Undeclared Variables

How and why and what are the problems?
Globals in ST80

- Globals and Class Variables are Associations
  - from Smalltalk global dictionary
  - from classPool dictionary of a class
- Associations of Pool Dictionaries
  - SharedPool subclasses since Squeak (Associations from the classPool)
Bytecode

• pushLiteralVariable: x

• storeIntoLiteralVariable: x

• “push the ivar 2 of the object in literal frame at x”

• No need to be an Association (VM does not check)
Undeclared: Why?

- Problem: compiling code with Variables not yet defined

- Examples:
  - Load users of a class before the class (code load)
  - remove ivar/classVar, subclasses are recompiled (during development)
Undeclared

- We have a global variable “Undeclared” that points to a Dictionary
- Undeclared is an entry #name-> nil
  - It is an Association in the dict (aka “binding”)
- Every using method points to this one Association instance

- Reading Undeclared: reads nil
- Writing: writes to the value
What happens if we define?

• If a global is defined
  • add class var
  • add global
• We take the binding (Association) from the Undeclared, change the value
  • All *users* of the Variable are changed at the same time
    • as they use the same object
• We call that “Undeclared repair”
Problems of Repair

• Only works for Globals
  • ivars / temps use other bytecode *and* do not have just one value
  • Here we recompile

• Wrong for Class Variables
  • Repair does not care about lookup semantics
Undeclared Repair: Ignores Semantics

- create a class with one method that returns a Global that does not exist (in the menu "Leave Variable Undeclared")
- create a second class with a Class Variable with the same name
- set this to some value
- now call the method in the first class that returns the unknown global
- You will see that the Undeclared repair changed the variable there to point to the class variable. A Variable that is not now know to that class...
Undeclared: Never cleaned up

- Undeclared Dictionary is cleaned when a repair is done for that one name
- But as temps/ivars are repaired by recompile, they stay forever
  - We would need to scan the whole system to be sure, which is too slow
- Explicit cleanup: #cleanOutUndeclared
  - Called by cleanUp and release cleaner
Undeclared clean: Iterate all methods

- Undeclared repair iterates over all installed methods
  - Slow
  - e.g. we used to scan when removing a class. Far too slow!

- But worse: a method not installed with an Undeclared is cut off from repair
  - It will have the Undeclared forever, even if the var gets defined (!!)
**Undeclared: They read nil**

- Problem: there is no way to hook into Undeclared read or write
- It is just “pushLiteralVariable”
- No way to ask at runtime “Do you want to define a class?”

- Pharo: we added a hack to improve TDD flow
  - #doesNotUnderstand: on UndefinedObject checks reflectively if the DNU is due to reading a undeclared global
Pharo: First Class Variables

- We do not use Associations anymore
- GlobalVariable / ClassVariable / UndeclaredVariable, subclass of Variable
- Second ivar is the value, thus #pushLiteralVariable bytecode works
- Implement #key and #value protocol, thus they can play the role of Associations in Dictionaries
- More state. For ClassVariable: defining class
Pharo: Undeclared repair

• Undeclared repair can not just re-use the binding

• We need to change the class
  • Undeclared -> Global (changeClass)
  • Undeclared -> ClassVariable (become, as ClassVar has more ivars)
Summary Pharo11

• ST80, but Variables instead of Associations
  • needs become:/changeClass to repair, not nice

• Hook into Undeclared read by an ugly #doesNotUnderstand: hack
  • Presents user with a dialog to define the missing Variable
Pharo12: #undeclaredVariableRead

- UndeclaredVariable code generation was changed to send a message to the Variable

- #runtimeUndeclaredRead/ #runtimeUndeclaredWrite:

  ```plaintext
  emitValue: aMethodBuilder
  aMethodBuilder
  pushLiteral: self;
  send: #runtimeUndeclaredRead
  ```
Pharo12: #undeclaredVariableRead

- For now: raises exception
  - We can now e.g. present user interaction instead of reading nil in interactive mode

- is just a reflective read for GlobalVariable/ClassVariable (after repair)

- We recompile on read and write to gain speed
Pharo12: Undeclared Repair Summary

• First Class Variables, as in Pharo11
  • ST80 style Undeclared repair, but using become:
    • Undeclared compile send #runtimeUndeclaredRead
      • We can now e.g. present user interaction instead of reading nil
      • We removed the #doesNotUnderstand: hack
Problems of #undeclaredVariableRead

• We still rely on the undeclared repair and #cleanOutDeclared
  • With all the problems noted

• Mapping ast->bytecode
  • We create a new AST and re-compile to get the mapping
  • But Undeclared was defined
  • This the method used to create bc-ast mapping is out-of-sync with the method we need the mapping for
Idea (without VM Change)

• We can understand UndeclaredVariable as “late bound Variable”
  • read/write re-lookup, delegate to result if not Undeclared
  • runtime read/write would need to know the context (and forward readInContext:)

• Remove become: magic. Undeclared stay Undeclared

• Repair by re-compile

• Not installed methods: compiler takes undeclareds from the prior method, uses overlay environment to force to use them for that compile, shadowing the now defined names
Properties

- Negative: Need compiler
  - But we need it already for undeclared ivars and temps
- Repair is semantically correct (see problem shadowing ClassVariable)
- Mapping is correct for non-installed methods
- Undeclared can be a weak set (we never need to clean up manually)
  - No need for a global variable (could be class var in UndeclaredVariable)
- no need for #become:
- no need for #cleanOutUndeclared
  - not-installed methods will never be “cut off”