МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное автономное бюджетное образовательное учреждение высшего образования

«**Севастопольский государственный университет**»

кафедра Информационных систем

Лисянский Александр Игоревич

Институт информационных технологий и управления в технических системах

курс 1 группа ИС/м-11-о

09.04.02 Информационные системы и технологии (уровень магистратуры)

**ОТЧЁТ**

о лабораторном практикуме №6

по дисциплине

«Математические Основы и Парадигмы Современных Языков Программирования»

Отметка о зачёте \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Руководитель практикума

                                                                 Строганов В.А.

(должность) (подпись) (инициалы, фамилия)

Севастополь 2016

1. Цель работы

Исследовать способы реализации и использования параметризованных классов и функторов в языке Scala, получить практические навыки их применения.

1. Постановка задачи

2.1. Написать типизированный класс Box[T] который позволяет сохранять в себя значение типа Т (метод save) и получать сохранённое значение (метод get).

2.2. Преобразовать класс Box из п.1 в ковариативный функтор. Доказать, что он является ковариативным функтором.

2.3. Преобразовать класс Box в контравариативный функтор. Доказать, что он является контравариативным функтором.

2.4. Преобразовать класс Box в инвариативный функтор. Доказать, что он является инвариативным функтором.

3. Ход работы

trait Function[T] {

def get: T

}

trait CovarFunctor[T] {

def map[R](f: T => R): CovarFunctor[R]

}

trait Presentater[T] extends Function[T] with CovarFunctor[T] { outer =>

def source: Presentater[\_] = this

def map[R](f: T => R) = new Presentater[R] {

override def source = outer

def get = f(source.get)

}

}

object Presentater {

def apply[T](value: T) = new Presentater[T] {

def get = value

}

}

trait Setter[T] {

def set: T => Unit

}

trait ContraFunctor[T] {

def contramap[R](f: R => T): ContraFunctor[R]

}

trait Consumer[T] extends Setter[T] with ContraFunctor[T] { outer =>

def storage: Consumer[\_] = this

def contramap[R](f: R => T) = new Consumer[R] {

override def storage = outer

override def set = (newValue: R) => storage.set(f(newValue))

}

}

object Consumer {

def apply[T] = new Consumer[T] {

def set = (newValue: T) => println("Consumed this: " + newValue)

}

}

trait InvariantFunctor[T] {

def xmap[R](in: R => T, out: T => R): InvariantFunctor[R]

}

trait Box[T] extends Function[T] with Setter[T] with InvariantFunctor[T] { outer =>

def xmap[R](in: R => T, out: T => R) = new Box[R] {

def get = out(outer.get)

def set = (newValue: R) => outer.set(in(newValue))

}

}

Выводы:

В ходе выполнения данной лабораторной работы были изучены такие концепции функционального программирования, как ковариативный, контравариативный и инвариативный функторы. Также были получены навыки обращения с trait и анонимными классами, заменяя обычное объявление класса и конструктора на объявления trait и реализации функции конструктора в методе apply объекта компаньона. Помимо этого, была получена практика общения с Scala разработчиками на английском языке и получена рекомендация книги для освоения Scala и основ функционального программирования в разрезе языка Scala.