BETA and Newspeak
Seminar Module Systems, SS2015

Fabio Niephaus, Matthias Springer

Hasso Plattner Institute, Software Architecture Group

May 21, 2015
Overview

Introduction

Unification: The Pattern

Nested Classes

Summary
Introduction

- **BETA**: “A modern language in the Simula tradition”
  - Designed by Birger Møller-Pedersen and Kristen Nygaard
    *(Scandinavian School)*
  - Class = Method = Pattern
  - Nested Patterns

- **Newspeak**: “A new programming language in the tradition of Self and Smalltalk”
  - Designed by Gilad Bracha et al.
  - Nested Classes
  - No globals: all names are late bound
Overview

Introduction

Unification: The Pattern

Nested Classes

Summary
Patterns

- Classes and methods are patterns
- “Patterns [are] templates for generating objects (instances).”
- Objects are executable

Pattern: (#
  Declaration1; Declaration2; ...; DeclarationN
  enter InputArguments
  do Implementation
  exit OutputArguments
#)
Unification of Abstraction Mechanisms: The Pattern

- Instances of a procedure are procedure activations
- Instances of a class are objects
- Instances of a function are function activations
- Instances of a type are values
Handout only: Similarities between Objects and Procedure Activations

- Procedure activation = Activation record + execution of code
- Activation record is similar to object: data items and local procedures (nested procedures in languages with block structure)
Example

(#

Account: (# balance: @integer;

Deposit:

(# amount: @integer

   enter amount

   do balance+amount->balance

   exit balance

#);

Withdraw: (# ... #);
#);

account: @Account;

K1: @integer;

do

100->&account.Deposit;
50->&account.Withdraw->K1;
#)
Handout only: Beta Syntax

- (@ Type) for static references
- (^ Type) for dynamic references
- (& Type) for instance creation
- ([]) acquires a references instead of object execution
- (&Account []) returns a dynamic reference to a new account instance
Subpatterns
Specialization by Simple Inheritance

Resulting properties = inherited properties + new properties
Resulting behavior = inherited behavior + new behavior
Resulting arguments = inherited arguments + new arguments
Resulting results = inherited results + new results

• Method execution starts at base method: use inner to call specialized method
• Non-virtual pattern: entire type hierarchy has same pattern
• Virtual pattern: subtypes can have different patterns
• Pattern variables: every object can have a different pattern
Pattern (Design) Patterns

- **Procedure Pattern**: sequence of actions
- **Function Pattern**: sequence of actions with return value(s), does not change state
- **Class Pattern**: template for generating objects
Expected Benefits of Unification[6]

• Design goals
  - “The pattern mechanism should be the ultimate abstraction mechanism, subsuming all other known abstraction mechanisms.”
  - “The unification should be more than just the union of existing mechanisms.”
  - “All parts of a pattern should be meaningful, no matter how the pattern is applied.”

• “Uniform treatment of all abstraction mechanisms. [...] It ensures orthogonality among class, procedure, etc.”

• Functionality: subpatterns, virtual patterns, nested patterns, pattern variables
Overview

Introduction

Unification: The Pattern

Nested Classes

Summary
Nested Classes

What is it?

- Class defined inside another class
- Part-of relationship: nested classes belong to the enclosing class
- Access to enclosing class (lookup depends on programming language)
Nested Classes

Benefits

- **Namespace** for classes, avoiding name clashes
- Group together what belongs together: increase **understandability** and **readability**
- A form of **encapsulation**, promoting development towards an interfaces instead of an implementation (using visibility annotations)
- Support for more advanced features (e.g. Class Hierarchy Inheritance)
Nested Classes
Programming Languages

- Java: nested classes are non-virtual
- Ruby: inner classes/modules are non-virtual
- BETA, Newspeak: nested classes are virtual and can be overridden
BETA Nested Classes

- Virtual methods can be overridden in subclasses
- Virtual classes can be overridden in subclasses
- Virtual patterns can be overridden in subclasses
BETA Nested Classes

Reservation: (#
    date: @Date;
    Display:< (# do date.PrintToConsole; INNER; #)
#)

TrainReservation: Reservation (#
    seat: @Seat;
    Display::< (# do seat.PrintToConsole; INNER; #)
#)

(#
    reservation: ^Reservation;
    do
    &reservation.Display
#)
Handout only: BETA Nested Classes

Reservation: (#
    date: @Date;
    DisplayReservation: (# do date.PrintToConsole; INNER; #)
    Display:< DisplayReservation
#)

TrainReservation: Reservation (#
    seat: @Seat;
    DisplayTrainReservation: DisplayReservation (#
        do seat.PrintToConsole; #)
        Display::< DisplayTrainReservation
#)
• Only virtual patterns can be overridden (denoted by :<)
• Overriding pattern must be a subpattern of superpattern
• Pattern execution starts with base pattern (inner instead of super)
Example: Nested Classes in Newspeak [2]
Methods: instance methods

Classes: nested class definitions

Slots: instance variables

Module Definition: a class object that acts as a module
  - has to be a top-level class
  - has its own namespace which is represented by a platform object
  - is stateless
  - its external dependencies are listed at the top of the class

Module: an instance of a module definition
platform Object instead of Global Namespace\[8\]

- **platform** contains references to top-level modules required by the application (mapping identifiers to top-level modules)
  - Provided by the system: collections, file system, drawing, kernel classes, . . .
  - Provided by the developer: custom libraries
  - Contains all dependencies required for deployment

- Created using IDE support, then object graph serialization
- **Platform>>main:args:** is the application’s entry point
- **platform** is similar to Squeak environments
Instance Creation in Newspeak

Nested Classes in a Nutshell for Smalltalkers

1. Message send to class object: invoke factory method (e.g. usingPlatform: or new)
2. Execute factory method: might initialize some slots
3. Generate class objects for nested classes lazily (s.t. optimizations)

```
Object subclass: #CombinatorialParsing
    instanceVariableNames: 'CombinatorialParser SequentialParser
... OrderedCollection LinkedList parent platform'.

CombinatorialParser class>>usingPlatform: platform
| inst | inst := self new.
    inst OrderedCollection: platform collections OrderedCollection.
    inst LinkedList: platform collections LinkedList.
^ inst

CombinatorialParsing>>StarParser
| nested | "important: nested is cached"
    nested := self CombinatorialParser subclass: #StarParser
        instanceVariableNames: 'parent subparser'.
    nested compile: 'parse: input ^ ...'
^ nested
```
Example: Class Hierarchy Inheritance

```
ShapeLibrary

Shape

Circle

Rectangle

ExtendShapes

?  

COLOR ShapeLibrary

Shape

COLOR Shape

Circle

Rectangle
```
Handout only: Example: Class Hierarchy Inheritance

- **ShapeLibrary**: a library for geometrical shapes, containing classes `Shape`, `Circle`, and `Rectangle`.
- `Shape` is superclass of `Circle` and `Rectangle`, and provides basic rendering functionality.
- **Challenge**: provide a module `ExtendShapes` which takes as input `I` any `ShapeLibrary` and generates a modified `ColorShapeLibrary` where `ColorShapeLibrary.Shape` has additional behavior for drawing colors.
  - `I` must have a nested class `Shape`.
  - Override `ColorShapeLibrary.Shape` with a new class whose superclass is `I.Shape` (like method overriding).
Example: Class Hierarchy Inheritance in Newspeak [1]
Object subclass: \#ShapeLibrary
  
  instanceVariableNames: 'Shape Circle Rectangle List Error Point'.

ShapeLibrary >> Rectangle

| nested | "nested is cached"

nested := self Shape subclass: #Rectangle ...

... ^ nested

Object subclass: \#ExtendShapes

instanceVariableNames: 'ShapesLibrary ColorShapeLibrary'.

ExtendShapes class >> withShapes: shapes

| inst | inst := self new.

inst ShapesLibrary: shapes.

^ inst

ExtendedShapes >> ColorShapeLibrary

| nested | "nested is cached"

nested := self ShapesLibrary subclass: #ColorShapeLibrary

  class compile: 'usingPlatform: platform ^ ...

nested compile:

Shape | nested | "nested is cached" nested := super Shape subclass: #Shape. "add behavior to nested"

^ nested
Handout only: Why Nested Classes are lazily initialized

- Consider nested classes are initialized in factory.

- `ExtendedShape`>`ColorShapeLibrary` class>`usingPlatform`: triggers `ShapeLibrary` class>`usingPlatform`: (super constructor call).

- `ShapeLibrary` class>`usingPlatform`: creates `Shape`, `Circle`, `Rectangle`.

- `ExtendedShape`>`ColorShapeLibrary` class>`usingPlatform`: creates (overrides) a new `Shape` class.

- **Problem**: `Circle` and `Rectangle` are still subclasses of the old `Shape` class.

- **Solution**: all names are late bound and method calls. The method call `Shape` creates the class on demand the superclass factory runs the subclass implementation (overridden).
Method Lookup in Newspeak \[1\]
Handout only: Method Lookup in Newspeak

- First enclosing classes (*lexical chain*)
- Then superclass hierarchy
- Never check superclass hierarchy of enclosing classes
- Different from BETA and Java: *comb semantics*
  - Check receiver class and superclass hierarchy
  - Check enclosing classes and superclass hierarchies
Newspeak Avoids Method Name Clashes by Superclasses

Example

class Super {
    //int m(){ return 42; }
}
class Outer {
    int m(){ return 91; }
    class Inner extends Super {
        int foo(){ return m(); }
    }
}

new Outer.Inner().foo()?
Poor Man’s Nested Classes\cite{4}

Classes as First Class Objects

- Scoping rules are different
- No convenient access to enclosing instance
- Hierarchy not reflected in the source code
- Bad tooling support
- No class hierarchy inheritance
Overview

Introduction

Unification: The Pattern

Nested Classes

Summary
Summary

- Pattern = Method = Class
- Nested patterns/classes: work like virtual methods in other programming languages
- More than Java nested classes: Java nested classes are not virtual
- Workaround for nested classes in other programming languages: factory
- No global namespace in Newspeak: platform object provides all dependencies
- Newspeak: all names are late bound
References