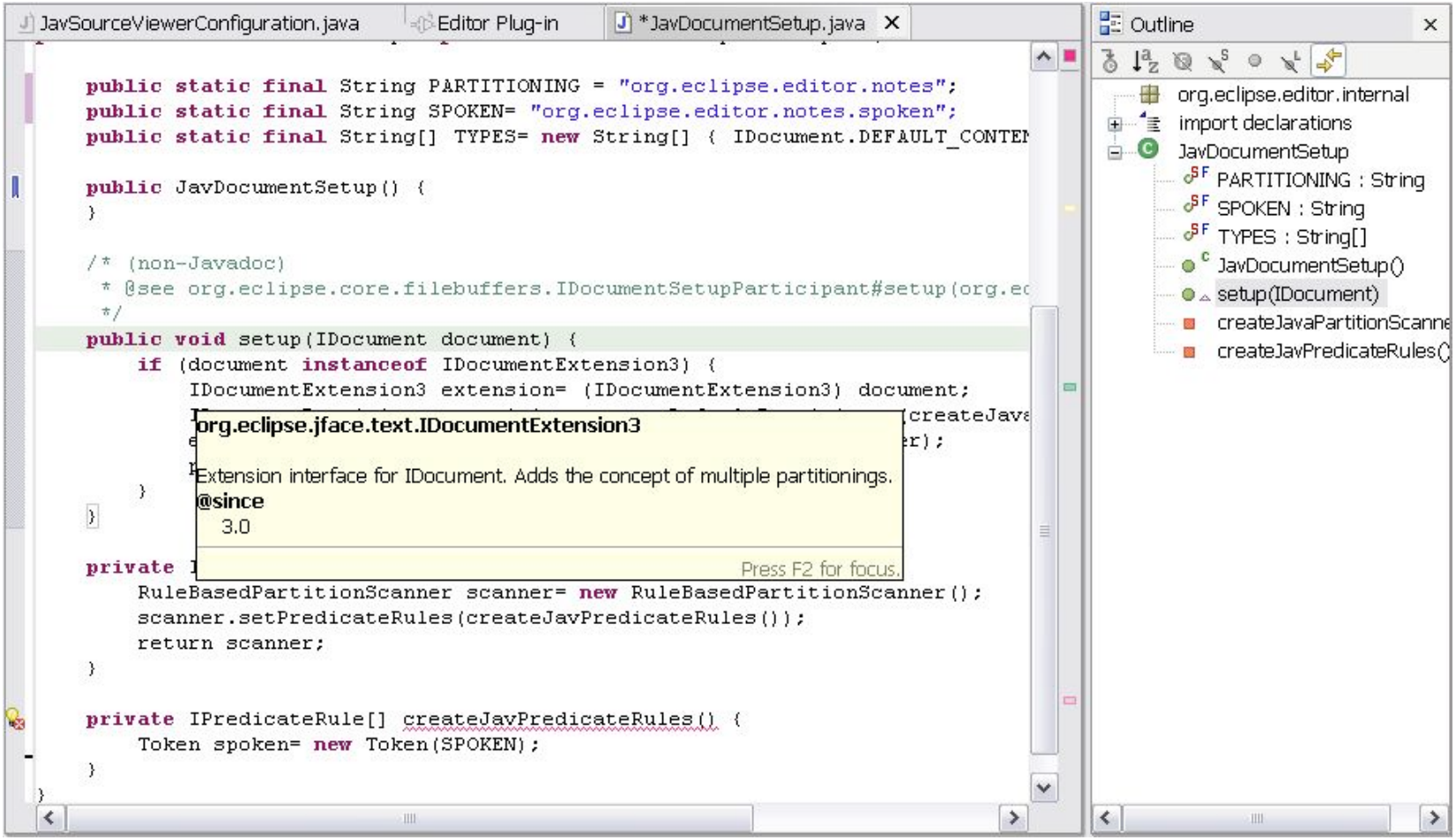
eclipse CON 2004

Text Editors and How to Implement Your Own

Kai-Uwe Mätzel
IBM OTI Labs Zurich, Switzerland

Example: Java Editor



Java Editor

- functionality
  - text coloring, current line highlighting, multi-column vertical ruler, overview ruler, error squiggles, different kind of hovers, dynamically updated outliner, content assist, bracket matching, find/replace, annotation navigation, …

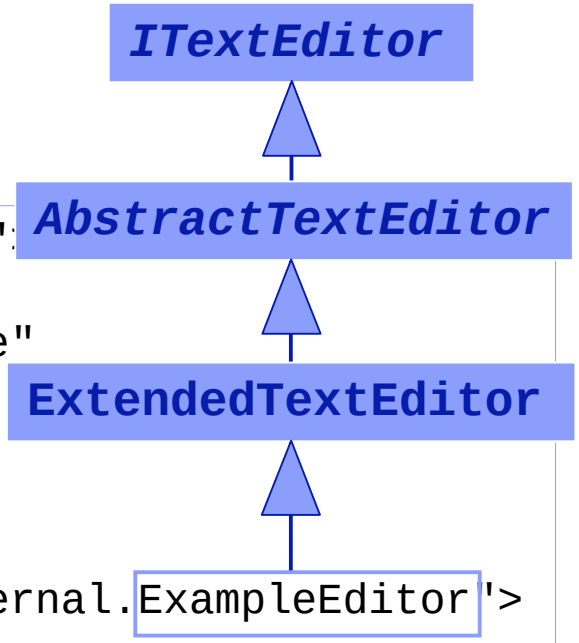
➢ variety conceptually different functions
➢ nested control flows
➢ interleaving control flows

Overview

- defining a text editor
- the basics of text editors
- connecting to the menu bar and tool bar
- adding actions to the text editor
- adding syntax highlighting
- available configuration options
- architecture
- outlook

Defining a text editor

```
<extension point="org.eclipse.ui.editors">
     <editor
           id="org.eclipse.editor.example"
           name="Example Editor"
           extensions="expl"
           default="true"
           icon="icons/example.gif"
           class="org.eclipse.editor.internal.ExampleEditor">
     </editor>
</extension>
```



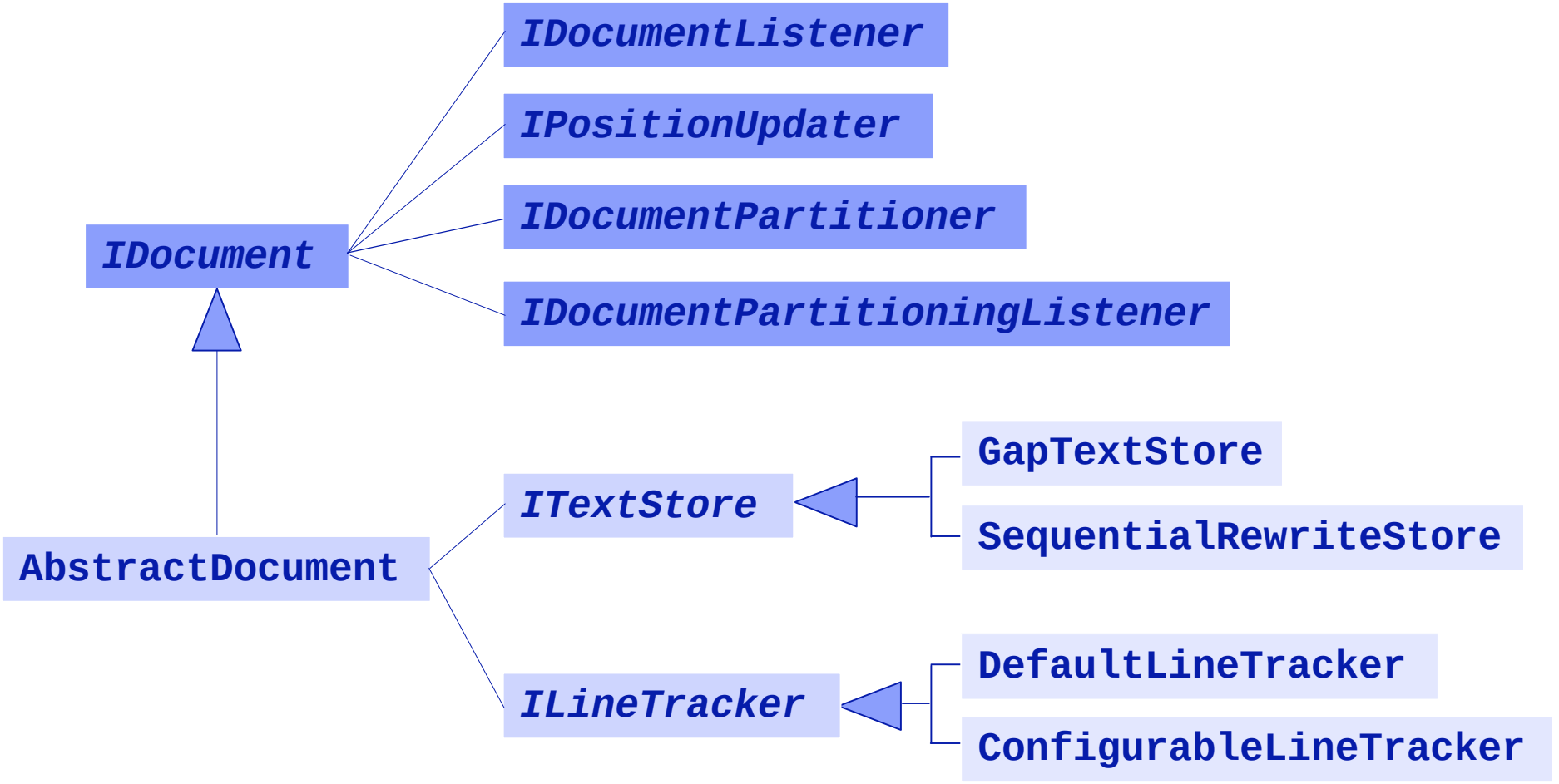
- can be opened on files with the **`expl`** extension
- lives in the workbench – not yet connected to the tool or menu bar
- works on documents

What are documents?

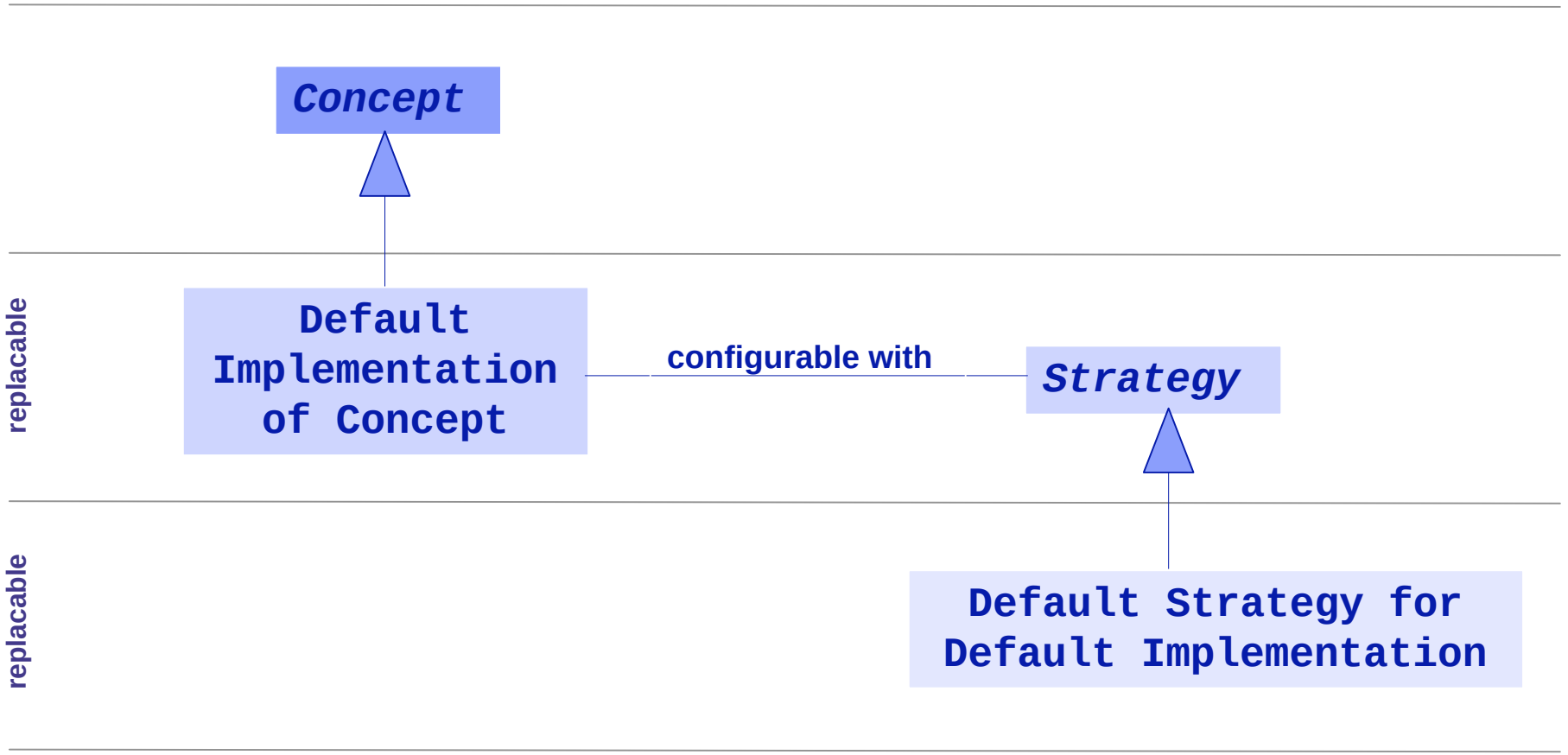
Documents store text and provide support for

- line information
- text manipulation
- document change listeners
- customizable position management
- search
- customizable partition management
- document partition change listeners

IDocument and its implementation



Recurring abstraction layers



Where do documents come from?

```
interface ITextEditor extends IEditorPart {
…
    IDocumentProvider getDocumentProvider();
}
```

- each editor is connected to a document provider
- document providers can be shared between editors

- document provider
  - maps editor inputs onto documents and annotation models
  - tracks and communicates changes to the editor inputs into editor understandable events (`IElementChangeListener`)
  - translates changes of the documents and annotation models into changes of the editor input (save)
  - manages dirty state, modification stamps, encoding
  - provides uniform access to editor inputs and their underlying elements

Available document providers

- `StorageDocumentProvider`
  - specialized on `IStorageEditorInput`
- `TextFileDocumentProvider`
  - specialized on `IFileEditorInput`
  - replaces `FileDocumentProvider`
  - thin layer on top of file buffers (`FileBuffers`)
  - file buffers assume most of the responsibility of document providers in an editor independent way

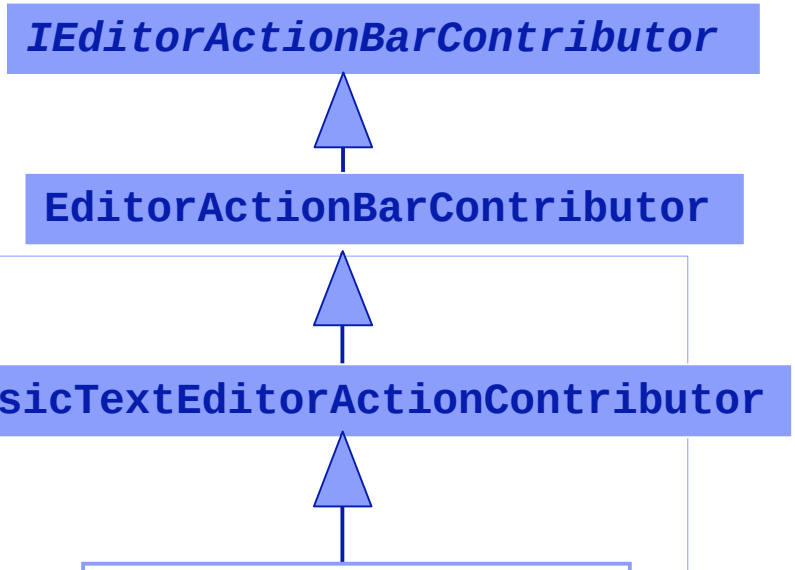
```
<extension point="org.eclipse.ui.editors.documentProviders">
   <provider
         class="org.eclipse.ui.editors.text.TextFileDocumentProvider"
         inputTypes="org.eclipse.ui.IStorageEditorInput"
         id="org.eclipse.ui.editors.text.StorageDocumentProvider">
   </provider>
</extension>
```

Basic setup

```
public class AbstractTextEditor implements IElementStateListener {

   public void setInput(IEditorInput input) {
     releaseInput();
     IDocumentProvider p= getDocumentProvider(input);
     p.addElementStateListener(this);
     p.connect(input);
     setDocument(p.getDocument(input));
     setAnnotationModel(p.getAnnotationModel(input));
   }

   public void elementDeleted(Object element) {
     if (isInput(element)) …
   }

   public void elementDirtyStateChanged(Object element) {…}
   …
}
```

Connecting to the workbench

- editor action bar contributor
  - connects editors with the tool and menu bar of the workbench
  - editor action bar contributors are shared on a per editor type base




```
<extension point="org.eclipse.ui.editors">
   <editor
     id="org.eclipse.editor.example"
     name="Example Editor"
     extensions="expl"
     default="true"
     icon="icons/example.gif"
     contributorClass="org.eclipse.editor.internal.ExampleActionContributor"
     class="org.eclipse.editor.internal.ExampleEditor">
   </editor>
</extension>
```

ExampleActionContributor

```
public class ExampleActionContributor
    extends BasicTextEditorActionContributor {

  public void setActiveEditor(IEditorPart part) {
    super.setActiveEditor(part);

    if (!(part instanceof ITextEditor)) return;
    IActionBars actionBars= getActionBars();
    if (actionBars == null) return;

    ITextEditor editor= (ITextEditor) part;
    actionBars.setGlobalActionHandler(
         IDEActionFactory.ADD_TASK.getId(),
         getAction(editor, IDEActionFactory.ADD_TASK.getId()));

    actionBars.setGlobalActionHandler(
         IDEActionFactory.BOOKMARK.getId(),
         getAction(editor, IDEActionFactory.BOOKMARK.getId()));
  }
}
```

Adding actions

- override `BasicTextEditorActionContributor.contributeToMenu()`

```
public void contributeToMenu(IMenuManager menu) {
   IMenuManager m= menu.findMenuUsingPath(IWorkbenchActionConstants.M_EDIT);
   m.appendToGroup(IWorkbenchActionConstants.FIND_EXT, getFindAction());
}
```

- contribute to extension point `org.eclipse.ui.editorActions`
- contribute to extension point `org.eclipse.ui.popupMenus`
- "automatic actions": selection and post selection listeners

- `ITextEditor.setAction(String id, IAction action)`
  - id.. command id or **RulerDoubleClick** or **RulerClick**
  - actions are automatically registered with the key binding service

AbstractTextEditor action management

- override `AbstractTextEditor.createActions()`

```
protected void createActions() {
  super.createActions();
  Action action= new ExampleAction();
  action.setHelpContextId("org.eclipse.editor.example.action.context");
  action.setActionDefinitionId("org.eclipse.editor.example.command");
  setAction("org.eclipse.editor.example.command", action);
  markAsSelectionDependentAction("org.eclipse.editor.example.command", true);
  markAsContentDependentAction("org.eclipse.editor.example.command", true);
}
```

- for context menu additions override
  - `editorContextMenuAboutToShow()`
  - `rulerContextMenuAboutToShow()`

Inside AbstractTextEditor

- `AbstractTextEditor` uses a `SourceViewer` as implementation

```
public abstract class AbstractExampleEditor {
  public void createPartControl(Composite parent) {
     fSourceViewer= new SourceViewer(parent, …);
  }
…
}
```

- customization via `SourceViewerConfiguration`

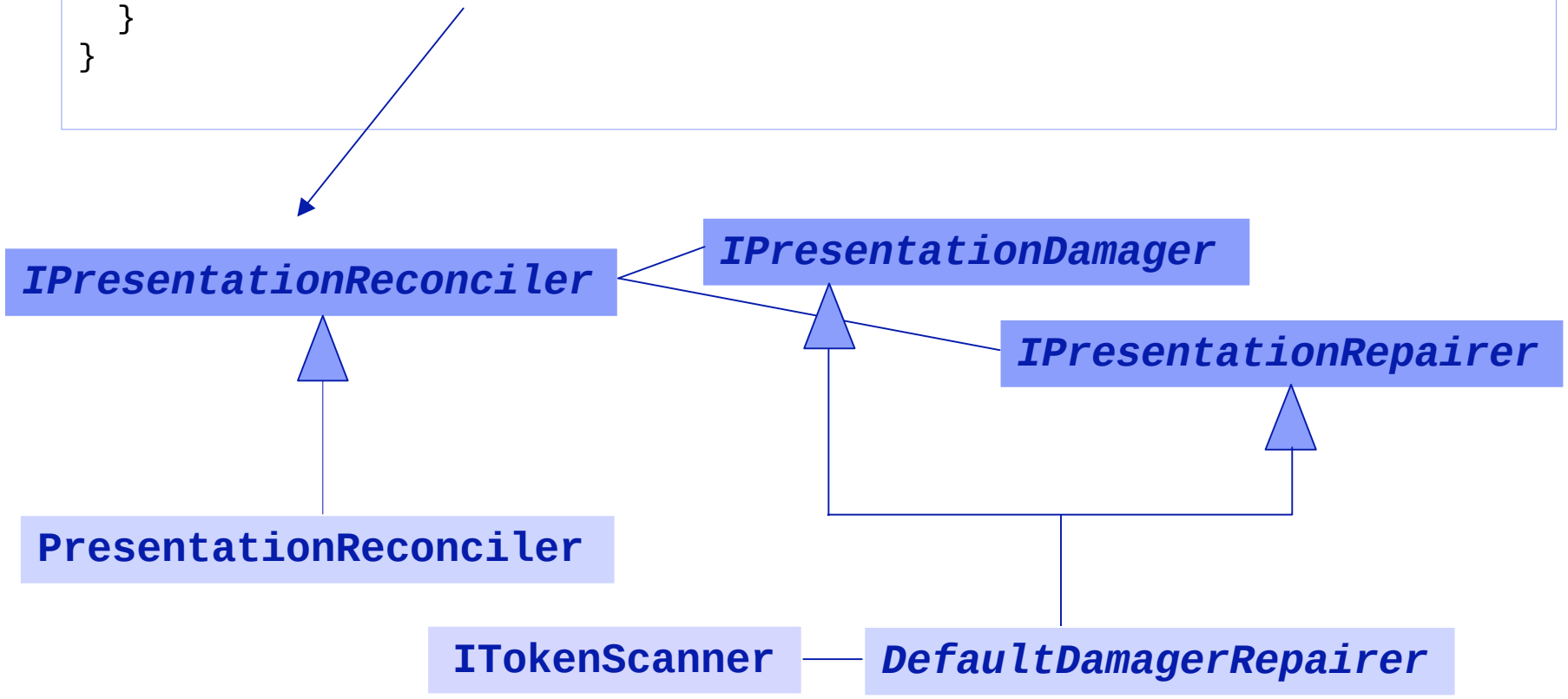
```
public class ExampleEditor extends ExtendedTextEditor {
  protected void initializeEditor() {
    super.initializeEditor();
    setSourceViewerConfiguration(new ExampleSourceViewerConfiguration());
  }
…
}
```

Adding an annotation hover

```
public class ExampleSourceViewerConfiguration
       extends SourceViewerConfiguration {

  public IAnnotationHover getAnnotationHover(ISourceViewer viewer) {
    return new ExampleAnnotationHover();
  }
}
…
public class ExampleAnnotationHover implements IAnnotationHover {

  private List getAnnotations(ISourceViewer viewer, int lineNumber) {
    IAnnotationModel model= viewer.getAnnotationModel();
    IDocument document= viewer.getDocument();
    return getAnnotationsAtLine(model, document, lineNumber);
  }

  public String getHoverInfo(ISourceViewer viewer, int lineNumber) {
    List annotations= getAnnotations(viewer, lineNumber);
    return printAnnotations(annotations);
  }
…
}
```

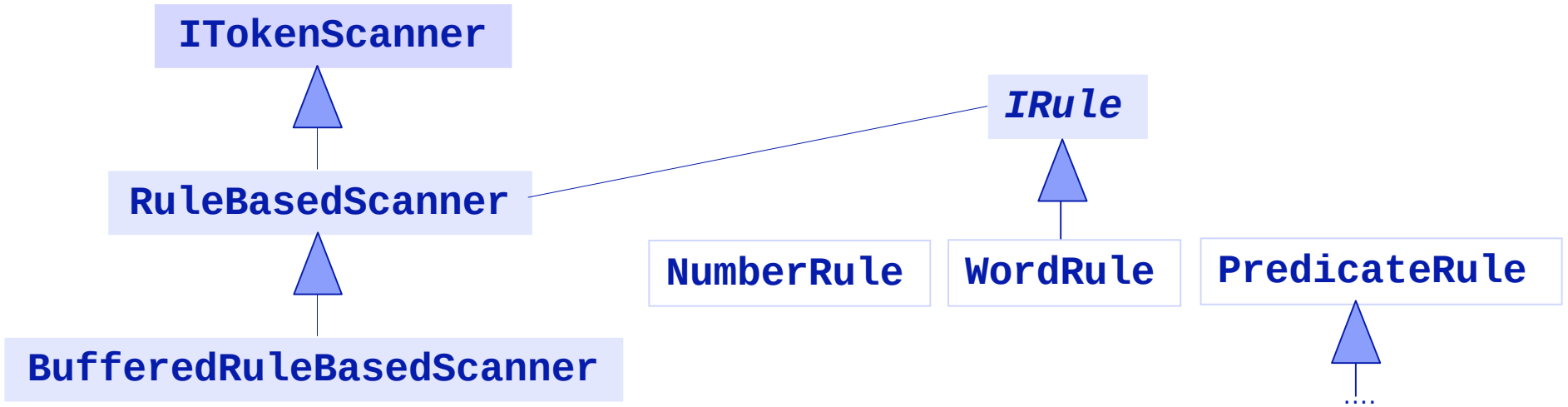
Adding syntax coloring

```
public class ExampleSourceViewerConfiguration {
…
  IPresentationReconciler getPresentationReconciler(ISourceViewer viewer) {
    return IPresentationReconciler;
  }
}
```



Adding syntax coloring

```
public class ExampleSourceViewerConfiguration {
  …
  IPresentationReconciler getPresentationReconciler(ISourceViewer viewer) {
    PresentationReconciler reconciler= new PresentationReconciler();
    DefaultDamagerRepairer dflt= new DefaultDamagerRepairer(ITokenScanner);
    reconciler.setDamager(dflt, IDocument.DEFAULT_CONTENT_TYPE);
    reconciler.setRepairer(dflt, IDocument.DEFAULT_CONTENT_TYPE);
    return reconciler;
  }
}
```



Adding syntax coloring

```
public class ExampleSourceViewerConfiguration {
  …
  private ITokenScanner createTokenScanner() {
    RuleBasedScanner scanner= new RuleBasedScanner();
    scanner.setRules(createRules());
    return scanner;
  }

  private IRule[] createRules() {
    IToken tokenA= new Token(new TextAttribute(getBlueColor());
    IToken tokenB= new Token(new TextAttribute(getGrayColor());

    return new IRule[] {
      new PatternRule(">", "<", tokenA, '\\', false),
      new EndOfLineRule("-- ", tokenB)
    };
  }
}
```

Adding syntax coloring

- BUT multi-line ">…<" tokens are not correctly colored
- `DefaultDamagerRepairer` relies on partitioning information of the document to be presented: The damager only damages a single line unless it detects changes in the document partitioning.

➢ When using the `DefaultDamagerRepairer` together with rules intended to span multiple lines, the multi-line regions must be reflected as document partitions.

Partitioning and source viewers

- partitioning is a semantic view onto the document
  - each partition has a content type
  - each character of a document belongs to a partition
  - documents support multiple partitionings
  - partitioning is always up-to-date
- allows for customizing viewer behavior based on content types
  - specify damager/repairers per content type
  - specify text hover per content type
  - …

Partitioning a document

- document creation and setup is managed by the file buffer manager
- ➢ participate in the document setup process of the file buffer manager

```
<extension point="org.eclipse.core.filebuffers.documentSetup">
   <participant
       extensions="expl"
       class="org.eclipse.editor.internal.ExampleDocumentSetup">
   </participant>
</extension>
```

Partitioning a document

```
public class ExampleDocumentSetup implements IDocumentSetupParticipant {
  public static final String PARTITIONING= "org.eclipse.editor";
  public static final String EXPL= "org.eclipse.editor.expl";
  public static final String[] TYPES= new String[] {
     IDocument.DEFAULT_CONTENT_TYPE, EXPL};

  public void setup(IDocument document) {
    IDocumentPartitioner p;
    p= new DefaultPartitioner(createJavaPartitionScanner(), TYPES);
    document.setDocumentPartitioner(PARTITIONING, p);
    p.connect(document);
  }

  private IPartitionTokenScanner createJavaPartitionScanner() {
    RuleBasedPartitionScanner scanner= new RuleBasedPartitionScanner();
    scanner.setPredicateRules(new IPredicateRule[] {
      new PatternRule(">", "<", new Token(EXPL), '\\', false) });
    return scanner;
  }
}
```

Revisiting syntax coloring

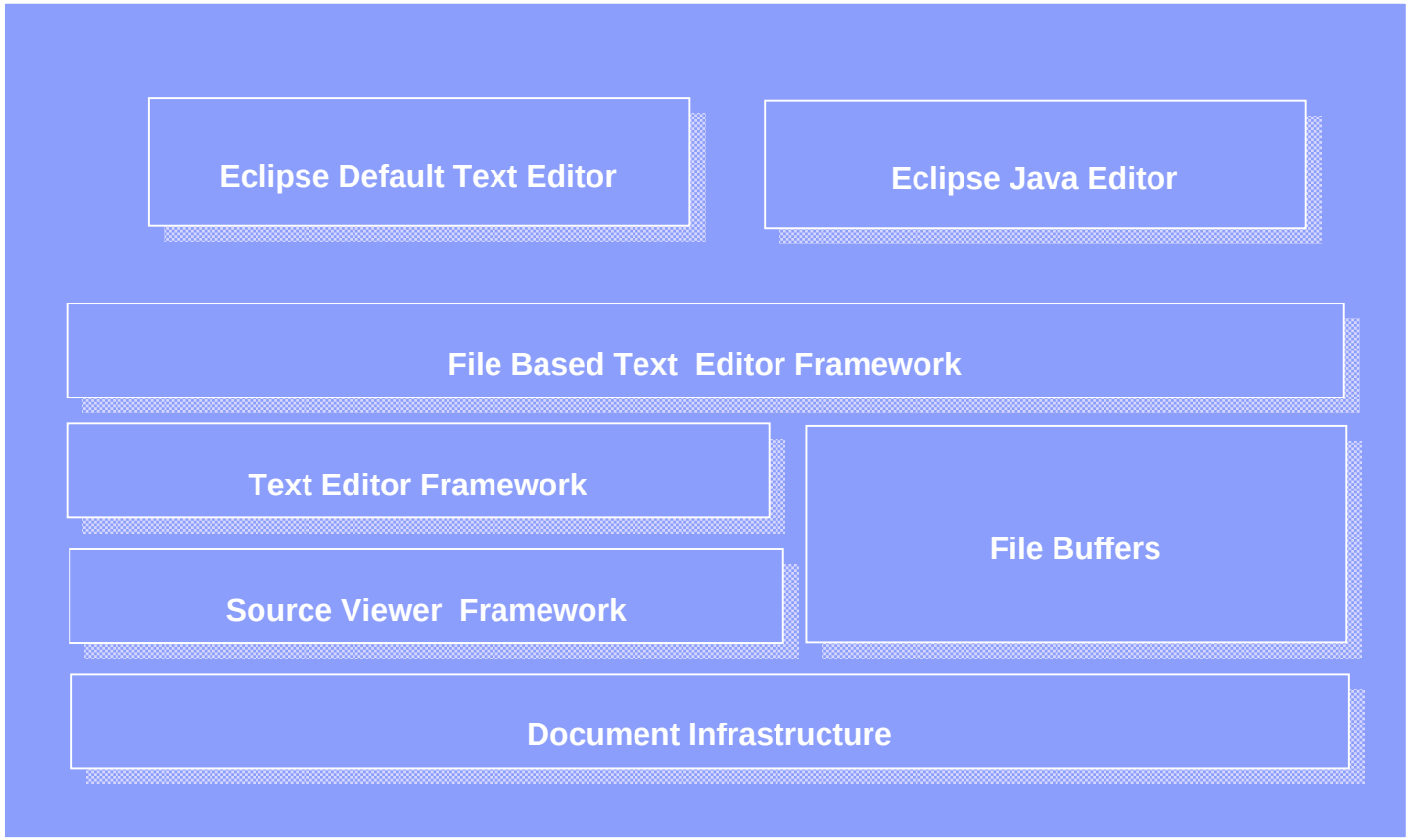
- Adapt `ExampleSourceViewerConfiguration` to available partitioning

```
IPresentationReconciler getPresentationReconciler(ISourceViewer viewer) {
  PresentationReconciler reconciler= new PresentationReconciler();
  reconciler.setDocumentPartitioning(ExampleDocumentSetup.PARTITIONING);

  DefaultDamagerRepairer dr;
  dr= new DefaultDamagerRepairer(createTokenScannerForEXPL());
  reconciler.setDamager(dr, ExampleDocumentSetup.EXPL);
  reconciler.setRepairer(dr, ExampleDocumentSetup.EXPL);

  dr= new DefaultDamagerRepairer(createTokenScannerForDefault());
  reconciler.setDamager(dr, IDocument.DEFAULT_CONTENT_TYPE);
  reconciler.setRepairer(dr, IDocument.DEFAULT_CONTENT_TYPE);
  return reconciler;
}
```

Default configuration scope

- `SourceViewerConfiguration`
  - vertical ruler annotation hover, auto indent strategy, content assist, content formatter, double clicking, prefixing, shifting, information presenter, overview ruler annotation hover, presentation reconicler, model reconciler, text hover, undo manager
- `Abstract/ExtendedTextEditor`
  - menu ids, key binding scopes, preference stores, source viewer configuration, annotation presentation, vertical ruler columns, change ruler column, line number ruler, overview ruler
- the set of `protected` methods

Overview of all architectural layers



Outlook

- will appear in the default configuration scope
  - linked positions
  - templates
  - folding
- see poster "Editor centric Workbench" for general direction