

Principles of Software Construction: Objects, Design, and Concurrency

Design for large-scale reuse: Libraries and frameworks

Michael Hilton

Bogdan Vasilescu



Administrivia

- Homework 4b due tonight

Key concepts from Tuesday

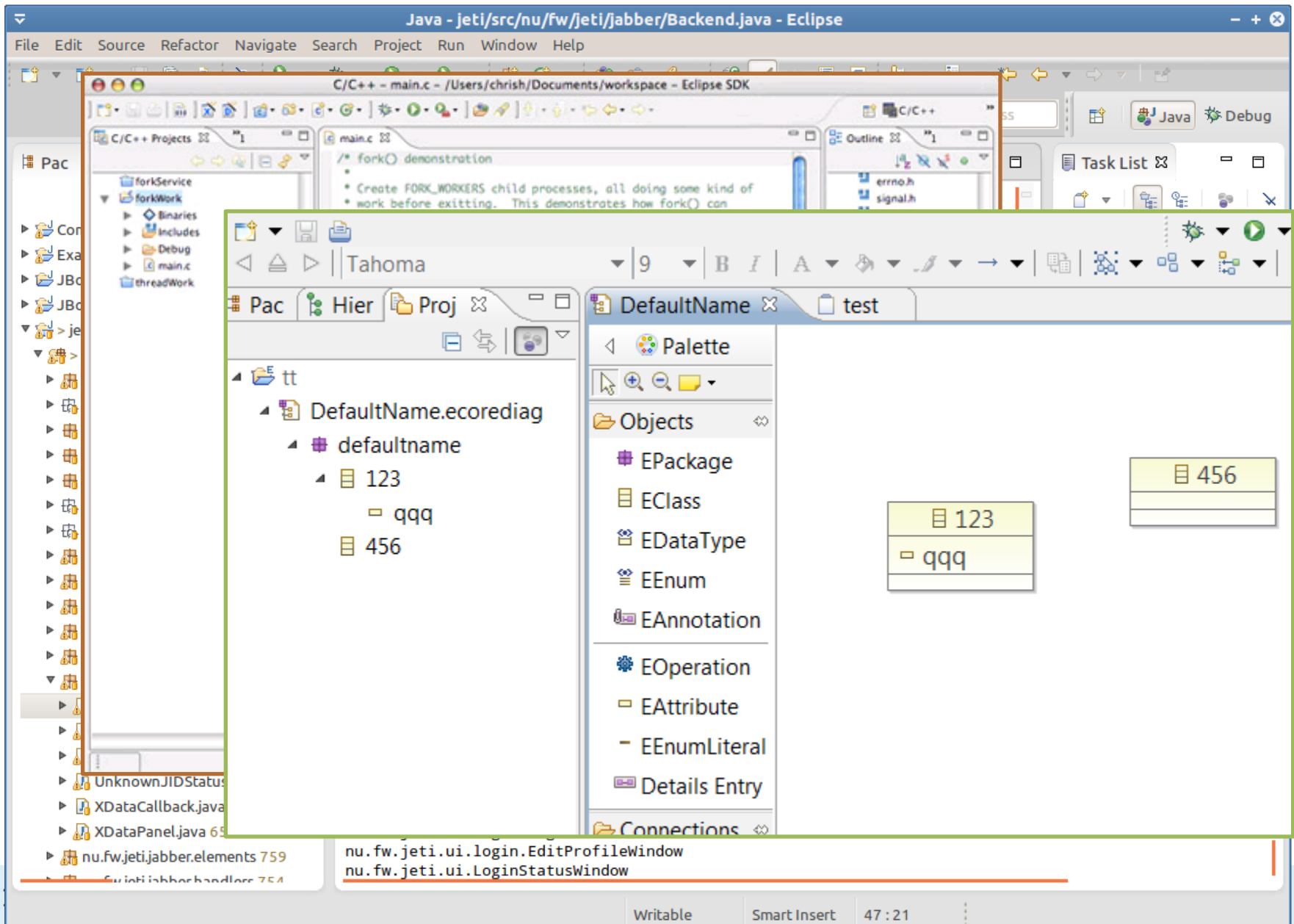
Today: Libraries and frameworks for reuse

Earlier in this course: Class-level reuse

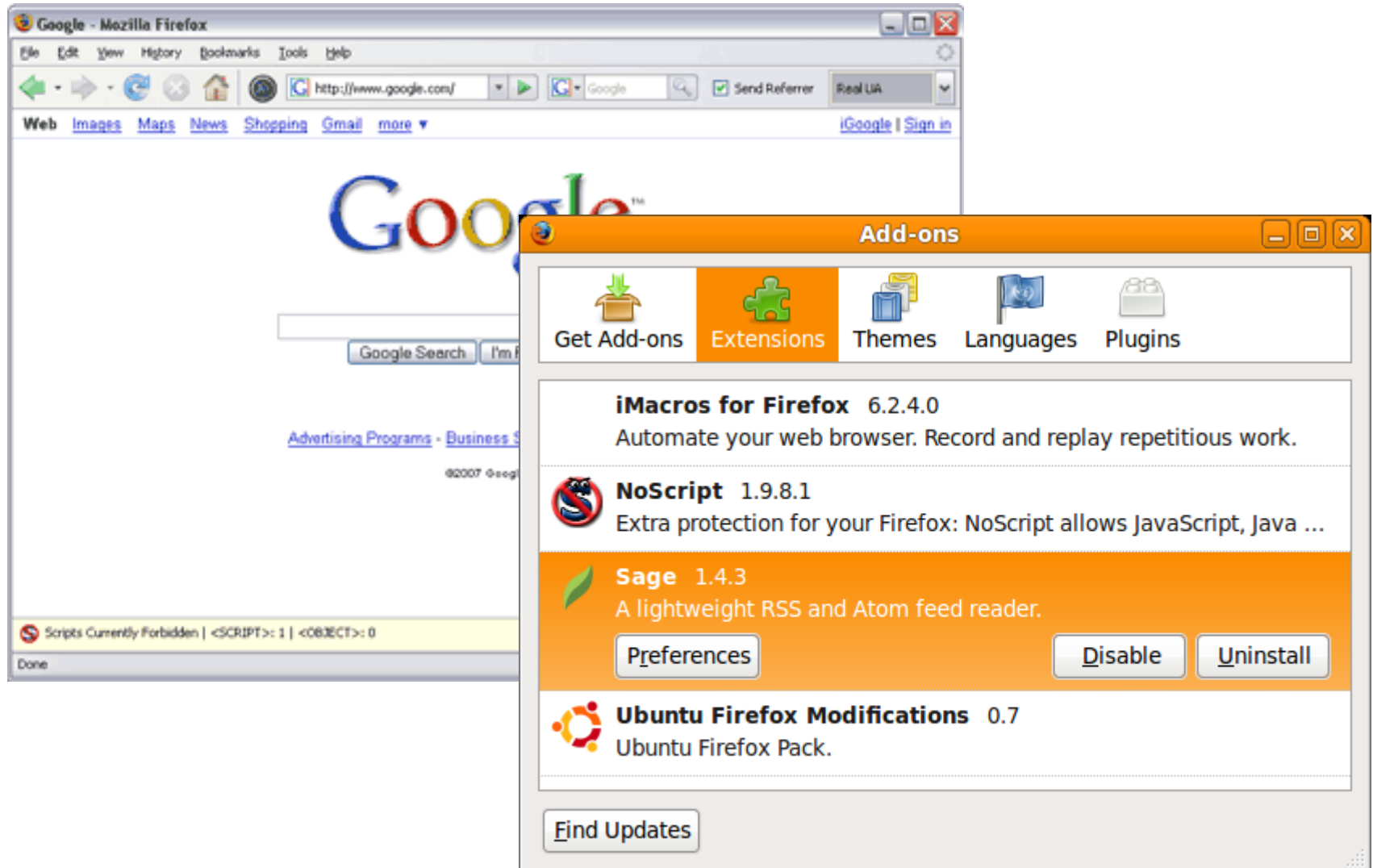
- Language mechanisms supporting reuse
 - Inheritance
 - Subtype polymorphism (dynamic dispatch)
 - Parametric polymorphism (generics)*
- Design principles supporting reuse
 - Small interfaces
 - Information hiding
 - Low coupling
 - High cohesion
- Design patterns supporting reuse
 - Template method,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

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

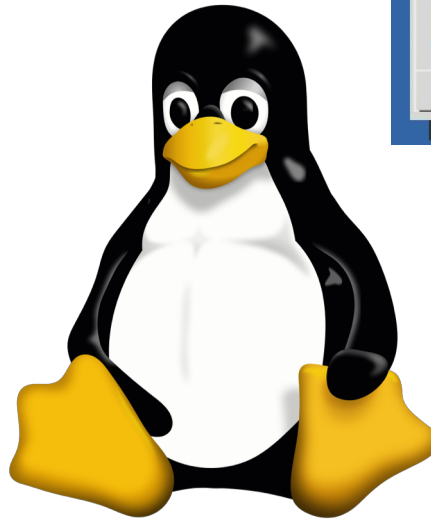
Reuse and variation: Family of development tools



Reuse and variation: 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(+)
```

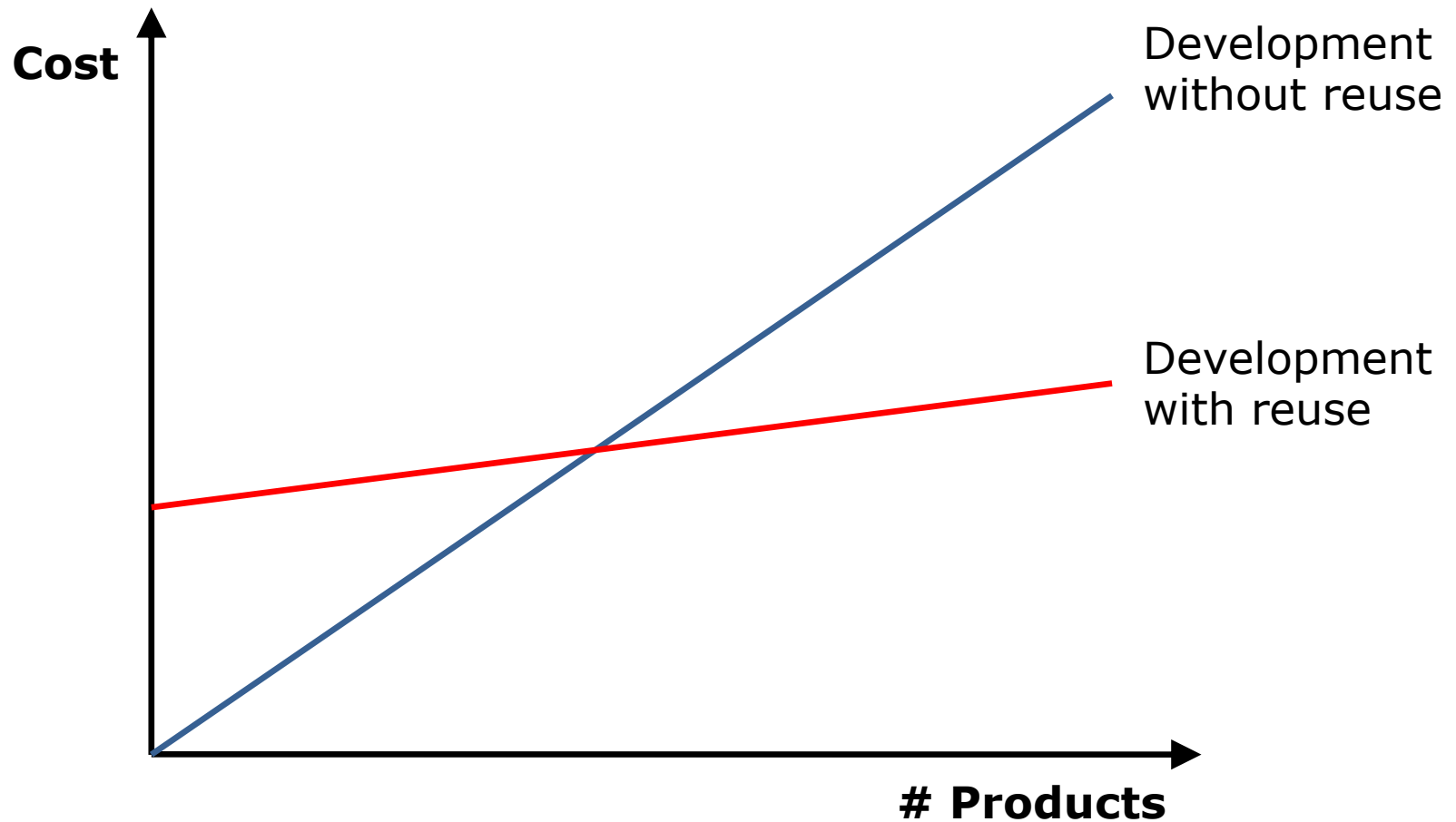
<Select> <Exit> <Help>



Reuse and variation: Product lines



The promise



Today: Libraries and frameworks for reuse

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

Today: Libraries and frameworks for reuse

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

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

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? What code will differ?

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() { ... }  
}
```

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

General distinction: Library vs. framework



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

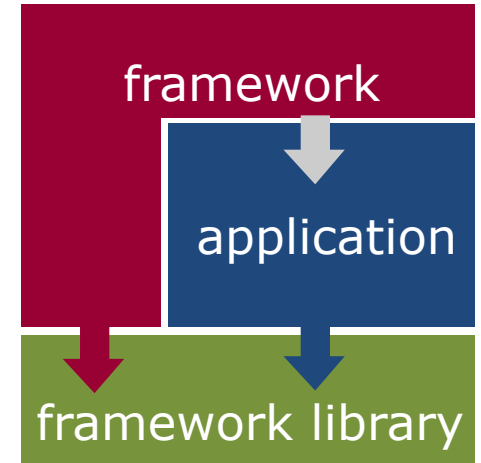
your code



Framework

Libraries and frameworks in practice

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



credit: Erich Gamma

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

Framework or library?

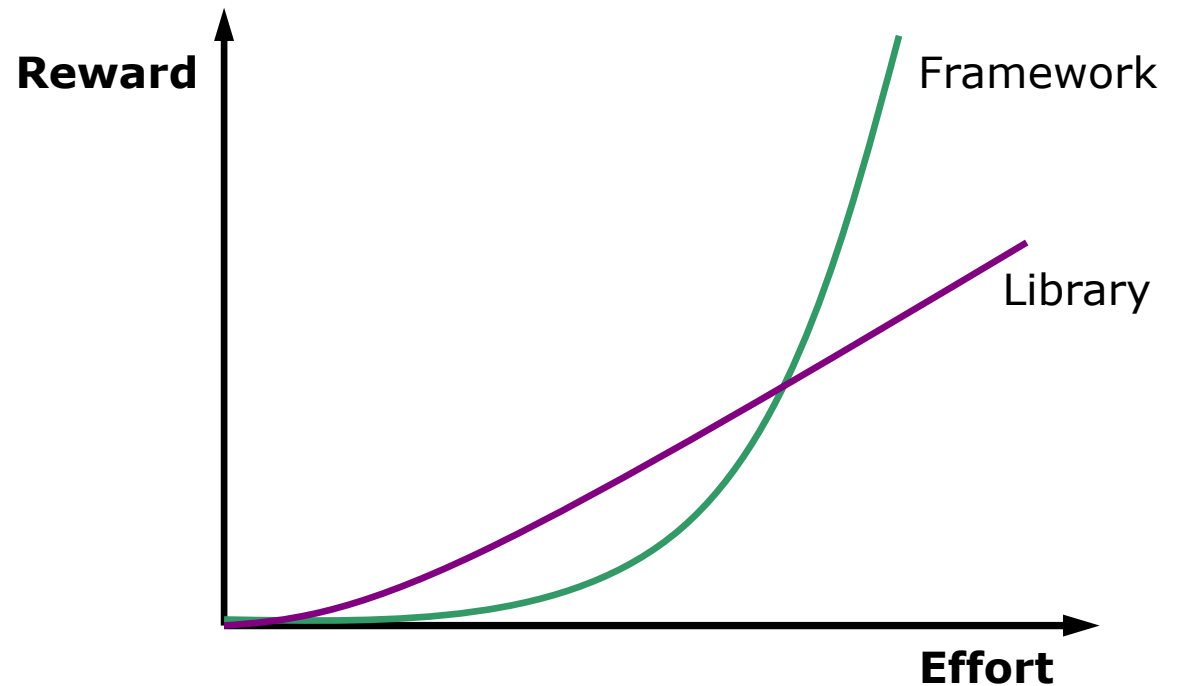
- Eclipse
- Java Collections
- The Java Logging Framework
- Java Encryption Services
- Wordpress
- Django

A Carcassonne framework?



Learning library and framework

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



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
- *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

Today: Libraries and frameworks for reuse

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

Whitebox vs. blackbox framework

Whitebox frameworks	Blackbox frameworks
Subclassing and overriding methods	Composition; implementing a plugin interface
Subclass has <code>main()</code> but gives control to framework	Plugin-loading mechanism loads plugins and gives control to 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 BevelBorder());
        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"; }  
}
```

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 JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        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

Whitebox frameworks	Blackbox frameworks
Subclassing and overriding methods	Composition; implementing a plugin interface
Subclass has the main method but gives control to framework	Plugin-loading mechanism loads plugins and gives control to framework
Need to understand implementation of superclass	Only need to understand interface
Only one extension at a time	Multiple plugins simultaneously
Compiled together	More modularity; separate deployment possible (.jar, .dll, ...)
Common pattern: Template method	Common patterns: Strategy, observer
“Developer frameworks”	“End-user frameworks”

Today: Libraries and frameworks for reuse

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