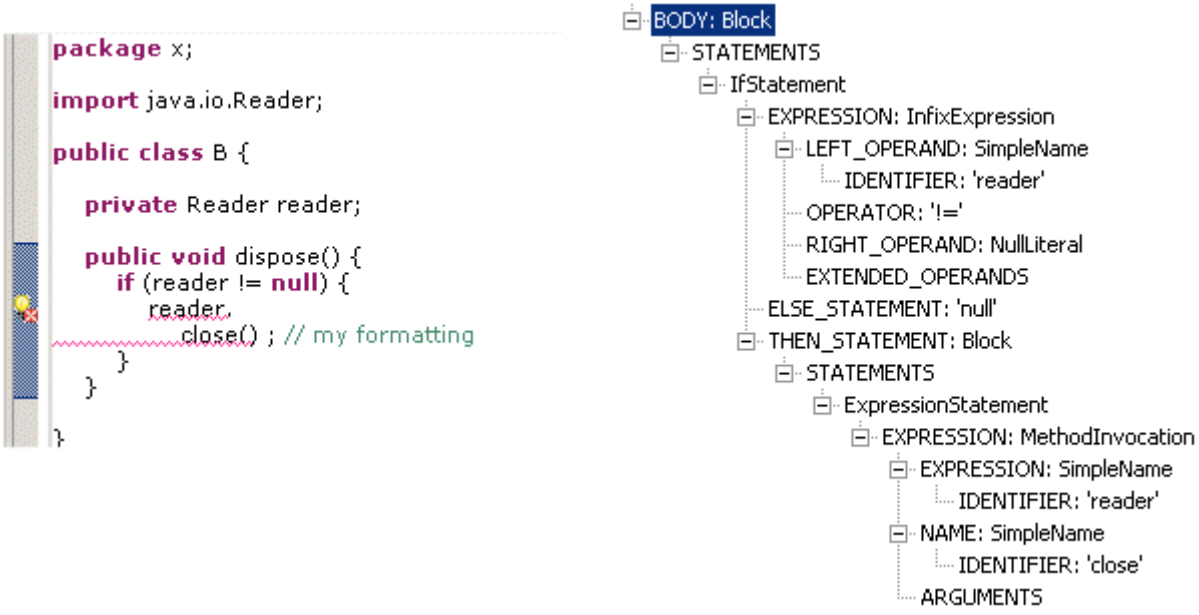
Infrastructure for Java code manipulations: AST Rewriter

Martin Aeschlimann, IBM OTI Labs Zurich
David Audel, IBM Paris Lab St. Nazaire

AST = Abstract Syntax Tree

Intermediate representation of a Java file created by the compiler



AST Rewrite

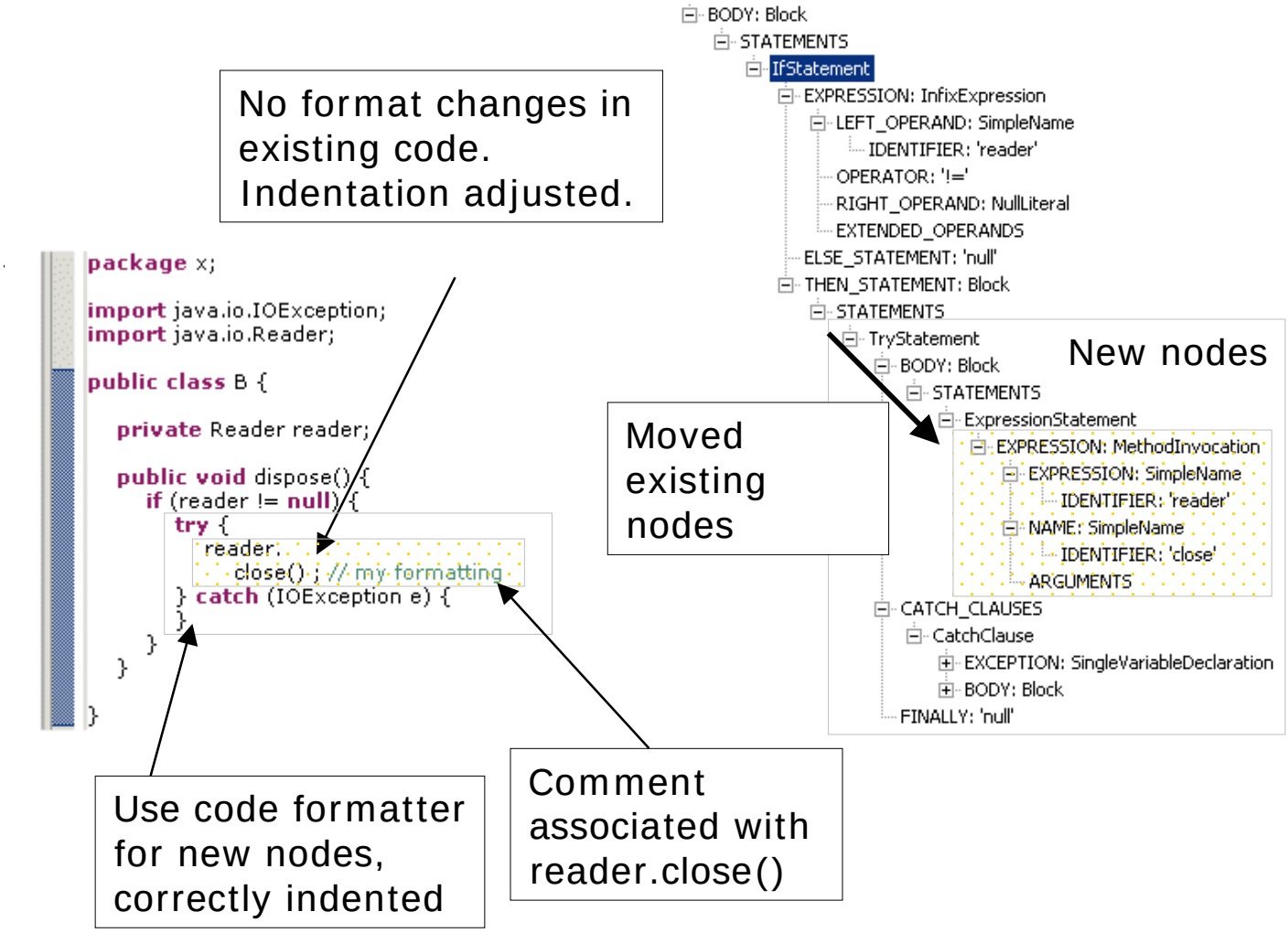
To programmatically change code, don't modify the text, but express the changes by manipulating the AST.

- Replace, remove and insert nodes and node properties
- Move and copy nodes in the same AST

AST Rewriter characteristics:

- Minimal text changes: Always preserve user written code with probably hand tuned formatting styles.
- Introduce new code in correct indentation and formatting
- Deal with marker positions on the underlying document
- Takes care of comments in the code. e.g. which comments to remove when a node is deleted.
- Track node ranges before/after the rewrite

Example: Surround statement with try / catch block:



Two flavors of the AST rewrite API:

Descriptive API

changes are only described, the AST stays unmodified.

+ allows to share an AST: AST are expensive to build and loose bindings when modified
+ create several modification proposals for the same AST, present previews but apply only one modification. (Use case: Quick fix)

Modifying API

the setter methods on the AST nodes are used

+ Easy to use, operate directly on nodes
+ AST is always up to date

Code example for the descriptive API:

```
public void surroundWithTry(ICompilationUnit cu, Statement statement)
    AST ast= statement.getAST();

    DescriptiveRewrite rewrite= new DescriptiveRewrite(ast);

    ASTNode copyTarget= rewrite.createCopyTarget(statement);

    TryStatement newTry= ast.newTryStatement();
    newTry.getBody().statements().add(copyTarget);

    rewrite.replace(statement, newTry);

    // ...configure the catch clause of the try statement...

    IDocument document= new Document(cu.getSource());

    TextEdit resultingTextEdit= rewrite.rewriteAST(document);

    resultingTextEdit.apply(document);
}
```

Input is the statement to nest

Create Rewriter instance

Create a copy of the statement to be used at the copy target

Create new try statement and nest the target node

Mark the original as replaced with the new try statement

Evaluate rewriter: get text changes corresponding to the declared node changes

Apply the text changes to the document

Problem: How to describe a change on a property that is not a node (e.g. a primitive type or a list)?

Solution: Introduction of AST Node properties to allow a homogenous access to the AST

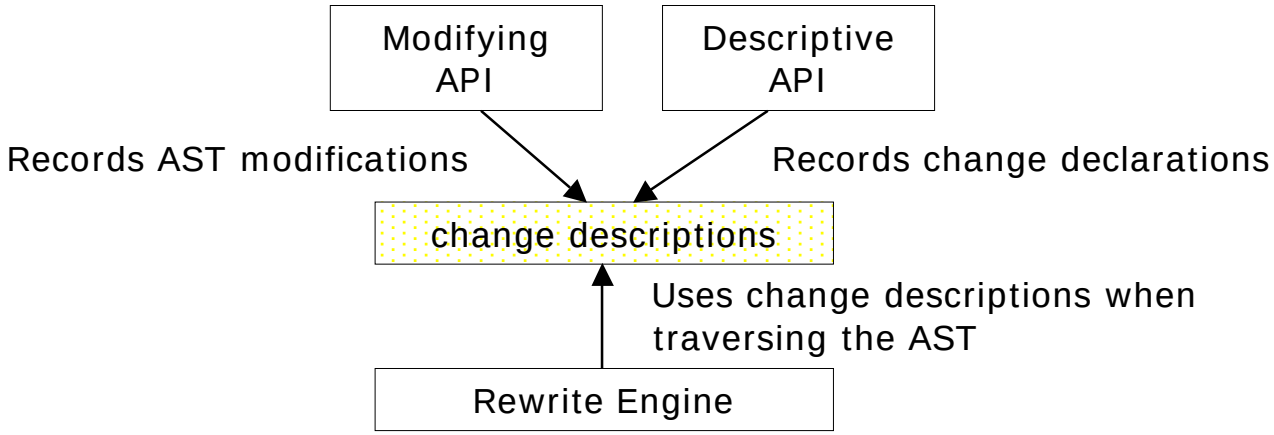
```
rewrite.set(node, MethodDeclaration.MODIFIER, 0);
rewrite.set(node, MethodDeclaration.IS_CONSTRUCTOR, false);

rewrite.getListRewrite(node, MethodDeclaration.PARAMETERS)
    .insertLast(newStatement);
```

AST returns edit scripts (text edits)

- Describes changes in term of inserts, removes, replacements..
- Edit scripts can be applied to a document copy to present a preview

Implementation

Modifying rewrite and descriptive rewrite both use the same rewrite engine



Status

Used by quick fix and refactoring.

Will become jdt.core API in 3.0