NPidE

null pointer IDE

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

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

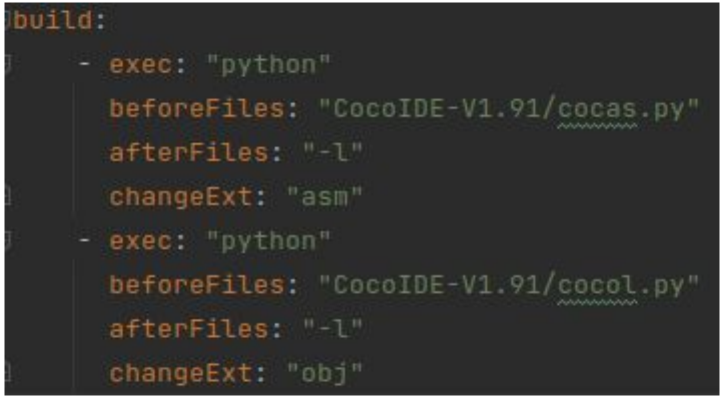
- The main feature of this IDE is its customizability. You can add supported languages, project types and so on in a declarative style using config files
- The first language for which we created config files is CdM-8 cocas assembly language

What we use?

- Kotlin
- ANTLR4
- Compose
- RSyntaxTextArea

What is a config file?

- YAML-file, which describes build/run/debug execution of project files
- is filled by developer of programming language
- Structure of build configuration for Cdm-8 cocas assembly language:

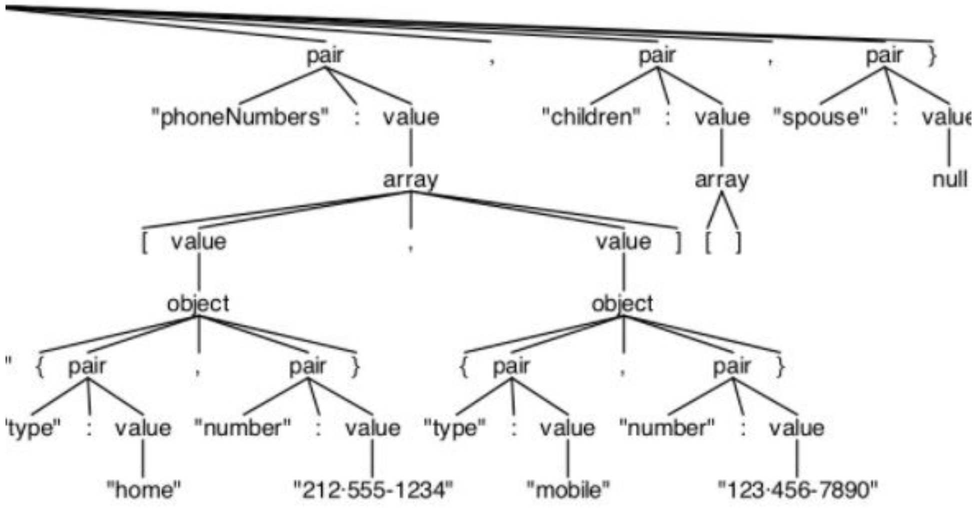


```yaml
build:
    - exec: "python"
      beforeFiles: "CocoIDE-V1.91/cocas.py"
      afterFiles: "-l"
      changeExt: "asm"
    - exec: "python"
      beforeFiles: "CocoIDE-V1.91/cocol.py"
      afterFiles: "-l"
      changeExt: "obj"
```

What is ANTLR4?

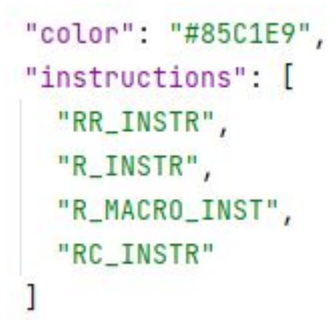
Tool that helps to create and traverse syntax trees

```
grammar JSON;

json
    : value
    ;

obj
    : '{' pair (',' pair)* '}'
    | '{' '}'
    ;

pair
    : STRING ':' value
    ;
```



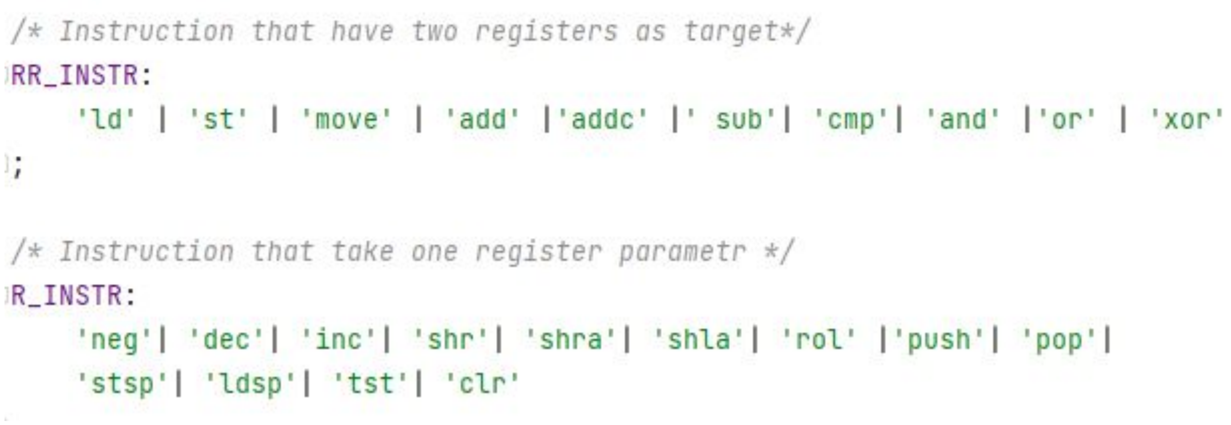
How to describe highlighting rules?

JSON that describes how to color lexer rules



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

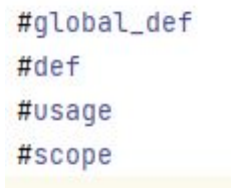
ANTLR lexer rules



```
/* 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'
```

How to describe symbol table?

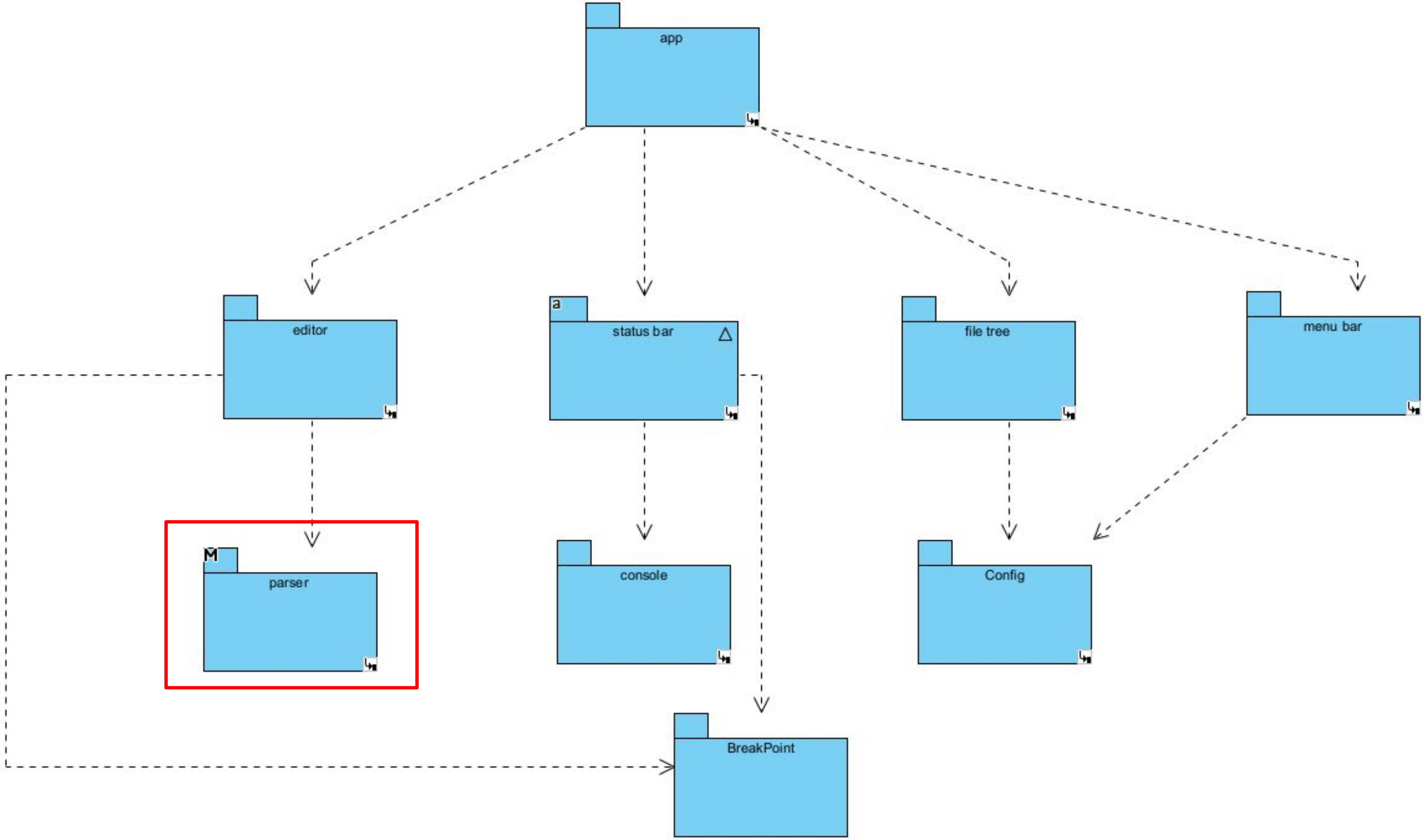
In ANTLR4 you have an opportunity to name every rule, so if you want to make symbol table you need to implement the following rules:



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

ARCHITECTURE

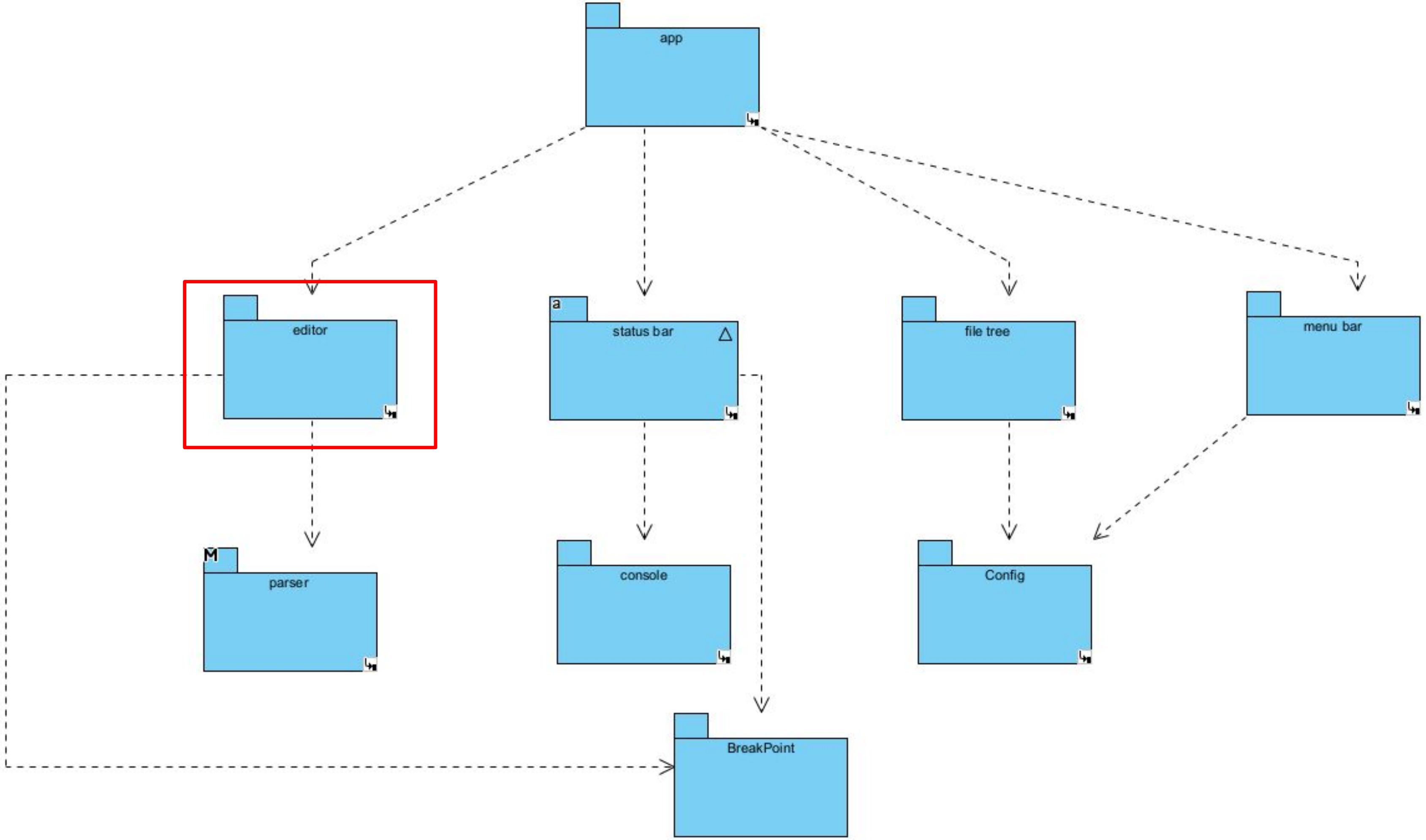
Modules


app
editor
status bar
file tree
menu bar
parser
console
Config
BreakPoint

Parser

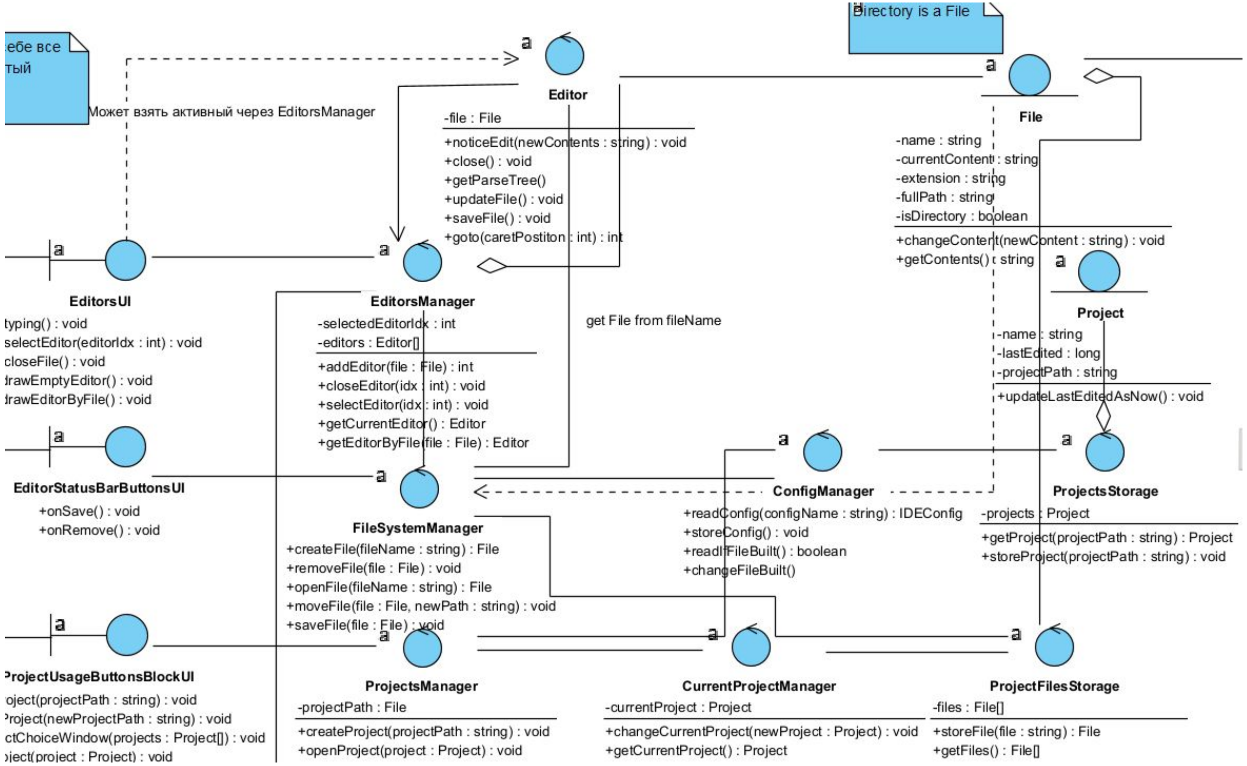
This module is responsible for analyzing the files being edited and create internal structure for describing this ones

- 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

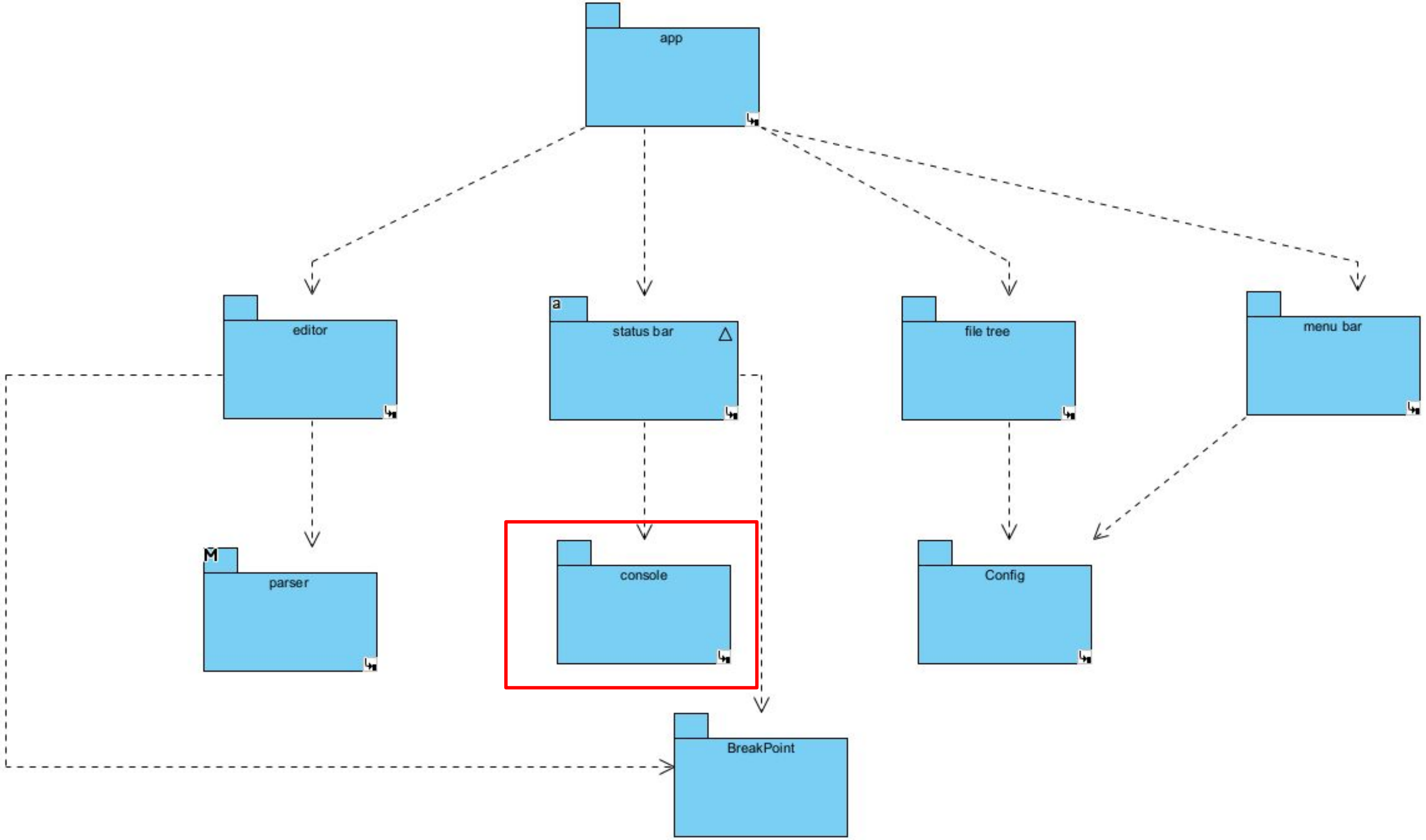

app
editor
status bar
file tree
menu bar
parser
console
Config
BreakPoint

Editor

- Editor - represents a state of a file editor
- Editors - controls currently open Editors

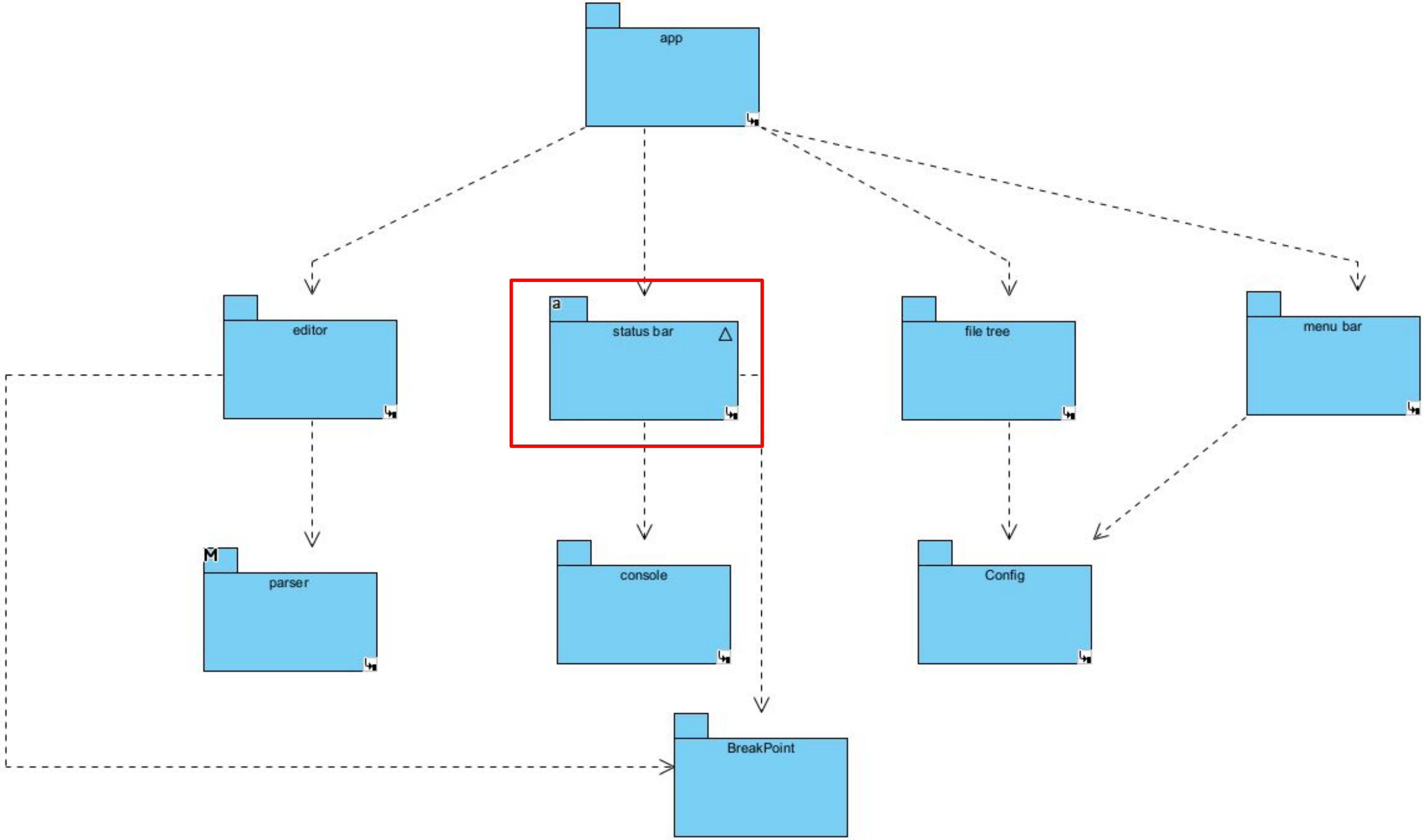


ебе все
тый
Может взять активный через EditorsManager
Directory is a File
Editor
-file : File
+noticeEdit(newContents : string) : void
+close() : void
+getParseTree()
+updateFile() : void
+saveFile() : void
+goto(caretPostiton : int) : int
File
-name : string
-currentContent : string
-extension : string
-fullPath : string
-isDirectory : boolean
+changeContent(newContent : string) : void
+getContents() : string
EditorsUI
typing() : void
selectEditor(editorIdx : int) : void
closeFile() : void
drawEmptyEditor() : void
drawEditorByFile() : void
EditorsManager
-selectedEditorIdx : int
-editors : Editor[]
+addEditor(file : File) : int
+closeEditor(idx : int) : void
+selectEditor(idx : int) : void
+getCurrentEditor() : Editor
+getEditorByFile(file : File) : Editor
get File from fileName
Project
-name : string
-lastEdited : long
-projectPath : string
+updateLastEditedAsNow() : void
EditorStatusBarButtonsUI
+onSave() : void
+onRemove() : void
FileSystemManager
+createFile(fileName : string) : File
+removeFile(file : File) : void
+openFile(fileName : string) : File
+moveFile(file : File, newPath : string) : void
+saveFile(file : File) : void
ConfigManager
+readConfig(configName : string) : IDEConfig
+storeConfig() : void
+readIfFileBuilt() : boolean
+changeFileBuilt()
ProjectsStorage
-projects : Project
+getProject(projectPath : string) : Project
+storeProject(projectPath : string) : void
ProjectUsageButtonsBlockUI
oject(projectPath : string) : void
Project(newProjectPath : string) : void
ctChoiceWindow(projects : Project[]) : void
oject(project : Project) : void
ProjectsManager
-projectPath : File
+createProject(projectPath : string) : void
+openProject(project : Project) : void
CurrentProjectManager
-currentProject : Project
+changeCurrentProject(newProject : Project) : void
+getCurrentProject() : Project
ProjectFilesStorage
-files : File[]
+storeFile(file : string) : File
+getFiles() : File[]


app
editor
status bar
file tree
menu bar
parser
console
Config
BreakPoint

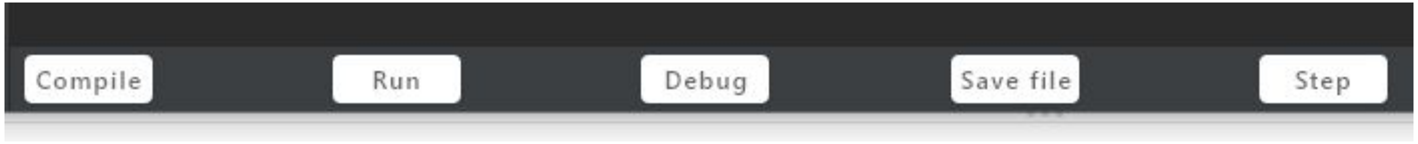
Console

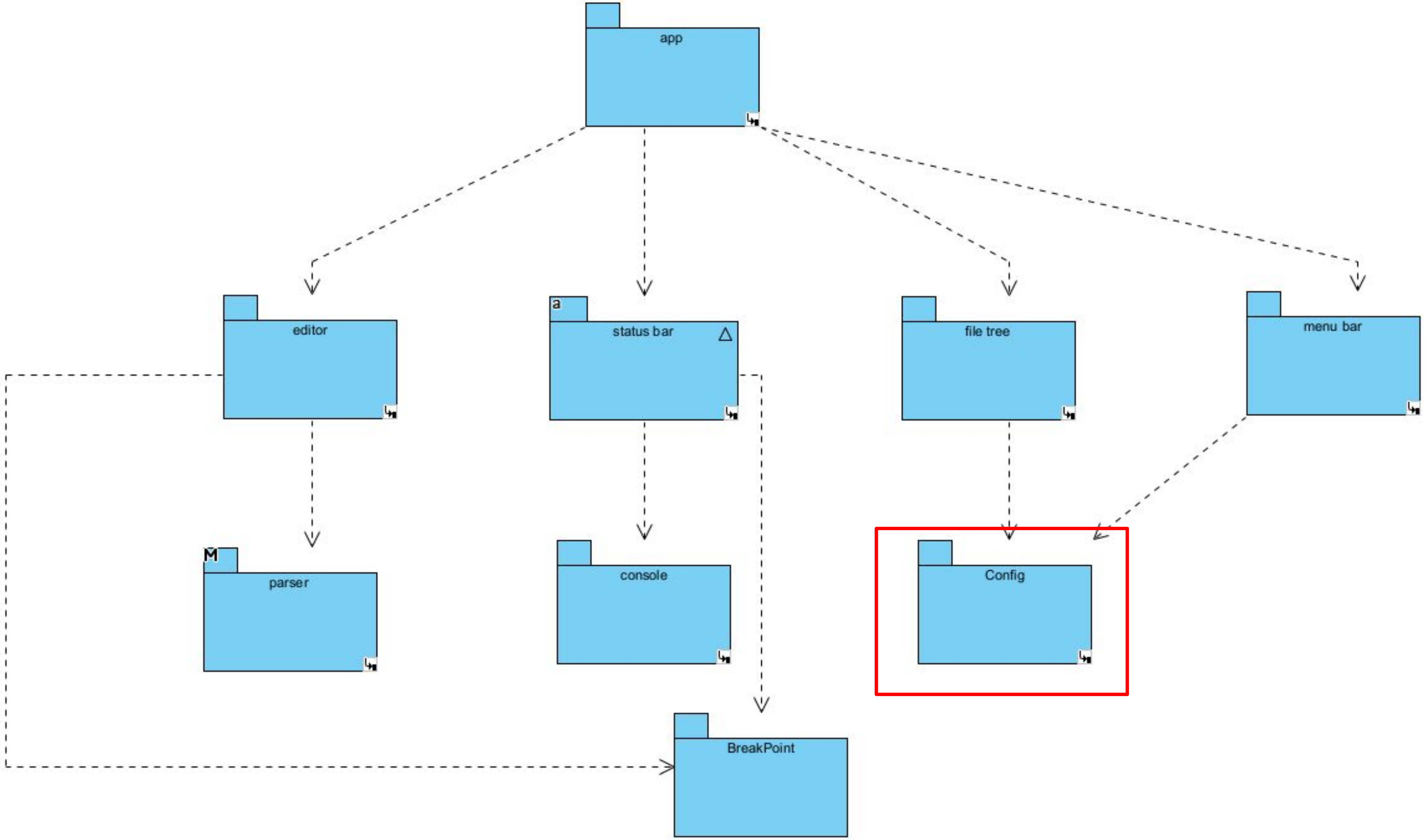
- Console - responsible for getting output from build/run/debug
- ConsoleView - responsible for drawing the aforementioned output


app
editor
a
status bar
file tree
menu bar
M
parser
console
Config
BreakPoint

StatusBar

- ButtonsBar - responsible for drawing bar for buttons
- ButtonUsage - responsible for handling button clicks

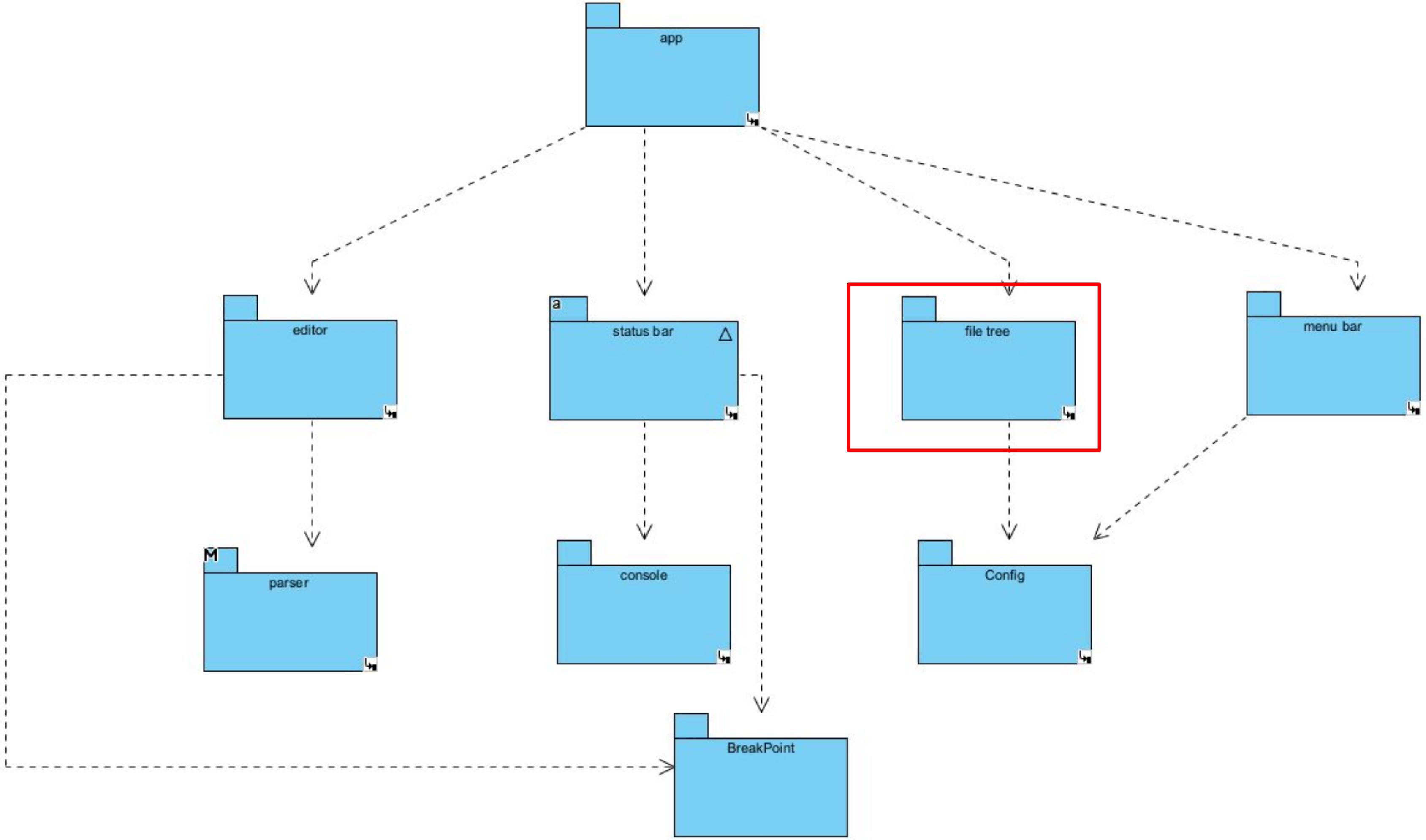



app
editor
a
status bar
file tree
menu bar
M
parser
console
Config
BreakPoint

Config

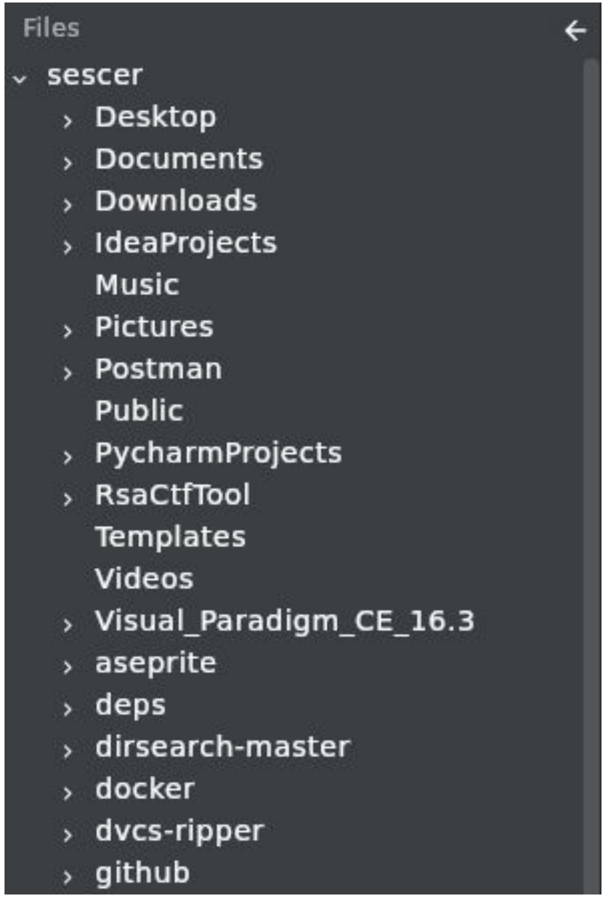
responsible for storing and reading configuration from YAML-file

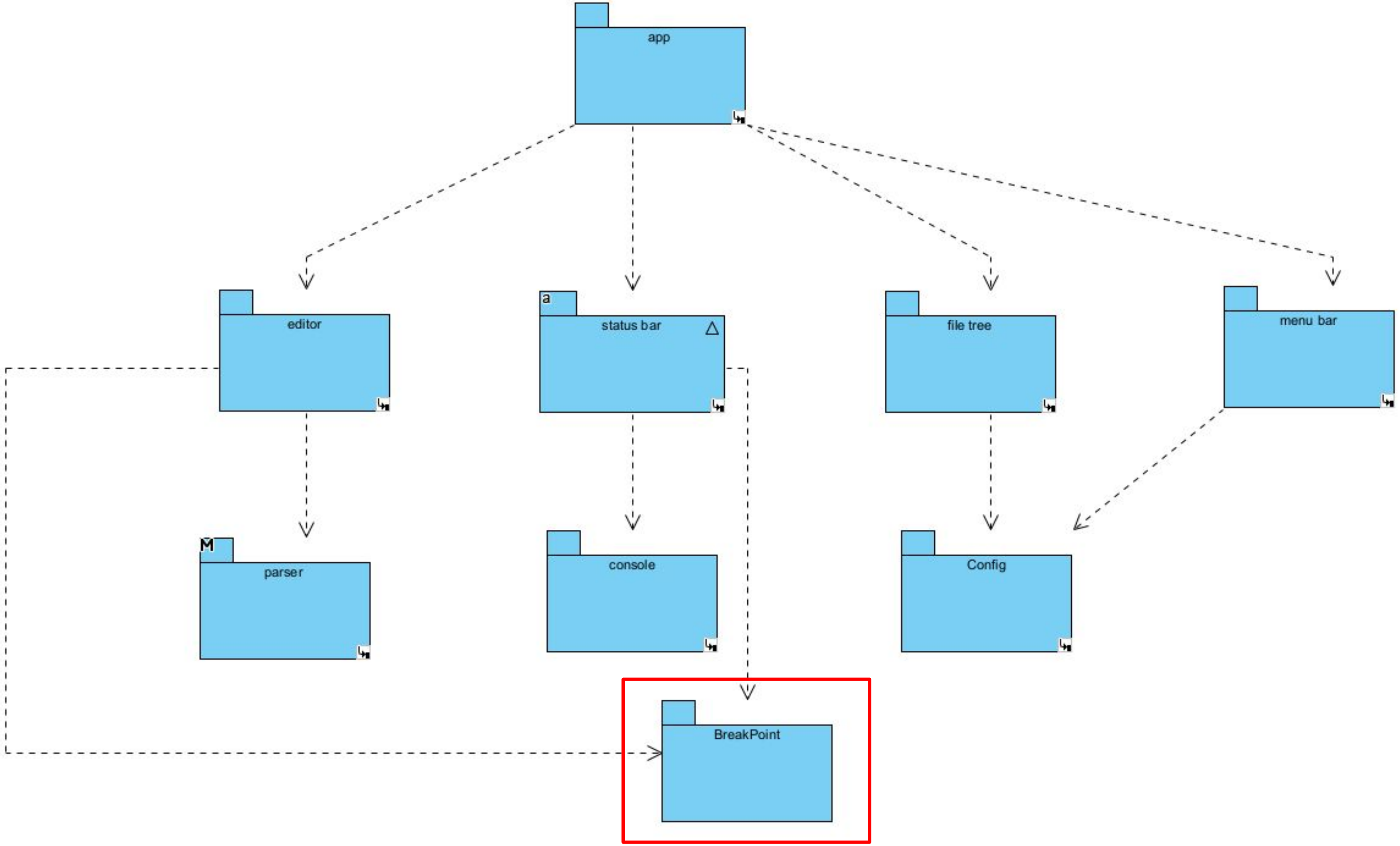
- Config Manager - manage project configuration file
- Config Parser - parse config file with configuration about build/run/debug
- Config Dialog - responsible for drawing the output


app
editor
a
status bar
file tree
menu bar
M
parser
console
Config
BreakPoint

FileTree

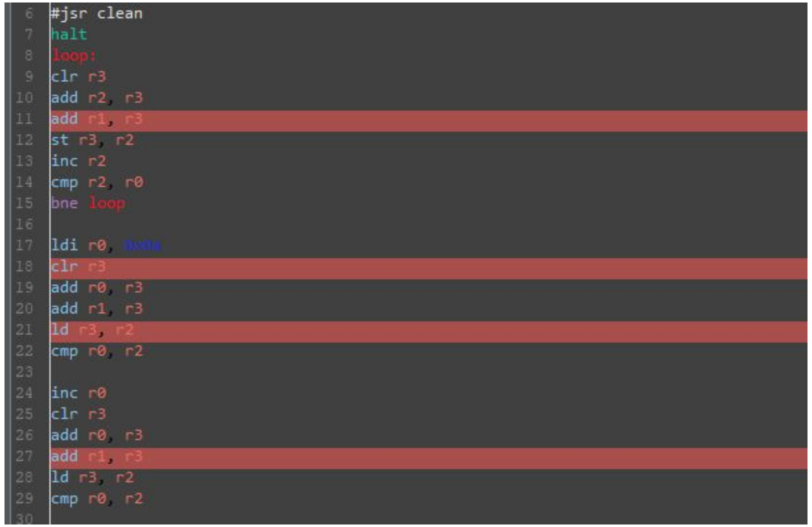
- FileDialog
- FileTree
- FileTreeView
- ExpandableFile




app
editor
a
status bar
file tree
menu bar
M
parser
console
Config
BreakPoint

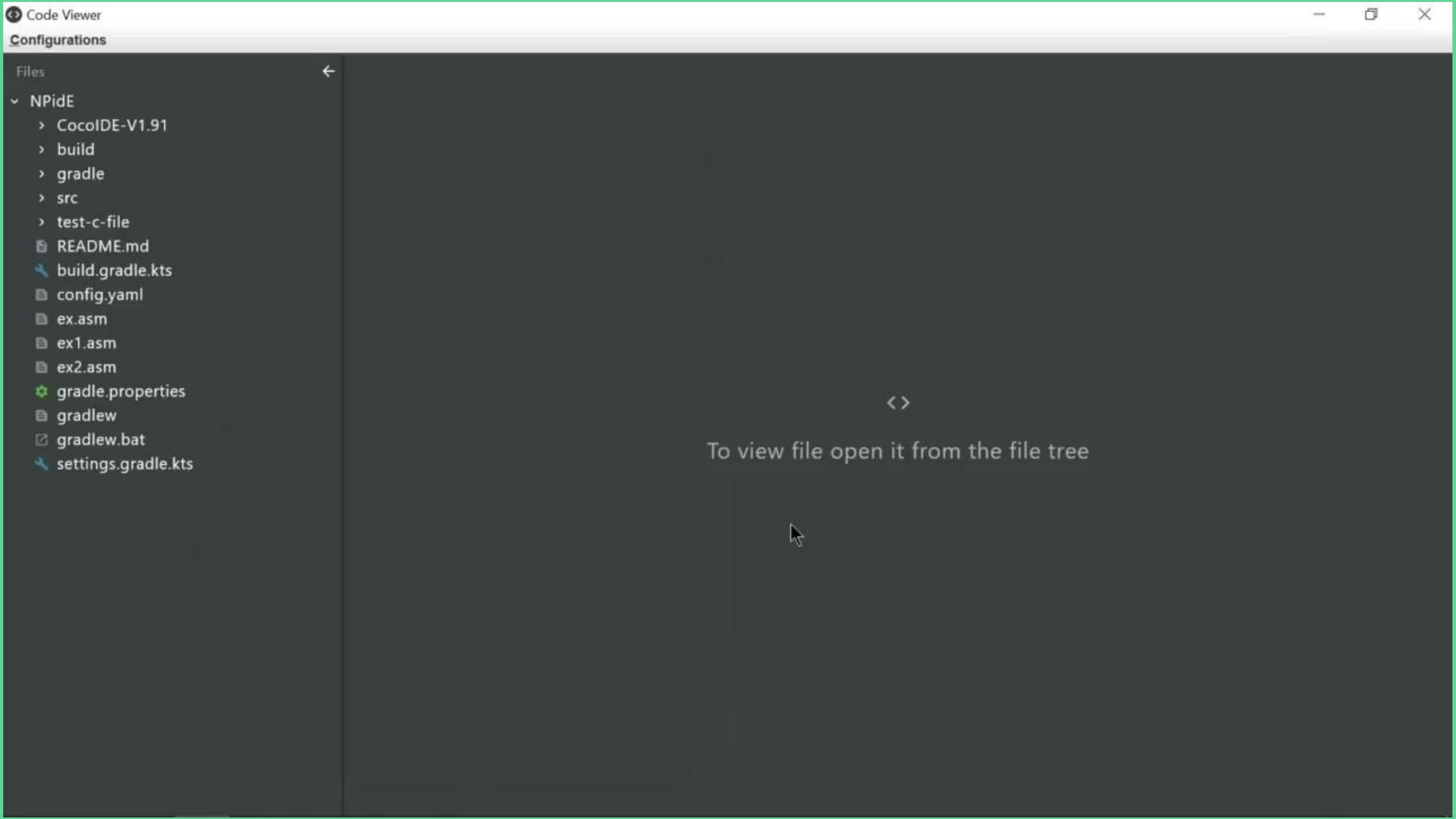
Breakpoints

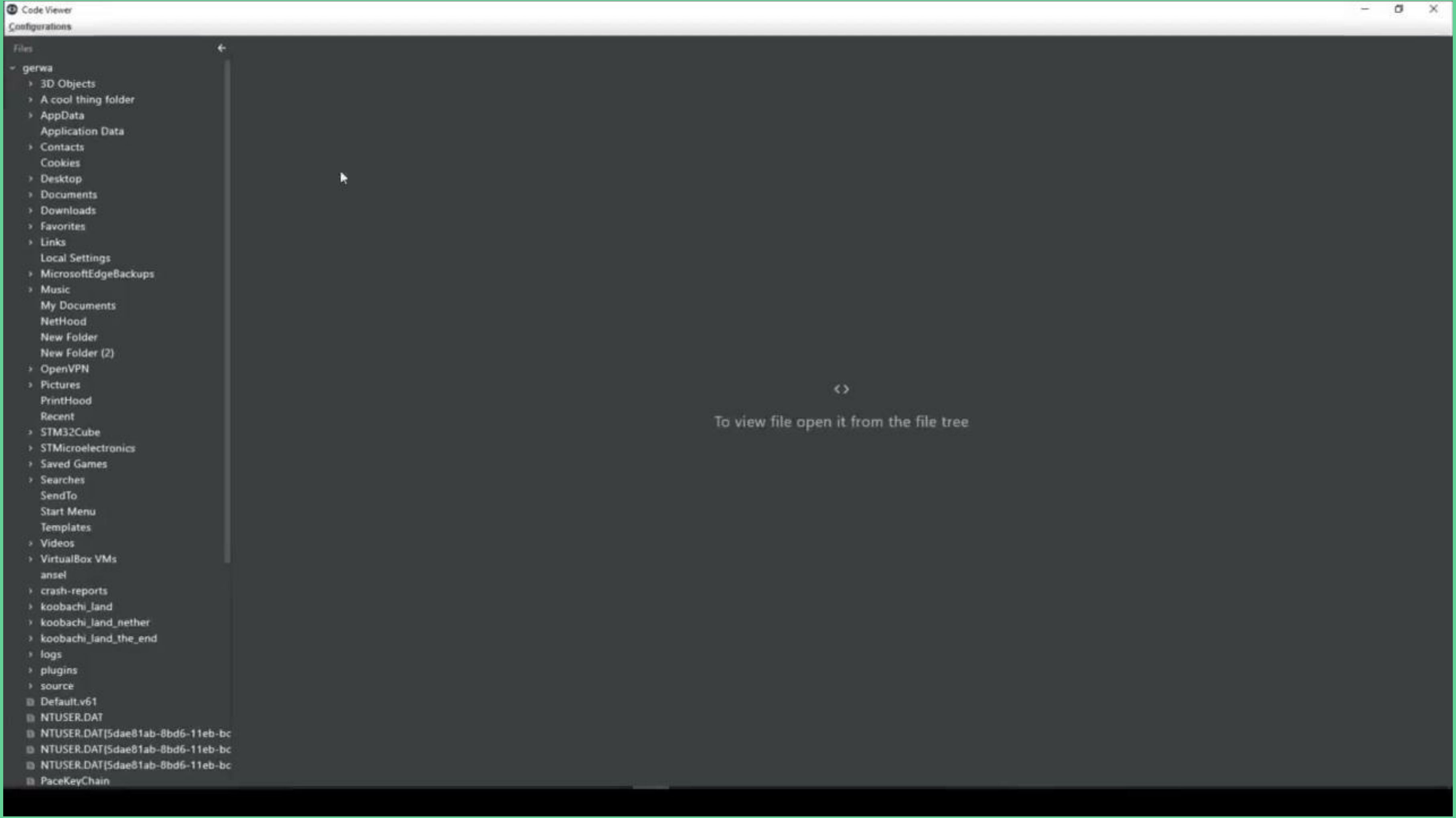
BreakpointStorage - responsible for adding/removing breakpoints



```
#jsr clean
halt
loop:
clr r3
add r2, r3
add r1, r3
st r3, r2
inc r2
cmp r2, r0
bne loop

ldi r0, 0x0a
clr r3
add r0, r3
add r1, r3
ld r3, r2
cmp r0, r2

inc r0
clr r3
add r0, r3
add r1, r3
ld r3, r2
cmp r0, r2
```

DEMO


Code Viewer
Configurations
Files
NPidE
CocoIDE-V1.91
build
gradle
src
test-c-file
README.md
build.gradle.kts
config.yaml
ex.asm
ex1.asm
ex2.asm
gradle.properties
gradlew
gradlew.bat
settings.gradle.kts
<>
To view file open it from the file tree



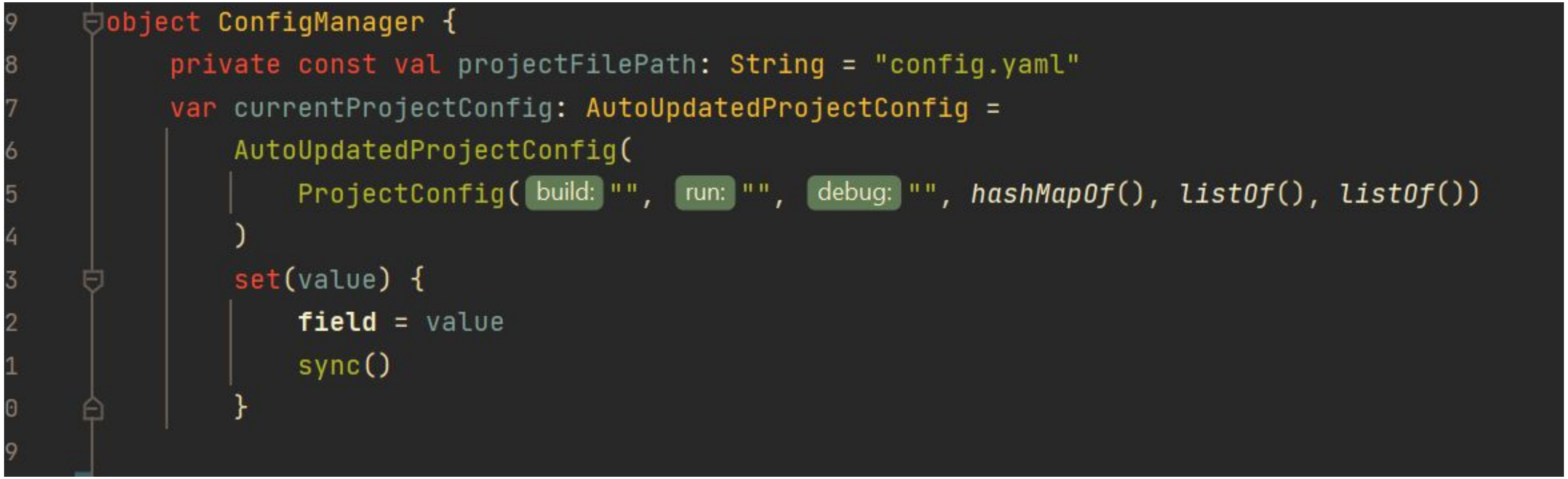
gerwa
3D Objects
A cool thing folder
AppData
Application Data
Contacts
Cookies
Desktop
Documents
Downloads
Favorites
Links
Local Settings
MicrosoftEdgeBackups
Music
My Documents
NetHood
New Folder
New Folder (2)
OpenVPN
Pictures
PrintHood
Recent
STM32Cube
STMicroelectronics
Saved Games
Searches
SendTo
Start Menu
Templates
Videos
VirtualBox VMs
ansel
crash-reports
koobachi_land
koobachi_land_nether
koobachi_land_the_end
logs
plugins
source
Default.v61
NTUSER.DAT
NTUSER.DAT{5dae81ab-8bd6-11eb-bc
NTUSER.DAT{5dae81ab-8bd6-11eb-bc
NTUSER.DAT{5dae81ab-8bd6-11eb-bc
PaceKeyChain

<>
To view file open it from the file tree

Design patterns

Singleton Pattern

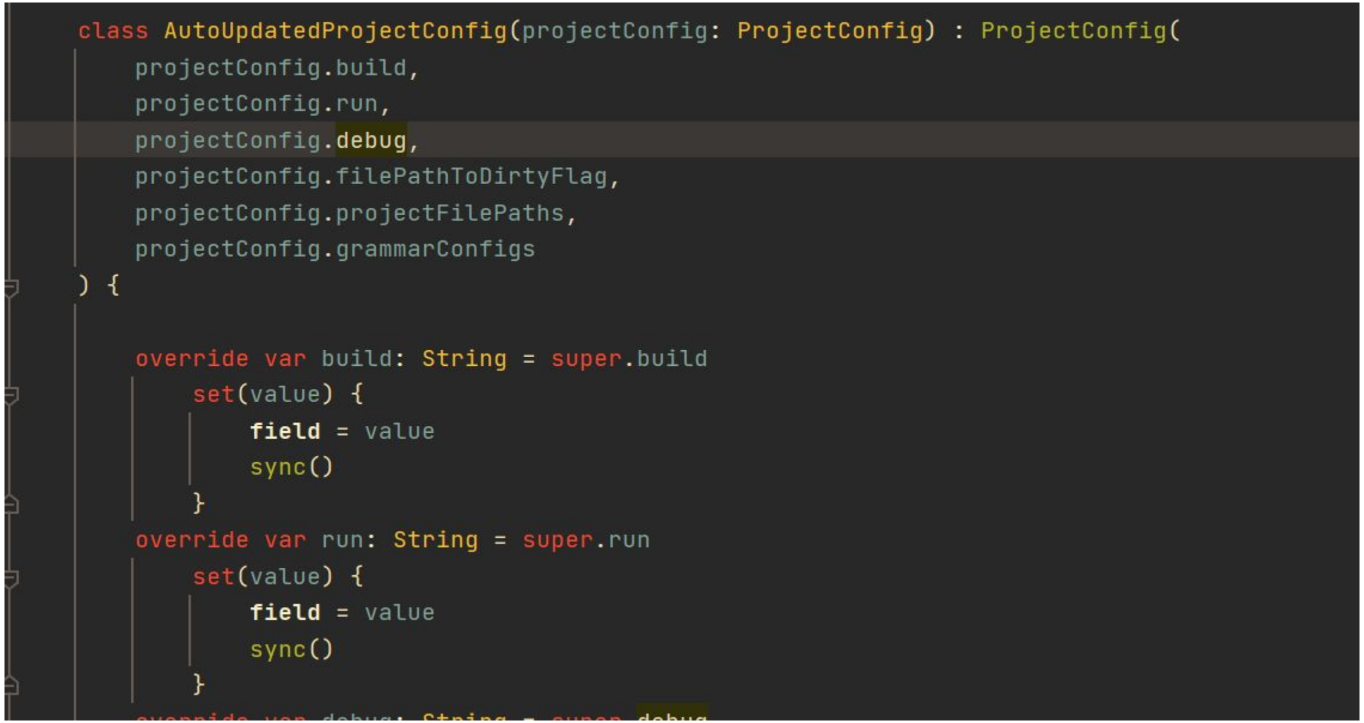
- ConfigManger
- BreakpointStorage
- Fonts
- AppTheme



```kotlin
object ConfigManager {
        private const val projectFilePath: String = "config.yaml"
        var currentProjectConfig: AutoUpdatedProjectConfig =
            AutoUpdatedProjectConfig(
                ProjectConfig( build: "", run: "", debug: "", hashMapOf(), listOf(), listOf())
            )
            set(value) {
                field = value
                sync()
            }
```

Proxy Pattern

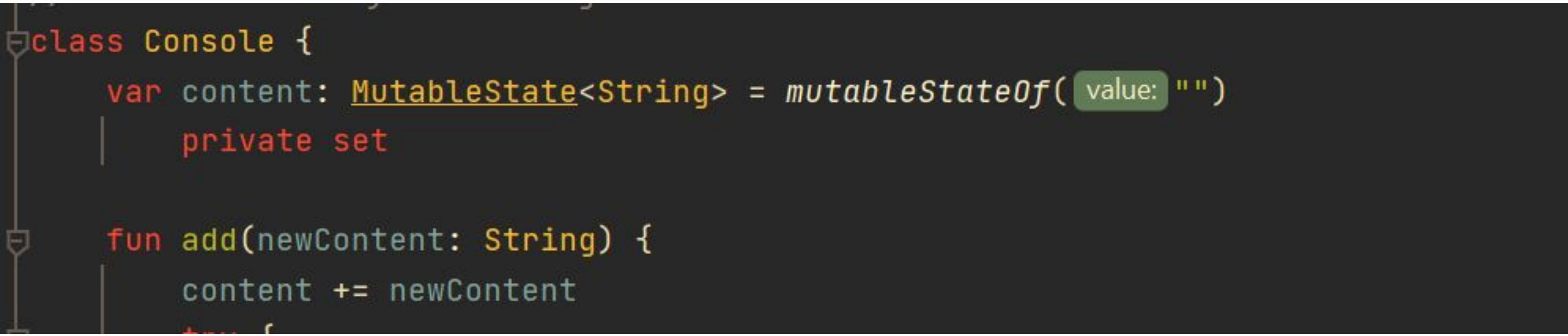
- Was used for adding “save on edit” functionality to the config class.



```kotlin
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
```

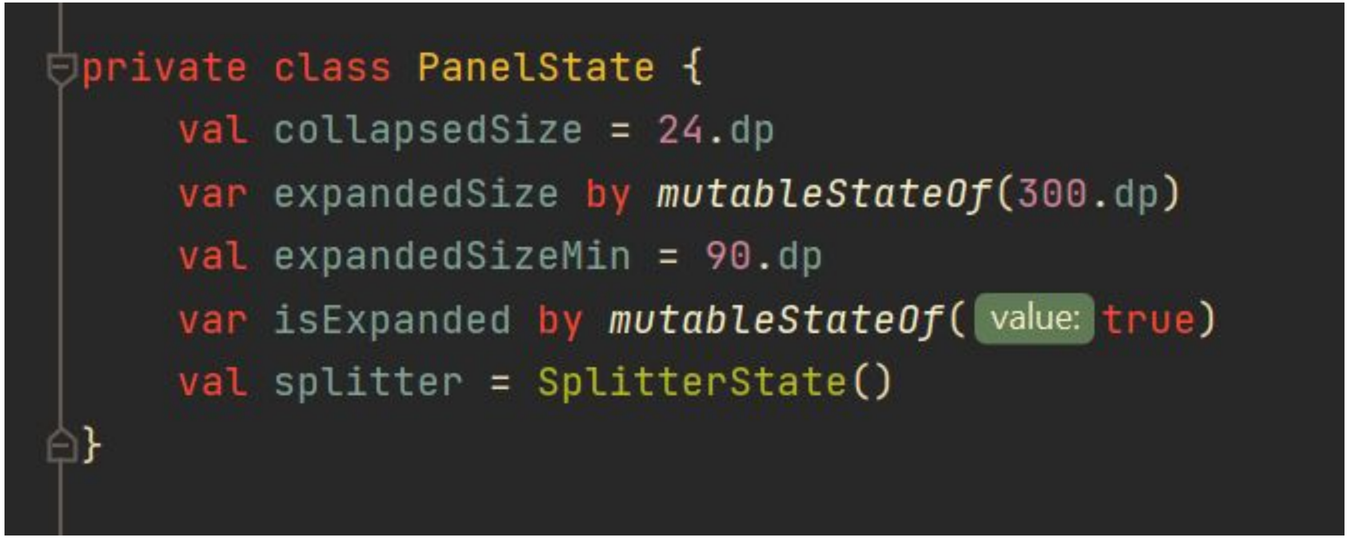
Observer pattern

All of our project



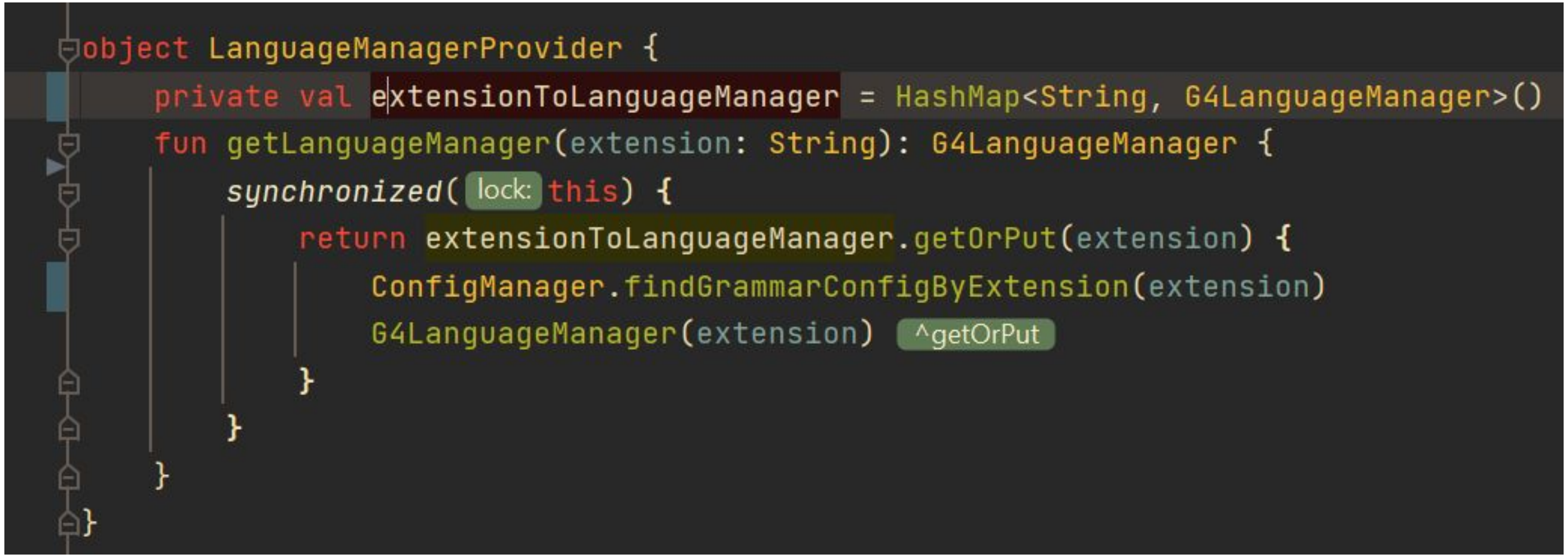
```
class Console {
    var content: MutableState<String> = mutableStateOf( value: "")
        private set

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

Delegation pattern



```kotlin
private class PanelState {
    val collapsedSize = 24.dp
    var expandedSize by mutableStateOf(300.dp)
    val expandedSizeMin = 90.dp
    var isExpanded by mutableStateOf( value: true)
    val splitter = SplitterState()
}
```

Object pool pattern



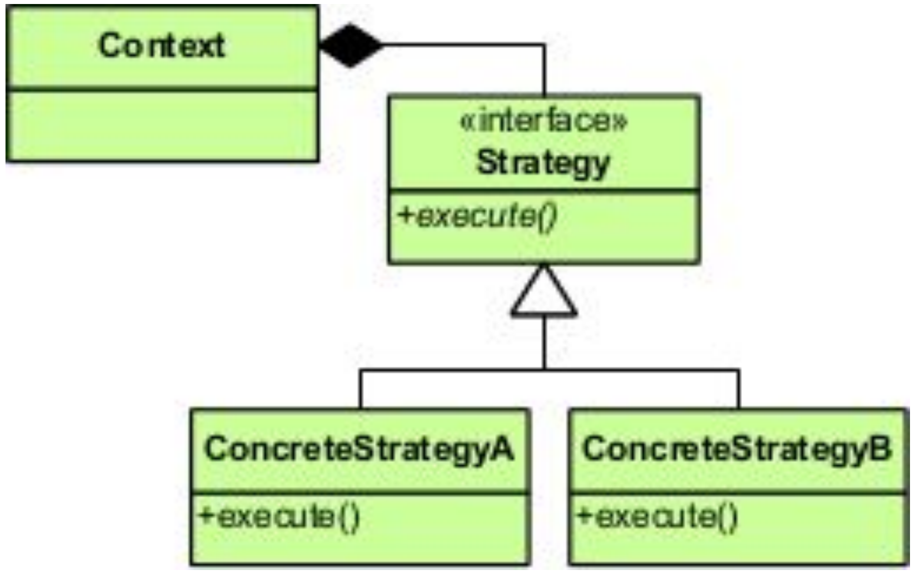
```
object LanguageManagerProvider {
    private val extensionToLanguageManager = HashMap<String, G4LanguageManager>()
    fun getLanguageManager(extension: String): G4LanguageManager {
        synchronized( lock: this) {
            return extensionToLanguageManager.getOrPut(extension) {
          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



Thank you for your attention!