

Software Product Lines

Part 5: Components and frameworks

Daniel Strüber, Radboud University

with courtesy of: **Sven Apel**, **Christian Kästner**, **Gunter Saake**

Last weeks: Configuration management and preprocessors

▶ Compile-time variability

▶ Version control systems

- ▶ Only useful for a handful of variants, but established
- ▶ Cannot flexibly combine features

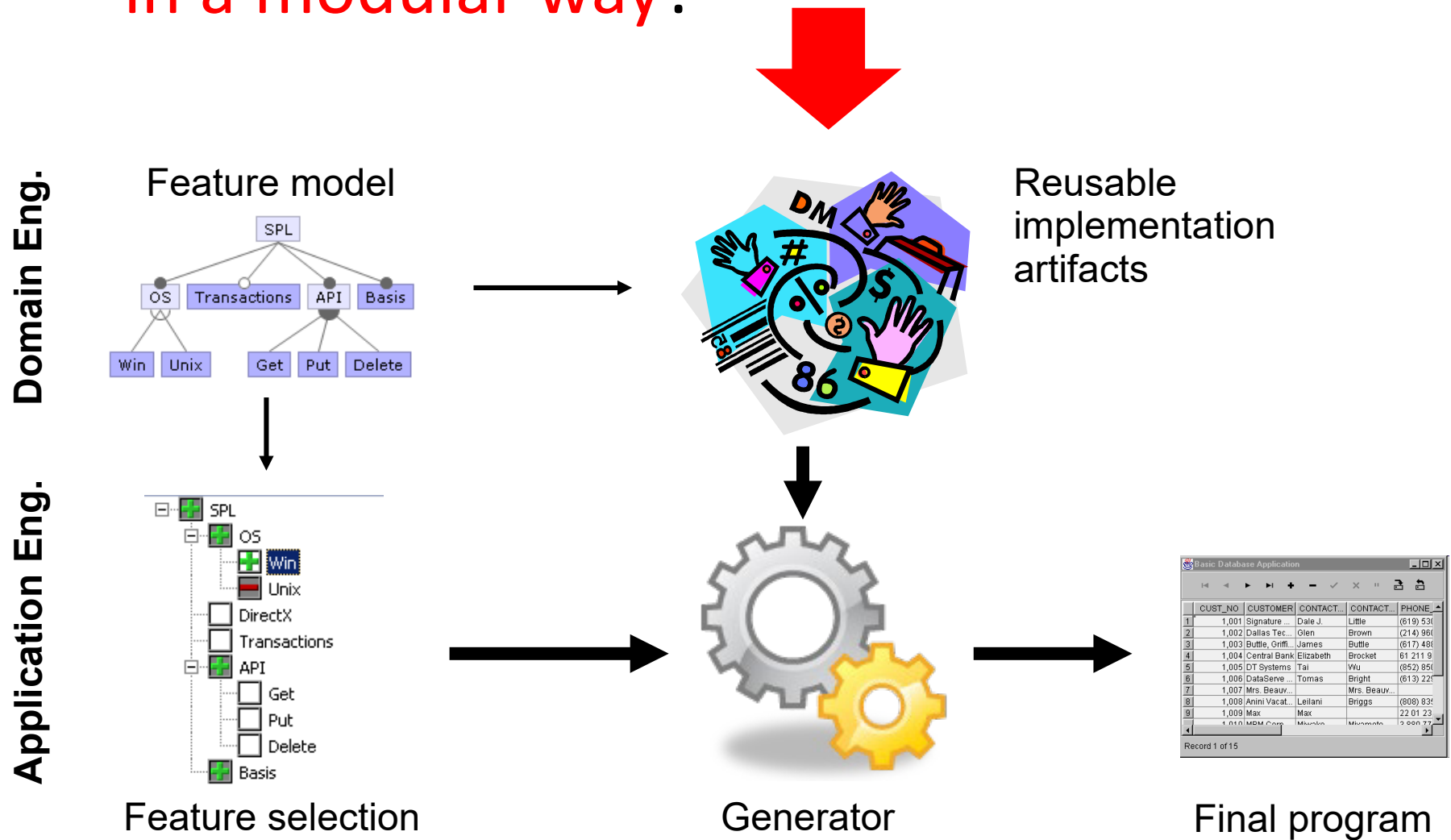
▶ Build systems

- ▶ Simple mechanism, highly flexible
- ▶ Only file-level changes (limited reuse capabilities)

▶ Preprocessors

- ▶ Simple concept: „mark and remove“
- ▶ Standard tools, very flexible, maximally fine-grained, feature-oriented
- ▶ Error-prone, hard to read, scattering/tangling...

How to implement variability in a modular way?



Agenda

- ▶ Components
- ▶ Frameworks
- ▶ Crosscutting concerns
- ▶ Preplanning problem



Components

Components

- ▶ Self-contained modular unit of implementation with interface (black box); offers a „service“
- ▶ Often „assembled“ with other components – even from different vendors – to software system (composition)
- ▶ Ideally: can run and be marketed on its own
- ▶ Explicit definition of context (e.g., JavaEE, COM+/DCOM, OSGi) and dependencies (imports, exports)
- ▶ Size concerns
 - ▶ Small enough to be developed and maintained as one unit
 - ▶ Large enough to offer meaningful functionality

Components vs. objects/classes

- ▶ Similar concepts: encapsulation, interfaces, information hiding
 - ▶ Objects structure a problem
 - ▶ Components offer reusable functionality increments
- ▶ Objects are smaller than components
 - ▶ „Components scale object-orientation“
- ▶ Objects often have dependencies to many other objects; components have fewer dependencies
- ▶ Interfaces of objects are often close to implementation; components abstract more

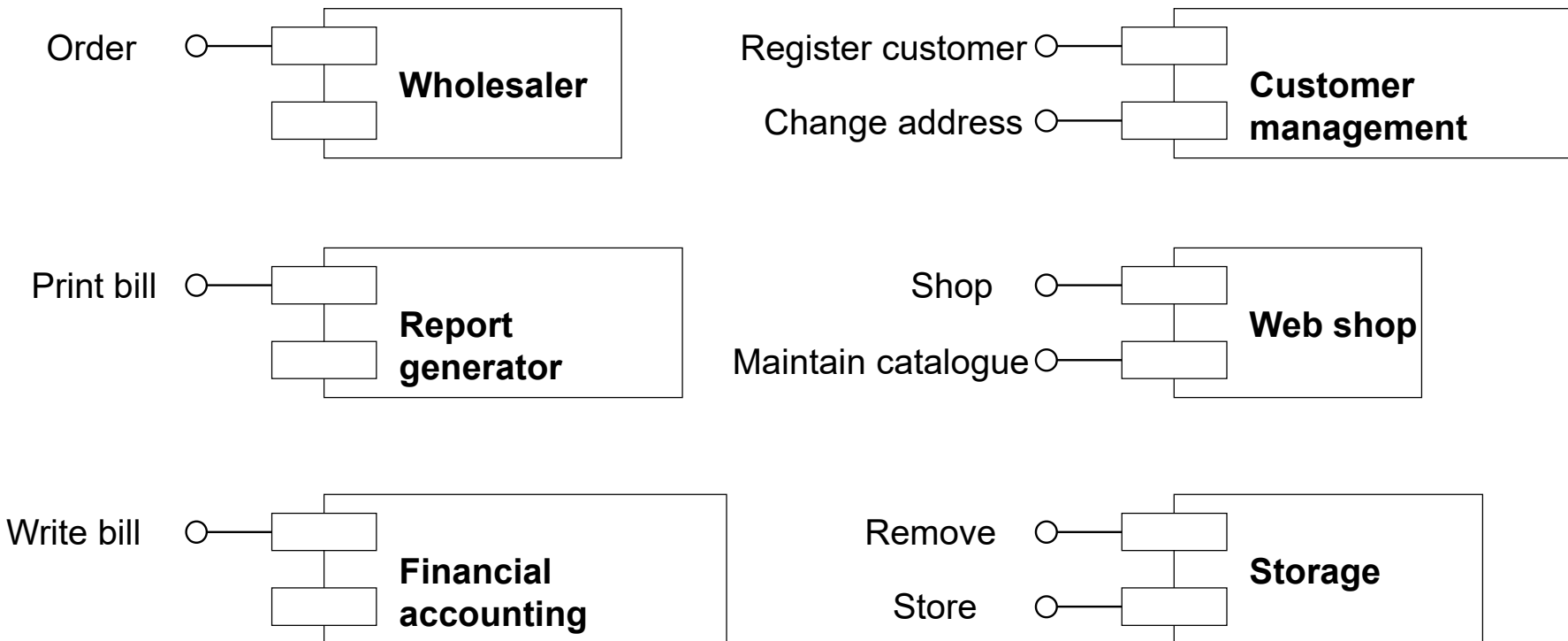
Vision: marketplaces for components

- ▶ Components can be bought and integrated into own systems
- ▶ *Best of Breed*: developer can choose the best supplier for each system part
- ▶ Suppliers can concentrate on a core competency and offer their solutions as components

Components of a web shop

► (UML notation: component diagram)

Scenario: Register customer → Go shopping → Create bill → Print bill



Product lines from components

- ▶ Features are implemented as components
 - ▶ for example, components for transaction management, log/recovery, buffer management, optimisation
 - ▶ Components may include runtime variability
- ▶ Components are retrieved based on feature selection (*mapping*)
- ▶ Developer has to integrate components (*glue code*)

Example: Component „Color“ in Java

```
package modules.colorModule;
//public interface
public class ColorModule {
    public Color createColor(r:Int,g:Int,b:Int) { ...}
    public void printColor(color: Color) {colorPrint... }

    public void mapColor(elem: Object, col: Color)
        { colorMapping...}
    public Color getColor(elem: Object)
        { colorMapping...}

    //just one module instance
    public static ColorModule getInstance()
        { return module; }
    private static ColorModule module =
        new ColorModule();
    private ColorModule() { super(); }
}
public interface Color { ... }

//hidden implementation
class ColorPrinter { ... }
class ColorMapping {...}
```

- ▶ Facade pattern
 - ▶ Hides implementation details
 - ▶ Common interface for many classes
- ▶ Singleton pattern
 - ▶ Only one instance of module

ColorModule.getInstance().createColor(...)

Services

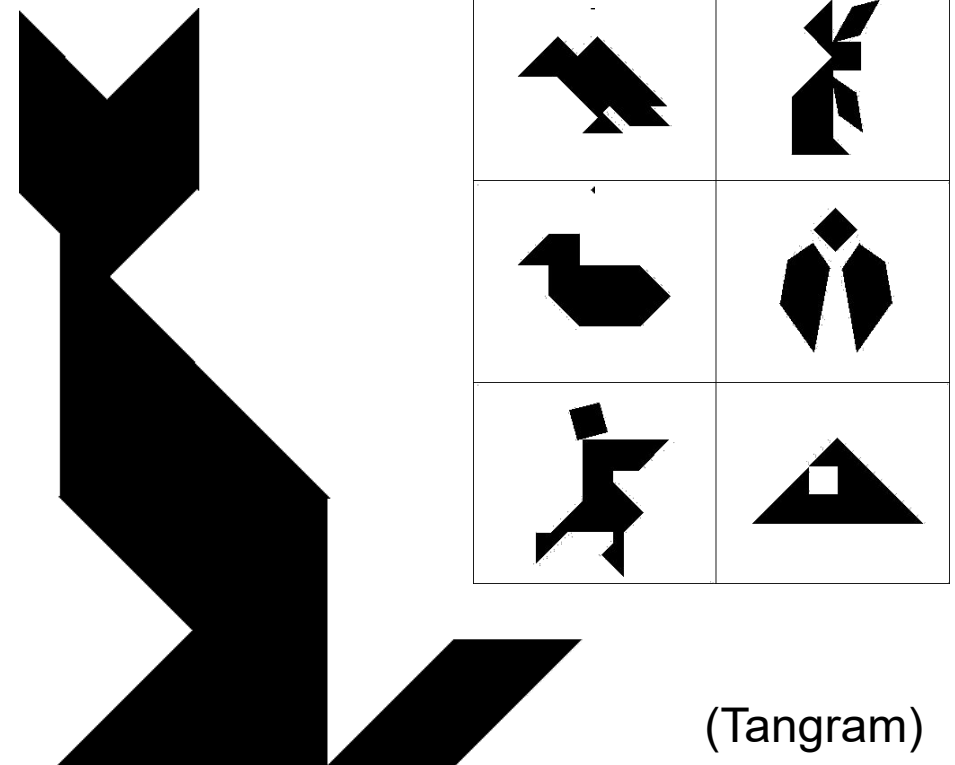
- ▶ Special type of component: encapsulate partial functionality (services) in distributed scenarios
 - ▶ Bus communication, Web Services, SOAP, REST...
- ▶ Abstract from programming languages, use dedicated exchange formats (based on XML, JSON etc.)
- ▶ Product lines via connection of services, usually via orchestration (workflow languages such as BPEL)
- ▶ Many tools available (often “management-oriented”)
- ▶ Aims at high degree of standardisation

How to tailor components?

- ▶ Marketplace for arbitrary components does not work; trade-off *use* vs. *reuse*
 - ▶ Too small → high effort for use (glue code)
 - ▶ Too large → hardly reusable
- ▶ **Example:** Developer searches the web for a software solution for problem, finds 2 solutions. How to decide?
 - ▶ Solution 1: small, 1K LoC, only parts of desired functionality
 - ▶ Solution 2: large, 100K LoC, contains entire functionality, which, however, is tangled with additional, not required functionality, possibly incompatible assumptions

How to tailor components?

- ▶ Without knowing the application context, component developers have to „guess“
- ▶ Solution approach: software product lines offer the required domain analysis
 - ▶ Systematic reuse
 - ▶ Which partial functionality used at which granularity?
 - ▶ Which parts always used together? → become component

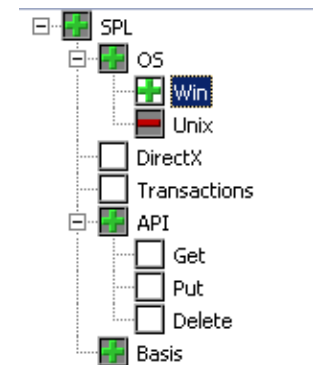
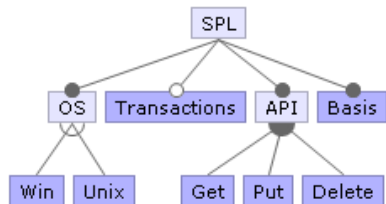


Product lines from components

Mapping
features <-> components

Component repository

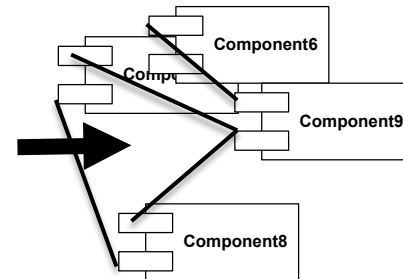
Feature model



Feature selection
as input

Feature selection

Component selection



Component
set

Developer writes
glue code to connect components

A screenshot of a window titled 'Basic Database Application'. It displays a table with columns: CUST_NO, CUSTOMER, CONTACT, CONTACT, and PHONE. The table contains 10 records. The status bar at the bottom indicates 'Record 1 of 15'.

	CUST_NO	CUSTOMER	CONTACT	CONTACT	PHONE
1	1,001	Signature...	Dale J.	Little	(619) 531
2	1,002	Dallas Tec...	Glen	Brown	(214) 961
3	1,003	Buttle, Grm...	James	Buttle	(617) 481
4	1,004	Central Bank	Elizabeth	Brocket	61 211 9
5	1,005	DT Systems	Tai	Wiu	(852) 851
6	1,006	DataServe...	Tomas	Bright	(613) 227
7	1,007	Mrs. Beauv...	Mrs. Beauv...		
8	1,008	Anini Vacat...	Leilani	Briggs	(808) 831
9	1,009	Max	Max		22 01 23
10	1,010	MDM Com...	Musko	Musko	2 00 77

Final program

Assessment:

Product lines from components

- ▶ Widely used in industry (for example Philips home electronics with Koala components)
- ▶ Systematic (planned) reuse of components
- ▶ Reuse in the large
- ▶ Easy division of labor

- ▶ Not fully automated, high development effort in application engineering (glue code)
- ▶ No free feature selection

Discussion: Modularity

- ▶ Components hide internal implementation details
- ▶ Ideally small interfaces
- ▶ Cohesive features

but ...

- ▶ Coarse granularity
 - ▶ Colors, weighting in graph as components?
 - ▶ Paging strategies, search algorithms, B-tree locking, VARCHAR as components?
- ▶ Functionality might be hard to encapsulate
 - ▶ Transaction management component?



Frameworks

Frameworks

- ▶ Incomplete set of abstract and concrete classes
- ▶ Abstract structure that can be instantiated and adapted to specific context
 - ▶ cf. *template method* and *strategy* patterns
- ▶ Reusable solution for a problem family in a domain
- ▶ Dedicated points for extensions:
hot spots (a.k.a. variation points, extension points)
- ▶ Inversion of control, framework decides control flow and execution order
 - ▶ Hollywood principle: „Don't call us, we'll call you.“

Plugins

- ▶ Extension of a framework
- ▶ Add special functions on demand
- ▶ Usually compiled separately; third-party
- ▶ Popular in end-user software
 - ▶ Email programs, graphic editors, media player, web browser

Web Portal

- ▶ Web application frameworks like Struts implement and offer core functionality
- ▶ Developers can concentrate on application logic and navigation between pages

```
<?php
class WebPage {
    function getCSSFiles();
    function getModuleTitle();
    function hasAccess(User u);
    function printPage();
}
?>
```

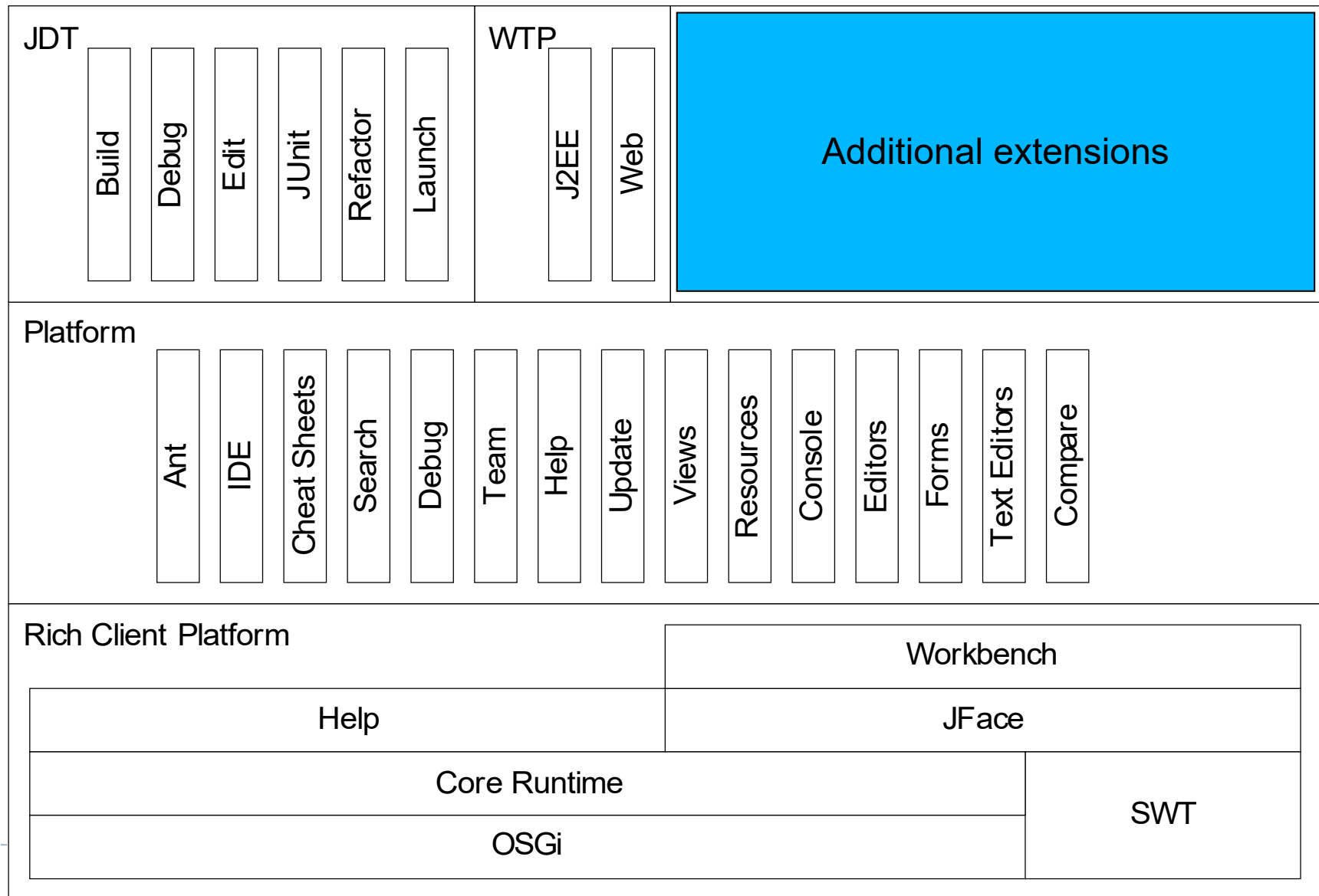
```
<?php
class ConfigPage extends WebPage {
    function getCSSFiles() {...}
    function getModuleTitle() {
        return "Configuration";
    }
    function hasAccess(User u) {
        return user.isAdmin();
    }
    function printPage() {
        print "<form><div>...";
    }
}
?>
```

Eclipse

- ▶ Eclipse as a framework for IDEs
 - ▶ Framework offers common functionality (editors, menus, projects, directory tree, copy & paste, undo, VCS integration, etc.)
 - ▶ Only language-specific extensions (syntax highlighting, compiler, type checking) have to be implemented
 - ▶ Framework from many smaller frameworks



Eclipse

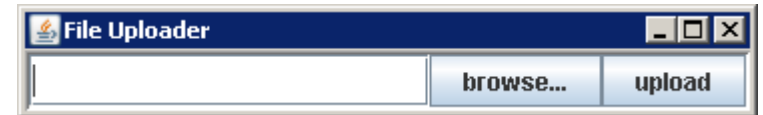


Additional framework examples

- ▶ Frameworks for graphical user interfaces
 - ▶ MacApp, Swing, SWT, MFC
- ▶ Multimedia-Frameworks
 - ▶ DirectX
- ▶ Instant Messenger-Frameworks
 - ▶ Miranda, Trillian, ...
- ▶ Compiler-Frameworks
 - ▶ Polyglot, abc, JastAddJ

Framework implementation: minimal example

- ▶ Family of dialogs with buttons and text fields



- ▶ 90% of source code identical
 - ▶ main method
 - ▶ initialize window, text field, button(s)
 - ▶ layout
 - ▶ close window
 - ▶ ...

Calculator

```
public class Calc extends JFrame {
    private JTextField textfield;
    public static void main(String[] args) { new Calc().setVisible(true); }
    public Calc() { init(); }
    protected void init() {
        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(/* code for calcuting */);
        this.setContentPane(contentPane);
        this.pack();
        this.setLocation(100, 100);
        this.setTitle("My Great Calculator");
        // code for closing window
    }
}
```

White-Box frameworks

- ▶ Extend by overwriting and adding methods
 - ▶ cf. *template method* pattern
- ▶ Implementation developer knows framework internals
 - ▶ → might be difficult to learn
- ▶ (Relatively) flexible extensions
- ▶ Might need many subclasses → hard to overview?
- ▶ Can directly access state of superclasses
- ▶ No plug-ins, not compiled separately

Calculator as a white-box framework

```
public abstract class GuiApplication extends JFrame {
    protected abstract String getApplicationTitle();           // abstract methods
    protected abstract String getButtonText();
    protected String getInititalText() {return "";}
    protected void buttonClicked() { }
    private JTextField textfield;
    public Application() { init(); }
    protected void init() {
        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(getInititalText());
        textfield.setPreferredSize(new Dimension(200, 20));
        contentPane.add(textfield, BorderLayout.WEST);
        button.addActionListener(/* ... buttonClicked(); ... */);
        this.setContentPane(contentPane);
        this.pack();
        this.setLocation(100, 100);
        this.setTitle(getApplicationTitle());
        // code for closing window
    }
    protected String getInput() { return textfield.getText();}
}
```

Calculator as a white-box framework

```
public abstract class GuiApplication extends JFrame {  
    protected abstract String getApplicationTitle();           // abstract methods  
    protected abstract String getButtonText();  
    protected String getInititalText() {return "";}  
    protected void buttonClicked() {}  
    private JTextField textfield;  
    public Application() { init(); }  
    protected void init() {  
        public class Calculator extends GuiApplication {  
            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())); }  
            protected String getApplicationTitle() { return "My Great Calculator"; }  
            public static void main(String[] args) {  
                new Calculator().setVisible(true);  
            }  
        }  
        this.setContentArea(contentArea);  
        this.pack();  
    }  
    protected void setContentArea(JComponent contentArea) {  
        public class Ping extends GuiApplication {  
            protected String getButtonText() { return "ping"; }  
            protected String getInititalText() { return "127.0.0.1"; }  
            protected void buttonClicked() { /* ... */ }  
            protected String getApplicationTitle() { return "Ping"; }  
            public static void main(String[] args) {  
                new Ping().setVisible(true);  
            }  
        }  
    }  
}
```

Modularity?

Black-Box frameworks

- ▶ Embed application-specific behavior via components with a special interface (**plug-ins**)
 - ▶ cf. *strategy* and *observer* patterns
- ▶ Implementation developer only needs to know interface
 - ▶ easier to learn, harder to design
- ▶ Flexibility is determined by the offered hot spots, often implemented with design patterns
- ▶ State only known if available via interface
- ▶ Loose coupling (esp. compared to white-box frameworks)

Calculator

```
public class GuiApplication extends JFrame {
    private JTextField textfield;
    private Plugin plugin;
    public GuiApplication(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("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(/* ... plugin.buttonClicked();... */);
        this.setContentPane(contentPane);
        ...
    }
    public String getInput() { return textfield.getText();}
}
```

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

Calculator

```
public class GuiApplication extends JFrame {
    private JTextField textfield;
    private Plugin plugin;
    public GuiApplication(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("ok");
        contentPane.add(button, BorderLayout.EAST);
        JTextField textField = new JTextField("");
        if (plugin != null)
            textField.setText(plugin.getInitialText());
        textfield.setText("");
        contentPane.add(textfield, BorderLayout.WEST);
        this.setContentPane(contentPane);
        ...
    }
    public String getInput() {
        return textfield.getText();
    }
}
```

Modularity?

Application does not know plug-ins

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

```
public class CalcPlugin implements Plugin {
    private GuiApplication application;
    public void setApplication(GuiApplication app) { this.application = 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.getText()));
    }
    public String getApplicationTitle() { return "My Great Calculator"; }
}
```

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


Further decoupling possible

```
public class GuiApplication extends JFrame implements InputProvider {  
    private JTextField textfield;  
    private Plugin plugin;  
    public GuiApplication(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("ok");
```

```
        contentPane.add(button, BorderLayout.EAST);
```

```
        textfield.setText("");
```

```
        textfield.addActionListener(this);
```

```
        textfield.setText(plugin.getInitialText());
```

```
        textfield.  
        contentP  
        if (plugin
```

```
        this.setC  
        ...  
    }  
    public String getInput()  
}
```

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

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

```
public class CalcPlugin implements Plugin {  
    private InputProvider application;  
    public void setApplication(InputProvider app) { this.application = 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"; }  
}
```

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

Modularity?

Only plug-in and InputProvider interface

Loading of plug-ins

- ▶ Typical in many frameworks: plugin loader ...
 - ▶ ... searches directory for dll/jar/xml files
 - ▶ ... tests if the file implements a plug-in
 - ▶ ... checks dependencies
 - ▶ ... initializes plug-in on loading
- ▶ Often additional GUI for plug-in installation and configuration
- ▶ Examples:
 - ▶ Eclipse (plugin directory + Jar)
 - ▶ Firefox (plugin directory + DLL)
- ▶ Alternative: determine plug-ins in config file or create a launcher program

Example plugin loader (using Java reflection)

```
public class Starter {  
  
    public static void main(String[] args) {  
        if (args.length != 1)  
            System.out.println("Plugin name not specified");  
        else {  
            String pluginName = args[0];  
            try {  
                Class<?> pluginClass = Class.forName(pluginName);  
                new Application((Plugin) pluginClass.newInstance())  
                    .setVisible(true);  
            } catch (Exception e) {  
                System.out.println("Cannot load plugin " + pluginName  
                    + ", reason: " + e);  
            }  
        }  
    }  
}
```

Multiple plug-ins

- ▶ cf. *observer* pattern
- ▶ Load and register multiple plug-ins
- ▶ On event, inform all plug-ins

- ▶ For different tasks:
more specific plug-in
interfaces

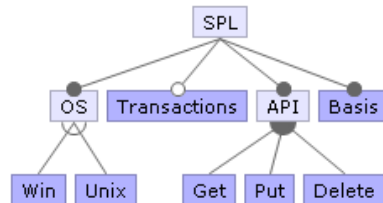
```
public class Application {  
    private List<Plugin> plugins;  
    public Application(List<Plugin> plugins) {  
        this.plugins = plugins;  
        for (Plugin plugin : plugins)  
            plugin.setApplication(this);  
    }  
    public Message processMsg (Message msg) {  
        for (Plugin plugin : plugins)  
            msg = plugin.process(msg);  
        ...  
        return msg;  
    }  
}
```

Frameworks for product lines

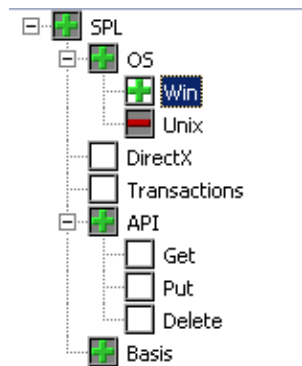
Domain Eng.

Application Eng.

Feature model



Feature selection



Mapping
features <-> plug-ins

Framework + plug-ins



Feature selection
as input



plug-in selection
(and possibly, generation
of launch configuration)

application =
framework with
desired plug-ins

Record 1 of 15

	CUST_NO	CUSTOMER	CONTACT	CONTACT	PHONE
1	1,001	Signature ...	Dale J.	Little	(619) 531
2	1,002	Dallas Tec...	Glen	Brown	(214) 961
3	1,003	Bottle, Grif...	James	Bottle	(617) 491
4	1,004	Central Bank	Elizabeth	Brocket	61 211 9
5	1,005	DT Systems	Tai	VWU	(852) 851
6	1,006	DataServe ...	Tomas	Bright	(613) 221
7	1,007	Mrs. Beau...		Mrs. Beau...	
8	1,008	Anini Vacat...	Lellani	Briggs	(808) 831
9	1,009	Max	Max		22 01 23
10	1,010	MDM Corp	Mundin	Mundin	2 000 77

Assessment

Frameworks for product lines

▶ Benefits

- ▶ Fully automation possible
- ▶ Modularity
- ▶ Tested in practice

▶ Drawbacks

- ▶ Development effort
 - ▶ Runtime overhead for framework architecture
 - ▶ Needs preplanning and requires suitable experience
 - ▶ Evolution and maintenance complicated
- ▶ Coarse granularity and large interfaces



Crosscutting concerns

Crosscutting concerns

- ▶ Claim: Not all concerns of a program can be modularized using objects, components, plugins
 - ▶ Applies to features as well, which are one type of concern
- ▶ Concerns are semantic units
- ▶ But their implementation can be scattered throughout the source code

Crosscutting concerns - example

```
class BusinessClass
//... data fields
//... logging stream
//... cache status
public void importantOperation(
    Data data, User currentUser, ...){
    // check authorization
    // lock objects for synchronization
    // check if buffer up-to-date
    // log start of actual operation
    // execute actual operation
    // log end of actual operation
    // unlock objects
}

public void alsoImportantOperation(
    OtherData data, User currentUser, ...){
    // check authorization
    // lock objects for synchronization
    // check if buffer up-to-date
    // log start of actual operation
    // execute actual operation
    // log end of actual operation
    // unlock objects
}
}
```

- ▶ Code for different concerns scattered
- ▶ Code replicated
- ▶ Operations in this example are modular, but **locking**, **logging**, **buffer** and **authentication** not

Scattering and tangling

▶ Code scattering

- ▶ Code that belongs to a concern is not modularized, but spread throughout the entire program
- ▶ Frequently copied code (e.g., redundant calls of a method)
- ▶ Or spread implementation of parts of the concern

▶ Code tangling

- ▶ Code that belongs to several concerns is jumbled within one class or method

Scattered Code

Code Scattering

```
class Graph {  
    Vector nv = new Vector(); Vector e;  
    Edge add(Node n, Node m) {  
        Edge e = new Edge(n, m);  
        nv.add(n); nv.add(m); ev.add(e);  
        if (Conf.WEIGHTED) e.weight = new Weight();  
        return e;  
    }  
    Edge add(Node n, Node m, Weight w)  
        if (!Conf.WEIGHTED) throw RuntimeException();  
        Edge e = new Edge(n, m);  
        nv.add(n); nv.add(m); ev.add(e);  
        e.weight = w; return e;  
    }  
    void print() {  
        for(int i = 0; i < ev.size(); i++) {  
            ((Edge)ev.get(i)).print();  
        }  
    }  
}
```

```
class Node {  
    new Color();  
    if (Conf.COLORED) Color.setDisplayColor(color);  
    System.out.print(id);  
}
```

```
class Edge {  
    Node a, b;  
    Color color = new Color();  
    Weight weight;  
    Edge(Node _a, Node _b) { a = _a; b = _b; }  
    void print() {  
        if (Conf.COLORED) Color.setDisplayColor(color);  
        a.print(); b.print();  
        if (!Conf.WEIGHTED) weight.print();  
    }  
}
```

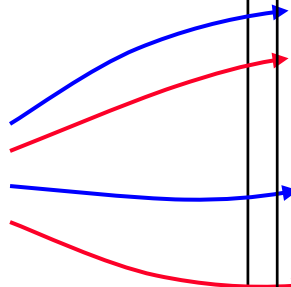
```
class Color {  
    static void setDisplayColor(Color c) { ... }  
}
```

```
class Weight { void print() { ... } }
```

Tangled Code

```
class Graph {
    Vector nv = new Vector(); Vector ev = new Vector();
    Edge add(Node n, Node m) {
        Edge e = new Edge(n, m);
        nv.add(n); nv.add(m); ev.add(e);
        if (Conf.WEIGHTED) e.weight = new Weight();
        return e;
    }
    Edge add(Node n, Node m, Weight w)
        if (!Conf.WEIGHTED) throw RuntimeException();
        Edge e = new Edge(n, m);
        nv.add(n); nv.add(m); ev.add(e);
        e.weight = w; return e;
    }
    void print() {
        for (Node n : nv) {
            for (Node m : nv) {
                if (n != m) {
                    Edge e = add(n, m);
                    e.print();
                }
            }
        }
    }
}
```

Code Tangling



```
class Node {
    int id = 0;
    Color color = new Color();
    void print() {
        if (Conf.COLORED) Color.setDisplayColor(color);
        System.out.print(id);
    }
}
```

```
class Edge {
    Node a, b;
    Color color = new Color();
    Weight weight;
    Edge(Node _a, Node _b) { a = _a; b = _b; }
    void print() {
        if (Conf.COLORED) Color.setDisplayColor(color);
        a.print(); b.print();
        if (!Conf.WEIGHTED) weight.print();
    }
}
```

```
class Color {
    static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

A question of size

Example: Session expiration in the Apache Tomcat Server

ApplicationSession

[illegible]

StandardSession

[illegible]

ServerSession

[illegible]

SessionInterceptor

[illegible]

StandardManager

[illegible]

StandardSessionManager

[illegible]

ServerSessionManager

```

1  #!/usr/bin/perl -w
2
3  use strict;
4  use warnings;
5
6  my $script = $0;
7  my $version = "1.0.0";
8  my $author = "John Doe";
9  my $email = "john.doe@example.com";
10
11  my $help = "
12  Usage: perl $script [options]
13
14  Options:
15  -h, --help            Display this help message
16  -v, --version          Display version information
17  -a, --author           Display author information
18  -e, --email            Display email address
19  -f, --file FILE        Process FILE
20  -o, --output FILE      Write output to FILE
21  -s, --size SIZE        Set size to SIZE
22  -t, --type TYPE        Set type to TYPE
23  -d, --debug            Enable debug mode
24  -q, --quiet            Suppress output
25  -n, --no-color          Disable color output
26  -m, --mode MODE        Set mode to MODE
27  -c, --config FILE      Use config file FILE
28  -p, --port PORT        Set port to PORT
29  -u, --url URL           Set URL to URL
30  -i, --interval INT     Set interval to INT
31  -r, --rate RATE         Set rate to RATE
32  -l, --limit LIMIT       Set limit to LIMIT
33  -w, --wait WAIT         Set wait time to WAIT
34  -x, --xargs ARGS...     Pass arguments to xargs
35  -y, --yes              Assume yes to all questions
36  -z, --zero              Separate entries with NUL
37  -Z, --zip              Compress output with zip
38  -A, --append            Append to existing file
39  -B, --backup            Create backup of existing file
40  -C, --clear             Clear existing file
41  -D, --dry-run           Show what would be done
42  -E, --error             Show error messages
43  -F, --force             Force action
44  -G, --group GROUP       Set group to GROUP
45  -H, --help-text         Display full help text
46  -I, --include INCLUDE   Include INCLUDE
47  -J, --join JOIN          Join with JOIN
48  -K, --key KEY            Set key to KEY
49  -L, --language LANG      Set language to LANG
50  -M, --max MAX            Set maximum to MAX
51  -N, --min MIN            Set minimum to MIN
52  -O, --optimize OPT       Set optimization to OPT
53  -P, --pattern PATTERN    Set pattern to PATTERN
54  -Q, --quote QUOTE        Set quoting to QUOTE
55  -R, --range RANGE        Set range to RANGE
56  -S, --sort SORT          Set sorting to SORT
57  -T, --test              Run tests
58  -U, --update             Update to latest version
59  -V, --verbose            Show verbose output
60  -W, --width WIDTH        Set width to WIDTH
61  -X, --xargs-opts OPTS    Set xargs options to OPTS
62  -Y, --yes-no YES-NO     Set yes/no to YES-NO
63  -Z, --zip-level LEVEL    Set zip level to LEVEL
64  -A, --append-mode MODE   Set append mode to MODE
65  -B, --backup-mode MODE   Set backup mode to MODE
66  -C, --clear-mode MODE    Set clear mode to MODE
67  -D, --dry-run-mode MODE  Set dry-run mode to MODE
68  -E, --error-mode MODE    Set error mode to MODE
69  -F, --force-mode MODE    Set force mode to MODE
70  -G, --group-mode MODE    Set group mode to MODE
71  -H, --help-text-mode MODE Set help-text mode to MODE
72  -I, --include-mode MODE  Set include mode to MODE
73  -J, --join-mode MODE     Set join mode to MODE
74  -K, --key-mode MODE      Set key mode to MODE
75  -L, --language-mode MODE Set language mode to MODE
76  -M, --max-mode MODE     Set max mode to MODE
77  -N, --min-mode MODE      Set min mode to MODE
78  -O, --optimize-mode MODE Set optimize mode to MODE
79  -P, --pattern-mode MODE  Set pattern mode to MODE
80  -Q, --quote-mode MODE    Set quote mode to MODE
81  -R, --range-mode MODE    Set range mode to MODE
82  -S, --sort-mode MODE     Set sort mode to MODE
83  -T, --test-mode MODE     Set test mode to MODE
84  -U, --update-mode MODE   Set update mode to MODE
85  -V, --verbose-mode MODE  Set verbose mode to MODE
86  -W, --width-mode MODE    Set width mode to MODE
87  -X, --xargs-opts-mode MODE Set xargs-opts mode to MODE
88  -Y, --yes-no-mode MODE   Set yes-no mode to MODE
89  -Z, --zip-level-mode MODE Set zip-level mode to MODE
90  -A, --append-mode MODE   Set append mode to MODE
91  -B, --backup-mode MODE   Set backup mode to MODE
92  -C, --clear-mode MODE    Set clear mode to MODE
93  -D, --dry-run-mode MODE  Set dry-run mode to MODE
94  -E, --error-mode MODE    Set error mode to MODE
95  -F, --force-mode MODE    Set force mode to MODE
96  -G, --group-mode MODE    Set group mode to MODE
97  -H, --help-text-mode MODE Set help-text mode to MODE
98  -I, --include-mode MODE  Set include mode to MODE
99  -J, --join-mode MODE     Set join mode to MODE
100 -K, --key-mode MODE      Set key mode to MODE
101 -L, --language-mode MODE Set language mode to MODE
102 -M, --max-mode MODE     Set max mode to MODE
103 -N, --min-mode MODE      Set min mode to MODE
104 -O, --optimize-mode MODE Set optimize mode to MODE
105 -P, --pattern-mode MODE  Set pattern mode to MODE
106 -Q, --quote-mode MODE    Set quote mode to MODE
107 -R, --range-mode MODE    Set range mode to MODE
108 -S, --sort-mode MODE     Set sort mode to MODE
109 -T, --test-mode MODE     Set test mode to MODE
110 -U, --update-mode MODE   Set update mode to MODE
111 -V, --verbose-mode MODE  Set verbose mode to MODE
112 -W, --width-mode MODE    Set width mode to MODE
113 -X, --xargs-opts-mode MODE Set xargs-opts mode to MODE
114 -Y, --yes-no-mode MODE   Set yes-no mode to MODE
115 -Z, --zip-level-mode MODE Set zip-level mode to MODE
116
117  Examples:
118  perl $script -f input.txt -o output.txt
119  perl $script -s 1024 -t text -d
120  perl $script -c config.ini -p 8080 -u http://example.com
121  perl $script -i 10 -r 100 -l 1000 -w 10
122  perl $script -x 'ls -la' -y
123  perl $script -z -Z
124  perl $script -A -B -C -D -E -F -G -H -I -J -K -L -M -N -O -P -Q -R -S -T -U -V -W -X -Y -Z
125
126  Copyright (c) 2023 John Doe. All rights reserved.
127  License: MIT
128  "
129
130  my $opts = {
131      help: 0,
132      version: 0,
133      author: 0,
134      email: 0,
135      file: "",
136      output: "",
137      size: 0,
138      type: "",
139      debug: 0,
140      quiet: 0,
141      no_color: 0,
142      mode: "",
143      config: "",
144      port: 0,
145      url: "",
146      interval: 0,
147      rate: 0,
148      limit: 0,
149      wait: 0,
150      xargs: "",
151      yes: 0,
152      zero: 0,
153      zip: 0,
154      append: 0,
155      backup: 0,
156      clear: 0,
157      dry_run: 0,
158      error: 0,
159      force: 0,
160      group: "",
161      help_text: 0,
162      include: "",
163      join: "",
164      key: "",
165      language: "",
166      max: 0,
167      min: 0,
168      optimize: "",
169      pattern: "",
170      quote: "",
171      range: "",
172      sort: "",
173      test: 0,
174      update: 0,
175      verbose: 0,
176      width: 0,
177      xargs_opts: "",
178      yes_no: "",
179      zip_level: 0,
180      append_mode: "",
181      backup_mode: "",
182      clear_mode: "",
183      dry_run_mode: "",
184      error_mode: "",
185      force_mode: "",
186      group_mode: "",
187      help_text_mode: "",
188      include_mode: "",
189      join_mode: "",
190      key_mode: "",
191      language_mode: "",
192      max_mode: "",
193      min_mode: "",
194      optimize_mode: "",
195      pattern_mode: "",
196      quote_mode: "",
197      range_mode: "",
198      sort_mode: "",
199      test_mode: "",
200      update_mode: "",
201      verbose_mode: "",
202      width_mode: "",
203      xargs_opts_mode: "",
204      yes_no_mode: "",
205      zip_level_mode: ""
206  };
207
208  my $args = @ARGV;
209  my $i = 0;
210  while ($i < $args) {
211      my $arg = $args[$i];
212      if ($arg =~ /^--help$/) {
213          print $help;
214          exit 0;
215      }
216      if ($arg =~ /^--version$/) {
217          print $version;
218          exit 0;
219      }
220      if ($arg =~ /^--author$/) {
221          print $author;
222          exit 0;
223      }
224      if ($arg =~ /^--email$/) {
225          print $email;
226          exit 0;
227      }
228      if ($arg =~ /^-f$/) {
229          $opts->{file} = $args[$i+1];
230          $i++;
231      }
232      if ($arg =~ /^-o$/) {
233          $opts->{output} = $args[$i+1];
234          $i++;
235      }
236      if ($arg =~ /^-s$/) {
237          $opts->{size} = $args[$i+1];
238          $i++;
239      }
240      if ($arg =~ /^-t$/) {
241          $opts->{type} = $args[$i+1];
242          $i++;
243      }
244      if ($arg =~ /^-d$/) {
245          $opts->{debug} = 1;
246      }
247      if ($arg =~ /^-q$/) {
248          $opts->{quiet} = 1;
249      }
250      if ($arg =~ /^-n$/) {
251          $opts->{no_color} = 1;
252      }
253      if ($arg =~ /^-m$/) {
254          $opts->{mode} = $args[$i+1];
255          $i++;
256      }
257      if ($arg =~ /^-c$/) {
258          $opts->{config} = $args[$i+1];
259          $i++;
260      }
251      if ($arg =~ /^-p$/) {
252          $opts->{port} = $args[$i+1];
253          $i++;
264      }
255      if ($arg =~ /^-u$/) {
266          $opts->{url} = $args[$i+1];
267          $i++;
268      }
269      if ($arg =~ /^-i$/) {
270          $opts->{interval} = $args[$i+1];
271          $i++;
272      }
273      if ($arg =~ /^-r$/) {
274          $opts->{rate} = $args[$i+1];
275          $i++;
276      }
277      if ($arg =~ /^-l$/) {
278          $opts->{limit} = $args[$i+1];
279          $i++;
280      }
281      if ($arg =~ /^-w$/) {
282          $opts->{wait} = $args[$i+1];
283          $i++;
284      }
285      if ($arg =~ /^-x$/) {
286          $opts->{xargs} = $args[$i+1];
287          $i++;
288      }
289      if ($arg =~ /^-y$/) {
290          $opts->{yes} = 1;
291      }
292      if ($arg =~ /^-z$/) {
293          $opts->{zero} = 1;
294      }
295      if ($arg =~ /^-Z$/) {
296          $opts->{zip} = 1;
297      }
298      if ($arg =~ /^-A$/) {
299          $opts->{append} = 1;
300      }
301      if ($arg =~ /^-B$/) {
302          $opts->{backup} = 1;
303      }
304      if ($arg =~ /^-C$/) {
305          $opts->{clear} = 1;
306      }
307      if ($arg =~ /^-D$/) {
308          $opts->{dry_run} = 1;
309      }
310      if ($arg =~ /^-E$/) {
311          $opts->{error} = 1;
312      }
313      if ($arg =~ /^-F$/) {
314          $opts->{force} = 1;
315      }
316      if ($arg =~ /^-G$/) {
317          $opts->{group} = $args[$i+1];
318          $i++;
319      }
320      if ($arg =~ /^-H$/) {
321          $opts->{help_text} = 1;
322      }
323      if ($arg =~ /^-I$/) {
324          $opts->{include} = $args[$i+1];
325          $i++;
326      }
327      if ($arg =~ /^-J$/) {
328          $opts->{join} = $args[$i+1];
329          $i++;
330      }
331      if ($arg =~ /^-K$/) {
33
```

Problems of crosscutting concerns

- ▶ **Concerns are buried in implementation**
 - ▶ What belongs to the concern?
 - ▶ During maintenance, have to scan the entire source code
- ▶ **Complicated collaborative development**
 - ▶ Different concerns can have different experts; all may have to work in parallel on the same code parts
- ▶ **Reduced productivity, difficult evolution**
 - ▶ When adding new code, the developer has to worry about aspects that are not directly relevant for the problem at hand (readability, understandability)

Alternative implementation (command pattern)

```
class SecureSystem extends System
    private User currentUser;
    public void login() { /* ... */ }

    public void executeOperation(Operation o) {
        if (o instanceof AuthorizeOrder)
            if (!currentUser.isAdmin())
                denyAccess();
            else
                o.execute();

        if (o instanceof StartShipping)
            if (!o.hasWriteAccess())
                denyAccess();
            else
                o.execute();
    }
}
```

- ▶ Authentication now modularized
- ▶ In turn, other concerns (specifically, the operations) no longer modular

Another attempt – method calls

- ▶ Extract authentication, logging, locking, buffer into separate methods
 - ▶ Scattering and tangling reduced to method calls
 - ▶ Clearer, but still explicit calls in code
- ▶ ➔ Many extension points in framework required; big component interfaces

```
class BusinessClass
    public void importantOperation(
        Data data, User currentUser, ...) {
        checkAuth(currentUser);
        startSynchronization();
        checkCache();
        logStart();
        // execute actual operation
        logEnd();
        endSynchronization();
    }
}
```


Another attempt - middleware

- ▶ Middleware takes care of crosscutting concerns; developer only writes actual operations (inversion of control)
 - ▶ Examples: Enterprise Java Beans provide support for distributed objects, persistence, transactions, authentication and authorization, and synchronization
 - ▶ Complex architecture
 - ▶ Middleware cannot capture all possible concerns, in particular, those concerning business logic

Tyranny of the dominant decomposition

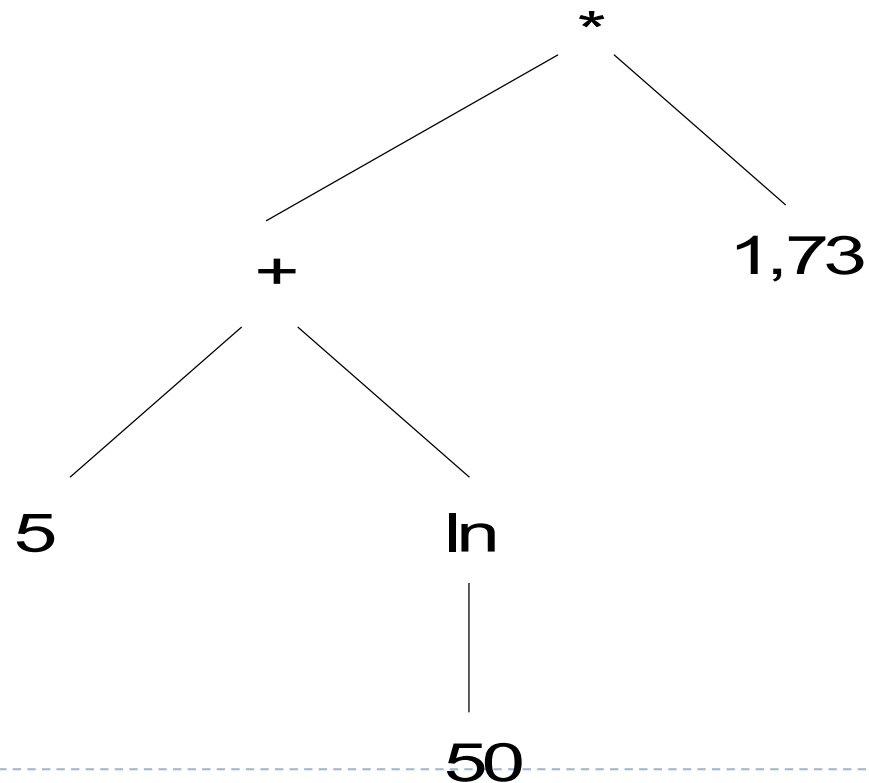
- ▶ Many concerns can be modularized, but usually, not all of them at the same time
 - ▶ Need to choose a “main dimension” of the modularization
 - ▶ Graph example: I can have colors and weights as modules...
 - ▶ ...but then the data structures (node, edge) are scattered
- ▶ Developers decide on a particular modularization (e.g. operations, authentication, data structures), but some other concerns are crosscutting
- ▶ Modularizing along several dimensions at the same time not possible

The expression problem

- ▶ A standard example for the *tyranny of dominant decomposition* problem
- ▶ Underlying question: To what extent is it possible to extract methods and data structures so that both can be extended independently...
 - ▶ without changing existing code
 - ▶ or even: without recompiling existing code
 - ▶ several times, in arbitrary order
 - ▶ without (non-trivial) code replication

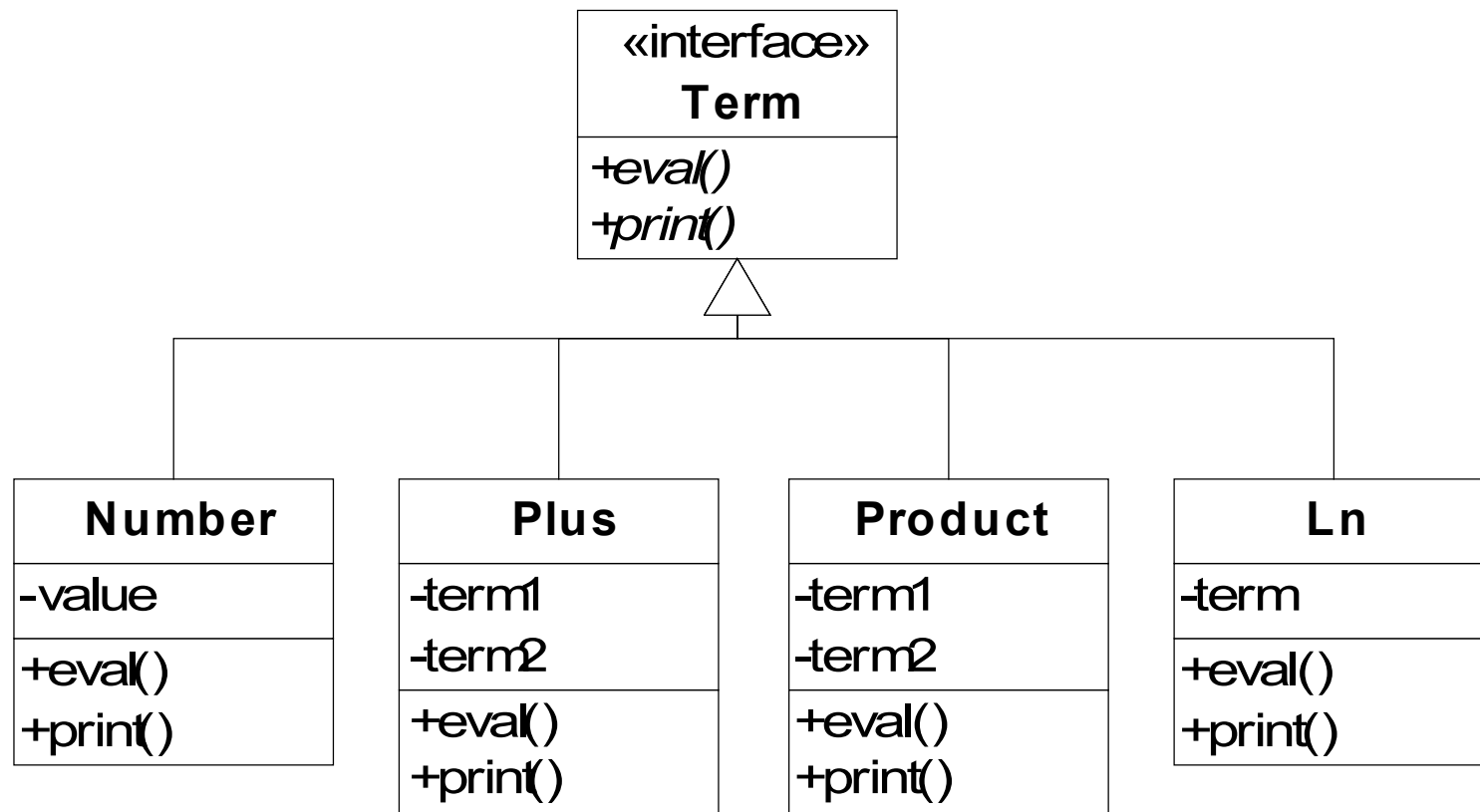
Expressions

- ▶ Task: Save arithmetic expression in trees to evaluate and print them



Implementation 1: data centric

- ▶ Recursive class structure (*composite* pattern)
- ▶ Each operation defines one method in each class

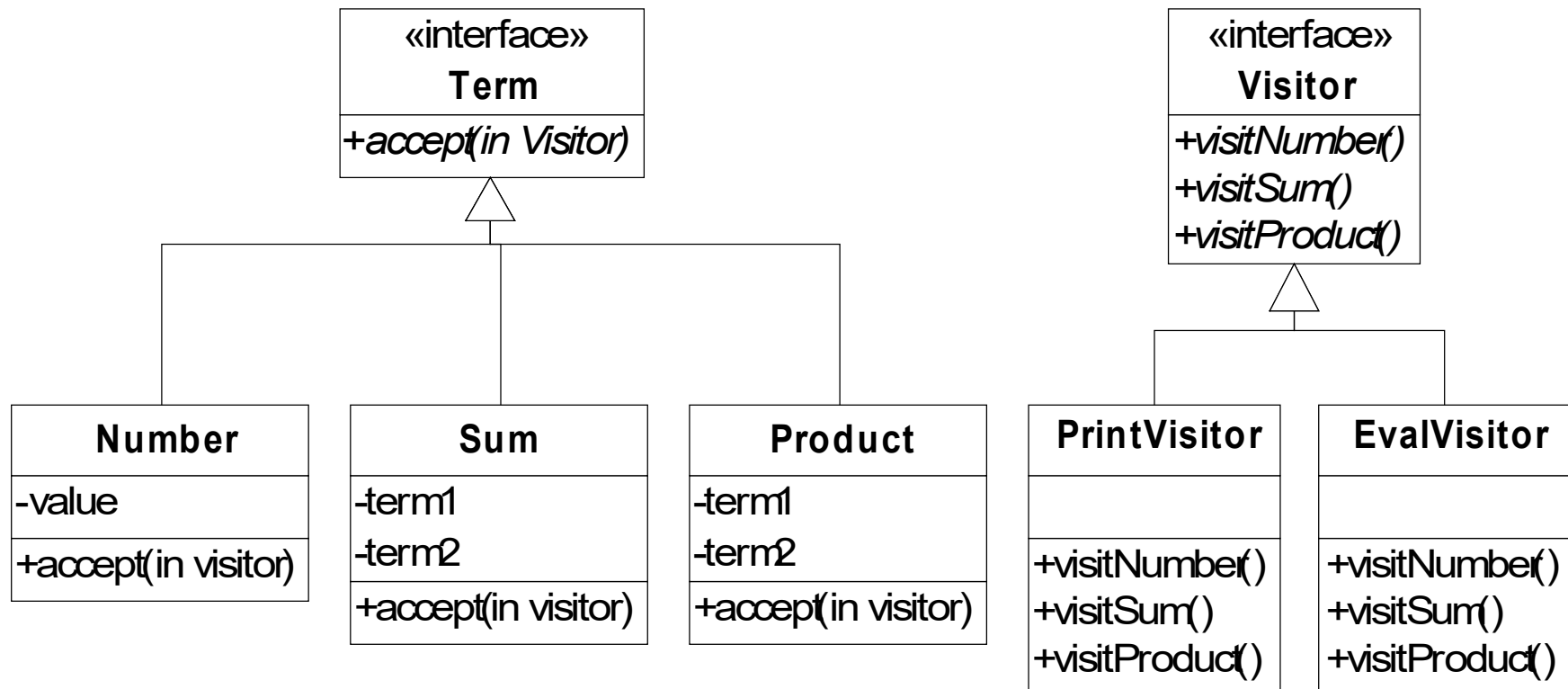


Implementation 1: problems

- ▶ Expressions are modular
- ▶ New operations (e.g. *drawTree* or *simplify*) cannot be added without significant effort
- ▶ All existing classes have to be changed!
- ▶ Operations are crosscutting to expressions

Implementation 2: method centric

- ▶ Only one method *accept* per class
- ▶ Methods are implemented with the *visitor* pattern



Code example: method centric

```
interface Term {  
    void accept(Visitor v);  
}  
class Number {  
    float value;  
    void accept(Visitor v) {  
        v.visitNumber(this);  
    }  
}  
class Sum {  
    Term term1, term2;  
    void accept(Visitor v) {  
        v.visitSum(this);  
    }  
}  
class Product {  
    Term term1, term2;  
    void accept(Visitor v) {  
        v.visitProduct(this);  
    }  
}
```

```
interface Visitor {  
    void visitNumber(Number n);  
    void visitSum(Sum s);  
    void visitProduct(Product p);  
}  
class PrintVisitor {  
    void visitNumber(Number n) {  
        System.out.print(n.value);  
    }  
    void visitSum(Sum s) {  
        System.out.print('(');  
        s.term1.accept(this);  
        System.out.print('+');  
        s.term2.accept(this);  
        System.out.print(')');  
    }  
    void visitProduct(Product p) {  
        s.term1.accept(this);  
        System.out.print('*');  
        s.term2.accept(this);  
    }  
}  
  
// Main:  
// term.accept(new PrintVisitor());
```


Implementation 2: problems

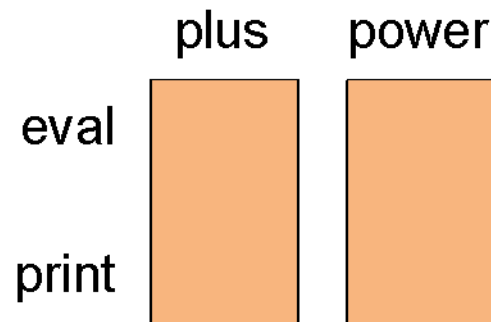
- ▶ Operations are modular
- ▶ New expression types (e.g. *min* or *power*) cannot be added without significant effort
- ▶ All existing visitor classes have to be changed!
- ▶ Expressions are crosscutting to operations

Expression problem

- ▶ Modularizing along expressions and operations at the same time almost impossible (complicated solutions with Java 1.5 generics exist)
- ▶ Data-centric approach
 - ▶ New expressions can be added directly: modular
 - ▶ New operations have to be added to all classes: not modular
- ▶ Method-centric approach
 - ▶ New operations can added as additional visitors: not modular
 - ▶ New expressions lead to changes of all visitor classes: not modular

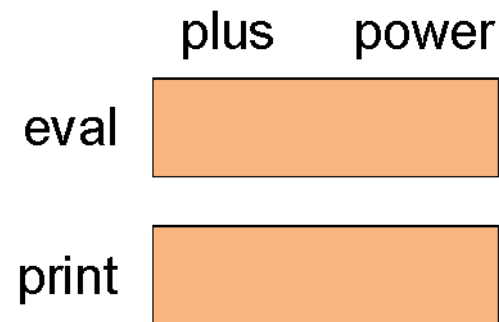
Expression Problem – graphically

Data-centric

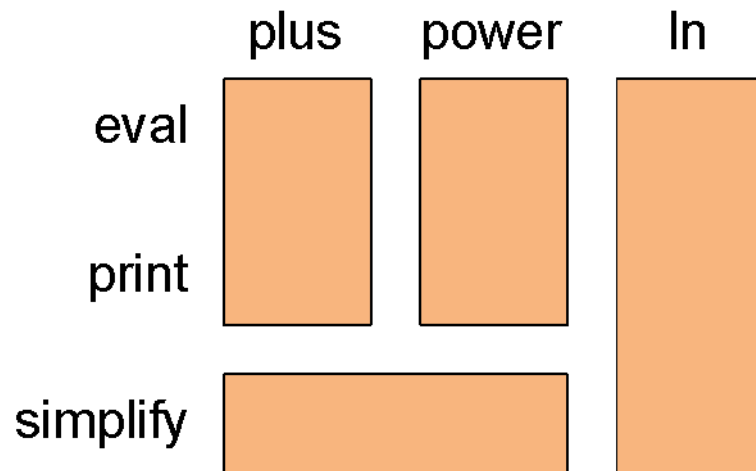


(a)

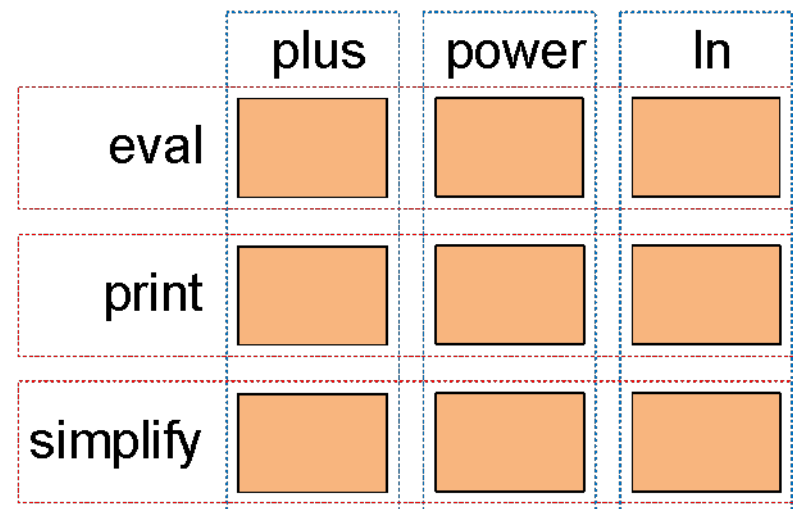
Method-centric



(b)



(c)



(d)

Typical examples for crosscutting concerns

- ▶ Logging: record each method call
- ▶ Caching/Pooling: code for each creation of an object
- ▶ Synchronization/Locking: extend many methods with lock/unlock calls
- ▶ Features in software product lines!

Dilemma

- ▶ Not always possible to modularize all concerns
- ▶ Some degree of scattered and tangled code is commonly accepted
- ▶ Some concerns are always „orthogonal“ to others: crosscutting concerns
- ▶ Often affects features of product lines

Preplanning problem

Preplanning problem

- ▶ Cannot add extensions ad hoc, but have to plan them in advance
- ▶ Need to explicitly design facilities for extension
 - ▶ Extension points in frameworks
 - ▶ Interfaces/parameters in components
- ▶ Without a suitable extension point, modular extension not possible

Preplanning problem: example

- ▶ Want to synchronize Stack methods
- ▶ Modular extension with subclass or delegation

Base code

```
class Stack { /* ... */ }
class Main {
    public static void main(
        String[] args) {
        Stack stack = new Stack();
        stack.push('foo');
        stack.push('bar');
        stack.pop();
    }
}
```

Later extension (unplanned)

```
class LockedStack extends Stack {
    private void lock() { /* ... */ }
    private void unlock() { /* ... */ }
    public void push(Object o) {
        lock();
        super.push(o);
        unlock();
    }
    public Object pop() {
        lock();
        Object result = super.pop();
        unlock();
        return result;
    }
}
```


Preplanning problem: example II

- ▶ Problem: have to change instantiation of stack in base code
 - ▶ cannot be done without changing base code (non-modular)
- ▶ Alternative
 - ▶ Design Pattern: factory instead of direct instantiation (would allow modular extension)
 - ▶ Framework with suitable extension point
- ▶ Extension points have to be anticipated (preplanning) or have to be added to the base code after-the-fact (non-modular)

Summary

- ▶ Feature modularization with components and frameworks
- ▶ No full automation, runtime overhead, coarse granularity
- ▶ Limitations related to crosscutting concerns and fine granularity
- ▶ Modularity requires planning
- ▶ Not suitable for all product lines (e.g., graph product line, embedded databases)

Outlook

- ▶ Advanced programming concepts
 - ▶ Understanding the limits of object-oriented programming
 - ▶ Feature-orientation
 - ▶ Aspect-orientation

Literature

- ▶ C. Szyperski: Component Software: Beyond Object-Oriented Programming. Addison-Wesley, 1998
[Reference book on component-oriented programming]
- ▶ R. Johnson and B. Foote, Designing reusable classes, Journal of Object-Oriented Programming, 1(2):22-35, 1988
[OOP reuse, especially frameworks]
- ▶ L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, Addison-Wesley, 2003
[architecture-driven product lines, usually frameworks]