Frameworks

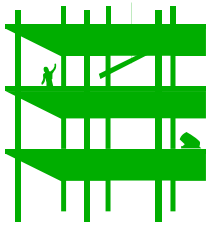
- Aims of this section:
  - Look at the notion of frameworks
  - Explore two frameworks supporting tool development
    - Eclipse
    - Argo
- Later
  - Look at Pattern Languages
    - collections of patterns that used together lead to solutions for a particular domain area
  - Illustrate with a pattern language for developing frameworks together with its use in the evolution of MViews/JViews for software tool construction

Frameworks

- "A framework is a set of classes that embodies an abstract design for solutions to a family of related problems"
  - Ralph Johnson, "Designing Reusable Classes", The Journal of Object-Oriented Programming, Vol.1,No.2, 1988, pp 22-35
- "A software framework is a reusable mini-architecture that provides the generic structure and behavior for a family of software abstractions, along with a context of memes/metaphors which specifies their collaboration and use within a given domain."
  - Brad Appleton "Patterns and Software: Essential Concepts and Terminology"
- Provide a prefrabricated structure or template for applications in a particular domain
  - eg an application framework provides the support for "default" behaviour for drawing windows, scollbars and menus
  - "Leveraging Object-Oriented Frameworks" Taligent white paper http://www.ibm.com/java/education/ooleveraging/index.html

Examples of frameworks

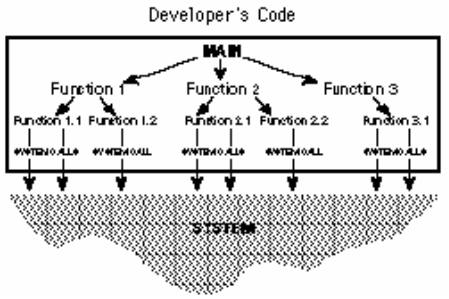
- Many of the Java APIs are frameworks for developing applications or applets for a particular domain
  - eg AWT, Swing for GUI applications
- Many IDEs provide application development frameworks
  - eg Eclipse, Argo UML
- Some widely successful and influential frameworks include:
  - ObjectTime
  - Unidraw/HotDraw
  - ET++
  - MVC
  - MacApp



Framework vs procedural and OOP

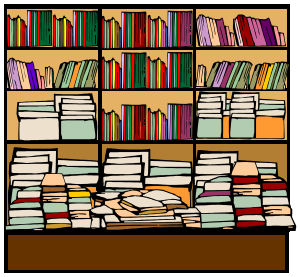
- Procedural
  - Developers code calls the "system" code via library calls
  - Developer responsible for overall behaviour and flow of control
  - system code provides underlying functionality
- Problems
  - difficult to extend "system"
  - difficult to factor common code

Figure 1



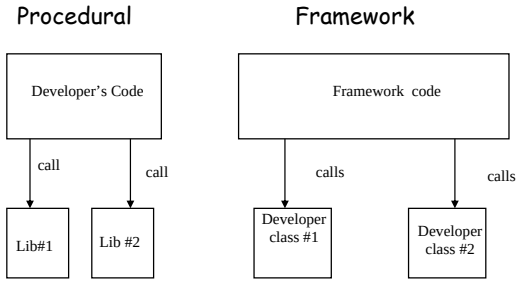
OOP and class libraries

- An improvement in terms of factoring out common code and improving maintainability
- But developer still responsible for the main program flow
  - client instantiates classes from class library
  - client calls functions
  - little predefined flow of control or interaction
  - little default behaviour



Framework oriented programming

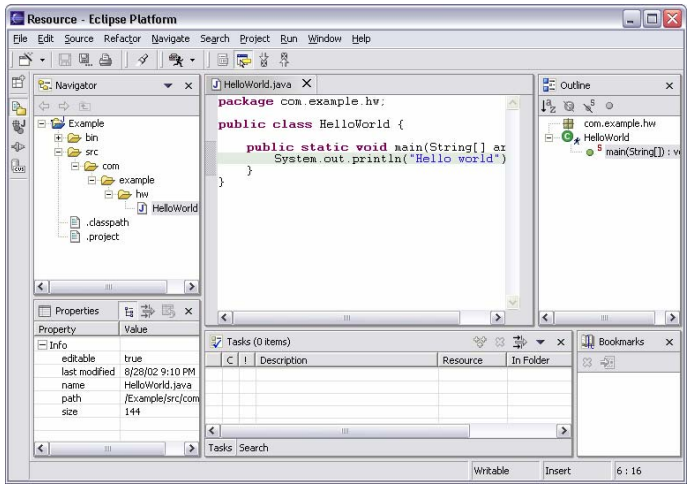
- Frameworks provide infrastructure and design
  - basic flow of control and internal structure "wired" in
- The framework calls the developers code (Hollywood principle)
  - roles reversed compared with procedural programming
  - Eg Applets in Java



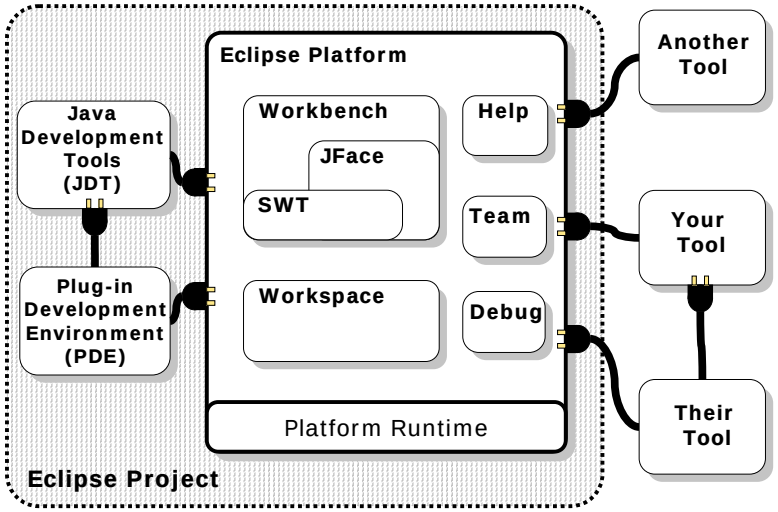
Eclipse

- Project Aims:
  - Provide open platform for application development tools
    - Run on a wide range of operating systems
    - GUI and non-GUI
  - Language-neutral
    - Permit unrestricted content types
    - HTML, Java, C, JSP, EJB, XML, GIF, …
  - Facilitate seamless tool integration
    - At UI and deeper
    - Add new tools to existing installed products
  - Attract community of tool developers
    - Including independent software vendors (ISVs)
    - Capitalize on popularity of Java for writing tools
- Material in this section from http://eclipse.org/eclipse/
  - (abridged version of slideset from this site)

Example



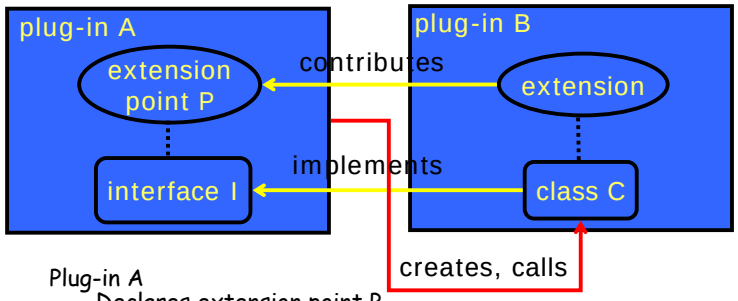
Architectural overview



Plug in approach

- Plug-in - smallest unit of Eclipse function
  - Big example: HTML editor
  - Small example: Action to create zip files
- Extension point - named entity for collecting "contributions"
  - Example: extension point for workbench preference UI
- Extension - a contribution
  - Example: specific HTML editor preferences
- Each plug-in
  - Contributes to 1 or more extension points
  - Optionally declares new extension points
  - Depends on a set of other plug-ins
  - Contains Java code libraries and other files
  - May export Java-based APIs for downstream plug-ins
  - Lives in its own plug-in subdirectory
- Details spelled out in the plug-in manifest (XML)

Example



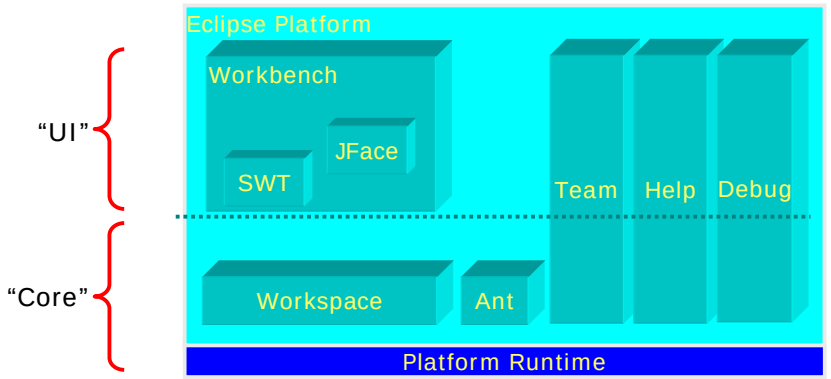
Plug-in A
Declares extension point P
Declares interface I to go with P

Plug-in B
Implements interface I with its own class C
Contributes class C to extension point P

Plug-in A instantiates C and calls its I methods

Eclipse Platform

- Eclipse Platform is the common base
- Consists of several key components



Workspace

- Manages projects which user is working on
- Projects consist of resources (eg source files, folders, projects) in a tree construct
  - Tools read, create, modify, and delete resources in workspace
- Plug-ins access via workspace and resource APIs
  - Allows fast navigation of workspace resource tree
  - Resource change listener for monitoring activity
  - Resource deltas describe batches of changes
  - Maintains limited history of changed/deleted files
  - Several kinds of extensible resource metadata
  - Workspace session lifecycle
  - Incremental project builders
    - Plugins to manage analysis & compilation (eg Java Builder in JDT)

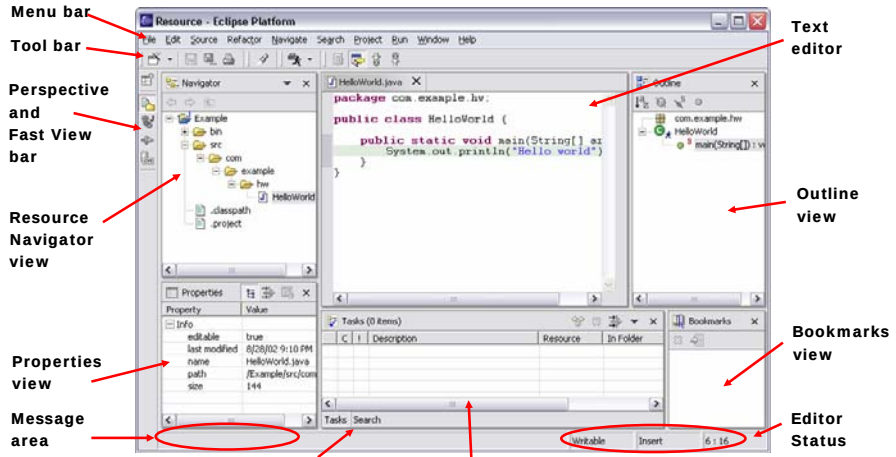
Workbench

- SWT – generic low-level graphics and widget set
  - Generic graphics and GUI widget set
  - OS-independent API
  - Uses native widgets where available, emulates otherwise
- JFace – UI frameworks for common UI tasks
  - Classes for handling common UI tasks
  - API and implementation are window-system independent
- Workbench – UI personality of Eclipse Platform, centred on:
  - Editors
  - Views
  - Perspectives

Workbench

- Editors appear in workbench editor area
  - Contribute actions to workbench menu and tool bars
  - Open, edit, save, close lifecycle
  - Extension point for contributing new types of editors
    - Eg: JDT provides Java source file editor
  - Eclipse Platform includes simple text file editor
- Views provide information on some object
  - By augmenting:
    - Editors, eg: Outline view summarizes content
    - Other views, eg: Properties view describes selection
  - Eclipse Platform includes many standard views: Resource Navigator, Outline, Properties, Tasks, Bookmarks, Search, …
- Perspectives are arrangements of views and editors
  - Different perspectives suited for different user tasks
  - Users can quickly switch between perspectives
  - Eclipse Platform includes standard perspectives: Resource, Debug, …

Workbench in use



Other components

- Team
  - Version and configuration management (VCM)
  - Share resources with team via a repository (project level assocn)
  - Eclipse Platform includes CVS repository provider
- Debug
  - Common debug UI and underlying debug model
- Help
  - Help books are HTML webs presented in standard web browser
  - Help mechanisms available to all plug-ins
  - Help search engine based on Apache Lucene
- Ant
  - Eclipse incorporates Apache Ant
  - Run Ant targets in build files inside or outside workspace
  - PDE uses Ant for building deployed form of plug-in

Platform Summary

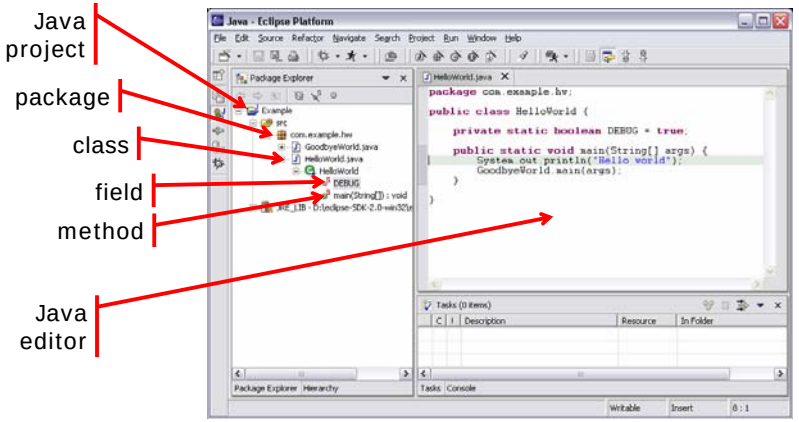
- Eclipse Platform provides the nucleus for IDE products
- Plug-ins, extension points, extensions
  - Open, extensible architecture
- Workspace, projects, files, folders
  - Common place to organize & store development artifacts
- Workbench, editors, views, perspectives
  - Common user presentation and UI paradigm
- Key building blocks and facilities
  - Help, team support, internationalization, …

JDT – Example Eclipse toolset

- **Java development environment**
- **Built on top of Eclipse Platform**
  - **Implemented as Eclipse plug-ins**
  - **Using Eclipse Platform APIs and extension points**
- **Included in Eclipse Project releases**

Provides Java Perspective

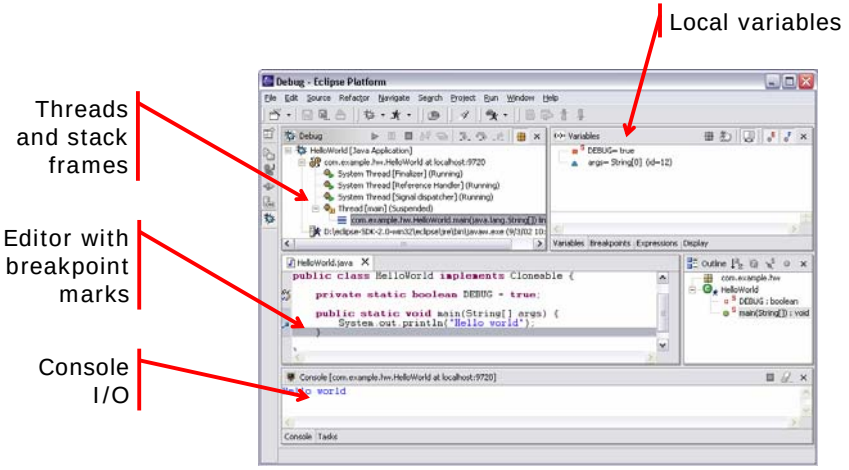
- **Java-centric view of files in Java projects**



Other features

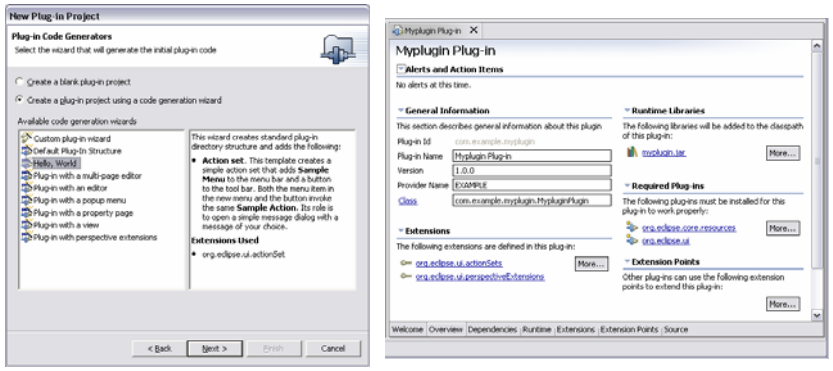
- Move up & down type hierarchies (super <-> sub class)
- Search for elements
- Javadoc tool tips
- Method signature completion suggestions
- Java specific spellcheck and correction suggestion
- Code templates and stub method creation
- Critiquing tools (eg identifier name suggestions)
- Code refactoring
- Java Compiler

Java debugger



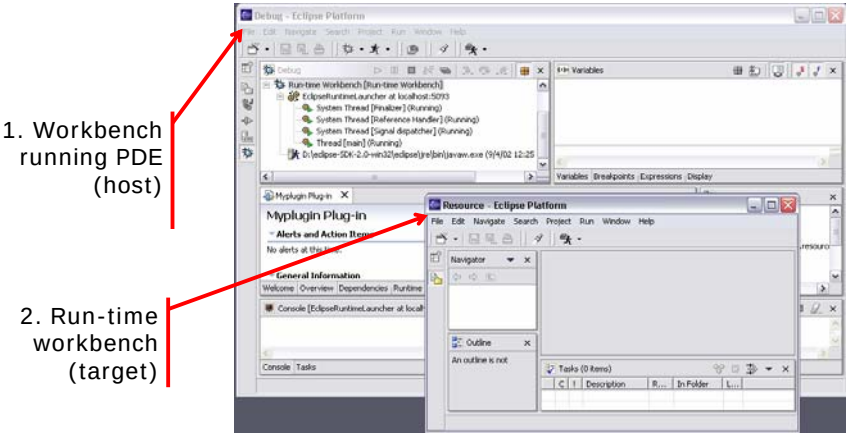
Plugin Development Environment PDE

- Specialized tools for developing Eclipse plug-ins
- PDE templates for creating simple plug-in projects
- Specialized PDE editor for plug-in manifest files



PDE

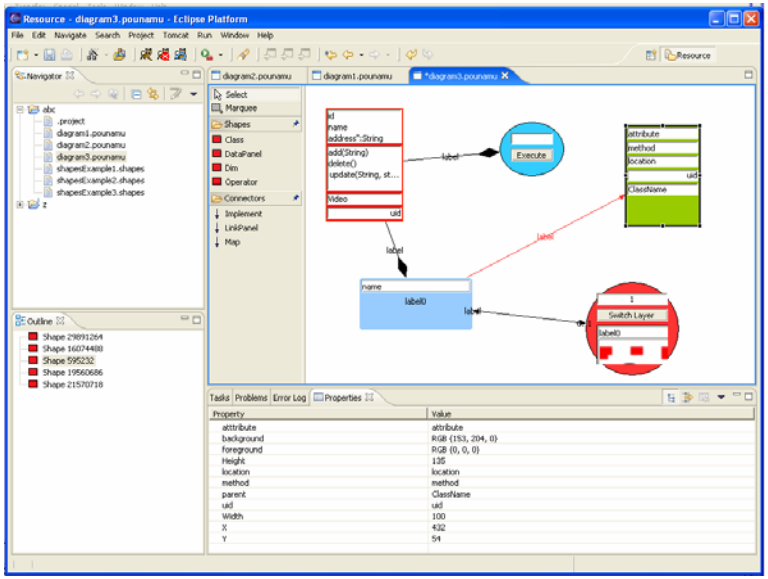
- PDE runs and debugs another Eclipse workbench



Lessons from Eclipse

- Rules for Enablers from Kent Beck's "Contributing to Eclipse"
- Invitation Rule - Whenever possible, let others contribute to your contributions.
- Lazy Loading Rule - Contributions are only loaded when they are needed.
- Safe Platform Rule - As the provider of an extension point, you must protect yourself against misbehavior on the part of extenders.
- Fair Play Rule - All clients play by the same rules, even me.
- Explicit Extension Rule - Declare explicitly where a platform can be extended.
- Diversity Rule - Extension points accept multiple extensions.
- Good Fences Rule - When passing control outside your code, protect yourself.
- Explicit API Rule - separate the API from internals.
- Stability Rule - Once you invite someone to contribute, don't change the rules.
- Defensive API Rule - Reveal only the API in which you are confident, but be prepared to reveal more API as clients ask for it.

Pounamu Eclipse plugin



Eclipse summary

- Eclipse has very rapidly developed significant momentum
  - See plugin site for list of commercial and open source plugins
    - http://eclipse.org/community/plugins.html
- Reasons for success
  - Plenty of basic support for tool building from framework
    - Enough stuff "for free" to overcome inertia of understanding the model and working within it
  - Plugin approach is highly successful
    - Principled enough to allow many plugins to collaborate
  - Open source, but allows commercial extension
- Problems
  - A LOT of things to get your head around if you are starting out developing a plugin
    - Need for more high level support tools to assist in Eclipse tool development (see EFPL lecture later)