

# Principles of Software Construction: Objects, Design, and Concurrency

## Part 4: Design for large-scale reuse

### Libraries and frameworks

**Charlie Garrod**

**Michael Hilton**

# Administrivia

- Homework 4b due tonight
  - Homework 4c due next Thursday
- Next required reading due next Tuesday
  - Effective Java, Items 40, 48, 50, and 52
- Final exam review session day/time?



# Key concepts from Tuesday

# Key concepts from Tuesday

- Avoid premature optimization
- Use a profiler to understand performance
- Pitfalls of common APIs
- Garbage collection

More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason —including blind stupidity.

—William A. Wulf

# Good programs, rather than fast ones

- Information hiding
- A good architecture scales
- Hardware is cheap, developers are not
- Optimize only clear, concise, well-structured implementations
- Those who exchange readability for performance will lose both

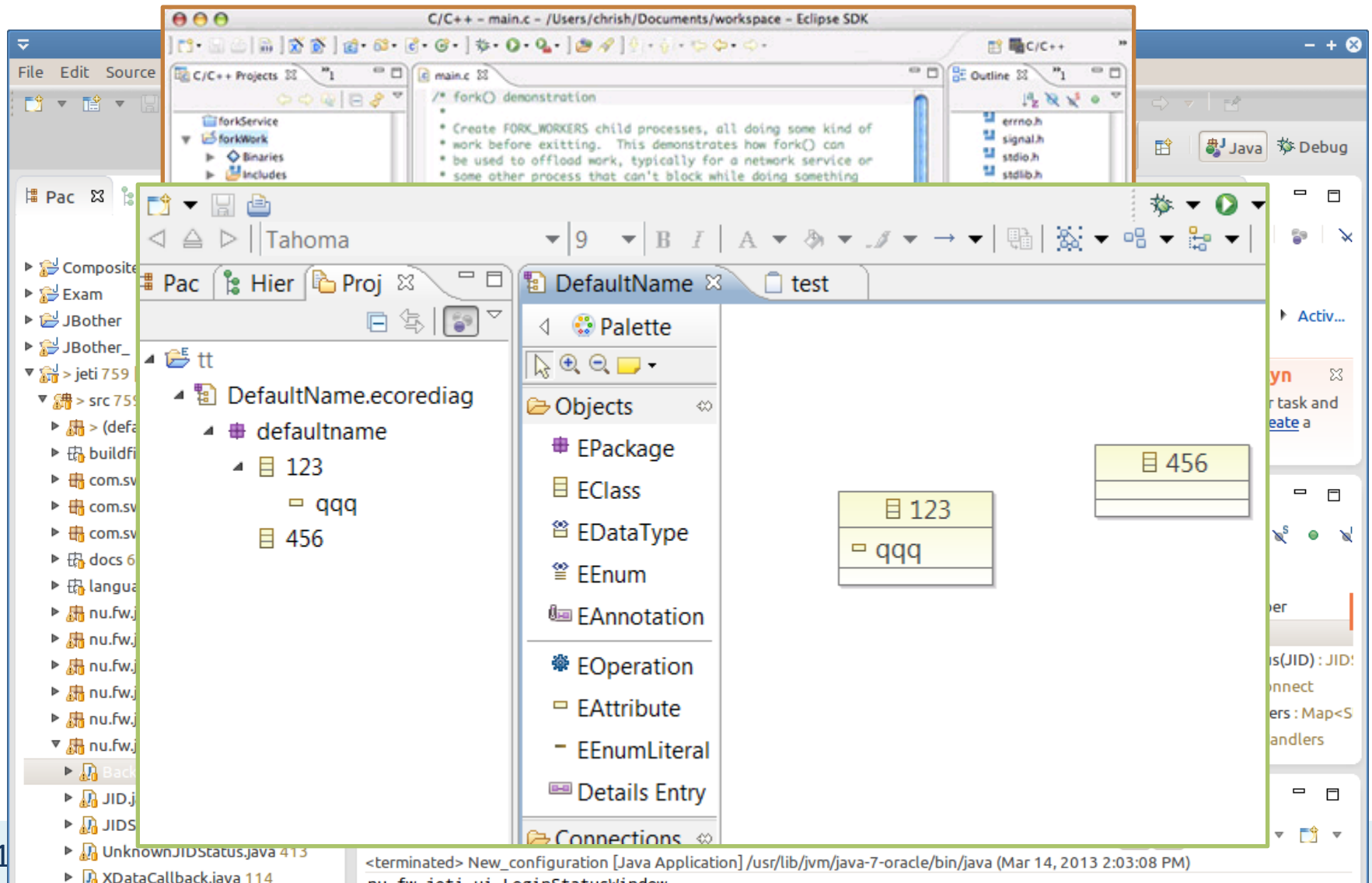
# Learning goals for today

- 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

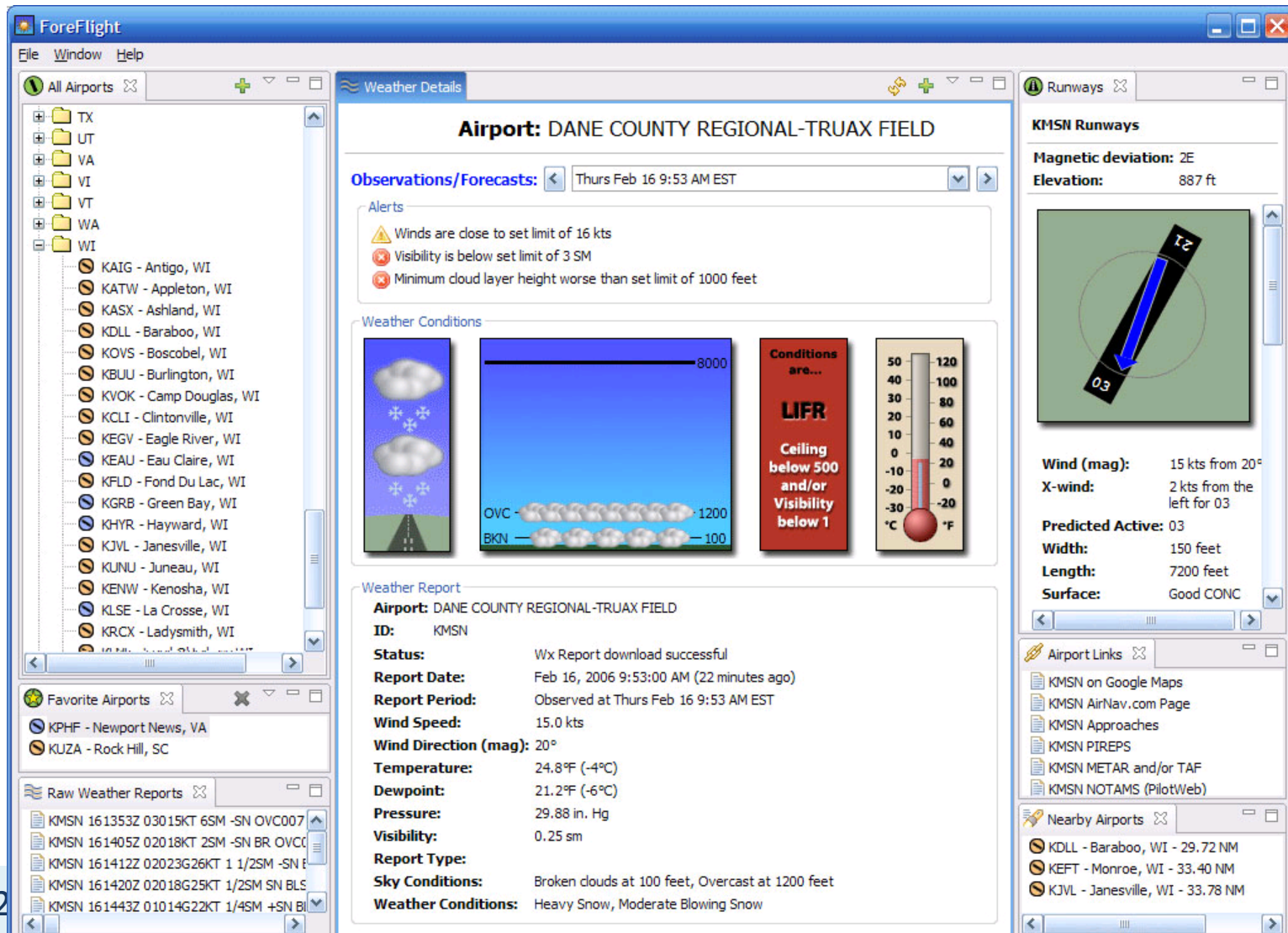
# Today: Libraries and frameworks for reuse

# Reuse and variation:

## Family of development tools



# Reuse and variation: Eclipse Rich Client Platform



The screenshot displays the ForeFlight application window, which is divided into several panes. The left pane shows a hierarchical list of airports, with 'WI' (Wisconsin) selected. The main pane is titled 'Airport: DANE COUNTY REGIONAL-TRUAX FIELD' and displays 'Observations/Forecasts' for Thursday, February 16, 2006, at 9:53 AM EST. The 'Alerts' section shows three warnings: 'Winds are close to set limit of 16 kts', 'Visibility is below set limit of 3 SM', and 'Minimum cloud layer height worse than set limit of 1000 feet'. The 'Weather Conditions' section includes a cloud icon, a sky view with a cloud layer at 1200 feet, a 'LIFR' (Low Visibility) status box, and a thermometer showing 24.8°F (-4°C). The 'Weather Report' section provides details for the report, including the status, report date, report period, wind speed, wind direction, temperature, dewpoint, pressure, visibility, report type, sky conditions, and weather conditions. The right pane shows 'KMSN Runways' with magnetic deviation, elevation, and a diagram of the runway. Below the runways, it lists wind information, predicted active runway, width, length, and surface. The bottom right pane shows 'Airport Links' and 'Nearby Airports'.

**ForeFlight**

File Window Help

All Airports

- TX
- UT
- VA
- VI
- VT
- WA
- WI
  - KAIG - Antigo, WI
  - KATW - Appleton, WI
  - KASX - Ashland, WI
  - KDL - Baraboo, WI
  - KOV - 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

Favorite Airports

- KPHF - Newport News, VA
- KUZA - Rock Hill, SC

Raw Weather Reports

- KMSN 161353Z 03015KT 6SM -SN OVC007
- KMSN 161405Z 02018KT 2SM -SN BR OVC007
- KMSN 161412Z 02023G26KT 1 1/2SM -SN
- KMSN 161420Z 02018G25KT 1/2SM SN BLS
- KMSN 161443Z 01014G22KT 1/4SM +SN BLS

**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**

Clouds: 8000

OVC - 1200

BKN - 100

**Conditions are...**

**LIFR**

Ceiling below 500 and/or Visibility below 1

Temperature: 24.8°F (-4°C)

Dewpoint: 21.2°F (-6°C)

Pressure: 29.88 in. Hg

Visibility: 0.25 sm

**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**

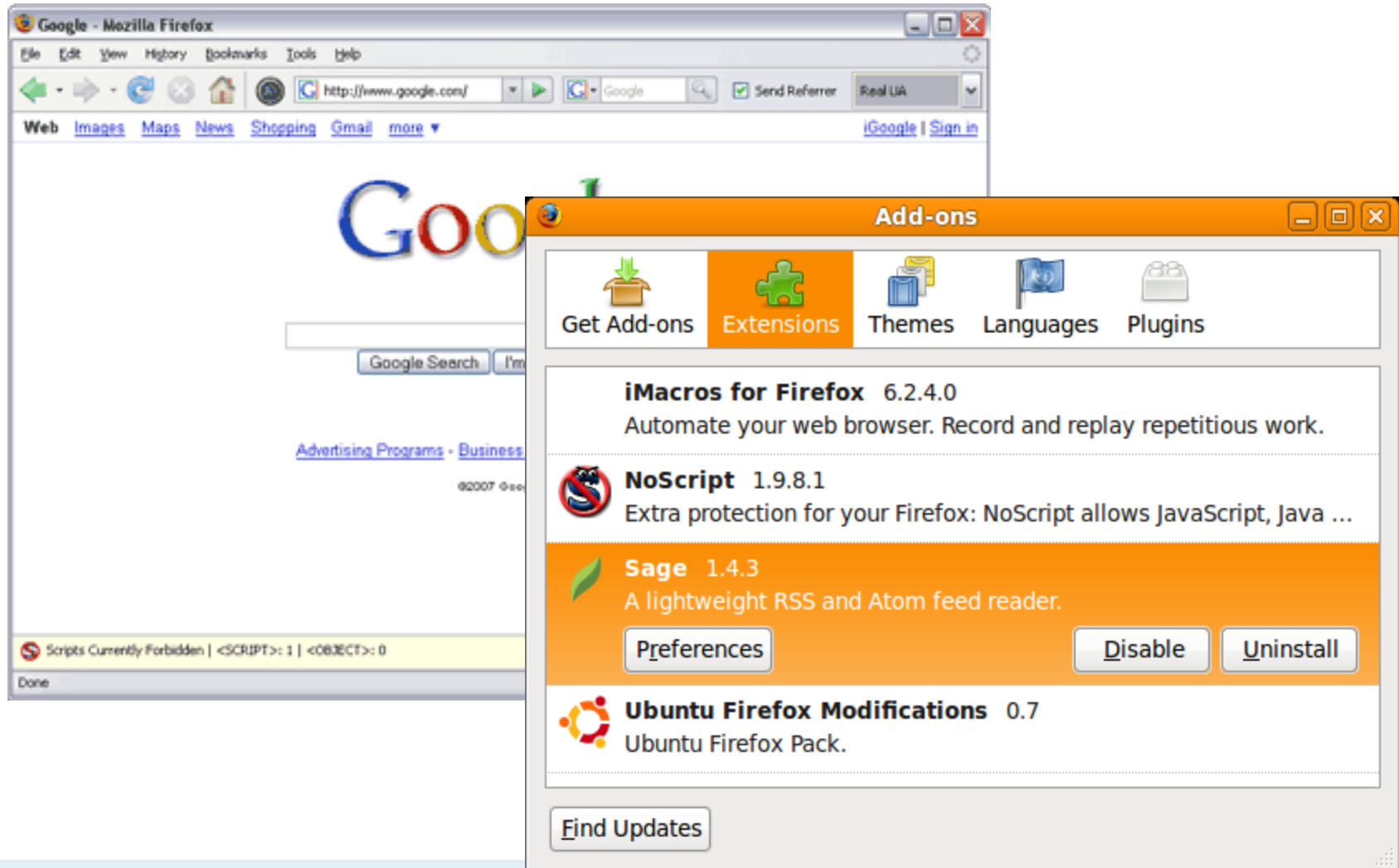
- KMSN on Google Maps
- KMSN AirNav.com Page
- KMSN Approaches
- KMSN PIREPS
- KMSN METAR and/or TAF
- KMSN NOTAMS (PilotWeb)

**Nearby Airports**

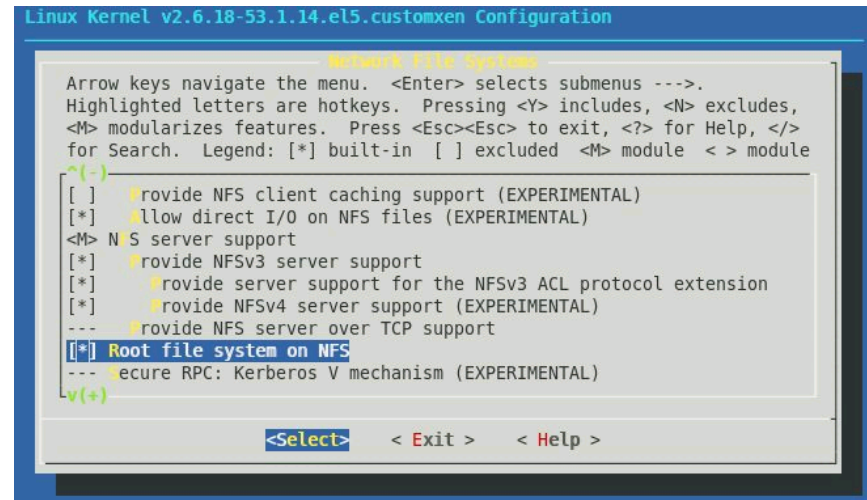
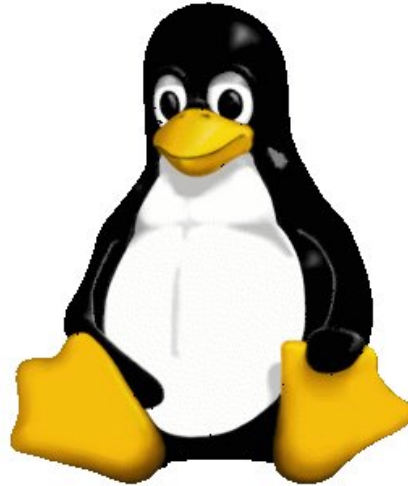
- KDL - Baraboo, WI - 29.72 NM
- KEFT - Monroe, WI - 33.40 NM
- KJVL - Janesville, WI - 33.78 NM

# Reuse and variation:

## Web browser extensions



# Reuse and variation: Flavors of Linux



# Reuse and variation: Product lines



# Earlier in this course: Class-level reuse

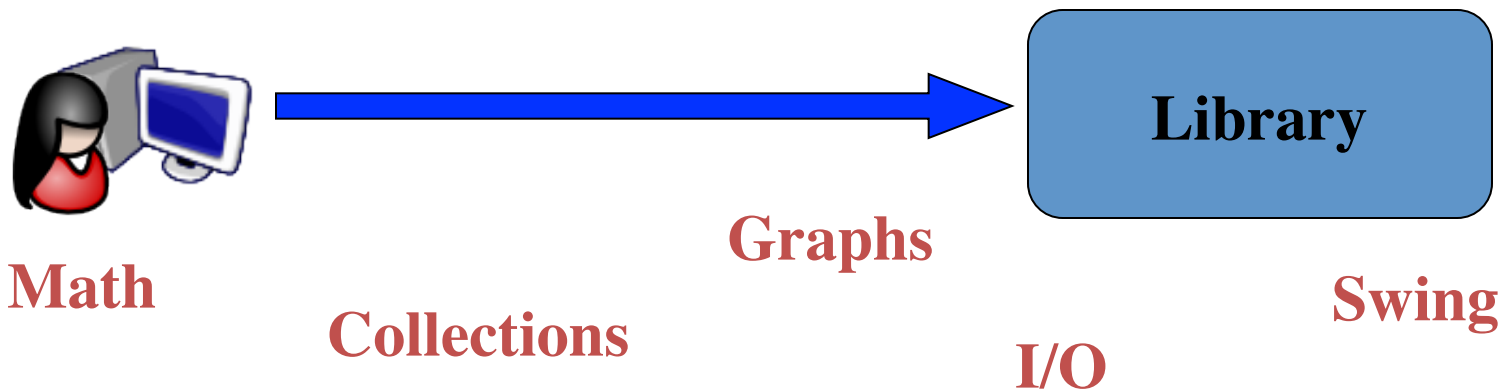
- Language mechanisms supporting reuse
  - Inheritance
  - Subtype polymorphism (dynamic dispatch) for delegation
  - Parametric polymorphism (generics)
- Design principles supporting reuse
  - Small interfaces
  - Information hiding
  - Low coupling
  - High cohesion
- Design patterns supporting reuse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

# Today: Libraries and frameworks for reuse

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

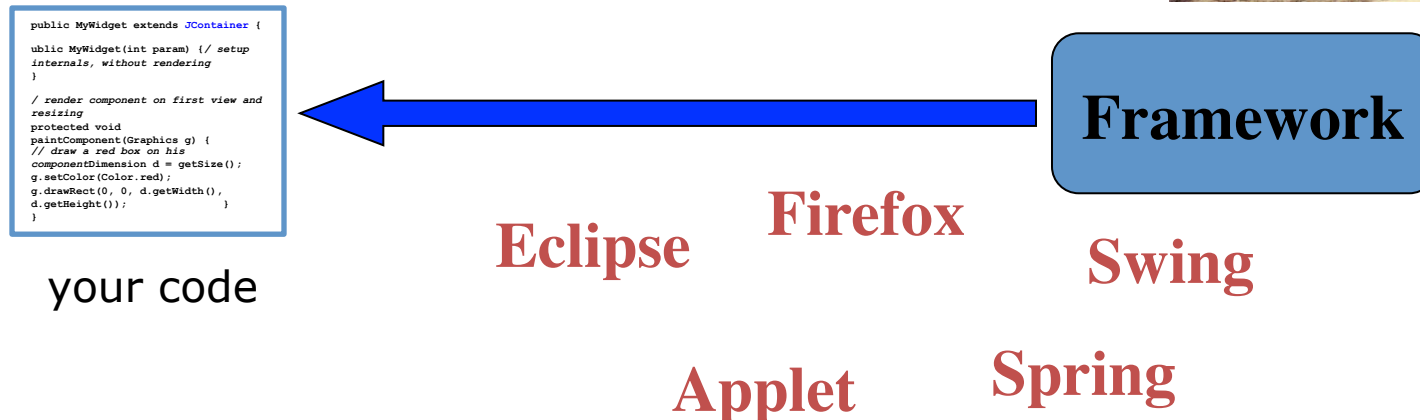
# Terminology: Libraries

- **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.”



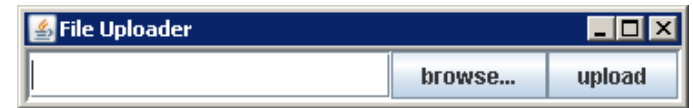
# A calculator example (without a framework)



```
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 buttons and text fields 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 getInititalText() { return ""; }  
    protected void buttonClicked() { }
```

```
public class Calculator extends Application {  
    protected String getApplicationTitle() { return "My Great Calculator"; }  
    protected String getButtonText() { return "calculate"; }  
    protected String getInititalText() { 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 getInititalText() { 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 this  
        componentDimension d = getSize();  
        g.setColor(Color.red);  
        g.drawRect(0, 0, d.getWidth(),  
            d.getHeight());  
    }  
}
```

your code



**Library**



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 this  
        componentDimension d = getSize();  
        g.setColor(Color.red);  
        g.drawRect(0, 0, d.getWidth(),  
            d.getHeight());  
    }  
}
```

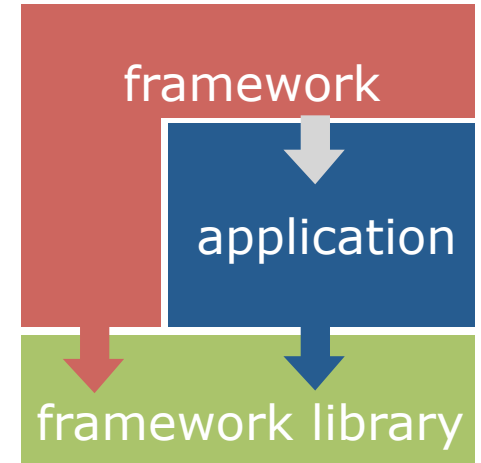
your code



**Framework**

# 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?

- Java Collections
- Eclipse
- The Java Logging Framework
- Java Encryption Services
- Wordpress
- Ruby on Rails

# A Scrabble framework?

- In what way is Homework 4 (Scrabble with Stuff) a 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 getInititalText() { return ""; }  
    protected void buttonClicked() { }
```

```
public class Calculator extends Application {  
    protected String getApplicationTitle() { return "My Great Calculator"; }  
    protected String getButtonText() { return "calculate"; }  
    protected String getInititalText() { 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 getInititalText() { 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.getInititalText());
        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 getInititalText();
    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 BevelBorder(  
        JButton button = new JButton();
```

```
public interface Plugin {  
    String getApplicationTitle();  
    String getButtonText();  
    String getInititalText();  
    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 getInititalText() { 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"; }
    }
}
```

# 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

# 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