NPidE

null pointer IDE

Roma Brek	19213
Pavel Vasilev	19213
Boris Patrushev	19213
Artём Tarasov	19214

What is the main *IDEA* of our *IDE?*

- The main feature of our IDE is its customizability. You can add an unlimited number of new languages (using what we call “***language distributions***”) so that our IDE can work with any of them
- The first language for which we created a “distribution” is **CdM-8** assembly language
- The second one is **Clojure** programming language

The second semester task

- The second semester task was to add **Clojure** programming language support to our IDE

What we use?

- Kotlin
- ANTLR4
- Compose
- RSyntaxTextArea

How is our IDE configured?

A *project* in NPidE have two configs:

- **project config** – “*config.yaml*” in project root
- **language distribution config** – “LanguageDistributionInfo.yaml”

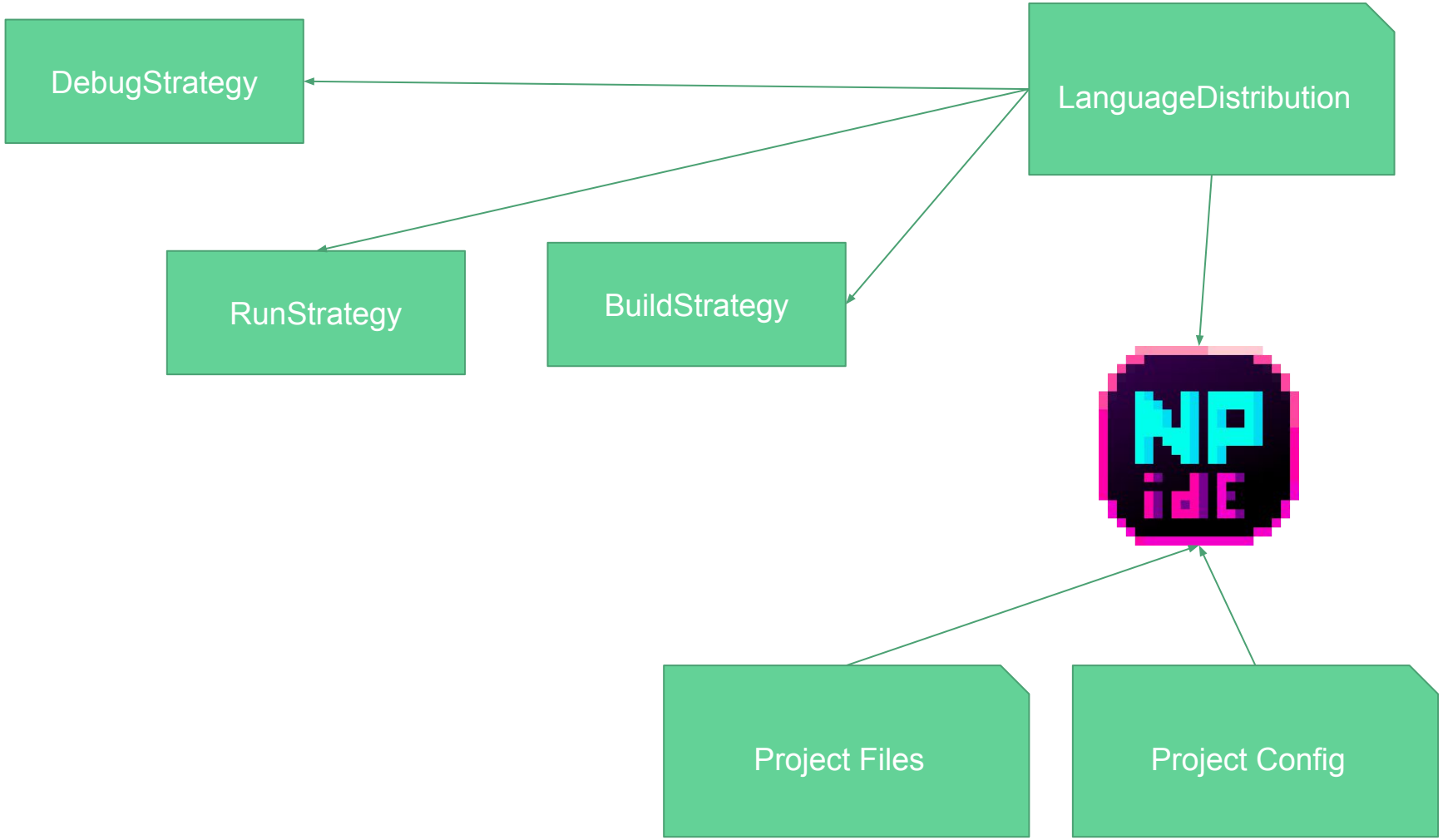
project config is used to configure the project and reference **language distribution config**

```yaml
projectName: "test-project"
entryPoint: "ru.nsu.fit.core"
languageDistribution: "clojure/LanguageDistributionInfo.yaml"
projectFiles:
- "/home/sescer/github/NPidE/examples/test-clojure/src/ru/nsu/fit/core.clj"
```

What is a language distribution file?

- YAML-file, which describes build/run/debug execution of project files
- Created by developer of programming language
- Structure of language distribution config file for clojure:

```yaml
buildStrategy:
    strategyClass: "ru.nsu_null.npide.ide.projectstrategies.defaults.delegators.BuilderDelegatorStrategy"
    extraParameters: {executable: "python", script: "clojure_script.py"}
runStrategy:
    strategyClass: "ru.nsu_null.npide.ide.projectstrategies.defaults.delegators.RunnerDelegatorStrategy"
    extraParameters: {executable: "python", script: "clojure_script.py"}
debugStrategy:
    strategyClass: "ru.nsu_null.npide.ide.projectstrategies.defaults.delegators.DebuggerDelegatorStrategy"
    extraParameters: {executable: "python", script: "clojure_script.py", "continue":"(c)"}
grammarConfigs:
    - sourceFileExtension: "clj"
      grammar: "grammar/clj.g4"
      syntaxHighlighter: "syntaxHighlighter/colors.json"
```


DebugStrategy
LanguageDistribution
RunStrategy
BuildStrategy
NP
ide
Project Files
Project Config

How to describe highlighting/goto rules?

JSON that describes how to color lexer rules

```
"color": "#85C1E9",
"instructions": [
  "RR_INSTR",
  "R_INSTR",
  "R_MACRO_INST",
  "RC_INSTR"
]
```

```
#global_def
#def
#usage
#scope
```

ANTLR lexer rules for CDM-8 asm

```
/* Instruction that have two registers as target*/
RR_INSTR:
    'ld' | 'st' | 'move' | 'add' |'addc' |' sub'| 'cmp'| 'and' |'or' | 'xor'
;

/* Instruction that take one register parametr */
R_INSTR:
    'neg'| 'dec'| 'inc'| 'shr'| 'shra'| 'shla'| 'rol' |'push'| 'pop'|
    'stsp'| 'ldsp'| 'tst'| 'clr'
```

ARCHITECTURE

Modules

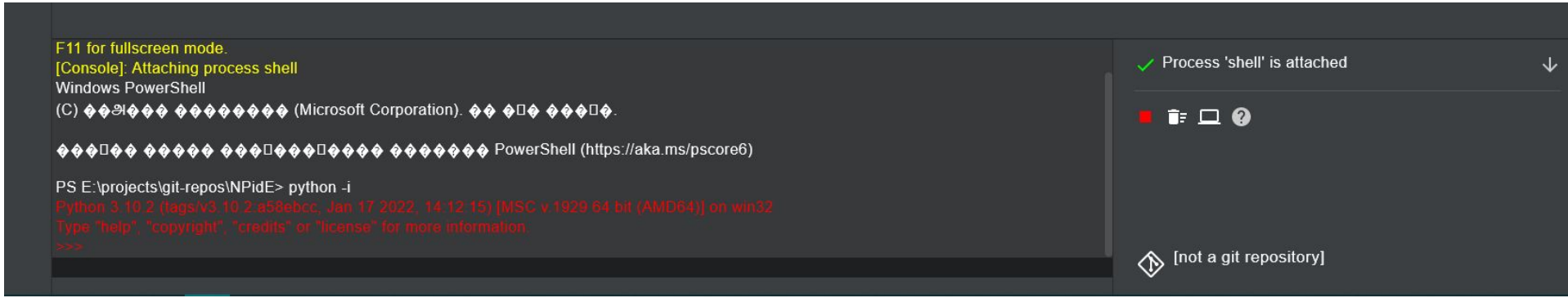
Parser

This module is responsible for analyzing the files being edited and create internal structure for describing this ones

- translation - creates symbol tables and so on
- generator - generates parser and lexer files based on provided grammar
- compose_support - allows to connect highlighting to our editing text area

Console

- Responsible for getting output from build/run/debug
- Is used to log errors or any messages
- Is used by user using ConsoleView to interact with the attached process



StatusBar

- ButtonsBar - responsible for drawing bar for buttons
- ButtonUsage - responsible for handling button clicks
- Calls NPIDE’s methods to attach strategies (build/run/debug) to console

Config

responsible for storing and reading configuration from YAML-file

- Config Manager - manage project configuration file
  - ProjectConfig
  - LanguageDistributionInfo
- Config Dialog - responsible for drawing the output

```
Roman
@Serializable
open class ProjectConfig(
    open var projectName: String,
    open var entryPoint: String,
    open var languageDistribution: String,
    open var projectFiles: List<String>,
)
```

```
Roman
@Serializable
data class LanguageDistributionInfo(
    val buildStrategy: StrategyInfo,
    val runStrategy: StrategyInfo,
    val debugStrategy: StrategyInfo,
    var grammarConfigs: List<GrammarConfig>
)
```

Configuration of project

Project name: asdf

Project entry point / main class: 42

Configuration of Language distribution file:

Language distribution file path

·was21/PycharmProjects/npide-langs/cocas/distr.yml

...

Configuration of Source file:

Source file path

...

Add project file source path

/home/gerwas21/NPidEprojects/test2/gamer.asm

Delete

Grammar configs loaded from language distribution:

asm

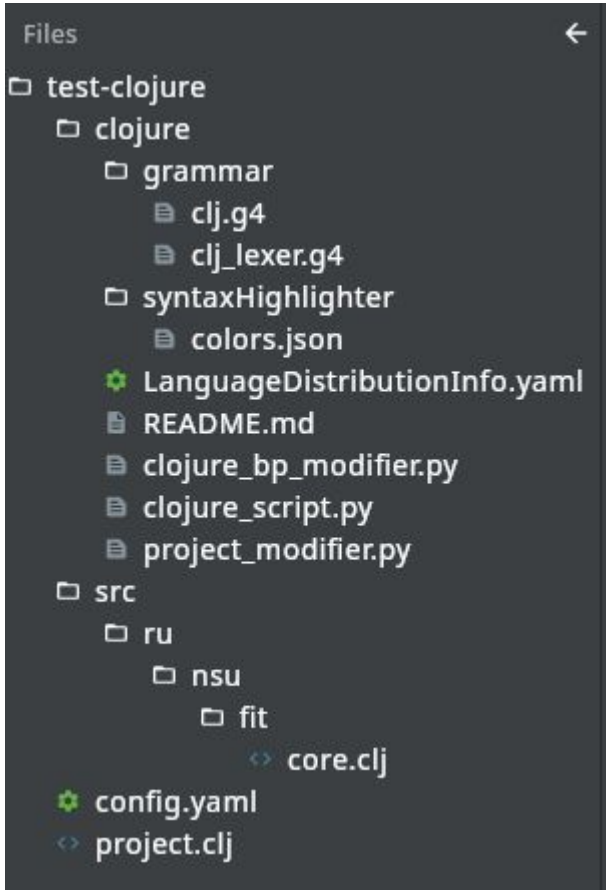
asm.g4

colors.json

Apply config

FileTree

- Lets user interact with file system
  - Move files
  - Display files
  - Choose files



Breakpoints

BreakpointStorage - project-local object responsible for adding/removing breakpoints in source files

```
6  #jsr clean
7  halt
8  loop:
9  clr r3
10 add r2, r3
11 add r1, r3
12 st r3, r2
13 inc r2
14 cmp r2, r0
15 bne loop
16
17 ldi r0, 0x00a
18 clr r3
19 add r0, r3
20 add r1, r3
21 ld r3, r2
22 cmp r0, r2
23
24 inc r0
25 clr r3
26 add r0, r3
27 add r1, r3
28 ld r3, r2
29 cmp r0, r2
30
```

Watches

Watches - if debugger implementation supports watches //TODO

```
o
ru.nsu.fit.core=> (factorial 3)[D[12D[13C

Break from: /home/sescer/github/NPidE/examples/test-clojure/.build1/src/ru/nsu/fit/core.clj:22 (type "(help)" for help)

  22: (defn factorial [n]
=> 23:      (if (= n 0) 1 (* n (factorial (break (- n 1))))))

ru.nsu.fit.core/factorial:23=> {n 3}
nil
ru.nsu.fit.core/factorial:23=>
```

Process 'debug' is attached

```
ru.nsu.fit.core/factorial:23=> = {n 3}
```

[not a git repository]

DEMO

Menu

NPIDE

Add new project

/home/gerwas21/NPidEprojects/MyProj

/home/gerwas21/NPidEprojects/test

/home/gerwas21/NPidEprojects/test2


Menu
NPIDE
Add new project
C:\Users\sonechka\Documents\Education\NPIdE\examples\test-clojure

Design patterns

Singleton Pattern

- NPIDE
- Fonts
- AppTheme

```
Roman +1
19  object NPIDE {
20      var state: State by mutableStateOf(CHOOSING_PROJECT)
21          private set
22
        Roman
23      enum class State {
24          CHOOSING_PROJECT,
25          IN_PROJECT
26      }
27
28      var currentProject: Project? = null
29      lateinit var configManager: ConfigManager
30      lateinit var projectStorage: ProjectStorage
31
32      @Suppress( ...names: "MemberVisibilityCanBePrivate")
33      lateinit var builder: BuilderStrategy
34      @Suppress( ...names: "MemberVisibilityCanBePrivate")
35      lateinit var runner: RunnerStrategy
36      lateinit var debugger: DebuggerStrategy
37
38      lateinit var console: Console
39
        Roman +1
40      fun openProject(project: Project) {
41          currentProject = project
42          console = Console()
43          configManager = ConfigManager(project)
44          projectStorage = ProjectStorage(project)
45          loadProjectWorkers()
46          state = IN_PROJECT
47      }
48
```

Proxy Pattern

- Used for adding “save on edit” functionality to the config class.

```
class AutoUpdatedProjectConfig(projectConfig: ProjectConfig) : ProjectConfig(
    projectConfig.build,
    projectConfig.run,
    projectConfig.debug,
    projectConfig.filePathToDirtyFlag,
    projectConfig.projectFilePaths,
    projectConfig.grammarConfigs
) {

    override var build: String = super.build
        set(value) {
            field = value
            sync()
        }
    override var run: String = super.run
        set(value) {
            field = value
            sync()
        }
    override var debug: String = super.debug
```

Observer pattern

All of our project

```
class Console {
    var content: MutableState<String> = mutableStateOf( value: "")
        private set

    fun add(newContent: String) {
        content += newContent
```

Delegation pattern

```
7   /**
8    * Represents a real process (as in a process in OS terms)
9    * as a [ConsoleProcess]
10   */
     p.vasilev +1
11  class RealConsoleProcess (
12      private val process: Process
13  ): ConsoleProcess {
14
15      override val outputStream: OutputStream by process::outputStream
16
17      override val inputStream: InputStream by process::inputStream
18
19      override val errorStream: InputStream by process::errorStream
20
21      override val isAlive: Boolean by MethodDelegator(process::isAlive)
22
        Roman
23      override fun destroy() = process.destroy()
24
25      override val exitValue: Int by MethodDelegator(process::exitValue)
26  }
27
```

Object pool pattern

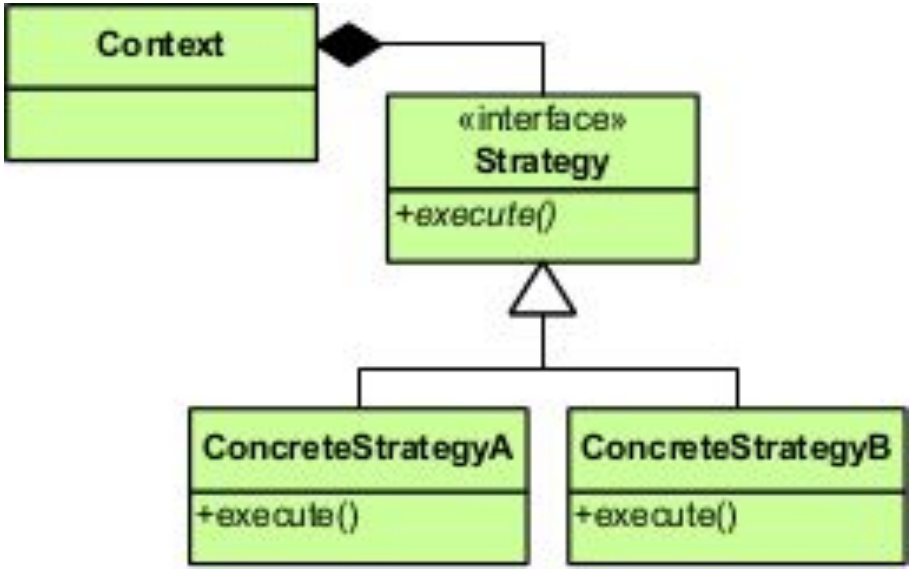
```
object LanguageManagerProvider {
    private val extensionToLanguageManager = HashMap<String, G4LanguageManager>()
    fun getLanguageManager(extension: String): G4LanguageManager {
        synchronized( lock: this) {
            return extensionToLanguageManager.getOrPut(extension) {
                ConfigManager.findGrammarConfigByExtension(extension)
                G4LanguageManager(extension)  ^getOrPut
            }
        }
    }
}
```

Factory

- We have “lexer creator” that creates lexer subclasses based on their names and then use common interface

Strategy

We load lexer and parser classes depends on extension name and traversing the parsing tree based on the algorithms of these classes



Project Strategies: Builder, Runner, Debugger

```
Roman
interface BuilderStrategy: ConsoleProcess {
    /**
     * Builds current project
     * @param enableDebugInfo true iff should build with debugging info
     * @param strategyContext contains information about the project,
     * @param extraParameters key-value configuration provided in the language distribution
     * @param breakPoints which are used to generate debug info
     * @param dirtyFlags maps file name to its 'dirtiness'
     * @param logger a [Logger] which can be used to log events or report errors
     */
    Roman
    fun build(
        enableDebugInfo: Boolean,
        strategyContext: ProjectStrategyContext,
        extraParameters: ExtraParameters,
        breakPoints: BreakPoints,
        dirtyFlags: DirtyFlags,
        logger: Logger
    )
}
```

Thank you for your attention!