Principles of Software Construction: Objects, Design, and Concurrency

Part 2: Designing (sub-) systems

Design for large-scale reuse: Libraries and frameworks

Charlie Garrod Bogdan Vasilescu

School of Computer Science
institute for SOFTWARE RESEARCH

Administrivia

- Homework 4b due Thursday
- Homework 4a feedback still available
  - Can regain 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
- Next required reading due Tuesday after spring break(!)
  - Effective Java, Items 51, 60, 62, and 64
- Final exam Monday, May 7th, 5:30 – 8:30 p.m.
  - Review session day/time?

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

institute for SOFTWARE RESEARCH

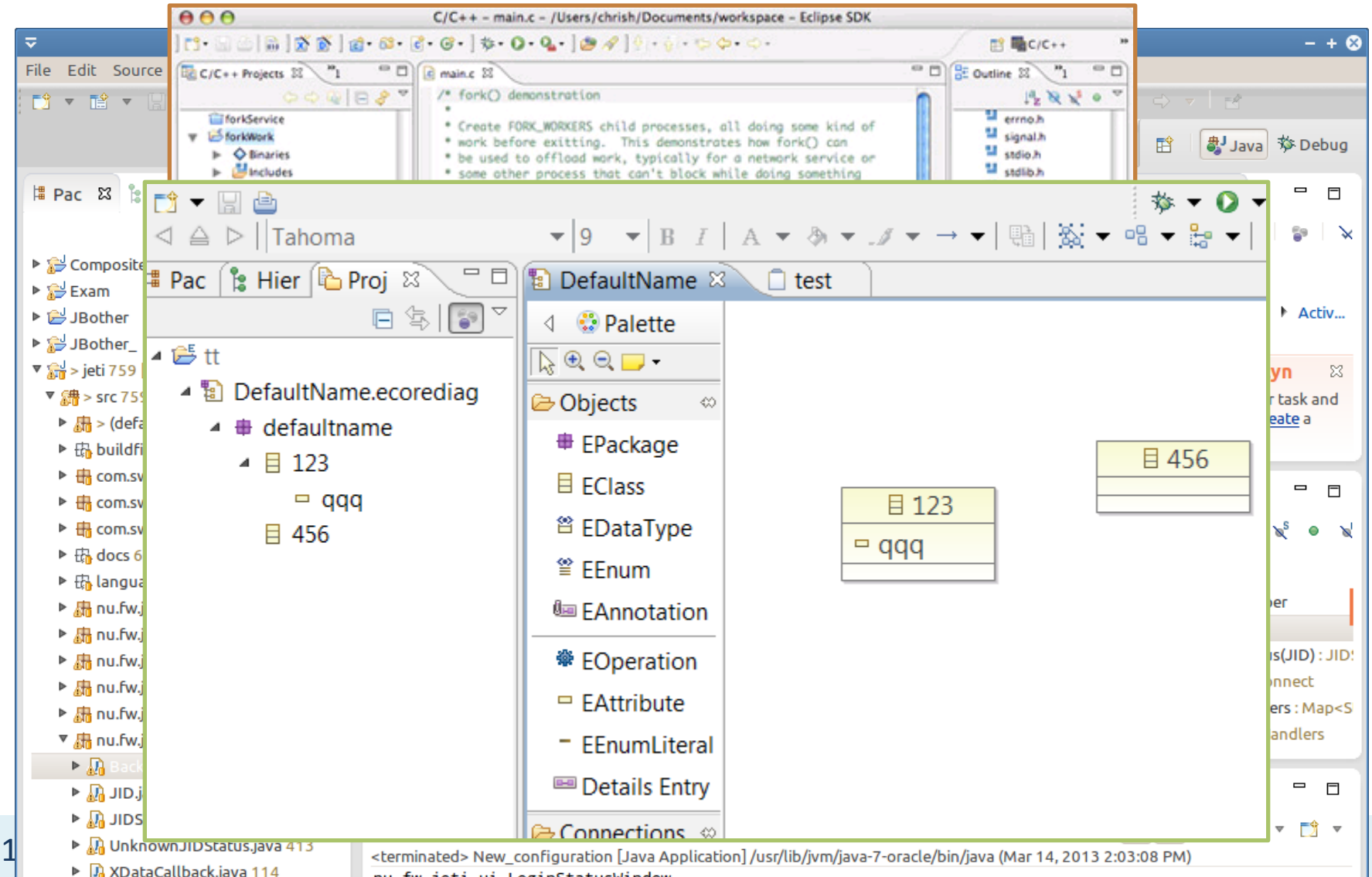
Learning goals for today

- 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

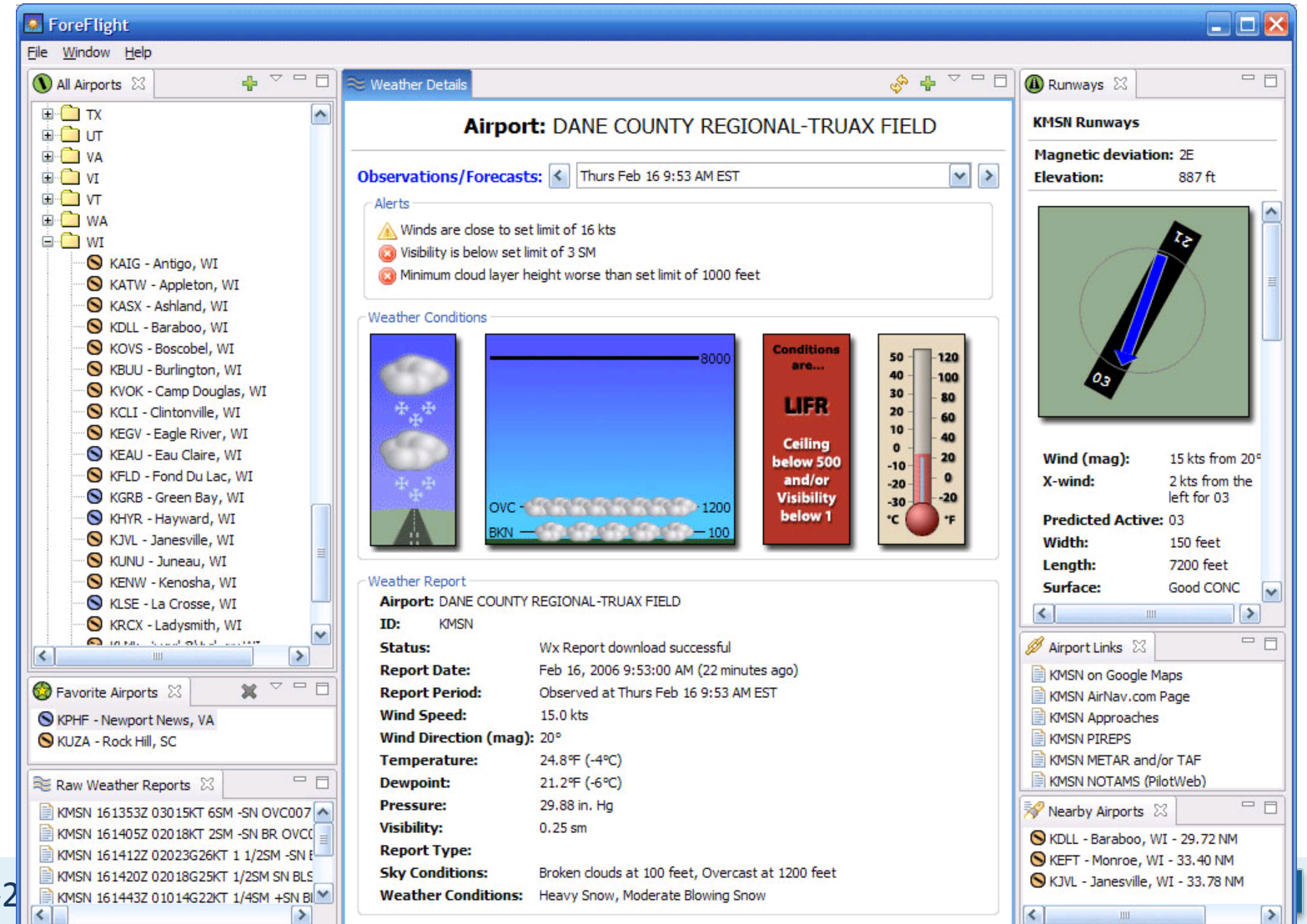
institute for SOFTWARE RESEARCH

Today: Libraries and frameworks for reuse

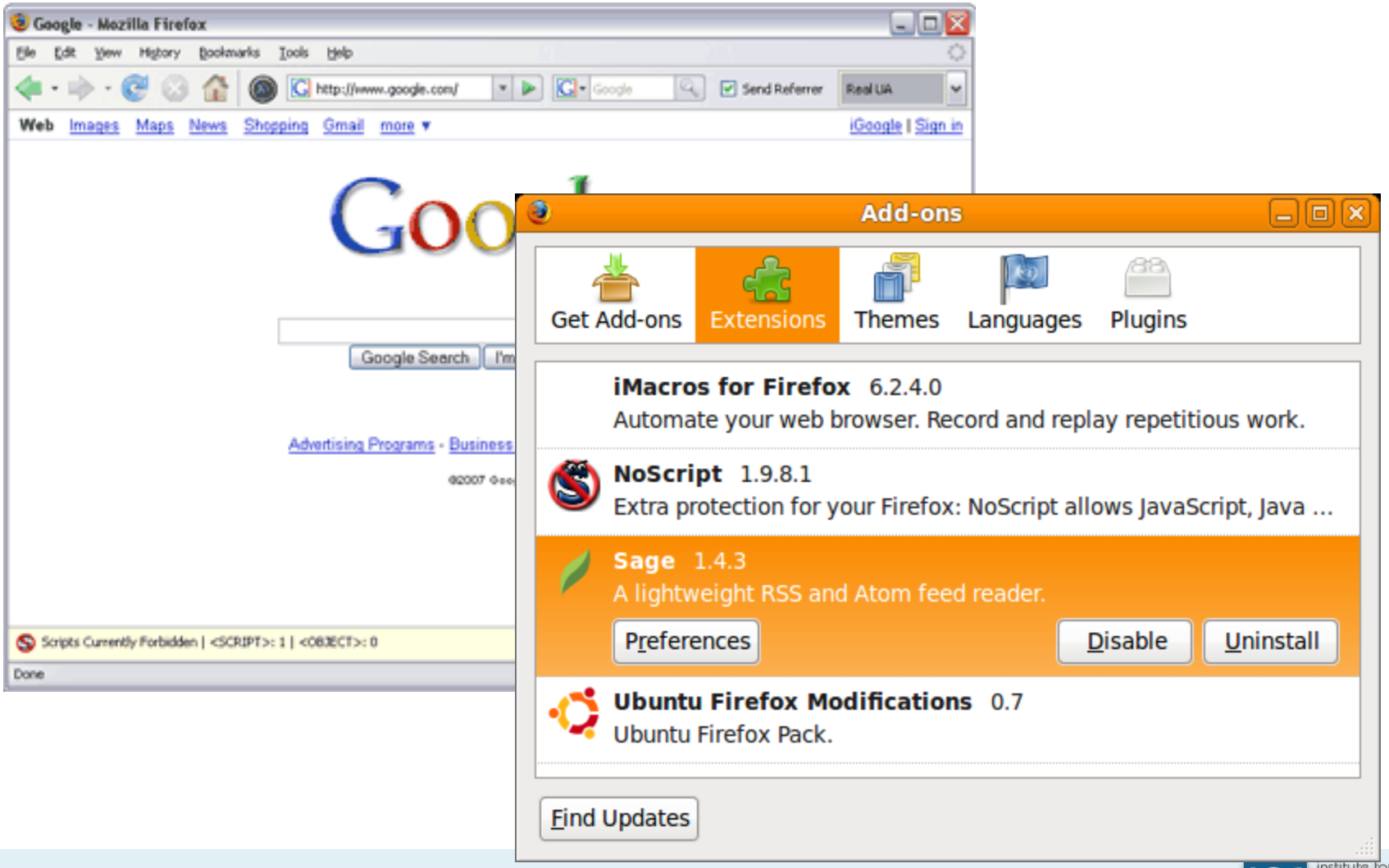
Reuse and variation:
Family of development tools



Reuse and variation:
Eclipse Rich Client Platform

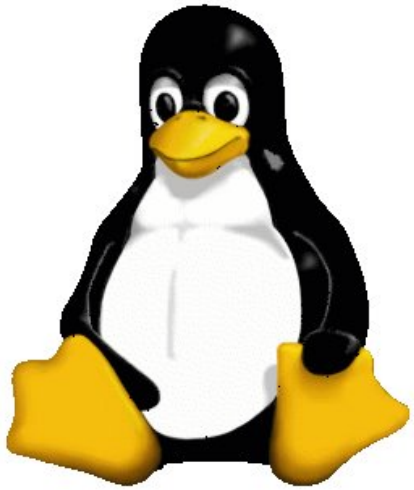


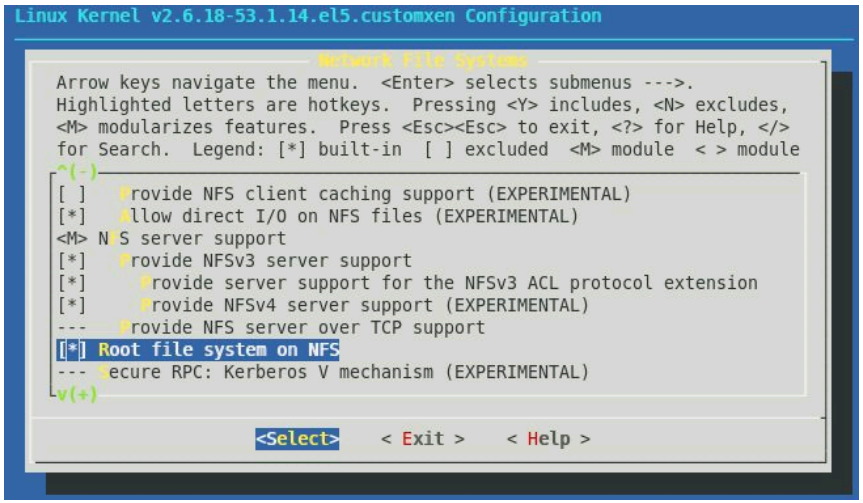
Reuse and variation: Web browser extensions



institute for SOFTWARE RESEARCH

Reuse and variation: Flavors of Linux





institute for SOFTWARE RESEARCH

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, …

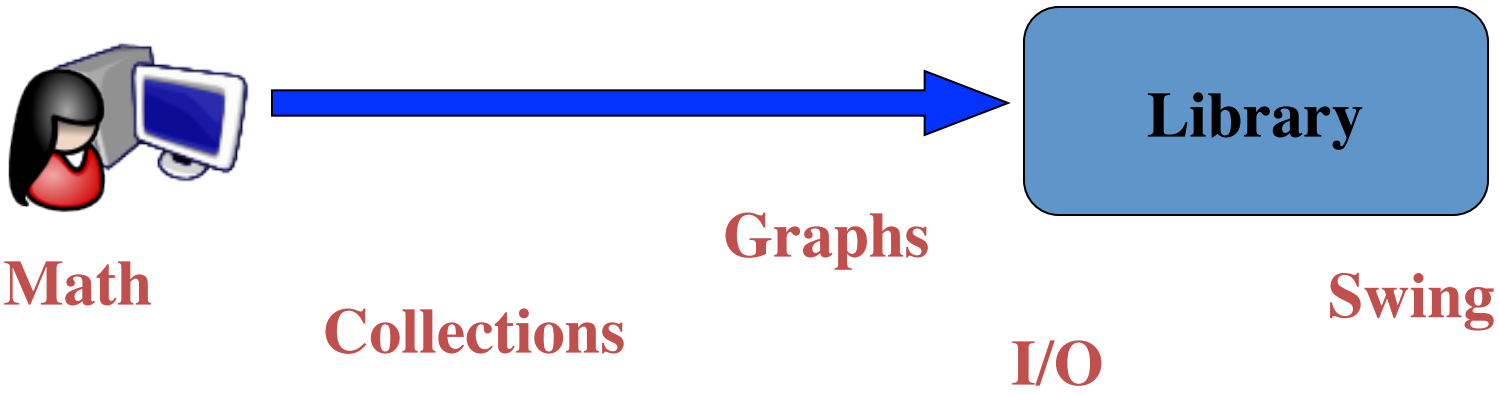
institute for SOFTWARE RESEARCH

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



institute for SOFTWARE RESEARCH

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 {

ublic 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

Framework

Eclipse **Firefox** **Swing**

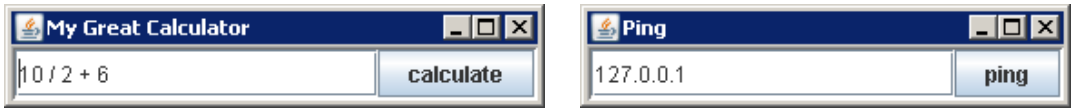
Applet **Spring**

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 getInititalText() { 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 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 {

ublic 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

Library

user interacts

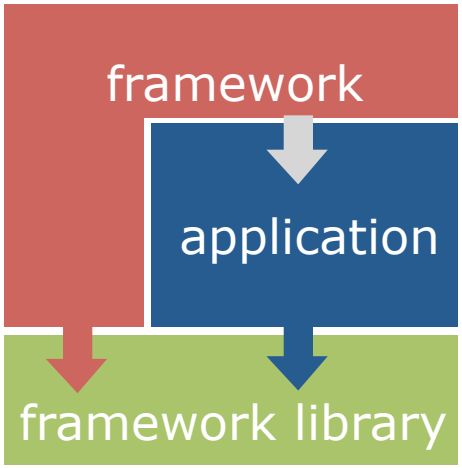
Framework

```
public MyWidget extends JContainer {

ublic 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
- The Java Logging Framework
- Java Encryption Services
- Wordpress
- Django

institute for SOFTWARE RESEARCH

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

institute for SOFTWARE RESEARCH

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

institute for SOFTWARE RESEARCH

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

institute for SOFTWARE RESEARCH

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

institute for SOFTWARE RESEARCH

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 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 JPa
      contentPane.setBorder(new Be
      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);
}
```


institute for SOFTWARE RESEARCH

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 JPa
      contentPane.setBorder(new Be
      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() { retu
  public String getInititalText() { re
  public void buttonClicked() {
      JOptionPane.showMessageDialog(nu
              + application.getInput() + " is "
              + calculate(application.getInput()));
  }
  public String getApplicationTitle() { return "My Great Calculator"; }
}
```

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

```
}
```


institute for SOFTWARE RESEARCH

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

institute for SOFTWARE RESEARCH

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

institute for SOFTWARE RESEARCH

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

institute for SOFTWARE RESEARCH