モダン Swing

S2SwingでGUI開発はどう変わるか

浜本 階生

浜本 階生

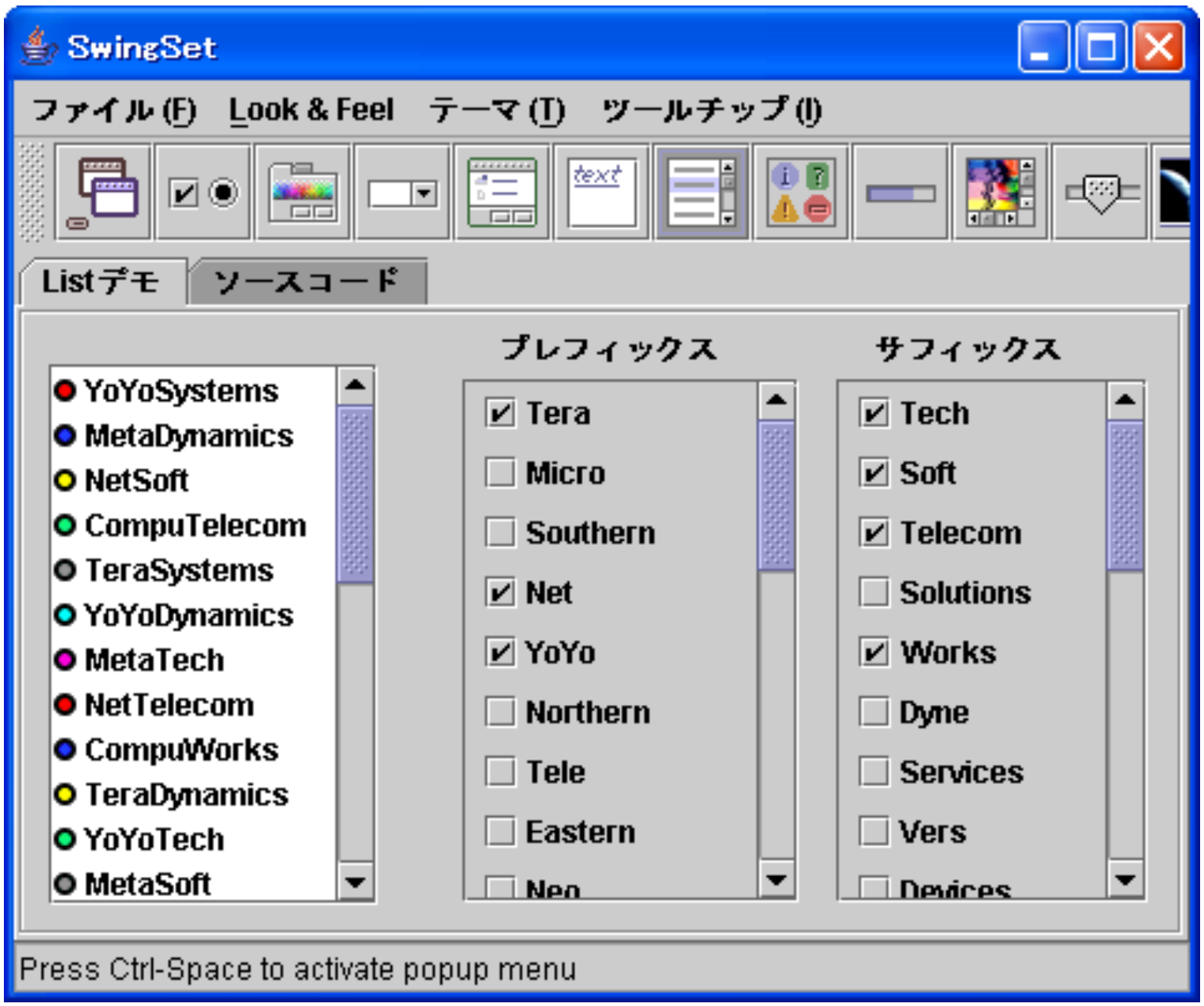
（はまもと かいせい）

http://d.hatena.ne.jp/kaiseh/

Swing

「醜い」

Metal Look & Feel



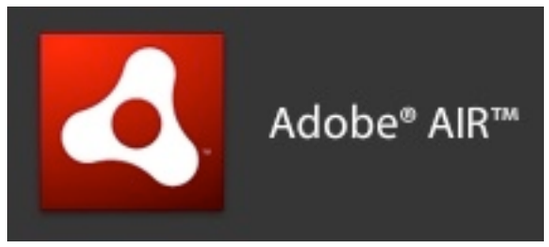
「遅い」

“If you see the Java applet loading, you click every visible link that you can to get off the page.”

http://blog.sharendipity.com/were-moving-to-flash-heres-why

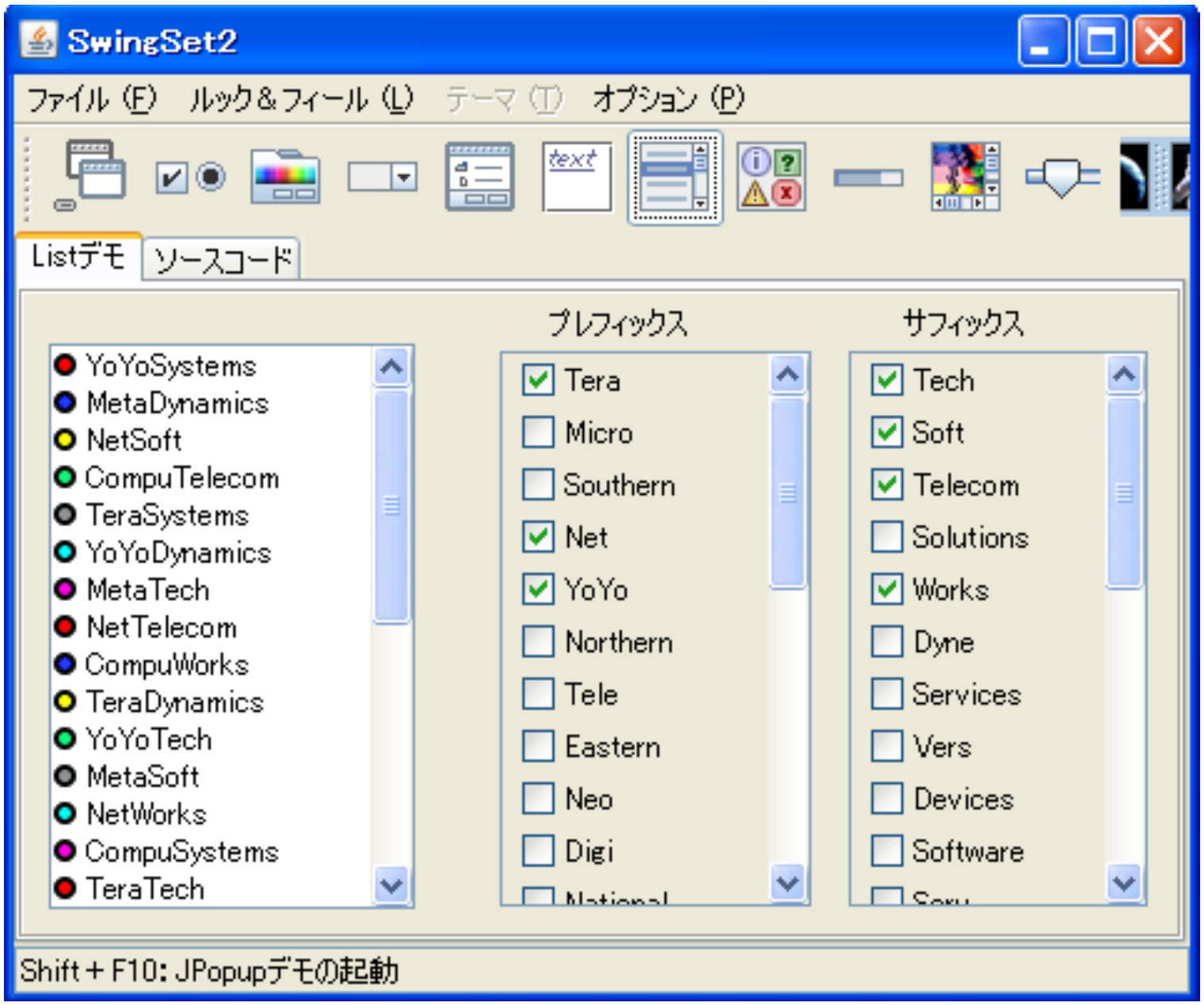
「古い」

Microsoft®
Silverlight™

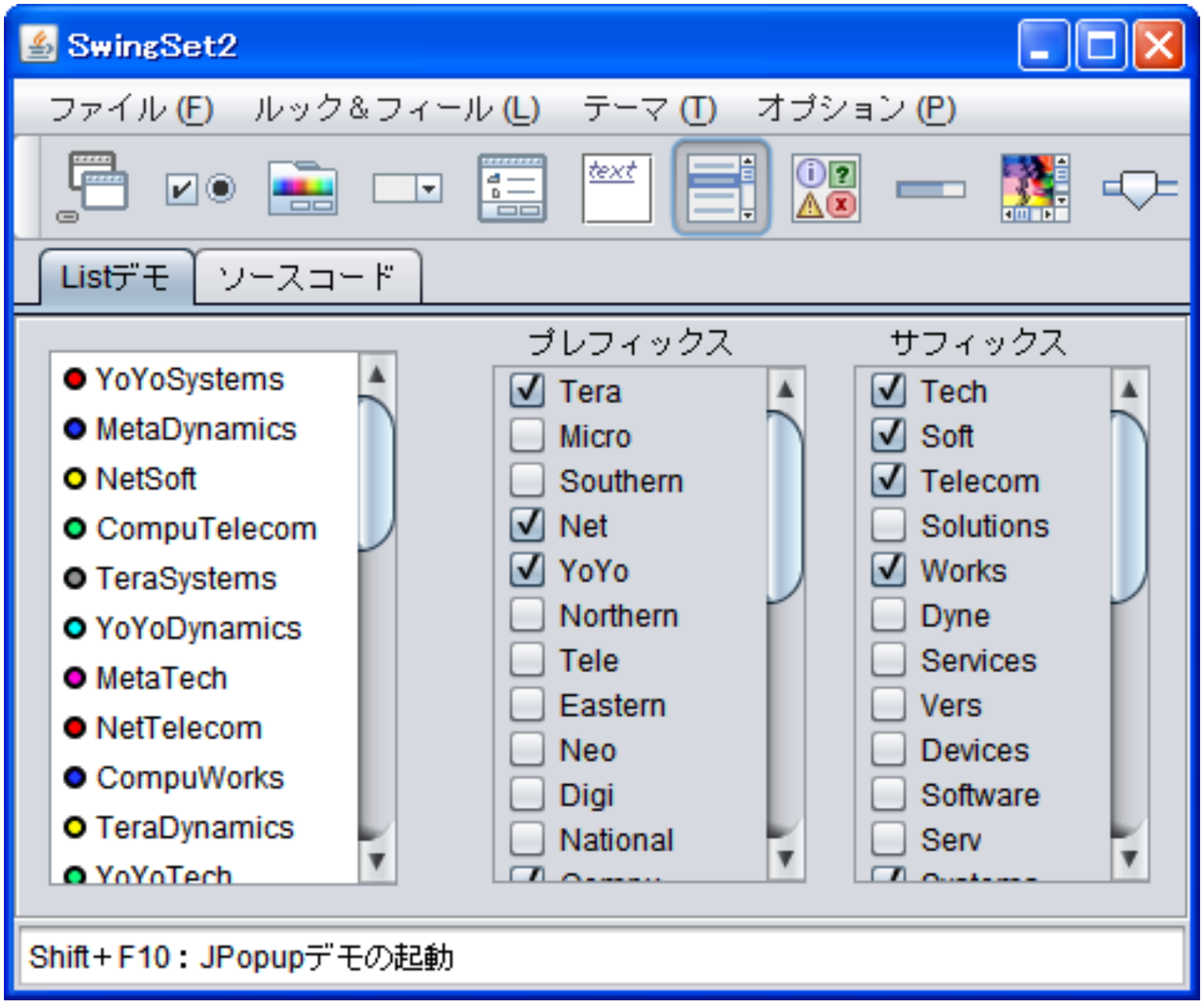


醜い？

Windows Look & Feel



Nimbus Look & Feel



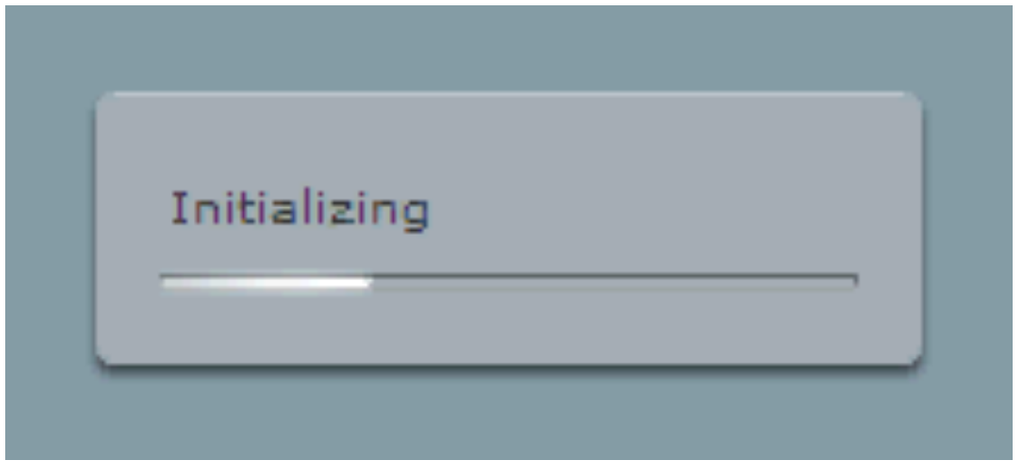
遅い？

Consumer JRE

- Java SE 6 Update 10以降
  - Java Quick Starter
  - Java2Dの高速化

裏技

```
<applet code=”MyApplet.class”>
    <param name=”image” value=”another_image.gif”/>
</applet>
```



古い？

むしろ...

- JavaFXはまだ未成熟
  - 頻繁な仕様変更
  - 貧弱なIDEサポート
- 登場が早すぎたアプレットと
  Java Web Start
  - 今こそ、Swingで復権を！

Swing Application Framework

Swing Application Framework

- Swingアプリのベストプラクティス
- JSR 296
- NetBeansでサポートあり
- Java 7に入るかも

提供する機能

- アプリケーションのライフサイクル管理
- リソースインジェクション
- 国際化
- アクション
- 非同期処理
- セッション保持（ウィンドウ位置の保存など）
- ストレージ（ファイルIOの抽象化）

最近の動向

2008/05	Spec LeadのHans Muller氏、Adobeに移籍
その後	休眠（？）
2009/03	新Spec LeadのAlexander Potochkin氏、復活宣言

Beans Binding

Beans Binding

- JavaBeanのプロパティを同期
- JSR 295
- NetBeansでサポートあり
- Java 7には入らない

データバインディング

- Webフレームワークや O/Rマッパーでおなじみ
- モデル/ロジックとビューの分離
- リッチクライアント開発でも必須！

リッチクライアントにおけるデータバインディング

- クライアントの状態は刻々と変わる
  - 状態変化に即応したユーザ体験を提供できてこその「リッチクライアント」
- プロパティの動的な監視が必要

Flex

```
<mx:TextInput id=”textInput1”/>
<mx:Text text=”{textInput1.text}”/>
```

WPF

```
<TextBox name=”TextBox1”/>
<TextBlock
   Text=”{Binding ElementName=TextBox1 Path=Text}”/>
```

Swing（Beans Binding以前）

```
JTextField textField1;
JLabel label1;
...

private void bind() {
    textField1.getDocument().addDocumentListener(
            new DocumentListener() {
        public void insertUpdate(DocumentEvent e) { sync(); }
        public void changedUpdate(DocumentEvent e) { sync(); }
        public void removeUpdate(DocumentEvent e) { sync(); }
    });
}

private void sync() {
    label1.setText(textField1.getText());
}
```

Beans Binding

```
JTextField textField1;
JLabel label1;
...
AutoBinding binding = Bindings.createAutoBinding(
    UpdateStrategy.READ_WRITE,
    textField1, BeanProperty.create("text"),
    label1, BeanProperty.create("text"));
binding.bind();
```

- 記述量は従来より減った
- でも、あまり簡潔とはいえない

S2Swing

S2Swing

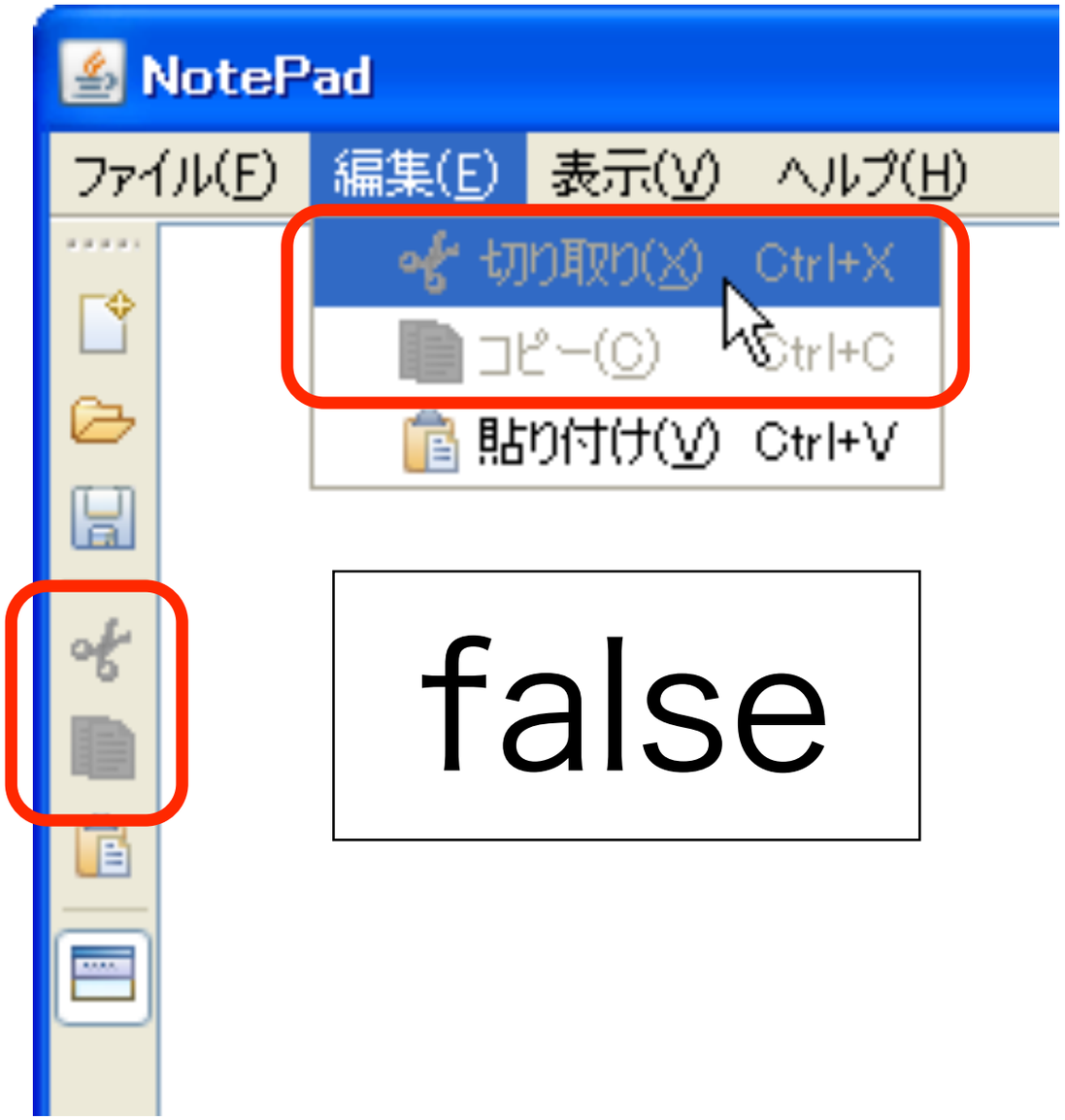
- SAFとBeans Bindingのラッパー
- より簡潔に、より便利に
- S2Containerを使用

提供する機能

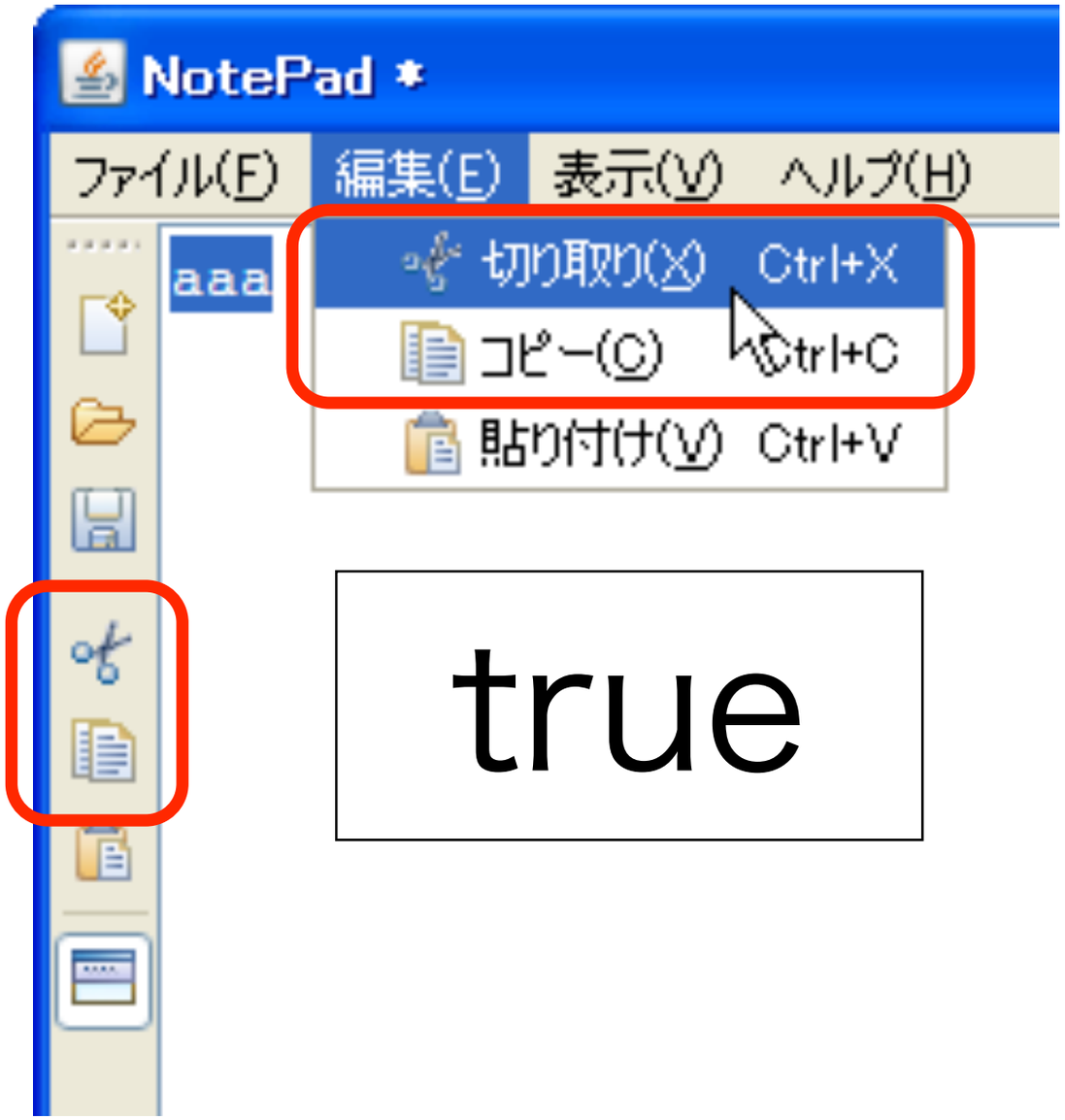
- Swing Application Framework拡張
  - **アクションの宣言的状態管理**
  - コンポーネント名の自動注入
- Beans Binding拡張
  - **バインディング記述の簡略化**
  - **PropertyChangeサポートの自動化**
  - **バリデータ、コンバータ**
- ユーティリティ
  - GUIビルダ

アクションの
宣言的状態管理

enabled状態



false



true

selected状態

false

true

Swing Application Frameworkの場合

```
private JPanel statusBar;
private boolean statusBarVisible;

public boolean isStatusBarVisible() { return statusBarVisible; }

public boolean setStatusBarVisible(boolean value) {
    boolean oldValue = statusBarVisible;
    statusBarVisible = value;
    statusBar.setVisible(value);
    firePropertyChange(“statusBarVisible”, oldValue, value);
}

@Action(selected = “statusBarVisible”)
public void toggleStatusBar() {
    setStatusBarVisible(!isStatusBarVisible());
}
```

- プロパティ名を指定してenabled状態、selected状態を管理（PropertyChangeのサポート必須）

S2Swingのアプローチ

```
private JPanel statusBar;

@S2Action(selected = “statusBar.visible”)
public void toggleStatusBar() {
    statusBar.setVisible(!statusBar.isVisible());
}
```

- OGNL式で宣言的に状態管理可能
- 極めて簡潔！
- 定常的に式を評価して監視している

バインディング記述の
簡略化

Beans Binding（再掲）

```
JTextField textField1;
JLabel label1;
...
AutoBinding binding = Bindings.createAutoBinding(
    UpdateStrategy.READ_WRITE,
    textField1, BeanProperty.create("text"),
    label1, BeanProperty.create("text"));
binding.bind();
```

S2Swingのアプローチ

```
JTextField textField1;
JLabel label1;
...
Binder binder = new Binder();
binder.add(textField1, “text”, label1, “text”);
binder.bind();
```

Binderクラス

- Beans Bindingにおける表現の違いを吸収
  - プロパティの種類差
    - ObjectProperty （Bean自身）
    - BeanProperty（ex. “foo.bar”）
    - ELProperty（ex. “${baz > 0}”）
  - バインディングの種類差
    - JComboBoxBinding
    - JListBinding
    - JTableBinding

PropertyChange
サポートの自動化

双方向バインド不可

```
public class Person {
    private String name;

    public String getName() { return name; }

    public void setName(String name) {
        this.name = name;
    }
}
```

- Swingコンポーネントのプロパティ変更をPersonオブジェクトに反映することは可能
- しかし、Personオブジェクトのプロパティ変更を検出することは不可能（単なるPOJOなので）

双方向バインド対応方法

```
public class Person {
    private PropertyChangeSupport pcs =
            new PropertyChangeSupport(this);
    private String name;

    public String getName() { return name; }

    public void setName(String name) {
        String oldValue = this.name;
        this.name = name;
        pcs.firePropertyChange(“name”, oldValue, name);
    }

    public void addPropertyChangeListener(PropertyChangeListener l) {
        pcs.addPropertyChangeListener(l);
    }
}
```

S2Swingのアプローチ

```
Person person =
    ObservableBeans.create(Person.class);
```

- PropertyChangeサポートをAOPで自動的に埋め込み
- POJOの双方向バインドに簡単対応

バリデータ、コンバータ

バリデータ、コンバータ

- データバインディングには入力値検証と変換がつきもの
- 定義の簡潔性とカスタマイズの柔軟性が非常に重要
- Beans Bindingは原始的サポートしか提供していない

S2Swingのアプローチ

```
public class Person {
    @Required
    @TrimConverter
    private String name;

    public String getName() { return name; }

    public void setName(String name) {
        this.name = name;
    }
}
```

- アノテーションをもとに、Beans Binding形式のバリデータ、コンバータを生成

エラーメッセージの
カスタマイズ

Person.properties （クラスと同名のプロパティファイル）

```
name.label = 名前
name.Required.failed = 必ず入力してください。
```

▼

“名前：必ず入力してください。”

デモ

S2Swing

http://s2swing.sandbox.seasar.org/