Operator Overloading in Java

Semester Project

Yacine Saidji – 6th semester CS

Advisor : Matthias Zenger

What you are going to see ...

- Why operator overloading ?
- What kind of operators ?
- Specification
- Semantic
- Implementation
- Future directions

Why operator overloading ?

- Scientific computing community, JavaGrande
- A complex class ! (this is a complex case)

- c = a.times(b).plus(c)
- c = a * b + c

- Available in Eiffel, Sather, C++

What kind of operators ?

- Binary operators 1 + 1 ~~!mybool~~
- Predefined ones : + * – << ...
- Free ones : @ # @# ^+ ...
- Forbidden ones :
  - //
  - ? :
  - += , etc...

Specification

- Lexical grammar
  - Defines OPERATOR token
- Syntactic grammar
  - Operator method declarations
    - class complex {
      complex this + (complex c) { ... }
      }
  - Binary Expression
- Semantic

Semantic

- Allowed

```
class complex {
        complex this + (float re) {...}
        complex (float re) + this {...}
}
```

- Disallowed

```
class complex {
        complex this + (float re) {...}
        float    this + (float re) {...}
}
```

More semantics

- **Left–operand** operator

```
class foo {
    int   this   +   (int i)   { ... }
}
```

- **Right–operand** operator

```
class bar {
    int   (int i)   +   this   {...}
}
```

- Expressions f + i i + b ~~b + i~~

Semantics, again...

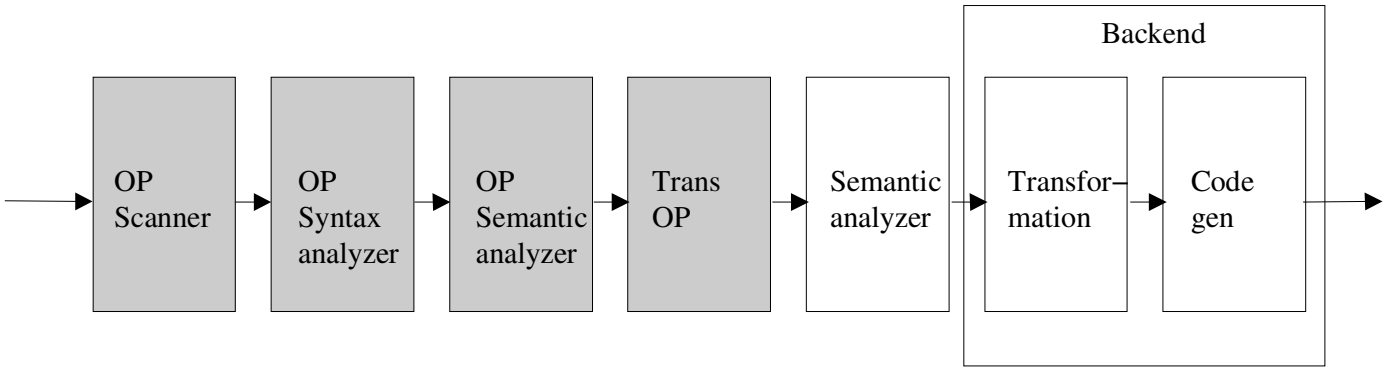
- Operator Method Invocation Expressions

```
class A {
    int  (int i) #  this      { return 1; }
    int  this    #  (int i)   { return 2; }

    public static void main(String[] args) {
        A a = new A();
        System.out.println( (1#a) + "," + (a#1) );
    }
}                   prints :  1,2
```

Implementation

- Jaco [Zenger98] Extensible Java compiler
- About 30 components organized in 4 families.



Scanner

- Longest match rule

  + returns PLUS
  @ returns OPERATOR (attached name @)

- Make a wish !

Parser

- JavaCUP
- Name mangling

  left–operand OP$$L$xx
  right–operand OP$$R$xx

- Free operators get an opcode

  used in Binary Expression

Type and Name checking (and more)

- In expression, lookup operator method
- Report ambiguities

  ...
  c = f + c; // c is complex , f is foo
  ...

  Is method complex.OP$$R$+(foo) accessible ?
  Is method foo.OP$$L$+(complex) accessible ?
  Are they both accessible ?

AST transformation

- Last stage !
- Transforms the tree into a regular Java tree.

- Binops get method calls.
  c + i c.OP$$L$(i)

- Watch out ! Left–to–right order of evaluation.
  i + c ((tmp = i) == tmp) ? c.OP$$R$(tmp)
  : c.OP$$R$(tmp)

Future directions

- Specify the precedence and associativity
- Tune the lexical grammar
- Provide a **super** form in expressions

- Java Community Process
- But before: bugs, bugs, bugs .... code AND spec

Big thanks to

- Matthias !

- Thanks for Jaco
- And thanks for your help !