

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 лабораторной работе
по предмету «Функциональное программирование»
на тему:
«Язык программирования «Scala»»

Выполнил: студент группы

БВТ1802

Самаков Владислав Владимирович

Руководитель:

Мосева Марина Сергеевна

Москва 2020

Задания к работе указаны в файлах с кодом.

Выполнение:

Файл Classes.scala

```
/* This task has no tests. It is an exercise for you to write different class
structures.
*
* а) Создать класс Animal, который имеет следующие поля:
*   - name: String (название)
*   - species: String (вид)
*   - food: String
*
*   Синтаксис: class MyClass(val publicField: Int, privateField: String) {
*       // остальные поля и методы
*   }
*
* б) Создайте объект-компаньон для класса Animal и добавьте следующие сущности как
поля:
*   - cat, mammal, meat
*   - parrot, bird, vegetables
*   - goldfish, fish, plants
*
*   Синтаксис: object MyClass {
*       // статические поля и методы
*   }
*
* в) Добавьте следующие метод в Animals:
*   def eats(food: String): Boolean
*
*   который проверяет ест ли животное определенную пищу
*
* г) Переопределите ваш класс Animal как трейт и создайте объекты класса-образца для
Mammals, Birds и Fishs.
*   Вам все еще нужно поле `species`?
*
* д) Добавьте следующие функции в объект-компаньон Animal:
*   def knownAnimal(name: String): Boolean // true если это имя одного из трех
животных из (б)
*   def apply(name: String): Option[Animal] // возвращает одно из трех животных в
соответствии с именем (Some) или ничего (None), см. ниже
*
* е) Создайте трейт Food со следующими классами-образцами:
*   - Meat
*   - Vegetables
*   - Plants
*   и добавьте это в определение Animal. Так же добавьте объект-компаньон с методом
apply():
*   def apply(food: String): Option[Food]
*/

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

```

/** Напишите отдельные функции, решающие поставленную задачу.
 *
 * Синтаксис:
 *   // метод
 *   def myFunction(param0: Int, param1: String): Double = // тело
 *
 *   // значение
 *   val myFunction: (Int, String) => Double (param0, param1) => // тело
 */
object Functions {

  /** a) Напишите функцию, которая рассчитывает площадь окружности
   *     $r^2 * \text{Math.PI}$ 
   */

  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 RectangleAreaCurried(a: Double)(b: Double) = a * b

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

```

```
// с) Напишите не карированную функцию для расчета площади прямоугольника.

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

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

HiOrder.scala

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

  /* a) Напишите функцию, которая принимает `f: (Int, Int) => Int`, параметры `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)
}
}
```

Patterns.scala

```
/** 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 следующим 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 Birds на Plants, класс Mammals
оставьте неизменным.

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)
}

```