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 № 1

по дисциплине «Функциональное программирование»

на тему:

«**Введение в Scala**»

Выполнила: студентка группы БВТ1802

Лаврухина Елена Павловна

Руководитель:

Мосева Марина Сергеевна

Москва 2020

Выполнение

Код программы

1. Classes

```
package exercise1
/*This task has no tests. It is an exercise for you to write different class
structures.
   a) 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 } */
    class Animal (name: String, food: String) {
      def eats (food: String): Boolean = {
        if (name == "cat" && food == "meat")
          true
        else if (name == "parrot" && food == "vegetables")
          true
        else if (name == "goldfish" && food == "plants")
          true
        else false
      }
    }
/*b) 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 } */
     object Animal {
       var mammal = ("cat", "meat")
       var bird = ("parrot", "vegetables")
       var fish = ("goldfish", "plants")
        def knownAnimal(name: String): Boolean = {
          if (mammal == "cat" || bird == "parrot" || fish == "goldfish")
            true
          else false
        }
     }
/*c) Добавьте следующие метод в Animals: def eats(food: String): Boolean,
     который проверяет ест ли животное определенную пищу*/
/*d) Переопределите ваш класс Animal как трейт и создайте объекты класса-образца для
Mammals, Birds и Fishs.
     Вам все еще нужно поле `species`? */
     trait Animal {
       case object Mammal extends Animal
       case object Fish extends Animal
       case object Bird extends Animal
        def knownAnimal(name: String): Boolean = {
          if ((Mammal == "cat") || (Bird == "parrot") || (Fish == "goldfish"))
            true
          else false
        }
        def apply(food: String): Option[Food] =
          return Some.apply(Animal.this)
     }
/*e) Добавьте следующие функции в объект-компаньон Animal: def knownAnimal(name:
String): Boolean
     // true если это имя одного из трех животных из (b)
```

```
     def apply(name: String): Option[Animal]
     // возвращает одно из трех животных в соответствии с именем (Some) или ничего
(None), см. ниже */
/*f) 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] */
      trait Food {
        case object Meat extends Food
        case object Vegetables extends Food
        case object Plants extends Food
          def apply(food: String): Option[Food] =
            return Some.apply(Food.this)
      }
      sealed trait Option[A] {
        def isEmpty: Boolean
      }
      case class Some[A](a: A) extends Option[A] {
        val isEmpty = false
      }
      case class None[A]() extends Option[A] {
        val isEmpty = true
      }
```

2. Patterns

```
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
        val food: Food
      }
      case class Mammal(name: String, food: Food, weight: Int) extends Animal
      case class Fish(name: String, food: Food) extends Animal
      case class Bird(name: String, 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" */
     val IntToString = (a: Int) => {
     var str = ""
       if (a == 1) {
         str = "it is one"
         str
       }
       else if (a == 2) {
         str = "it is two"
         str
       }
       else if (a == 3) {
         str = "it is three"
         str
       }
       else {
         str = "what is that"
         str
       }
     }
  // 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" */
     val IsMaxAndMoritz = (a: String) => {
       if (a == "max" || a == "Max" || a == "moritz" || a == "Moritz") {
         true
       }
       else 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
     val IsEven = (a: Int) => {
       if (a % 2 == 0)
         true
       else 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`? */
     val KNB = (a: Hand, b: Hand) => {
       a match {
         case Rock => {
           b match {
             case Rock => Draw
             case Paper => Lose
             case Scissor => Lose
           }
         }
         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 = KNB(a, b)
  // Примечание: используйте определение Animals
  //e) Верните вес (weight: Int) объекта Mammal, иначе верните -1.
      def Weight(animal: Animal): Int ={
        animal match {
          case Mammal(name, food, weight)=> weight
          case _ => -1
        }
  }
  // примените функцию из пункта (e) здесь, не изменяйте сигнатуру
      def testExtractMammalWeight(animal: Animal): Int = Weight(animal)
  //f) Измените поле еда объектов классов Fishes и Birds на Plants, класс Mammals
оставьте неизмененным.
      def UpdateFood(animal: Animal): Animal = {
        var a = animal
        animal match {
          case Fish(name, food) => a = Fish(name, Plants)
          case Bird(name, food) => a = Bird(name, Plants)
          case Mammal(name, food, weight) => a = Mammal(name, food, weight)
        }
        a
      }
  // примените функцию из пункта (f) здесь, не изменяйте сигнатуру
      def testUpdateFood(animal: Animal): Animal = UpdateFood(animal)
}
```

3. Functions

```
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 */
  val S = (r: Double) => r*r*Math.PI
  // примените вашу функцию из пункта (a) здесь, не изменяя сигнатуру
  def testCircle(r: Double): Double = S(r)
/*b) Напишите не каррированную функцию, которая рассчитывает площадь прямоугольника a *
b.*/
  def RectangleCurried(a: Double)(b: Double) = a*b
  // примените вашу функцию из пункта (b) здесь, не изменяя сигнатуру
  def testRectangleCurried(a: Double, b: Double): Double = RectangleCurried(a)(b)
  // c) Напишите не каррированную функцию для расчета площади прямоугольника.
  val Rectangle = (a1: Double, b1: Double) => a1*b1
  // примените вашу функцию из пункта (c) здесь, не изменяя сигнатуру
  def testRectangleUc(a: Double, b: Double): Double = Rectangle(a, b)
}
```

4. HiOrder

```
package exercise1
/* Напишите ваши решения в виде функций. */
    object HigherOrder {
      val plus: (Int, Int) => Int = _ + _
```

```
      val multiply: (Int, Int) => Int = _ * _
/*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 */
        (a:Int, b:Int) => {
          if (a > b)
            a
          else
            b
        }
      def testAnonymousNTimes(a: Int, b: Int, n: Int): Int = {
        (a, b)
      }
    }
```