Eclipse: Principles, Practices and Patterns

The philosophy behind Eclipse

Marco Savard

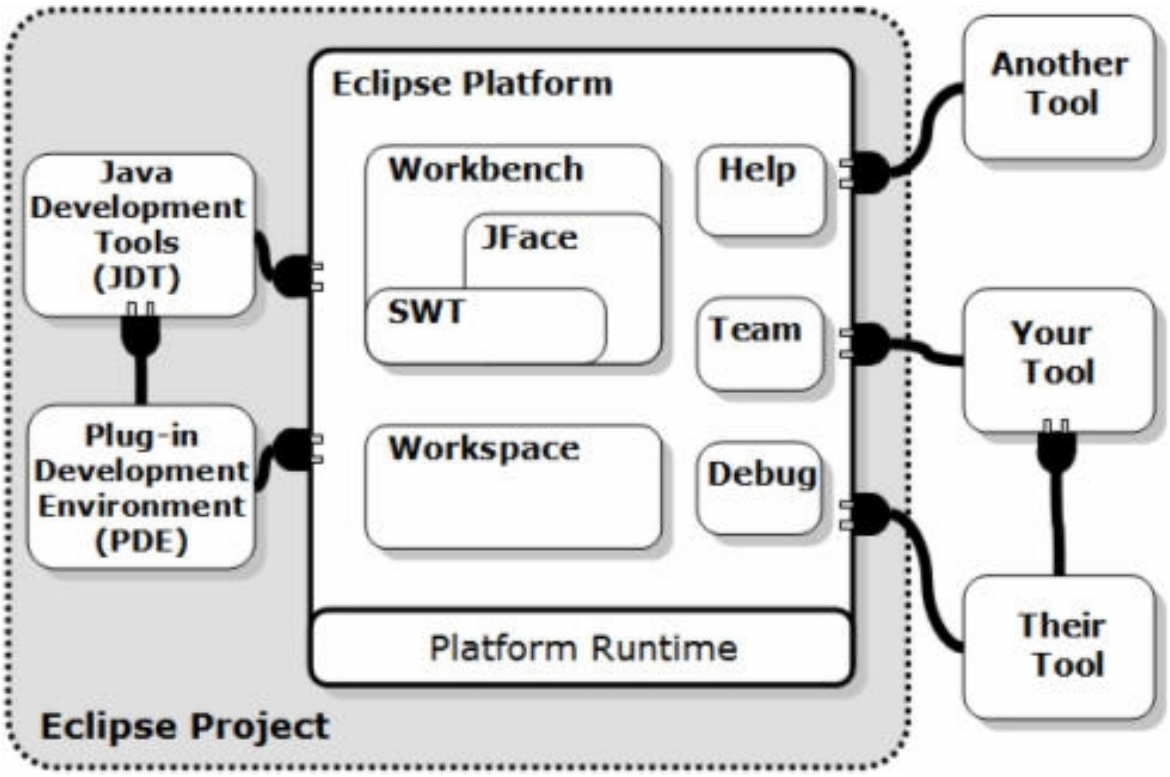
Behind Eclipse

- Erich Gamma: the father of Design Patterns and JUnit
- Kent Beck: the father of XP
- Several others

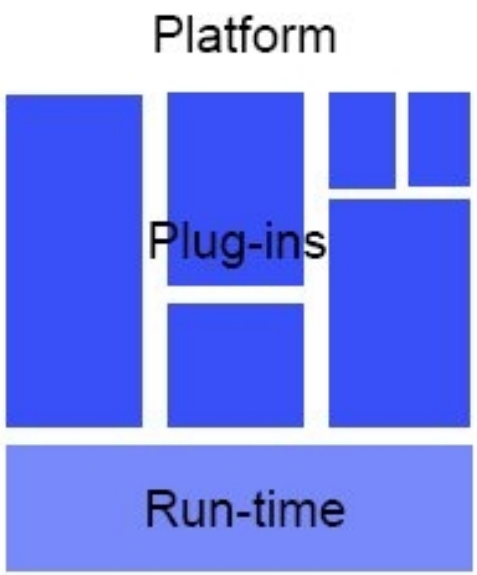
Principle #1

- Everything is a contribution
- Extension and Extension Points
- Eclipse: run-time kernel + contributing plug-ins

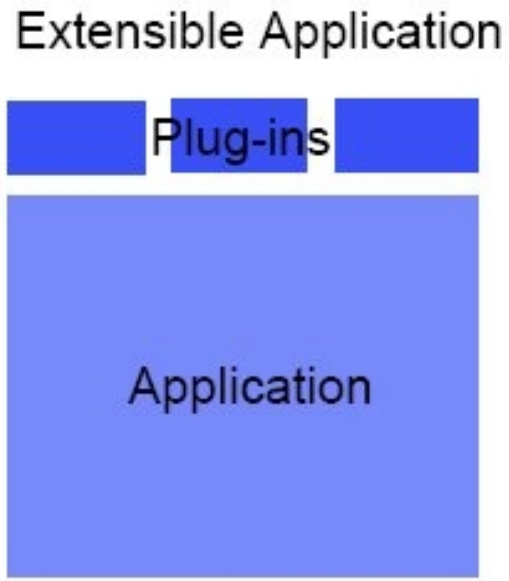
Eclipse Architecture



Eclipse Architecture

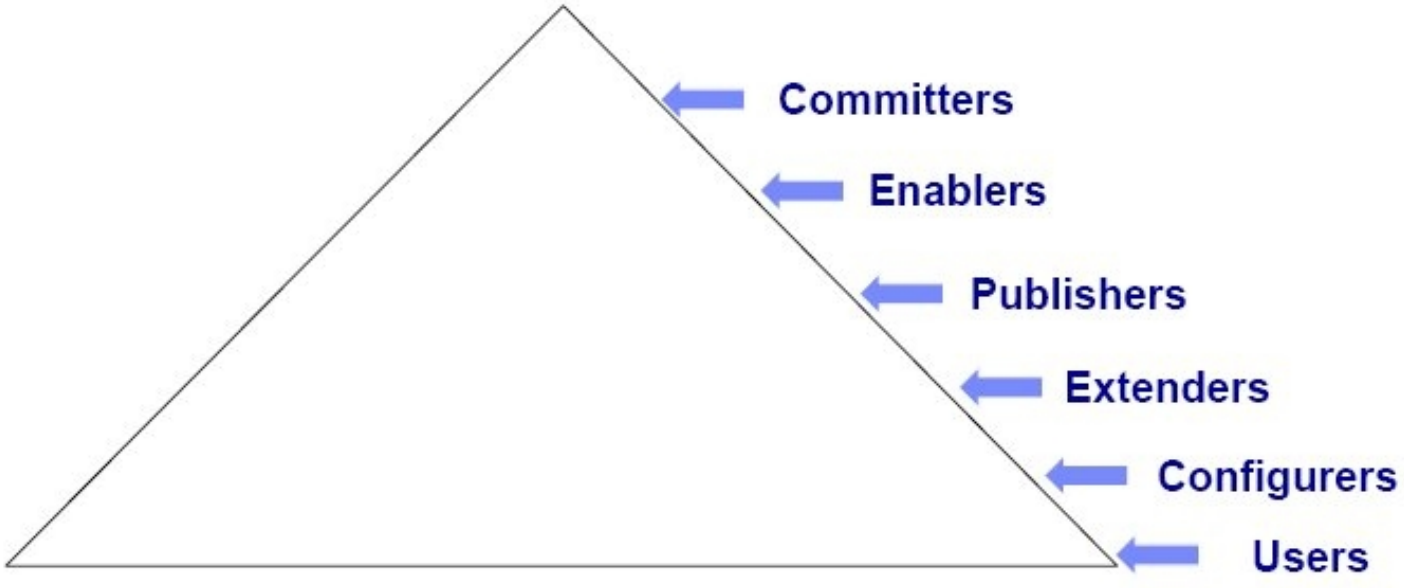


Eclipse



Other IDEs

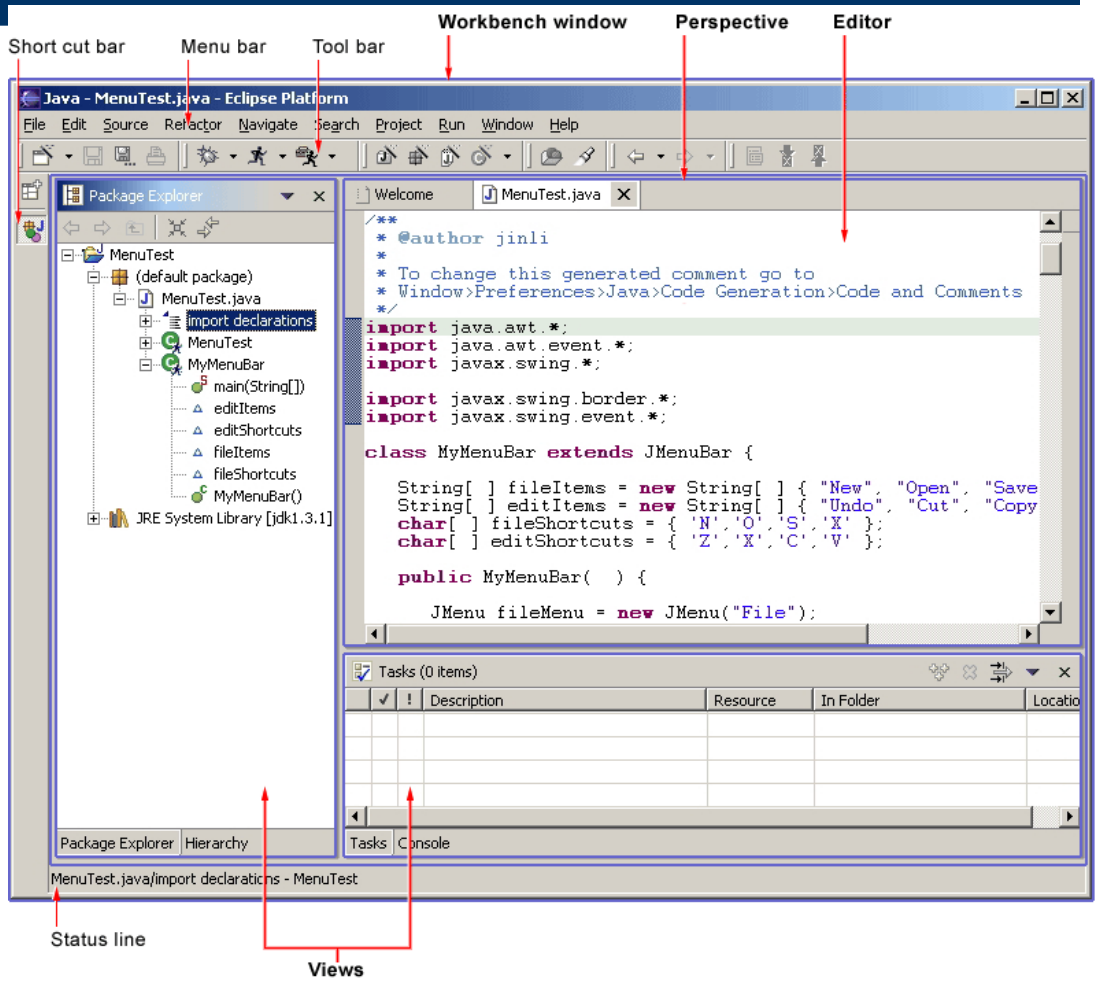
The Contribution Pyramid



Kinds of Contributions

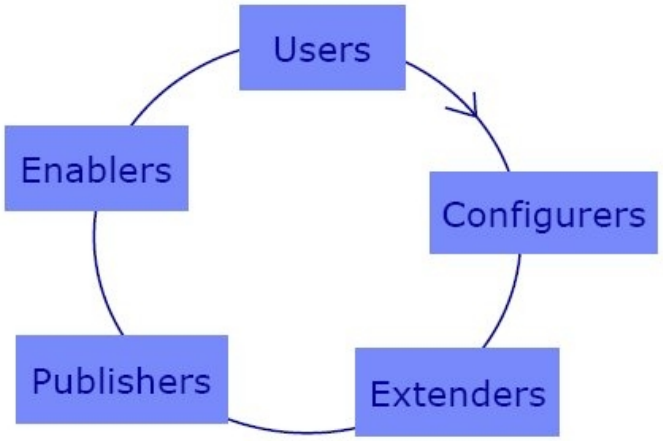
- A menu item
- A tool
- A preference page
- A help chapter
- An editor
- A view
- A perspective

Workbench Elements



Principle #2

- Plug-in must be compliant to the plug-in interface
- The Contribution Cycle:



Principle #3

- «Separate Declaration and Implementation»
- Manifest: contributed class, icon, item name.
- Lazy loading.
- The iceberg model

Principle #3

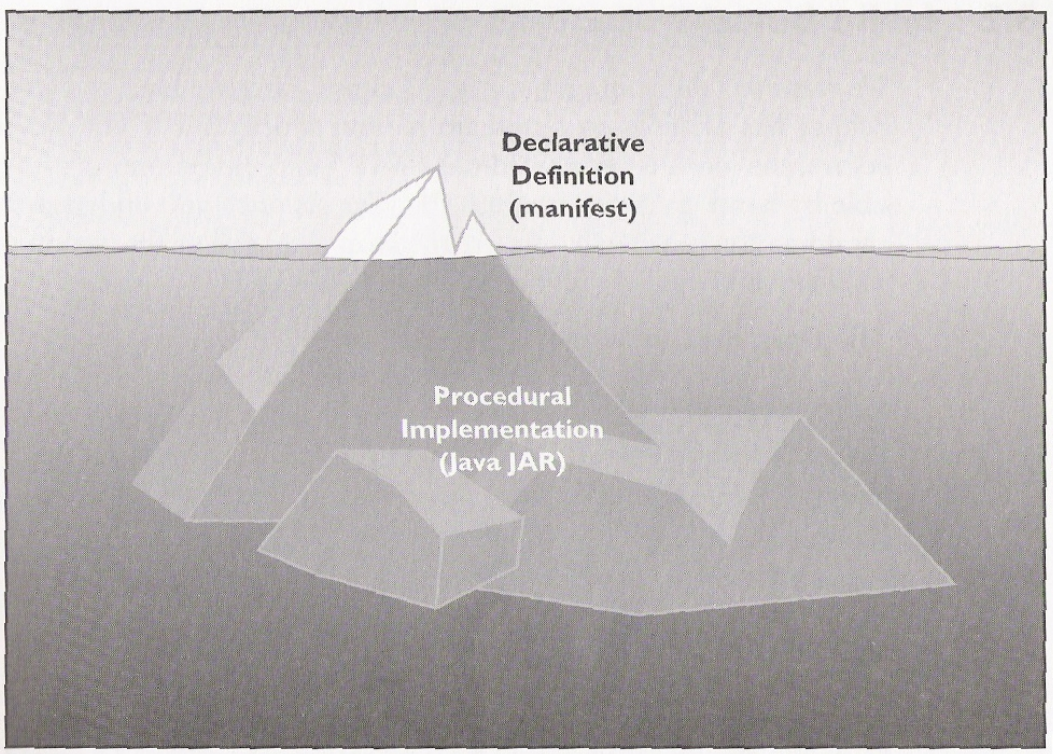


Figure 3.2 *The Manifest Describes Your Plug-In's Contribution*

Anatomy of a plugin.xml

```
<plugin>
 <extension point="org.eclipse.ui.editors">
  <editor
     class="ca.gc.drdc.oasis.sp.stat.editors..”
     contributorClass="ca.gc.drdc.oasis..”
     extensions="java“
     icon="icons/palm16.gif“
     id="ca.gc.drdc.oasis.s”
  />..
</plugin>
```

Publishing a plug-in

- Plug-in: a standalone Java project
- Feature: coherent groups of plug-ins
- Update Site: for publishing
- PDE: Plug-in Development Environment

#4: Good Fences Principle

- Protect yourself (w/ a try/catch) when your code is invoking another plug-in
- Your exceptions (including RuntimeException) must be caught before crashing Eclipse.
- Do not call window.dispose() or System.exit()!

#5: Responsibility Principle

- Assume your errors!
  - Do not hide
  - Do not make the caller plug-in responsible for your mistakes

#5: Contract Principle

- Never change your public API
- Classes in ‘internal’ may change
- No ‘internal’ in import clauses

Part 2: Practices

Best Practices of Eclipse

Coding Conventions

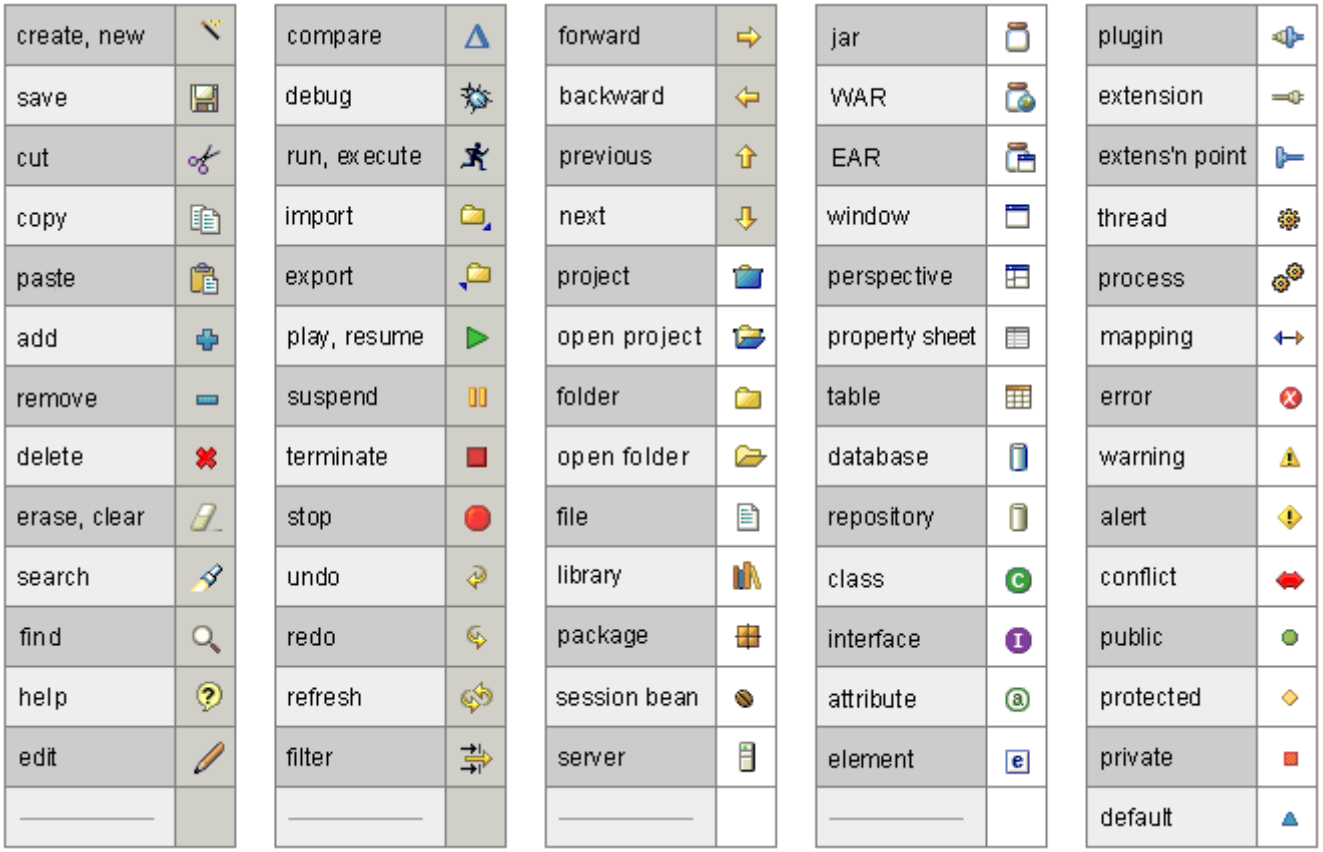
- Mostly compliant with Sun's Code Conventions for the Java Programming Language covers filenames.
- Deviate something from Sun's practices:
  - Packages: internal, (Non-API) tests, examples. Only lowercase.
  - Classes: TextXXX, I prefix for interface
  - Methods: Java Bean Conventions for getters, setters, predicates.
  - Variables: Short name for short range.

- Ref: http://wiki.eclipse.org/index.php/Coding_Conventions

UI Guidelines

- G1.5 and G1.6: Headline and Sentence style capitalization
- G1.8: Model dialog to handle errors requiring user's action
- G2.2 Use the 256 official colors, and the eight greyscale palette.
- G2.8 Widget alignment (7 pixels between icons, etc.)
- G3.1 Each command must have a label, a tool tip and a color icon.
- G6.20 New contributing views must cooperate with existing views (Outline, Search, Error, Progress, etc.)
- Ref: http://wiki.eclipse.org/index.php/User_Interface_Guidelines

UI Guidelines: reuse standard icons



Practice #1: Project names are unique

- Org.eclipse.core…
- Project names = package name

Practice #2: Separate Core and UI

- Org:eclipse.*.common
  - Utility classes, reusability
- Org.eclipse.*.core
  - Headless, Impl
- Org.eclipse.ui
  - actions, editors, views, wizards

Practice #3: Hide the internals

- Org.eclipse.*.core
  - Org.eclipse.*.core.impl
  - Org.eclipse.*.core.internal
  - Org.eclipse.*.core.tests
  - Org.eclipse.*.core.examples
  - Org.eclipse.*.core.<services>

Part 3: Patterns

Design Patterns
Behind Eclipse

The Simplest Pattern

- Singleton : one instance
- Often Used in conjunction w/ Factory

Singleton: formal definition

- If class C a singleton, then
  - -C() for all constructors
  - -$ C singleItem, lazy initiated
  - +$ C getInstance(): the only one +$ method:
    - 1st call: create singleItem
    - Modifiers: public, static, synchronize
  - And several public methods (not static)

Classes of Patterns

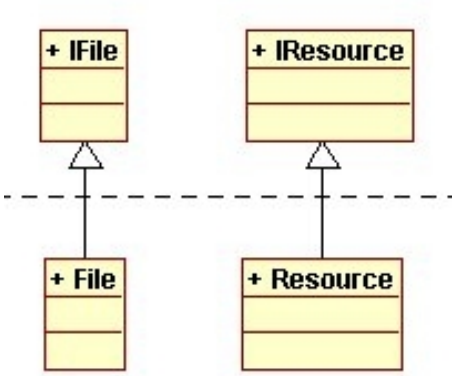
- Creational : constructs objects
- Structural : object permanent structure
- Behavioral : method contents

Creational Patterns

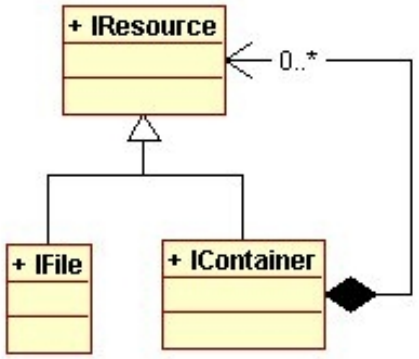
- Factory Method: encapsulate of constructor
  - Object createObject()
  - Pro: allows redefinition
- Factory Class: a class w/ factory methods
- Abstract Factory : different implementations

Structural Patterns

- Bridge



- Composite



- Adapter: TreeViewer
- Façade: JavaCore
- Proxy: Iresource
- IAdaptable

Behavioral Patterns

- Command: IAction
- Memento: IMemento
- Observer: IResourceChangedListener
- Strategy: Layout
- Visitor: ASTVisitor