

# JDT meets the Tiger

**Dirk Bäumer**

IBM Research  
Zurich, Switzerland  
JDT/UI lead

**Philippe Mulet**

IBM Rational Software  
AIM Paris Laboratory, Saint-Nazaire, France  
JDT/Core lead

# Outline

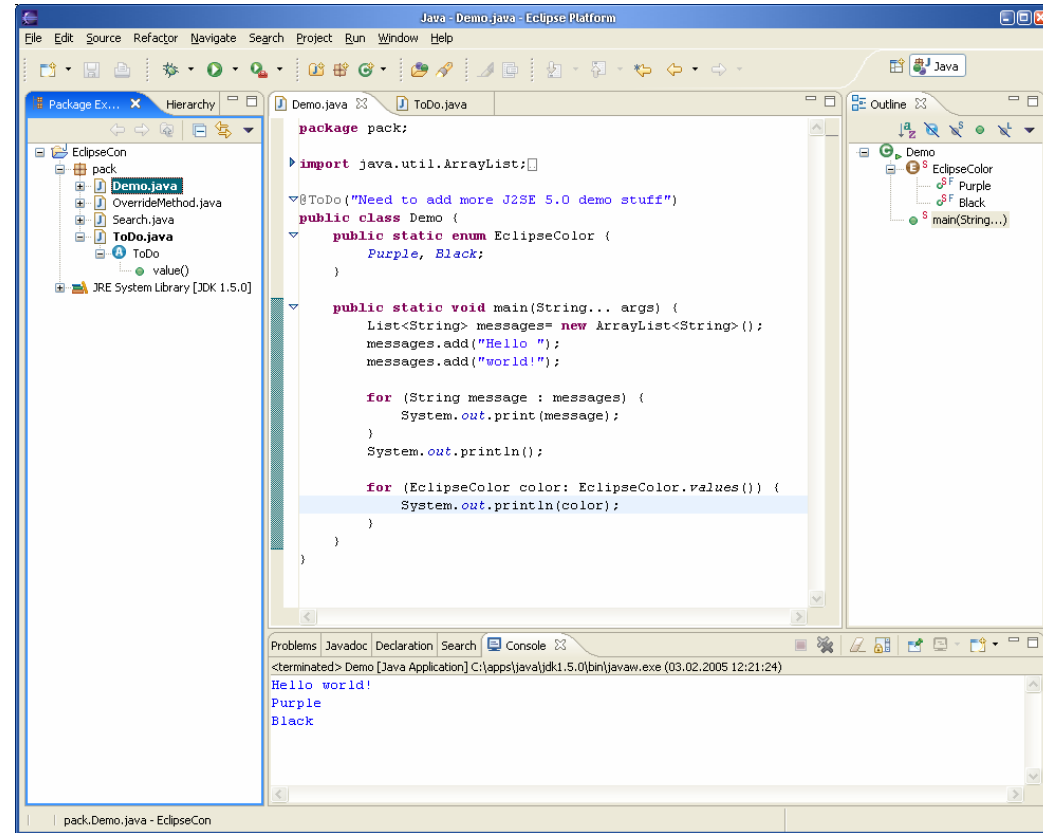
- J2SE 5.0 Overview
- Eclipse's J2SE 5.0 support
- JDT developer guide
  - New & evolved services
  - Migrating
- Questions ?

# J2SE 5.0 and Eclipse

- JSR 176 : J2SE 5.0
  - JSR 014 : Generics
  - JSR 201 : Enumerations, Autoboxing, Varargs, Enhanced for loop, Static imports
  - JSR 175 : Metadata facility (known as annotations)
- Taming the Tiger as a Eclipse user
  - R3.0 : Cheetah updates for Eclipse 3.0
  - R3.1 : Cheetah merged, progress through milestone builds
- ...as a plug-in developer
  - 3.1 is API compatible (source & binary) with 3.0 (don't disrupt existing clients)
  - Numerous API additions to support new language constructs
    - Some got previewed in 3.0
  - API replacements only were absolutely necessary (e.g. AST)
    - Old API is deprecated but still functional on J2SE 1.4 source

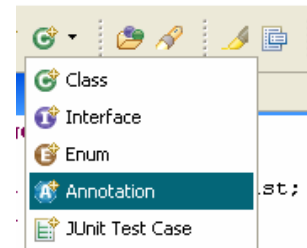
# Demo

- Generics
  - Code assist
  - Refactoring & Source Actions
- Enumerations & Annotations
  - Creation
  - Code assist
  - Refactoring & Source Actions
- Searching
  - Type parameters
  - Parameterized types & methods
- Static Imports & Varargs
  - Add & Organize Imports
  - Refactoring
- Quick Fix & Quick Assist
  - Convert for loop
  - Add Type Parameter
  - ...



# Java Model

- Still a pure source model, no element for `List<String>`
- Enum is **IType**
  - new predicate: `IType#isEnum()`
  - enum constants are **IField**, flagged with **AccEnum**
- Annotation type is **IType**
  - new predicate: `IType#isAnnotation()`
  - annotation members are **IMethod**, flagged with **AccAnnotation**
- New element: **ITypeParameter**
  - `IType#getTypeParameters()`, `IType#getTypeParameter(String name)`
- Vararg methods are flagged with **AccVarargs**



# Code Select

- Kept old API returning `IJavaElement[]`
- Allow navigating to declaring element (F3)
- Can select `ITypeParameter`
- Need accurate information to search selected element
  - True element does not surface parameterized information
  - Introduced parameterized selected element
- Extra information available `#getKey()`
- Tuned selection range
  - `java.util.List<String>` → package `java.util`
  - `java.util.List<String>` → type `java.util.List` + `<String>`

```
import java.util.List;

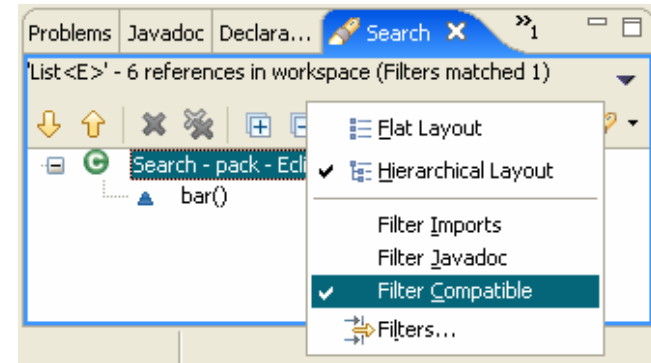
public class X {

    List<String> ls;
}

java.util.List<E>
An ordered collection (also known as a list) of the elements of the collection. Each element in the collection is inserted at the end of the collection. The elements are ordered according to their integer index (position in the collection).
```

# Searching

- Parameterized type aware searching
- More modes to control matching semantics
  - EXACT : search for `List<String>` doesn't find `List<Integer>`
  - EQUIVALENT
    - Search for `List<? extends Number>`
      - Finds `List`, `List<Integer>`
      - Does not find `List<String>`
  - ERASURE
    - Erased type, search for `List`
      - Finds `List`, `List<E>`, `List<String>`, `List<Integer>`, ...
    - Erased method, search for: `foo(String arg)`
      - Finds `<T extends String> void foo(T arg)`
- Search match can tell its specific matching mode
  - `SearchMatch#getMatchRule()`



# DOM Abstract Syntax Tree

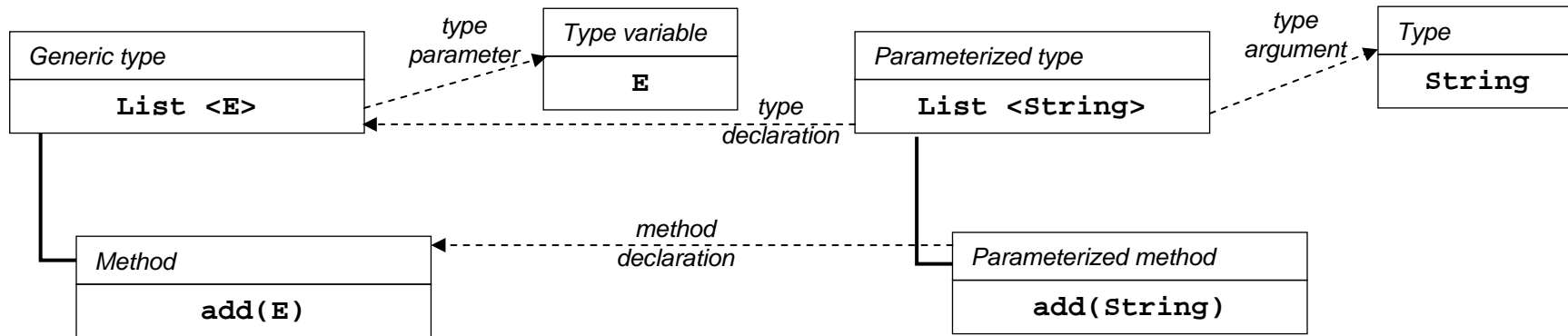
- New properties, new nodes (see Appendix)
- New version of AST tree to avoid breaking existing clients
  - Select level on AST creation: `Parser#newParser(astLevel)`
  - JLS2 handles source up to J2SE 1.4
  - JLS3 handles source up to J2SE 5.0, has a different structure and more nodes than AST.JLS2:
    - JLS2: `MethodDeclaration.getModifiers()` → `int`
    - JLS3: `MethodDeclaration.modifiers()` → `List`
    - JLS2: `TypeDeclaration#getSuperclass()` → `Name` node
    - JLS3: `TypeDeclaration#getSuperclassType()` → `Type` node
  - If you created an AST with JLS3, calling old API throws an `IllegalArgumentException`
  - Controlled upgrade to the new AST API's. JLS2 API will be marked as deprecated in 3.1

# DOM Bindings

- Representation for parameterized types/methods, wildcards etc... (in contrast to Java Model)
- Can navigate from parameterized binding back to generic binding
- New Compatibility rules
  - `ITypeBinding#isAssignmentCompatible()`
  - `ITypeBinding#isCastCompatible()`
  - `ITypeBinding#isSubTypeCompatible()`

`class List<E> { ... }`

`List<String> ls= ...;`

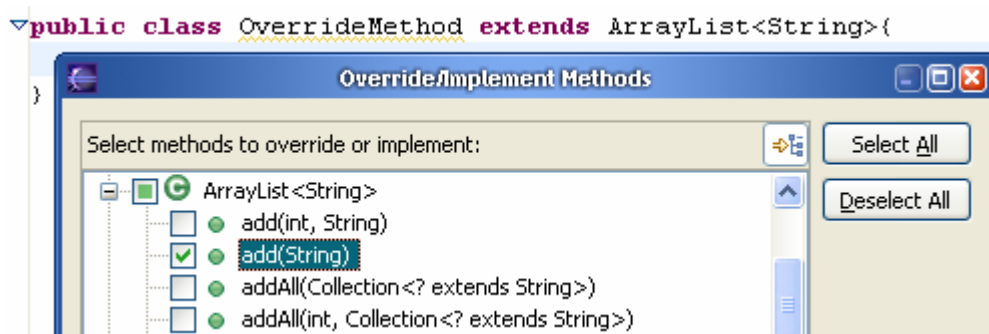


# How to convert your Code

- Rendering of Java Elements
  - No action needed if you use the API provided by JDT/UI
  - Otherwise add support for rendering enums, annotations and enum constant declarations.
    - ❗ the default visibility of enum constants is public
  - Add support for rendering type parameters. E.g. List<E>
- Searching
  - Default match rule is exact match. This means searching for a parameterized type only finds the same instantiations. For example searching for List<String> only finds List<String> not List<Integer>.
    - ❗ Consider using SearchPattern#R\_ERASURE\_MATCH and SearchPattern#R\_EQUIVALENT\_MATCH for extended searches

## How to convert your Code - continued

- Code that uses Java elements (IJavaElement and subclasses) to analyze and manipulate source code has to be converted to use the AST if it has to deal with generic and parameterized types.
  - Create an AST via `ASTParser#createAST`
  - Do the analysis using an `ASTVisitor` and by inspecting bindings
  - Modify the AST using the `ASTRewriter`



*AddUnimplementedMethodOperation acts as an example how to do the conversion.*

# How to convert AST clients

- Convert your code to use new JLS3 level. This can be done without considering new J2SE 5.0 features.
- Handle new node types (see Appendix):
  - Especially look at code that deals with TypeDeclaration nodes. This code has to handle AnnotationTypeDeclaration and EnumDeclaration as well.
  - ❗ Don't forget references to `ASTNode#TYPE_DECLARATION`
  - Handle EnhancedForStatement everywhere you processed ForStatement nodes
  - ❗ When dealing with types make sure that you use Type nodes. Name nodes can't hold all valid type names anymore (e.g. '? extends Number' is not a valid Java identifier).
- Look at usages of `ITypeBinding` and check if the code is referring to a generic type or to a parameterized type.

# Summary

- Eclipse & J2SE 5.0 support
  - Eclipse 3.1 will ship with full J2SE 5.0 support
  - Track J2SE 5.0 progress demonstrated in milestone builds
    - M5 is ready to use for J2SE 5.0 programming
  - Community feedback especially on J2SE 5.0 is essential
    - bug reports: <http://bugs.eclipse.org/bugs>
    - mailing lists: <http://www.eclipse.org/mail/index.html>
    - newsgroups: <news://news.eclipse.org/eclipse.tools.jdt>
- Converting JDT extensions to J2SE 5.0
  - No action required if extensions don't deal with new language constructs
  - Search clients: consider new modes for erasure and compatibility search
  - AST clients: a fair amount of work ;-)

# Questions

**Martin Aeschlimann**

**Jérôme Lanneluc**

**Kai-Uwe Mätzel**

**André Weinand**

**Kent Johnson**

**Tom Eicher**

**Frédéric Fusier**

**Jim des Rivières**

**Philippe Mulet**

**David Audel**

**Olivier Thomann**

**Christof Marti**

**Tobias Widmer**

**Daniel Megert**

**Markus Keller**

**Erich Gamma**

**Dirk Bäumer**

## Appendix - New JLS2 AST node types

- AnnotationTypeDeclaration, AnnotationTypeMemberDeclaration, EnhancedForStatement, EnumConstantDeclaration, EnumDeclaration, MarkerAnnotation, MemberValuePair, NormalAnnotation, ParameterizedType, QualifiedType, SingleMemberAnnotation, TypeParameter, WildcardType.