domain to define commonalities and extension points (cold spots and hot spots)
  - Analyze trade-offs in the use vs. reuse dilemma
- Know common framework implementation choices

This and next lecture

- Design for reuse: Libraries and frameworks
  - Motivation: reuse with variation
  - Examples, terminology
  - Whitebox and blackbox frameworks
  - Design considerations
  - Implementation details
    - Responsibility for running the framework
    - Loading plugins

Reuse and variation

Reuse and variation: The Standard Widget Toolkit
The promise:

Earlier in this course: Class-level reuse

- Design for Reuse
- Language mechanisms supporting reuse
  - Inheritance
  - Subtype polymorphism (dynamic dispatch) for delegation
  - Parametric polymorphism (generics)
- Design principles supporting reuse
  - Small interfaces
  - Information Hiding
  - Low coupling
  - High cohesion
- Design patterns supporting reuse
  - Template method, decorator, strategy, composite, adapter, ...

Approaches to reuse and variation

- "Clone and own"
- Subroutines
- Libraries
- Frameworks
- APIs
- Platforms
- Configuration
- Software product lines

LIBRARIES AND FRAMEWORKS

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

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.”
Consider a family of programs consisting of the following:

- Initialization of GUI
- Layout
- Main method
- What source code might be shared?

**A simple example framework**

- Consider a family of programs consisting of buttons and text fields only:

  - What source code might be shared?
    - Main method
    - Initialization of GUI
    - Layout
    - Closing the window

**A calculator example (without a framework)**

```java
import javax.swing.*; // GUI components
import java.awt.*; // AWT components
import java.awt.event.*; // Event handling

public class Calc extends JFrame {
    protected JTextField textfield;

    public Calc() { init(); }

    public static void main(String[] args) {
        new Calc().setVisible(true);
    }

    protected void init() {
        // GUI initialization
    }
}
```

**A simple example framework**

```java
import javax.swing.*; // GUI components
import java.awt.*; // AWT components
import java.awt.event.*; // Event handling

public class Application extends JFrame {
    private JTextField textfield;

    public Application() { init(); }

    public static void main(String[] args) {
        new Application().setVisible(true);
    }

    protected void init() {
        // GUI initialization
    }
}
```

**A simple example framework**

```java
import javax.swing.*; // GUI components
import java.awt.*; // AWT components
import java.awt.event.*; // Event handling

public class Application extends JFrame {
    private JTextField textfield;

    public Application() { init(); }

    public static void main(String[] args) {
        new Application().setVisible(true);
    }

    protected void init() {
        // GUI initialization
    }
}
```

**A simple example framework**

```java
import javax.swing.*; // GUI components
import java.awt.*; // AWT components
import java.awt.event.*; // Event handling

public class Application extends JFrame {
    private JTextField textfield;

    public Application() { init(); }

    public static void main(String[] args) {
        new Application().setVisible(true);
    }

    protected void init() {
        // GUI initialization
    }
}
```

**A simple example framework**

```java
import javax.swing.*; // GUI components
import java.awt.*; // AWT components
import java.awt.event.*; // Event handling

public class Application extends JFrame {
    private JTextField textfield;

    public Application() { init(); }

    public static void main(String[] args) {
        new Application().setVisible(true);
    }

    protected void init() {
        // GUI initialization
    }
}
```

**A simple example framework**

```java
import javax.swing.*; // GUI components
import java.awt.*; // AWT components
import java.awt.event.*; // Event handling

public class Application extends JFrame {
    private JTextField textfield;

    public Application() { init(); }

    public static void main(String[] args) {
        new Application().setVisible(true);
    }

    protected void init() {
        // GUI initialization
    }
}
```

**A simple example framework**

```java
import javax.swing.*; // GUI components
import java.awt.*; // AWT components
import java.awt.event.*; // Event handling

public class Application extends JFrame {
    private JTextField textfield;

    public Application() { init(); }

    public static void main(String[] args) {
        new Application().setVisible(true);
    }

    protected void init() {
        // GUI initialization
    }
}
```
A customizable set of cooperating plugins.

**The Java Logging Framework**

- Provides flow of control
- Provides defaults
- Reuse
  - Reuse of design and code
  - Reuse of a macro architecture
- Framework provides architectural guidance

**OO frameworks**

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

**Framework or Library?**

- **Java Collections**
- **Eclipse**
- **The Java Logging Framework**
- **Java Encryption Services**
- **Wordpress**
- **Ruby on Rails**
- **Homework 1 Graph Implementation**

**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, hot spot**: A place where a framework supports extension with a plugin

**General distinction: Library vs. Framework**

- **Library**: your code interacts
- **Framework**: your code

**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
Platform/software ecosystem

- Hardware/software environment (frameworks, libraries) for building applications
- Ecosystem: Interaction of multiple parties on a platform, third-party contributions, co-dependencies,
  - Typically describes more business-related and social aspects

Blackbox frameworks

- Extension via implementing a plugin interface
- Common design pattern(s):
  - Strategy
  - Observer
- Design steps:
  - Ecosystem: Interaction of multiple parties on a platform, extension via implementing a plugin interface
  - Plugin: Abstract variable code as methods of an interface
  - Common design pattern(s):
    - Identifier the common code and the variable code
    - Extension via subclassing and overriding
  - Decide whether there might be one or multiple plugs
  - Plugin-loading mechanism loads plugins and gives control to the framework

Whitebox frameworks

- Extension via subclassing and overriding methods
- Common design pattern(s):
  - Template Method
- Design steps:
  - Identify the common code and the variable code
  - Abstract variable code as method calls
  - Subclass has main method but gives control to framework

Is this a whitebox or blackbox framework?

```java
public abstract class AbstractClass {
    public abstract String getInititalText();
    public abstract String getButtonText();
    public abstract String getApplicationTitle();
}
```

An example blackbox framework

```java
public class MyBlackBoxFramework {
    public void loadPlugin(Plugin plugin) {
        this.plugin = plugin;
        plugin.setApplication(this);
    }
}
```

```java
public interface Plugin {
    void setApplication(Application app);
    void buttonClicked();
    String getInititalText();
    String getButtonText();
    String getApplicationTitle();
}
```
An example blackbox framework

```java
private class Application extends JFrame {
    protected void init() {
        this.application = app; 
    }
    @Override
    public static void main(String[] args) {
        new Application(new CalcPlugin()).setVisible(true); 
    }
}
```

An aside: Plugins could be reusable, too...

```java
public interface Plugin {
    String getButtonText();
    String getInititalText();
    void buttonClicked();
    public interface InputProvider {
        public String getInput();
    }
    public static void main(String[] args) {
        // Framework code
    }
}
```

Whitebox vs. blackbox framework summary

- **Whitebox frameworks use subclassing**
  - Allows to extend existing method
  - Need to understand implementation of superclass
  - Only one extension at a time
  - Compiled together
  - Often so-called developer frameworks
- **Blackbox frameworks use composition**
  - Allows to extend only functionality exposed in interface
  - Only need to understand the interface
  - Multiple plugins
  - Often provides more modularity
  - Separate deployment possible (.jar, .dll, ...)
  - Often so-called end-user frameworks, platforms

Scrabble Framework

- In what way is Homework 4 (Scrabble with Stuff) a framework?

Framework design considerations

- Once designed there is little opportunity for change
- Key decision: Separating common parts from variable parts
  - Identify hot spots vs. cold spots
  - Too few extension points: Limited to a narrow class of users
  - Too many extension points: Hard to learn, slow
  - Too generic: little reuse value
- The golden rule of framework design:
  - Writing a plugin/extension should NOT require modifying the framework source code

The cost of changing a framework

Consider adding an extra method. Many changes require changes to all plugins.
The use vs. reuse dilemma

- Large rich components are very useful, but rarely fit a specific need
- Small or extremely generic components often fit a specific need, but provide little benefit

“maximizing reuse minimizes use”

C. Szyperski

Domain engineering

- Understand users/customers in your domain
  – What might they need? What extensions are likely?
- Collect example applications before starting a framework/component
- Make a conscious decision what to support
  – Called scoping
- e.g., the Eclipse policy:
  – Interfaces are internal at first
    • Unsupported, may change
  – Public stable extension points created when there are at least two distinct customers

Typical framework design and implementation

- Identify common parts and variable parts
- Implement common parts
  – Also design and write sample plugins/applications
- Provide plugin interface/extension/callback mechanisms for variable parts
  – Use well-known design principles and patterns: Strategy, Decorator, Observer, Command, Template Method, Factories...
**Evolutionary design: Extract interfaces from classes**

- 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

---

**Methods to load plugins**

- Client writes main(), creates a plugin, and passes it to framework
  - (see blackbox example above)
- Framework writes main(), client passes name of plugin as a command line argument or environment variable
  - (see next slide)
- Framework looks in a magic location
  - Config files or .jar files are automatically loaded and processed
- GUI for plugin management

---

**Running a framework**

- Some frameworks are runnable by themselves
  - e.g. Eclipse
- Other frameworks must be extended to be run
  - MapReduce, Swing, Servlets, JUnit

---

**FRAMEWORK MECHANICS**

---

**An example plugin loader using Java Reflection**

```java
public static void main(String[] args) {
    if (args.length != 1)
        System.out.println("Plugin name not specified");
    else {
        String pluginName = args[0];
        try {
            Class<?> pluginClass = Class.forName(pluginName);
            new Application((Plugin) pluginClass.newInstance()).setVisible(true);
        } catch (Exception e) {
            System.out.println("Cannot load plugin " + pluginName + ", reason: " + e);
        }
    }
}
```

---

**Another plugin loader using Java Reflection**

```java
public static void main(String[] args) {
    File config = new File("config");
    BufferedReader reader = new BufferedReader(new FileReader(config));
    Application application = new Application();
    Line line = null;
    while ((line = reader.readLine()) != null) {
        try {
            Class<?> pluginClass = Class.forName(pluginName);
            application.addPlugin((Plugin) pluginClass.newInstance());
        } catch (Exception e) {
            System.out.println("Cannot load plugin " + pluginName + ", reason: " + e);
        }
    }
    reader.close();
    application.setVisible(true);
}
```
GUI-based plugin management

Supporting multiple plugins
- Observer design pattern is commonly used
- Load and initialize multiple plugins
- Plugins can register for events
- Multiple plugins can react to same events
- Different interfaces for different events possible

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.

Example: An Eclipse plugin
- public class Application {
  private List<Plugin> plugins;
  public Application(List<Plugin> plugins) {
    this.plugins = plugins;
  }
  public Message processMsg(Message msg) {
    for (Plugin plugin: plugins) {
      msg = plugin.processMsg(msg);
    }
    return msg;
  }
}

Example: A JUnit Plugin
- At last, code!
  - XMLEditor.java
    - inherits TextEditor behavior
    - open, close, save, display, select, cut/copy/paste, search/replace,
    - REALLY nice to have for our editor
    - extends with syntax highlighting
      - XMLDocumentProvider partitions into tags and comments
      - XMLConfiguration shows how to color partitions

Example: An Eclipse plugin
- A popular Java IDE
- More generally, a framework for tools that facilitate "building, deploying and managing software across the lifecycle.
- Plugin framework based on OSGI standard
- Starting point: Manifest file
  - Plugin name
  - Activator class
  - Meta-data

Example: An Eclipse plugin
- 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: A JUnit Plugin
- @Before
  - public void setUp() {
    emptyList = new ArrayList<String>();
  }
- @After
  - public void tearDown() {
    assertEquals(0, emptyList.size());
  }
- @Test
  - public void testEmptyList() {
    assertEmptyList("Empty list should have 0 elements.");
  }
**SWING DISCUSSION**

**Java Swing: It’s a library?**

- Create a GUI using pre-defined containers
  - JFrame, JPanel, JDialog, JMenuBar
- Use a layout manager to organize components in the container
- Add pre-defined components to the layout
  - Components: JLabel, JTextField, JButton

  This is no different that the File I/O library.

**Swing: Layout managers**

```
panel.setLayout(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
// create and position the button
JButton button = new JButton("Click Me!");
c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 0; // first column
c.gridy = 1; // second row
c.gridwidth = 2; // span two columns
c.weightx = 1.0; // use all horizontal space
c.anchor = GridBagConstraints.WEST;
c.insets = new Insets(0,5,0,5); // add side padding
panel.add(button, c);
```

**Swing: Events**

```
// create an anonymous MouseAdapter, which extends // the MouseListener class
button.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        System.err.println("You clicked me! " + 
        "Do it again!");
    }
});
```

**Where is the boundary?**

```
Container
  └── EventListener
      └── MouseAdapter
          └── JComponent
              └── JPanel
                  └── JButton
                      ├── MyWidget
                      └── $1
```

But this extending a class to add custom behaviors, right?
Swing: Custom components

```java
public MyWidget extends JPanel {
    public MyWidget(int param) {
        setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        add(label, c);
        add(textfield, c);
        addButton, c);
    }

    public void setParameter(int param) {
        // update the widget, as needed
    }
}
```

Swing: Custom components

```java
public MyWidget extends JContainer {
    public MyWidget(int param) {
        // setup internals, without rendering
    }

    protected void paintComponent(Graphics g) {
        Dimension d = getSize();
        g.setColor(Color.red);
        g.drawRect(0, 0, d.getWidth(), d.getHeight());
    }
}
```

Recall event-based programming

- A style of programming where the control-flow of the program is driven by (usually-) external events

Learning a framework

- Documentation
- Tutorials, wizards, and examples
- Communities, email lists and forums
- Other client applications and plugins

Framework Costs

DESIGN CHALLENGES
Framework design exercises

- Think about a framework for:
  - Video playing software
  - Viewing, printing, editing a portable document format
  - Compression and archiving software
  - Instant messaging software
  - Music editing software

- Questions
  - What are the dimensions of variability/extensibility?
  - What interfaces would you need?
  - What are the core methods for each interface?
  - How do you set up the framework?

Summary

- Reuse and variation essential
  - Avoid reimplementing from scratch
- Object-oriented design principles for library design
- From low-level code reuse to design/behavior reuse with frameworks
- Design for reuse with domain analysis: find common and variable parts
- Use design patterns for framework design and implementation