
Chapter 3.

Logic-based Software Analysis

Interactive Software Development Assistants

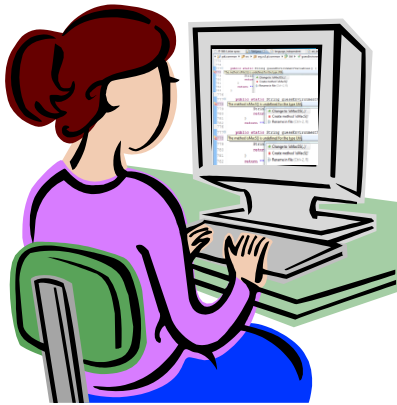
Logic-based Software Representation

Logic-based Software Analysis

Motivation ▶

Software Development Assistants

- Example: „Monitor, advise, act“ cycle in the Eclipse IDE



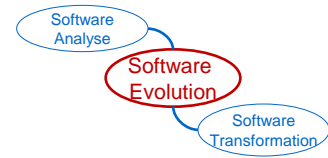
```
776  
777  
778  
779  
780  
781  
782
```

```
public static String guessEnvironmentV  
    String  
    return  
}  
return ""
```

The method isMacS() is undefined for the type Util

- Change to 'isMacOS(..)'
- Create method 'isMacS()'
- Rename in file (Ctrl+2, R)

- Behind the scenes: Software Analysis and Transformation (SAT)
- Research goal: Make SAT development quick and easy
 - ◆ Focus on conceptual essence of SAT tasks
 - ◆ ... not on APIs, implementation, manual tuning, ...



Approach

- Logic based Software Artefact Representation
- Logic-based Software Analysis
- Logic based Software Transformation
- JTransformer: Logic-based Analysis and Transformation for Java

Case Studies

- Design patterns
- Metrics and Smells
- Architecture Analysis
- Performance Analysis
- Smells and Refactorings

Logic-Based Software Representation

Logic-Based Program Representation

```
package demo;
```

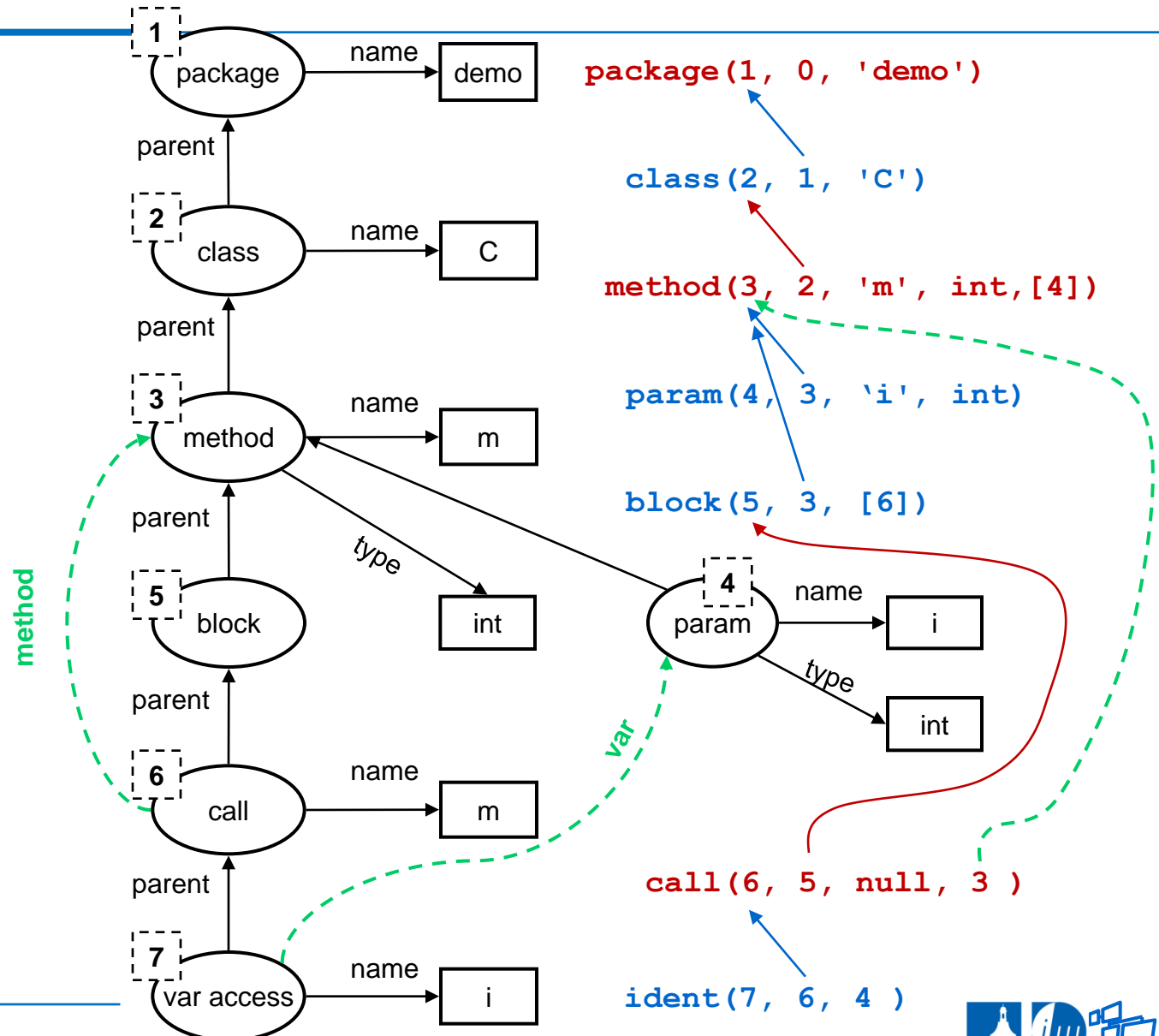
```
class C {
```

```
  int m(int i) {
```

```
    m(i);
```

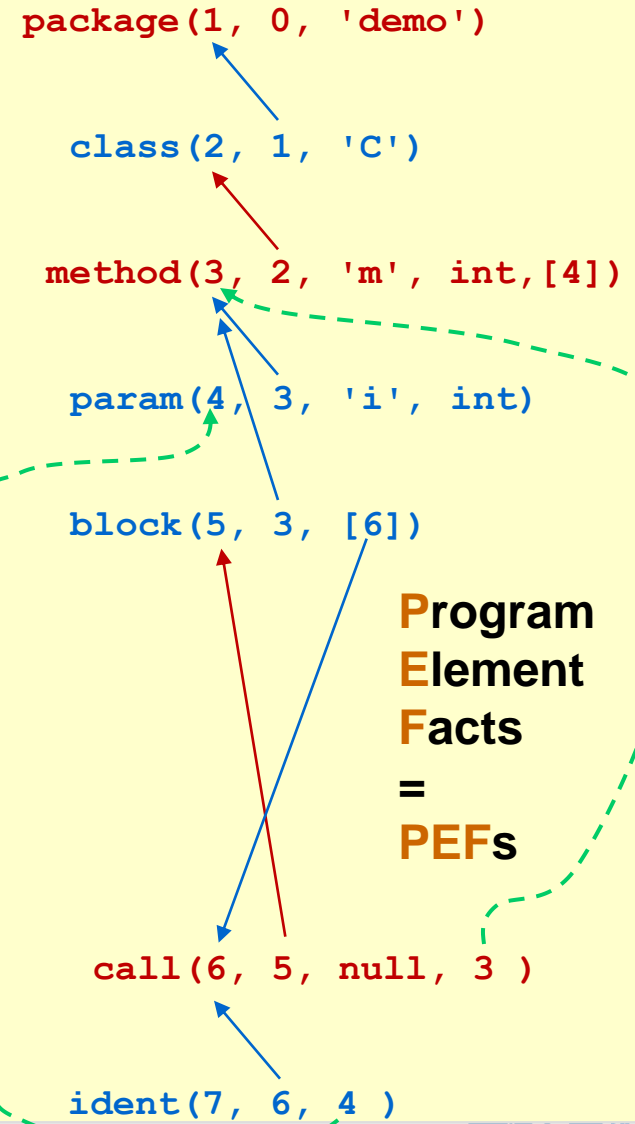
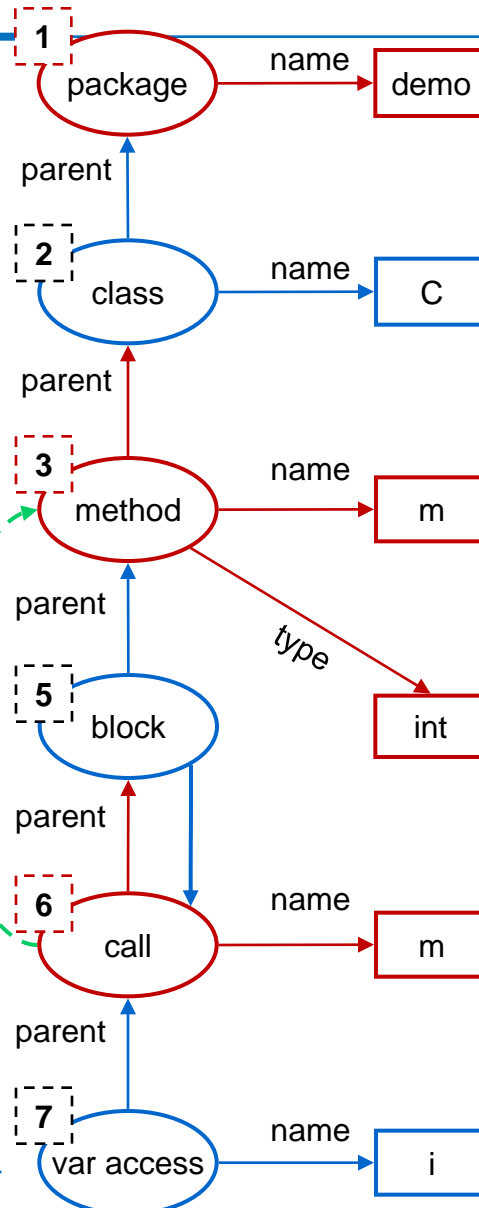
```
  }
```

```
}
```



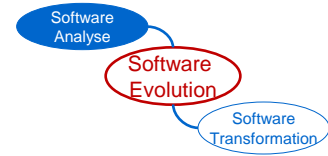
Logic-Based Program Representation

Logic terms
encode
arbitrary
graphs!

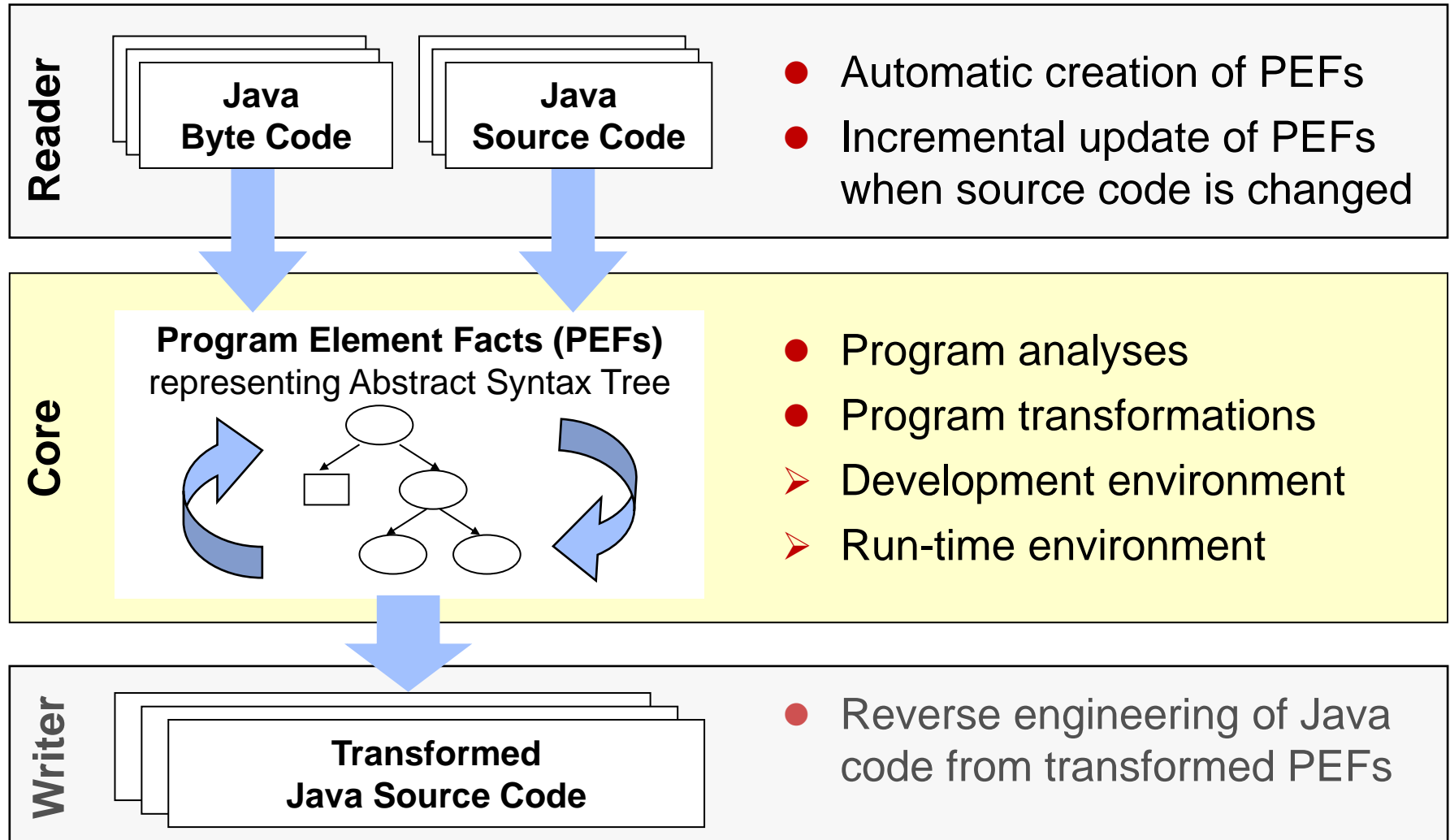


Program
Element
Facts
=
PEFs

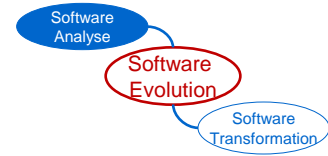
Program Element Facts (PEFs)



- Complete representation of Java 1.4 Abstract Syntax Tree
 - ◆ Projects, files and packages
 - ◆ Interface elements (types and their members)
 - ◆ Code elements (statements and expressions)
 - ◆ Comments (javadoc and block comments)
- Complete representation of Java 1.5 / 1.6 Abstract Syntax Tree
 - ◆ Generics
 - ◆ Annotations
 - ◆ Syntactic sugar (foreach, ...)



Logic-Based Software Analysis



➤ Metrics

- ◆ Anything you want → „Cultivate Plugin“ (Daniel Speicher)

➤ Bad Smells

- ◆ Hints about need for refactorings

➤ Program comprehension

- ◆ **Design Patterns**, Crosscutting Concerns

● Architecture analysis

- ◆ Dependencies, cycles, architectural rule enforcement

● Performance analysis

- ◆ Smelly Database access patterns

● Refactoring Preconditons

- ◆ Type-Constraints, hierarchy structure, ...

Metrics: Depth of Inheritance (DOI)

Find out the inheritance depth of any class.
Learn to write recursive predicates.

```
metric_doi(Class, 0) :-  
    classT(Class, _, _, _),  
    not(extendsT(Class, _Super)).  
  
metric_doi(Class, DOI) :-  
    classT(Class, _, _, _),  
    extendsT(Class, Super),  
    metric_doi(Super, DOI_Super),    // recursive call  
    DOI is DOI_Super + 1.           // evaluation of arithm. expr.
```

- Predicate defined by multiple clauses → Disjunction
- Recursion in the second clause.

Metrics: Depth of Inheritance

Write a smell detector that uses the previous metric.
Should only report values above a certain limit for source classes.

```
smell_doi(Limit, DOI, FQN):-  
    metric_doi(Class,DOI),           % use the metric  
    not(externT(Class))              % for source classes only  
    DOI >= Limit,                   % metric has critical value  
    fullQualifiedName(Class,FQN).    % get full name of class (JT)  
  
?- smell_doi(5, DOI, FQN).          % Find classes at depth >= 5.
```

- Try it out on JHotDraw.

Bad Smells: Indicators of Refactoring needs

```
non_private_field(Class,Field,FieldType,FieldName,Modif) :-  
    fieldT(Field,Class,FieldType,FieldName,_),  
    modifierT(Field,Modif),  
    ( Modif = public  
    ; Modif = package  
    ; Modif = protected  
    ).
```

```
field_without_getter(Field,Class,Type,Name,Getter) :-  
    non_private_field(Class,Field,Type,Name,_Modif),  
    concat(get, Name, Getter),  
    % No method with signature "Type Getter()" :  
    not( methodT(_Meth,Class,Getter,[],Type,_,_) ).
```


Singleton Pattern

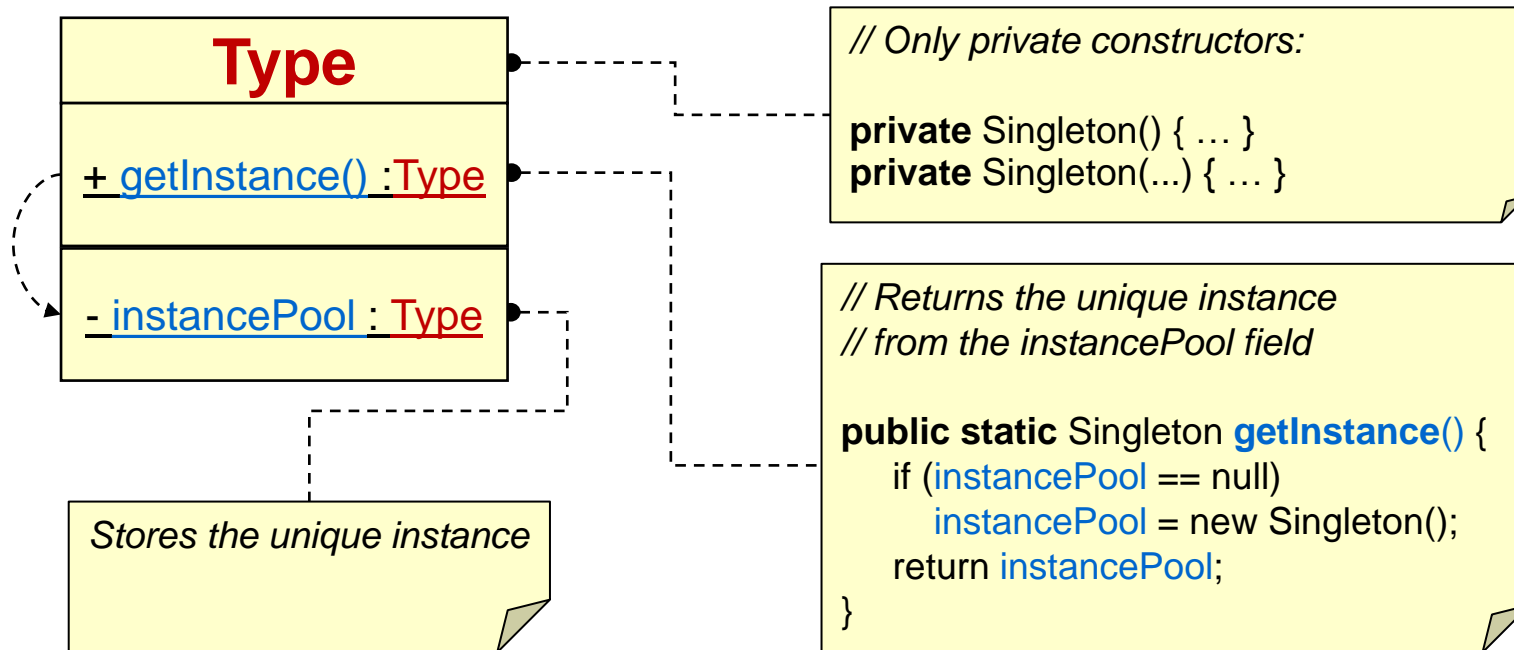
Singleton Intention: Limit the number of instances of a class

- Typically: Just one instance
 - ◆ Motivation: Central access point
 - ◆ E.g. Facade, Repository, System, Abstract Factory

- Also: Fixed number of instances
 - ◆ Motivation 1: limited resources.
 - ◆ Motivation 2: avoid expensive object creation by „Object Pool“
 - e.g. create 1000 Enterprise Java Beans, use when needed, put each back into pool after use

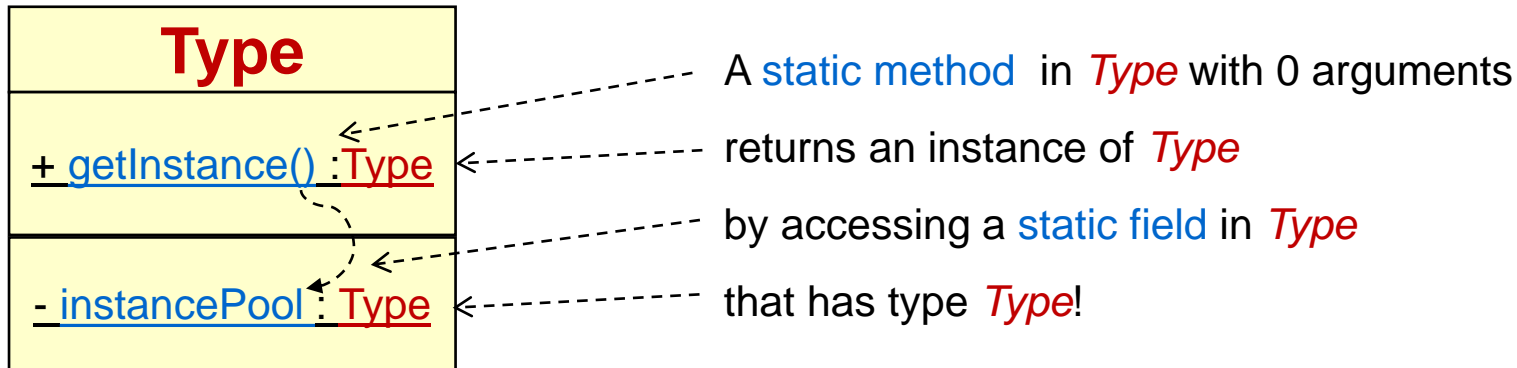
Singleton Pattern

Singleton Structure + Implementation



Singleton Pattern Detection

Singleton Structure + Implementation



Singleton Detector Predicate

```
staticMethodReturnsOwnInstance (Type, Method, Field) :-  
    methodT (Method, Type, _, [], type (_, Type, 0), _, _),  
    modifierT (Method, static),  
  
    fieldT (Field, Type, type (_, Type, 0), _, _),  
    modifierT (Field, static),  
  
    getFieldT (_, _, Method, _, _, Field).
```

Singleton Pattern Mining

Singleton Detector Predicate

```
staticMethodReturnsOwnInstance (Type, Method, Field) :-  
    methodT (Method, Type, _, [], type (_, Type, 0), _, _),  
    modifierT (Method, static),  
    fieldT (Field, Type, type (_, Type, 0), _, _),  
    modifierT (Field, static),  
    getFieldT (_, _, Method, _, _, Field).
```

Running the Singleton Detector

Query ?- `staticMethodReturnsOwnInstance(Type, Method, Field)`.

- ◆ Returns tuples of values for `<Type, Method, Field>` that represent singletons.
- ◆ Generates **all results** via backtracking.

Learning by Doing

1. Generate the JTransformer-Example project
 - ◆ Eclipse → „New“ → „Example“ → „JTransformer“
2. Open JT-Tutorial/patterns/singleton.pl
3. Consult it (F9)
4. Run the query
 - ◆ ?- `classMethodReturnsOwnInstance(Type, Method, Field).`
5. Inspect results using multi-way linking (Query – Source – Factbase)
 - ◆ Prolog console → Source editor
 - ◆ Prolog console → Factbase inspector
 - ◆ Source editor ↔ Factbase inspector → Reengineered source

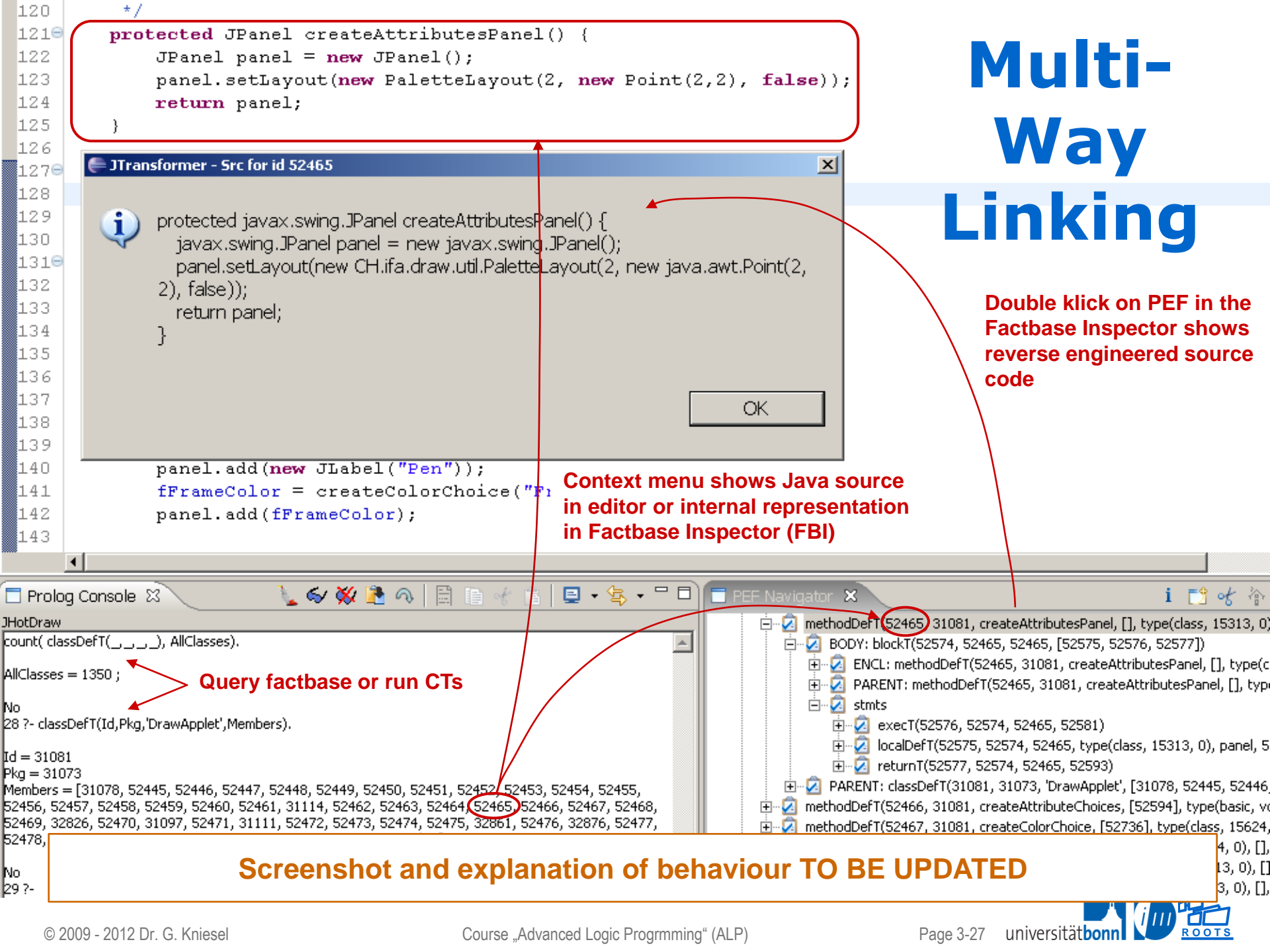
Multi-Way Linking

Double click on PEF in the Factbase Inspector shows reverse engineered source code

Context menu shows Java source in editor or internal representation in Factbase Inspector (FBI)

Query factbase or run CTs

Screenshot and explanation of behaviour TO BE UPDATED



Conclusions

- Logic Programming for Java Software Analysis
- Java program elements → Facts
- Java program analyses → Predicates
- Complete development & runtime environment

Learn More

- Go to the JTransformer website
- Follow the „Getting Started“ Tutorial
- Explore the website
- Try out
- Register in the StarTransformer mailing list
- Ask for help if you need some