Design Principles for the Grace AST

Andrew P. Black
Portland State University
&
Victoria University of Wellington

Portland State
UNIVERSITY

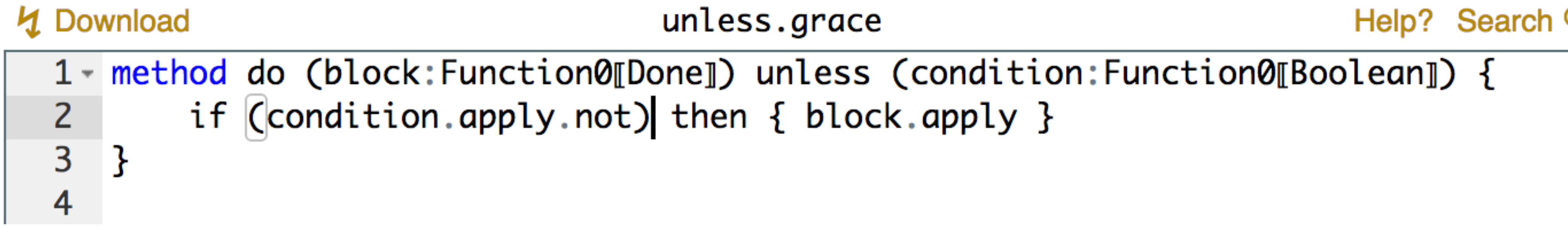
Why do we care?

- Isn’t the AST an internal part of the implementation?
- Why should its design me worth public debate?

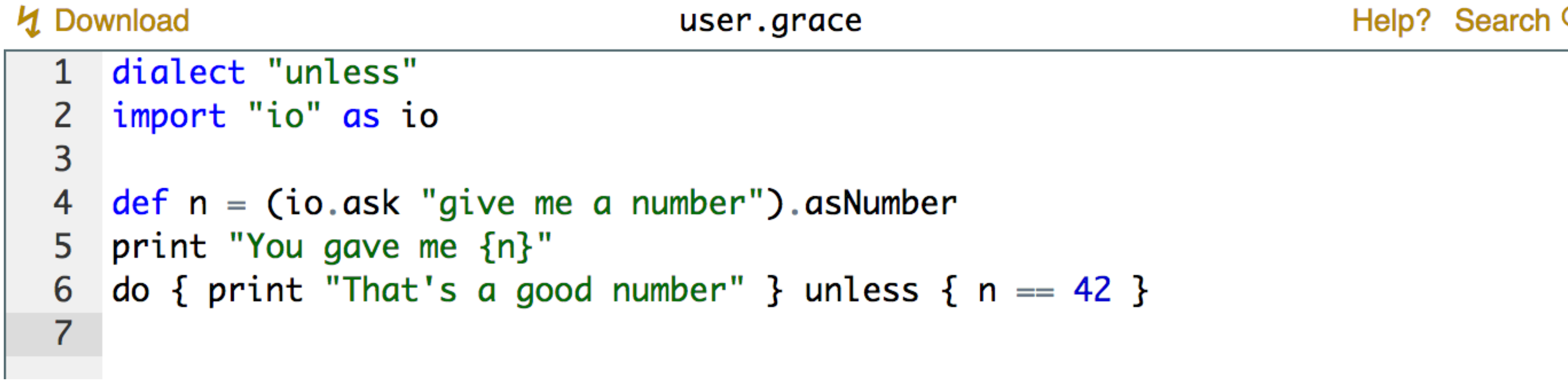
Dialects

- Grace has dialects—variant languages designed to support a specific teaching (or other) objective
  - parallel programming, graphics, security ...
- Dialects can *extend* Grace by defining methods

Example



```
method do (block:Function0⟦Done⟧) unless (condition:Function0⟦Boolean⟧) {
    if (condition.apply.not) then { block.apply }
}
```



```
dialect "unless"
import "io" as io

def n = (io.ask "give me a number").asNumber
print "You gave me {n}"
do { print "That's a good number" } unless { n == 42 }
```

Dialects

- Grace has dialects—variant languages designed to support a specific teaching objective
- Dialects can *extend* Grace by defining methods
- Dialects can *restrict* Grace by defining a checker that walks an AST representing the dialect user's code, and generates error messages

Example

```
def thisDialect is public = object {
    method parseChecker (moduleObj) {
        moduleObj.accept(bsVisitor)
    }
}


def bsVisitor = object {
    inherit ast.baseVisitor
    method asString {
        "the beginningStudent visitor"
    }

    method visitArray(v) -> Boolean {
        DialectError.raise("square brackets are not used in this dialect; " ++
              "for a list, use list(_, _, ... )") with (v)
        false
    }
    method visitVarDec(v) -> Boolean {
        def name = v.nameString
        if (false == v.dtype) then {
            DialectError.raise "no type given to var '{v.nameString}'"
                with (v.name)
        }
        if (unicode.inCategory(name, "Lu")) then {
            DialectError.raise("by convention, variables start " ++
                "with a lower-case letter") with (v.name)
        }
        true
    }
    ...
```

Consequences

- The author of a dialect must know enough about the AST to write a simple tree-walker, examine the dialectical module, and generate error messages.
- Hence, the AST is (to some extent) part of the Grace language definition.

Portland State UNIVERSITY

What is an AST, anyway?

- **A**bstract **S**yntax **T**ree
- It’s a **tree**, that represents the **syntax** of a program
- It’s **abstract**, in the sense that it contains just the information needed for *your* particular purpose
  - less information than the full parse tree

What’s the Problem?

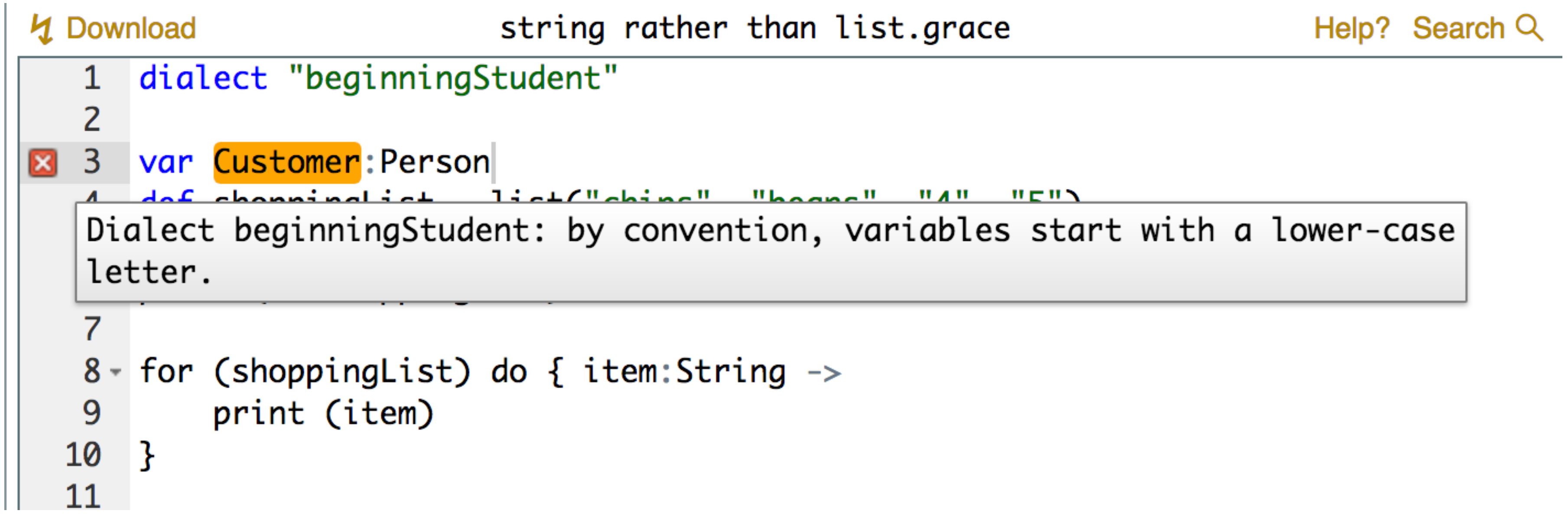
- We don’t know the “particular purpose” of a dialect-writer
  - dialects are as varied as courses in computing,
  - or other purpose to which Grace might be put

Portland State UNIVERSITY

What do we know (1)?

- Dialects produce error messages
  - Example:

```
if (unicode.inCategory(name, "Lu")) then {
    DialectError.raise("by convention, variables start " ++
        "with a lower-case letter") with (v.name)
}
```



string rather than list.grace

```
dialect "beginningStudent"

var Customer:Person
```

```
Dialect beginningStudent: by convention, variables start with a lower-case
letter.
```

```

for (shoppingList) do { item:String ->
    print (item)
}

```

Principles

1. The AST must provide access to exact source-code ranges

Portland State UNIVERSITY

What do we know (2)?

- Dialects are dialects of Grace!
- Grace has, by design, certain properties
- Dialect-writers probably want to exploit those properties
- Example:
    - each variables has a unique defining occurrence, which can be determined statically

Principles

1. The AST must provide access to exact source-code ranges

2. Information deducible by the compiler should be accessible through the AST

    - does not imply that it's pre-computed

What do we know (3)?

- The dialect may be grouping syntax in varied ways
- Example: def and var declarations

	def	var
object	fieldDef	fieldVar
block or method	tempDef	tempVar

Portland State UNIVERSITY

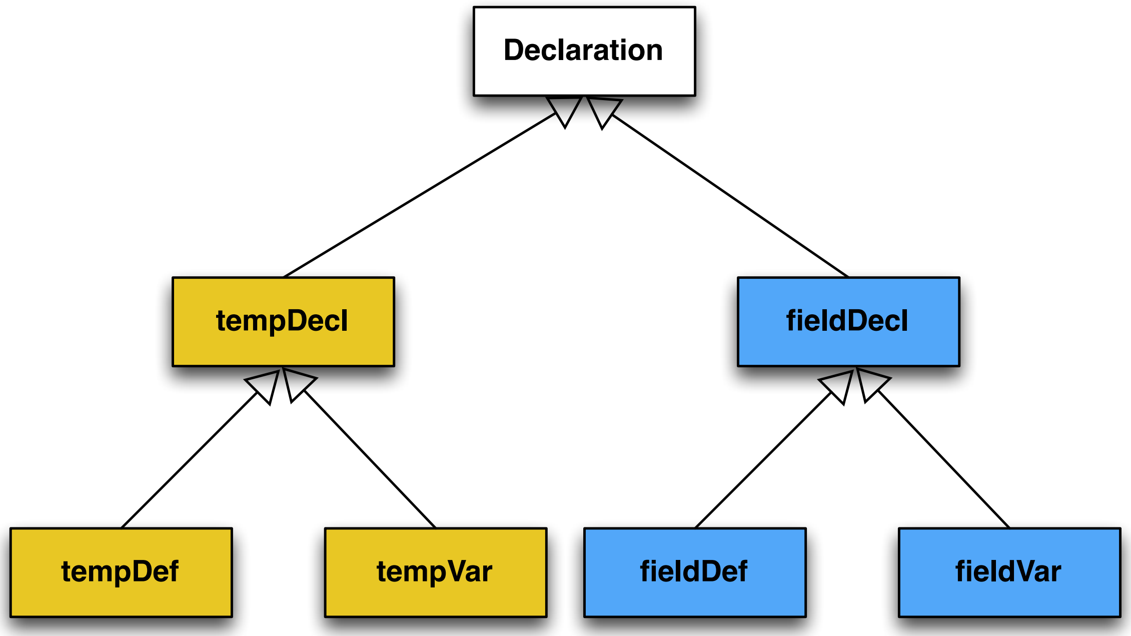
What do we know (3)?

- The dialect may be treating syntax in varied ways
- Example: def and var declarations

	def	var
object	fieldDef	fieldVar
block or method	tempDef	tempVar

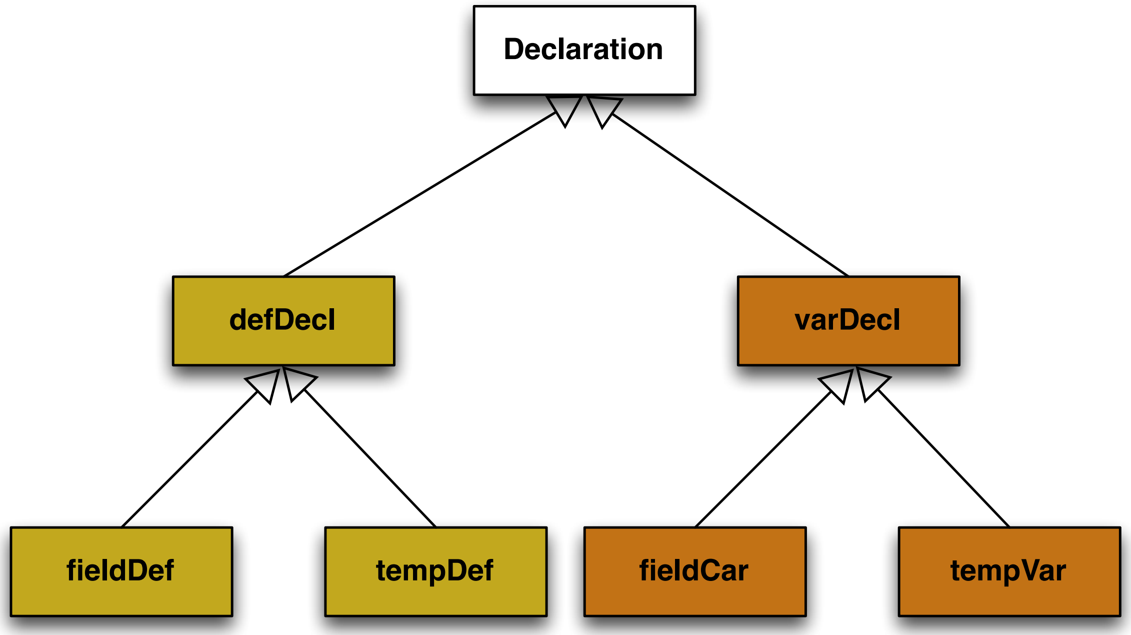
Portland State UNIVERSITY

- First discussion (with Richard Roberts):



- Rationale:
  - *compilation* of temps and fields will be different, *compilation* of defs and vars will be similar

- Second thoughts (implementation):



- Rational:
  - *content* of defs and vars will be different, *content* of temps and fields will be similar

- Third thoughts:
  - Visitors are not object-oriented!
- Visitors expose the class hierarchy
  - This is an implementation detail that ought to be hidden
- The public interface does *not* include the implementation classes
  - only the *interfaces* should be public

Two approaches

1. ~~Class-Based discrimination~~
   - ~~e.g., visitors~~
   - ~~`visitField`, `visitTemp`, *vs.* `visitDef`, `visitVar`: can't be combined~~
2. Predicate-based discrimination ✓
   - `isDef`, `isVar`, `isField`, `isTemp`: easy to combine

Instead of...

```
method visitArray(v) -> Boolean {
    ...
}
method visitVarDec(v) -> Boolean {
    ...
}
method visitDefDec(v) -> Boolean {
    ..
}
```

Prefer:

```
method visitNode(v) -> Boolean {
    if (v.isArray) then {
        ...
    } elseif (v.isVarDec) then {
        ...
    } elseif (v.isDefDec) then {
        ...
    }
    ...
}
```

Principles

1. The AST must provide access to exact source-code ranges
2. Information deducible by the compiler should be accessible through the AST
    1. does not imply that it's pre-computed
3. Provide predicates to distinguish syntactic elements; don't force the dialect writer to use a visitor

Questions

- Should we even *allow* the dialect writer to write a visitor?
- Is abstraction important? Or is an AST just a data structure?