

Principles of Software Construction: Objects, Design, and Concurrency

Libraries and Frameworks

(Design for large-scale reuse)

Claire Le Goues

Bogdan Vasilescu



Earlier in this course: **Class-level** reuse

Language mechanisms supporting reuse

- Inheritance
- Subtype polymorphism (dynamic dispatch)
- Parametric polymorphism (generics)*

Design principles supporting reuse

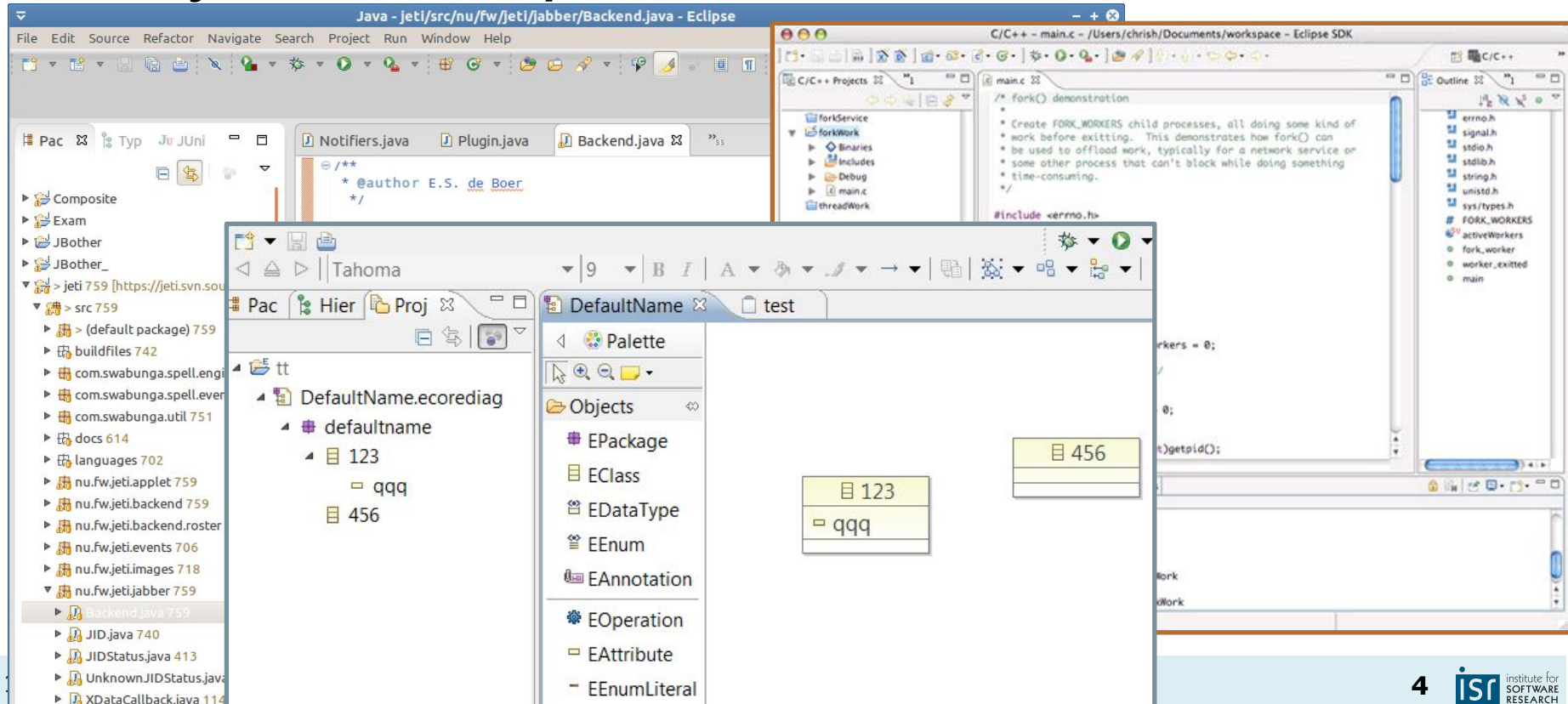
- Small interfaces
- Information hiding
- Low coupling
- High cohesion

Design patterns supporting reuse

- Template method, decorator, strategy, composite, adapter, ...

* Effective Java items 26, 29, 30, and 31

Reuse and variation: Family of development tools



Reuse and variation: Eclipse Rich Client Platform

The screenshot displays the ForeFlight application window. The left sidebar contains a tree view of airports, with 'WI' (Wisconsin) selected. The main pane is titled 'Weather Details' and shows information for 'Airport: DANE COUNTY REGIONAL-TRUAX FIELD'. The 'Observations/Forecasts' section shows the date 'Thurs Feb 16 9:53 AM EST' and three alerts: '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 clouds at 8000, 1200, and 100 feet, a 'LIFR' (Low Instrument Flight Rules) warning, and a thermometer showing 24.8°F (-4°C). The 'Weather Report' section provides details for the airport (ID: KMSN), status (Wx Report download successful), report date (Feb 16, 2006 9:53:00 AM), report period (Observed at Thurs Feb 16 9:53 AM EST), wind speed (15.0 kts), wind direction (20°), temperature (24.8°F (-4°C)), dewpoint (21.2°F (-6°C)), pressure (29.88 in. Hg), visibility (0.25 sm), report type (Broken clouds at 100 feet, Overcast at 1200 feet), and weather conditions (Heavy Snow, Moderate Blowing Snow). The right sidebar shows 'Runways' for KMSN, including magnetic deviation (2E), elevation (887 ft), and a runway diagram. Below this, it lists 'Predicted Active' conditions: 03, width: 150 feet, length: 7200 feet, and surface: Good CONC. The bottom right section 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
KDLL - Baraboo, WI
KOVS - Boscobel, WI
KBUE - Burlington, WI
KYOK - Camp Douglas, WI
KCLI - Clintonville, WI
KEGV - Eagle River, WI
KEAU - Eau Claire, WI
KFLD - Fond Du Lac, WI
KGRB - Green Bay, WI
KH4R - Hayward, WI
KJVL - Janesville, WI
KJUN - Juneau, WI
KENW - Kenosha, WI
KLSE - La Crosse, WI
KRCC - 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 P
KMSN 161420Z 02018G29KT 1/2SM SN BLS
KMSN 161443Z 01014G22KT 1/4SM +SN B

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

Sky Conditions: Heavy Snow, Moderate Blowing Snow

Weather Conditions: Heavy Snow, Moderate Blowing Snow

Runways

KMSN Runways

Magnetic deviation: 2E

Elevation: 887 ft

Runway diagram showing Runway 03.

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

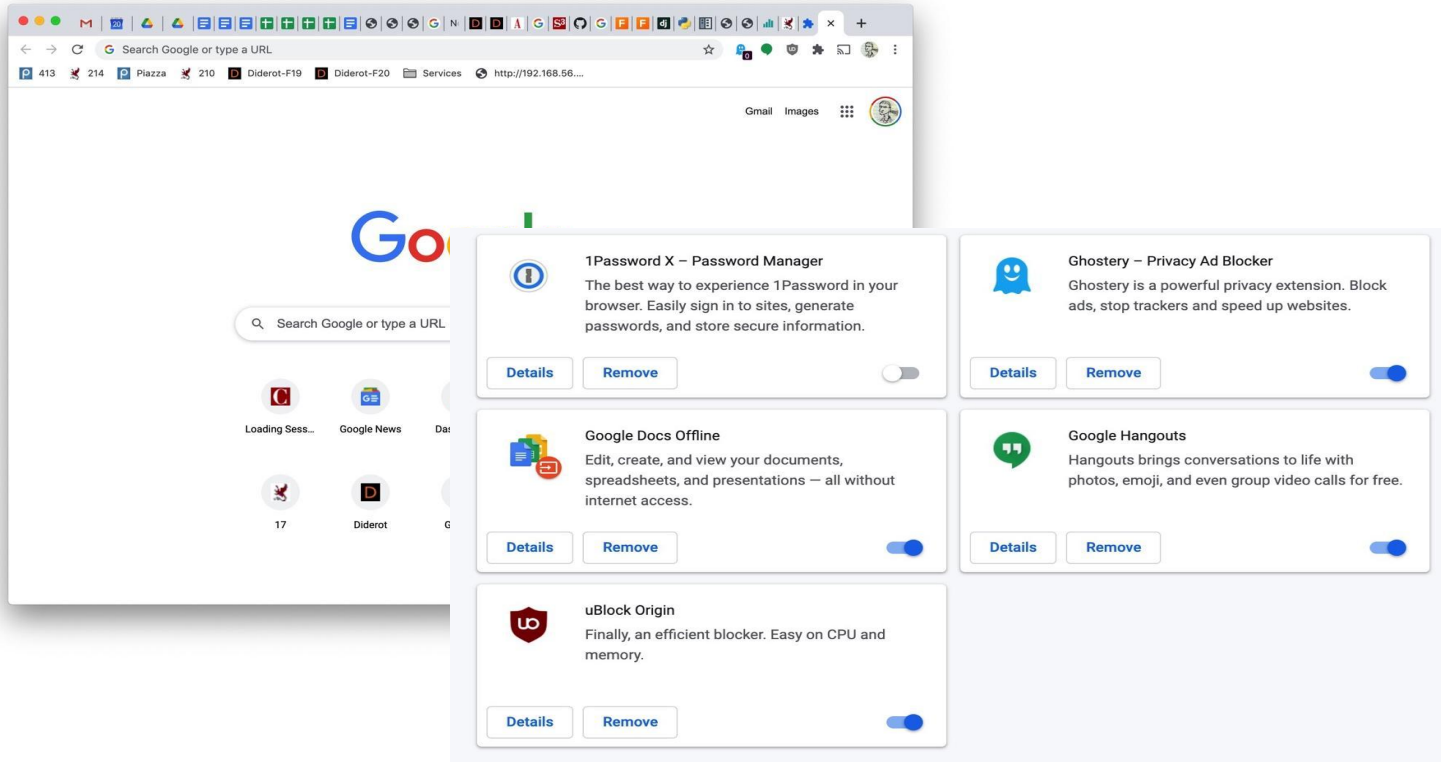
Airport Links

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

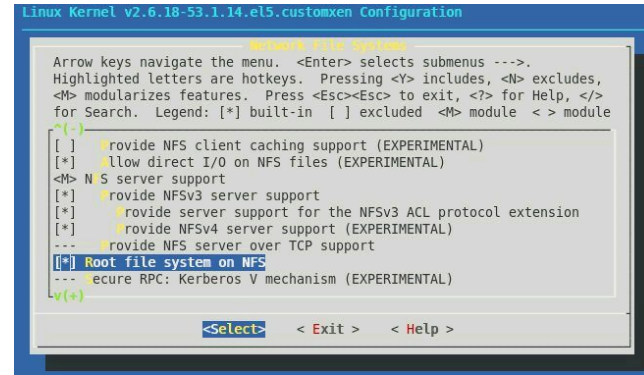
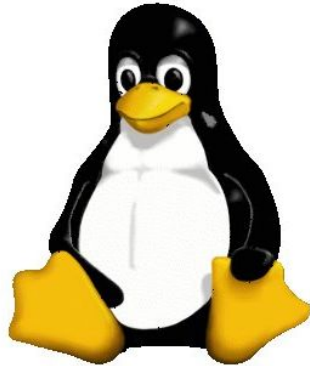
Nearby Airports

- 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



Reuse and variation: Product lines



Today: Reuse **at scale**

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

Today: Reuse **at scale**

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

Terminology: Library



- *Library*: A set of classes and methods that provide reusable functionality
- Client calls library; library executes and returns data
- Client controls
 - Program structure
 - Control flow

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

your code



Library

- E.g.: Math, Collections, Graphs, I/O, Swing

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.”
- Framework controls
 - Program structure
 - Control flow

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

your code



Framework

- E.g.: Eclipse, Firefox, Spring, Swing, IntelliJ, NanoHttpd, Express

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 ""; }

    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) { ... }
    }

    textField.setPreferredSize(new Dimension(200, 20));
    contentPane.add(textField, BorderLayout.WEST);
    button.addActionListener((e) -> { buttonClicked(); });
    this.setContentPane(contentPane);
    this.pack();
}
```

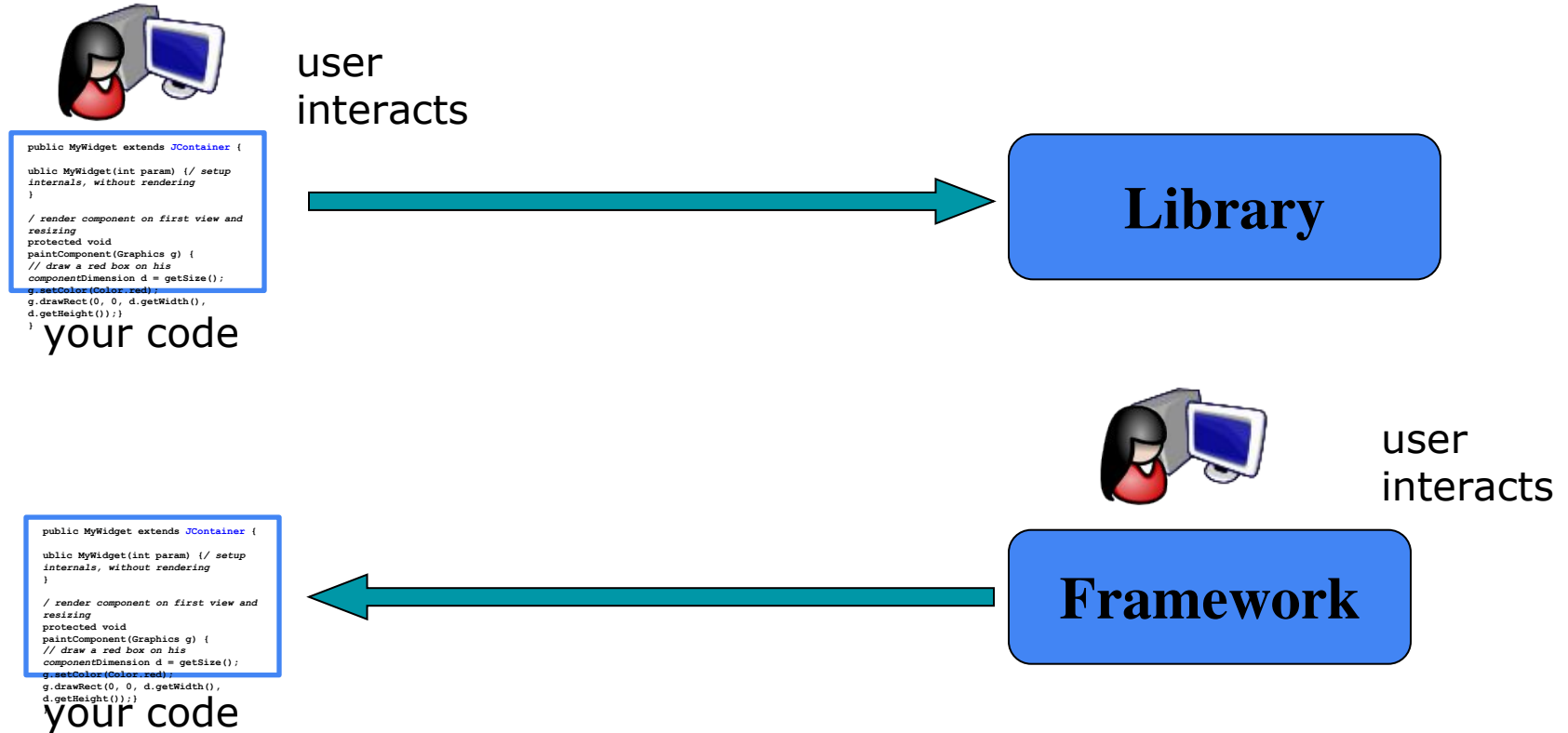

Using the example framework again

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

```
    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) { ... }  
    }
```

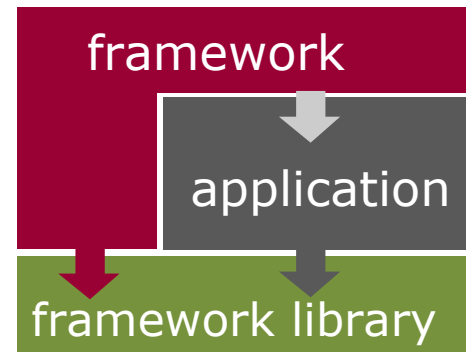
```
    public class Ping extends Application {  
        protected String getApplicationTitle() { return "Ping"; }  
        protected String getButtonText() { return "ping"; }  
        protected String getInititalText() { return "127.0.0.1"; }
```

General distinction: Library vs. framework



Libraries and frameworks in practice

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

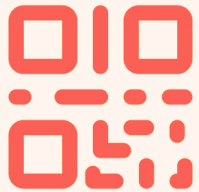


credit: Erich Gamma

Framework or library?

- IntelliJ / VSCode
- Java Collections / Node Streams

slido



**Join at slido.com
#995286**

① Start presenting to display the joining instructions on this slide.

slido



IntelliJ / VSCode: Framework or library? Motivate (+andrewid)

① Start presenting to display the poll results on this slide.

slido



Java Collections / Node Streams: Framework or library? Motivate (+andrewid)

① Start presenting to display the poll results on this slide.

Is Santorini 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

Today: Libraries and frameworks for reuse

- Terminology and examples
- **Whitebox and blackbox frameworks**
- Designing a framework
- Implementation details

WHITE-BOX VS BLACK-BOX* FRAMEWORKS

* outdated terms, not aware of common replacements; maybe Inheritance-Based vs Delegation-Based Frameworks

Whitebox (inheritance-based) frameworks

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

Blackbox (delegation-based) frameworks

- Extension via implementing a plugin interface
- Common design pattern(s):
 - Strategy
 - Command
 - 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 ""; }
```

```
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()));  
    }
```

```
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 BevelBorder(BevelBorder.LOWERED));  
        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 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;  
    }  
}
```

```
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 "  
            + app.getInput() + " is "  
            + calculate(app.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;
```

```
    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) { }  
    public String getButtonText() { return "calculate"; }  
    public String getInititalText() { return "10 / 2 + 6"; }  
    public void buttonClicked() {  
        JOptionPane.showMessageDialog(null, "The result of "  
            + app.getInput() + " is "  
            + calculate(app.getInput()));  
    }
```

```
    public String getApplicationTitle() { return "My Great Calculator"; }
```

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

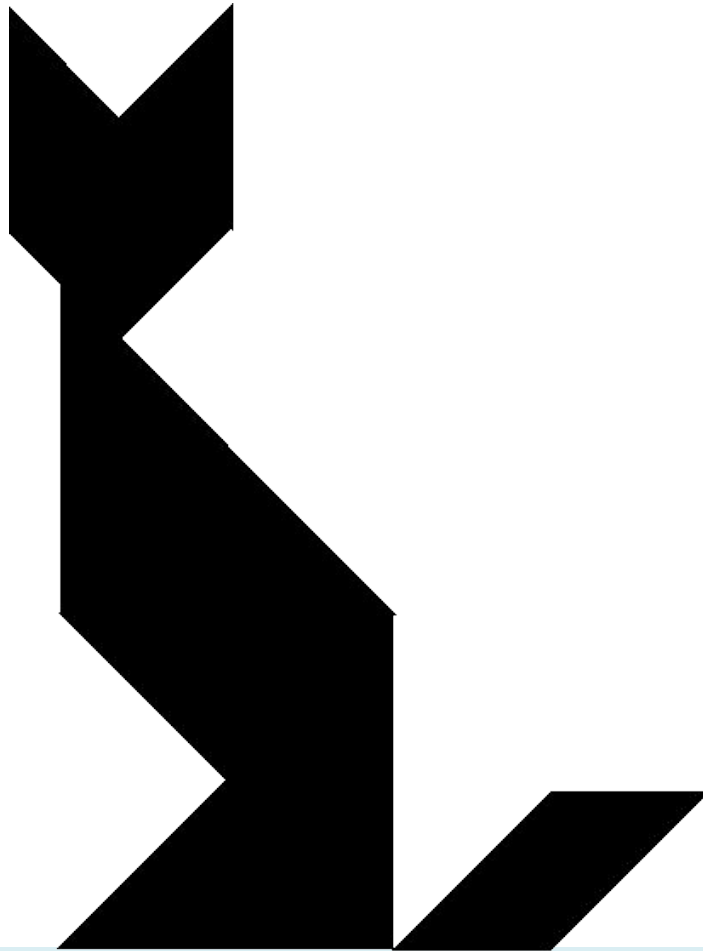
Frameworks summary

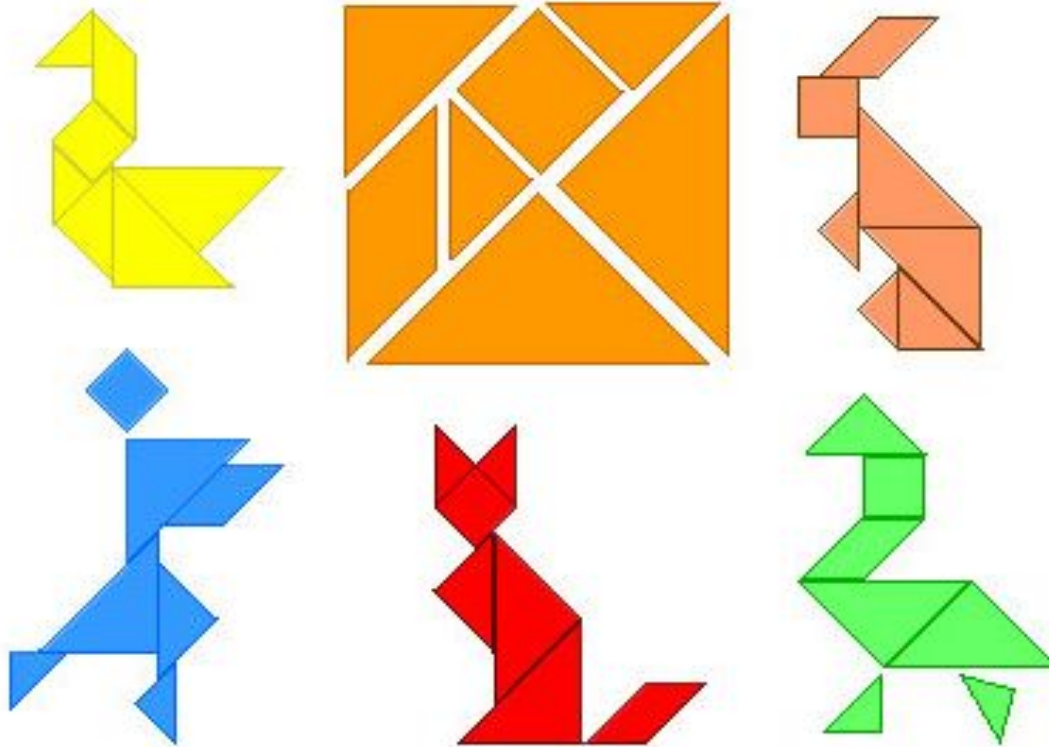
- Whitebox frameworks use subclassing
 - Allows extension of every nonprivate method
 - Need to understand implementation of superclass
 - Only one extension at a time
 - Compiled together
 - Often so-called developer frameworks
- Blackbox frameworks use composition
 - Allows extension of functionality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

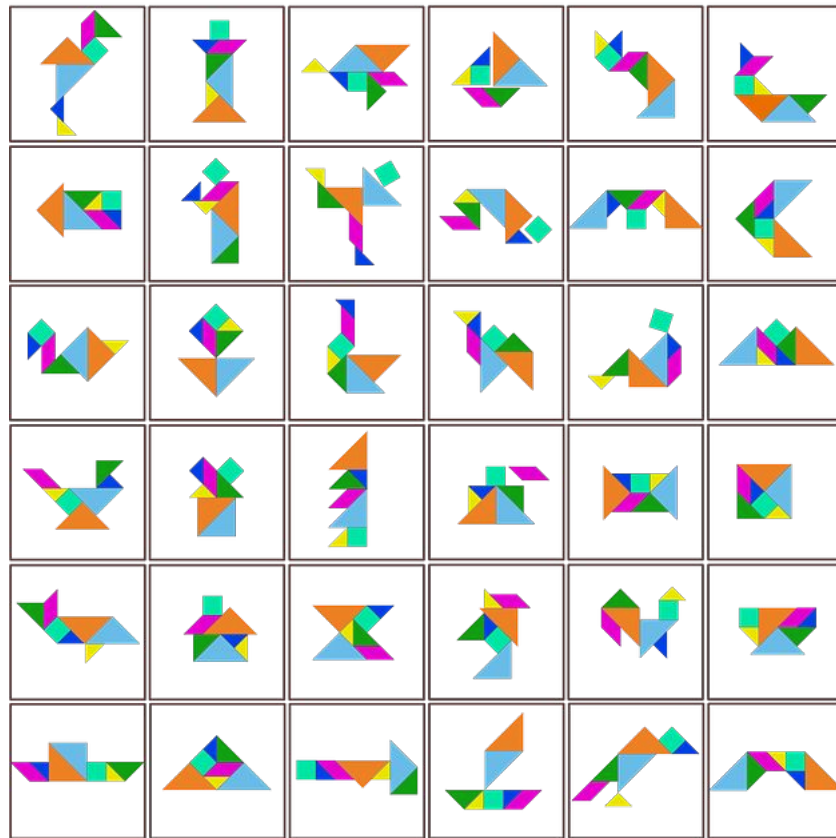
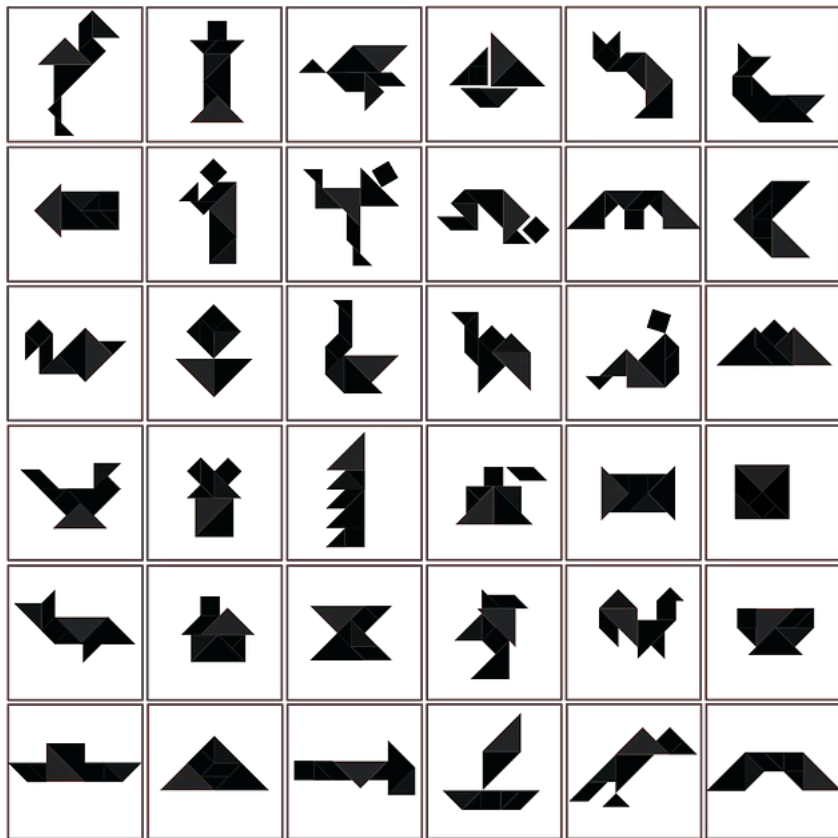
USE VS REUSE: DOMAIN ENGINEERING





(one modularization: tangrams)

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 (*scoping*)
- e.g., the Eclipse policy:
 - Plugin interfaces are internal at first
 - Unsupported, may change
 - Public stable extension points created when there are at least two distinct customers

The cost of changing a framework

```
public class Application extends JFrame {
    private JTextField textfield;
    private Plugin plugin;
    public Application(Plugin p) { this.plugin=p; p.setApplication(this); init(); }
    protected void init() {
        JPanel contentPane = new JPanel(new BorderLayout());
        contentPane.setBorder(new BevelBorder(BevelBorder.LOWERED));
        JButton button = new JButton();
        if (plugin != null)
            button.setText(plugin.getButtonText());
        else
            button.setText("Calculate");
        contentPane.add(button);
        textfield = new JTextField(20);
        if (plugin != null)
            textfield.setText(plugin.getInititalText());
        textfield.setText("");
    }
}
```

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

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

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

```
this.setCon }
```

The cost of changing a framework

```
public class Application extends JFrame {
    private JTextField textfield;
    private Plugin plugin;
    public Application(Plugin p) { this.plugin=p; p.setApplication(this); init(); }
    protected void init() {
        JPanel contentPane = new JPanel(new BorderLayout());
        contentPane.setBorder(new BevelBorder(BevelBorder.LOWERED));
        JButton button = new JButton();
        if (plugin != null)
            button.setText(plugin.getButtonText());
        else
            button.setText("Calculate");
        contentPane.add(button);
        textfield = new JTextField(20);
        if (plugin != null)
            textfield.setText(plugin.getInititalText());
        textfield.setText("");
    }
}
```

Consider adding an extra method.
Requires changes to *all* plugins!

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

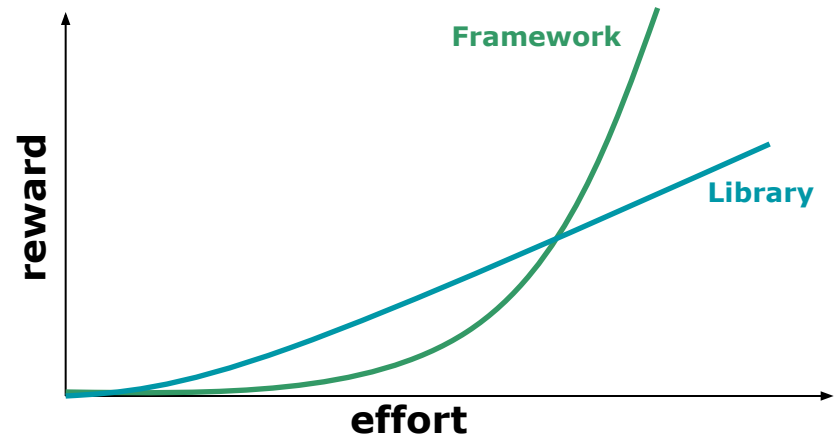
```
public class CalcPlugin implements Plugin {
    private Application application;
    public void setApplication(Application app) { this.application = app; }
    public String getButtonText() { return "calculate"; }
    public String getInititalText() { return "10 / 2 + 6"; }
    public void buttonClicked()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

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

```
this.setCon }
```

Learning a framework

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



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

(next time)

FRAMEWORK MECHANICS

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