МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Математическая кибернетика и информационные технологии»

Лабораторная работа №4

по дисциплине

Функциональное программирование

на тему

«Future и Promise»

Выполнил: студент группы

БВТ2201

Шамсутдинов Рустам

Москва 2025

Введение

Цель работы:

Освоить принципы асинхронного программирования в Scala, научиться использовать Future и Promise, выполнять параллельные вычисления, обрабатывать ошибки и комбинировать асинхронные задачи.

Задания:

Задание 1. Простой Future

Создать программу, в которой асинхронная операция (моделируемая задержкой) выполняется в Future.

1. Запустить вычисление,

2. Вывести сообщение о старте и завершении,

3. Показать результат асинхронной операции.

Задание 2. Работа с Promise

Создать Promise[Int] и связанный с ним Future.

1. В отдельном потоке выполнить вычисление (например, умножение двух чисел);

2. Завершить Promise вручную успешным результатом или ошибкой;

3. Показать, как Future реагирует на выполнение Promise.

Ход работы

Задание 1. Создать программу, в которой асинхронная операция (моделируемая задержкой) выполняется в Future.

```scala
import scala.concurrent._
import ExecutionContext.Implicits.global
import scala.concurrent.duration._
import scala.util.{Success, Failure}

object AsyncLab {

  def runTask1(): Unit = {
    println("\ntask 1:\n")

    val futureResult = Future {
      println("Task 1 - Async computation started")
      Thread.sleep(2000)
      println("Task 1 - Async computation finished")
      42
    }

    val result = Await.result(futureResult, 3.seconds)
    println(s"Task 1 - Result: $result")
    println("Task 1 - Completed")
  }
```

Рис. 1 – простой Future.

Задание 2. Создать Promise[Int] и связанный с ним Future.

```scala
def runTask2(): Unit = {
  println("\ntask 2:\n")

  val promise = Promise[Int]()
  val promiseFuture = promise.future

  Future {
    println("Task 2 - Separate thread computation started")
    try {
      val a = 26
      val b = 2
      Thread.sleep(1000)
      val result = a * b
      promise.success(result)
      println("Task 2 - Computation finished, promise fulfilled")
    } catch {
      case e: Exception =>
        promise.failure(e)
        println("Task 2 - Error during computation, promise failed")
    }
  }

  val finalResult = Await.result(promiseFuture, 3.seconds)
  println(s"Task 2 - Promise future result: $finalResult")
  println("Task 2 - Completed")
}

```

Рис. 2 – работа с Promise.

```scala
  def main(args: Array[String]): Unit = {
    runTask1()
    runTask2()
  }
}

```

Рис. 3 – вызов функий

```
task 1:

Task 1 - Async computation started
Task 1 - Async computation finished
Task 1 - Result: 42
Task 1 - Completed

task 2:

Task 2 - Separate thread computation started
Task 2 - Computation finished, promise fulfilled
Task 2 - Promise future result: 52
Task 2 - Completed
[success] Total time: 3 s, completed Nov 27, 2025, 3:42:32 AM
[info] 6. Monitoring source files for root/run...
[info]    Press <enter> to interrupt or '?' for more options.
[info] Received input event: CancelWatch.
sbt:async>
```

Рис. 4 – результат

Заключение

Вывод: проделав работу, мы освоили принципы асинхронного программирования в Scala, научились использовать Future и Promise, выполнять параллельные вычисления, обрабатывать ошибки и комбинировать асинхронные задачи.