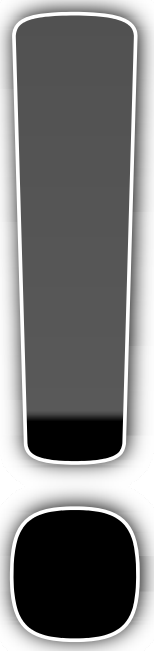


Please,
feel free to interrupt me at any time.

Do you use:

- Multiple Inheritance?
- Mixins?

Traits

A Language Feature for PHP?

Stefan Marr
PHP Unconference Hamburg
26-27 April 2008

HPI Hasso Plattner Institut

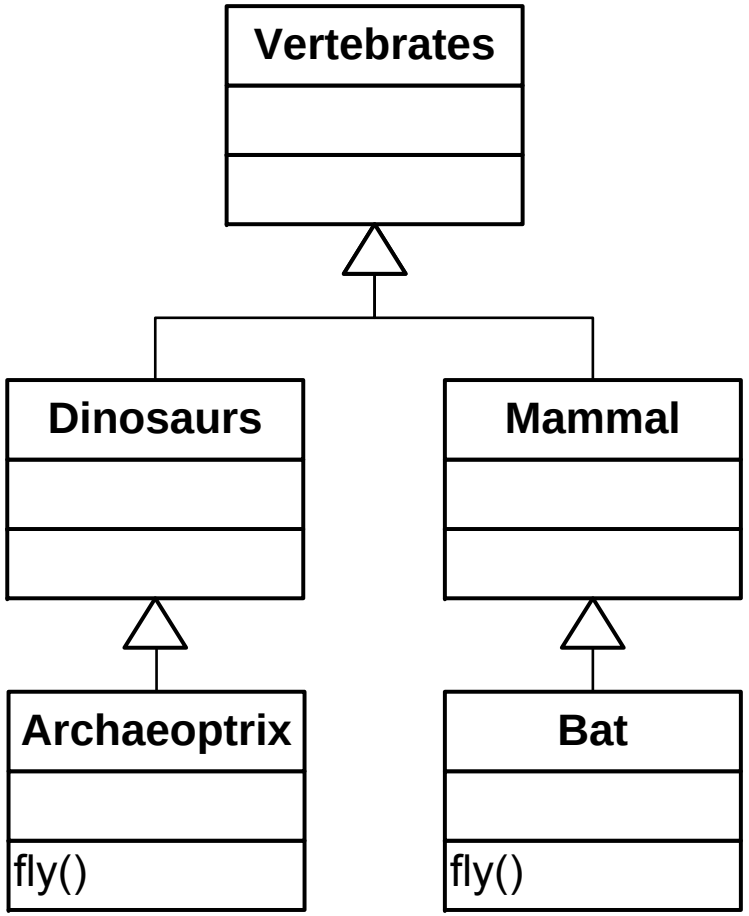
Agenda

1. Introduction to Traits
2. Traits Applied
3. Traits for PHP: Trait versus Graft
4. Roundup and Conclusion

Which problem is to be solved?

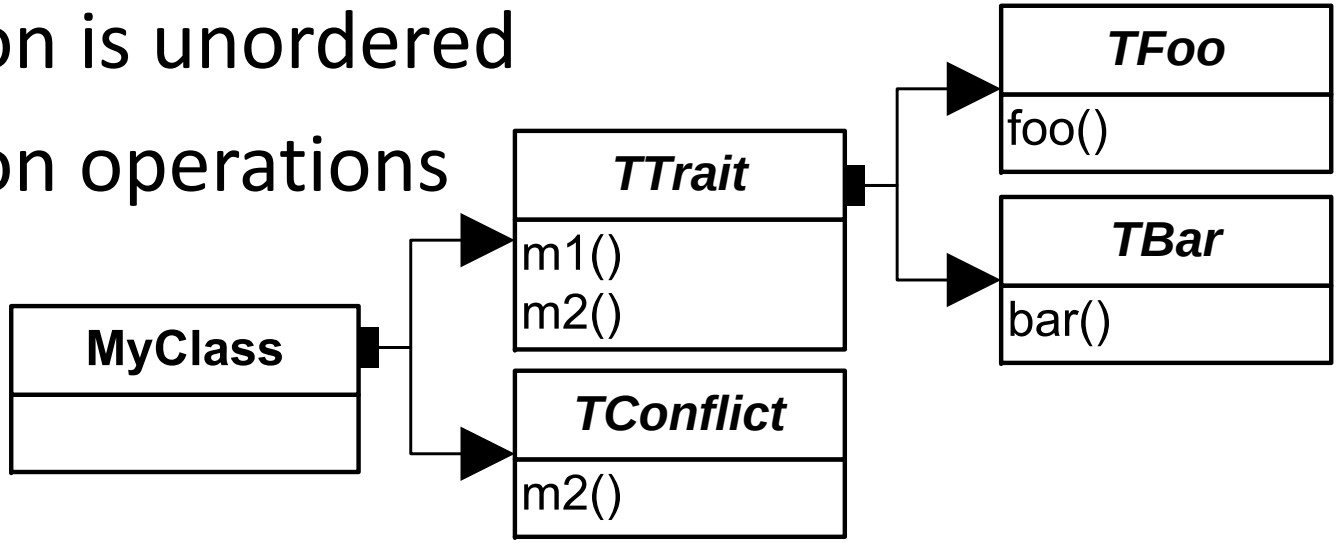
- Single inheritance
- Simple but not expressive enough

- Leads to
  - Code duplication
  - Inappropriate hierarchies

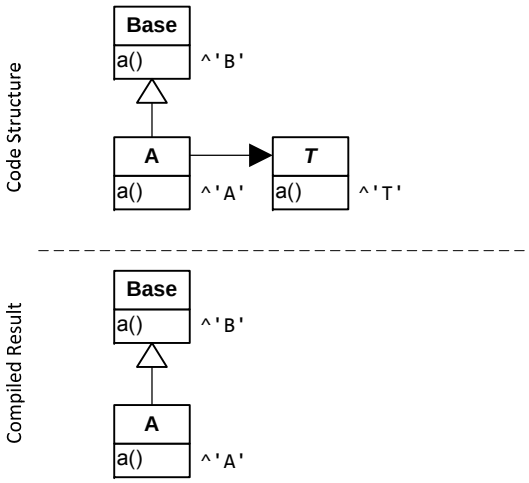


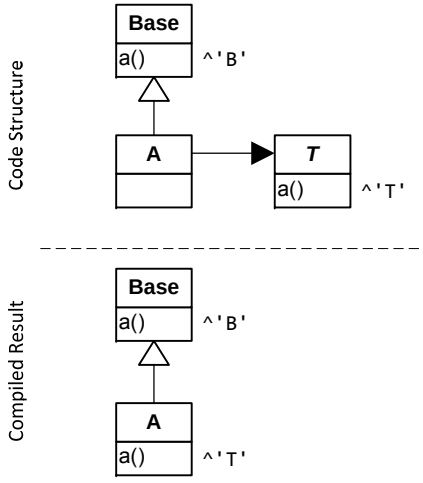
Traits

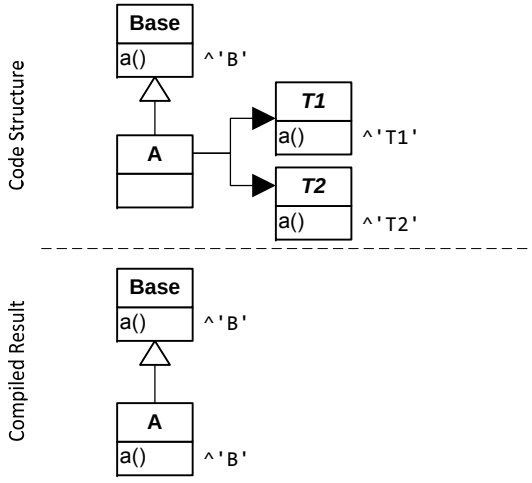
- Set of methods
- Possibly composed of other Traits
- Composition is unordered
- Composition operations
  - Sum
  - Exclusion
  - Aliasing
- Can be flattened away

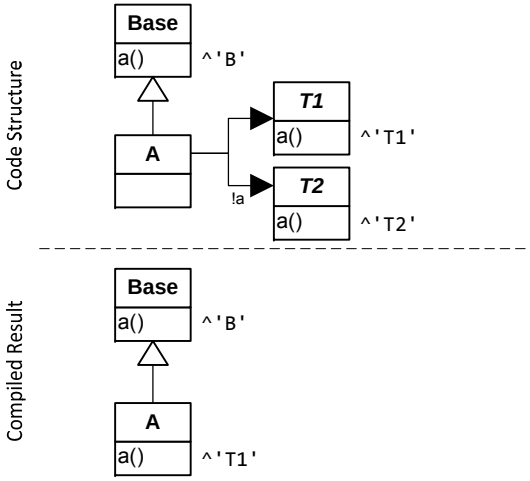


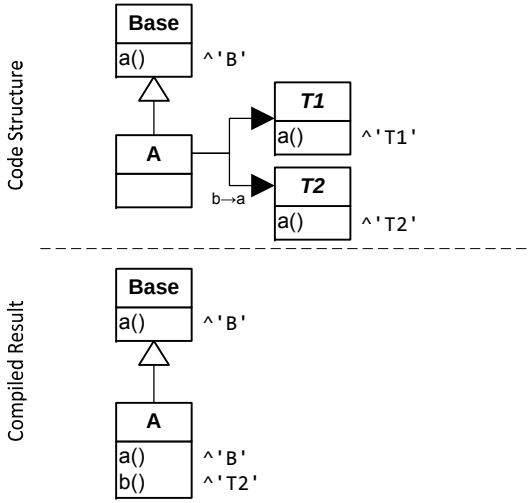
Whiteboard


Code Structure
Base
a()
^'B'
A
a()
^'A'
T
a()
^'T'
Compiled Result
Base
a()
^'B'
A
a()
^'A'


Code Structure
Base
a()
^'B'
A
T
a()
^'T'
Compiled Result
Base
a()
^'B'
A
a()
^'T'


Code Structure
Base
a()
^'B'
A
T1
a()
^'T1'
T2
a()
^'T2'
Compiled Result
Base
a()
^'B'
A
a()
^'B'


Code Structure
Base
a()
^'B'
A
T1
a()
^'T1'
!a
T2
a()
^'T2'
Compiled Result
Base
a()
^'B'
A
a()
^'T1'

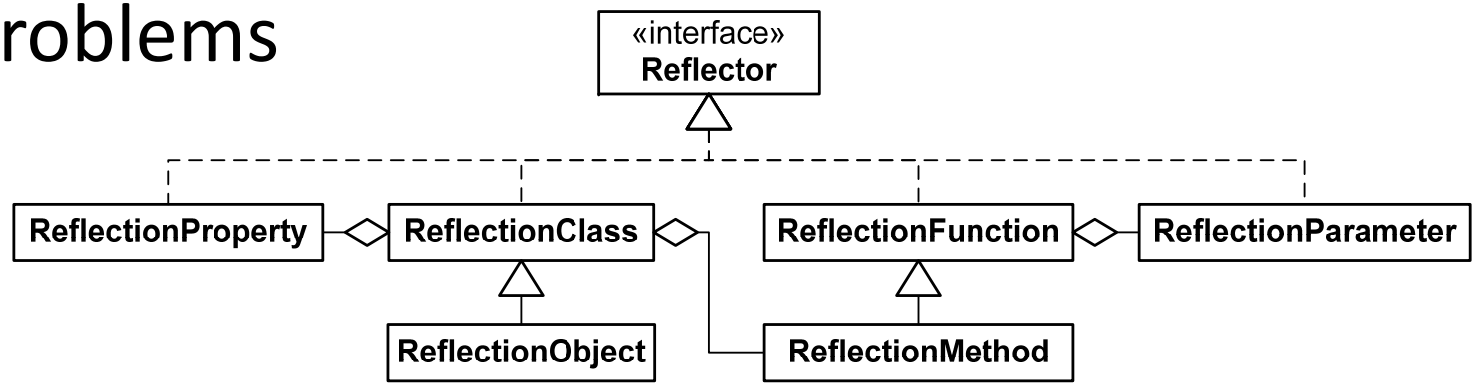

Code Structure
Base
a()
^'B'
A
T1
a()
^'T1'
b→a
T2
a()
^'T2'
Compiled Result
Base
a()
^'B'
A
a()
^'B'
b()
^'T2'

Case Studies Illustrating Benefits of Traits

TRAITS APPLIED

Use Case for Traits in a PHP Library

- ezcReflection enhances PHP Reflection API
  - Adds XDoclet like annotations for PHP
  - Enhances type hinting using PHPDoc type annotations
- Illustrates solution for code duplication problems



Refactoring ezcReflection with Traits

original

with Traits
green parts are extracted to the Traits on the right-hand side, and will be removed from the classes

Case Study: java.io [6]

- Mature code base, in wide use
- java.io Library from Java 1.4.2
- 79 classes, 25000 lines of code
- Results
  - 12 classes changed
  - 30 duplicated methods removed
  - 14 Traits introduces

Case Study: Smalltalk Collection Classes [10]

- Problem: code reuse vs. conceptual categorization
  - Unnecessary inheritance
  - Code duplication
  - Methods to high in hierarchy
  - Conceptual shortcomings
- Results for core collection classes
  - 23 classes changed,
    12% less code, 19.4% less methods

Case Study: Smalltalk Collection Classes [10]

- Lessons Learned
  - Traits simplify refactoring
  - Very fine-grained traits
  - Tools are important
  - Traits enable to defer design of class hierarchy
- Improvements
  - Uniformity, understandability, reuse

Check for Limitations

- ~~Code duplication~~ ✓
- ~~Inappropriate hierarchies~~ ✓
- ~~Diamond problem~~ ✓
- ~~Fragile hierarchies~~ ✓
- ~~Lack of composition control~~ ✓

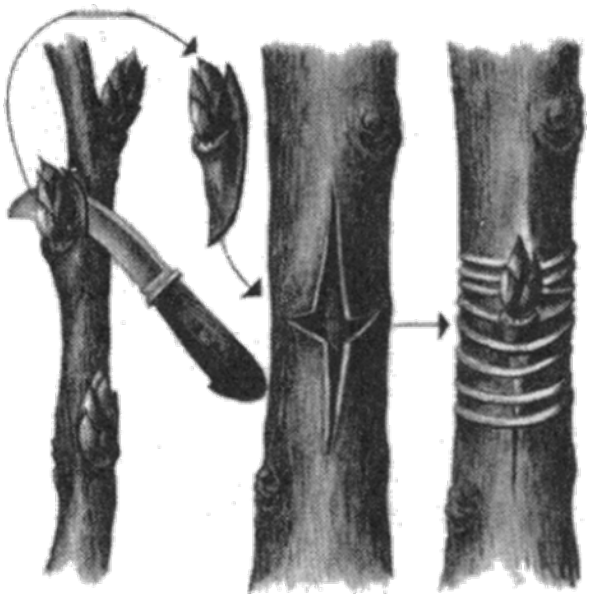
Trait versus Graft

TRAITS FOR PHP

HPI Hasso Plattner Institut

What is a Graft?

- Stateful, non-breakable entity of reuse
  - Derived from the Traits notion
  - Proposed with RFC: Non Breakable Traits for PHP
- Why Graft?



From *The Grafter's Handbook* by R.J. Garner

A Comparison

Traits

- Purpose, reuse of
  - behavior
- Features
  - Stateless
  - Flattenable
  - Explicit conflict resolution
  - Highly composable
- Notion
  - Language-base Copy and Paste

Grafts

- Purpose, reuse of
  - modules smaller then classes
- Features
  - Stateful
  - Hides complexity
  - Conflict avoidiance
  - Strong encapsulation
- Notion
  - Delegation with self-impersonation

Traits Syntax and Semantics

```
trait A {
  public function small() { echo 'a'; }
  public function big()   { echo 'A'; }

  abstract public function doBFoo();

  public function doAFoo() {
    echo 'AFoo uses BFoo: ';
    $this->doBFoo();
  }
}

trait B {
  public function small() { echo 'b'; }
  public function big()   { echo 'B'; }

  public function doBFoo() {
    echo 'B-FOO';
  }
}
```

```
class Talker {
  use A, B {
    B::small instead A::small,
    A::big instead B::big,
    B::big as talk
  }
}

$talker = new Talker();
$talker->small();  // b
$talker->big();    // A
$talker->talk();   // B

$talker->doAFoo(); // AFoo uses
                   // BFoo: B-FOO
$talker->doBFoo(); // B-FOO
```

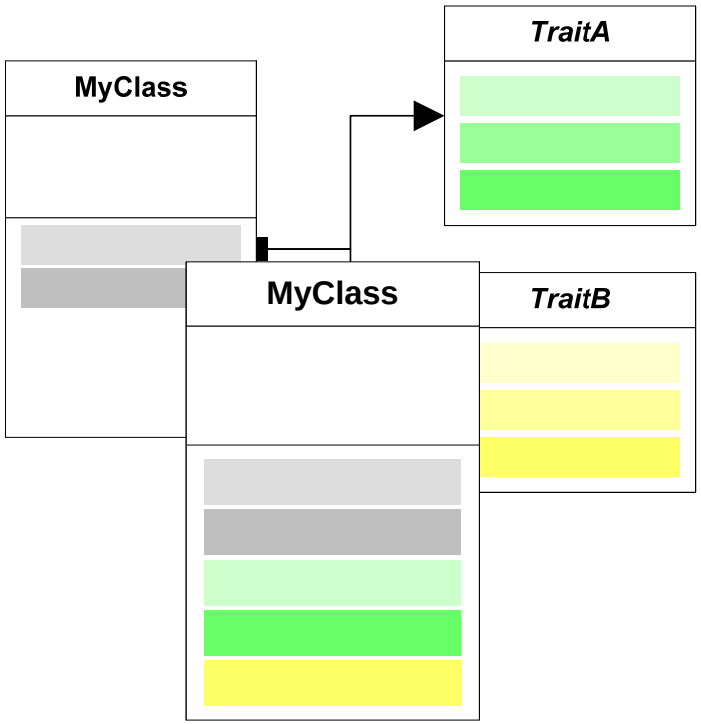
Grafts Syntax and Semantics

```
graft Counter {
  private $cnt = 0;
  public function inc() {
    $this->cnt++;
  }
  public function reset() {
    $this->cnt = -1;
    $this->inc();
  }
}
graft DB {
  private $db;
  public function connect() {
    $this->db = new FooDB('param');
  }
  public function reset() {
     $this->db->flush();
     $this->db = null;
  }
  public function doFoo(){echo 'foo';}
}
```

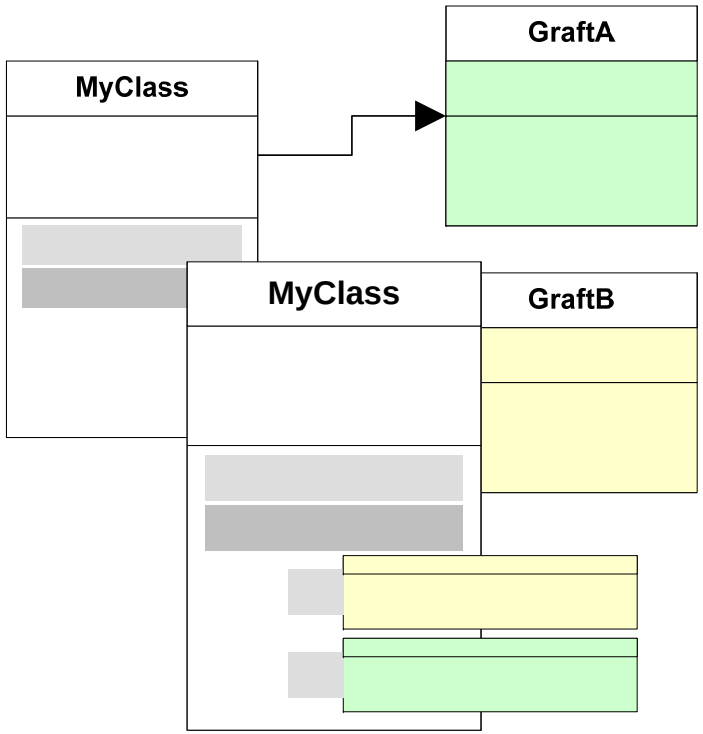
```
class MyPage {
  include Counter {
    public incCnt() as inc();
    public resetCnt() as reset();
  }
  include DB {
    public connect();
    public reset();
  }
  public function inc() {
    /* next page */
  }
}

$page = new MyPage();
$page->connect();
$page->incCnt();   //($Countercnt == 1)
$page->resetCnt(); // $Countercnt = 0;
$page->inc();      // goto next page
$page->doFoo();    // FATAL ERROR
```

My Mental Image

Traits



Grafts



Trait Support, Discussion, Resume

ROUNDUP AND CONCLUSION

Environments Supporting Traits

- Squeak: In standard distribution since 3.9 [11]

```
Trait named: #TDrawing uses: {}
draw
   ^self drawOn: World canvas in: bounds
bounds
   self requirement

Object subclass: #Circle
   instanceVariableNames: ''
   traits: { TDrawing }
   ...
```

Environments Supporting Traits

- Scala: static-typed language running on JVM [7]

```
trait TDrawing {
  def draw() : Any =
      drawOn(World.canvas(), bounds)
  def bounds() : Any
}

class Circle extends TDrawing {
  ...
}
```

Environments Supporting Traits

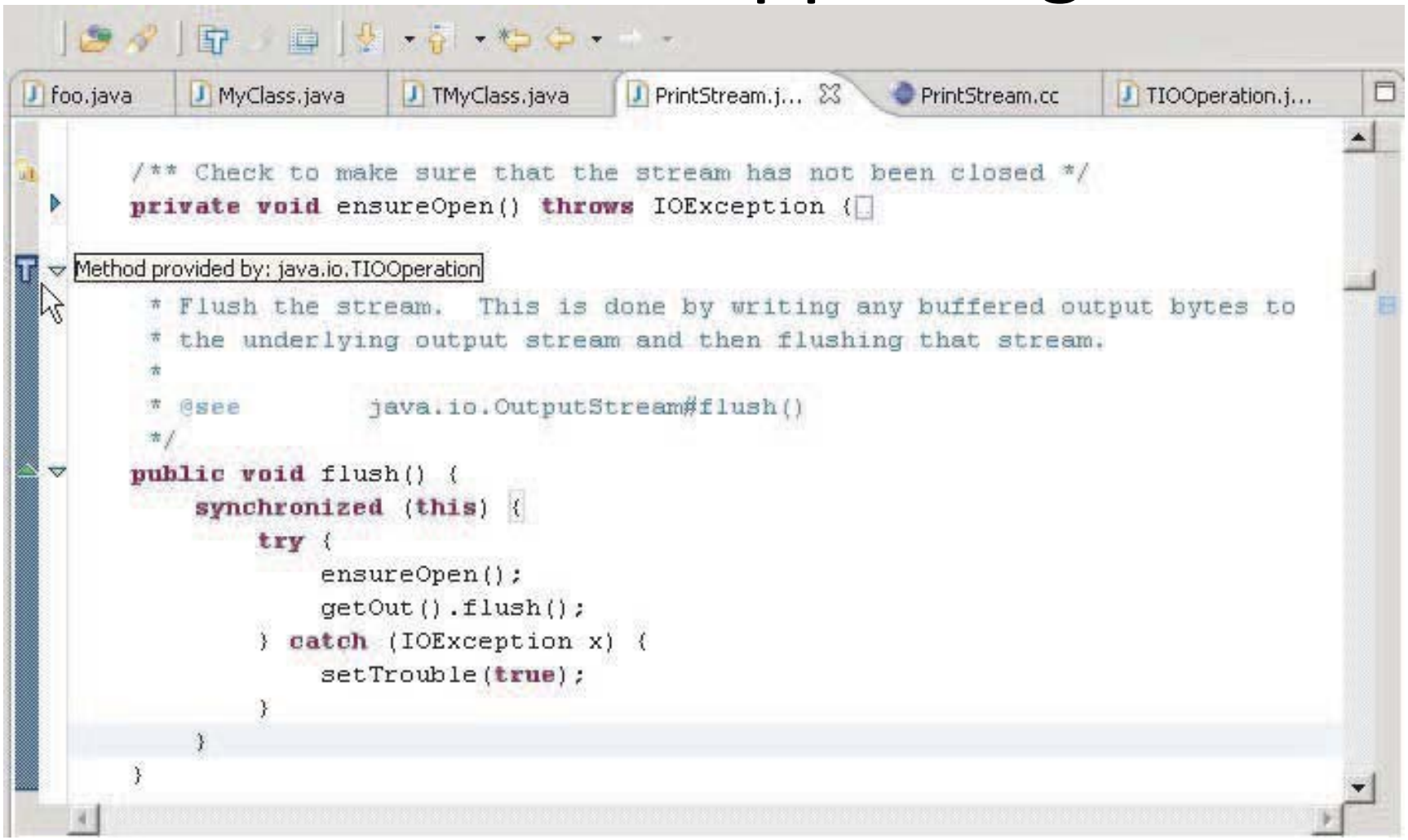
- Perl6: Flavor of Traits with states called Roles [12]

```
role TDrawing {
  method draw() {
    return drawOn(World.canvas(),
      bounds);
  }
}

class Circle does TDrawing {
  ...
}
```

Environments Supporting Traits

- Approaches for plain Java
  - Using AspectJ [2]
  - Utilizing an Eclipse Plugin [8]

Environments Supporting Traits



Environments Supporting Traits

- C#: Prototype Implementation for Rotor [9]

```
trait TDrawing {
   public Object draw() {
      return drawOn(World.canvas(),
        this.bounds);
   }
   requires { public Object bounds(); }
}

class Circle {
   uses { TDrawing }
   ...
}
```

Resume

- Traits are a appropriate language construct to:
  - Achive conceptual consistent class hierarchies
  - Avoid code duplication
  - Enhance class compositon capabilites

- Available for different languages
- Two different flavors proposed for PHP

Request for Comments

Traits for PHP

Version:	1.5
Date:	2008-03-06
Author:	Stefan Marr
Wiki:	http://wiki.php.net/rfc/traits
reST:	http://www.stefan-marr.de/rfc-traits-for-php.txt
HTML:	http://www.stefan-marr.de/artikel/rfc-traits-for-php.html

Non Breakable Traits for PHP

Version:	1.0
Date:	2008-02-29
Author:	Joshua Thompson
Wiki:	http://wiki.php.net/rfc/nonbreakabletraits

Basic Literature

Main Article about Traits

[3] S. DUCASSE, O. NIERSTRASZ, N. SCHÄRLI, R. WUYTS, AND A. P. BLACK, *Traits: A Mechanism for Fine-Grained Reuse*, ACM Trans. Program. Lang. Syst., 28 (2006), pp. 331–388.

Traits Extensions

[1] A. BERGEL, S. DUCASSE, O. NIERSTRASZ, AND R. WUYTS, *Stateful Traits and their Formalization*, Journal of Computer Languages, Systems and Structures, 34 (2007), pp. 83–108.

[4] S. DUCASSE, R. WUYTS, A. BERGEL, AND O. NIERSTRASZ, *User-Changeable Visibility: Resolving Unanticipated Name Clashes in Traits*, SIGPLAN Not., 42 (2007), pp. 171–190.

Literature

[1] A. Bergel, S. Ducasse, O. Nierstrasz, and R. Wuyts, *Stateful Traits and their Formalization*, Journal of Computer Languages, Systems and Structures, 34 (2007), pp. 83–108.

[2] S. Denier, *Traits Programming with AspectJ*, RSTI - L'objet, 11 (2005), pp. 69–86.

[3] S. Ducasse, O. Nierstrasz, N. Schärli, R. Wuyts, and A. P. Black, *Traits: A Mechanism for Fine-Grained Reuse*, ACM Trans. Program. Lang. Syst., 28 (2006), pp. 331–388.

[4] S. Ducasse, R. Wuyts, A. Bergel, and O. Nierstrasz, *User-Changeable Visibility: Resolving Unanticipated Name Clashes in Traits*, SIGPLAN Not., 42 (2007), pp. 171–190.

[5] S. Marr and F. Menge, *ezcReflection*, SVN Repository, eZ Components, January 2008. http://svn.ez.no/svn/ezcomponents/experimental/Reflection.

[6] E. R. Murphy-Hill, P. J. Quitslund, and A. P. Black, *Removing Duplication from java.io: a Case Study using Traits*, in OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, New York, NY, USA, 2005, ACM, pp. 282–291.

Literature

[7] PROGRAMMING METHODS LABORATORY, *Traits for Scala*, Programming Language, Ecole Polytechnique Fédérale de Lausanne, 2006. http://www.scala-lang.org/intro/traits.html.

[8] P. J. QUITSLUND, E. R. MURPHY-HILL, AND A. P. BLACK, *Supporting Java traits in Eclipse*, in eclipse '04: Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange, New York, NY, USA, 2004, ACM, pp. 37–41.

[9] S. REICHHART, *Traits in C#*. Video of Talk at Microsoft Research, September 2005.

[10] N. SCHÄRLI, *Traits — Composing Classes from Behavioral Building Blocks*, PhD thesis, University of Berne, Feb. 2005.

[11] SOFTWARE COMPOSITION GROUP, *Traits for Squeak*, Smalltalk VM, University of Berne, December 2006. http://www.iam.unibe.ch/ scg/Research/Traits/index.html.

[12] L. WALL, *Apocalypse 12: Class Composition with Roles*, tech. report, The Perl Foundation, 2004. http://www.perl.com/pub/a/2004/04/16/a12.html?page=11.

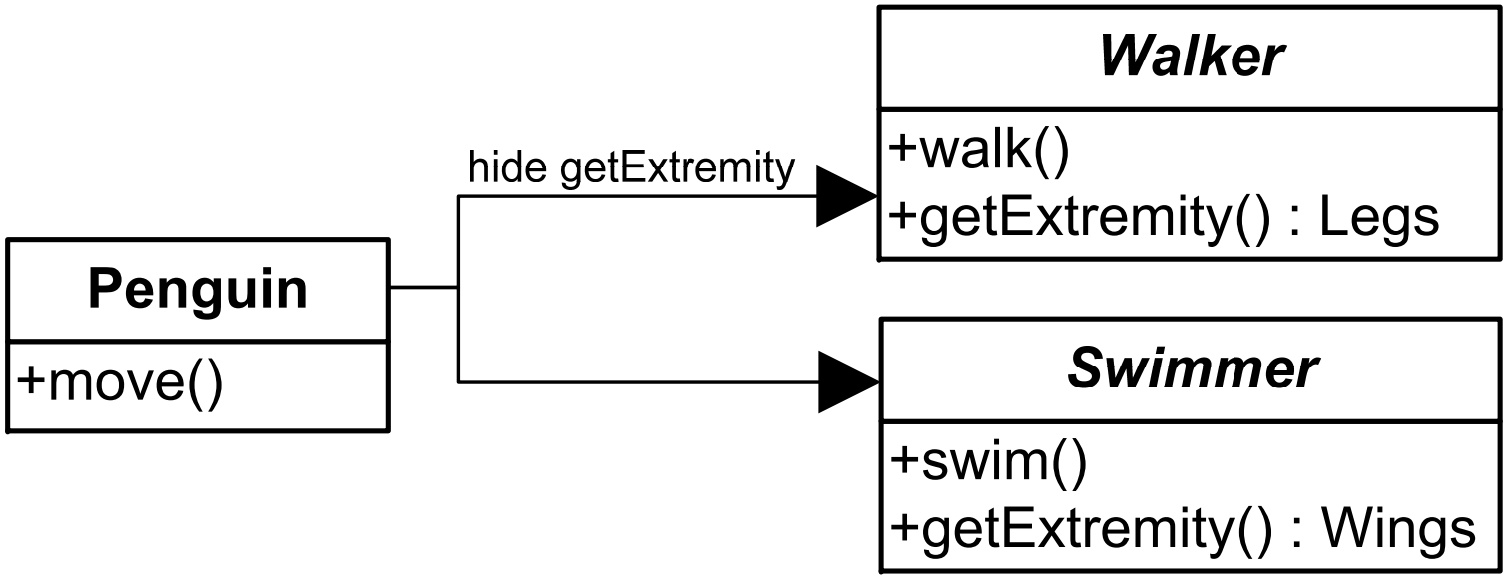
Any questions?
Discussion
- What du you think?
- Traits vs. Grafts

User-Changeable Visibility and Stateful Traits

TRAITS ENHANCED

User-Changeable Visibility

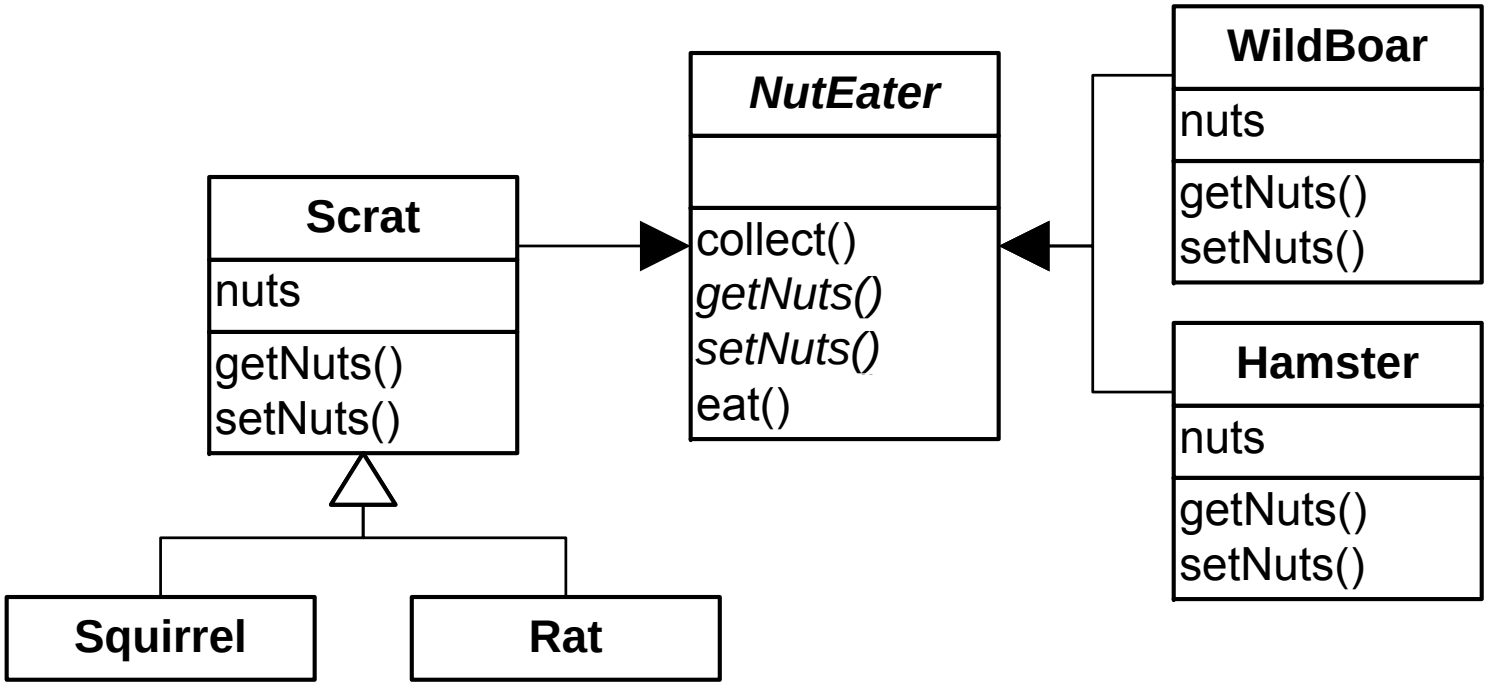
- Add flexibility to conflict handling
- Introduces “Trait private” methods
- Methods visibility changeable on composition



Stateless Traits

- Problem: stateless Traits are incomplete, not self-contained units of reuse
  - Required state accessors bloat the interfaces
  - Encapsulation is violated by public accessors
  - Reuse, composition is hampered
  - State access introduces fragility

Stateless Traits



Stateful Traits

- Solution requirements:
  - Preserver faltening property
  - Minimal and general approach
  - Language independed
- Solution
  - State is private to the Traits scope
  - Client can request access under new private name
  - Client can merge state

Stateful Traits