Eclipse (1/3)

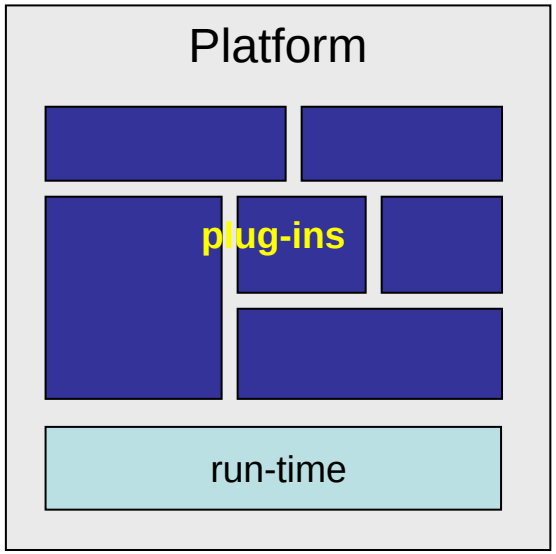
Deepak Dhungana
dhungana@ase.jku.at
Institute for System Engineering and Automation

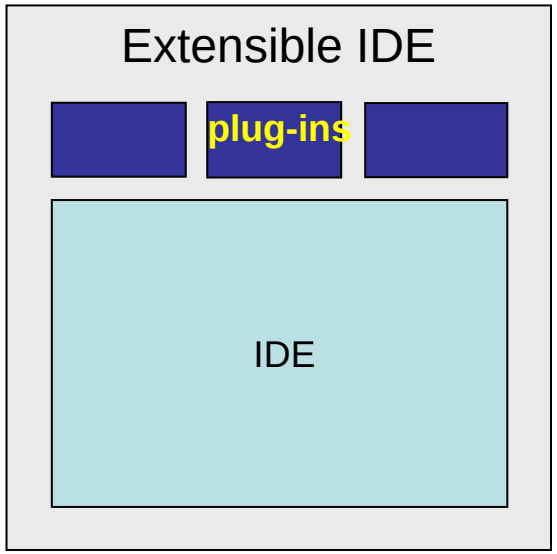
Thomas Wuerthinger
wuerthinger@ssw.jku.at
Institute for System Software

Johannes Kepler University Linz, Austria
http://www.jku.at

What is Eclipse?

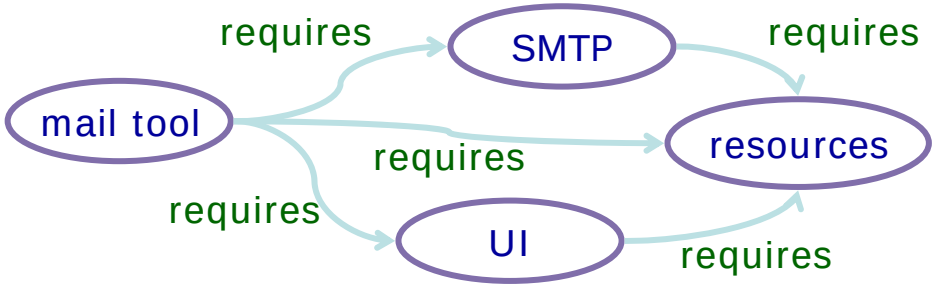
- Eclipse is an extensible platform
  - And not an extensible IDE
  - Everything is a plug-in
  - Plug-in = Component = Bundle (= Module)
  - Plug-ins build on other plug-ins
    - Explicit dependencies





What is the Eclipse Runtime?

- Java component (*plug-in*) model
  - dependency management
  - activation management
  - classloaders

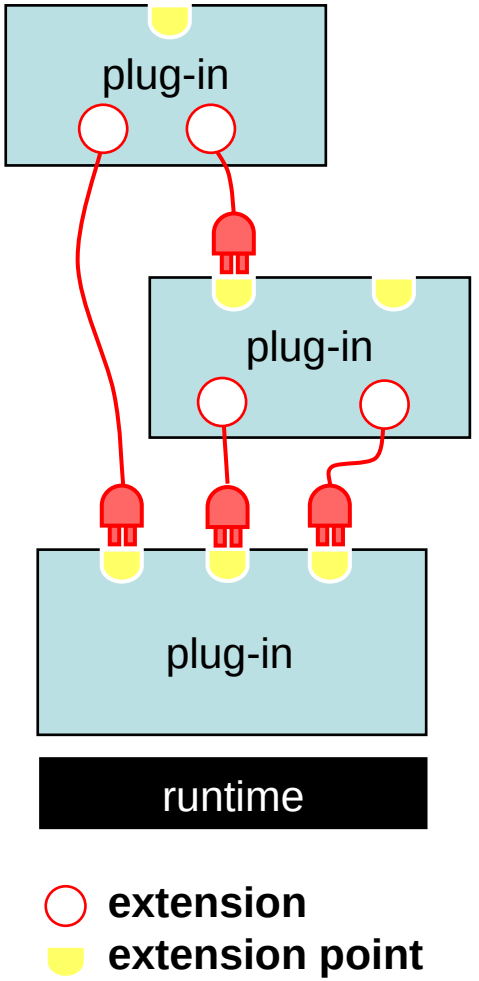


- extension registry manages
  - extension points
  - corresponding extensions

- OSGI based
  - dynamic install / uninstall / update of ***bundles*** (= plug-ins)

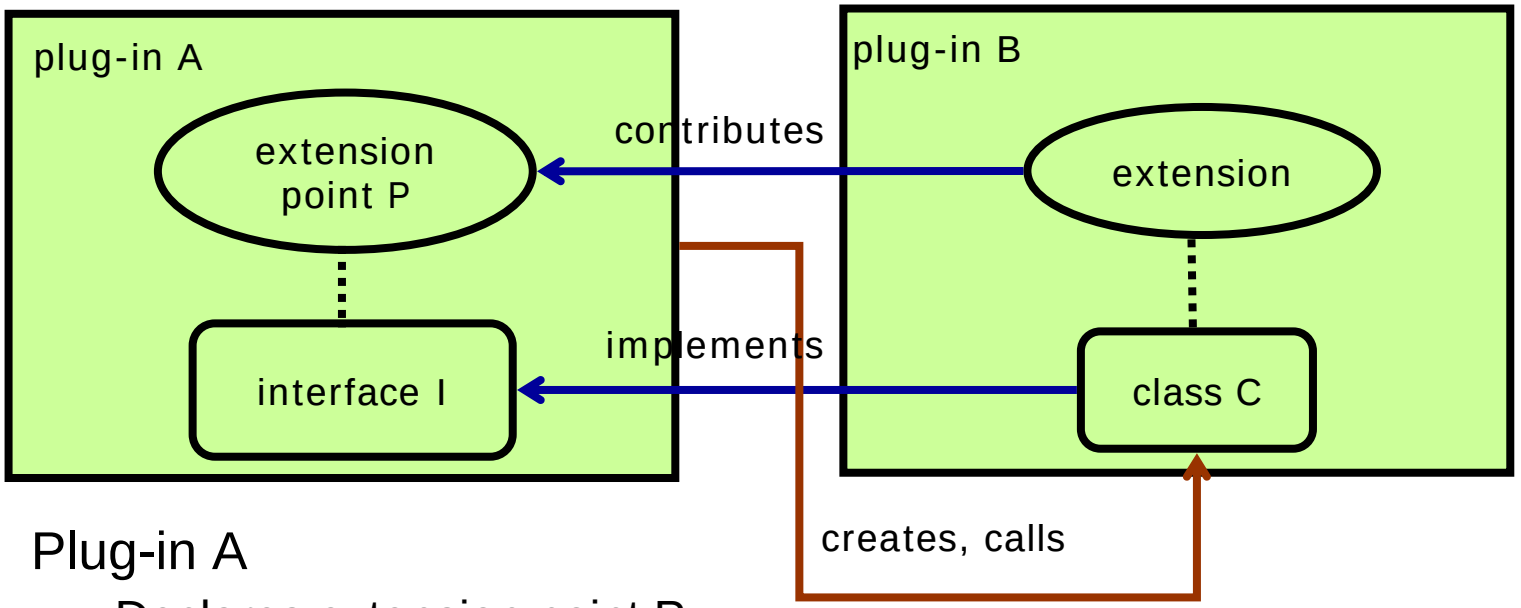
What is a Plug-in?

- **plug-in** = component
  - set of contributions
  - smallest unit of eclipse functionality
  - details spelled out in ***plug-in manifest***
  - large example: mail client
  - small example: spam filter
- **extension point**
  - named entity for collecting contributions
  - example: extension point to add spam filters
- **extension** – a contribution
  - example: a specific spam filter
- **runtime** – controls and manages contributions



Eclipse Plug-in Architecture

- Typical arrangement



- 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

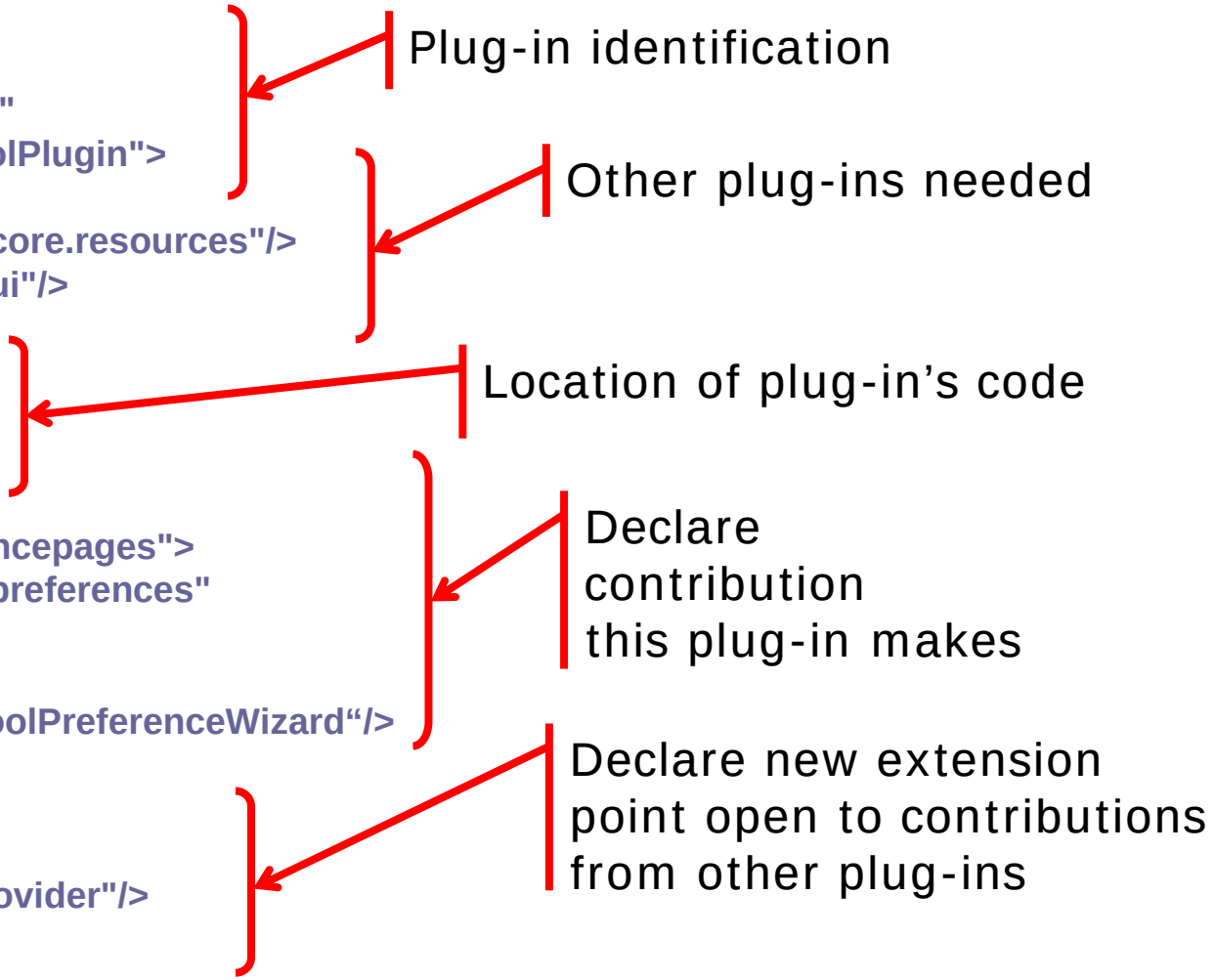
Manifest Files

- MANIFEST.MF
  - Contains the OSGi part of the plug-in specification
  - Plug-in name and version
  - Lifecycle information, e.g. activator class
  - Required Plug-ins
    - When these dependencies are not fulfilled, the plug-in is not started
  - Exported Packages
    - Only these packages are visible for other plug-ins
- plugin.xml
  - Contains the Eclipse-specific part of the plug-in specification
  - Extension point definitions
  - Extension definitions

Plug-in Manifest

plugin.xml

```
<plugin
    id = “com.example.tool"
    name = “Example Plug-in Tool"
    class = "com.example.tool.ToolPlugin">
 <requires>
    <import plugin = "org.eclipse.core.resources"/>
    <import plugin = "org.eclipse.ui"/>
 </requires>
 <runtime>
    <library name = “tool.jar"/>
 </runtime>
 <extension
    point = "org.eclipse.ui.preferencepages">
   <page id = "com.example.tool.preferences"
     icon = "icons/knob.gif"
     title = “Tool Knobs"
     class = "com.example.tool.ToolPreferenceWizard“/>
 </extension>
 <extension-point
    name = “Frob Providers“
    id = "com.example.tool.frobProvider"/>
</plugin>
```

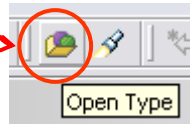


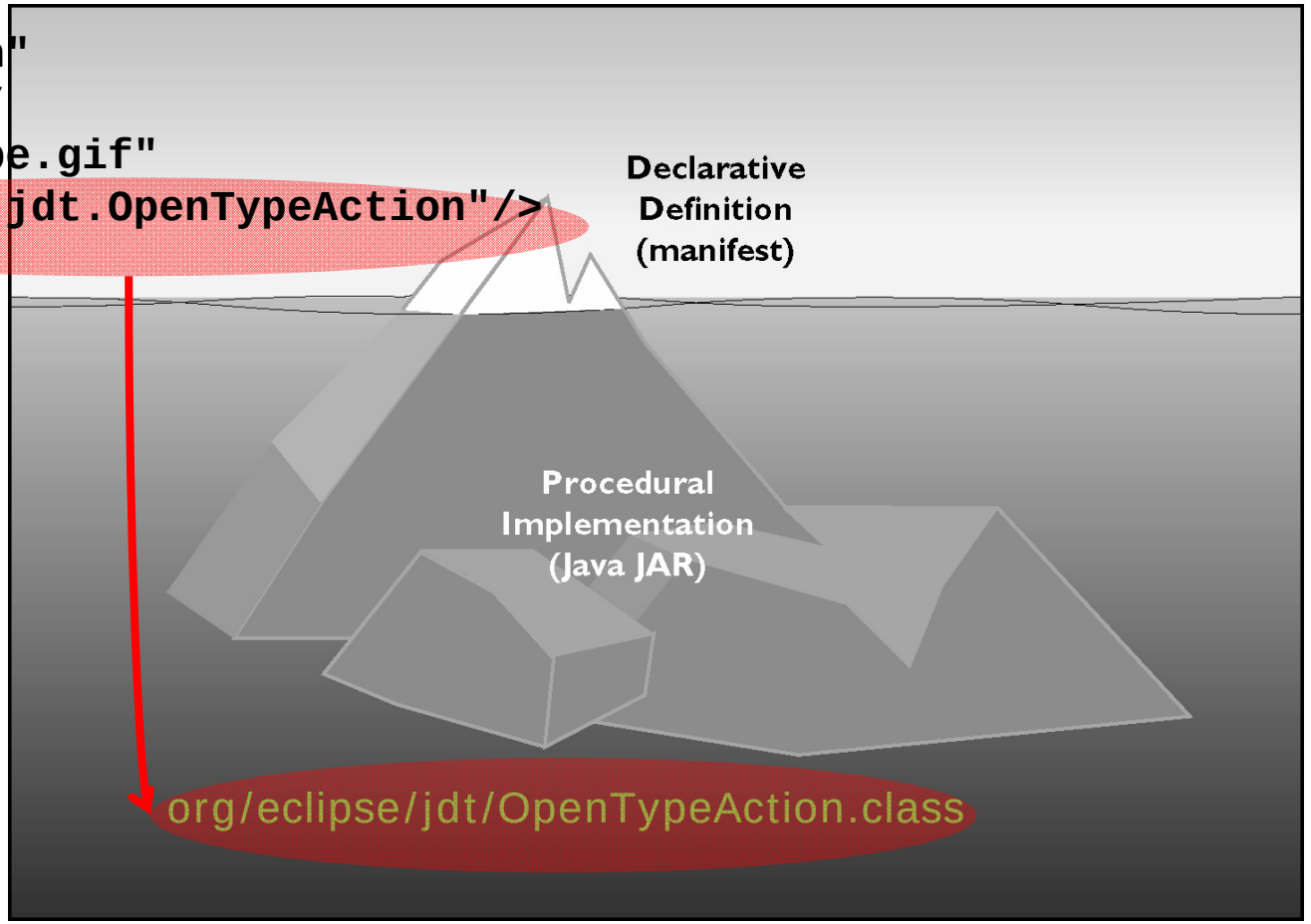
Declaration vs. Implementation

```
<action
   toolbarPath=“search"
   toolTip=“Open Type”
   icon="icons/opentype.gif"
   class="org.eclipse.jdt.OpenTypeAction"/>
```

Lazy loading!

lazily instantiated using reflection



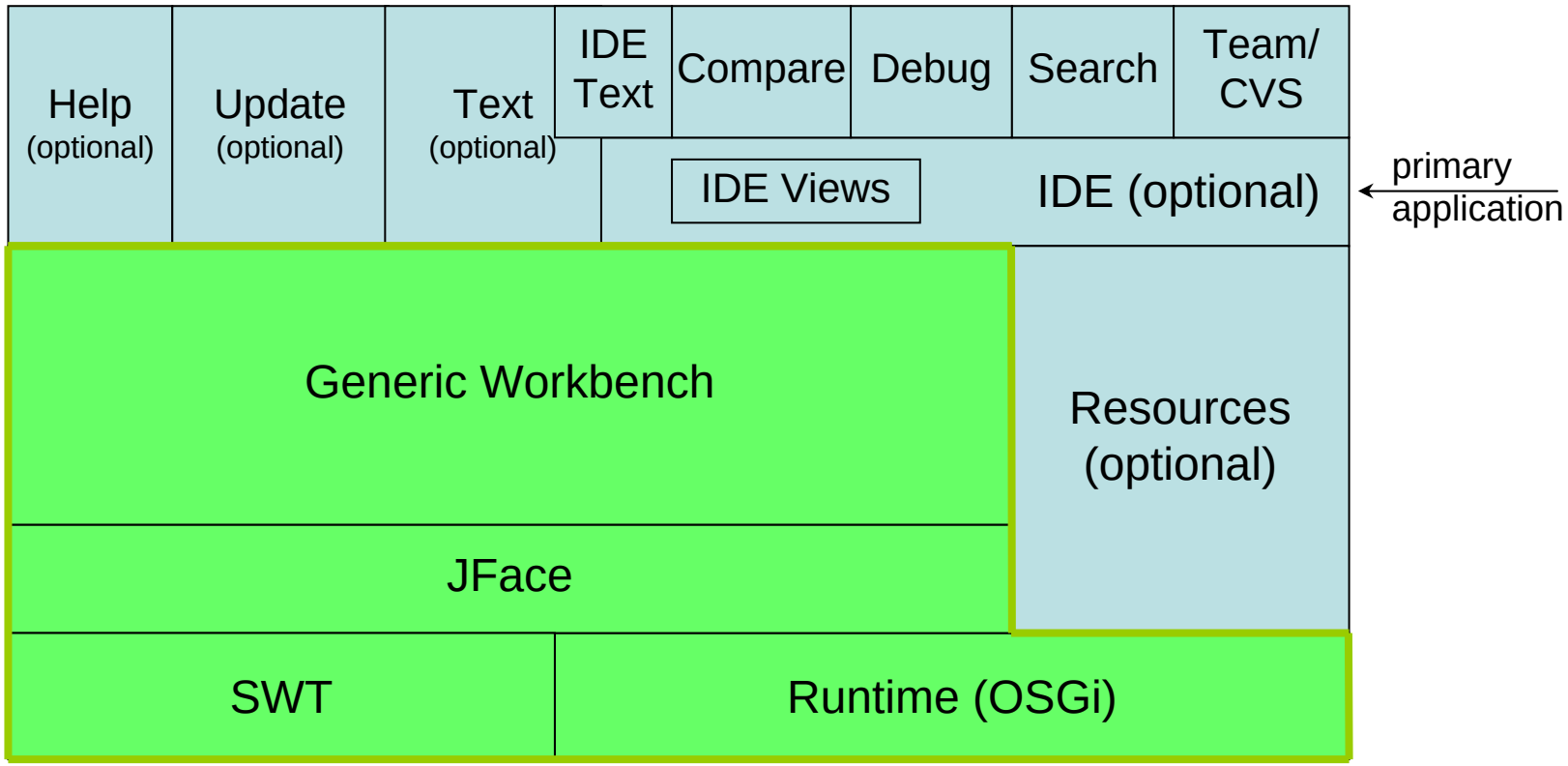


Plug-in Activation

- Each plug-in gets its own Java class loader
  - Delegates to required plug-ins
  - Restricts class visibility to exported APIs
- Contributions processed without plug-in activation
  - Example: Menu constructed from manifest info for contributed items
- Plug-ins are activated only as needed
  - Example: Plug-in activated only when user selects its menu item
  - Scalable for large base of installed plug-ins
  - Helps avoid long start up times

Demonstration

Architecture (since Eclipse 3.0)



SWT, JFace & Workbench

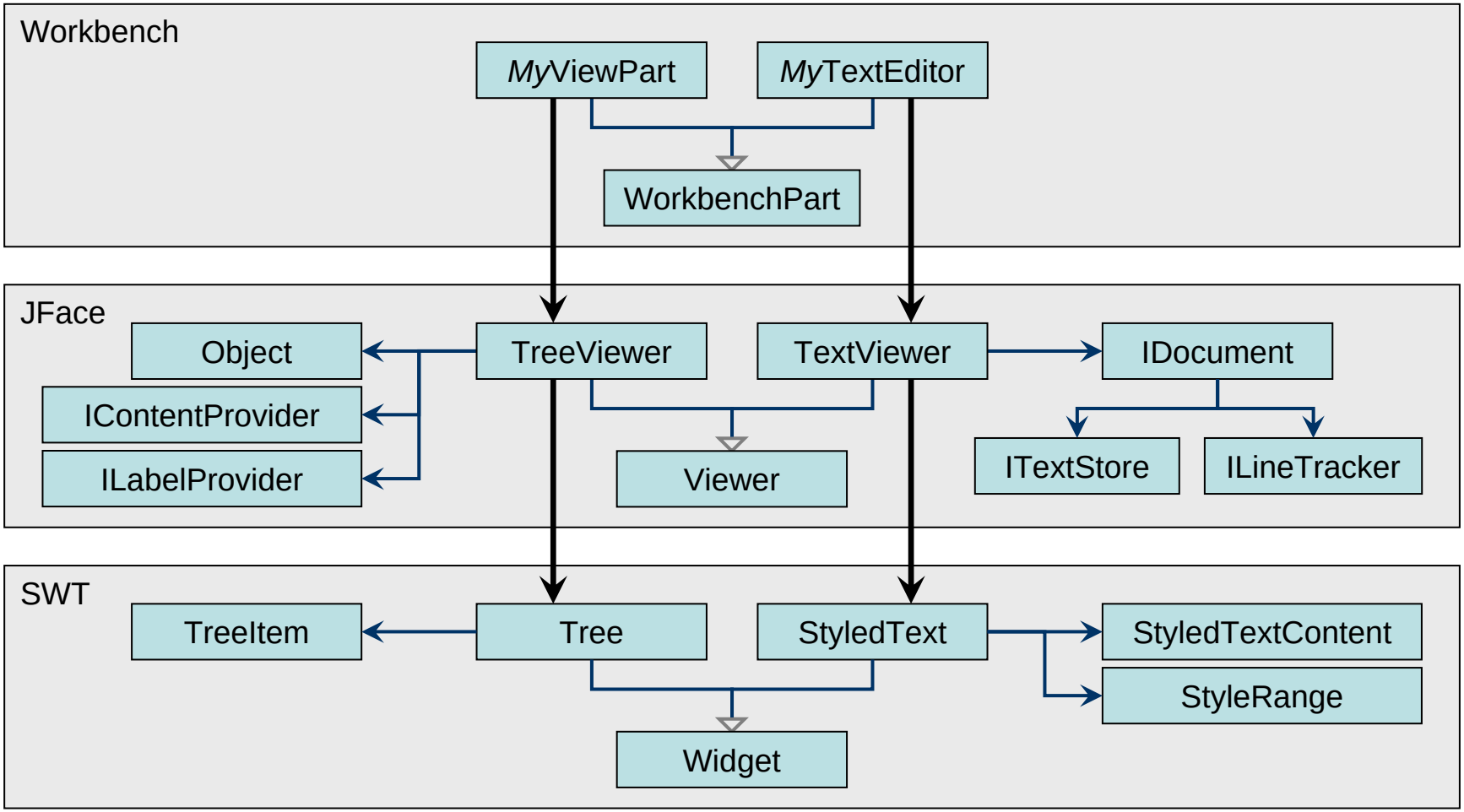
- Standard Widget Toolkit (SWT)
  - Alternative to Swing
    - Completely independent of AWT and Swing
  - Thin and portable layer on top of native widgets
    - No pluggable look&feels
    - Lower overhead than Swing
  - Graphics and drawing
    - GC, Image, Color, Font, ...
  - Complete set of widgets
    - Button, Text, List, Label, Group, Menu, ToolBar, TabFolder, ...
    - Table, Tree, StyledText, Browser, ...
    - Widgets not supported by a platform are emulated in Java

http://www.java2s.com/Code/Java/SWT-JFace-Eclipse/CatalogSWT-JFace-Eclipse.htm

SWT, JFace & Workbench

- JFace
  - Builds on SWT
  - Adds viewers following the Model-View-Control pattern
    - Connects widgets to model elements
    - TableViewer, TreeViewer, TextViewer
  - Higher level application support
    - Actions, Commands, Dialogs, Preferences, ...
    - Management of SWT resources
- Workbench
  - Workbench parts: Editors and Views
    - Don’t mix the terms “Viewer” and “View”
  - “The typical Eclipse look”
  - Declarative definition of menus and toolbars
    - Extension points for contributions to the UI

SWT, JFace & Workbench



SWT – Drawing

- Resource management
  - SWT objects directly encapsulate operation system resources
  - Must be explicitly freed by calling “dispose”
  - Examples: “Color”, “Font”, “Image”, “GC”
  - Objects are bound to a specific device
    - Do not re-use e.g. a display-color for printing
- Device-independent data classes
  - Occupy no operating system resources
  - Examples: “RGB”, “FontData”, “ImageData”
- Class “org.eclipse.swt.graphics.GC”
  - Offers all standard methods for drawing lines, figures, text, ...
  - Some features: antialiasing, clipping, transformation
  - Constructor for creating a “GC” for any “Drawable”
    - All controls
    - Devices: “Display” and “Printer”
    - Images
  - For paint events, a suitable GC is passed as an argument

http://www.eclipse.org/swt/snippets/

SWT – Widgets

- Programming similar to Swing
  - Composite-pattern
  - Event-oriented: Listeners are notified when user performs input
    - “SelectionListener”, “MouseListener”, “KeyListener”
    - Adapter available if a listener has multiple methods
  - Sizing and positioning of widgets done by “Layout”
    - “RowLayout”, “GridLayout”, “FormLayout”

- Differences
  - Parent widget must be specified in constructor
    - Not possible to change parent later
  - Style-bits in constructor
    - Possible values are constants in class “SWT”
    - Most styles cannot be changed later
    - Example: “READ_ONLY”, “SINGLE” – “MULTI”, “WRAP”
  - User-defined painting is done by adding a “PaintListener”

SWT – Widgets

- “Shell”
  - The top-level window
  - Style-bits determine the trimming (resizable, dialog, ...)
- “Label”
- “Button”
  - Check-boxes and Radio-buttons are special styles
- “Text”
  - Multi-line is a special style
- “List”
- “Combo”
- “Menu”
- “Canvas”
  - User-drawn control
- “Tray”
  - Icons shown in the system tray
- “Composite”
  - For grouping of widgets
- “ScrolledComposite”
  - For Scrolling of widgets
- “Table”
- “Tree”
  - With all Windows-features
- “Toolbar”
- “Coolbar”
  - User-movable toolbars
- “StyledText”
  - The Eclipse text editor
- “Browser”
  - The system web-browser
  - Only if available

SWT – Example (1)

```
private Display display;
private Shell shell;

public void main() {
  display = new Display();

  openShell();

  while (!shell.isDisposed()) {
    if (!display.readAndDispatch()) {
      display.sleep();
    }
  }
  display.dispose();
}
```

Create the display

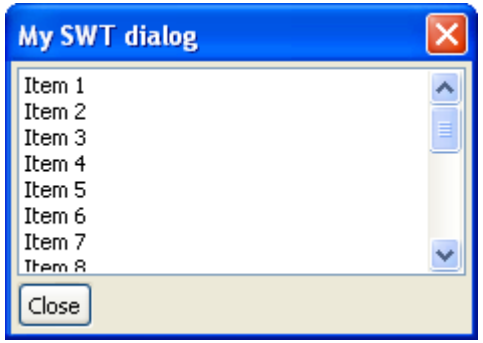
Open a shell (see next slide)

Event loop: Process window events until the shell is closed

Process the next event

Sleep if nothing to do

Dispose the display
The shell and all widgets are already disposed



SWT – Example (2)

```
private SelectionListener buttonListener = new SelectionAdapter() {
  public void widgetSelected(SelectionEvent e) {
    shell.close();
  }
};

public void openShell() {
  shell = new Shell(display, SWT.DIALOG_TRIM);
  shell.setText("My SWT dialog");
  shell.setLayout(new RowLayout(SWT.VERTICAL));

  List list = new List(shell, SWT.BORDER | SWT.V_SCROLL);
  list.setLayoutData(new RowData(200, 100));
  for (int i = 1; i < 20; i++) {
    list.add("Item " + i);
  }

  Button button = new Button(shell, SWT.PUSH);
  button.setText("Close");
  button.addSelectionListener(buttonListener);

  shell.pack();
  shell.open();
}
```

Listener when button is selected, i.e. pressed

Close the shell

Create a top level window with dialog trimmings

Layout for child widgets

Create a list with a border and a vertical scrollbar

Layout data (class according to layout of parent

Create a push button and add a selection listener

Apply the layout and show the window

SWT – Threading issues

- Each display is bound to a thread
  - Called the “user-interface thread”
  - This thread executes the main event loop
    - Dispatch of operating system events
- SWT is not thread-safe
  - Resource objects must only be accessed by user interface thread
  - “SWTException” when method called from wrong thread
    - Better than an unexpected behavior
- Execute code from a non-UI thread
  - “display.syncExec(runnable)” or “display.asyncExec(runnable)”
  - The run-method of the “Runnable” is executed in the UI thread

Develop and Run SWT Applications

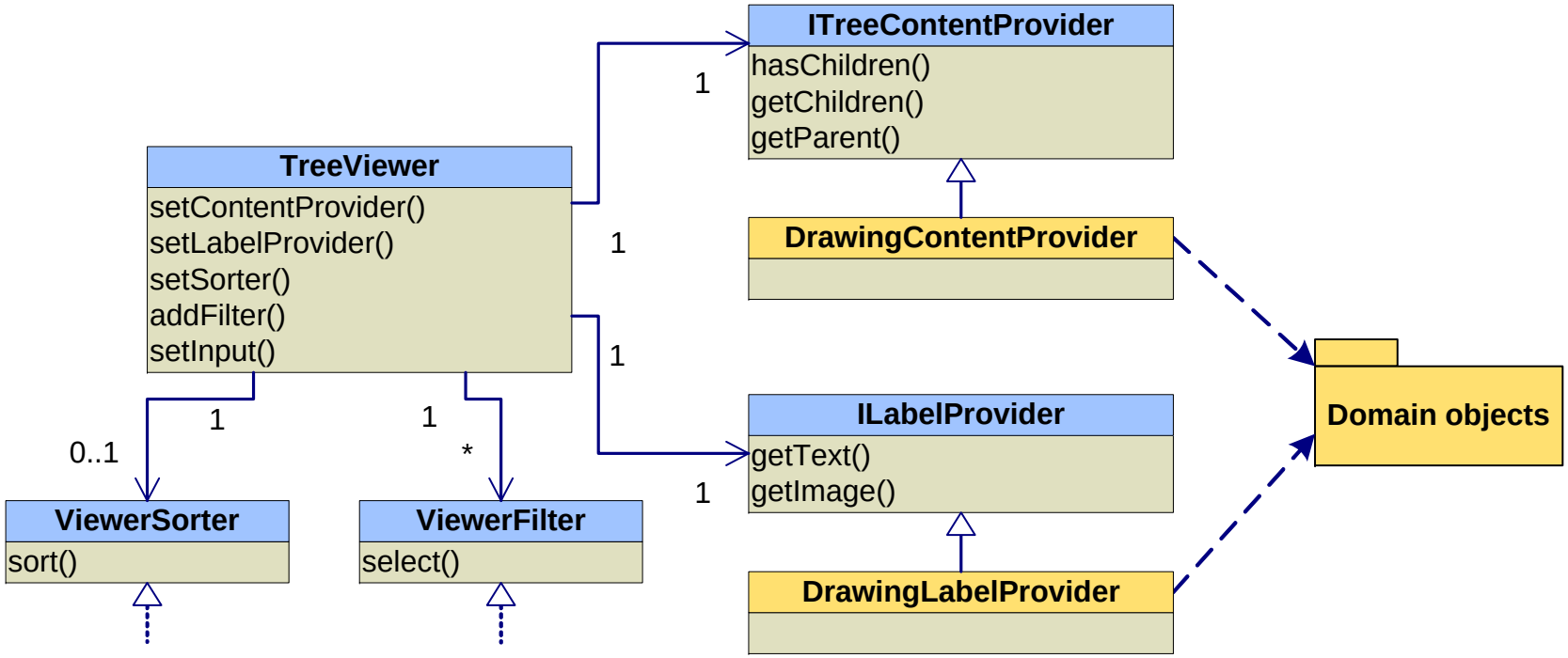
- SWT is independent from Eclipse
  - No dependency on OSGi or the Eclipse runtime
- Download the SWT-bundle from the Eclipse homepage
  - Or: use the files from the plug-in org.eclipse.swt
- SWT consists of Java classes and a native library
  - swt.jar must be on the classpath
  - swt*.dll (on Windows) must be in the native library path

http://wiki.eclipse.org/index.php/JFaceSnippets

JFace – Viewer

- Viewer connect widgets to a model
  - For the more complex SWT widgets like tables and trees
- SWT
  - Add “TableItem” or “TreeItem” objects to a “Table” or “Tree”
  - One method for structure, labels, sorting and filtering
  - Difficult to update when domain objects changed
    - Re-build the whole content
- JFace
  - Configure a “TableViewer” or “TreeViewer”
  - Adapter on domain objects for structure and labels
  - Strategy objects for sorting and filtering
  - Adapters and Strategies are re-useable for multiple viewers
  - Simple to update when domain objects are changed
    - Efficient handling included in viewer

JFace – TreeViewer



JFace – TreeViewer

- Content provider
  - Example: “DrawingContentProvider”
  - Given an input object, returns a set of corresponding domain objects
- Label provider
  - Example: “DrawingLabelProvider”
  - Returns a string and icon for presenting a domain object
- Sorter
  - Sort a set of domain objects
- Filter
  - Determine the domain objects that should be shown in the viewer
  - Multiple filters can be applied
- Update when domain objects change
  - Content and label provider add a listener on domain objects

Workbench

- Workbench Window
  - The top level window
  - User can open multiple windows
- Menu Bar, Tool Bar
  - Shared area where all plug-ins can contribute actions
- Status line
  - Shared area
- Editors
  - Editor tool bar is integrated in main tool bar
- Views
  - Views have their own tool bar
- Page
  - Area where editors and views are shown
  - One workbench window has exactly one page

Editors vs. Views

- Editors
  - Opened by the user on a specific element
  - Open-modify-safe life cycle
  - Explicit input data (instance of “IEditorInput”)
  - Multiple editors of the same kind can be open
  - Always shown in the editor area
  - Contribute to the global tool bar and menu
- Views
  - Opened without specifying an input
  - Views know where to fetch its input
  - Changes are immediately applied
  - Open at most once (usually)
  - Can be moved around by the user
  - Have their own tool bars and menus

Kinds of Views

- Navigator views
  - Present hierarchical structures
  - Allow to open an editor for a selected element
  - Example: Navigator, Package Explorer
- Outline views
  - Present the structure of the contents of the active editor
  - Example: Outline
- Information / Detail views
  - Provide additional information about the selected objects
  - Usually track the current selection
  - Example: Properties
- Result / Output views
  - Show the result or output of an operation
  - Example: Search, Console
- Collector views
  - Collect particular artifacts
  - Allow navigation to the original location
  - Example: Tasks, Bookmarks, Problems

Basic Extension Points

- org.eclipse.ui.views
  - Add a view that can be opened using “Show View”
  - Views are grouped in categories
- org.eclipse.ui.editors
  - Add an editor for a specific file type or extension
- org.eclipse.ui.actionSets
  - Add actions to the main menu or toolbar
- org.eclipse.ui.viewActions
  - Add declarative actions to the menu or toolbar of a view
- org.eclipse.ui.editorActions
  - Add declarative actions to an editor
- org.eclipse.ui.popupMenus
  - Contribute to the popup menu of a view or editor
  - The popup menu itself must be provided by the part
  - Viewer contribution
    - Entry is shown in popup menu of a specified part
  - Object contribution
    - Entry is shown if an entry of a specific class is selected

ID-Strings

- Used to identify extensions in the manifest
  - Used by Java code to reference an extension
  - Normal string constants in Java code
- Naming conventions
  - All id-strings should be unique
  - Use naming conventions of packages
    - Avoids problems if tools of different vendors are combined
- Strings are not checked by the compiler
  - Be sure to always change all references
    - Frequent bug
  - Use one string constant per id-string

Actions

- Defined in JFace
  - Basic interfaces and classes
  - Label, image, run-method
  - Can be added to menus, toolbars and status lines
- Workbench specifies how to add actions to
  - Main menu and toolbar
  - View menu and toolbar
  - Editor specific menu and toolbar
  - Context menu
  - Status line
- Programmatic vs. declarative actions
  - Programmatic: Action instantiated in Java source code
  - Declarative: Action specified in plugin.xml
    - Contribute actions to parts declared in other plug-ins
    - Part must support extension