Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**
МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/12 Интеллектуальный анализ больших данных в системах поддержки принятия решений**

О Т Ч Е Т
по лабораторной работе № 1

Название: Введение, классы, объекты

Дисциплина: Языки программирования для работы с большими данными

Студент ИУ6-23М _____ М.А. Гейне
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ П.В. Степанов
(Подпись, дата) (И.О. Фамилия)

Москва, 2023

ЗАДАНИЕ

4. Создать приложение для ввода пароля из командной строки и сравнения его со строкой-образцом.

5. Создать программу ввода целых чисел как аргументов командной строки, подсчета их суммы (произведения) и вывода результата на консоль.

Создание проекта

Лабораторные работы было решено выполнять на языке программирования Scala по согласованию с преподавателем.

Для лабораторных работ был создан общий проект инструмента сборки sbt, в котором для каждой лабораторной работы создаётся отдельный под-проект (Multi-project build). В конфигурационном файле указаны параметры сборки как общие для всех проектов, так и частные для каждого под-проекта. К общим параметрам относятся версия Scala и название организации. Для под-проектов устанавливаются такие параметры, как имена их директорий, имена проектов, их зависимости и параметры для выполнения заданий над ними. Текст файла конфигурации сборки sbt приведён в листинге 1.

Листинг 1 – Код build.sbt

```
ThisBuild / scalaVersion := "3.2.2"
ThisBuild / organization := "ru.bmstu"

lazy val general = (project in file("general"))
  .settings(
    name := "General"
  )

lazy val lab1 = (project in file("lab1"))
  .settings(
    name := "Lab 1",
    assembly / assemblyJarName := "lab1.jar",
    assembly / target := file("exec/")
  )
  .dependsOn(general)
```

В настоящий момент объявлено 2 проекта: general и lab1. В проекте general планируется хранить общие для всех проектов функции, которые могут быть многократно использованы. Примером такого объявления является оператор «|>», используемый в некоторых функциональных языках программирования (например, OCaml) для связывания выхода функции слева от оператора с входом функции справа от оператора. Весь код проекта в настоящее время записан в 1 файле и приведён в листинге 2.

Листинг 2 – Исходный код General.scala

```scala
package bmstu.general

import scala.util.chaining._

// a pipe operator from OCaml PL
extension [A,B](a:A)
  infix def |>(f: A => B): B = a.pipe(f)
```

В проекте lab1 были выполнены задания лабораторных работ 1 и 2. Проект организован в стиле стандартных проектов на языке Scala. Структура проекта представлена в листинге 3.

Листинг 3 – Структура проекта lab1

```
lab1/
├── src
│   └── main
│       └── scala
│           ├── main.scala
│           └── var1
│               ├── task1.scala
│               └── task2.scala
└── target
    ├── scala-3.2.2
    │   └── *output omitted*
    └── streams
        └── *output omitted*

182 directories, 92 files
```

В директории var1 хранятся исходные коды решений задач ЛР1. Программа main.scala является начальной точкой исполнения всех задач; её код приведён в листинге 4.

Листинг 4 – Исходный код main.scala

```
import var1._

@main def main(args: String* ): Unit =
  val arglist = args.toList
  arglist match {
    case "-p"::tasks::tail if isListOfInts(tail) => run_tasks(tasks,
(tail.map(_.toInt).toSeq))
    case "--help"::tail => print_help()
    case tail if isListOfInts(tail) => run_tasks("12",
(tail.map(_.toInt).toSeq))
    case tail => illegal_args(tail.mkString(" "))
  }

  def run_tasks(tasks: String, int: Seq[Int]) =
    tasks.split("").foreach { key =>
      key match {
      case "1" =>{
        println("Variant 1")
        println("Question 4")
        var1.task1.main()
      }
      case "2" => {
        println("Variant 1")
        println("Question 5")
        var1.task2.main(int)
      }
      case arg => illegal_args("Task № "+arg)
      }
      println("-------------------------")
    }

  def isListOfInts(lst: List[String]) =
    lst.map(_.toIntOption).filter(_.isEmpty).isEmpty

  def print_help() =
    println("""
    lab1 [OPTIONS] [INTEGERS]
      [OPTIONS]
        --help      Prints help message
        -p TASKS    Allows partial execution of tasks. TASK is a sting of
digits from 1 to 2.
      [INTEGERS]
        List of integers separated by space; used for Task 2 (Sum and mul)
    """)

  def illegal_args(arg: Any) =
    println("Illegal arguments found: "+arg)
    println("Usage:")
    print_help()
```

Аннотация @main позволяет описать точку входа в программу в виде метода вместо класса, как было необходимо делать в более ранних версиях Scala. В действительности будет создан класс с именем, совпадающим с именем метода; в этом классе будет создан метод main, совпадающий с сигнатурой метода main из Java с Array[String] в качестве аргумента и возвращаемым типом Unit; в сгенерированном методе main будет вызван изначально написанный метод с аргументами, разобранными из Array[String].

В файле описано простое приложение для работы в консольном режиме. Оно разбирает поступившие на вход аргументы командной строки, проверяет их на корректность и запускает нужные задания. Приложение позволяет запустить либо все задания одновременно, либо только некоторые из них, если будет задан ключ «-p». Программа также содержит краткое сообщение с инструкцией по использованию и сообщение об ошибке в случае, если что-то пошло не так. Далее рассмотрим решения задач ЛР1.

Задача 1

Текст задания: создать приложение для ввода пароля из командной строки и сравнения его со строкой-образцом.

В приложении задаётся строка-образец password. Приложение запрашивает у пользователя пароль, считывает его из командной строки и сравнивает с образцом. Результат сравнения выводится в терминал. Текст программы приведён в листинге 5.

Листинг 5 – Текст решения задачи 1

```
package var1.task1
import bmstu.general._
import scala.io.StdIn

val password = "1234567890"

def main() =
  println("Enter password: ")
  val input = StdIn.readLine()
  (if password.equals(input) then "Password is correct!" else "Password is
incorrect") |> (println(_))
```

Результат работы программы приведён на рисунке 1.

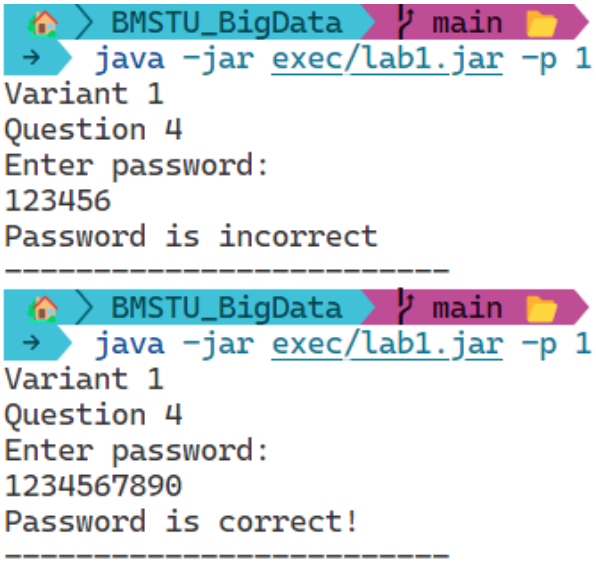

Рисунок 1 – Результат решения задачи 1

Задача 2

Текст задания: создать программу ввода целых чисел как аргументов командной строки, подсчета их суммы (произведения) и вывода результата на консоль.

Аргументы командной строки принимает основная программа. В ней предусмотрена проверка, что кроме аргументов ключа «-p» в аргументах командной строки содержатся только целые числа, выполняется преобразование аргументов в последовательность целых чисел (которая, теоретически, может быть и пустой), а затем вызывается нужная подпрограмма с этой последовательностью в аргументах.

Подпрограмма второй задачи в аргументах получает последовательность целых чисел и подсчитывает сумму и произведение с помощью функции fold и выводит в консоль. Первый аргумент fold должен быть нейтральным, т.е. для сложения 0, а для умножения 1. Текст решения приведён в листинге 6.

Листинг 6 – Текст решения задачи 2

```
package var1.task2
import scala.io.StdIn
import bmstu.general

def main(arr: Seq[Int] ): Unit =
  println(s"CML arguments sum: ${arr.fold(0)(_ + _)}")
  println(s"CML arguments mul: ${arr.fold(1)(_ * _)}")
```

Результат работы программы приведён на рисунке 2.

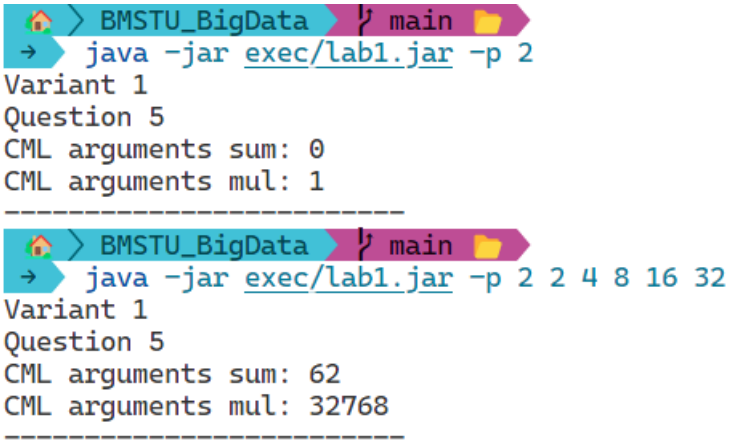

Рисунок 2 – Решение задачи 2

Рисунок 2 иллюстрирует, что возможно исполнение программы без аргументов в виде последовательности целых чисел.

ВЫВОД

Изучены базовые возможности и особенности языка программирования Scala.

Изучены способы работы со стандартным средством сборки проектов sbt. Изучены способы создания мульти-проектов, задания настроек для всего проекта и отдельно для под-проектов. Изучены методы сборки и запуска проектов.

Написано базовое консольное приложение, позволяющее производить частичное выполнение заданий, производящее разбор аргументов командной строки и имеющее встроенную справку. Изучены средства работы с вводом/выводом. Изучен ряд функций языка, в том числе функции высшего порядка map и fold.