

Principles of Software Construction: Objects, Design, and Concurrency

Part 2: Designing (sub-) systems

Design for large-scale reuse: Libraries and frameworks

Charlie Garrod

Chris Timperley



Administrivia

- Homework 4b due Thursday
- Homework 4a feedback still available
 - Can regain up to 75% of lost Homework 4a credit
 - Directly address TA comments when you turn in Homework 4c
 - Turn in revised design documents + scans of our feedback + description of what you changed
- Next required reading due Tuesday
 - Effective Java, Items 51, 60, 62, and 64



https://commons.wikimedia.org/wiki/File:1_carcassonne_aerial_2016.jpg

Key concepts from last Thursday

Key concepts from last Thursday

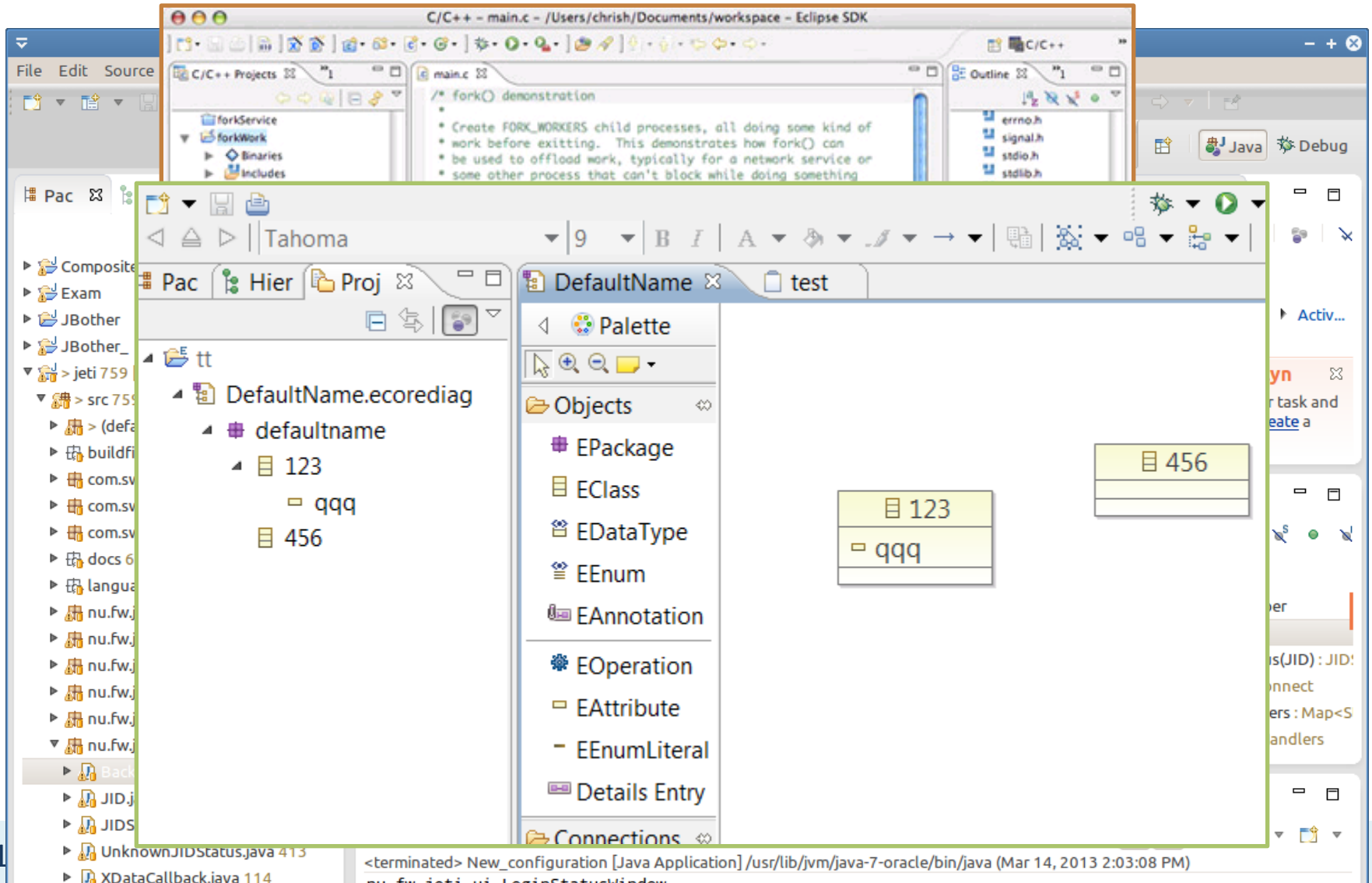
- Java Collections
 - Design patterns to achieve various design goals
 - Iterator to abstract internal structure
 - Decorator to alter behavior at runtime
 - Template method and factory method to support customization
 - Adapter to convert between implementations
 - Strategy pattern for sorting
 - Marker interface to refine a specification
 - For widespread use:
 - Design for extensibility, reuse
 - Design for change
 - Prelude to API design

Learning goals for today

- 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

Today: Libraries and frameworks for reuse

Reuse and variation: Family of development tools



Reuse and variation: Eclipse Rich Client Platform

ForeFlight

File Window Help

All Airports

- TX
- UT
- VA
- VI
- VT
- WA
- WI
 - KAIG - Antigo, WI
 - KATW - Appleton, WI
 - KASX - Ashland, WI
 - KDLL - Baraboo, WI
 - KOVS - Boscobel, WI
 - KBUU - Burlington, WI
 - KVOK - Camp Douglas, WI
 - KCLI - Clintonville, WI
 - KEGV - Eagle River, WI
 - KEAU - Eau Claire, WI
 - KFLD - Fond Du Lac, WI
 - KGRB - Green Bay, WI
 - KHYR - Hayward, WI
 - KJVL - Janesville, WI
 - KUNU - Juneau, WI
 - KENW - Kenosha, WI
 - KLSE - La Crosse, WI
 - KRCX - Ladysmith, WI

Weather Details

Airport: DANE COUNTY REGIONAL-TRUAX FIELD

Observations/Forecasts: Thurs Feb 16 9:53 AM EST

Alerts

- Winds are close to set limit of 16 kts
- Visibility is below set limit of 3 SM
- Minimum cloud layer height worse than set limit of 1000 feet

Weather Conditions

Conditions are... **LIFR**

Ceiling below 500 and/or Visibility below 1

Weather Report

Airport: DANE COUNTY REGIONAL-TRUAX FIELD
ID: KMSN

Status: Wx Report download successful

Report Date: Feb 16, 2006 9:53:00 AM (22 minutes ago)

Report Period: Observed at Thurs Feb 16 9:53 AM EST

Wind Speed: 15.0 kts

Wind Direction (mag): 20°

Temperature: 24.8°F (-4°C)

Dewpoint: 21.2°F (-6°C)

Pressure: 29.88 in. Hg

Visibility: 0.25 sm

Report Type:

Sky Conditions: Broken clouds at 100 feet, Overcast at 1200 feet

Weather Conditions: Heavy Snow, Moderate Blowing Snow

Runways

KMSN Runways

Magnetic deviation: 2E
Elevation: 887 ft

Wind (mag): 15 kts from 20°
X-wind: 2 kts from the left for 03

Predicted Active: 03

Width: 150 feet
Length: 7200 feet
Surface: Good CONC

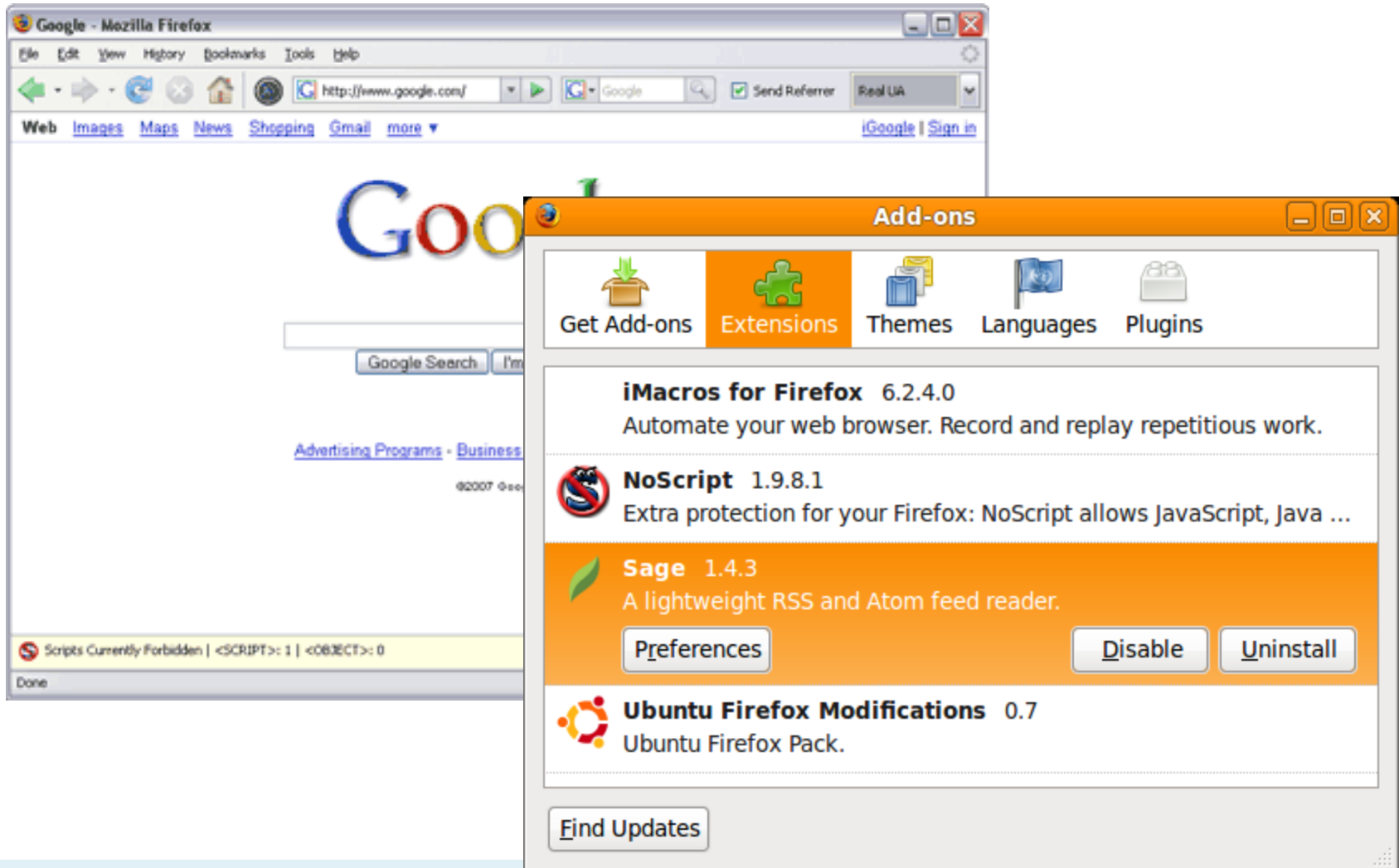
Airport Links

- KMSN on Google Maps
- KMSN AirNav.com Page
- KMSN Approaches
- KMSN PIREPS
- KMSN METAR and/or TAF
- KMSN NOTAMS (PilotWeb)

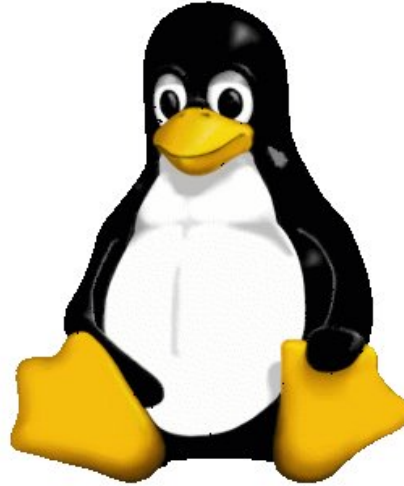
Nearby Airports

- KDLL - Baraboo, WI - 29.72 NM
- KEFT - Monroe, WI - 33.40 NM
- KJVL - Janesville, WI - 33.78 NM

Reuse and variation: Web browser extensions



Reuse and variation: Flavors of Linux



```
Linux Kernel v2.6.18-53.1.14.el5.customxen Configuration

Network File Systems
Arrow keys navigate the menu. <Enter> selects submenus ---.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module <> module

~(-)
[ ] Provide NFS client caching support (EXPERIMENTAL)
[*] Allow direct I/O on NFS files (EXPERIMENTAL)
<M> NFS server support
[*] Provide NFSv3 server support
[*] Provide server support for the NFSv3 ACL protocol extension
[*] Provide NFSv4 server support (EXPERIMENTAL)
--- Provide NFS server over TCP support
[[*]] Root file system on NFS
--- Secure RPC: Kerberos V mechanism (EXPERIMENTAL)
v(+)
```



Reuse and variation: Product lines



Earlier in this course: Class-level reuse

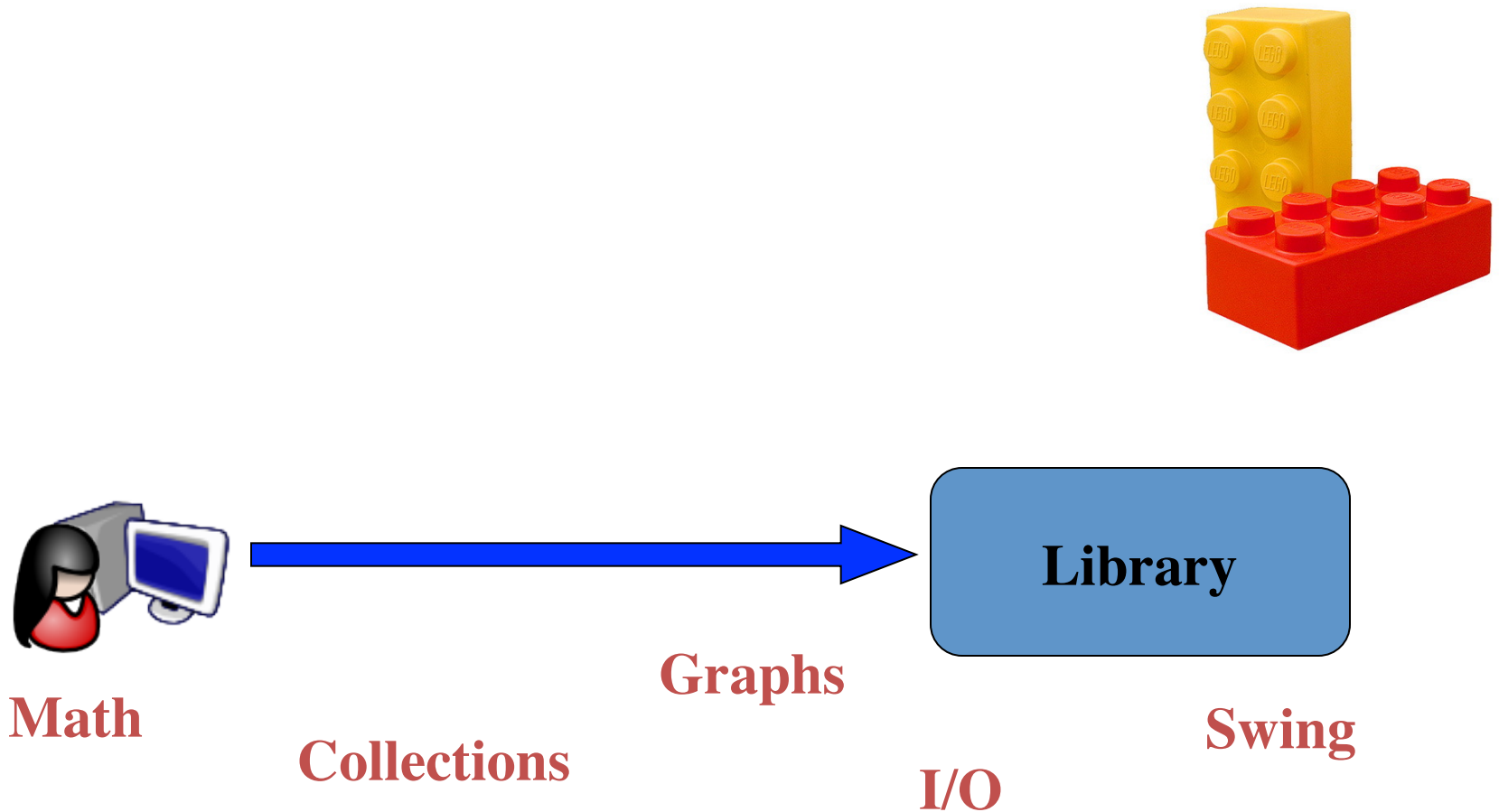
- Language mechanisms supporting reuse
 - Inheritance
 - Subtype polymorphism (dynamic dispatch)
 - Parametric polymorphism (generics)
- Design principles supporting reuse
 - Small interfaces
 - Information hiding
 - Low coupling
 - High cohesion
- Design patterns supporting reuse
 - Template method,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

Today: Libraries and frameworks for reuse

- Examples, terminology
- Whitebox and blackbox frameworks
- Design considerations
- Implementation details
 - Responsibility for running the framework
 - Loading plugins

Terminology: Libraries

- **Library**: A set of classes and methods that provide reusable functionality



Terminology: Frameworks

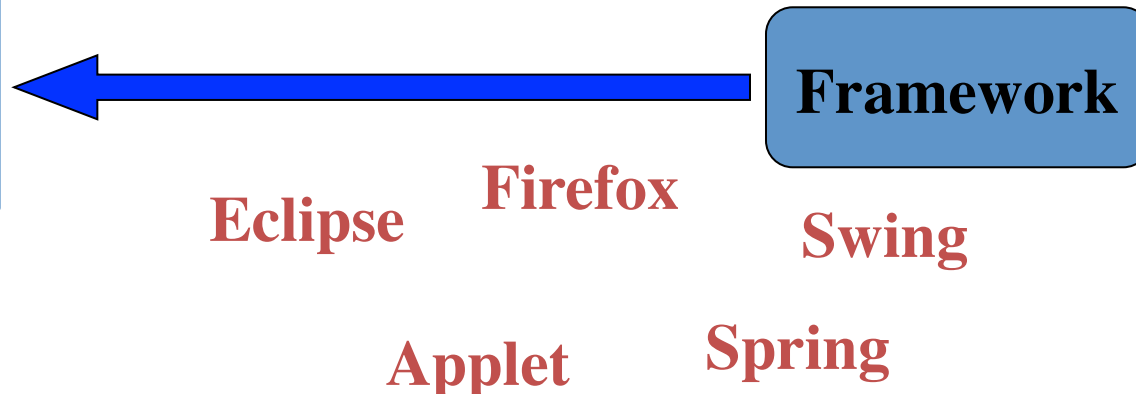
- Framework: Reusable skeleton code that can be customized into an application
- Framework calls back into client code
 - The Hollywood principle: “Don’t call us. We’ll call you.”



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

    / render component on first view and
    resizing
    protected void
    paintComponent(Graphics g) {
        // draw a red box on his
        componentDimension d = getSize();
        g.setColor(Color.red);
        g.drawRect(0, 0, d.getWidth(),
        d.getHeight());
    }
}
```

your code



A calculator example (without a framework)



```
public class Calc extends JFrame {
    private JTextField textField;
    public Calc() {
        JPanel contentPane = new JPanel(new BorderLayout());
        contentPane.setBorder(new BevelBorder(BevelBorder.LOWERED));
        JButton button = new JButton();
        button.setText("calculate");
        contentPane.add(button, BorderLayout.EAST);
        textField = new JTextField("");
        textField.setText("10 / 2 + 6");
        textField.setPreferredSize(new Dimension(200, 20));
        contentPane.add(textfield, BorderLayout.WEST);
        button.addActionListener(/* calculation code */);
        this.setContentPane(contentPane);
        this.pack();
        this.setLocation(100, 100);
        this.setTitle("My Great Calculator");
        ...
    }
}
```


A simple example framework

- Consider a family of programs consisting of a button and text field only:



- What source code might be shared?

A calculator example (without a framework)



```
public class Calc extends JFrame {
    private JTextField textField;
    public Calc() {
        JPanel contentPane = new JPanel(new BorderLayout());
        contentPane.setBorder(new BevelBorder(BevelBorder.LOWERED));
        JButton button = new JButton();
        button.setText("calculate");
        contentPane.add(button, BorderLayout.EAST);
        textField = new JTextField("");
        textField.setText("10 / 2 + 6");
        textField.setPreferredSize(new Dimension(200, 20));
        contentPane.add(textfield, BorderLayout.WEST);
        button.addActionListener(/* calculation code */);
        this.setContentPane(contentPane);
        this.pack();
        this.setLocation(100, 100);
        this.setTitle("My Great Calculator");
        ...
    }
}
```

A simple example framework

```
public abstract class Application extends JFrame {
    protected String getApplicationTitle() { return ""; }
    protected String getButtonText() { return ""; }
    protected String getInitialText() { return ""; }
    protected void buttonClicked() { }
    private JTextField textField;
    public Application() {
        JPanel contentPane = new JPanel(new BorderLayout());
        contentPane.setBorder(new BevelBorder(BevelBorder.LOWERED));
        JButton button = new JButton();
        button.setText(getButtonText());
        contentPane.add(button, BorderLayout.EAST);
        textField = new JTextField("");
        textField.setText(getInitialText());
        textField.setPreferredSize(new Dimension(200, 20));
        contentPane.add(textField, BorderLayout.WEST);
        button.addActionListener((e) -> { buttonClicked(); });
        this.setContentPane(contentPane);
        this.pack();
        this.setLocation(100, 100);
        this.setTitle(getApplicationTitle());
        ...
    }
}
```

Using the example framework

```
public abstract class Application extends JFrame {  
    protected String getApplicationTitle() { return ""; }  
    protected String getButtonText() { return ""; }  
    protected String getInitialText() { return ""; }  
    protected void buttonClicked() { }
```

```
public class Calculator extends Application {  
    protected String getApplicationTitle() { return "My Great Calculator"; }  
    protected String getButtonText() { return "calculate"; }  
    protected String getInitialText() { return "(10 - 3) * 6"; }  
    protected void buttonClicked() {  
        JOptionPane.showMessageDialog(this, "The result of " + getInput() +  
            " is " + calculate(getInput()));  
    }  
    private String calculate(String text) { ... }  
}
```

```
    button.addActionListener((e) -> { buttonClicked(); });  
    this.setContentPane(contentPane);  
    this.pack();  
    this.setLocation(100, 100);  
    this.setTitle(getApplicationTitle());  
    ...  
}
```

Using the example framework again

```
public abstract class Application extends JFrame {  
    protected String getApplicationTitle() { return ""; }  
    protected String getButtonText() { return ""; }  
    protected String getInitialText() { return ""; }  
    protected void buttonClicked() { }
```

```
public class Calculator extends Application {  
    protected String getApplicationTitle() { return "My Great Calculator"; }  
    protected String getButtonText() { return "calculate"; }  
    protected String getInitialText() { return "(10 - 3) * 6"; }  
    protected void buttonClicked() {  
        JOptionPane.showMessageDialog(this, "The result of " + getInput() +  
            " is " + calculate(getInput()));  
    }  
    private String calculate(String text) { ... }  
}
```

```
public class Ping extends Application {  
    protected String getApplicationTitle() { return "Ping"; }  
    protected String getButtonText() { return "ping"; }  
    protected String getInitialText() { return "127.0.0.1"; }  
    protected void buttonClicked() { ... }  
}
```

General distinction: Library vs. framework



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

    // render component on first view and
    // resizing
    protected void
    paintComponent(Graphics g) {
        // draw a red box on his
        componentDimension d = getSize();
        g.setColor(Color.red);
        g.drawRect(0, 0, d.getWidth(),
        d.getHeight());
    }
}
```

your code

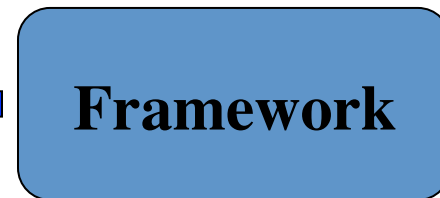


user
interacts

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

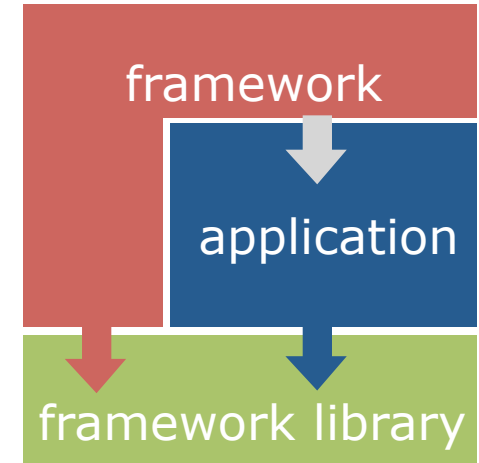
    // render component on first view and
    // resizing
    protected void
    paintComponent(Graphics g) {
        // draw a red box on his
        componentDimension d = getSize();
        g.setColor(Color.red);
        g.drawRect(0, 0, d.getWidth(),
        d.getHeight());
    }
}
```

your code



Libraries and frameworks in practice

- Defines key abstractions and their interfaces
- Defines object interactions & invariants
- Defines flow of control
- Provides architectural guidance
- Provides defaults



credit: Erich Gamma

Framework or library?

- Eclipse
- Java Collections

Framework or library?

- Eclipse
- Java Collections
- The Java Logging Framework
- Java Cryptographic Extensions
- Wordpress
- Django

- On a piece of paper:
 1. Describe the software (\leq one sentence)
 2. Describe one way the software is like a library.
 3. Describe one way the software is like a framework.

A Carcassonne framework?



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 that gets called in a sequence according to the protocol and the state of the plugin

WHITE-BOX VS BLACK-BOX FRAMEWORKS

Whitebox frameworks

- Extension via subclassing and overriding methods
- Common design pattern(s):
 - Template method
- Subclass has main method but gives control to framework

Blackbox frameworks

- Extension via implementing a plugin interface
- Common design pattern(s):
 - Strategy
 - Observer
- Plugin-loading mechanism loads plugins and gives control to the framework

Whitebox vs. blackbox frameworks

- Whitebox frameworks
 - Extension via subclassing and overriding methods
 - Common design pattern(s): Template method
 - Subclass has main method but gives control to framework
- Blackbox frameworks
 - Extension via implementing a plugin interface
 - Common design pattern(s): Strategy, Observer
 - Plugin-loading mechanism loads plugins and gives control to the framework

Is this a whitebox or blackbox framework?

```
public abstract class Application extends JFrame {
    protected String getApplicationTitle() { return ""; }
    protected String getButtonText() { return ""; }
    protected String getInitialText() { return ""; }
    protected void buttonClicked() { }
```

```
public class Calculator extends Application {
    protected String getApplicationTitle() { return "My Great Calculator"; }
    protected String getButtonText() { return "calculate"; }
    protected String getInitialText() { return "(10 - 3) * 6"; }
    protected void buttonClicked() {
        JOptionPane.showMessageDialog(this, "The result of " + getInput() +
            " is " + calculate(getInput()));
    }
    private String calculate(String text) { ... }
}
```

```
public class Ping extends Application {
    protected String getApplicationTitle() { return "Ping"; }
    protected String getButtonText() { return "ping"; }
    protected String getInitialText() { return "127.0.0.1"; }
    protected void buttonClicked() { ... }
}
```

An example blackbox framework

```
public class Application extends JFrame {
    private JTextField textField;
    private Plugin plugin;
    public Application() { }
    protected void init(Plugin p) {
        p.setApplication(this);
        this.plugin = p;
        JPanel contentPane = new JPanel();
        contentPane.setBorder(new BorderLayout());
        JButton button = new JButton();
        button.setText(plugin != null ? plugin.getButtonText() : "ok");
        contentPane.add(button, BorderLayout.EAST);
        textField = new JTextField("");
        if (plugin != null) textField.setText(plugin.getInitialText());
        textField.setPreferredSize(new Dimension(200, 20));
        contentPane.add(textField, BorderLayout.WEST);
        if (plugin != null)
            button.addActionListener((e) -> { plugin.buttonClicked(); } );
        this.setContentPane(contentPane);
        ...
    }
    public String getInput() { return textField.getText(); }
}
```

```
public interface Plugin {
    String getApplicationTitle();
    String getButtonText();
    String getInitialText();
    void buttonClicked();
    void setApplication(Application app);
}
```

An example blackbox framework

```
public class Application extends JFrame {
    private JTextField textField;
    private Plugin plugin;
    public Application() { }
    protected void init(Plugin p) {
        p.setApplication(this);
        this.plugin = p;
        JPanel contentPane = new JPanel()
        contentPane.setBorder(new BorderLayout());
        JButton button = new JButton("Calculate");
```

```
public interface Plugin {
    String getApplicationTitle();
    String getButtonText();
    String getInitialText();
    void buttonClicked();
    void setApplication(Application app);
}
```

```
public class CalcPlugin implements Plugin {
    private Application app;
    public void setApplication(Application app) { this.app = app; }
    public String getButtonText() { return "calculate"; }
    public String getInitialText() { return "10 / 2 + 6"; }
    public void buttonClicked() {
        JOptionPane.showMessageDialog(null, "The result of "
            + application.getInput() + " is "
            + calculate(application.getInput()));
    }
    public String getApplicationTitle() { return "My Great Calculator"; }
}
```

```
}
```

An aside: Plugins could be reusable too...

```
public class Application extends JFrame implements InputProvider {
    private JTextField textField;
    private Plugin plugin;
    public Application() { }
    protected void init(Plugin p)
        p.setApplication(this);
        this.plugin = p;
        JPanel contentPane = new
        contentPane.setBorder(new
        JButton button = new JButton();
}
```

```
public interface Plugin {
    String getApplicationTitle();
    String getButtonText();
    String getInititalText();
    void buttonClicked() ;
    void setApplication(InputProvider app);
}
```

```
public class CalcPlugin implements Plugin {
    private InputProvider app;
    public void setApplication(InputProvider app) { this.app = app; }
    public String getButtonText() { return "Calculate"; }
    public String getInititalText() { return "0"; }
    public void buttonClicked() {
        JOptionPane.showMessageDialog(null,
            + application.getInput() + " is "
            + calculate(application.getInput()));
    }
    public String getApplicationTitle() { return "My Great Calculator"; }
}
```

```
public interface InputProvider {
    String getInput();
}
```

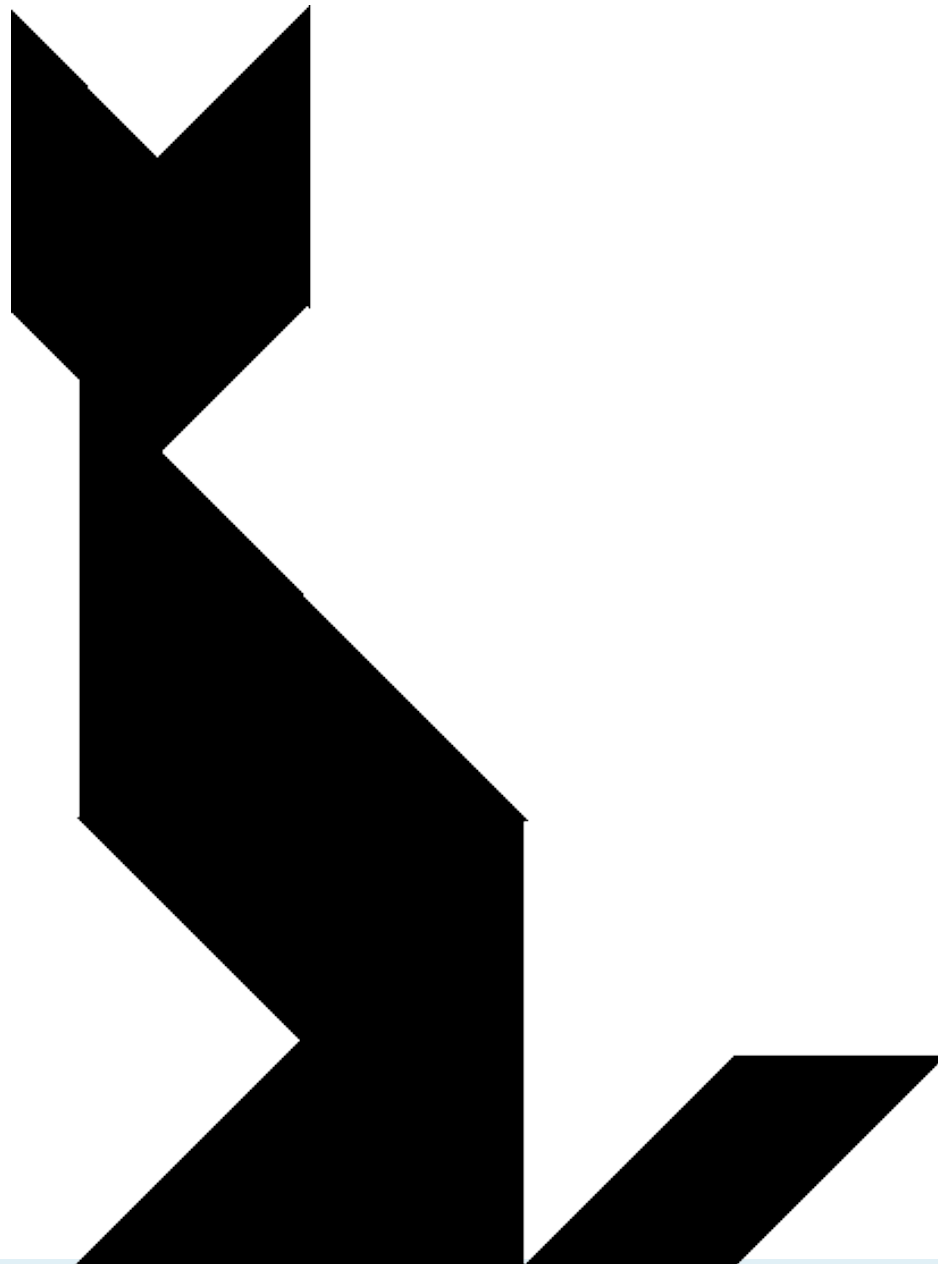
Whitebox vs. blackbox framework summary

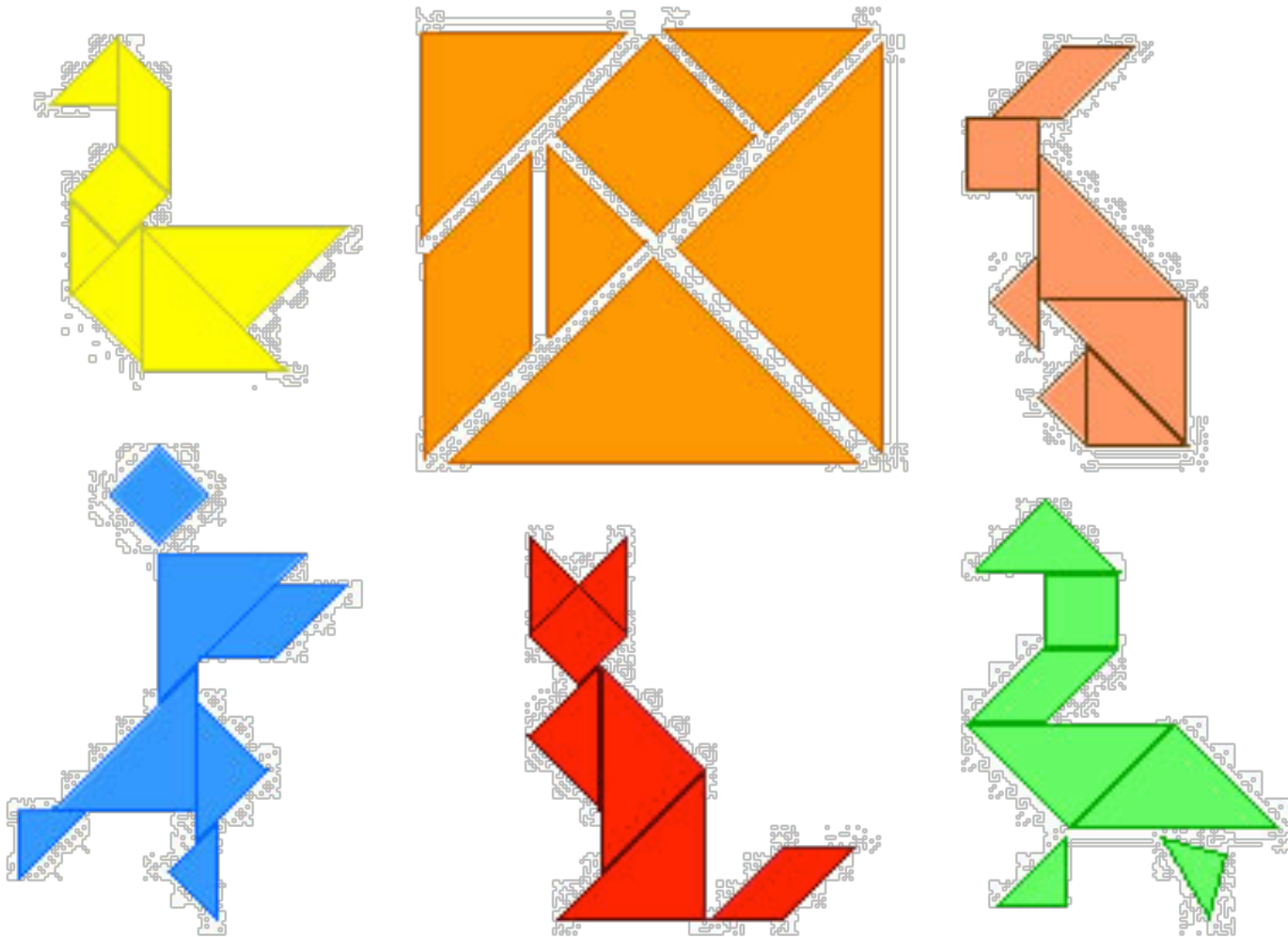
- Whitebox frameworks use subclassing
 - Allows extension of every nonprivate method
 - Need to understand implementation of superclass
 - Only one extension at a time
 - Compiled together
 - Often so-called developer frameworks
- Blackbox frameworks use composition
 - Allows extension of functionality 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

Framework design considerations

- Once designed there is little opportunity for change
- Key decision: Separating common parts from variable parts
 - What problems do you want to solve?
- Possible problems:
 - Too few extension points: Limited to a narrow class of users
 - Too many extension points: Hard to learn, slow
 - Too generic: Little reuse value

USE VS REUSE: DOMAIN ENGINEERING





(one modularization: tangrams)

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 designing a framework
- 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

- Define your domain
 - Identify potential common parts and variable parts
- Design and write sample plugins/applications
- Factor out & implement common parts as framework
- Provide plugin interface & callback mechanisms for variable parts
 - Use well-known design principles and patterns where appropriate...
- Get lots of feedback, and iterate

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

(credit: Erich Gamma)

Not discussed here...

- Framework implementation details
 - Mechanics of running the framework
 - Mechanics of loading plugins

Summary

- Reuse and variation essential
 - Libraries and frameworks
- Whitebox frameworks vs. blackbox frameworks
- Design for reuse with domain analysis
 - Find common and variable parts
 - Write client applications to find common parts