

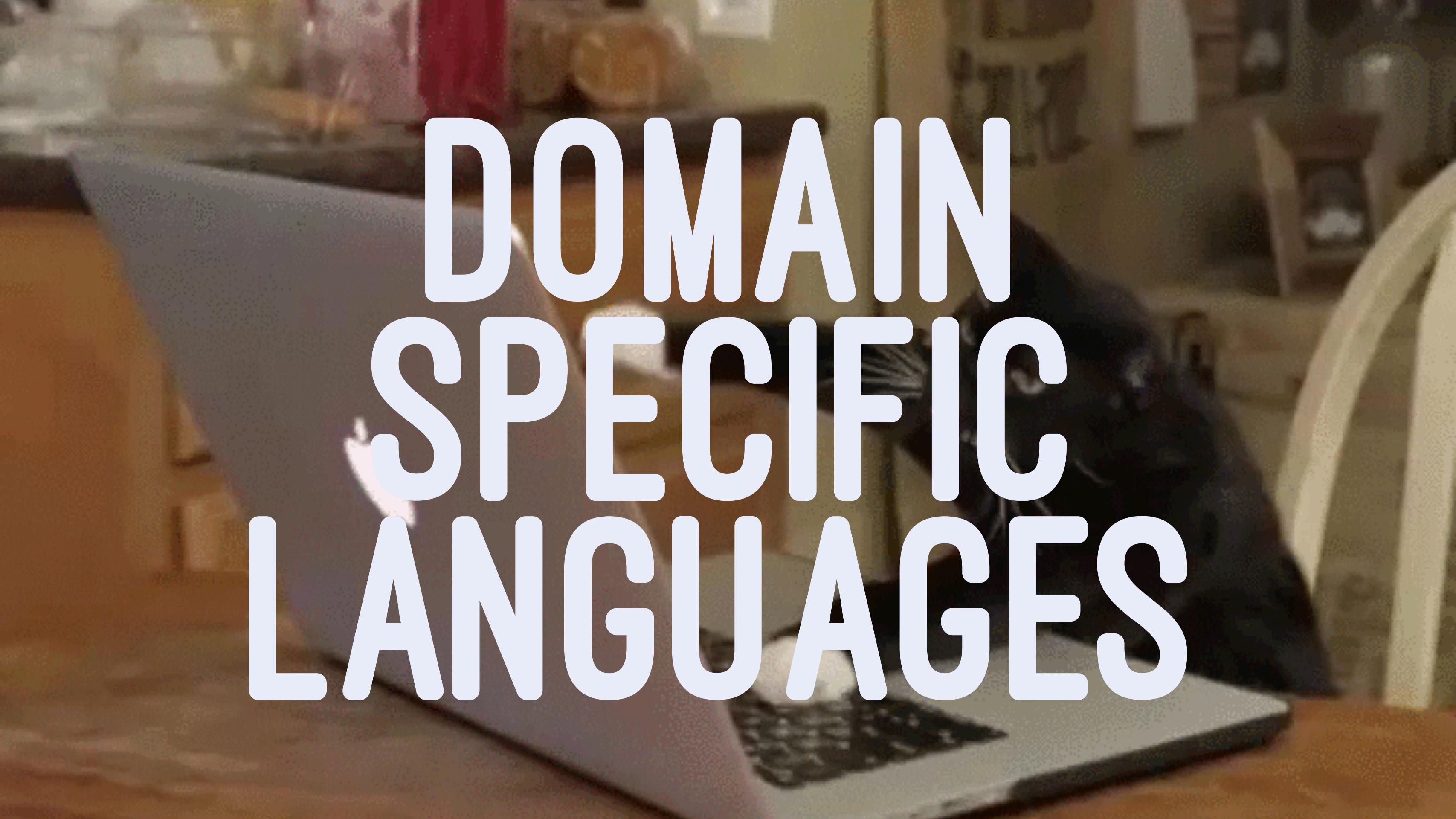


> 50 ENGINEERS

- > 50 ENGINEERS
- > 690K LOC

- 50 ENGINEERS
- 690K LOC
- COMMIT LANDS EVERY 7 MINUTES

PROBLEM
MOVE FAST WITH QUALITY

A person is seen from the side, focused on a laptop screen. The background is slightly blurred, emphasizing the person and the text.

DOMAIN
SPECIFIC
LANGUAGES

WRITING DOMAIN SPECIFIC LANGUAGES

RAHUL MALIK (@RMALIK)

MOBICONF

A COMPUTER PROGRAMMING LANGUAGE OF
LIMITED EXPRESSIVENESS
FOCUSED ON A
PARTICULAR DOMAIN
> MARTIN FOWLER

EXAMPLES

- > **MAKEFILE**
- > **PODFILE**
- > **QUICKCHECK**

**WHY SHOULD
WE WRITE
THEM?**

EASE OF USE

TYPE-SAFETY

DECLARATIVE

(INTENT) → ACTION



Typical Ambiguity

@CodaFi_

Follow



“C is a basically domain-specific language for writing unsafe code”

Yes, yes it is.

RETWEETS

15

LIKES

39



10:58 PM - 12 Apr 2017

FUNCTIONAL LANGUAGES

- > STRONG TYPE SYSTEM
- > HIGHER-ORDER FUNCTIONS / LAZY EVALUATION
 - > ELEGANT SYNTAX

RECIPE FOR CREATING A NEW DSL

- > MODEL THE PROBLEM (REIFY)
- > BUILD MODULAR / COMPOSABLE ABSTRACTIONS
 - > WORK ITERATIVELY

PINCORE

SWIFT PACKAGE INIT

FILES & FOLDERS

```
→ mkdir MyLibrary
```

```
→ swift package init
```

```
Creating library package: MyLibrary
```

```
Creating Package.swift
```

```
Creating .gitignore
```

```
Creating Sources/
```

```
Creating Sources/MyLibrary.swift
```

```
Creating Tests/
```

```
Creating Tests/LinuxMain.swift
```

```
Creating Tests/MyLibraryTests/
```

```
Creating Tests/MyLibraryTests/MyLibraryTests.swift
```

CREATING FOLDERS...

```
let frameworkName = "MyLibrary"
// make the directory structure
c.mkdir(frameworkName)
c.currentDirectory += "/\\"(frameworkName)"
let coreDirectories = ["Sources", "Tests"]
// add placeholder .gitkeep files to directories
coreDirectories.forEach(c.mkdir)
coreDirectories.forEach(c.gitkeep)
```

CREATING FOLDERS...

```
let frameworkName = "MyLibrary"
// make the directory structure
c.mkdir(frameworkName)
c.currentDirectory += "/\\"(frameworkName)"
let coreDirectories = ["Sources", "Tests"]
// add placeholder .gitkeep files to directories
coreDirectories.forEach(c.mkdir)
coreDirectories.forEach(c.gitkeep)
c.currentDirectory += "/Tests"
c.mkdir("\\"(frameworkName)Tests")
```

CREATING FILES...

```
// ctx - Context variables for all templates

// Create files from templates
evalTemplate(resource: ".gitignore", context: ctx, to: Path(""))
evalTemplate(resource: "Package.swift", context: ctx, to: Path(""))
evalTemplate(resource: "Tests.swift", context: ctx,
             to: Path("Tests/\\(frameworkName)Tests/\\(frameworkName)Tests.swift"))

// Repeat for every file ...
```



WHATS WRONG WITH THIS?

- > ERROR PRONE
- > HARD TO FOLLOW
- > IMPERATIVE

MODEL THE
PROBLEM

RECURSIVE ENUMS

```
enum Tree<A> {  
    case Empty  
    indirect case Node(left: Tree<A>, value: A, right: Tree<A>)  
}
```

ALGEBRAIC DATA TYPES (ADT)

TYPE FORMED BY COMBINING OTHER TYPES.



REIFY WITH RECURSIVE ENUMS

```
enum DirectoryItem {  
    case File(name: String, content: String)  
    indirect case Folder(name: String, contents: [DirectoryItem])  
}
```

FILES / FOLDERS WITH ADT

```
let package = DirectoryItem.Folder(  
    name: "MyLibrary",  
    contents: [  
        .File(name: ".gitignore", content: ""),  
        .Folder(  
            name: "Sources",  
            contents: [.File(name: "MyLibrary.swift", content: "...")]),  
        .Folder(  
            name: "Tests",  
            contents: [  
                .File(name: "LinuxMain.swift", content: "..."),  
                .Folder(name: "MyLibraryTests",  
                    contents: [.File(name: "MyLibraryTests.swift", content: "...")])  
            ]  
        )  
    ]  
)
```

```
func createFiles(atPath path: String, rootDirectory item: DirectoryItem) {  
    switch item {  
        case .File(name: let name, contents: let contents):  
            writeFile(contents: contents, atPath: Path(name))  
    }  
}
```

```
func createFiles(atPath path: String, rootDirectory item: DirectoryItem) {  
    switch item {  
        case .File(name: let name, contents: let contents):  
            writeFile(contents: contents, atPath: Path(path + name))  
        case .Folder(name: let name, items: let subdirs):  
            c.mkdir(name)  
            let newPath = path + "/\u{202A}(name)"  
            c.currentDirectory = newPath  
            _ = subdirs.map { createFiles(atPath: newPath, rootDirectory: d) }  
            c.currentDirectory = path  
    }  
}
```

LS

```
func ls(_ item: DirectoryItem, _ path: String) {  
    switch item {  
        case .File(name: let name, content:_):  
            print([path, name].joined(separator: "/"))  
        case .Folder(name: let name, contents: let contents):  
            contents.map { ls($0, [path, name].joined(separator: "/"))}  
    }  
}
```

DEEP DSL

1 + 2 = Expr(+ , 1 , 2)

PLANK



plank

CODE GENERATION

```
typealias CodeGenerator = () -> [String]
```

FIBONACCI

```
func fibonacci(_ i: Int) -> Int {  
    if i <= 2 {  
        return 1  
    } else {  
        return fibonacci(i - 1) + fibonacci(i - 2)  
    }  
}
```

FIRST ATTEMPT

```
func generatefib () {  
    return [  
        "if i <= 2 {",  
        "    return 1",  
        "} else {",  
        "    return fibonacci(i - 1) + fibonacci(i - 2)",  
        "}",  
    ]  
}
```

IF / ELSE STRUCTURE

```
if /* some condition */ {  
    /* line 1 */  
    /* line 2 */  
    /* ... */  
} else {  
    /* line 1 */  
    /* line 2 */  
    /* ... */  
}
```

IF

```
func ifStmt(_ condition: String,  
           _ body: CodeGenerator) -> [String] {  
    return [  
        "if \(condition) {",  
        -->body,  
        "}."  
    ]  
}
```

ELSE

```
func elseStmt(_ body: CodeGenerator) -> [String] {  
    return [  
        " else {",  
        -->body,  
        "} "  
    ]  
}
```

IF + ELSE

```
func ifElseStmt(_ condition: String,  
                body: @escaping CodeGenerator) -> (CodeGenerator) -> [String] {  
    return { elseBody in [  
        ifStmt(condition, body),  
        elseStmt(elseBody)  
    ].flatMap { $0 }  
}  
}
```

FIBONACCI WITH DSL

```
ifElseStmt("i <= 2") {  
    // If  
    "return 1"  
} ({  
    // Else  
    "return fibonacci(i - 1) + fibonacci(i - 2)"  
})
```

HIGHER ORDER FUNCTIONS

SHALLOW DSL

$$1 + 2 = 3$$

SHALLOW

$$1 + 2 = 3$$

DEEP

$$1 + 2 = \text{Expr}(+, 1, 2)$$

**LET'S BUILD
SOMETHING
NEW!**

GRAPHQL

WHAT IS GRAPHQL?

GRAPHQL IS A QUERY LANGUAGE FOR APIs.
IT'S AN ALTERNATIVE TO THE TRADITIONAL RESTFUL ENDPOINTS.

GITHUB



ISSUES IN REPOSITORY

```
{  
  repository(owner:"pinterest", name:"plank") { // Repo  
    name,  
    description,  
    homepageUrl  
  }  
}
```

ISSUES IN REPOSITORY

```
{  
  repository(owner:"pinterest", name:"plank") { // Repo  
    name,  
    description,  
    homepageUrl,  
    owner { // User  
      avatarUrl  
    }  
  }  
}
```

ISSUES IN REPOSITORY

```
{  
  repository(owner:"pinterest", name:"plank") { // Repo  
    name,  
    description,  
    homepageUrl,  
    owner { // User  
      avatarUrl  
    },  
    issues(first:10) { // Issues  
      nodes {  
        title,  
        state  
      }  
    }  
  }  
}
```

JSON RESPONSE

```
{  
  "data": {  
    "repository": {  
      "name": "plank",  
      "description": "A tool for generating immutable model objects",  
      "homepageUrl": "https://pinterest.github.io/plank/",  
      "owner": {  
        "avatarUrl": "https://avatars1.githubusercontent.com/u/541152?v=3"  
      },  
      "issues": {  
        "nodes": [  
          {  
            "title": "Model value type in Schema.Map / Schema.Array as a separate enum",  
            "state": "OPEN"  
          },  
          ]  
        }  
      }  
    }  
}
```

TYPES OF DSLS

- EXTERNAL
- INTERNAL / EMBEDDED

PROBLEM:
HOW CAN WE WRITE CONCISE
TYPE-SAFE QUERIES?

```
let query = [
  "{",
  " repository(owner:\"pinterest\", name:\"plank\") {",
  "   name,
  "   description,
  "   homepageUrl,
  "   owner {",
  "     avatarUrl",
  "   }",
  "   issues(first:10) {",
  "     nodes {",
  "       title,
  "       state",
  "     }",
  "   }",
  " }",
"}"].joined(separator: " ")
```



QUERY

STRUCTURE

```
repository(owner: "pinterest", name: "plank") {  
  name,  
  description,  
  owner {  
    avatarUrl  
  }  
}
```

STRUCTURE

```
object(arguments) {  
    field1,  
    field2,  
    object {  
        field3  
    }  
}
```



```
enum Node {  
    case Field(label: String)  
    indirect case Object(label: String, arguments: [InputArgument]?, fields:[Node])  
}  
  
enum InputType {  
    case string(String)  
    case integer(Int)  
}  
  
typealias InputArgument = (String, InputType)
```

QUERY WITH NODES : REPOSITORY

```
.Object(label: "repository",
    arguments: [("owner", .string("pinterest")), ("name", .string("plank"))],
    fields: [.Field(label: "name"),
              .Field(label: "description"),
              .Field(label: "homepageUrl"),
    ]
)
```

QUERY WITH NODES : REPOSITORY OWNER

```
.Object(label: "repository",
    arguments: [("owner", .string("pinterest")), ("name", .string("plank"))],
    fields: [.Field(label: "name"),
              .Field(label: "description"),
              .Field(label: "homepageUrl"),
              .Object(label: "owner",
                      arguments: nil,
                      fields: [.Field(label: "avatarUrl")])],
    )
)
```

QUERY WITH NODES : REPOSITORY ISSUES

```
.Object(label: "repository",
    arguments: [("owner", .string("pinterest")), ("name", .string("plank"))],
    fields: [.Field(label: "name"),
              .Field(label: "description"),
              .Field(label: "homepageUrl"),
              .Object(label: "owner",
                      arguments: nil,
                      fields: [.Field(label: "avatarUrl")]),
              .Object(label: "issues",
                      arguments: [("first", .integer(10))],
                      fields: [
                          .Object(label: "nodes",
                                  arguments: nil,
                                  fields: [.Field(label: "title"),
                                            .Field(label: "state")])
                      ])
    ])
```

NODE → QUERY STRING : FIELD

```
func renderQueryString(node: Node) -> String {  
    switch node {  
        case .Field(label: let label):  
            return label  
    }  
}
```

NODE → QUERY STRING : OBJECT

```
func renderQueryString(node: Node) -> String {  
    switch node {  
        case .Field(label: let label):  
            return label  
        case .Object(label: let label, arguments: .none, fields: let fields):  
            // Object without arguments  
            let fieldString = fields.map(renderQueryString).joined(separator: " ")  
            return "\((label)) { \((fieldString)) }"  
    }  
}
```

NODE → QUERY STRING : OBJECT WITH ARGS

```
func renderQueryString(node: Node) -> String {  
    switch node {  
        case .Field(label: let label):  
            return label  
        case .Object(label: let label, arguments: .none, fields: let fields):  
            // Object without arguments  
            let fieldString = fields.map(renderQueryString).joined(separator: " ")  
            return "\(label) { \(fieldString) }"  
        case .Object(label: let label, arguments: .some(let args), fields: let fields):  
            // Object with arguments  
            let fieldString = fields.map(renderQueryString).joined(separator: " ")  
            let argString = args.map(renderArgument).joined(separator: ", ")  
            return "\(label) (\(argString)) { \(fieldString) }"  
    }  
}
```

FIELD ARGUMENTS

```
func renderArgument(arg: InputArgument) -> String {  
    switch arg.1 {  
        case .string(let argVal):  
            return "\u{arg.0}:\\""\u{argVal}\\""  
        case .integer(let argVal):  
            return "\u{arg.0}:\u{argVal}"  
    }  
}
```

TYPES

FIELDS

```
typealias FieldSelection<T> = () -> [T]

enum QueryRoot {
    case repository(owner: String,
                    name: String,
                    FieldSelection<RepositoryField>)
    // Other root cases...
}
```

```
enum RepositoryField {  
    case name  
    case description  
    case homepageUrl  
    case owner(FieldSelection<UserField>)  
    case issues(first: Int, FieldSelection<IssueField>)  
}
```

```
enum UserField {  
    case login  
    case avatarUrl  
}
```

```
enum IssueField {  
    case title  
    case state  
}
```

FIELD SELECTION DSL

```
.repository(owner: "pinterest", name: "plank") {  
  .name,  
  .description,  
  .owner {[  
    .login,  
    .avatarUrl  
  ]},  
  .issues(first: 10) {[  
    .nodes {[  
      .title,  
      .state  
    ]}  
  ]}  
}  
}
```

(FIELDS) → NODES ?

(FIELDS) → NODES

```
protocol NodeRenderer {  
    func renderNode() -> Node  
}
```

QUERYROOT

```
enum QueryRoot: NodeRenderer {
    func renderNode() -> Node {
        switch self {
        case .repository(owner: let owner, name: let name, let fieldsFn):
            return Node.Object(label: "repository",
                arguments: [("owner", .string(owner)),
                            ("name", .string(name))],
                fields: fieldsFn().map { $0.renderNode() })
        }
    }
}
```

USER, REPOSITORY, ISSUE...

```
enum UserField: NodeRenderer { /*...*/ }
```

```
enum RepositoryField: NodeRenderer { /*...*/ }
```

```
enum IssueField: NodeRenderer { /*...*/ }
```

(FIELDS) → (NODES) → QUERY

```
func query(_ fields: FieldSelection<QueryRoot>) -> String {  
    return "{" +  
        fields().map { $0.renderNode() }  
            .map { $0.renderQueryString() }  
            .joined(separator: " ") +  
        "}"  
}
```

DEMO





A photograph of a person from the waist up, wearing a black VR headset and holding a white game controller. They are standing in a room with a blue chair visible in the background. The image is slightly blurred.

THANK YOU!

@RMALIK

GITHUB.COM/RAHUL-MALIK

APPENDIX

SWITCH STATEMENT DSL

SWITCH STRUCTURE

```
switch /* variable name */ {  
    case1:  
        /* case 1 logic */  
    case2:  
        /* case 2 logic */  
    caseN:  
        /* case n logic */  
    default:  
        /* default logic */  
}
```

SWITCH / SWITCHCASE

```
enum SwitchCase {
    case Case(condition: String, body: CodeGenerator)
    case Default(body: CodeGenerator)
}

func switchStmt(_ switchVariable: String, body: () -> [SwitchCase]) -> [String] {
    return [
        "switch (\(switchVariable)) {",
        body().map { $0.render() }
            .flatMap { $0 }
            .joined(separator: "\n"),
        "}"
    ]
}
```

RENDERING CASE STATEMENTS

```
enum SwitchCase {  
    func render() -> [String] {  
        switch self {  
            case .Case(let condition, let body):  
                return [  
                    "case \(condition):",  
                    body  
                ]  
            case .Default(let body):  
                return [  
                    "default:",  
                    body  
                ]  
        }  
    }  
}
```

FIBONACCI WITH SWITCH DSL

```
switchStmt("i") {  
    .Case("0") {[  
        "return 1"  
    ]},  
    .Case("1") {[  
        "return 1"  
    ]},  
    .Default {[  
        "return fibonacci(i - 1) + fibonacci(i - 2)"  
    ]}  
}
```

CUSTOM OPERATORS

```
prefix operator -->
```

```
prefix func --> (body: CodeGenerator) -> String {  
    return body().flatMap {  
        $0.components(separatedBy: "\n").map { " " + $0 }  
    }.joined(separator: "\n")  
}
```