Grupo de Investigación
en Reutilización y
Orientación a Objeto

Refactoring generics in JAVA: a case study on Extract Method

Authors: Raúl Marticorena rmartico@ubu.es
Carlos López clopezno@ubu.es
Yania Crespo yania@infor.uva.es
Javier Pérez jperez@infor.uva.es

14th European Conference on Software Maintenance and Reengineering
March 15-18, 2010 Universidad Rey Juan Carlos Madrid, Spain
Conference location at: Universidad Politécnica de Madrid

Outline

- Introduction
- Extract Method without Generics
- Extract Method with Generics
- Current Work
- Conclusions and Future Work

Introduction: Refactoring

- Refactoring [Fowler, 2000]
  - *"Process of changing a software system in such a way that it does not alter external behavior of the code yet improve its internal structure"*
  - *Well known catalog with a large number of refactorings*
    - *e.g. www.refactoring.com*
  - *Included in most of current tools and IDEs*

- Open Research Trends
  - Define new refactorings
  - Identify code defects *(Bad Code Smells)*
  - Refactoring engines
  - Tool support with certain language independence
  - Support evolution of programming languages

Introduction: Extract Method Refactoring

- One of the most common refactorings
- Refactoring's Rubicon
  - *"You have a code fragment that can be grouped together"*
    - *"Turn the fragment into a method whose name explains the purpose of the method"*
- Example:

```java
void printOwning(double amount){
     printBanner();
     System.out.println("name:" + _name);
     System.out.println("amount: "-+ _amount);
}
```

```java
void printOwning(double amount){
     printBanner();
     printDetails(amount);
}

void printDetails(double amount){
     System.out.println("name:" + _name);
     System.out.println("amount: "-+ _amount);
}
```

Introduction: Generics

- Included in mainstream programming languages
  - Previously in C++, Eiffel, etc.
  - Java – version 1.5
  - .NET – framework 2.0

- With common points:
  - Formal parameters in classes
- Some variants:
  - Formal parameters in methods
  - Bound clauses / where clauses
- And some particular variants:
  - e.g. wildcard types in Java

```
public class A<E> {
   E att;
   <T> void m(T element){
        ...
   }
   void c(List<? extends E> l){
        ...
   }
}
```

Introduction: Refactoring tools with generics?

- Selected refactoring: Extract Method
  - Most extended
  - Modify the method's body
- Code without / with generics
- Benchmark with different cases
- Using Java 1.6
- Goals
  - Assess the behavior of current refactoring tools
  - Search for full language support in the presence of new language features

Extract Method without Generics

- Usual cases
- Benchmark → code fragment without generics:
  - A) Without variables
  - B) With input variables (read the value)
  - C) With input variables, one of them acting also as an output variable (read several variables and write one)
  - D) With input variables and one output variable with type declaration
  - E) Several variables are modified but no accessed in the control flow after the modifications
  - F) Loop reentrance
  - G) Loop reentrance with nested loop
  - H) Add exceptions in method signature
  - I) Add exceptions with nested `try`

Extract Method without Generics

- Results

	Eclipse 3.5.0	Netbeans 6.5.1	RefactorIt 2.7.beta	IntelliJ IDEA 8.1.3	CodeGuide 8.0
A	☑	☑	☑	☑	☑
B	☑	☑	☑	☑	☑
C	👓 (always returns a value)	☑	☑	☑	☒
D	☑	☑	☑	☑	☑
E	☑	☑	☑	☑	☑
F	☑	☑	☑	☑	☒
G	☒	☑	☑	☑	☒
H	☑ (2 exceptions)	☑ (2 exceptions)	☑ (only IOException)	☑ (only IOException)	☑ (only IOException)
I	☑	☑	☑	☑	☑

Extract Method without Generics

- Assess precondition checking
- Benchmark:
  - A) Return of several variables
  - B) Return of several variables with loop (loop reentrance)
  - C) Return of several variables with nested loops (loop reentrance)
  - D) Code fragment is not complete
  - E) Conditional return
  - F) No jumps out of the fragment
  - G) Method extracted with same signature

Extract Method Refactoring without Generics

- Assess precondition checking

	Eclipse 3.5.0	Netbeans 6.5.1	RefactorIt 2.7.beta	IntelliJ IDEA 8.1.3	CodeGuide 8.0
A	☑	☑	☑	☑	☒
B	☑	☒	☑	☑	☒
C	☒	☒	☑	☑	☒
D	☑	☑	☑	☑	☑
E	☑	☑ (additional generated code)	☑	☑ (additional generated code)	☑
F	☑	☑ (additional generated code)	☑	☑ (additional generated code)	☑
G	☑	☒	☑	☑	☒ (if method exists)

Extract Method with Generics

- New cases
- Benchmark:
  - A) With class formal parameter
  - B) Using unknown type
  - C) Method formal parameter inferred from generic array type
  - D) Type inference from declarations
  - E) Bounded unknown type with formal parameter
  - F) Simple bound in method formal parameter
  - G) Multiple bound in method formal parameter

Extract Method with Generics

A) With class formal parameter

Before

```
class A<E> {
     public E remove(int index) {
          RangeCheck(index);
          modCount++;
          E oldValue = (E) elementData[index];
          int numMoved = size - index - 1;
          if (numMoved > 0)
               System.arraycopy(elementData, index + 1, elementData, index,numMoved);
          elementData[--size] = null;
          return oldValue;
     }
}
```

After

```
class A<E> {
     public E remove(int index) {
          RangeCheck(index);
          modCount++;
          E oldValue = n(index);
          elementData[--size] = null;
          return oldValue;
     }

     E n(int index) {
          E oldValue = (E) elementData[index];
          int numMoved = size - index - 1;
          if (numMoved > 0)
               System.arraycopy(elementData, index + 1, elementData, index, numMoved);
          return oldValue;
     }
}
```

Extract Method with Generics

B) Using unknown type

Before

```
class A<E> { ...
     public boolean addAll(Collection<? extends E> c) {
          Object[] a = c.toArray();
          int numNew = a.length;
          ensureCapacity(size + numNew);
          System.arraycopy(a, 0, elementData, size, numNew);
          size += numNew;
          return numNew != 0;
     }
}
```

After

```
class A<E> { ...
     public boolean addAll(Collection<? extends E> c) {
          int numNew = n(c);
          size += numNew;
          return numNew != 0;
     }
     ...
     int n(Collection<? extends E> c) {
          Object[] a = c.toArray();
          int numNew = a.length;
          ensureCapacity(size + numNew);
          System.arraycopy(a, 0, elementData, size, numNew);
          return numNew;
     }
}
```

Extract Method with Generics

C) Method formal parameter inferred from generic array type

Before

```
public <T> T[] toArray(T[] a) {
              if (a.length < size)
                     return (T[]) Arrays.copyOf(elementData, size, a.getClass());
              System.arraycopy(elementData, 0, a, 0, size);
              if (a.length > size)
                     a[size] = null;
              return a;
}
```

```
public <T> T[] toArray(T[] a) {
    if (a.length < size)
        return (T[]) Arrays.copyOf(elementData, size, a.getClass());
    n(a);
    return a;
}

<T> void n(T[] a) {
    System.arraycopy(elementData, 0, a, 0, size);
    if (a.length > size)
        a[size] = null;
}
```

After

Extract Method with Generics

D) Type inference from declarations

Before

```
public static <T> List<T> toList(T[] arr) {
        List<T> list = new ArrayList<T>();
        for (T elt : arr) {
            list.add(elt);
        }
        return list;
}
```

After

```
public static <T> List<T> toList(T[] arr) {
    List<T> list = new ArrayList<T>();
    n(arr, list);
    return list;
}
static <T> void n(T[] arr, List<T> list) {
    for (T elt : arr) {
        list.add(elt);
    }
}
```

Extract Method with Generics

E) Bounded unknown type with formal parameter

Before

```
public static <T> void copy(List<? super T> dst, List<? extends T> src) {
    for (int i = 0; i < src.size(); i++) {
        dst.set(i, src.get(i));
    }
}
```

```
public static <T> void copy(List<? super T> dst, List<? extends T> src) {
    n(dst, src);
}

static <T> void n(List<? super T> dst, List<? extends T> src) {
    for (int i = 0; i < src.size(); i++) {
        dst.set(i, src.get(i));
    }
}
```

After

Extract Method with Generics

F) Simple bound in method formal parameter

```
public static <S extends Readable, T extends Appendable> void copy(S src,
        T trg, int size, boolean flag) throws IOException {
    CharBuffer buf = CharBuffer.allocate(size);
    int i = src.read(buf);
    while (i > 0) {
        buf.flip();
        trg.append(buf);
        buf.clear();
        i = src.read(buf);
    }
}
```

Before

```
public static <S extends Readable, T extends Appendable> void copy(S src,
        T trg, int size, boolean flag) throws IOException {
    CharBuffer buf = CharBuffer.allocate(size);
    int i = src.read(buf);
    n(buf, i, src, trg);
}

static <S extends Readable, T extends Appendable> void n(CharBuffer buf,
        int i, S src, T trg) throws IOException {
    while (i > 0) {
        buf.flip();
        trg.append(buf);
        buf.clear();
        i = src.read(buf);
    }
}
```

After

Extract Method with Generics

G) Multiple bound in method formal parameter

```java
public static <S extends Readable & Cloneable, T extends Appendable &
Cloneable> void copy(S src, T trg, int size) throws IOException {
    CharBuffer buf = CharBuffer.allocate(size);
    int i = src.read(buf);
    while (i > 0) {
        buf.flip();
        trg.append(buf);
        buf.clear();
        i = src.read(buf);
    }
    src.close();
    terg.close();
}
```

Before

```java
public static <S extends Readable & Cloneable, T extends Appendable &
Cloneable> void copy(S src, T trg, int size) throws IOException {
    CharBuffer buf = CharBuffer.allocate(size);
    int i = src.read(buf);
    n(buf, i, src, trg);
}

static <S extends Readable & Cloneable, T extends Appendable & Cloneable> void
n(CharBuffer buf, int i, S src, T trg) throws IOException {
    while (i > 0) {
        buf.flip();
        trg.append(buf);
        buf.clear();
        i = src.read(buf);
    }
}
```

After

Extract Method with Generics

- Results

	Eclipse 3.5.0	Netbeans 6.5.1	RefactorIt 2.7.beta	IntelliJ IDEA 8.1.3	CodeGuide 8.0
A	☑	☑	☒	☑	☑
B	☑	☑	☒	☑	☑
C	☒	☒	☒	☑	☒
D	☑	☒	☒	☑	☒
E	☒	☒	☒	☑	☒
F	☒	☒	☒	☑	☒
G	☒	☒	☒	☑	☒

Current Work

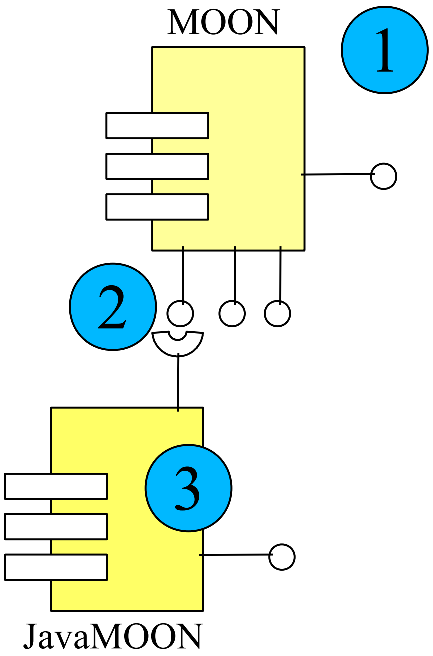
- MOON [Crespo 2000]
  - Minimal Object-Oriented Notation
    - abstractions for refactoring
    - 50 classes
  - Storing:
    - Classes
    - Relationships
    - Variants on the type system
    - Entities
      - Concepts in source code with type
      - *self reference*, *super reference*, *local variable*, *method formal argument*, *class attribute* and *function result*
    - Expresssions
    - Instructions
      - *creation*, *assignment*, *call* and *compound instructions*

Current work

① General concepts: defined and implemented on MOON

② Extensible:
- Defined on MOON
- Implemented on concrete language (framework instantiation)

③ Particular: defined and implemented on a concrete language



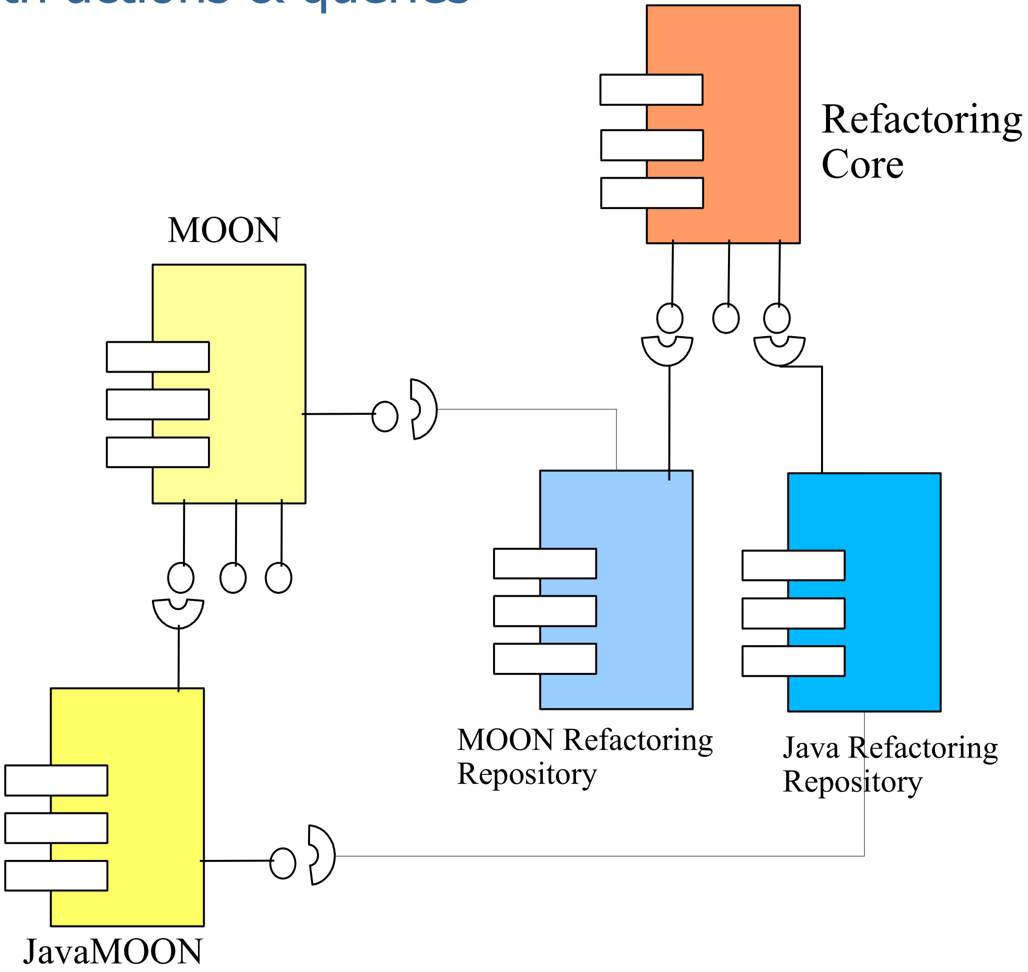
`ObjectMoon`
`Name`

`ClassDef`
`AttDec`
`FormalPar`

`JavaBoundS`
`JavaWildcardType`

Current Work

- Frameworks as solution
    - Repositories with actions & queries

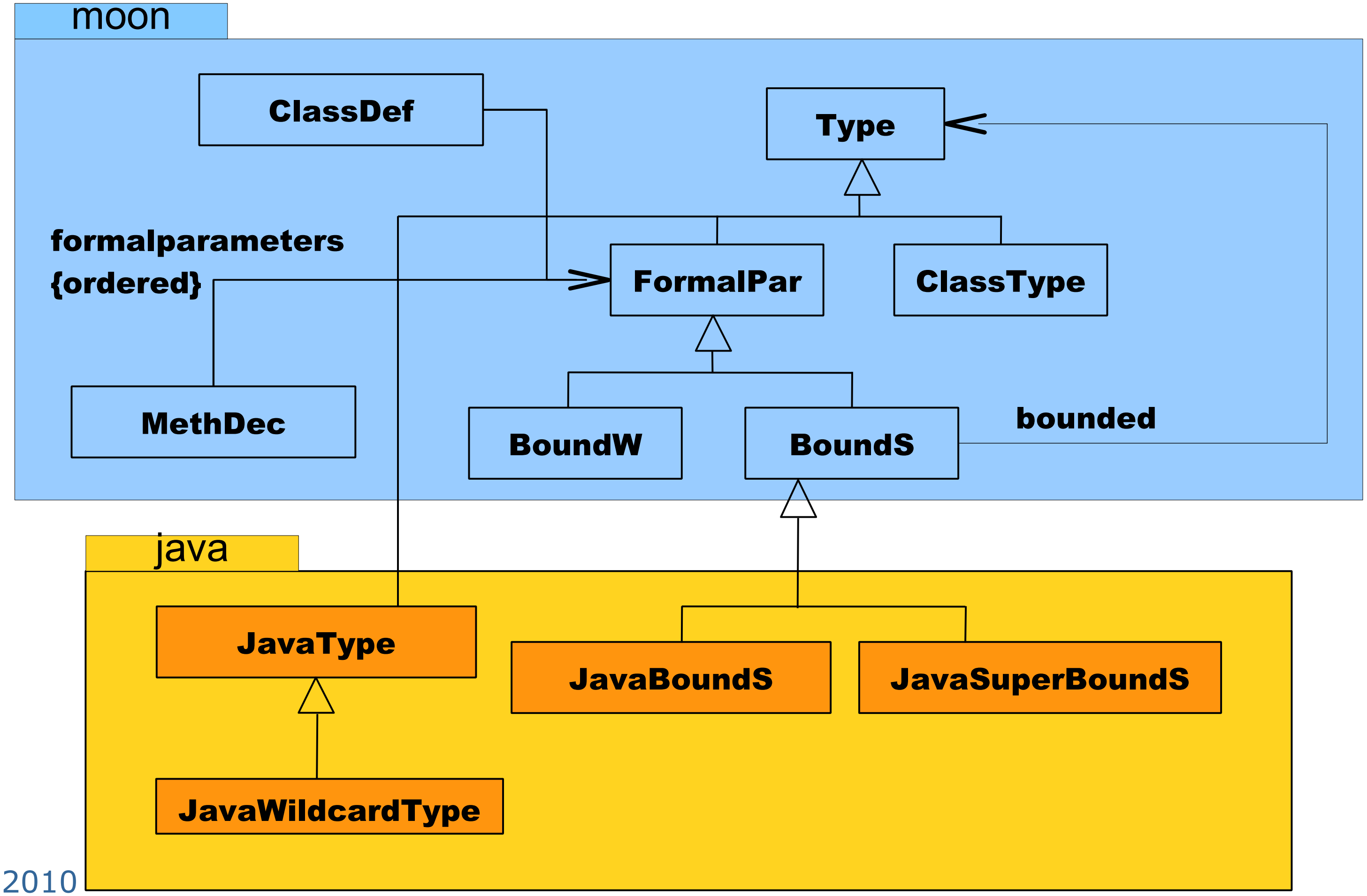


Current Work

- Generics support in the metamodel and Java extension

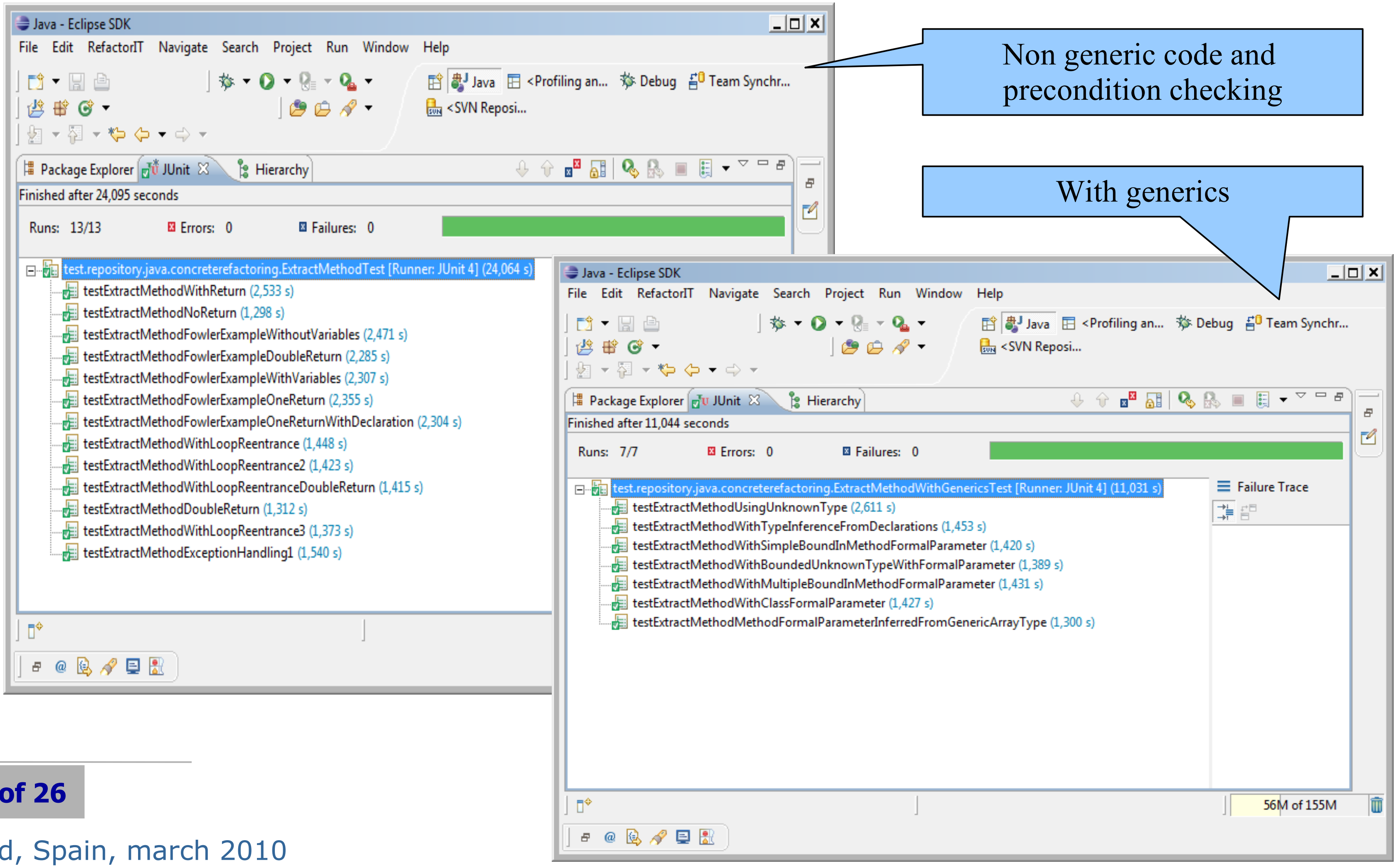
MOON

Java



Current Work

- Benchmarks implemented as JUnit tests



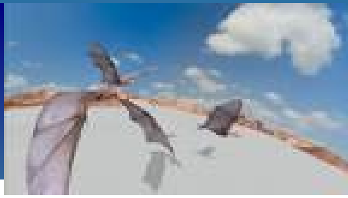
Non generic code and precondition checking

```
Finished after 24,095 seconds
Runs: 13/13    Errors: 0    Failures: 0
test.repository.java.concreterefactoring.ExtractMethodTest [Runner: JUnit 4] (24,064 s)
  testExtractMethodWithReturn (2,533 s)
  testExtractMethodNoReturn (1,298 s)
  testExtractMethodFowlerExampleWithoutVariables (2,471 s)
  testExtractMethodFowlerExampleDoubleReturn (2,285 s)
  testExtractMethodFowlerExampleWithVariables (2,307 s)
  testExtractMethodFowlerExampleOneReturn (2,355 s)
  testExtractMethodFowlerExampleOneReturnWithDeclaration (2,304 s)
  testExtractMethodWithLoopReentrance (1,448 s)
  testExtractMethodWithLoopReentrance2 (1,423 s)
  testExtractMethodWithLoopReentranceDoubleReturn (1,415 s)
  testExtractMethodDoubleReturn (1,312 s)
  testExtractMethodWithLoopReentrance3 (1,373 s)
  testExtractMethodExceptionHandling1 (1,540 s)
```

With generics

```
Finished after 11,044 seconds
Runs: 7/7    Errors: 0    Failures: 0
test.repository.java.concreterefactoring.ExtractMethodWithGenericsTest [Runner: JUnit 4] (11,031 s)
  testExtractMethodUsingUnknownType (2,611 s)
  testExtractMethodWithTypeInferenceFromDeclarations (1,453 s)
  testExtractMethodWithSimpleBoundInMethodFormalParameter (1,420 s)
  testExtractMethodWithBoundedUnknownTypeWithFormalParameter (1,389 s)
  testExtractMethodWithMultipleBoundInMethodFormalParameter (1,431 s)
  testExtractMethodWithClassFormalParameter (1,427 s)
  testExtractMethodMethodFormalParameterInferredFromGenericArrayType (1,300 s)
```

Conclusions and Future Work



- Evolution of programming languages notably affects refactoring tools
  - Benchmarks are required to test new language features in refactoring
- Architectures should be ready to include new language features
  - Ease of extending metamodel is required
- Refactorings with generics
  - Define and build new refactorings
- Study the effects of new features in concrete languages over well known refactorings
  - e.g. annotations (Java) / attributes (.NET), asserts, DbC
  - e.g. new features in Java 7

Thank you very much

Authors: Raúl Marticorena rmartico@ubu.es
Carlos López clopezno@ubu.es
Yania Crespo yania@infor.uva.es
Javier Pérez jperez@infor.uva.es