

10 - Integrated Development Environments

Motivation

Writing a toy application with a text editor is comparatively easy.

Developing a large software system with just a text editor is much more difficult if not impossible.

Tools available to help you with the job:

- Support in managing code
- Support for carrying out boring (but error-prone) tasks
- Integration in one central workplace to improve (team) efficiency

Use of object-orientation is essential here, because it helps structuring the system.

Integrated Development Environment: IDE

Central tool of the implementation phase

Supports almost all aspects of the implementation process.

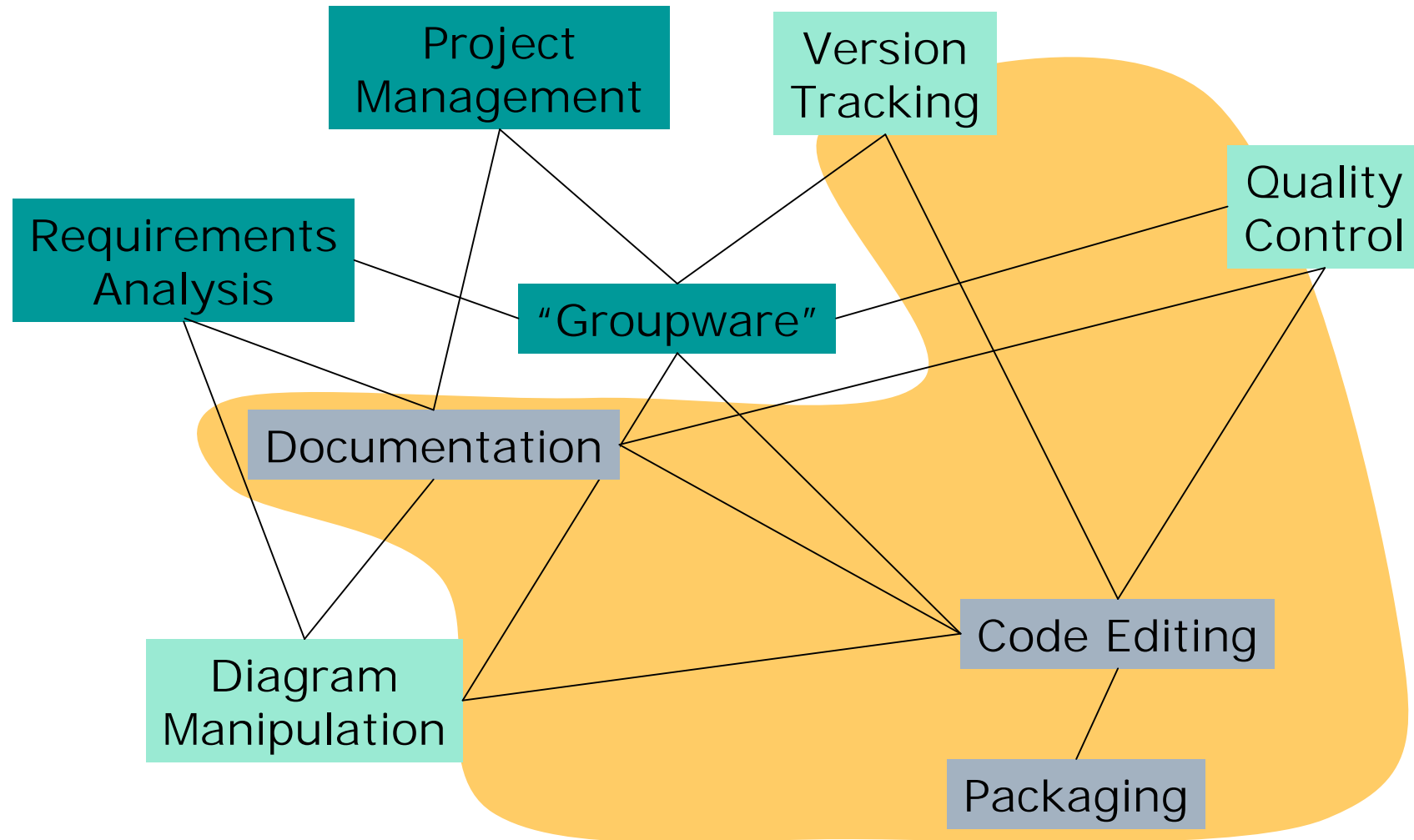
An IDE integrates a set of tools useful for software development in a single piece of software:

Your working environment.

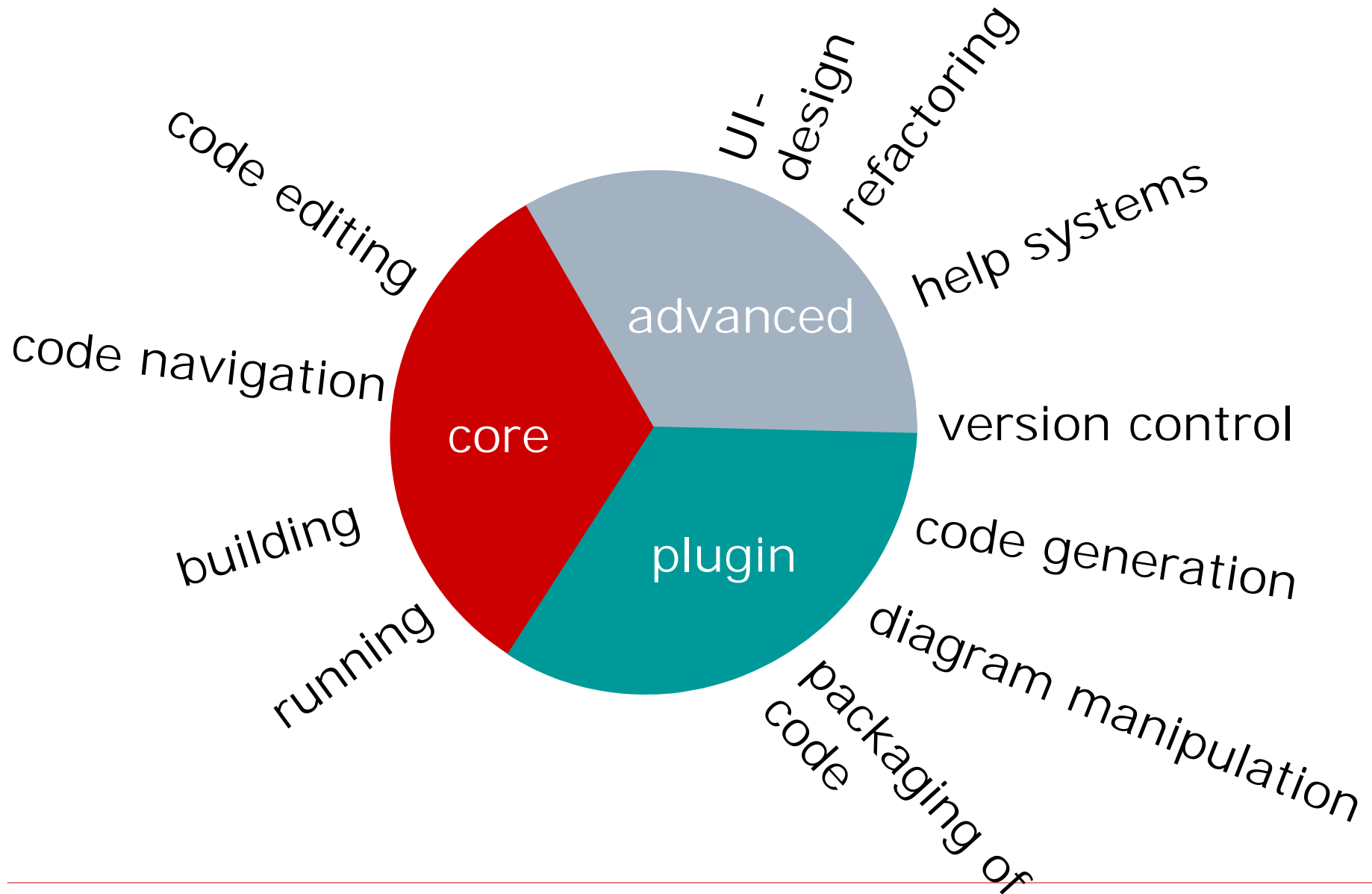
Short edit/compile/debug roundtrip:

- no change of tools
- no format conversions

Code Centric Tool: Integrated Development Environment (IDE)



IDE Functionality



Purpose

An IDE is much more than a text editor, it eases manipulation of code through ...

- syntax highlighting
- error highlighting
- documentation access
- code navigation
- code generation through code templates
- support for refactoring
- different levels of code analysis.

Basic Functionality

- Code manipulation:
Basically text editor functionality with additional completion features (suggestion of possible completions for the current “word”; syntax-based editors)
- Highlighting:
Color codify the language’s syntax to make it easier to read
- Navigation:
Offer rich ways to move through code (not only by file structure, but also by, e.g., inheritance hierarchy)

Basic Idea

Code is more than just plain text.

Like a compiler, a development environment can “understand” the structure of the code.

Therefore, (source) code can be computed.

Easy example (older than IDEs):
pretty printing; reformat code ...

- for printing (80, 120, ... columns)
- for a changed code style guide
- from inexperienced programmers
- from source code generators

Pretty Printing

The screenshot shows the Eclipse IDE interface with the 'Format' menu open. The menu items and their shortcuts are:

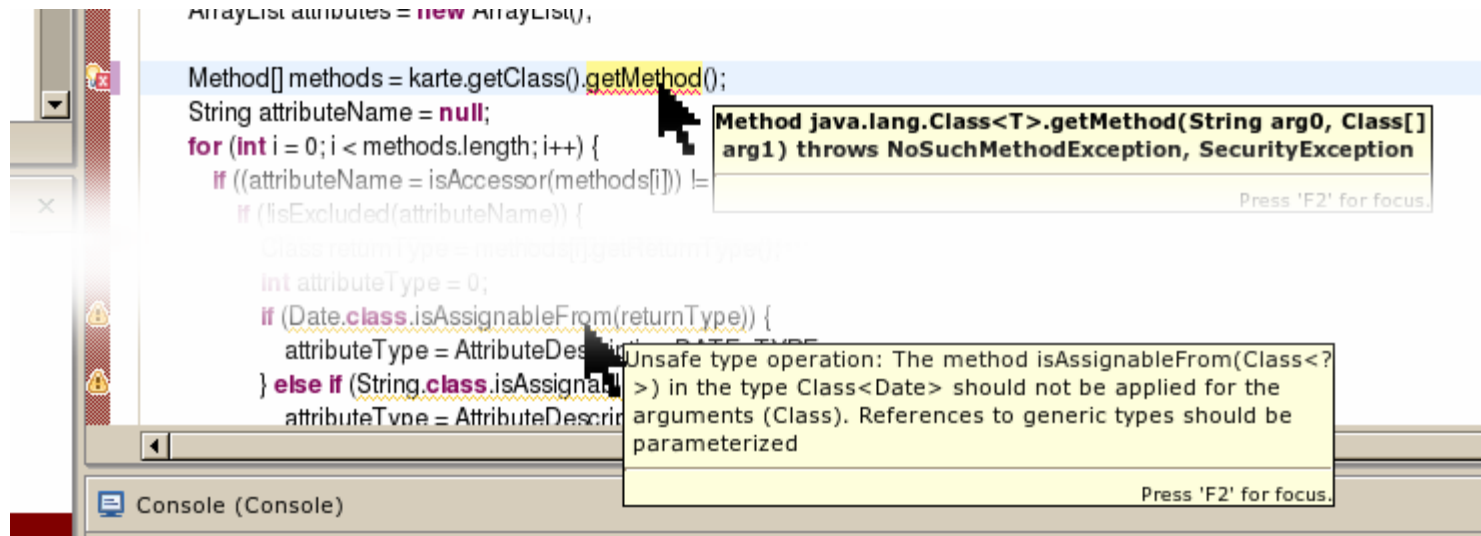
- Toggle Comment (Ctrl+/,)
- Add Block Comment (Ctrl+Shift+/,)
- Remove Block Comment (Ctrl+Shift+\)
- Shift Right
- Shift Left
- Format (Ctrl+Shift+F)
- Format Element
- Correct Indentation (Ctrl+I)
- Sort Members

The code editor displays the following Java code for the `FormatMe` class:

```
public class FormatMe {  
    protected String idontlikethecodelayout;  
  
    int areNotNice;  
  
    FormatMe () {  
        idontlikethecodelayout = "really" ;  
    } // constructor  
  
    int onelineFunctions () {  
        return areNotNice ;  
    }  
  
    static final String CONSTANT = "static fields not at end!" ;  
  
    static void really (FormatMe fm) {  
        fm.idontlikethecodelayout += ", really" ;  
    }  
}  
}  
// class FormatMe
```

A red box highlights the formatted code, and a red arrow points from the 'Format' menu item to the code.

Highlighting



Highlighting of ...

- language syntax
- errors (red)
- warnings (yellow)

Context-sensitive display of information

Code Navigation

Navigating through code as if it were hyperlinked documents

Different access paths:

- package structure
- inheritance hierarchy
- definition-use dependencies
- call graphs

Bridges the gap between

- code writing and
- code execution

Good navigational support is particularly important if you work with other people's code.

Type Hierarchy and Navigation

The screenshot shows the Eclipse IDE interface. On the left, the 'Hierarchy' view displays a tree of classes. The 'MuMeDokument' class is selected, and its superclasses are listed: Dokument, Fund, Object, AbstractFund, AbstractAsset, AbstractMutableFund, AbstractFund, AbstractAsset, AbstractMutableAsset, AbstractAsset, Fund, AbstractFund, AbstractAsset, and Asset. Below this, the 'MuMeDokument' class is expanded to show its members: LOG, muMeDokumentPersistor, muMeDokumentDocument, xmlBean, and several constructors. On the right, the 'MuMeDokument.java' editor shows the source code. A red circle highlights the `private static Logger LOG = Logg` line. A red arrow points from this line to the 'Logger' class in the 'Hierarchy' view. Another red arrow points from the 'Logger' class in the 'Hierarchy' view to the `private static Logger LOG = Logg` line in the code editor. A mouse cursor is positioned over the `Logger` text in the code editor.

Class hierarchy

Super types

Clicking "Logger" opens its implementation (hold Ctrl in Eclipse)

Advanced Features

Templates:

pre-stored fragments of code, saves typing

Documentation access:

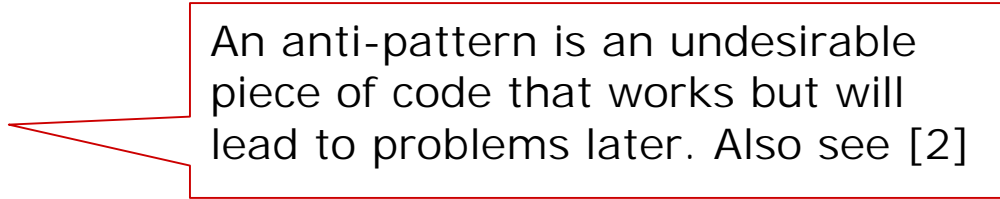
Find the right piece of documentation depending on context

Refactoring:

restructuring patterns for code that can be automatically carried out by the IDE

Code analysis:

track down common anti-patterns



An anti-pattern is an undesirable piece of code that works but will lead to problems later. Also see [2]

Templates

Code templates are pieces of code with holes

Templates cover common expressions

- o small scale examples:
 - control-flow constructs like
 - n for-loops
 - n if-expression
- o large scale examples:
 - n entire classes
 - n new files
 - n new methods

Templates Example

template for
switch block

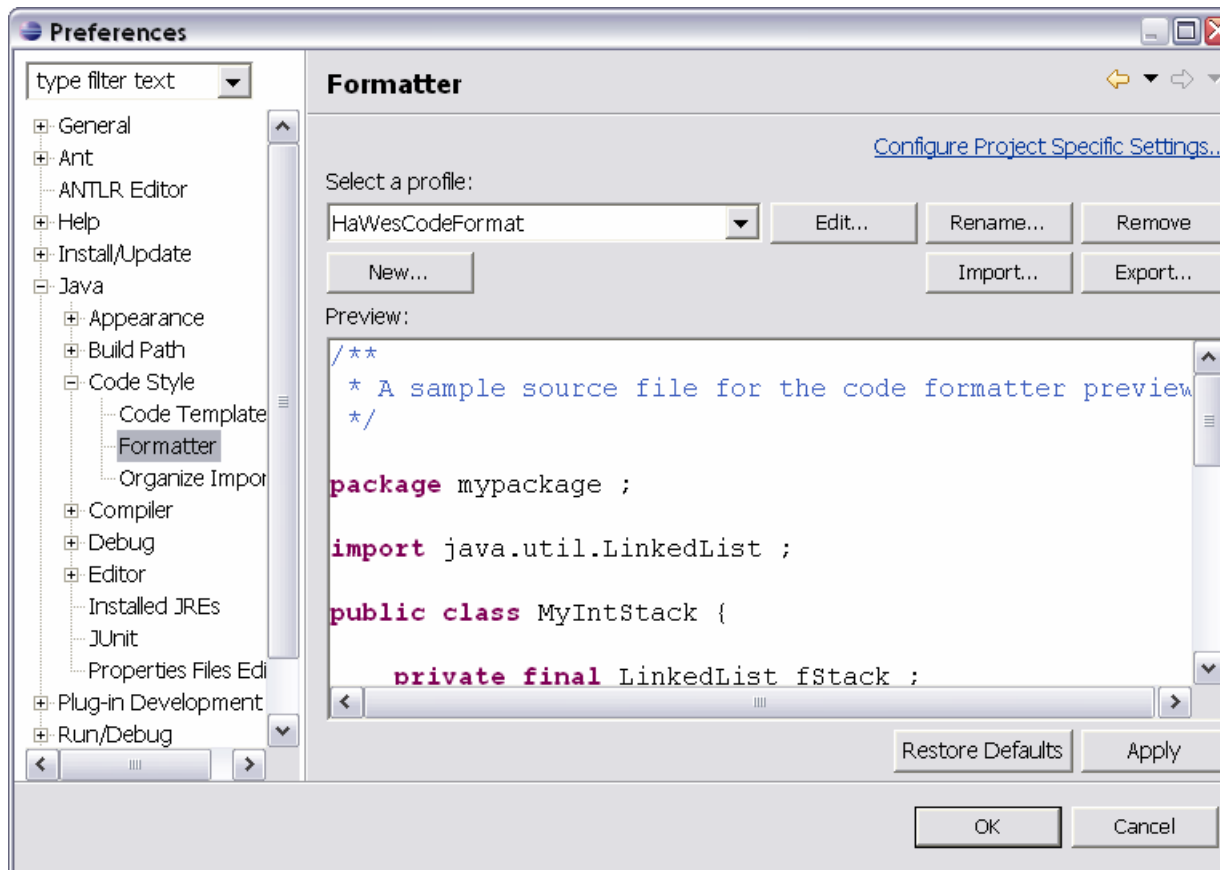
The screenshot shows the Eclipse IDE Preferences dialog, specifically the Java section and the Templates tab. The 'switchs' template is selected in the table, and its code is shown in the 'Code' field below. A red box highlights the code, and a callout points to it from the text 'template for switch block'.

Name	Description
sbpr	SessionBean provider/resolver met...
sbpr2	SessionBean provider/resolver met...
sleep	try / thread.sleep
sup	add call to super method
switchd	switch statement (with default)
switchs	switch statement

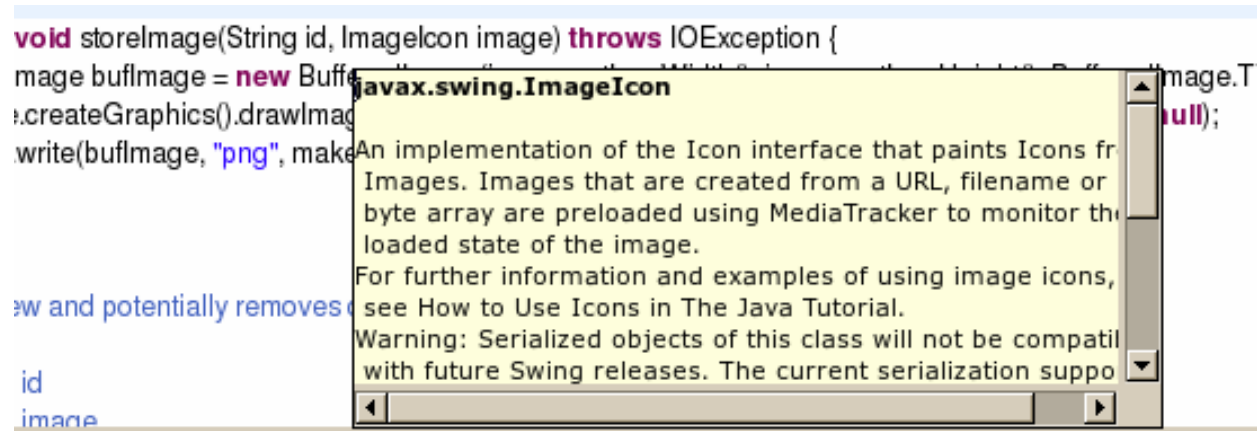
```
switch () {  
  case :  
    break;  
  case :  
    break;  
}
```

Customizable Templates

Templates and code format can be defined or changed by users.



Documentation Access



Various forms of context sensitive access to the language's documentation:

- Tooltips
- Jumping to the right position in the online documentation
- Navigating to comments

Also see documentation tools (covered later).

Documentation Generation

Document code by, e.g., source code comments.

IDEs support documentation language, e.g., Java doclets.

JavaDoc, HTML as
sublanguage of Java

code completion,
navigation, etc. work

```
/**
 * Generate a <code>SELECT...FROM</code> statement without where clause for
 * every class.
 * @param classes      an array containing all classes of the current model
 * @param queriedCls   the base class for which to query (root of all classes queried)
 * @param subQueriedCls the actual class for which to add an entry in this invocation
 * @param sqlSymTab    the {@link SQLSymbolTable} to be used
 * @return a table storing the prefix
 */
static private HashMap<AssetClass, String> prefix(
    AssetClass [] classes,
    AssetClass queriedCls,
    AssetClass subQueriedCls,
    SQLSymbolTable sqlSymTab)
{
    HashMap<AssetClass, String> prefix = new HashMap<>();
    String prefix = "SELECT ' " + subQueriedCls.getName() + "' AS " + subQueriedCls.getName() + " FROM " + queriedCls.getName() + " WHERE " + subQueriedCls.getName() + " IS NOT NULL";
}
```

```
public class SQLSymbolTable
    extends SymbolTable
{
    static public final int NO_TYPE = -1;

    private HashMap<AssetClass, String> tables;
    private HashMap<Member, String> columnNames;
    private HashMap<Member, Integer> columnTypes;
    private HashMap<Member, Integer> columnTypeParameters;
    private HashMap<Member, String> lobPrimaryKeys;
}
```

Refactoring

Refactoring means improving the design of existing code without changing its external behavior.

- done incrementally during implementation
- demand driven (i.e., you refactor when the existing solution is not sufficient any more)

Many refactoring patterns have been identified.

- Applying the patterns usually involves much typing and a mass of boring changes (error-prone process!)
- Patterns are supported (carried out semi-automatically) by some IDEs.

Refactoring Patterns

Some refactorings:

- Extracting Method: extract a new method from a single piece of spaghetti-code:

```
double computeAvg(int[] values) {  
    int sum = 0;  
    for (int i = 0; i < values.length; i++)  
        sum += values[i];  
    double avg = sum / values.length;  
    return avg;  
}
```

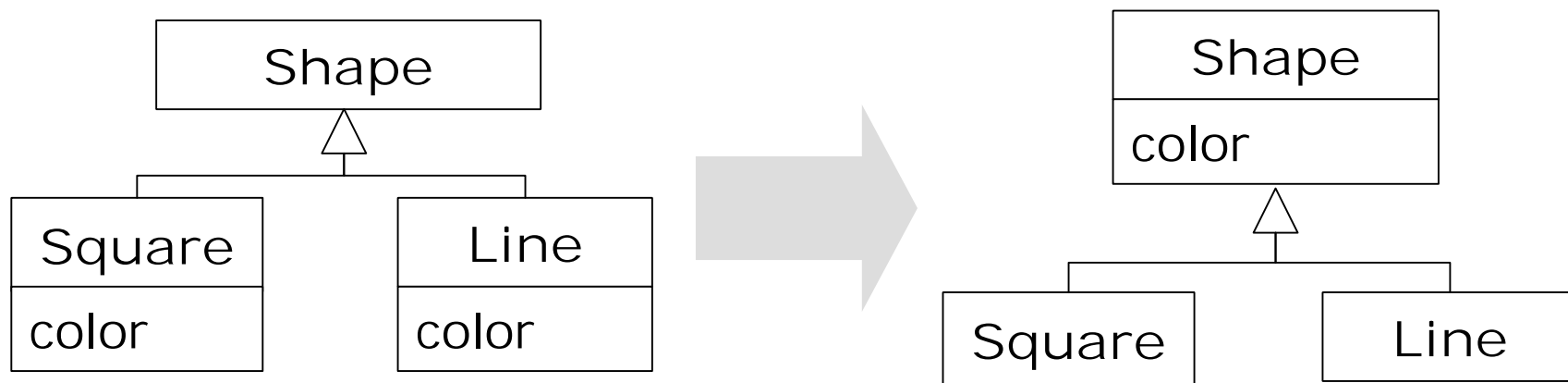
```
int computeSum(int[] values) {  
    int sum = 0;  
    for (int i = 0; i < values.length; i++)  
        sum += values[i];  
    return sum;  
}  
double computeAvg(int[] values) {  
    return computeSum(values) /  
        values.length;  
}
```

- Encapsulate Field: create accessor and mutator methods (getXYZ and setXYZ) for a field. Replace accesses to the field with accesses to the methods.

Refactoring Patterns (2)

Some more refactorings:

- Pull up/Push down: moving data members or methods up or down the inheritance hierarchy



- Extract interface: use part of a class definition to create an interface. Let the class implement the interface and use the interface instead of the class wherever possible.

Code Analysis

Code analysis tools examine source code to find code with unwanted properties. These do not necessarily need to be errors.

Common things an analyzer tries to find:

- difficult to understand code (e.g., overly long classes)
- code not following conventions (e.g., language-specific capitalizations)
- code not covered by unit-tests
- patterns that are likely to lead to problems later (e.g., redundancies, wrong implementations of `clone()` or `finalize()` in Java)

`clone()` method creates an exact copy of an object.

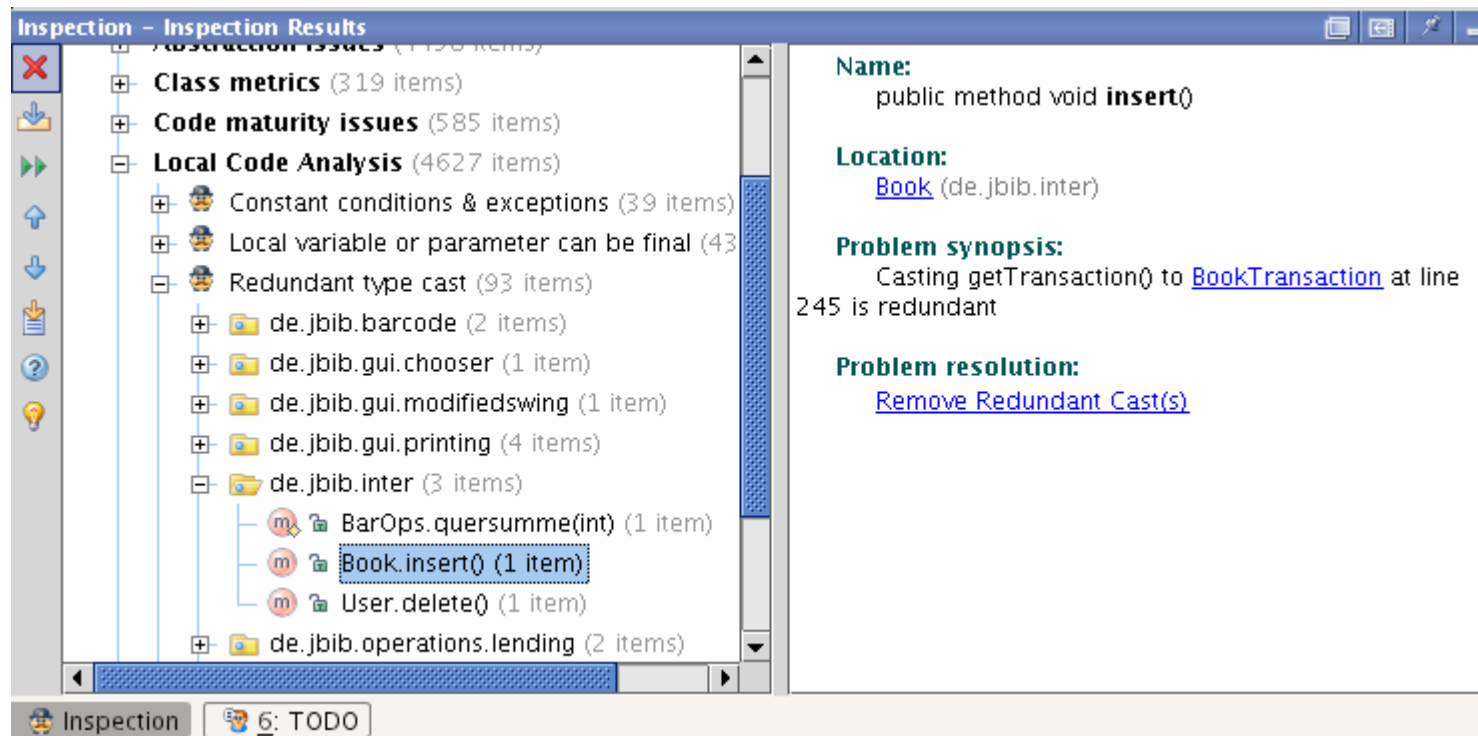
`finalize()` is called when an object is garbage-collected. A common error is to supply a wrong signature resulting in the supplied `finalize` never being called.

Code Analysis (2)

Code analysis cannot find semantic errors.

- Computers don't know the developer's intention.
- You can implement a method to do something entirely stupid and neither compiler nor analyzer will find it.

Code Analysis (3)



UI Design

It is desirable that developers can design user interface dialogs graphically and an IDE generates appropriate code.

- This code is usually in the same language as your application but should not be edited.

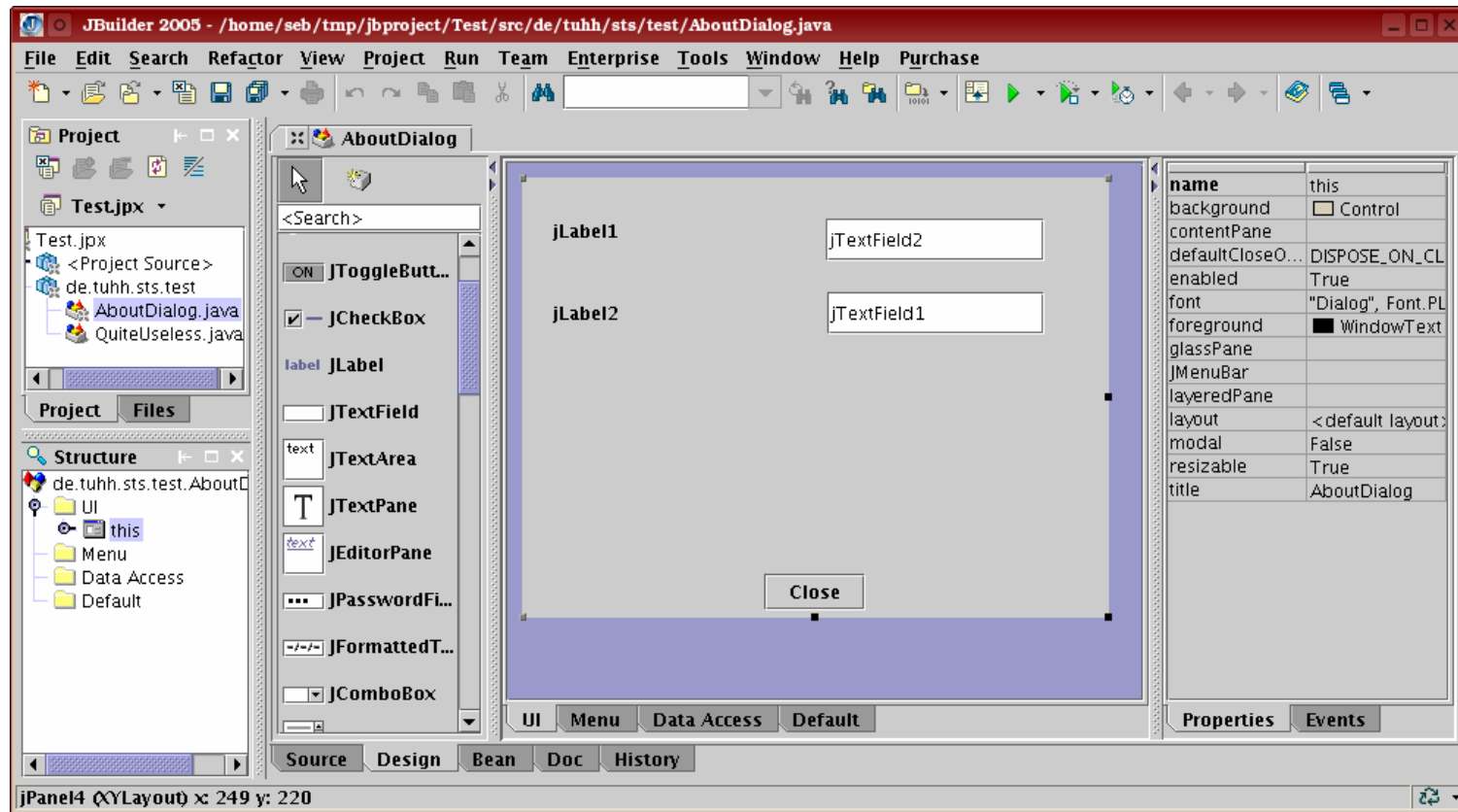
Advantages:

- Easier to build visually pleasing dialogs
- Faster to get first results

Disadvantages:

- Generated code might not fit your favorite design pattern (e.g., MVC) or other side conditions
- Can become confusing with very complex or non-standard dialogs

UI-Design



Often useful to quickly prototype dialogs. Complicated dialogs may actually be quicker to do by writing code.

Plugins

A plugin is a dependent piece of software that extends the host application.

Plugins can be tightly integrated with their host application.

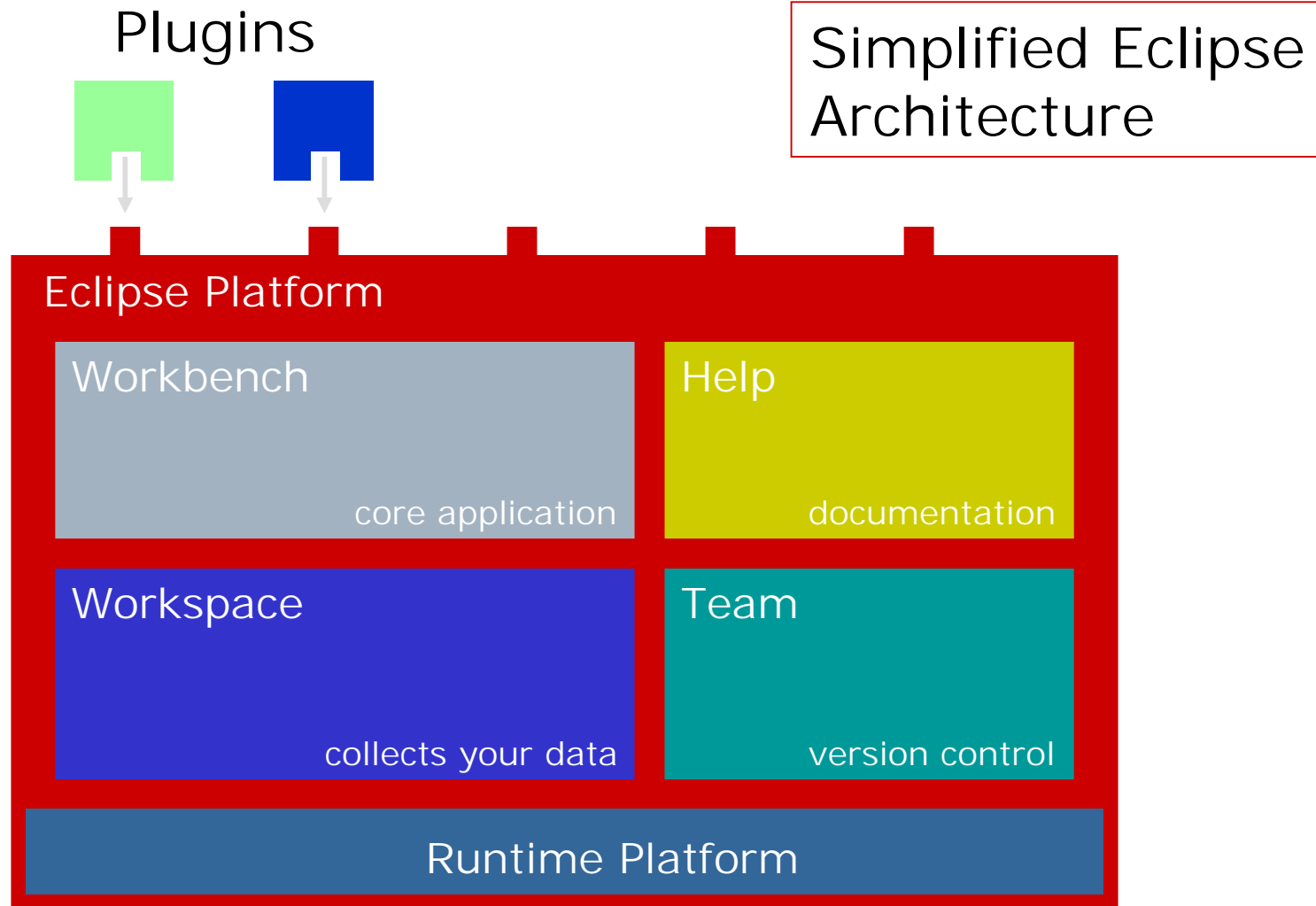
Host application provides ...

- hook points that allow plugins to extend it in a framework-like manner, and
- loading and discovery mechanisms that loads the plugins at runtime.

Other vendors can extend the IDE to integrate support for their technology.

The concept of plugins is not specific to IDEs.

Plugin Architecture



Plugin Architecture (2)

Modern plugin-based systems implement much of their own functionality as plugins.

- Only core functionality is provided in the system itself.
- This allows to treat own functionality just the same as external plugins.

Dependencies:

- All plugins use the core platform functionality.
- Plugins can also depend on other plugins.
- Usually, plugins try to be as self-reliant as possible to make installation easy.

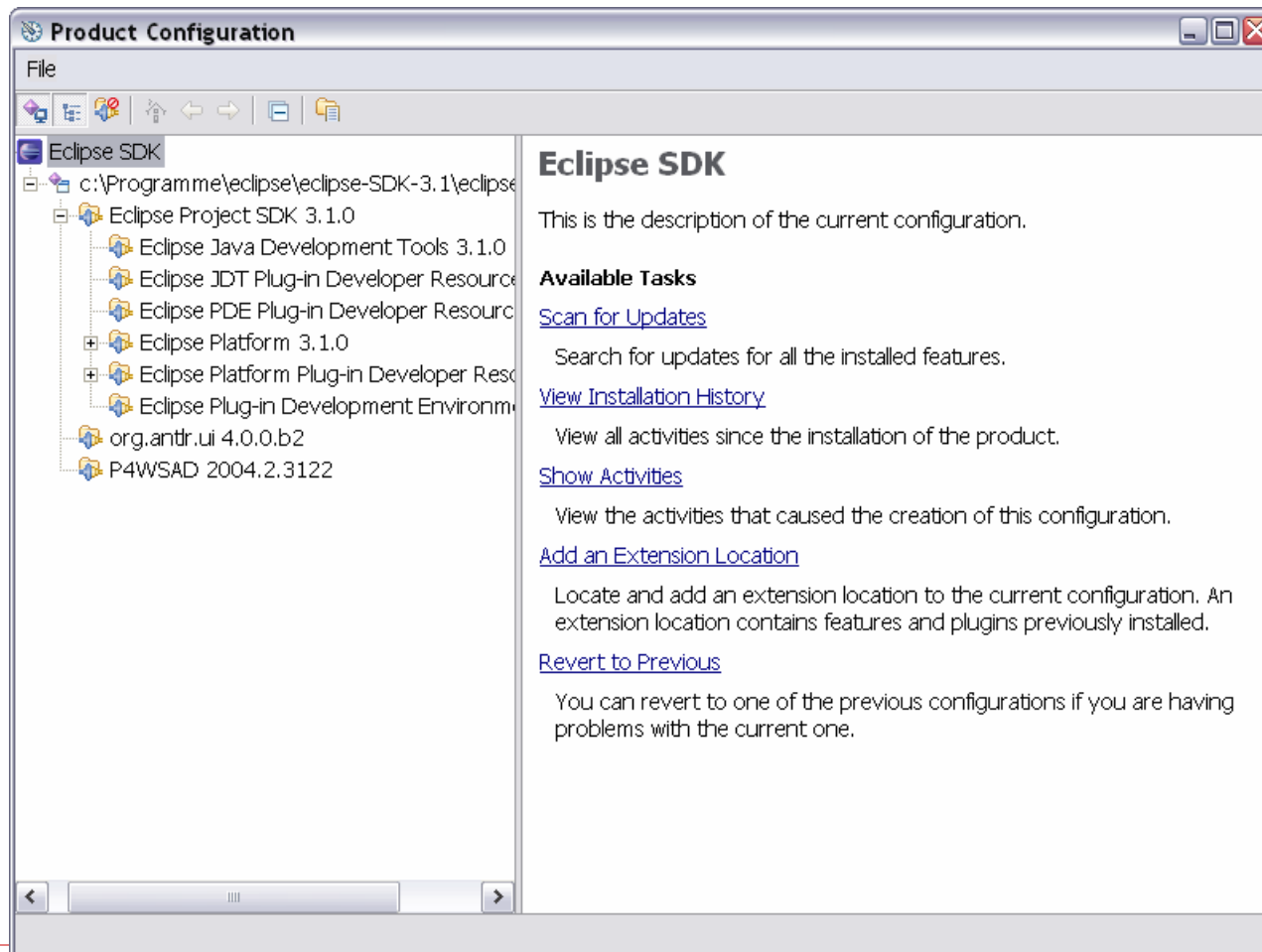
Plugins for IDEs

Typical plugin functionality for IDEs includes:

- Connection to repository systems.
- Support for frameworks (code generation, graphical editors for configuration files), e.g., Struts, EJB, ...
- New languages (compiler, highlighting, running, ...), e.g., C++, XML, ...
- In eclipse: plugin for plugin development.

Plugin Management in eclipse

Download and update of plugins via HTTP.
Management of plugin versions etc.



Overview of Tools

The following slides select a small collection of Java IDEs.

The selection is...

- o OO-based
- o STS-biased
- o alphabetically ordered
- o incomplete
- o ...

Overview of Java IDEs

Eclipse

- o Free, opensource
- o Extensible via plugins (many available)
- o Well-documented
- o “Tooling platform”, language support (e.g., for Java) and advanced features (e.g. J2EE) not in core IDE
- o Basic refactorings
- o www.eclipse.org

Intelli-J IDEA

- o Commercial (company: JetBrains)
- o Extensive refactoring support
- o Code analysis (watching for anti-patterns)
- o UI designer
- o www.jetbrains.com/idea

Overview of Java IDEs (2)

JBuilder

- o Commercial (Borland), Free (limited) version available
- o UI-designer
- o Basic refactorings
- o Enterprise version has web service and EJB support
- o Plugins
- o www.borland.com/jbuilder

JCreator

- o Commercial (Xinox)
- o Basic IDE
- o Fast (native Windows code), Windows only
- o www.jcreator.com

Overview of Java IDEs (3)

Netbeans

- o Free, opensource
- o J2EE support
- o Not many refactorings
- o www.netbeans.org

Sun Studio

- o Commercial (Sun Microsystems)
- o Based on Netbeans
- o www.sun.com/software/products/studio

Websphere Application Developer

- o Commercial (IBM)
- o Based on Eclipse
- o www.ibm.com/software/awdtools/studioappdev

Overview of Java IDEs (4)

Together/J

- Commercial (Borland)
- Combines UML modeling and Java programming; code follows model changes
- Supports patterns (e.g., those named by the “Gang of Four”)
- Support for JavaBeans and J2EE.
- <http://www.borland.com/together>

References

- [1] Eric Allen. Bug Patterns in Java. APress, 2002
- [2] W.H. Brown et al. Anti Patterns. John Wiley & Sons, 1998