Федеральное агентство связи

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра Математической Кибернетики и Информационных Технологий

Отчет по курсовой работе

по предмету «Функциональное программирование»

Выполнил: студент группы

БВТ1802

Денисович Лев Сергеевич

Руководитель:

Мосева Марина Сергеевна

Москва 2020

Задание на курсовую работу:

Необходимо реализовать проект, состоящий из выполненных лабораторных, реализовать в рамках проекта тестирование каждой из лабораторных по отдельности и совместно всего проекта.

Исходный код доступен по ссылке:

https://github.com/NiceNickname/FPCourse

build.sbt

```
lazy val commonSettings = Seq(
  version := "1.0",
  scalaVersion := "2.13.2",
  libraryDependencies +=
    "org.scalatest" %% "scalatest" % "3.0.8" % Test,
)

lazy val root = project
  .in(file("."))
  .aggregate(lab1, lab2, lab3, lab4)
  .settings(commonSettings)

lazy val lab1 = project
  .settings(commonSettings)

lazy val lab2 = project
  .settings(commonSettings)

lazy val lab3 = project
  .settings(commonSettings)

lazy val lab4 = project
  .settings(commonSettings)
```

Данный build.sbt содержит корневой проект и 4 подпроекта, каждый из которых соответствует одной из выполненных лабораторных работ. Кроме того, в проекте используется библиотека Scalatest, позволяющая писать тесты для написанного кода.

Корневой проект

В корневом проекте файлов с исходным кодом нет, потому что все файлы с кодом относятся к лабораторным работам и, как следствие, находятся в подпроектах.

Лабораторная работа 1

В данной работе находятся 4 файла с исходным кодом и 4 файла с тестами.

Файлы с исходным кодом:

Файл Classes.scala

```scala
package exercise1

sealed trait Animal {
  val name: String
  val food: String
  def eats(food: String): Boolean = return this.food.equals(food)
}

case class Mammals(name: String, food: String) extends Animal
case class Birds(name: String, food: String) extends Animal
case class Fishs(name: String, food: String) extends Animal

object Animal {

  sealed trait Food

  case object Meat extends Food
  case object Vegetables extends Food
  case object Plants extends Food

  val cat = Mammals("cat", "meat")
  val parrot = Birds("parrot", "vegetables")
  val goldfish = Fishs("goldfish", "seaweed")

  def knownAnimal(name: String): Boolean =
    name.equals(cat.name) || name.equals(parrot.name) ||
    name.equals(goldfish.name)

  def apply(name: String): Option[Animal] = {
    name match {
      case cat.name => Some(cat)
      case parrot.name => Some(parrot)
      case goldfish.name => Some(goldfish)
      case other => None
    }
  }
}
```

Файл Functions.scala

```scala
`package exercise1

/** Напишите отдельные функции, решающие поставленную задачу.
  *
  * Синтаксис:
```

```
  *   // метод
  *   def myFunction(param0: Int, param1: String): Double = // мело
  *
  *   // значение
  *   val myFunction: (Int, String) => Double (param0, param1) => // мело
  */
object Functions{

  /* a) Напишите функцию, которая рассчитывает площадь окружности
   *    r^2 * Math.PI
   */

  def CircleArea(r: Double): Double = r * r * Math.PI


  // примените вашу функцию из пункта (a) здесь, не изменяя сигнатуру
  def testCircle(r: Double): Double = CircleArea(r)


  /* b) Напишите карированную функцию которая рассчитывает площадь прямоугольника a *
b.
   */

  def RectangeAreaCurried(a: Double)(b:Double) = a * b

  // примените вашу функцию из пукта (b) здесь, не изменяя сигнатуру
  def testRectangleCurried(a: Double, b: Double): Double = RectangeAreaCurried(a)(b)


  // c) Напишите не карированную функцию для расчета площади прямоугольника.

  def RectangleArea(a: Double, b:Double): Double = a * b

  // примените вашу функцию из пункта (c) здесь, не изменяя сигнатуру
  def testRectangleUc(a: Double, b: Double): Double = RectangleArea(a, b)
}
```

Файл HiOrder.scala

```
package exercise1

/** Напишите ваши решения в виде функций. */
object HigherOrder{

  /* a) Напишите функцию, которая принимает `f: (Int, Int) => Int`, параменты `a` и
`b`
   *    и коэффициент умножения `n` и возвращает n * f(a, b). Назовите `nTimes`.
   */

  def nTimes(f: (Int, Int) => Int, a: Int, b: Int, n: Int): Int = n * f(a, b)

  // примените вашу функцию (a) здесь, не изменяйте сигнатуру
  def testNTimes(f: (Int, Int) => Int, a: Int, b: Int, n: Int): Int = nTimes(f, a, b,
n)

  /* b) Напишите анонимную функцию, функцию без идентификатора ((a, b) => ???) для
`nTimes` которая
   *    выполняет следующее:
   *          if (a > b) a else b
   */
```

```
  def testAnonymousNTimes(a: Int, b: Int, n: Int): Int = nTimes((a: Int,b: Int) =>
  { if (a > b) a else b },  a: Int, b: Int, n: Int)

}
```

Файл Patterns.scala

```
package exercise1

/** Напишите решение в виде функции.
 *
 * Синтаксис:
 *   val a: Int = ???
 *
 *   a match {
 *     case 0 => true
 *     case _ => false
 *   }
 */
object PatternMatching {

  sealed trait Hand
  case object Rock    extends Hand
  case object Paper   extends Hand
  case object Scissor extends Hand

  sealed trait Result
  case object Win  extends Result
  case object Lose extends Result
  case object Draw extends Result

  sealed trait Food
  case object Meat       extends Food
  case object Vegetables extends Food
  case object Plants     extends Food

  sealed trait Animal {

    val name: String
    var food: Food
  }
  case class Mammal(name: String, var food: Food, weight: Int) extends Animal
  case class Fish(name: String, var food: Food)                extends Animal
  case class Bird(name: String, var food: Food)                extends Animal



  /* a) Напишите функцию, которая ставит в соответствие числу строку слудеющим
образом:
   * Если:
   *     1 => "it is one"
   *     2 => "it is two"
   *     3 => "it is three"
   *     иначе => "what's that"
   */

  def intToString(value: Int): String =
    value match {
      case 1 => "it is one"
      case 2 => "it is two"
      case 3 => "it is three"
      case other => "what's that"
```

```
    }



  // примените вашу функцию из пункта (a) здесь, не изменяя сигнатуру
  def testIntToString(value: Int): String = intToString(value)

  /* b) Напишите функцию которая возвращает true если переменная `value` принимает
значение:
   *     "max" или "Max
   *     "moritz" или "Moritz"
   */

  def isMaxAndMoritz(value: String): Boolean =
    value match {
      case "max" | "Max" | "moritz" | "Moritz"=> true
      case other => false
    }

  // примените функции из пункта (b) здесь, не изменяя сигнатуру
  def testIsMaxAndMoritz(value: String): Boolean = isMaxAndMoritz(value)

  // c) Напишите функцию проверки является ли `value` четным

  def isEven(value: Int): Boolean =
  value % 2 match {
    case 0 => true
    case 1 => false
  }

  // примените функции из пункта (c) здесь, не изменяя сигнатуру
  def testIsEven(value: Int): Boolean = isEven(value)



  /* d) Напишите функцию, моделирующую игру в Камень ножницы бумага
   *     1. камень побеждает ножницы
   *     2. ножницы побеждают бумагу
   *     3. бумага побеждает камень
   *    Выиграет ли игрок `a`?
   */

  def winsA(a: Hand, b: Hand): Result =
    a match {
      case Rock => b match {
        case Rock => Draw
        case Paper => Lose
        case Scissor => Win
      }
      case Paper => b match {
        case Rock => Win
        case Paper => Draw
        case Scissor => Lose
      }
      case Scissor => b match {
        case Rock => Lose
        case Paper => Win
        case Scissor => Draw
      }
    }
  // примените вашу функцию из пункта (d) здесь, не изменяя сигнатуру
  def testWinsA(a: Hand, b: Hand): Result = winsA(a, b)
```

```
  // Примечание: используйте определение Animals

  // e) Верните вес (weight: Int) объекта Mammal, иначе верните -1.

  def extractMammalWeight(animal: Animal): Int =
    animal match {
      case mammal: Mammal => mammal.weight
      case other => -1
    }

  // примените функцию из пункта (e) здесь, не изменяйте сигнатуру
  def testExtractMammalWeight(animal: Animal): Int = extractMammalWeight(animal)


  // f) Измените поле еда объектов классов Fishes и exercise1.Birds на Plants, класс
Mammels оставьте неизмененным.

  def updateFood(animal: Animal): Animal =
    animal match {
      case fish: Fish => fish.food = Plants; fish
      case bird: Bird => bird.food = Plants; bird
      case other => animal
    }

  // примените функцию из пункта (f) здесь, не изменяйте сигнатуру
  def testUpdateFood(animal: Animal): Animal = updateFood(animal)
}
```

Файлы с тестами

Файл AnimalTest.scala

```
package exercise1

import org.scalatest.FunSuite

class AnimalTest extends FunSuite {

  test("Animal.apply creates animal if valid parameter is passed") {
    assert(Animal.apply("cat") == (Some(Animal.cat)))
    assert(Animal.apply("parrot") == (Some(Animal.parrot)))
    assert(Animal.apply("goldfish") == (Some(Animal.goldfish)))
  }
  test("Animal.apply returns None if wrong parameter is passed") {
    assert(Animal.apply("wrongParam") == None)
  }

  test("KnownAnimal returns true if valid animal is passed") {
    assert(Animal.knownAnimal("cat") && Animal.knownAnimal("parrot") &&
    Animal.knownAnimal("goldfish") == true)
  }

  test("KnownAnimal returns false if invalid animal is passed") {
    assert(Animal.knownAnimal("tiger") == false)
  }
  test("Animal.eats checks if animal eats given food") {
```

```
    val lion = Mammals("lion", "meat")
    assert(lion.eats("meat") == true)
    assert(lion.eats("plants") == false)
  }
}
```

Файл FunctionsTest.scala

```
package exercise1

import org.scalatest.FunSuite

class FunctionsTest extends FunSuite {
  test("CircleArea calculates circle area")  {
    assert(Functions.CircleArea(5) == 5 * 5 * Math.PI)
  }

  test("testCircle calls circleArea with given radius")  {
    assert(Functions.testCircle(5) == 5 * 5 * Math.PI)
  }

  test("RectangleArea calculates area of the given rectangle")  {
    assert(Functions.RectangleArea(2, 3) == 6)
  }

  test("testRectangleUc calls RectangleArea with given parameters")  {
    assert(Functions.testRectangleUc(2, 3) == 6)
  }

  test("RectangleAreaCurried calculates area of the given rectangle")  {
    assert(Functions.RectangeAreaCurried(2)(3) == 6)
  }

  test("testRectangleCurried calls RectangleAreaCurried with given parameters")  {
    assert(Functions.testRectangleCurried(2, 3) == 6)
  }
}
```

Файл HigherOrderTest.scala

```
package exercise1

import org.scalatest.FunSuite

class HigherOrderTest extends FunSuite {

  test("nTimes multiplies function result by n") {
    assert(HigherOrder.nTimes((a: Int, b:Int) => a + b, 2, 3, 4) == 20)
  }
  test("testNtimes calls nTimes with given parameters") {
    assert(HigherOrder.testNTimes((a: Int, b: Int) => a + b,  2, 3, 4)  == 20)
  }

  test("testAnonymousNTimes calls nTimes with given parameters and hardcoded
function") {
    assert(HigherOrder.testAnonymousNTimes(2, 3, 4)  == 12)
    assert(HigherOrder.testAnonymousNTimes(3, 2, 4)  == 12)
  }
```

```
}
```

Файл PatternsTest.scala

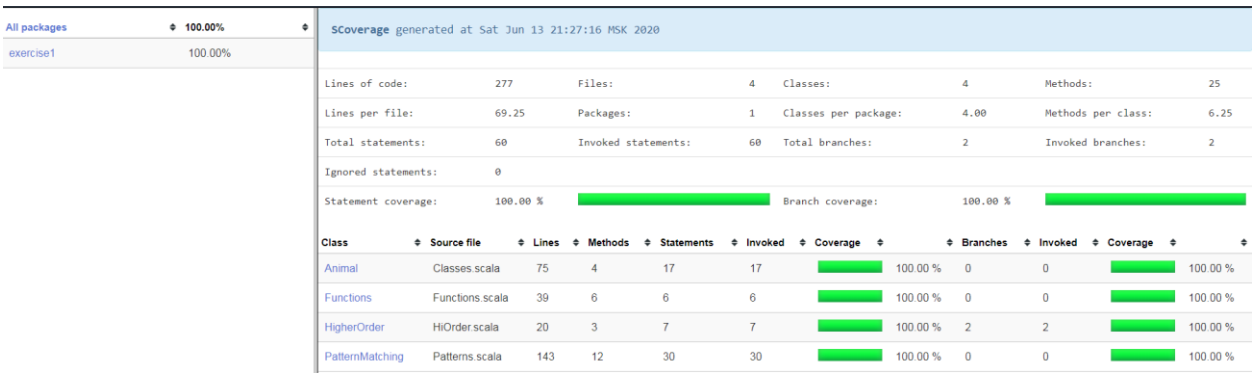
```
package exercise1

import org.scalatest._

class PatternsTest extends FunSuite {

  test("intToString recognizes numbers from 1 to 3 inclusively") {
    assert(PatternMatching.intToString(1) == "it is one")
    assert(PatternMatching.intToString(2) == "it is two")
    assert(PatternMatching.intToString(3) == "it is three")
    assert(PatternMatching.intToString(102019) == "what's that")
  }

  test("testIntToString calls intToString with given parameters") {
    assert(PatternMatching.testIntToString(1) == "it is one")
    assert(PatternMatching.testIntToString(2) == "it is two")
    assert(PatternMatching.testIntToString(3) == "it is three")
    assert(PatternMatching.testIntToString(102019) == "what's that")
  }

  test("isMaxAndMoritz recognizes (M)max and (M)moritz") {
    assert(PatternMatching.isMaxAndMoritz("max") &&
    PatternMatching.isMaxAndMoritz("Max") &&
    PatternMatching.isMaxAndMoritz("moritz") &&
    PatternMatching.isMaxAndMoritz("Moritz")  == true)

    assert(PatternMatching.isMaxAndMoritz("notMax")  == false)
  }

  test("testIsMaxAndMoritz calls isMaxAndMoritz with given parameter") {
    assert(PatternMatching.testIsMaxAndMoritz("max") &&
      PatternMatching.testIsMaxAndMoritz("Max") &&
      PatternMatching.testIsMaxAndMoritz("moritz") &&
      PatternMatching.testIsMaxAndMoritz("Moritz")  == true)

    assert(PatternMatching.testIsMaxAndMoritz("notMax")  == false)
  }

  test("isEven returns true if number is even and false otherwise") {
    assert(PatternMatching.isEven(10)  == true)
    assert(PatternMatching.isEven(11)  == false)
  }

  test("testIsEven calls isEven with given parameter") {
    assert(PatternMatching.testIsEven(10)  == PatternMatching.isEven(10))
    assert(PatternMatching.testIsEven(11)  == PatternMatching.isEven(11))
  }

  test("winsA returns rock-paper-scissor game result for first player") {
    assert(PatternMatching.winsA(PatternMatching.Rock, PatternMatching.Rock)  ==
PatternMatching.Draw)
    assert(PatternMatching.winsA(PatternMatching.Rock, PatternMatching.Paper)  ==
PatternMatching.Lose)
    assert(PatternMatching.winsA(PatternMatching.Rock, PatternMatching.Scissor)  ==
```

```
PatternMatching.Win)
    assert(PatternMatching.winsA(PatternMatching.Paper, PatternMatching.Rock)  ==
PatternMatching.Win)
    assert(PatternMatching.winsA(PatternMatching.Paper, PatternMatching.Paper)  ==
PatternMatching.Draw)
    assert(PatternMatching.winsA(PatternMatching.Paper, PatternMatching.Scissor)  ==
PatternMatching.Lose)
    assert(PatternMatching.winsA(PatternMatching.Scissor, PatternMatching.Rock)  ==
PatternMatching.Lose)
    assert(PatternMatching.winsA(PatternMatching.Scissor, PatternMatching.Paper)  ==
PatternMatching.Win)
    assert(PatternMatching.winsA(PatternMatching.Scissor, PatternMatching.Scissor)
== PatternMatching.Draw)
  }

  test("testWinsA calls winsA with given parameters") {
    assert(PatternMatching.testWinsA(PatternMatching.Rock, PatternMatching.Scissor)
==
    PatternMatching.winsA(PatternMatching.Rock, PatternMatching.Scissor))
  }

  test("extractMammalWeight returns mammal's weight and -1 if not mammal is passed")
{
    assert(PatternMatching.extractMammalWeight(PatternMatching.Mammal("cat",
PatternMatching.Meat, 5))  == 5)
    assert(PatternMatching.extractMammalWeight(PatternMatching.Fish("goldfish",
PatternMatching.Vegetables))  == -1)
  }

  test("testExtractMammalWeight calls extractMammalWeight with given parameter") {
    assert(PatternMatching.testExtractMammalWeight(PatternMatching.Mammal("cat",
PatternMatching.Meat, 5))  == 5)
    assert(PatternMatching.testExtractMammalWeight(PatternMatching.Fish("goldfish",
PatternMatching.Vegetables))  == -1)
  }

  test("updateFood changes animal food") {
    val fish = PatternMatching.Fish("goldfish", PatternMatching.Plants)
    PatternMatching.updateFood(fish)
    assert(fish.food  == PatternMatching.Plants)

    val bird = PatternMatching.Bird("parrot", PatternMatching.Vegetables)
    PatternMatching.updateFood(bird)
    assert(bird.food  == PatternMatching.Plants)
  }

  test("testUpdateFood calls updateFood with given parameter") {
    val fish = PatternMatching.Fish("goldfish", PatternMatching.Vegetables)
    PatternMatching.testUpdateFood(fish)
    assert(fish.food  == PatternMatching.Plants)
  }


}
```

Результаты тестов:

```
sbt:root> project lab1
[info] Set current project to lab1 (in build file:/C:/dev/FP/CourseTest/)
sbt:lab1> test
[info] AnimalTest:
[info] - Animal.apply creates animal if valid parameter is passed
[info] - Animal.apply returns None if wrong parameter is passed
[info] - KnownAnimal returns true if valid animal is passed
[info] - KnownAnimal returns false if invalid animal is passed
[info] - Animal.eats checks if animal eats given food
[info] FunctionsTest:
[info] - CircleArea calculates circle area
[info] - testCircle calls circleArea with given radius
[info] - RectangleArea calculates area of the given rectangle
[info] - testRectangleUc calls RectangleArea with given parameters
[info] - RectangleAreaCurried calculates area of the given rectangle
[info] - testRectangleCurried calls RectangleAreaCurried with given parameters
[info] HigherOrderTest:
[info] - nTimes multiplies function result by n
[info] - testNtimes calls nTimes with given parameters
[info] - testAnonymousNTimes calls nTimes with given parameters and hardcoded function
[info] PatternsTest:
[info] - intToString recognizes numbers from 1 to 3 inclusively
[info] - testIntToString calls intToString with given parameters
[info] - isMaxAndMoritz recognizes (M)max and (M)moritz
[info] - testIsMaxAndMoritz calls isMaxAndMoritz with given parameter
[info] - isEven returns true if number is even and false otherwise
[info] - testIsEven calls isEven with given parameter
[info] - winsA returns rock-paper-scissor game result for first player
[info] - testWinsA calls winsA with given parameters
[info] - extractMammalWeight returns mammal's weight and -1 if not mammal is passed
[info] - testExtractMammalWeight calls extractMammalWeight with given parameter
[info] - updateFood changes animal food
[info] - testUpdateFood calls updateFood with given parameter
[info] Run completed in 1 second, 378 milliseconds.
[info] Total number of tests run: 26
[info] Suites: completed 4, aborted 0
[info] Tests: succeeded 26, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
[success] Total time: 3 s, completed 13.06.2020 21:49:42
sbt:lab1>
```

На данном скриншоте представлен отчет об уровне покрытия тестами исходного кода первой лабораторной работы. Как можно видеть, он составляет 100 процентов.



All packages	100.00%
exercise1	100.00%

SCoverage generated at Sat Jun 13 21:27:16 MSK 2020

Lines of code:	277	Files:	4	Classes:	4	Methods:	25
Lines per file:	69.25	Packages:	1	Classes per package:	4.00	Methods per class:	6.25
Total statements:	60	Invoked statements:	60	Total branches:	2	Invoked branches:	2
Ignored statements:	0						
Statement coverage:	100.00 %			Branch coverage:	100.00 %		

Class	Source file	Lines	Methods	Statements	Invoked	Coverage	Branches	Invoked	Coverage
Animal	Classes.scala	75	4	17	17	100.00 %	0	0	100.00 %
Functions	Functions.scala	39	6	6	6	100.00 %	0	0	100.00 %
HigherOrder	HiOrder.scala	20	3	7	7	100.00 %	2	2	100.00 %
PatternMatching	Patterns.scala	143	12	30	30	100.00 %	0	0	100.00 %

Лабораторная работа 2

Файлы с исходным кодом:

Файл Compositions.scala

```
package exercise2

/** Напишите ваши решения в тестовых функциях.  */
object Compositions {

  // a) Используйте данные функции. Вы можете реализовать свое решение прямо в
тестовой функции.
  // Нельзя менять сигнатуры

  def testCompose[A, B, C, D](f: A => B)
                             (g: B => C)
                             (h: C => D): A => D = h compose g compose f

  // b) Напишите функции с использованием `map` и `flatMap`. Вы можете реализовать
свое решение прямо в тестовой функции.
  // Нельзя менять сигнатуры

  def testMapFlatMap[A, B, C, D](f: A => Option[B])
                                (g: B => Option[C])
                                (h: C => D): Option[A] => Option[D] =
_.flatMap(f).flatMap(g).map(h)
  // c) Напишите функцию используя for. Вы можете реализовать свое решение прямо в
тестовой функции.
  // Нельзя менять сигнатуры

  def testForComprehension[A, B, C, D](f: A => Option[B])
                                      (g: B => Option[C])
                                      (h: C => D): Option[A] => Option[D] = for {
first <- _

second <- f(first)

third <- g(second) } yield h(third)


}
```

Файл RecurciveData.scala

```
package exercise2

import scala.collection.immutable.List

/** Напишите свои решения в виде функций. */
object RecursiveData {

  // a) Реализуйте функцию, определяющую является ли пустым `List[Int]`.
  def ListIntEmpty(list: List[Int]) : Boolean = list match {
    case x :: tail   => false
    case Nil         => true
  }

  // используйте функцию из пункта (a) здесь, не изменяйте сигнатуру
  def testListIntEmpty(list: List[Int]): Boolean = ListIntEmpty(list)

  // b) Реализуйте функцию, которая получает head `List[Int]`или возвращает -1 в
случае если он пустой.

  def ListIntHead(list: List[Int]) : Int = list match {
    case x :: tail   => x
    case Nil         => -1
  }

  // используйте функцию из пункта (a) здесь, не изменяйте сигнатуру
  def testListIntHead(list: List[Int]): Int = ListIntHead(list)

  // c) Можно ли изменить `List[A]` так чтобы гарантировать что он не является
пустым?

  def ListNotEmpty[A](head: A, list: List[A]) : List[A] = list match {
    case Nil          => head :: list
    case x :: tail    => list
  }

  /* d) Реализуйте универсальное дерево (Tree) которое хранит значения в виде листьев
и состоит из:
   *      node - левое и правое дерево (Tree)
   *      leaf - переменная типа A
   */

  class Tree[A](LeftNode: Tree[A], RightNode: Tree[A], leaf: A)

}
```

Файл RecursiveFunc.scala

```
package exercise2



import scala.annotation.tailrec
import scala.collection.immutable.List

/** Реализуйте функции для решения следующих задач.
  * Примечание: Попытайтесь сделать все функции с хвостовой рекурсией, используйте
аннотацию для подствержения.
```

```scala
* рекурсия будет хвостовой если:
  *   1. рекурсия реализуется в одном направлении
  *   2. вызов рекурсивной функции будет последней операцией перед возвратом
*/
object RecursiveFunctions {

  def length[A](as: List[A]): Int = {
    @tailrec
    def loop(rem: List[A], agg: Int): Int = rem match {
      case x :: tail  => loop(tail, agg + 1)
      case Nil         => agg
    }

    loop(as, 0)
  }

  /* a) Напишите функцию которая записывает в обратном порядке список:
   *        def reverse[A](list: List[A]): List[A]
   */

  def reverse[A](list: List[A]): List[A] = {
    @tailrec
    def loop(rem: List[A], result: List[A]): List[A] = rem match {
      case x :: tail  =>  loop(tail, x :: result)
      case Nil         => result
    }
    loop(list, Nil)
  }

  // используйте функцию из пункта (a) здесь, не изменяйте сигнатуру
  def testReverse[A](list: List[A]): List[A] = reverse(list)

  /* b) Напишите функцию, которая применяет функцию к каждому значению списка:
   *        def map[A, B](list: List[A])(f: A => B): List[B]
   */

  def Map[A, B](list: List[A])(f: A => B): List[B] = {
    @tailrec
    def loop(rem: List[A], result: List[B])(f: A => B): List[B] = rem match {
      case x :: tail => loop(tail, result :+ f(x))(f)
      case Nil       => result
    }
    loop(list, Nil)(f)
  }

  // используйте функцию из пункта  (b) здесь, не изменяйте сигнатуру
  def testMap[A, B](list: List[A], f: A => B): List[B] = Map(list)(f)

  /* c) Напишите функцию, которая присоединяет один список к другому:
   *        def append[A](l: List[A], r: List[A]): List[A]
   */

  def Append[A](l: List[A], r: List[A]) : List[A] = {
    @tailrec
    def loop(rem: List[A], result: List[A]) : List[A] = rem match {
      case x :: tail => loop(tail, result :+ x)
      case Nil       => result
    }
    loop(r, l)
  }

  // используйте функцию из пункта  (c) здесь, не изменяйте сигнатуру
```

```
  def testAppend[A](l: List[A], r: List[A]): List[A] = Append(l, r)

  /* d) Напишите функцию, которая применяет функцию к каждому значению списка:
   *        def flatMap[A, B](list: List[A])(f: A => List[B]): List[B]
   *
   *    она получает функцию, которая создает новый List[B] для каждого элемента типа
A в
   *    списке. Поэтому вы создаете List[List[B]].
   */

  def FlatMap[A, B](list: List[A])(f: A => List[B]): List[B] = {
    @tailrec
    def loop(rem: List[A], result: List[B])(f: A => List[B]): List[B] = rem match {
      case x :: tail => loop(tail, result ++ f(x))(f)
      case Nil       => result
    }
    loop(list, Nil)(f)
  }

  // используйте функцию из пункта  (d) здесь, не изменяйте сигнатуру
  def testFlatMap[A, B](list: List[A], f: A => List[B]): List[B] = FlatMap(list)(f)

  /* e) Вопрос: Возможно ли написать функцию с хвостовой рекурсией для `Tree`s? Если
нет, почему? */

      // Нет. Одним из признаков хвостовой рекурсии являтся рекурсия в одном
направлении, что невозможно для древовидной структуры.

}
```

Файл CompositionTest.scala

```
package exercise2


import org.scalatest.FunSuite

class CompositionTest extends FunSuite {

  test("testCompose should compose given functions") {
    assert(Compositions.testCompose((i:Int) => "Compose" * i)((i: String) => i * 2)
    ((i:String) => i.dropRight(3))(2) == "ComposeComposeComposeComp")
  }

  test("testFlatMap should compose given functions") {
    assert(Compositions.testMapFlatMap((i:Int) => if (i > 0) Some(i) else None)
    ((i:Int) => if (i > 10) Some(i) else None)
    ((i:Int) => i * 2)(Some(-1)) == None)
  }

  test("testForComprehension should compose given functions") {
    assert(Compositions.testForComprehension((i:Int) => if (i > 0) Some(i) else None)
    ((i:Int) => if (i > 10) Some(i) else None)
    ((i:Int) => i * 2)(Some(11)) == Some(22))
  }

}
```

Файл RecursiveDataTest.scala

```
package exercise2

import org.scalatest.FunSuite

class RecursiveDataTest extends FunSuite{

  test("ListIntEmpty returns true if list is empty and false otherwise") {
    assert(RecursiveData.ListIntEmpty(Nil) == true)
    assert(RecursiveData.ListIntEmpty(List(1, 2, 3, 4)) == false)
  }

  test("testListIntEmpty returns true if list is empty and false otherwise") {
    assert(RecursiveData.testListIntEmpty(Nil) == true)
    assert(RecursiveData.testListIntEmpty(List(1, 2, 3, 4)) == false)
  }

  test("ListIntHead returns first element of the list and -1 if the list is empty") {
    assert(RecursiveData.ListIntHead(List(1, 2, 3, 4)) == 1)
    assert(RecursiveData.ListIntHead(Nil) == -1)
  }

  test("testListIntHead returns first element of the list and -1 if the list is
empty") {
    assert(RecursiveData.testListIntHead(List(1, 2, 3, 4)) == 1)
    assert(RecursiveData.testListIntHead(Nil) == -1)
  }

  test("ListNotEmpty returns adds element to the list if it's empty") {
    assert(RecursiveData.ListNotEmpty(1, Nil) == List(1))
    assert(RecursiveData.ListNotEmpty(1, List(1, 2, 3 ,4)) == List(1, 2, 3, 4))
  }

}
```

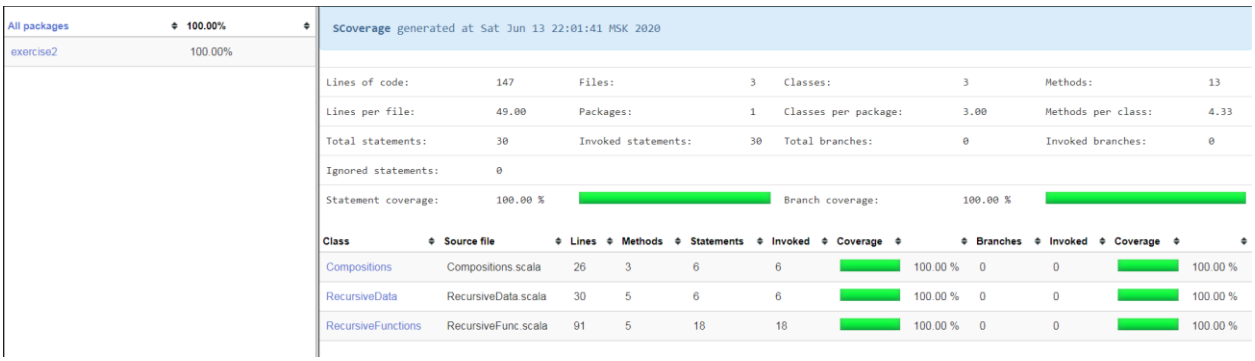
Файл RecurciveFuncTest.scala

```
package exercise2

import org.scalatest.FunSuite
import RecursiveFunctions._

import scala.collection.immutable.List

class RecursiveFuncTest extends FunSuite {

  test("length returns lenght of the passed list") {
    assert(length(List(1, 2, 3, 4)) == 4)
  }

  test("reverse takes a list and returns same but reversed list") {
    assert(reverse(List(1, 2, 3, 4)) == List(4, 3, 2, 1))
  }

  test("testReverse takes a list and returns same but reversed list") {
    assert(testReverse(List(1, 2, 3, 4)) == List(4, 3, 2, 1))
  }
```

```
  test("Map should apply given function to each element of the passed list") {
    assert(Map(List(1, 2, 3, 4))((x: Int) => x * 2) == List(2, 4, 6, 8))
  }

  test("testMap should apply given function to each element of the passed list") {
    assert(testMap(List(1, 2, 3, 4), (x: Int) => x * 2) == List(2, 4, 6, 8))
  }

  test("Append concatenates two lists") {
    assert(Append(List(1, 2, 3), List(4, 5, 6)) == List(1, 2, 3, 4, 5, 6))
  }

  test("testAppend concatenates two lists") {
    assert(testAppend(List(1, 2, 3), List(4, 5, 6)) == List(1, 2, 3, 4, 5, 6))
  }

  test("FlatMap applies given function to each element of the passed list") {
    assert(FlatMap(List(2, 3, 4, 5))((x:Int) => List.range(1, x)) == List(1, 1, 2, 1,
2, 3, 1, 2, 3, 4))
  }

  test("testFlatMap applies function to each element of the passed list") {
    assert(testFlatMap(List(2, 3, 4, 5), (x:Int) => List.range(1, x)) == List(1, 1,
2, 1, 2, 3, 1, 2, 3, 4))
  }
}
```

Результаты:

```
sbt:lab2> test
[info] Compiling 3 Scala sources to C:\dev\FP\CourseTest\lab2\target\scala-2.13\test-classes ...
[info] RecursiveDataTest:
[info] - ListIntEmpty returns true if list is empty and false otherwise
[info] RecursiveFuncTest:
[info] - length returns lenght of the passed list
[info] - testListIntEmpty returns true if list is empty and false otherwise
[info] - reverse takes a list and returns same but reversed list
[info] - ListIntHead returns first element of the list and -1 if the list is empty
[info] - testListIntHead returns first element of the list and -1 if the list is empty
[info] - ListNotEmpty returns adds element to the list if it's empty
[info] - testReverse takes a list and returns same but reversed list
[info] - Map should apply given function to each element of the passed list
[info] - testMap should apply given function to each element of the passed list
[info] - Append concatenates two lists
[info] - testAppend concatenates two lists
[info] - FlatMap applies given function to each element of the passed list
[info] - testFlatMap applies function to each element of the passed list
[info] CompositionTest:
[info] - testCompose should compose given functions
[info] - testFlatMap should compose given functions
[info] - testForComprehension should compose given functions
[info] Run completed in 1 second, 293 milliseconds.
[info] Total number of tests run: 17
[info] Suites: completed 3, aborted 0
[info] Tests: succeeded 17, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
[success] Total time: 10 s, completed 13.06.2020 22:00:46
sbt:lab2>
```



All packages 100.00%

exercise2 100.00%

SCoverage generated at Sat Jun 13 22:01:41 MSK 2020

Lines of code:	147	Files:	3	Classes:	3	Methods:	13
Lines per file:	49.00	Packages:	1	Classes per package:	3.00	Methods per class:	4.33
Total statements:	30	Invoked statements:	30	Total branches:	0	Invoked branches:	0
Ignored statements:	0						
Statement coverage:	100.00 %			Branch coverage:	100.00 %		

Class	Source file	Lines	Methods	Statements	Invoked	Coverage		Branches	Invoked	Coverage	
Compositions	Compositions.scala	26	3	6	6		100.00 %	0	0		100.00 %
RecursiveData	RecursiveData.scala	30	5	6	6		100.00 %	0	0		100.00 %
RecursiveFunctions	RecursiveFunc.scala	91	5	18	18		100.00 %	0	0		100.00 %

Лабораторная работа 3

Файлы с исходным кодом:

Файл Adts.scala

```
package exercise3

import scala.util.Try

object Adts {

  // a) Дан List[Int], верните элемент с индексом n


  // примените функцию из пункта (a) здесь, не изменяйте сигнатуру
  def testGetNth(list: List[Int], n: Int): Option[Int] = Some(list(n))

  // b) Напишите функцию, увеличивающую число в два раза.

  def Double(n: Option[Int]): Option[Int] = if (n.isDefined)  Some(n.get * 2)  else
None

  // примените функцию из пункта (b) здесь, не изменяйте сигнатуру
  def testDouble(n: Option[Int]): Option[Int] = Double(n)

  // c) Напишите функцию, проверяющую является ли число типа Int четным. Если так,
верните Right. В противном случае, верните Left("Нечетное число.").

  def IsEven(n: Int): Either[String, Int] = n % 2 match {
    case 0 => Right(n)
    case 1 => Left("Odd number")
  }

  // примените функцию из пункта (c) здесь, не изменяйте сигнатуру
  def testIsEven(n: Int): Either[String, Int] = IsEven(n)

  // d) Напишите функцию, реализующую безопасное деление целых чисел. Верните Right с
результатом или Left("Вы не можете делить на ноль.").

  def SafeDivide(a: Int, b: Int): Either[String, Int] = {
    if (b == 0) Left("Cannot divide by zero")
    else Right(a / b)
  }

  // примените функцию из пункта (d) здесь, не изменяйте сигнатуру
  def testSafeDivide(a: Int, b: Int): Either[String, Int] = SafeDivide(a, b)
```

```
  // e) Обработайте исключения функции с побочным эффектом вернув 0.

  def GoodOldJava(impure: String => Int, str: String): Try[Int] = Try(impure(str))

  // примените функцию из пункта (e) здесь, не изменяйте сигнатуру
  def testGoodOldJava(impure: String => Int, str: String): Try[Int] =
GoodOldJava(impure, str)

}
```

Файл Maps.scala

```
package exercise3

/** Напишите вашу реализацию в тестовые функции.
  *
  * https://docs.scala-lang.org/overviews/collections/maps.html
  */
object Maps {

  case class User(name: String, age: Int)

  /* a) В данной Seq[User] сгруппируйте пользователей по имени (`groupBy`) и
вычислите средний возраст: `name -> averageAge`
   *    Вы можете реализовать ваше решение в теле тестовой функции. Не изменяйте
сигнатуру.
   */
  def testGroupUsers(users: Seq[User]): Map[String, Int] = {
    var groups = users.groupBy(_.name)
    groups.map(x => (x._1, x._2.foldLeft(0)(_ + _.age) / x._2.length))
  }

  /* b) Дана `Map[String, User]` состоящая из имен пользователей `User`, сколько имен
пользователей, содержащихся в Map, содержат подстроку "Adam"?
   *    Вы можете реализовать ваше решение в теле тестовой функции. Не изменяйте
сигнатуру.
   */
  def testNumberFrodos(map: Map[String, User]): Int = {
    var count = 0
    map.keys.foreach { key =>
      if (map(key).name.contains("Adam")) count += 1
    }
    count
  }

  /* c) Удалите всех пользователей возраст которых менее 35 лет.
   *    Вы можете реализовать ваше решение в теле тестовой функции. Не изменяйте
сигнатуру.
   */
  def testUnderaged(map: Map[String, User]): Map[String, User] = {
    var result = map
    result.keys.foreach { key =>
      if (result(key).age < 35) result = result.-(key)
    }
    result
  }

}
```

Файл Sequence.scala

```scala
package exercise3

/** Напишите свои решения в тестовых функциях.
  *
  * Seq(1, 2) match {
  *   case head +: tail => ???
  *   case Nil          => ???
  *   case s            => ???
  * }
  *
  * https://www.scala-lang.org/api/2.12.0/scala/collection/Seq.html
  */
// Примечание: напишите функции с хвостовой рекурсией

object Sequence {

  /* a) Найдите последний элемент Seq.
   *
   */
  def testLastElement[A](seq: Seq[A]): Option[A] = Some(seq.last)

  /* b) Объедините две Seqs (то есть Seq(1, 2) и Seq(3, 4) образуют Seq((1, 3), (2,
4))) - если Seq длиннее игнорируйте оставшиеся элементы.
   *
   */
  def testZip[A](a: Seq[A], b: Seq[A]): Seq[(A, A)] = a.zip(b)

  /* c) Проверьте, выполняется ли условие для всех элементов в Seq.
   *
   */
  def testForAll[A](seq: Seq[A])(cond: A => Boolean): Boolean = seq.forall(cond)

  /* d) Проверьте, является ли Seq палиндромом
   *
   */
  def testPalindrome[A](seq: Seq[A]): Boolean = seq.reverse == seq

  /* e) Реализуйте flatMap используя foldLeft.
   *
   */
  def testFlatMap[A, B](seq: Seq[A])(f: A => Seq[B]): Seq[B] =
seq.foldLeft(Seq[B]())(_ ++ f(_))
}
```

Файл Strings.scala

```scala
package exercise3

/** Напишите ваши решения в тестовых функциях.
  *
  * https://www.scala-lang.org/api/2.12.3/scala/collection/immutable/StringOps.html
  */
object Strings {

  /* a) Преобразуйте все символы типа Char в верхний регистр (не используйте
заглавные буквы).
   *
```

```
   */
  def testUppercase(str: String): String = str.toUpperCase

  /* b) Вставьте следующие значения в строку:
   *       Hi my name is <name> and I am <age> years old.
   *
   */
  def testInterpolations(name: String, age: Int): String = s"Hi, my name is $name and
I am $age years old."

  /* c) Добавьте два числа в следующую строку:
   *       Hi,
   *       now follows a quite hard calculation. We try to add:
   *         a := <value of a>
   *         b := <value of b>
   *
   *         result is <a + b>
   *
   *
   */
  def testComputation(a: Int, b: Int): String = "Hi,\n" +
    "now follows a quite hard calculation. We try to add:\n" +
    s"  a := $a\n" +
    s"  b := $b\n\n" +
    s"  return $a + $b"

  /* d) Если длина строки равна 2, верните всю строку, иначе верните первые два
символа строки.
   */
  def testTakeTwo(str: String): String = str.length match {
    case 2 => str
    case _ => str.substring(0,2)
  }
}
```

Файлы с тестами:

Файл AdtsTest.scala

```
package exercise3

import org.scalatest.FunSuite
import Adts._


class AdtsTest extends FunSuite{

  test("testGetNth should return n-th element of the list") {
    assert(testGetNth(List(1, 2, 3, 4), 3) == Some(4))
  }

  test("Double should return doubled number and None if None is passed") {
    assert(Double(Some(3)) == Some(6))
    assert(Double(None) == None)
  }

  test("testDouble should return double number and None if None is passed") {
```

```
    assert(testDouble(Some(3)) == Some(6))
    assert(testDouble(None) == None)
  }

  test("IsEven should return passed number if it's even and string \"Odd number\"
otherwise") {
    assert(IsEven(4) == Right(4))
    assert(IsEven(5) == Left("Odd number"))
  }

  test("testIsEven should return passed number if it's even and string \"Odd number\"
otherwise") {
    assert(testIsEven(4) == Right(4))
    assert(testIsEven(5) == Left("Odd number"))
  }

  test("SafeDivide should divide two numbers unless divisor is 0") {
    assert(SafeDivide(10, 2) == Right(5))
    assert(SafeDivide(10, 0) == Left("Cannot divide by zero"))
  }

  test("testSafeDivide should divide two numbers unless divisor is 0") {
    assert(testSafeDivide(10, 2) == Right(5))
    assert(testSafeDivide(10, 0) == Left("Cannot divide by zero"))
  }

  def impureFunc(str: String): Int = {
    2/0
  }

  def pureFunc(str: String): Int = {
    2
  }

  test("GoodOldJava should return Try[Int]") {
    assert(GoodOldJava(impureFunc, "Hello, world!") != util.Success(2))
  }

  test("testGoodOldJava should return Try[Int]") {
    assert(testGoodOldJava(pureFunc, "Hello, world!") == util.Success(2))
  }

}
```

Файл MapsTest.scala

```
package exercise3

import org.scalatest.FunSuite
import Maps._

class MapsTest extends FunSuite {
  val Frodo = User("Frodo", 15)
  val Legolas1 = User("Legolas", 1000)
  val Legolas2 = User("Legolas", 500)

  val map = Map("Frodo" -> Frodo, "Legolas" -> Legolas1, "Legolas" -> Legolas2)
  val seq = Seq(Frodo, Legolas1, Legolas2)

  test("testGroupUsers should group users by name and calculate average age of each
```

```
group") {
    assert(testGroupUsers(seq) == Map("Frodo" -> 15, "Legolas" -> 750))
  }

  test("testNumberFrodos counts \"Adam\" count in passed map[String->User]") {
    assert(testNumberFrodos(Map("Frodo" -> Frodo, "Adam" -> User("Adam", 40))) == 1)
  }

  test("testUnderaged removes from the map all users under 35 years old") {
    assert(testUnderaged(map) == Map("Legolas" -> Legolas1, "Legolas" -> Legolas2))
  }

}
```

Файл SequenceTest.scala

```
package exercise3

import org.scalatest.FunSuite
import Sequence._

class SequenceTest extends FunSuite {

  test("testLastElement returns last element of the sequence") {
    assert(testLastElement(Seq(1, 2, 3, 4)) == Some(4))
  }

  test("testZip zips combines two sequences") {
    assert(testZip(Seq(1, 2), Seq(3, 4)) == Seq((1, 3), (2, 4)))
  }

  test("testForAll checks codition for all elements") {
    assert(testForAll(Seq(1, 2, 3, 4))((x: Int) => x < 5) == true)
  }

  test("testPalindrom returns true if sequence is palindrom and false otherwise") {
    assert(testPalindrome(Seq(1, 2, 3, 4)) == false)
    assert(testPalindrome(Seq(1, 2, 2, 1)) == true)
  }

  test("testFlatMap should apply function to each element of the sequence") {
    assert(testFlatMap(Seq(1, 2, 3, 4))((x: Int) => Seq.range(1, x)) == Seq(1, 1, 2,
1, 2, 3))
  }
}
```

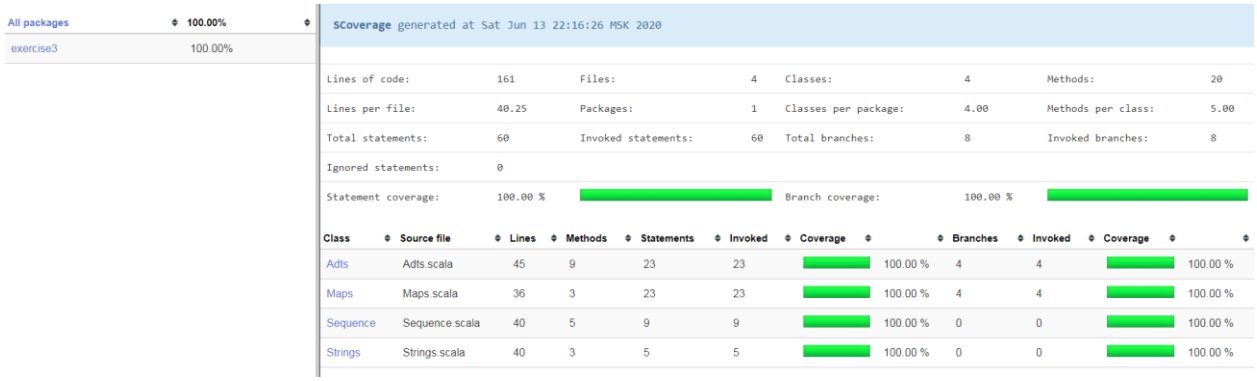
Файл StringsTest.scala

```
package exercise3

import org.scalatest.FunSuite
import Strings._

class StringsTest extends FunSuite {

  test("testUppercase returns string with all capital letters") {
    assert(testUppercase("testString") == "TESTSTRING")
  }
```

```
  test("testInterpolation inserts given name and age into string") {
    assert(testInterpolations("Lev", 19) == "Hi, my name is Lev and I am 19 years
old.")
  }

  test("testComputation inserts numbers into string") {
    assert(testComputation(4, 5) == "Hi,\n" +
      "now follows a quite hard calculation. We try to add:\n" +
      "  a := 4\n" +
      "  b := 5\n\n" +
      "  return 4 + 5")
  }

  test("testTakeTwo takes first two characters of the string") {
    assert(testTakeTwo("Substring") ==  "Su")
  }
}
```

Результаты:

```
sbt:lab3> test
[info] Compiling 4 Scala sources to C:\dev\FP\CourseTest\lab3\target\scala-2.13\test-classes ...
[info] SequenceTest:
[info] - testLastElement returns last element of the sequence
[info] - testZip zips combines two sequences
[info] - testForAll checks codition for all elements
[info] - testPalindrom returns true if sequence is palindrom and false otherwise
[info] - testFlatMap should apply function to each element of the sequence
[info] StringsTest:
[info] - testUppercase returns string with all capital letters
[info] - testInterpolation inserts given name and age into string
[info] - testComputation inserts numbers into string
[info] - testTakeTwo takes first two characters of the string
[info] MapsTest:
[info] - testGroupUsers should group users by name and calculate average age of each group
[info] - testNumberFrodos counts "Adam" count in passed map[String->User]
[info] - testUnderaged removes from the map all users under 35 years old
[info] AdtsTest:
[info] - testGetNth should return n-th element of the list
[info] - Double should return doubled number and None if None is passed
[info] - testDouble should return double number and None if None is passed
[info] - IsEven should return passed number if it's even and string "Odd number" otherwise
[info] - testIsEven should return passed number if it's even and string "Odd number" otherwise
[info] - SafeDivide should divide two numbers unless divisor is 0
[info] - testSafeDivide should divide two numbers unless divisor is 0
[info] - GoodOldJava should return Try[Int]
[info] - testGoodOldJava should return Try[Int]
[info] Run completed in 1 second, 408 milliseconds.
[info] Total number of tests run: 21
[info] Suites: completed 4, aborted 0
[info] Tests: succeeded 21, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
[success] Total time: 9 s, completed 13.06.2020 22:16:20
```



All packages	100.00%
exercise3	100.00%

SCoverage generated at Sat Jun 13 22:16:26 MSK 2020

Lines of code:	161	Files:	4	Classes:	4	Methods:	20
Lines per file:	40.25	Packages:	1	Classes per package:	4.00	Methods per class:	5.00
Total statements:	60	Invoked statements:	60	Total branches:	8	Invoked branches:	8
Ignored statements:	0						
Statement coverage:	100.00 %			Branch coverage:	100.00 %		

Class	Source file	Lines	Methods	Statements	Invoked	Coverage		Branches	Invoked	Coverage	
Adts	Adts.scala	45	9	23	23		100.00 %	4	4		100.00 %
Maps	Maps.scala	36	3	23	23		100.00 %	4	4		100.00 %
Sequence	Sequence.scala	40	5	9	9		100.00 %	0	0		100.00 %
Strings	Strings.scala	40	3	5	5		100.00 %	0	0		100.00 %

Лабораторная работа 4

Файлы с исходным кодом:

Файл TypeClasses.scala

```scala
package exercise4


  object TypeClasses {

    // a) Определите тайп-класс Reversable, который представляет в обратном порядке значения.


    trait Reversable[T] {
      def reverse(x: T): T
    }

    // b) Реализуйте функцию Reverse для String.

    object Reversable {
      implicit object ReversableString extends Reversable[String] {
        def reverse(str: String) : String = str.reverse
      }
    }


    def reverse[T](str: T)(implicit rev: Reversable[T]): T = rev.reverse(str)
    // примените тайп-класс-решение из пункта (a) здесь
    def testReversableString(str: String): String = reverse(str)



    // c) Определите тайп-класс Smash таким образом чтобы в нем была функция smash, которая выполняет операцию со значениями одного типа.

    trait Smash[T] {
      def smash(a: T, b: T): T
    }

    object Smash {
      implicit object SmashInt extends Smash[Int] {
        def smash(a: Int, b: Int): Int = a + b
      }

      implicit object SmashDouble extends Smash[Double] {
```

```
        def smash(a: Double, b: Double): Double = a * b
      }

      implicit object SmashString extends Smash[String] {
        def smash(a: String, b:String): String = a.concat(b)
      }
    }

    // d) Реализуйте  функции Smash для типа Int и Double.
    //    Используйте сложение для типа Int у умножение для типа Double.

    def smash[T](a: T, b: T)(implicit sm : Smash[T]) : T = sm.smash(a, b)


    // примените тайп-класс-решение из пункта (d) здесь
    def testSmashInt(a: Int, b: Int): Int = smash(a, b)

    // примените тайп-класс-решение из пункта (d) здесь
    def testSmashDouble(a: Double, b: Double): Double = smash(a, b)


    // e) Реализуйте функцию Smash для типа String. Необходимо выполнить конкатенацию
строк, которые будут получены в качестве параметра.



    // примените тайп-класс-решение из пункта (d) здесь
    def testSmashString(a: String, b: String): String = smash(a, b)
  }
```

Файлы с тестами:

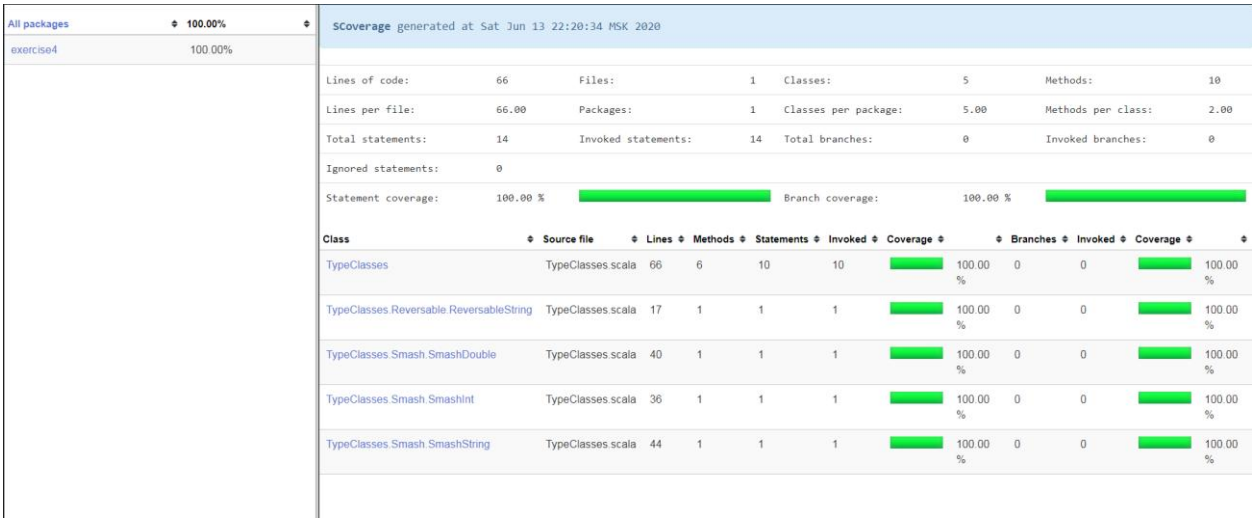
Файл TypeClassesTest.scala

```
package exercise4

import org.scalatest.FunSuite
import TypeClasses._

class TypeClassesTest extends FunSuite {

  test("testReversableString should return reversed string") {
    assert(testReversableString("reverse") == "esrever")
  }

  test("smash should add integer numbers and multiply double numbers") {
    assert(smash(2, 3) == 5)
    assert(smash(2.5, 3.0) == 7.5)
    assert(smash("Concat", "String") == "ConcatString")
  }

  test("testSmashInt should add integer numbers") {
    assert(testSmashInt(2, 3) == 5)
  }

  test("testSmashDouble should multiply double numbers") {
    assert(testSmashDouble(2.5, 3.0) == 7.5)
  }

  test("testSmashString should concatenate strings") {
```

```
    assert(testSmashString("Concat", "String") == "ConcatString")
  }



}
```

Результаты:

```
sbt:lab4> test
[info] Compiling 1 Scala source to C:\dev\FP\CourseTest\lab4\target\scala-2.13\test-classes ...
[info] TypeClassesTest:
[info] - testReversableString should return reversed string
[info] - smash should add integer numbers and multiply double numbers
[info] - testSmashInt should add integer numbers
[info] - testSmashDouble should multiply double numbers
[info] - testSmashString should concatenate strings
[info] Run completed in 836 milliseconds.
[info] Total number of tests run: 5
[info] Suites: completed 1, aborted 0
[info] Tests: succeeded 5, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
[success] Total time: 5 s, completed 13.06.2020 22:20:21
sbt:lab4> coverage
```



All packages	100.00%
exercise4	100.00%

SCoverage generated at Sat Jun 13 22:20:34 MSK 2020

Lines of code:	66	Files:	1	Classes:	5	Methods:	10				
Lines per file:	66.00	Packages:	1	Classes per package:	5.00	Methods per class:	2.00				
Total statements:	14	Invoked statements:	14	Total branches:	0	Invoked branches:	0				
Ignored statements:	0										
Statement coverage:	100.00 %			Branch coverage:	100.00 %						

Class	Source file	Lines	Methods	Statements	Invoked	Coverage		Branches	Invoked	Coverage	
TypeClasses	TypeClasses.scala	66	6	10	10		100.00 %	0	0		100.00 %
TypeClasses.Reversable.ReversableString	TypeClasses.scala	17	1	1	1		100.00 %	0	0		100.00 %
TypeClasses.Smash.SmashDouble	TypeClasses.scala	40	1	1	1		100.00 %	0	0		100.00 %
TypeClasses.Smash.SmashInt	TypeClasses.scala	36	1	1	1		100.00 %	0	0		100.00 %
TypeClasses.Smash.SmashString	TypeClasses.scala	44	1	1	1		100.00 %	0	0		100.00 %

Тестирование всего проекта:

```
[info] Run completed in 317 milliseconds.
[info] Total number of tests run: 0
[info] Suites: completed 0, aborted 0
[info] Tests: succeeded 0, failed 0, canceled 0, ignored 0, pending 0
[info] No tests were executed.
[info] StringsTest:
[info] - testUppercase returns string with all capital letters
[info] - testInterpolation inserts given name and age into string
[info] SequenceTest:
[info] - testComputation inserts numbers into string
[info] MapsTest:
[info] - testTakeTwo takes first two characters of the string
[info] AdtsTest:
[info] - testLastElement returns last element of the sequence
[info] - testZip zips combines two sequences
[info] - testForAll checks codition for all elements
[info] - testPalindrom returns true if sequence is palindrom and false otherwise
[info] - testFlatMap should apply function to each element of the sequence
[info] - testGetNth should return n-th element of the list
[info] - Double should return doubled number and None if None is passed
[info] - testDouble should return double number and None if None is passed
[info] - IsEven should return passed number if it's even and string "Odd number" otherwise
[info] - testIsEven should return passed number if it's even and string "Odd number" otherwise
[info] - SafeDivide should divide two numbers unless divisor is 0
[info] - testSafeDivide should divide two numbers unless divisor is 0
[info] - testGroupUsers should group users by name and calculate average age of each group
[info] - GoodOldJava should return Try[Int]
[info] - testNumberFrodos counts "Adam" count in passed map[String->User]
[info] - testGoodOldJava should return Try[Int]
[info] - testUnderaged removes from the map all users under 35 years old
[info] TypeClassesTest:
[info] - testReversableString should return reversed string
[info] - smash should add integer numbers and multiply double numbers
[info] - testSmashInt should add integer numbers
[info] - testSmashDouble should multiply double numbers
[info] - testSmashString should concatenate strings
[info] Run completed in 6 seconds, 393 milliseconds.
[info] Total number of tests run: 21
[info] Suites: completed 4, aborted 0
[info] Tests: succeeded 21, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
[info] Run completed in 5 seconds, 693 milliseconds.
[info] Total number of tests run: 5
[info] Suites: completed 1, aborted 0
[info] Tests: succeeded 5, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
```

```
[info] RecursiveDataTest:
[info] - ListIntEmpty returns true if list is empty and false otherwise
[info] - testListIntEmpty returns true if list is empty and false otherwise
[info] - ListIntHead returns first element of the list and -1 if the list is empty
[info] - testListIntHead returns first element of the list and -1 if the list is empty
[info] - ListNotEmpty returns adds element to the list if it's empty
[info] CompositionTest:
[info] - testCompose should compose given functions
[info] - testFlatMap should compose given functions
[info] - testForComprehension should compose given functions
[info] RecursiveFuncTest:
[info] - length returns lenght of the passed list
[info] - reverse takes a list and returns same but reversed list
[info] - testReverse takes a list and returns same but reversed list
[info] - Map should apply given function to each element of the passed list
[info] - testMap should apply given function to each element of the passed list
[info] - Append concatenates two lists
[info] - testAppend concatenates two lists
[info] - FlatMap applies given function to each element of the passed list
[info] - testFlatMap applies function to each element of the passed list
[info] FunctionsTest:
[info] - CircleArea calculates circle area
[info] Run completed in 7 seconds, 206 milliseconds.
[info] Total number of tests run: 17
[info] Suites: completed 3, aborted 0
[info] Tests: succeeded 17, failed 0, canceled 0, ignored 0, pending 0
```

Вывод

Сделанные ранее лабораторные работы были полностью протестированы с помощью библиотеки scalatest и плагина для sbt scoverage. Данные средства сильно упрощают написание кода на языке программирования scala.