**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 №3(5)**

по предмету «Функциональное программирование»

Выполнил: студент группы

Митрохин Ярослав Игоревич

Руководитель:

Мосева Марина Сергеевна

Москва 2020

*Цель работы:* реализовать представленные классы, учитывая заданный синтаксис.

*Ход работы:*

*Реализованный класс Maps(код программы):*

object Maps {  
  
 case class User(name: String,age: Int)  
  
 def main(args: Array[String]): Unit = {  
 var userer: Map[String,User] = *Map*("1"->*User*("Kir",34),"2"->*User*("Yir",35),"1"->*User*("Mir",36))  
 *print*(*testNumberFrodos*(userer))  
 }  
 def testGroupUsers(users: Seq[User]): Map[String,Int] = {  
 var m: Map[String,Int] = *Map*()  
 var nam = users.groupBy(\_.name)  
 for (e <- nam){  
 var aver = e.\_2.toBuffer.foldLeft[Int](0)((a,next)=>a+next.age)/e.\_2.toBuffer.size  
 m+=(e.\_1->aver)  
 }  
 m  
 }  
  
 def testNumberFrodos(map: Map[String, User]): Int = {  
 var c: Int =0;  
 for (num <- map) if (num.\_2.name.indexOf("Adam")!=(-1)) c+=1;  
 c  
 }  
  
 def testUnderaged(map: Map[String,User]): Map[String,User] = map.filter(k=>k.\_2.age>=35)  
}

*Реализовать класс Adts(код программы):*

import scala.util.{Try,Failure,Success}  
object Adts{  
  
 def GetNth(list: List[Int],n: Int): Option[Int] = list match {  
 case head :: tail => *Option*(list(n))  
 case *Nil* => null  
 case l => *Option*(l(n))  
 }  
  
 def testGetNth(list: List[Int],n:Int):Option[Int]= *GetNth*(list,n)  
  
 def DoubleD(n: Option[Int]): Option[Int] = n match {  
 case Some(a) => *Option*(a\*2)  
 case None =>null  
 }  
  
 def testDouble(n: Option[Int]): Option[Int]= *DoubleD*(n)  
  
 def IsEven(n: Int): Either[String,Int] = *Either*.*cond*(n%2==0,n,"Нечетное число") match{  
 case Left(i) => *Left*(i)  
 case Right(s)=> *Right*(s)  
 }  
  
 def testIsEven(n: Int): Either[String,Int] = *IsEven*(n)  
  
 def SafeJava(a: Int, b:Int): Either[String,Int] = *Try*(a/b) match{  
 case Success(a) => *Right*(a)  
 case Failure(error) =>*Left*("Вы не можете делить на ноль.")  
 }  
  
 def testSafeDivide(a:Int, b:Int): Either[String,Int] = *SafeJava*(a,b)  
  
 def GoodOldJava(impure: String => Int, str: String): Try[Int] = *Try*(impure(str)).toEither match {  
 case Right(k)=> Success(k)  
 case Left(l) => Success(0)  
 }  
  
 def testGoodOldJava(impure: String => Int, str: String): Try[Int] = *GoodOldJava*(impure,str)  
  
 def main(args: Array[String]): Unit = {  
 *println*(*testSafeDivide*(5,2))  
 }  
}

*Реализовать класс Sequence(код программы):*

import scala.annotation.tailrec  
  
object Sequence{  
  
 def LastElement[A](seq: Seq[A]): Option[A] = seq match {  
 case head*+:*tail =>*LastElement*(tail)  
 case last*+:Nil*=> *Option*(last)  
 }  
  
 def testLastElement[A](seq: Seq[A]): Option[A] = *LastElement*(seq)  
  
 def zipped[A](a: Seq[A],b: Seq[A]): Seq[(A,A)] = {  
 def loop[A](a: Seq[A], b: Seq[A], c: Seq[(A,A)]): Seq[(A, A)] = a match {  
 case ahead *+:* atail => b match {  
 case blast *+: Nil* => c :+ (ahead,blast)  
 case bhead *+:* btail =>loop(atail,btail,c:+(ahead,bhead))  
 }  
 case *Nil* => c  
 }  
 loop(a,b,*Nil*)  
 }  
  
 def testZip[A](a: Seq[A],b: Seq[A]): Seq[(A,A)] ={*zipped*(a,b)}  
  
 def ForAll[A](seq: Seq[A])(cond: A=>Boolean): Boolean = {  
 def loop[A](a: Seq[A])(cond: A=> Boolean): Boolean = a match{  
 case head *+:* tail => if (cond(head)) loop(tail)(cond) else false  
 case *Nil* => true  
 }  
 loop(seq)(cond)  
 }  
  
 def testForAll[A](seq: Seq[A])(cond: A=>Boolean): Boolean = *ForAll*(seq)(cond)  
  
 def Palindrom[A](seq: Seq[A]): Boolean = {  
 def loop[A](a:Seq[A],b:Seq[A]): Boolean = {  
 a match {  
 case head*+:*tail => loop(tail,b=head+:b)  
 case *Nil* => seq.equals(b)  
 }  
 }  
 loop(seq,*Nil*)  
 }  
  
 def testPalindrom[A](seq: Seq[A]): Boolean =*Palindrom*(seq)  
  
 def FlatMap[A,B](seq: Seq[A])(f: A=>Seq[B]): Seq[B] = {  
 seq.foldLeft[Seq[B]](*Nil*)((a,next)=>f(next).++:(a))  
 }  
  
 def testFlatMap[A,B](seq: Seq[A])(f: A=>Seq[B]): Seq[B] = *FlatMap*(seq)(f)  
  
 def main(args: Array[String]): Unit = {  
 *print*(*FlatMap*(*Seq*(*Seq*(1,2),*Seq*(3,4))))  
 }  
}

*Реализовать класс String(код программы):*

object Strings {  
  
 def main(args: Array[String]): Unit = {  
 }  
  
 def Uppercase(str: String): String = str.toUpperCase  
  
 def testUppercase(str: String): String = *Uppercase*(str)  
  
 def Interpolations(name: String,age: Int): String = {  
 return "Hi my name is %s and I am %d years old.".format(name,age)  
 }  
  
 def testInterpolations(name: String, age:Int): String = *Interpolations*(name,age)  
  
 def Computation(a: Int, b: Int): String = "Hi, \n now follows a quite hard calculation. We try ta add: \n\ta := %d \n\tb := %d\n\n\tresult is %d".format(a,b,a+b)  
  
 def testComputation(a: Int, b: Int): String = *Computation*(a,b)  
  
 def TakeTwo(str: String): String = {  
 if (str.length==2) return str  
 else return str.take(2)  
 }  
  
 def testTaleTwo(str: String): String = *TakeTwo*(str)  
}

*Выводы:* реализовал заданные классы, изучил конструкцию Seq, реализовал стандартные функции scala, использовал хвостовую рекурсию при реализации функций.. В конце получен корректный результат.