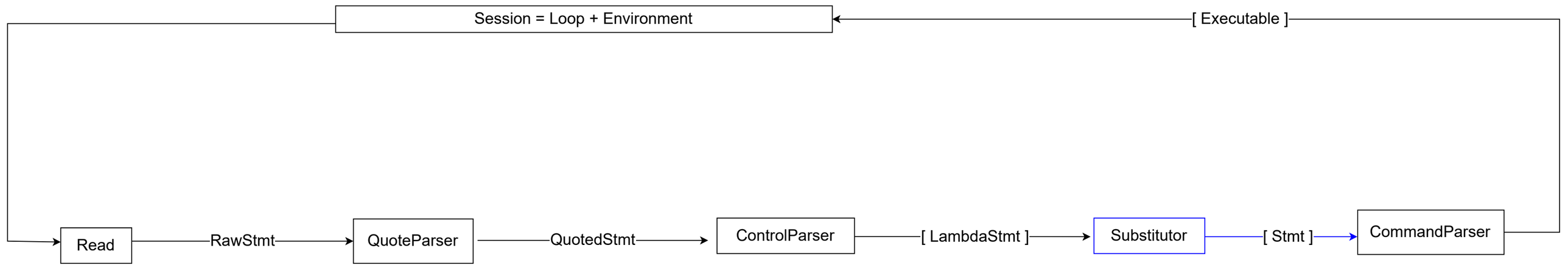
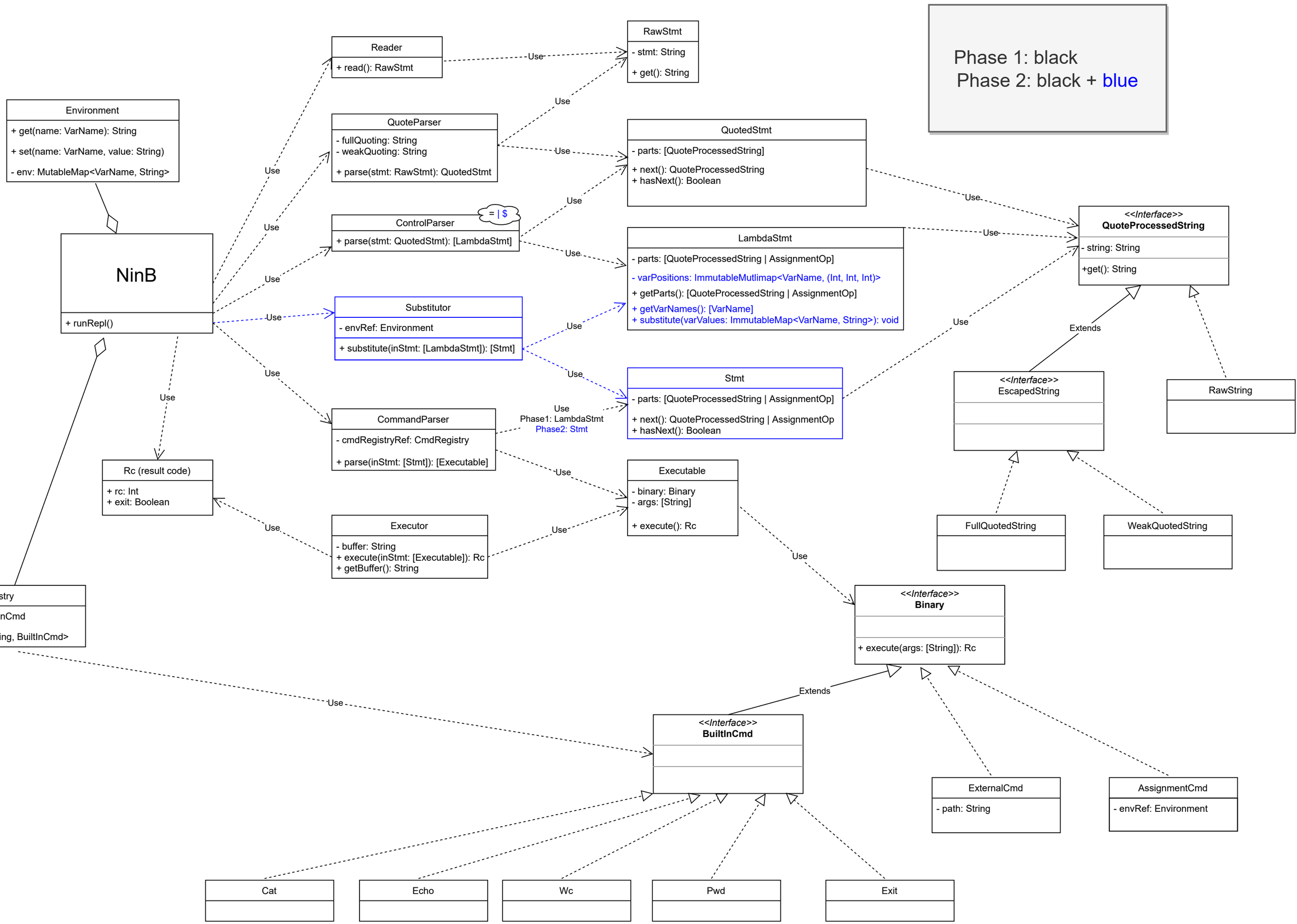
This page intentionally left blank

Data Flow Diagram

Phase 1: black
Phase 2: black + blue



Structural Diagram



Structural Diagram (Verbose)

1. NinB (NinB is not Bash) реализует основную логику: инициализация Environment, CmdRegistry и компонент (Reader, …, Executor), а также основной цикл.
  Тело цикла выглядит примерно так:
  rawStmt = reader.read();
  quotedStmt = quoteParser.parse(rawStmt);
  lambdaStmt = controlParser.parse(quotedStmt);
  stmt = substitutor.substitute(lambdaStmt);
  executable = commandParser.parse(stmt);
  resultCode = executor.execute(executable);

  if (resultCode…) { … }

 По сути, тело цикла отражает флоу данных через компоненты — это мы спроектировали на практике (V0).

2. Environment хранит словарь переменная-значение. В нашей постановке значение может быть только строкой. Environment должен быть доступен для Substitutor и Assignment.

3. Reader как-то обрабатывает пользовательский ввод, выдаёт RawStmt.

4. RawStmt — просто строка.

5. QuoteParser сканирует строку на наличие символов [' - fullQuoting | " - weakQuoting].
Результатом будет QuotedStmt -- набор объектов, состоящих из строки и степени её закавыченности.

6. ControlParser сканирует строку на наличие символов ['|' | $ | =], разбивая её по '|', выделяя подстроки для подстановки по '$' и заменяя вхождения '=' объектом AssignmentOp.

Результатом будет массив LambdaStmt.

7. LambdaStmt — массив подстрок и информация о возможных подстановках — словарь (имя переменной -> индекс подстроки с её вхождением, индекс первого символа, индекс последнего символа + 1), причём одному ключу может соответствовать множество значений. LambdaStmt соответствует одной инструкции (между пайпами).

Есть методы для самой подстановки. Пайплайн работы с LambdaStmt: получить набор ключей, найти соответствующие значения через Environment (если определённой переменной нет, подставляется пустая строка), передать этот подсловарь в substitute(...), после чего получить набор подстрок с выполненными подстановками через getParts().

8. Substitutor — через интерфейс LambdaStmt производит подстановку, конвертирует LambdaStmt в Stmt.

9. Stmt — массив подстрок, некоторые из которых могут быть помечены как закавыченные. Закавыченные строки не требуется разбирать CommandParser'y.

10. CommandParser — разбирает строку, учитывая пробельные символы. Конвертирует Stmt в executable, выделяя исполняемую команду и набор агрументов по следующему принципу: если Stmt содержит AssignmentOperator, порождает AssignmentCmd, иначе рассматривает первый терм и, если он соответствует одной из внутренних команд (т.е. занесён в CmdRegistry), порождает BuiltInCmd, иначе ExternalCmd.

11. Executable — исполняемая команда и набор аргументов. Если команда читает из stdin, последний аргумент трактуется как пользовательский ввод

12. Binary -- корень дерева иерархии исполняемых команд. Каждую команду можно исполнить с набором аргументов через execute(args).

Для однородности определение переменной (x=3) тоже рассматривается как команда, принимающая два аргумента: название переменной и её значение -- и обновляющая Environment.

Кроме того, имеется набор встроенных команд (cat, echo, ...). Для добавления новой команды достаточно добавить новый класс и зарегистрировать его в CmdRegistry.

Наконец, есть возможность вызвать сторонний бинарник -- ExternalCmd.

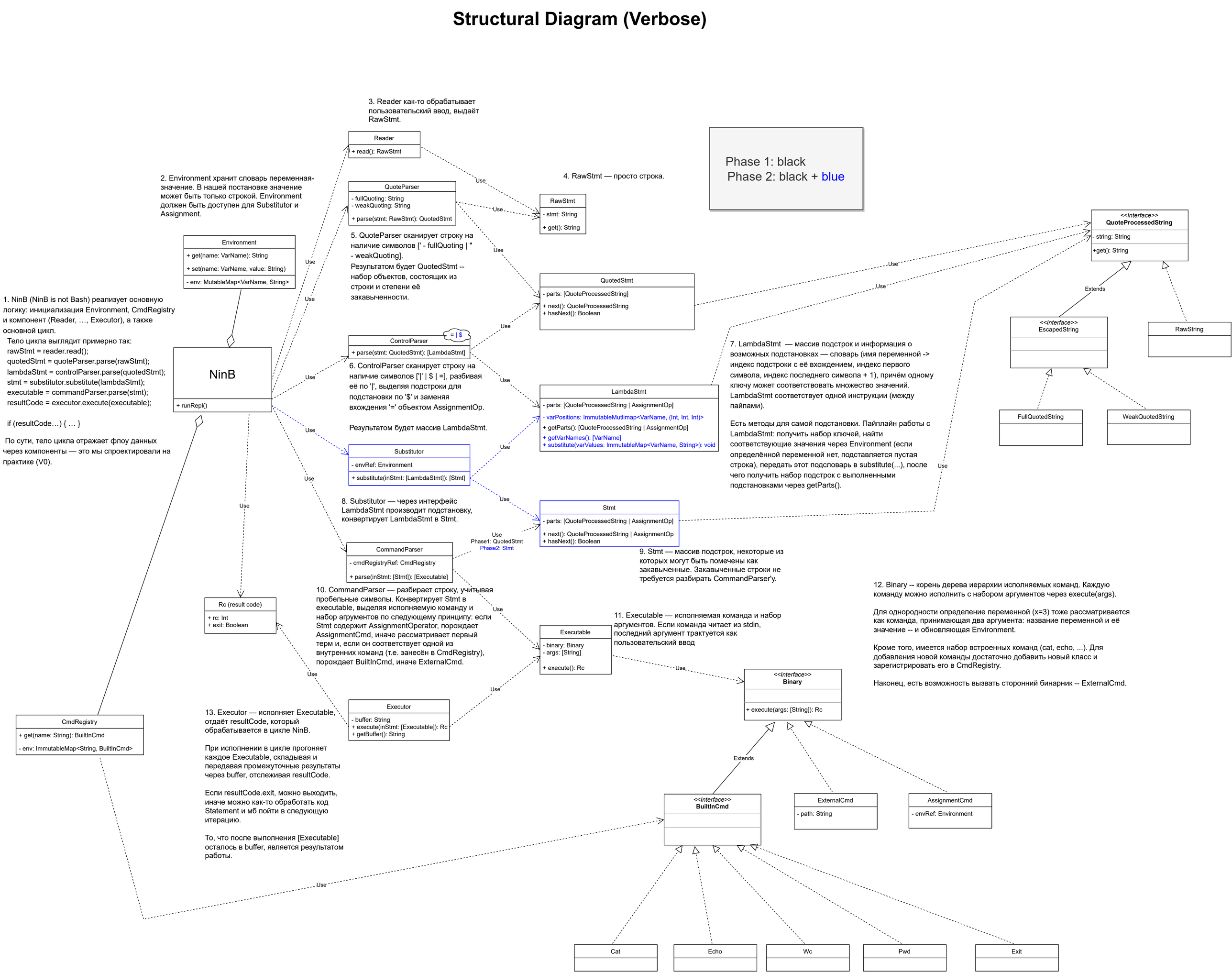
13. Executor — исполняет Executable, отдаёт resultCode, который обрабатывается в цикле NinB.

При исполнении в цикле прогоняет каждое Executable, складывая и передавая промежуточные результаты через buffer, отслеживая resultCode.

Если resultCode.exit, можно выходить, иначе можно как-то обработать код Statement и мб пойти в следующую итерацию.

То, что после выполнения [Executable] осталось в buffer, является результатом работы.

Phase 1: black
Phase 2: black + blue



This page intentionally left blank