GOOL: A Generic Object-Oriented Language

Jacques Carette, Brooks MacLachlan, Spencer Smith

Computing and Software Department
Faculty of Engineering
McMaster University

PEPM 2020

McMaster University

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Observations

OO languages:

- Conceptually similar

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Observations

OO languages:

- Conceptually similar
- Mainly *shallow* syntactic differences

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Observations

OO languages:

- Conceptually similar
- Mainly *shallow* syntactic differences
- Like Romance languages

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Observations

OO languages:

- Conceptually similar
- Mainly *shallow* syntactic differences
- Like Romance languages
- We tend to say similar things in all of them

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

The Goal

One language to express them all.

- Is it possible?
- Capture the meaning of OO programs
- DSL for domain of OO programs
- Currently targets Java, Python, C#, C++

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Requirements

- **mainstream**: Most potential users

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Requirements

- **mainstream**: Most potential users
- **readable**: Human beings are a target audience

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Requirements

- **mainstream**: Most potential users
- **readable**: Human beings are a target audience
- **idiomatic**: For readability, understandability

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Requirements

- **mainstream**: Most potential users
- **readable**: Human beings are a target audience
- **idiomatic**: For readability, understandability
- **documented**: For readability, understandability

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Requirements

- **mainstream**: Most potential users
- **readable**: Human beings are a target audience
- **idiomatic**: For readability, understandability
- **documented**: For readability, understandability
- **patterns**: More efficient coding

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Requirements

- **mainstream**: Most potential users
- **readable**: Human beings are a target audience
- **idiomatic**: For readability, understandability
- **documented**: For readability, understandability
- **patterns**: More efficient coding
- **expressivity**: Works for real examples

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Requirements

- **mainstream**: Most potential users
- **readable**: Human beings are a target audience
- **idiomatic**: For readability, understandability
- **documented**: For readability, understandability
- **patterns**: More efficient coding
- **expressivity**: Works for real examples
- **common**: Reduce code duplication

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Approach

Start from real OO programs

McMaster University

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Approach

Start from real OO programs

What we can say vs. **want to say** vs. need to say

Readability Features

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Example: Blocks

- Semantically meaningless
- Reflect how people write programs

```
# Initialize dimensions
h = input('What is the height? ')
w = input('What is the width? ')
d = input('What is the depth? ')

# Calculate volume and surface area
v = h * w * d
sa = 2 * (h * w + w * d + h * d)
```

McMaster University

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Some ingredients

- Variables distinct from values (viz use/mention)
- Smart constructors for common idioms

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

GOOL Language

Types	`bool`, `int`, `float`, `char`, `string`, `infile` (read mode), `outfile` (write mode), `listType`, `obj`
Variables	`var`, `extVar`, `classVar`, `objVar`, `$->` (infix operator for `objVar`), `self`, [`listVar`]
Values	`valueOf` (value from variable), `litTrue`, `litFalse`, `litInt`, `litFloat`, `litChar`, `litString`, `?!`, `?&&`, `?<`, `?<=`, `?>`, `?>=`, `?==`, `?!=`, `#~`, `#/^`, `#—`, `#+`, `#-`, `#*`, `#/`, `#^`, `inlineIf`, `funcApp`, `extFuncApp`, `newObj`, `objMethodCall`, [`selfFuncApp`, `objMethodCallNoParams`]
Statements	`varDec`, `varDecDef`, `assign`, `&=`, `&+=`, `&-=`, `&++`, `&~-`, `break`, `continue`, `returnState`, `throw`, `free`, `comment`, `ifCond`, `ifNoElse`, `switch`, `for`, `forRange`, `forEach`, `while`, `tryCatch`, `block`, `body` [`bodyStatements` (single-block body), `oneLiner` (single-statement body)]
List API	`listAccess`, `at` (same as `listAccess`), `listSet`, `listAppend`, `listIndexExists`, `indexOf`, `listSlice`
Scope	`public`, `private`
Binding	`static_`, `dynamic_`
Functions	`function`, `method`, `param`, `pointerParam`, `mainFunction`, `docFunc`, [`pubMethod`, `privMethod`]
State Variables	`stateVar`, `constVar`, [`privMVar`, `pubMVar` (dynamic), `pubGVar` (static)]
Classes	`buildClass`, `docClass`, [`pubClass`, `privClass`]
Packages	`buildModule`, `fileDoc`, `docMod`, `prog`, `package`, `doxConfig`, `makefile`

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Encoding

Tagless with type families – 2 Layers of abstraction

1. Over target language
2. Over language-specific representational data structures

```
class (TypeSym repr) => VariableSym repr where
  type Variable repr
  var  ::  Label -> repr (Type repr)
    -> repr (Variable repr)
```

McMaster University

Introduction

Requirements

Creation

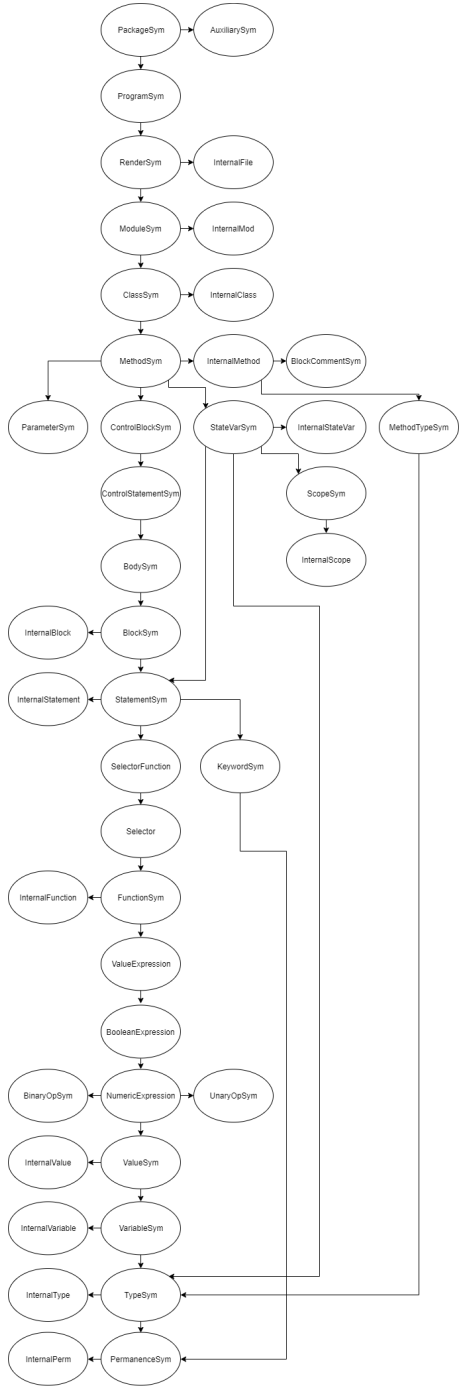
Implementation

Patterns

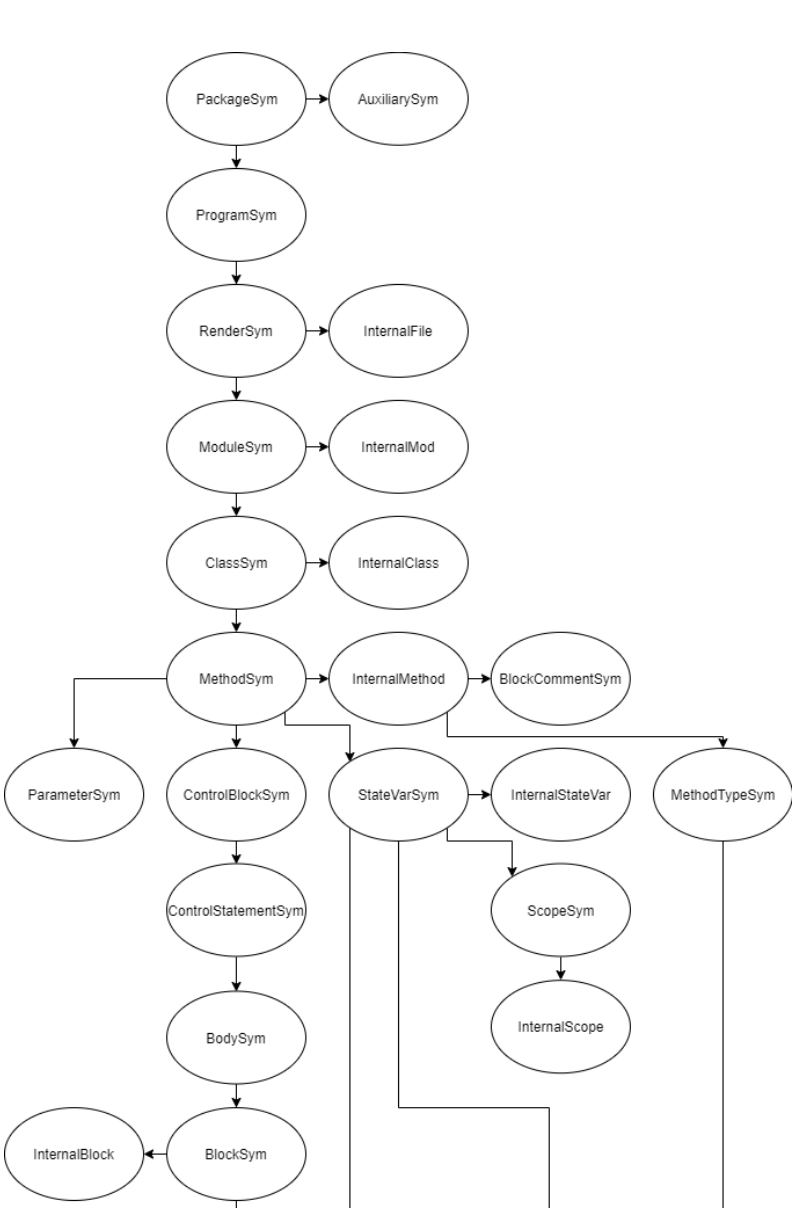
Example

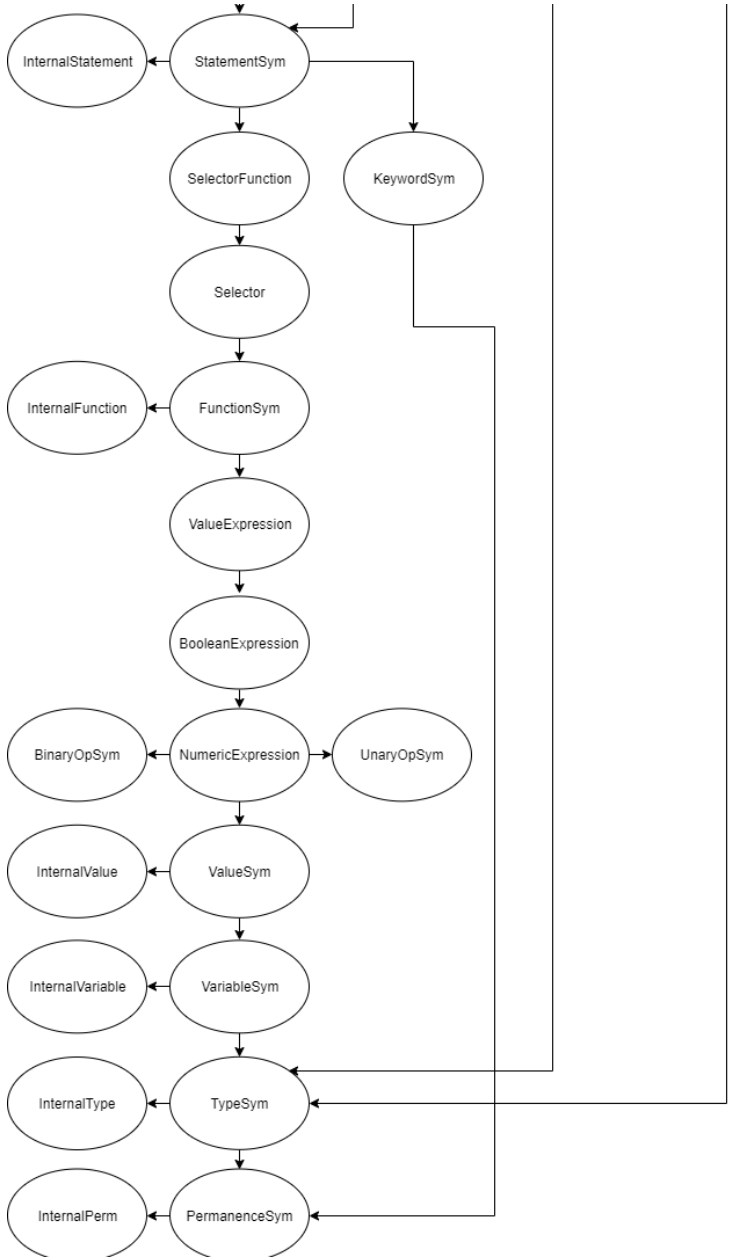
Current Work

Conclusions



Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

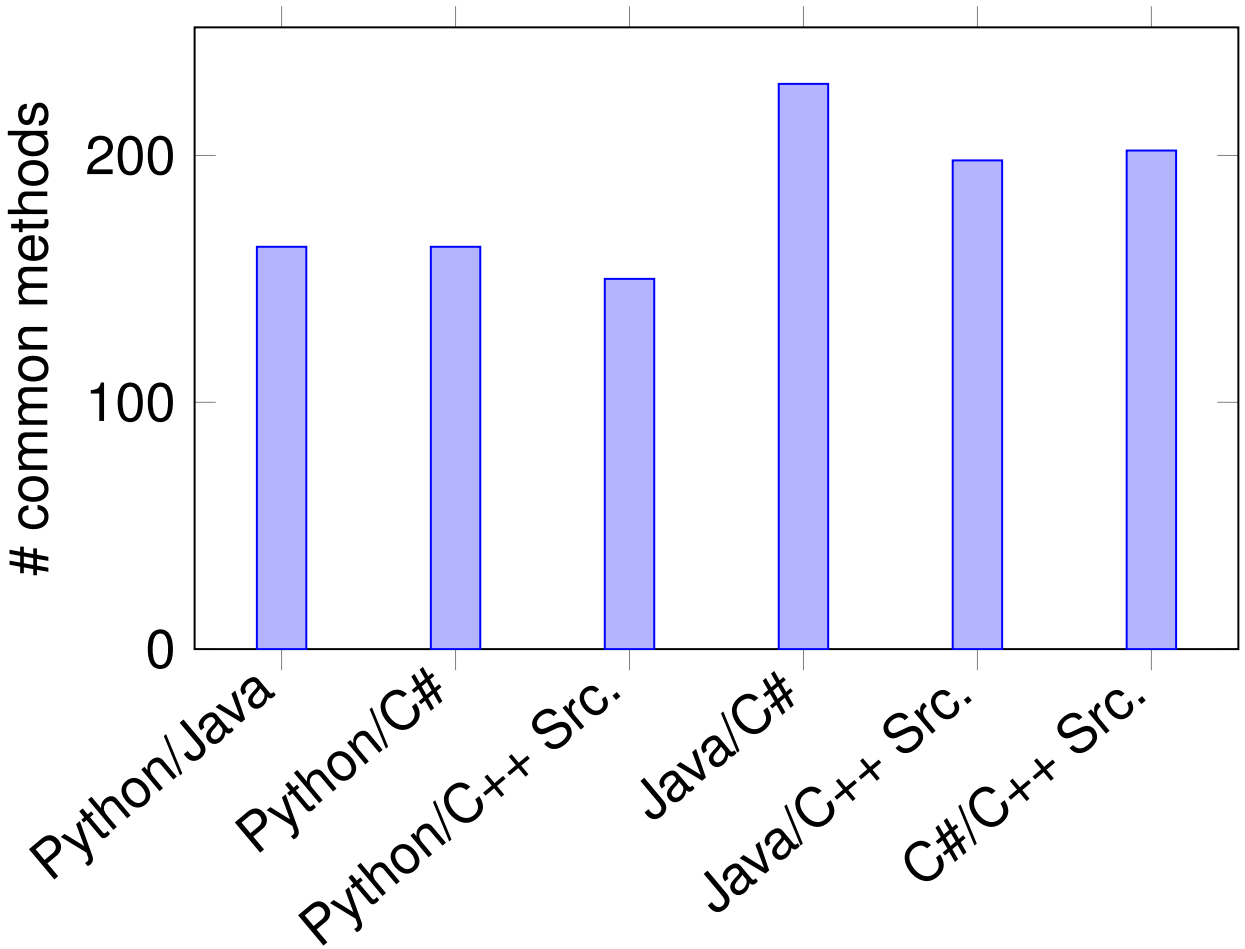




Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Statistics

- 43 classes, 328 methods
- 300 functions that abstract over commonalities
- 40% more common methods between Java and C# than Java and Python



Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Things we need/want to say

- Command line arguments
- Lists
- I/O
- Procedures with Input/Output/Both parameters
- Getters and setters
- Design patterns

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Example: List Slicing

GOOL: *hideous, we know*

```
listSlice sliced (valueOf old) (Just $ litInt 1)
  (Just $ litInt 3) Nothing
```

Python:

```
sliced = old [1:3:]
```

Example: List Slicing

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

GOOL: *hideous, we know*

```
listSlice sliced (valueOf old) (Just $ litInt 1)
  (Just $ litInt 3) Nothing
```

Java:

```
ArrayList<Double> temp = new ArrayList<Double>(0);
for (int i_temp = 1; i_temp < 3; i_temp++) {
    temp.add(old.get(i_temp));
}
sliced = temp;
```

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Example: List Slicing

GOOL: *hideous, we know*

```
listSlice sliced (valueOf old) (Just $ litInt 1)
  (Just $ litInt 3) Nothing
```

C#:

```
List<double> temp = new List<double>(0);
for (int i_temp = 1; i_temp < 3; i_temp++) {
    temp.Add(old[i_temp]);
}
sliced = temp;
```

Example: List Slicing

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

GOOL: *hideous, we know*

```
listSlice sliced (valueOf old) (Just $ litInt 1)
  (Just $ litInt 3) Nothing
```

C++:

```
vector<double> temp(0);
for (int i_temp = 1; i_temp < 3; i_temp++) {
    temp.push_back(old.at(i_temp));
}
sliced = temp;
```

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Example: Setters

GOOL:

```
setMethod "FooClass" foo
```

Python:

```
def setFoo(self, foo):
    self.foo = foo
```

Example: Setters

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

GOOL:

```
setMethod "FooClass" foo
```

Java:

```
public void setFoo(int foo)  throws Exception {
    this.foo = foo;
}
```

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Example: Setters

GOOL:

setMethod ”FooClass” foo

C#:

```
public void setFoo(int foo) {
    this.foo = foo;
}
```

Example: Setters

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

GOOL:

setMethod ”FooClass” foo

C++:

```
void FooClass::setFoo(int foo) {
    this->foo = foo;
}
```

Complete Example

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

As part of Drasil, we can look at the Projectile program

Design 1

- Documented
- Bundled inputs

Design 2

- Logging
- More modular

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Currently working on

- More types
- Smarter generation - ex. import statements
- Interfacing with external libraries
- User-decisions - ex. which type to use for lists?
- More patterns

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Language of Design

Drasil project - http://github.com/JacquesCarette/Drasil

- Generate scientific software
- Design language lets users guide design
- GOOL is the backend

Introduction
Requirements
Creation
Implementation
Patterns
Example
Current Work
Conclusions

Conclusion: it works

GOOL used to generate some scientific software (glass breakage, projectile simulation)

Together new:

- Idiomatic code generation
- Human-readable, documented code generation
- Coding patterns are language idioms