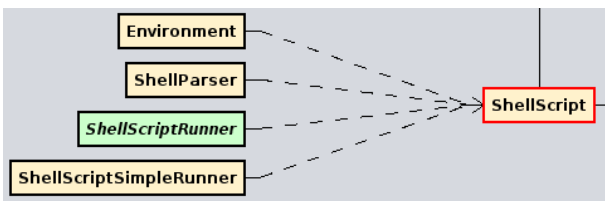


Описание структуры классов

Начнем с главного класса, Shell

Object	
com::au::mit::shell::common::	Shell
Fields	
- autoTaskFactory : AutoTaskFactory	
- commandRunner : CommandRunner	
- environment : Environment	
- parser : ShellParser	
- scriptRunner : ShellScriptRunner	
Constructors	
+ Shell(ShellParser, ShellScriptRunner, CommandRunner, Command[]) : void	
Methods	
+ start() : void	

Его мы создаем, с указанием нужных нам параметров (можно было сделать через builder, но и так слишком много классов уже) и запускаем через start(). В нем у нас хранятся основные объекты для парсинга и исполнения команд.

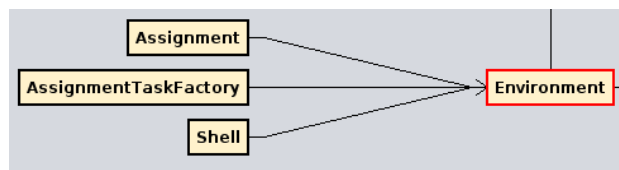


В процессе парсинга мы получаем ShellScript – по сути список TaskDescription, для команд, объединенных через pipe. А он, в свою очередь, хранит название команды и список ее аргументов.

Object	
com::au::mit::shell::common::command::tasks::	TaskDescription
Properties	
+ args : List<Argument>	
+ commandName : String	
Constructors	
+ TaskDescription(String, List<Argument>) : void	

После того как парсер отдал нам ShellScript, мы в нем заменяем переменные с помощью Environment.

Object	
com::au::mit::shell::common::	Environment
Fields	
- variables : Map<String, String>	
Constructors	
+ Environment() : void	
Methods	
+ addVariable(String, String) : void	
- replaceVariables(String) : String	
- replaceVariables(List<Argument>) : List<Argument>	
+ replaceVariables(ShellScript) : void	



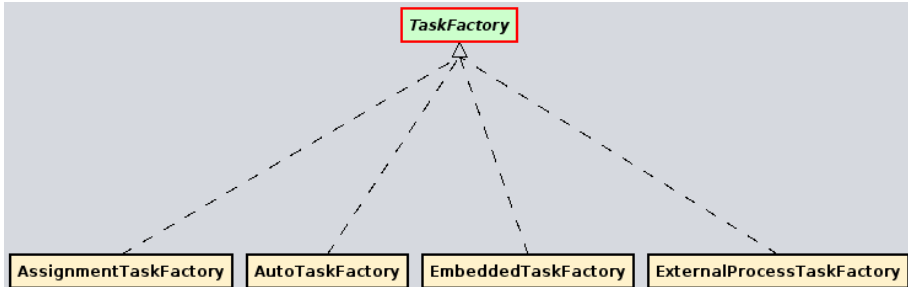
Далее, мы вызываем сохраненный в Shell'e ShellScriptRunner (интерфейс как всегда для гибкости, пока что реализует его только один класс)

«interface»	
com::au::mit::shell::common::scripts::runner::	ShellScriptRunner
Methods	
+ run(ShellScript, TaskFactory, CommandRunner) : void	

Сам runner должен с помощью TaskFactory из TaskDescription сделать непосредственно различные задачи

«interface» com::au::mit::shell::common::command::tasks:: TaskFactory
Methods + tryCreate(String, List<Argument>) : Task

При этом есть специальный класс AutoTaskFactory, которых хранит список фабрик, и пробует поочередно создавать с их помощью задачи, а именно вызывает у них метод tryCreate, который возвращает либо null, либо не null.



Далее, эти задачи отдаются в CommandRunner, и у этого интерфейса есть 2 реализации: SimpleCommandRunner и MultiThreadCommandRunner.

«interface» com::au::mit::shell::common::command::runner:: CommandRunner
Methods + run(Command, PipedInputStream, List<Argument>) : CommandResult + start() : void + stop() : void

Разница у них в том, что первый запускает команды разделенные pipe'ом поочередно, а второй запускает их в N потоков и они передают друг другу результаты выполнения через PipedInputStream и PipedOutputStream. Это позволяет делать такие вещи как:

cat /dev/urandom | echo

(Пример бесполезный, но если расширить список команд до filter и takeFirst, то будет осмысленней)

Не слишком премудрый объект Command:

Object com::au::mit::shell::common::command:: Command
Properties «readOnly» + name : String
Constructors + Command() : void
Methods + run(PipedInputStream, PipedOutputStream, List<Argument>) : void

