Eclipse 4 Programming Model and Practices

Jin Mingjian

Agenda

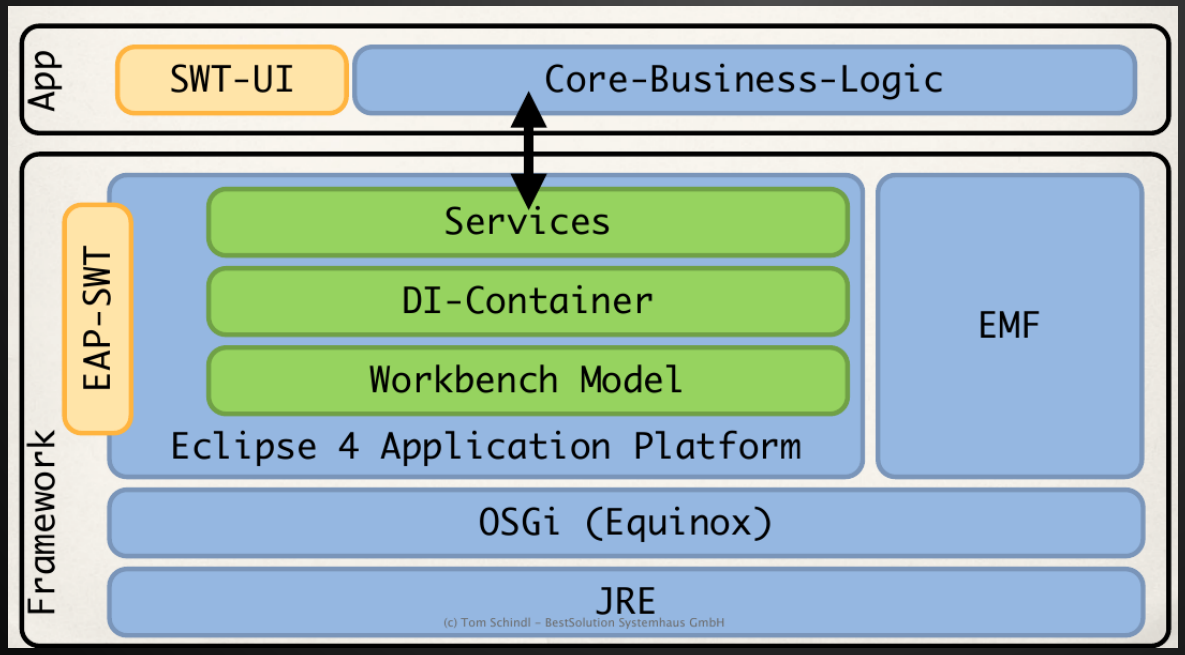
- Eclipse 4 and Kepler
- Programming Model
- Good Practices
- Random Thoughts

Eclipse 4

- **Modeled UI**
  - **modeling(MDD): EMF**
- **Dependency Injection**
  - **home made engine(? Sisu)**
- **CSS Styling**
  - **CSS 2.1**
- **Services Reload**
  - ECommandService, EPartService, EModelService, ESelectionService, EContextService, Logger, IEventBroker, IPresentationEngine
- **Web**
  - **Orion**

Eclipse 4 (Unchanged)

- OSGI Bundle
- other OSGI goods(like, Declarative Services)
- Eclipse Extension/Extension Points



by Tom Schindl, http://tomsondev.bestsolution.at

Kepler (new)

- Platform
  - performance fix
  - initial Eclipse 4 API released
  - UI improvement
- JDT
  - enhancements to content assist
  - enhancements to null annotation, leak analysis
- Tooling and releng
  - PDE: launch config, plugin image browser
  - CBI

Programming Model

- "How to interact with the programmed system by its programmer"
- Old(Eclipse 3.x)
  - Kinds of APIs with numerous parameters
    - include interfaces/classes to be inherited
  - patterns: factory, fluent interface
  - APIs force the app flow and introduce strong coupling
- Why new(Eclipse 4.x)?
  - A simplified model will largely reduce the development and maintenance costs

Programming Model

- Modeled UI
  - MVC[1]: separates business from presentation
  - renderer agnostic - workbench rendered by other renderer besides SWT
  - static model - e4xmi
  - dynamic model - DOM in browser
  - built-in models for UI
    - MWindow, MToolbar, MPartSashContainer, MPart, MUILabel, MContribution, MContext
  - tooling to define the "application model"

[1] : http://en.wikipedia.org/wiki/Model–view–controller

Programming Model

- Modeled UI
  - one sample application model to UI

by Lars, http://www.vogella.com/articles/EclipseRCP/article.html

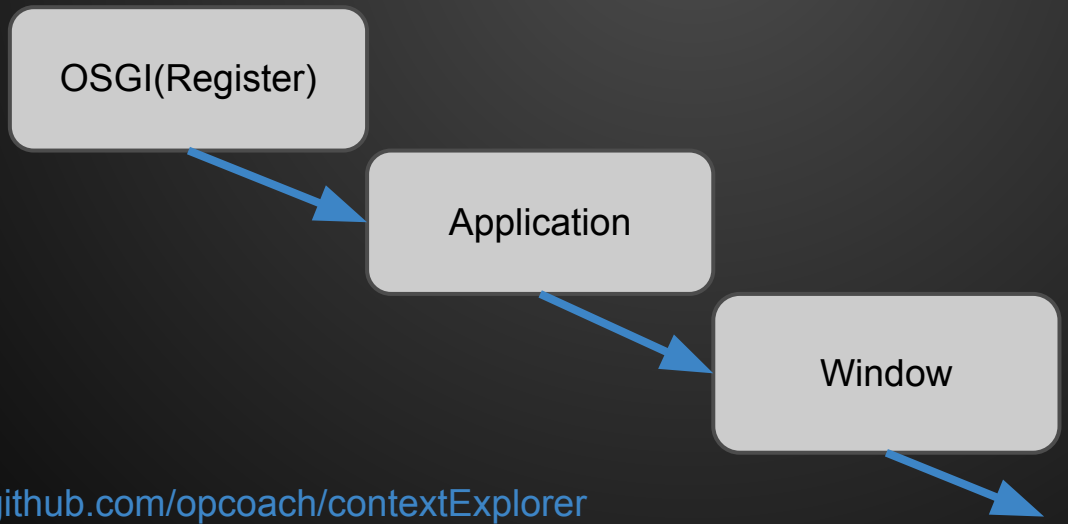
Programming Model

- **Dependency Injection**
  - thrives with Google Guice
  - **IoC** - managed
  - **Decoupling** - DBC favored
  - **JSR 330** - annotations
    - @Inject
    - @Named...
  - **Eclipse specific** - less portable
    - @CanExecute, @Execute
    - @Preference
    - @EventTopic, @UIEventTopic
    - @Optional, @AboutToShow...

http://wiki.eclipse.org/Eclipse4/RCP/Dependency_Injection

Programming Model

- **Dependency Injection**
  - Internal to Eclipse homebrew DI
    - IEclipseContext - hierarchical
      - k/v, set/get
      - convention for global : class name -> instance
    - MContext - workbench model (@Active)



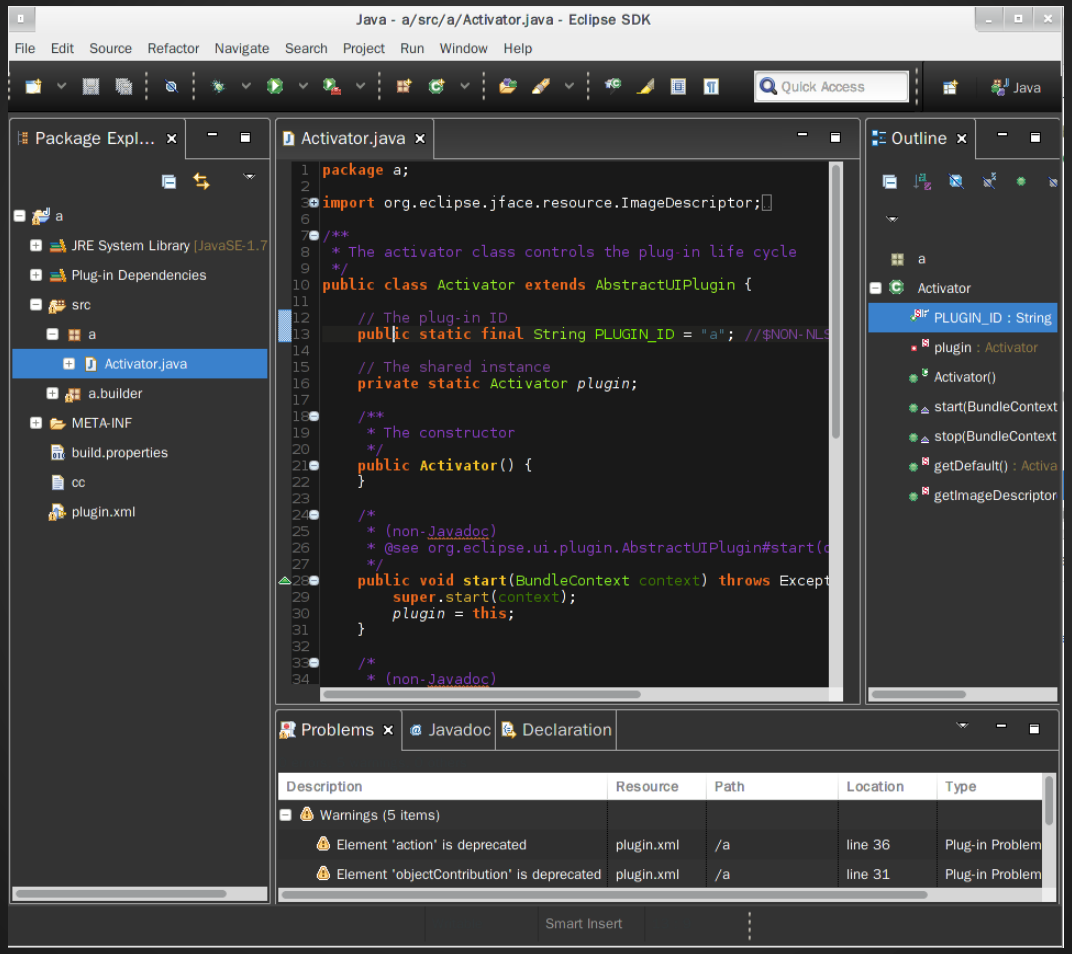
https://github.com/opcoach/contextExplorer

Programming Model

- **Dependency Injection**
  - ContextInjectionFactory#make
    - injection by manual when needed
  - Example: Darker theme[1]

[1] https://github.com/jinmingjian/eclipse.themes.darker

Programming Model



https://github.com/jinmingjian/eclipse.themes.darker

Good Practices

- "Consume the services by dependency injection way as possible."
  - "Service is verbose to use it correctly"
  - DI way cuts down boilerplates and makes your life easier
  - Example:

```
124    @Inject
125    private void setPrefReferences(
126        @Preference(nodePath = Activator.PLUGIN_ID) IEclipsePreferences prefDarker,
127        @Preference(nodePath = "org.eclipse.ui.editors") IEclipsePreferences prefUI,
128        @Preference(nodePath = "org.eclipse.jdt.ui") IEclipsePreferences prefJDT,
129        @Preference(nodePath = "org.eclipse.pde.ui") IEclipsePreferences prefPDE) {
130      this.prefUI = prefUI;
131      this.prefJDT = prefJDT;
132      this.prefDarker = prefDarker;
133      this.prefPDE = prefPDE;
134    }
```

Good Practices

- "Consume platform services as possible."
  - Overhaul for Eclipse platform
  - Example: Event Service
    - a mechanism for hooking/intercepting into the whole platform

```
@Inject
IEventBroker eventBroker;

@Execute
public void onExecute() {
  eventBroker.subscribe(IThemeEngine.Events.THEME_CHANGED,
    new EventHandler() {
      public void handleEvent(Event event) {
        ITheme currentTheme = (ITheme) event.getProperty(IThemeEngine.Events.THEME);
```

Good Practices

- "Injection to your objects in a full managed way is simple."
  - Injection **not** come automatically
  - Sometimes you may miss APIs
  - Bind to IEclipseContext
    - direct control
    - RCP

Good Practices

- Model Processor Extension
  - *"A model processor is a normal injectable POJO class whose processing is triggered by an @Execute annotated method."*
  - Example:
  - The @inject-ed instances will be injected at a sometime

```
31    <extension
32          id="id1"
33          point="org.eclipse.e4.workbench.model">
34       <processor
35             beforefragment="true"
36             class="eclipse.themes.darker.ui.DarkerThemer">
37       </processor>
38    </extension>
```

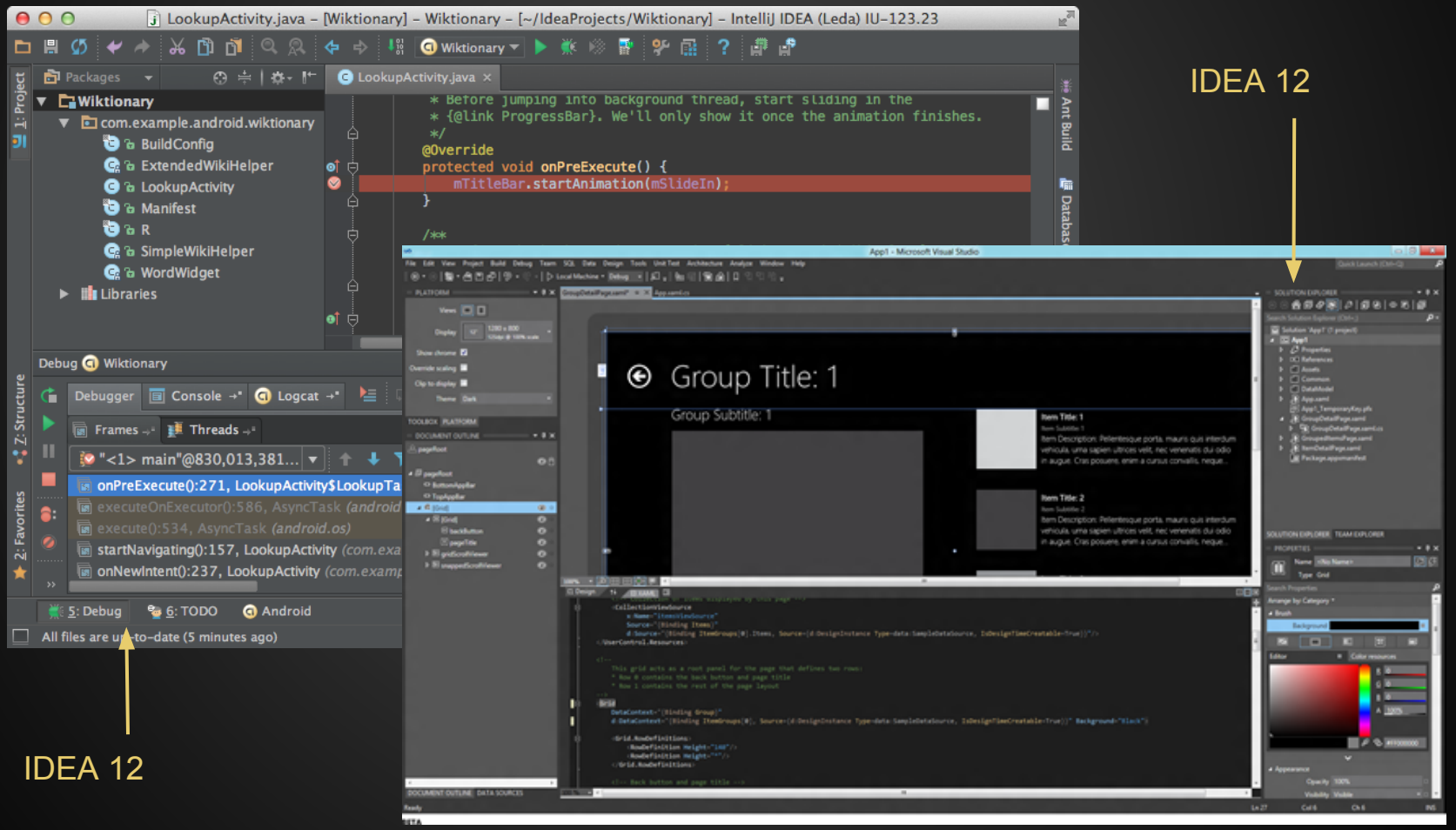
- "Do not assume that @inject-ed objects could be injected always(rightly)."

Good Practices

- "Eclipse 4 Tooling is your friend."
  - new introduced techniques paid for newcomers
  - CSS spy
  - CSS editor
  - Application model editor
  - Live editor
    - dynamically model workbench UI here

http://git.eclipse.org/c/e4/org.eclipse.e4.tools.git/

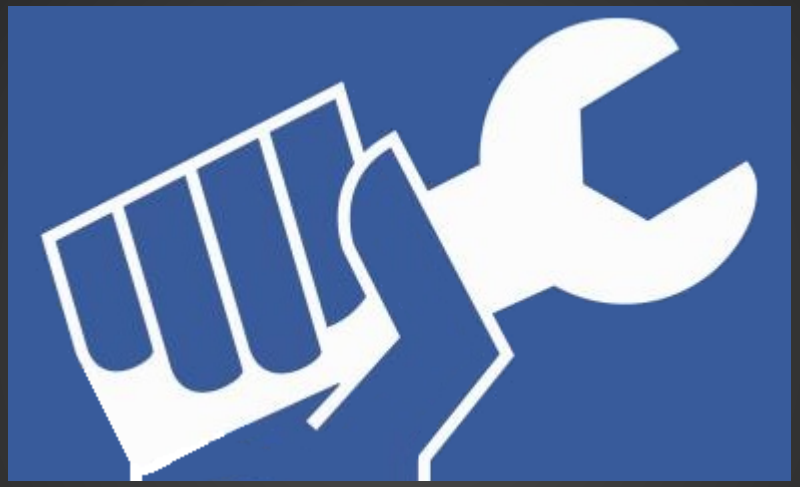
Random Thoughts



Random Thoughts

- Future of Workbench theme(Darker)
  - SWT standalones or RCPs are fine
  - many widgets of IDE Workbench do not support to being styled in all platforms
    - Text, Button, Label...
    - rewrite the workbench UI to use other styleable widgets
  - port SWT API to top of other flexible ui tech
    - on Swing(dead...)
    - on OpenGL(excited!)
      - Bling(l33t labs, EclipsCon13)
    - on JavaFx(dormant)

Thank you!



Eclipse Hackathon