

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.

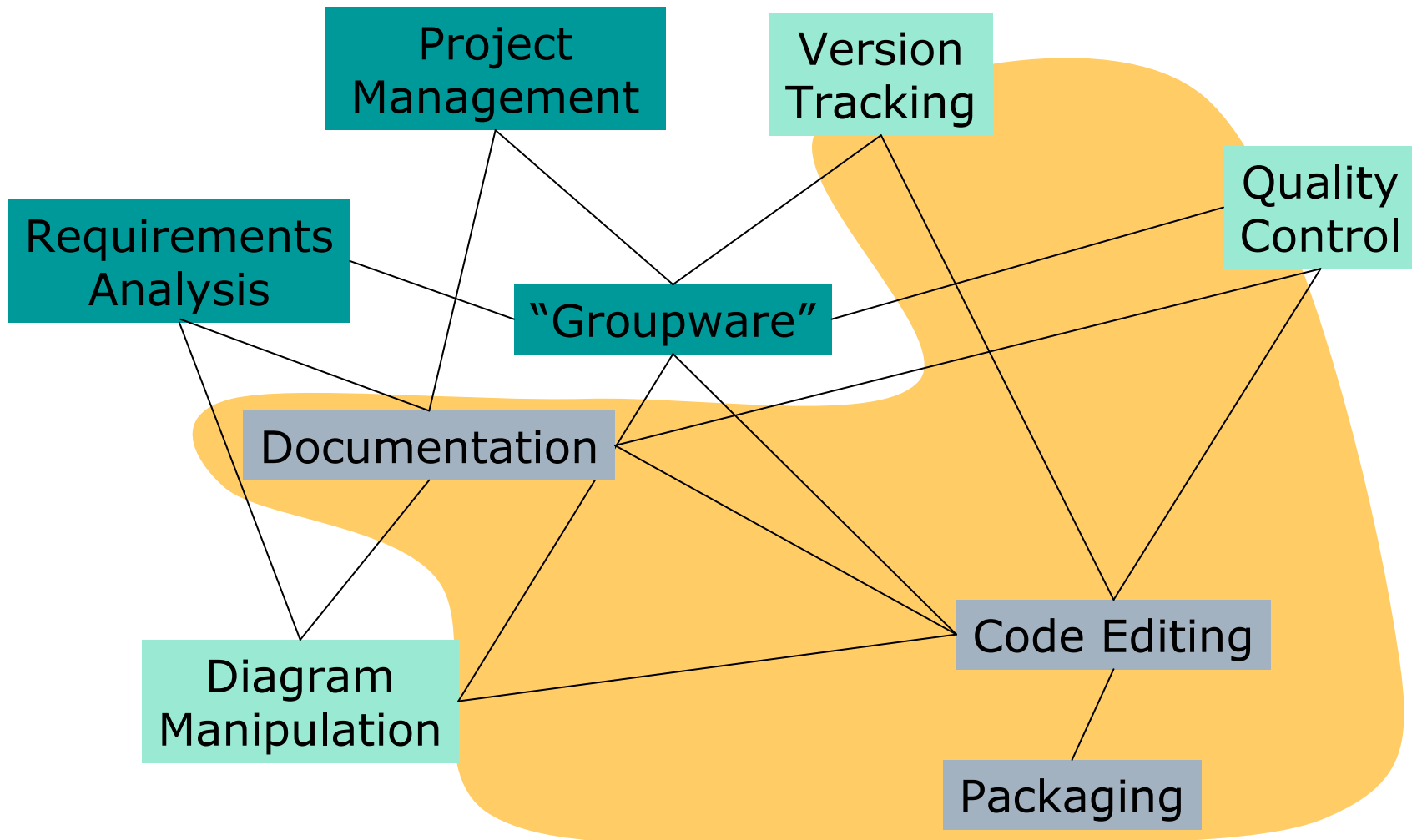
An IDE *integrates* a set of tools useful for software *development* in a single piece of software:

Your working *environment*.

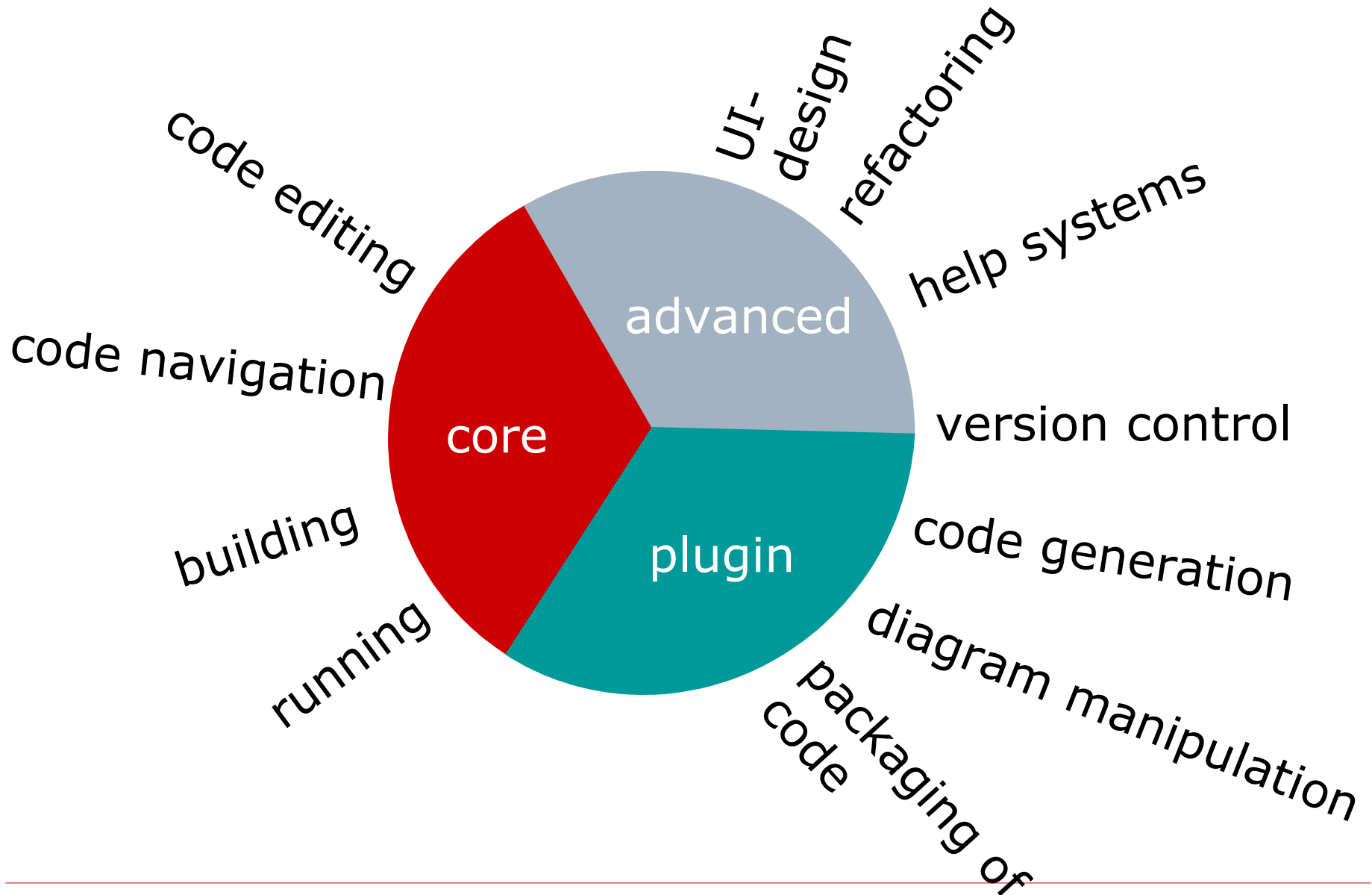
Short edit/compile/debug *roundtrip*:

- 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:

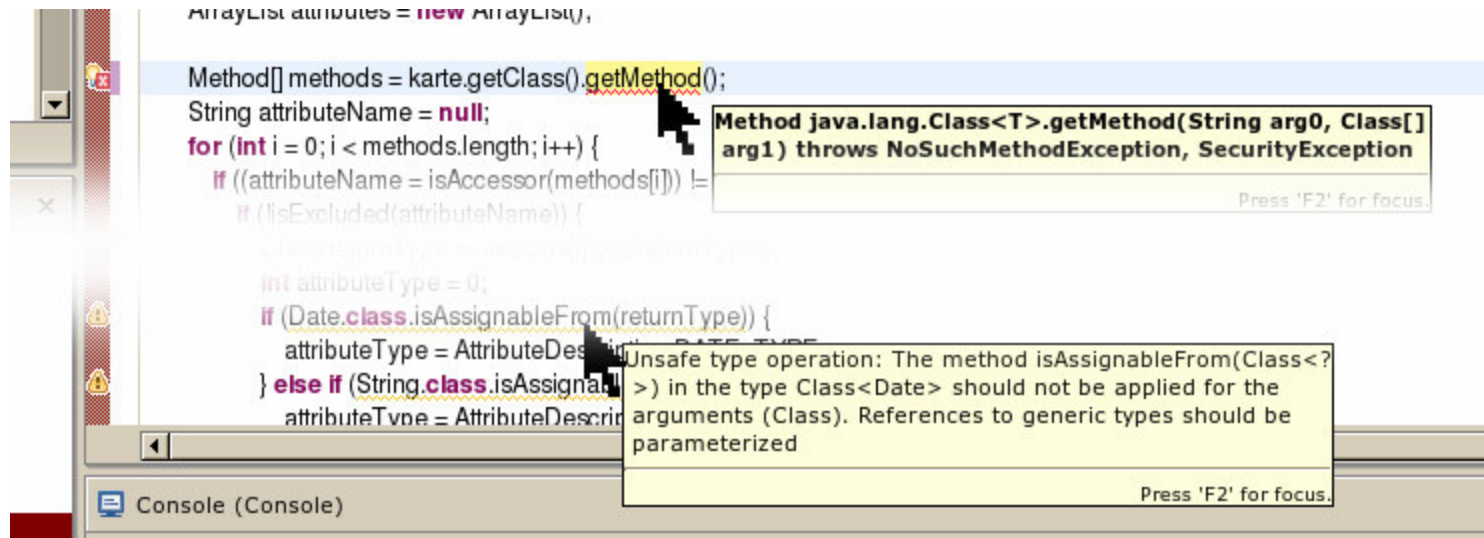
- Toggle Comment: Ctrl+/
- Add Block Comment: Ctrl+Shift+//
- Remove Block Comment: Ctrl+Shift+\
- Shift Right
- Shift Left
- Format: Ctrl+Shift+F
- Format Element
- Correct Indentation: Ctrl+I
- Sort Members
- Organize Imports

The code editor displays the following Java code, which is the result of applying the 'Format' action:

```
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
```

Red arrows indicate the transformation from the unformatted code on the right to the formatted code on the left.

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

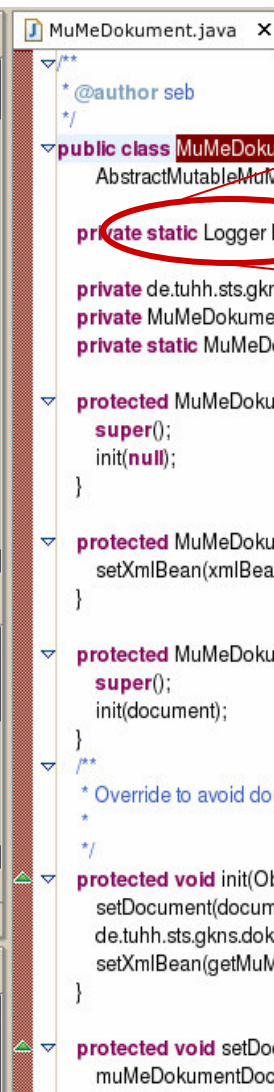
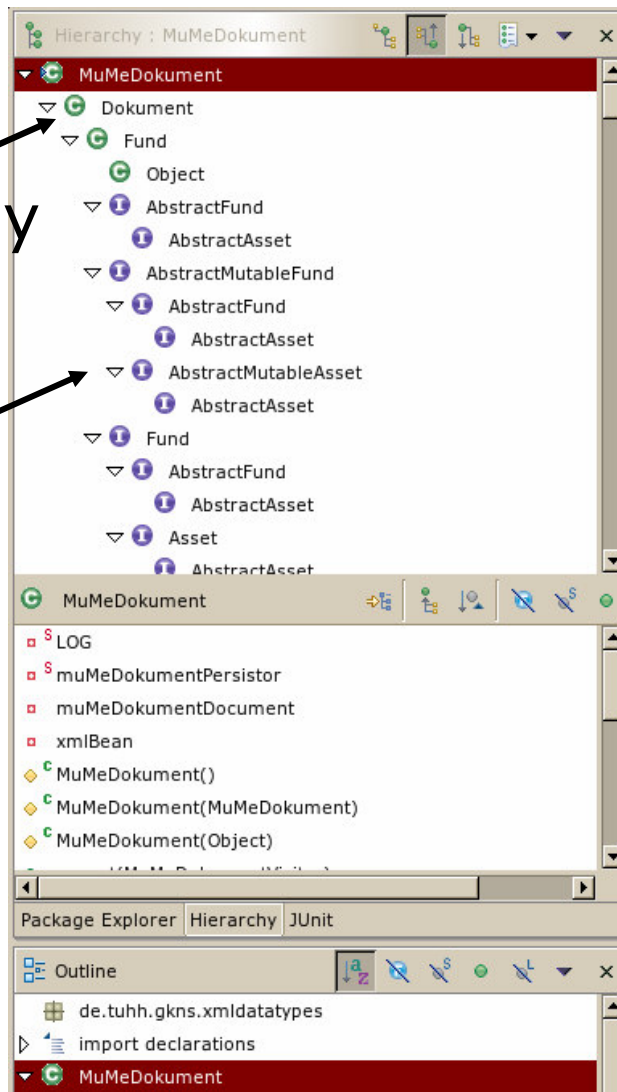
- ❑ code writing and
- ❑ code execution

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

Type Hierarchy and Navigation

Class hierarchy

Super types



```
public class MuMeDokument extends  
    AbstractMutableMuMeDokument  
  
    private static Logger LOG = Logg  
  
    private de.tuhh.sts.gkns.  
    private MuMeDokumen  
    private MuMeDokumentDocument
```

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

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

Templates Example

template for
switch block

The screenshot shows the Eclipse IDE Preferences dialog, specifically the Java Templates section. The left sidebar shows the 'Templates' folder expanded under 'Java'. The main area displays a table of templates and a code editor below it.

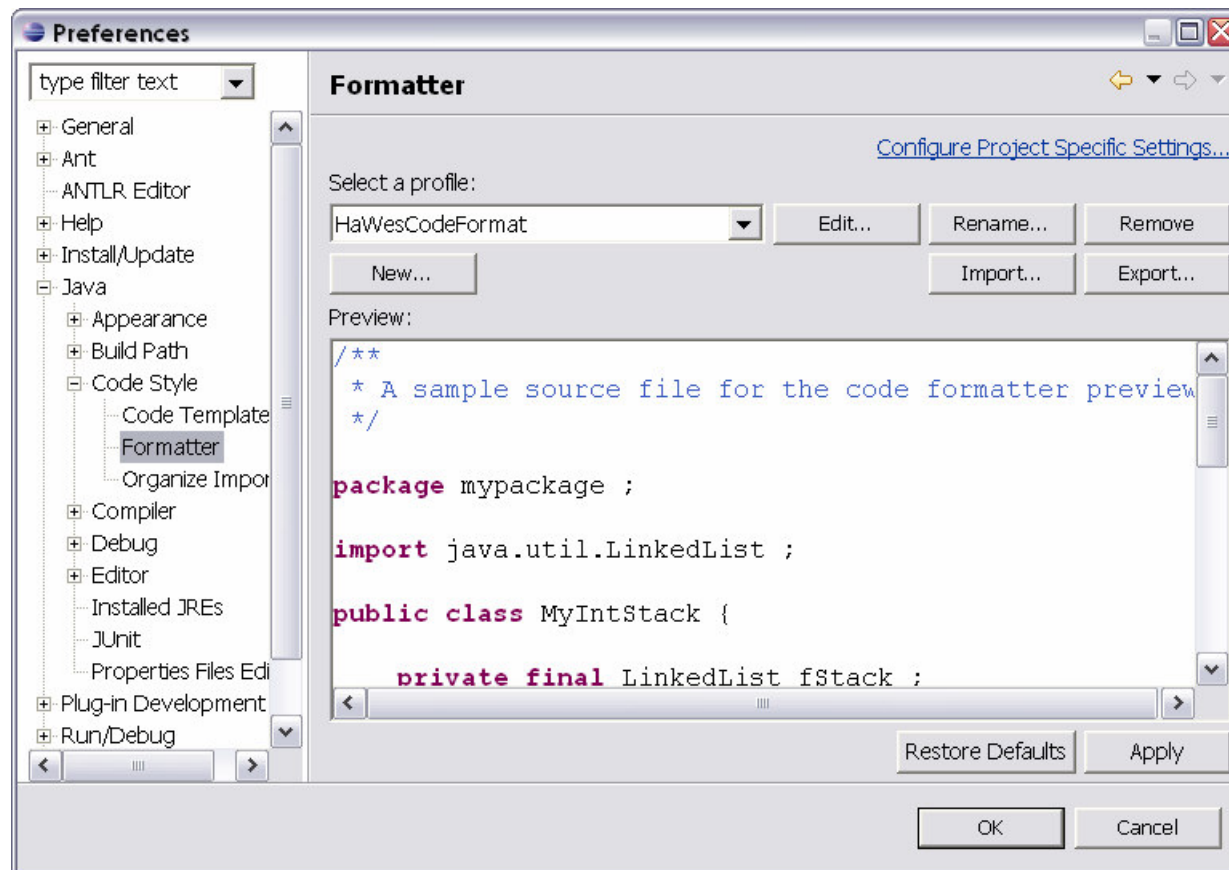
Name	Description
sbpr	SessionBean provider/resolver met...
sbpr2	SessionBean provider/resolver met...
sleep	try / thread.sleep
sup	add call to super method
switchd	switch statement (with default)
switchs	switch statement

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

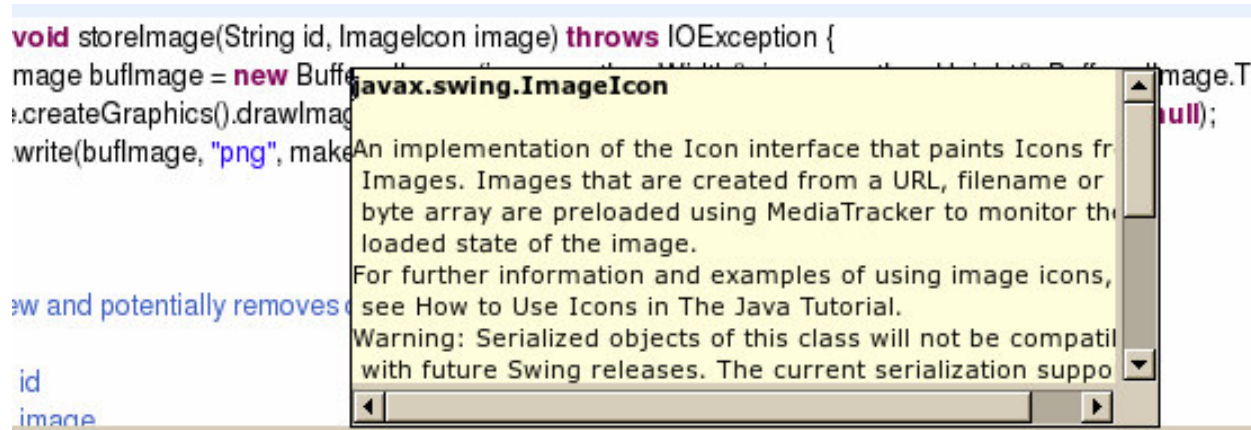
Buttons at the bottom: Add..., Edit..., Delete, Restore Original, Macro...

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> prefixMap = new HashMap<>();
    String prefix = "SELECT '" + subQueriedCls.getName() + "' FROM '" + queriedCls.getName() + "'";
}
```

```
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;  
}
```

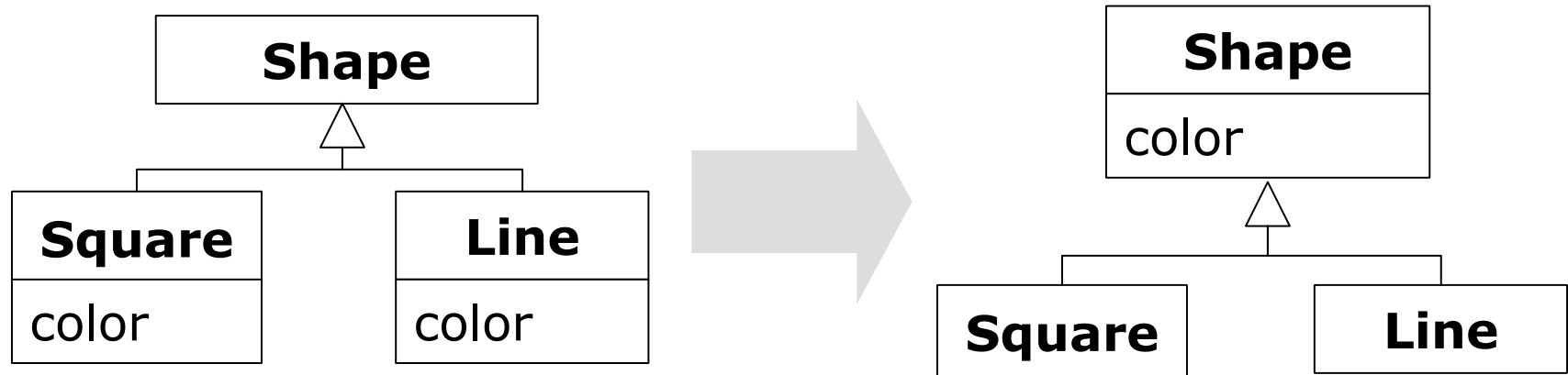
```
int computeSum(int[] values) {  
    int sum = 0;  
    for (int i = 0; i < values.length; i++)  
        sum += values[i];  
    return sum;  
}  
double computeAvg(int[] values) {  
    return computeSum(values) /  
        values.length;  
}
```

- ❑ 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)

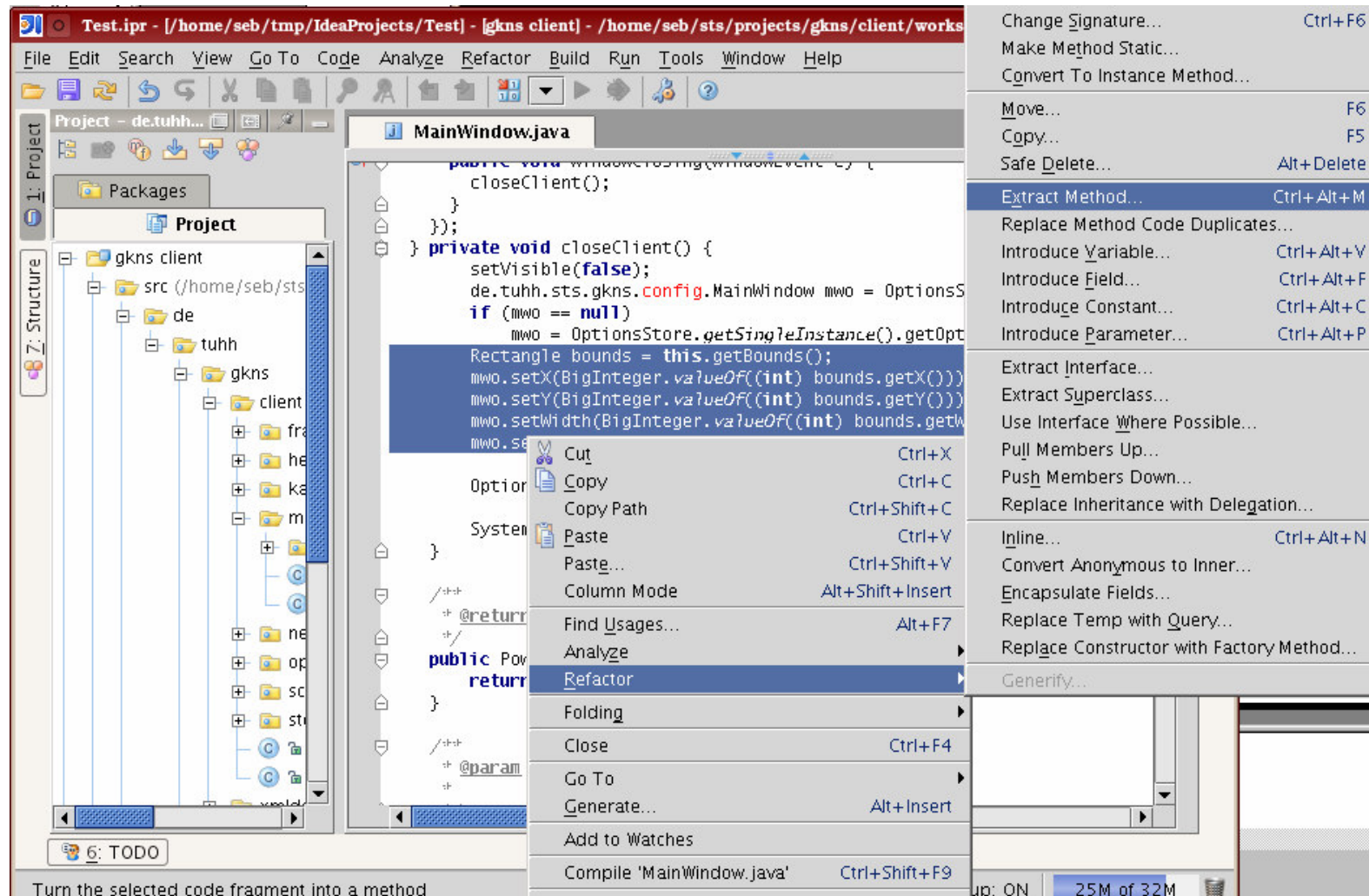
Some more refactorings:

- ❑ 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.

Example: Extracting Method



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)

The screenshot displays the 'Inspection - Inspection Results' window in an IDE. The left pane shows a tree view of inspection categories and their counts:

- Abstraction Issues (1150 items)
- Class metrics (319 items)
- Code maturity issues (585 items)
- Local Code Analysis (4627 items)
 - Constant conditions & exceptions (39 items)
 - Local variable or parameter can be final (43 items)
 - Redundant type cast (93 items)
 - de.jbib.barcode (2 items)
 - de.jbib.gui.chooser (1 item)
 - de.jbib.gui.modifiedswing (1 item)
 - de.jbib.gui.printing (4 items)
 - de.jbib.inter (3 items)
 - BarOps. quersumme(int) (1 item)
 - Book.insert() (1 item)**
 - User.delete() (1 item)
 - de.jbib.operations.lending (2 items)

The right pane provides details for the selected issue:

- Name:** public method void **insert()**
- Location:** [Book](#) (de.jbib.inter)
- Problem synopsis:** Casting getTransaction() to [BookTransaction](#) at line 245 is redundant
- Problem resolution:** [Remove Redundant Cast\(s\)](#)

At the bottom of the window, there are two tabs: 'Inspection' and '6: TODO'.

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.

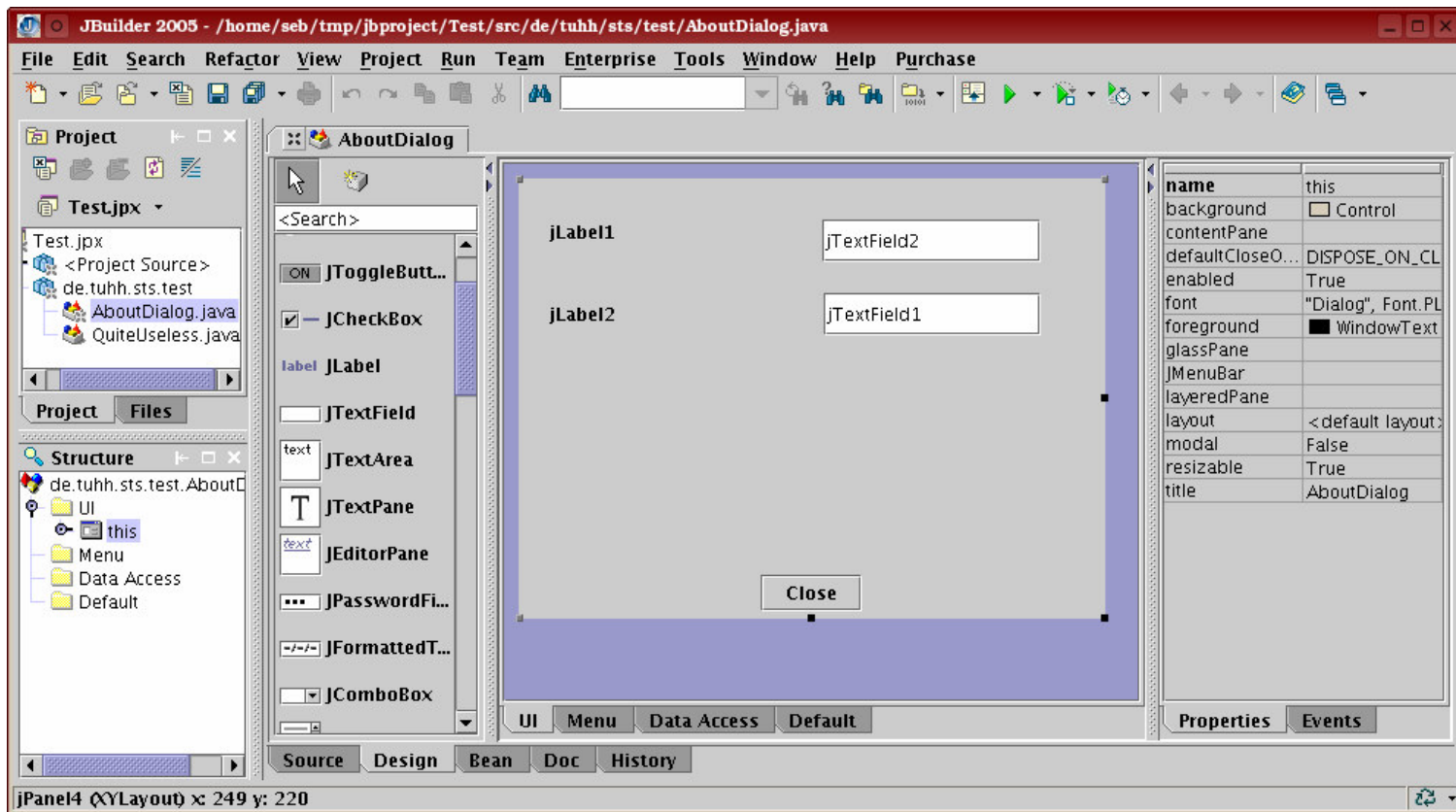
Advantages:

- ❑ 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.

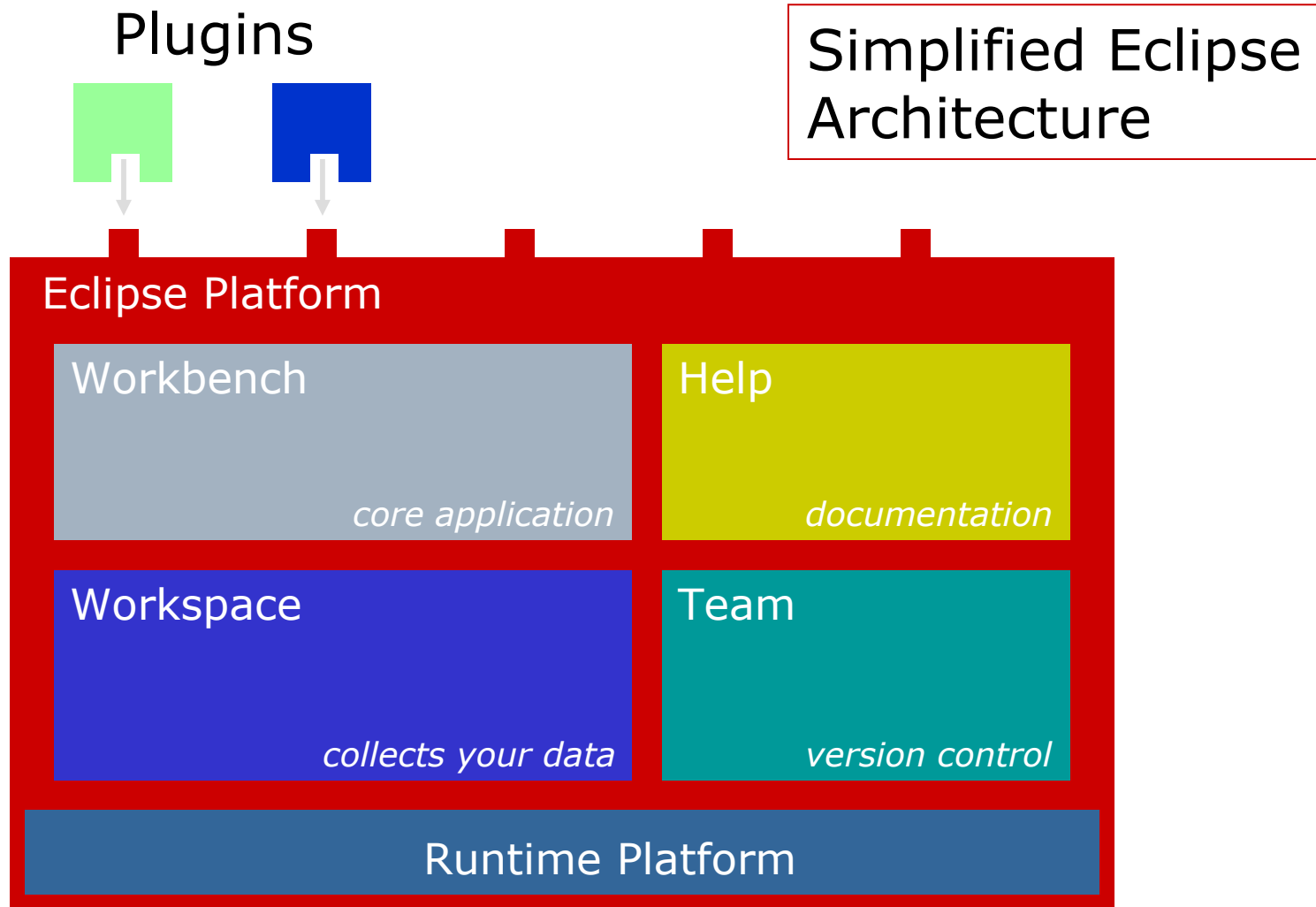
Host application provides ...

- 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.

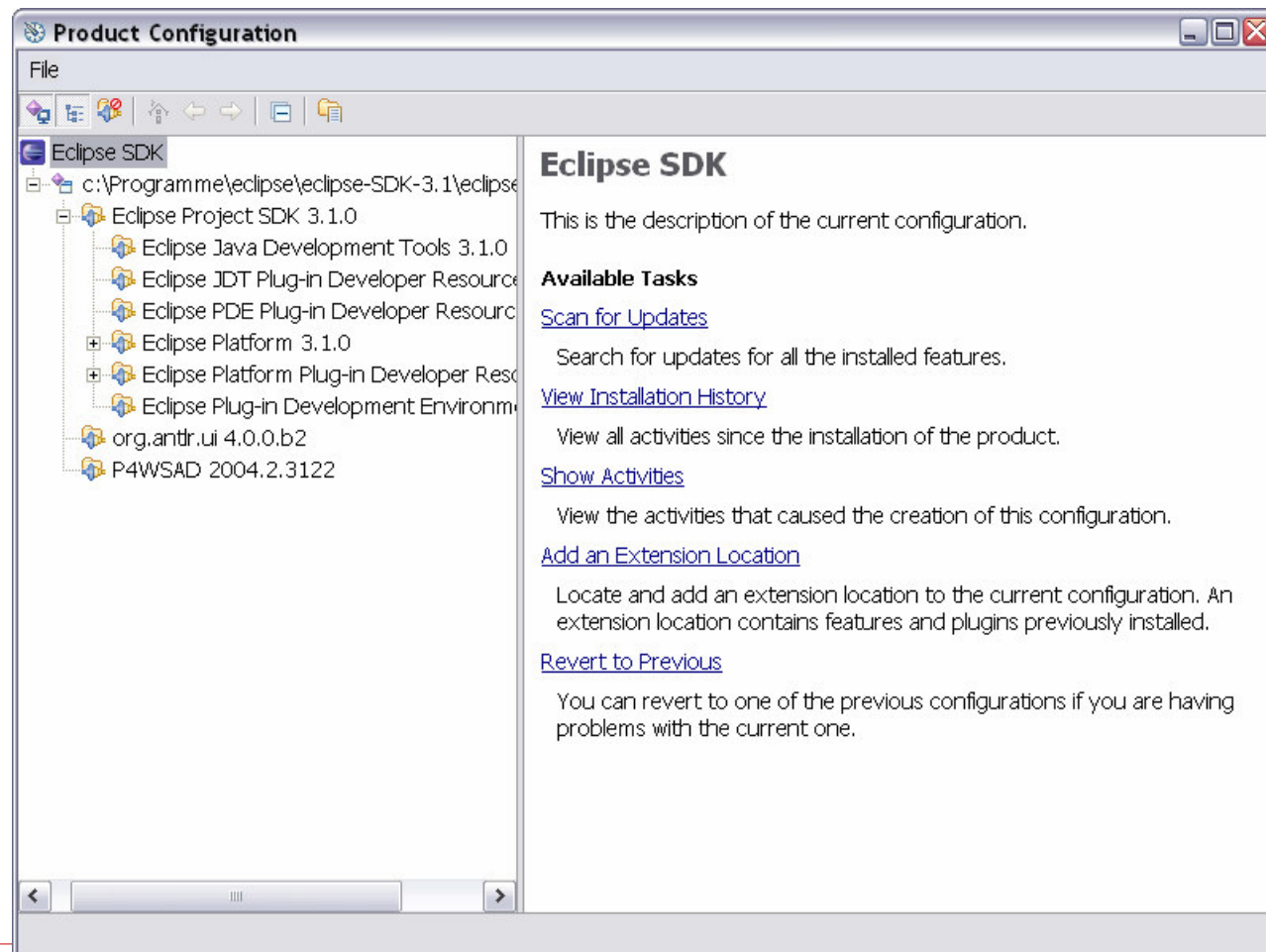
Plugins for IDEs

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...

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

Overview of Java IDEs

Eclipse

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

Intelli-J IDEA

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

Overview of Java IDEs (2)

JBuilder

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

JCreator

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

Overview of Java IDEs (3)

Netbeans

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

Sun Studio

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

Websphere Application Developer

- Commercial (IBM)
- Based on Eclipse
- 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