

# GOOL: A Generic Object-Oriented Language

Jacques Carette, Brooks MacLachlan, Spencer Smith

Computing and Software Department  
Faculty of Engineering  
McMaster University

PEPM 2020

Introduction

Requirements

Creation

Implementation

Patterns

Conclusions

OO languages:

- Structurally similar

Introduction

Requirements

Creation

Implementation

Patterns

Conclusions

OO languages:

- Structurally similar
- Mainly *shallow* syntactic differences

Introduction

Requirements

Creation

Implementation

Patterns

Conclusions

OO languages:

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

Introduction

Requirements

Creation

Implementation

Patterns

Conclusions

OO languages:

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

# 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

Conclusions

- **mainstream:** Most potential users

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



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

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

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

- **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

- **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

# Approach

Introduction

Requirements

**Creation**

Implementation

Patterns

Conclusions

Start from real OO programs

# Approach

Introduction

Requirements

**Creation**

Implementation

Patterns

Conclusions

Start from real OO programs

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

# Readability Features

Introduction

Requirements

**Creation**

Implementation

Patterns

Conclusions

## Example: Blocks

- Semantically meaningless
- Reflect how people write programs

[example code would really help here —JC]



# Some ingredients

Introduction

Requirements

**Creation**

Implementation

Patterns

Conclusions

- Variables distinct from values (viz use/mention)

# Some ingredients

Introduction

Requirements

**Creation**

Implementation

Patterns

Conclusions

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

# GOOL Language

Introduction

Requirements

Creation

Implementation

Patterns

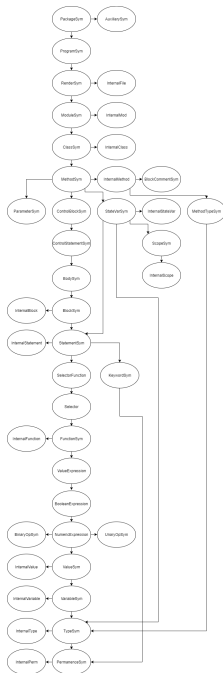
Conclusions

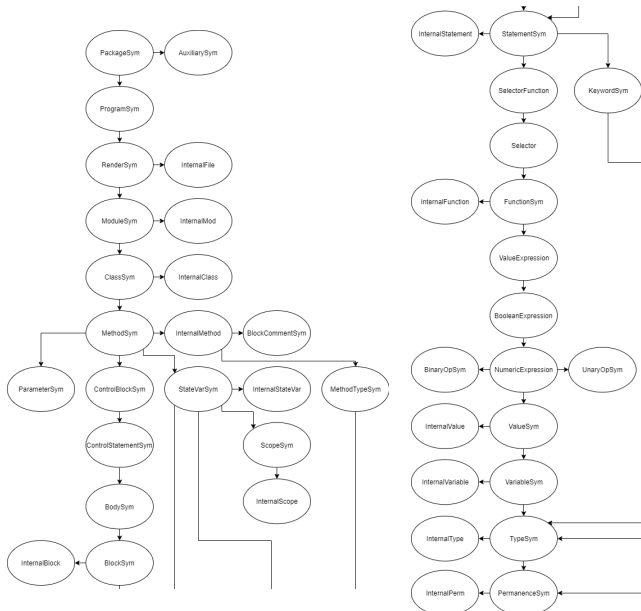
Types	bool, int, float, char, string, infile ( <b>read mode</b> ), outfile ( <b>write mode</b> ), listType, obj
Variables	var, extVar, classVar, objVar, \$-> ( <b>infix operator for objVar</b> ), self, [listVar]
Values	valueOf ( <b>value from variable</b> ), litTrue, litFalse, litInt, litFloat, litChar, 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 ( <b>single-block body</b> ), oneLiner ( <b>single-statement body</b> )]
List API	listAccess, at ( <b>same as</b> 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 ( <b>dynamic</b> ), pubGVar ( <b>static</b> )]
Classes	buildClass, docClass, [pubClass, privClass]
Packages	buildModule, fileDoc, docMod, prog, package, doxConfig, makefile

## Tagless with type families – 2 Layers of abstraction

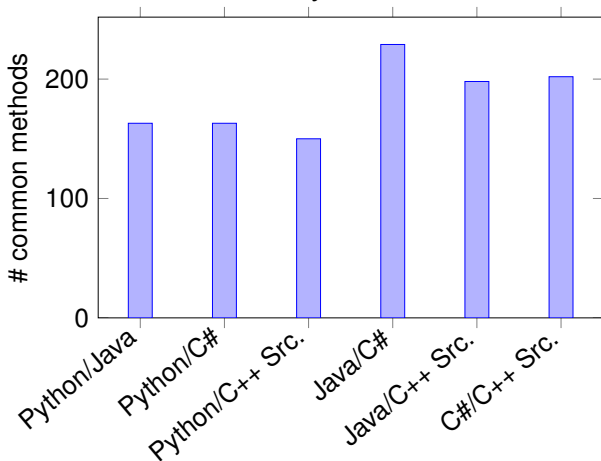
- ① Over target language
- ② Over language-specific representational data structures

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





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



# Things we need/want to say

Introduction

Requirements

Creation

Implementation

Patterns

Conclusions

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



## Example: List Slicing

Introduction

Requirements

Creation

Implementation

Patterns

Conclusions

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

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;
```

## Example: List Slicing

Introduction

Requirements

Creation

Implementation

Patterns

Conclusions

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

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;
```

## Example: Setters

Introduction

Requirements

Creation

Implementation

**Patterns**

Conclusions

GOOL:

setMethod "FooClass" foo

Python:

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

## Example: Setters

Introduction

Requirements

Creation

Implementation

**Patterns**

Conclusions

GOOL:

setMethod "FooClass" foo

Java:

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

## Example: Setters

Introduction

Requirements

Creation

Implementation

**Patterns**

Conclusions

GOOL:

setMethod "FooClass" foo

C#:

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

## Example: Setters

Introduction

Requirements

Creation

Implementation

Patterns

Conclusions

GOOL:

setMethod "FooClass" foo

C++:

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



## Currently working on

Introduction

Requirements

Creation

Implementation

Patterns

Conclusions

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

[Split into a slide for each? Or pick a couple important ones and just do a slide for each of those? —BM]

# Language of Design

Introduction

Requirements

Creation

Implementation

Patterns

Conclusions

## Drasil project

- Generate scientific software
- Design language allows users to influence design
- GOOL is the backend

# Complete Example

Introduction

Requirements

Creation

Implementation

Patterns

Conclusions

## Projectile program

### Design 1

- Documented
- Bundled inputs

### Design 2

- Logging
- More modular

We currently use GOOL to generate some examples of scientific software (glass breakage, projectile simulation)

Together new:

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

With respect to “The Goal” — It is possible