Министерство образования Республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Дисциплина «Современные языки программирования»

ЛАБОРАТОРНАЯ РАБОТА № 1

Тема: «Использование языка программирования Kotlin»

Вариант № 23

Выполнила: Сильченко А.В., гр.310901

Проверил: Усенко Ф. В.

Минск 2024

Задание: Классу Университет добавить методы Провести лекцию(), Провести семинар(), Провести лабораторный практикум(). Эти методы должны менять поля знания и оценки класса Студент. Добавить класс Детский сад.

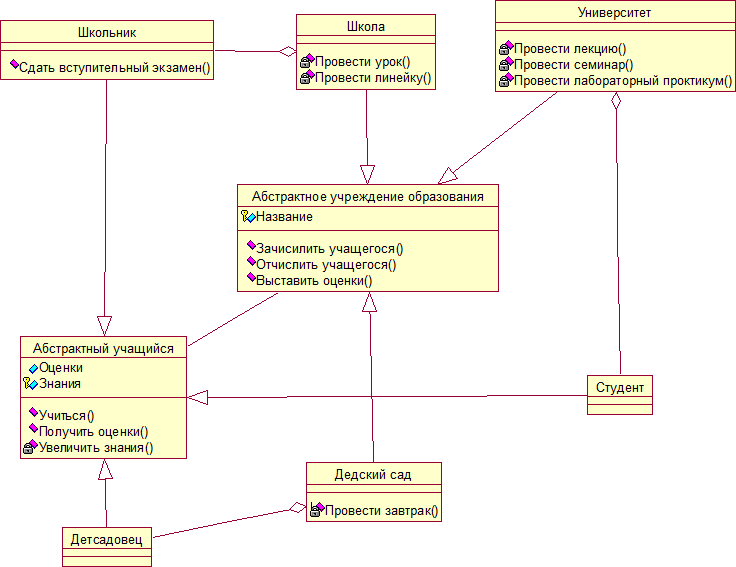


Рисунок 1 – Диаграмма классов.

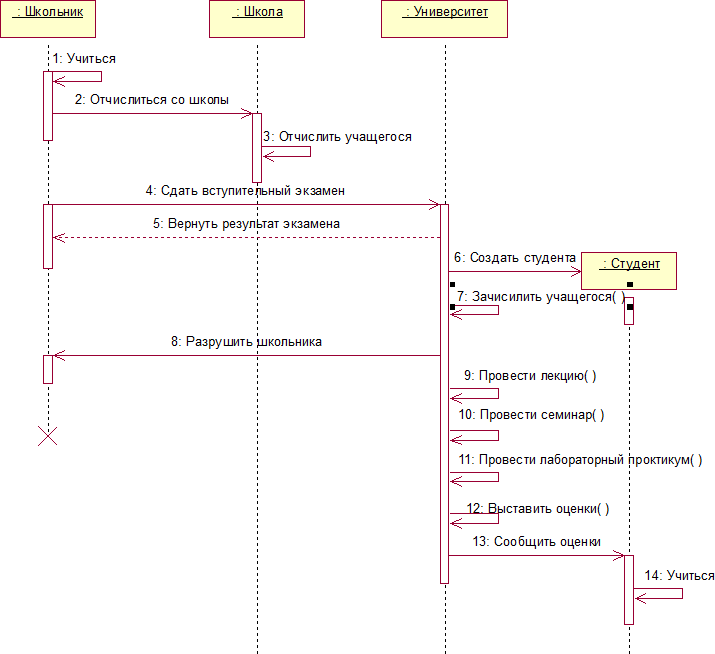


Рисунок 2 – Диаграмма последовательности.

Листинг кода:

Файл Main.kt:

import Student.\*  
import AbstractEducationalInstitution.\*  
import kotlin.system.exitProcess  
  
fun main(args: Array<String>){  
 *println*("Добро пожаловать в образовательную систему!")  
  
 // Создаем людей разных уровней  
 val kindergartener = Kindergartener()  
 val pupil = Pupil()  
 val student = Student()  
  
 // Создаем учебные заведения  
 val kindergarten = Kindergarten()  
 val school = School()  
 val university = University()  
  
 while(true) {  
 *println*()  
 *println*("1. Поступить в детсад")  
 *println*("2. Поступить в школу")  
  
 var choice: Int  
 try {  
 choice = *readln*().*toInt*()  
 } catch (e: Exception) {  
 *println*("Некорректный ввод")  
 continue  
 }  
 *println*()  
 when (choice) {  
 1-> {  
 // Поступление в детсад и последующий выпуск в школу  
 *println*("Поступаем в ${kindergarten.getInstitutionName()}")  
 kindergarten.enrollStudent(kindergartener)  
 kindergartener.study()  
 kindergartener.graduate()  
 kindergarten.expelStudent(kindergartener)  
 }  
 2-> {  
 //поступление в школу  
 *println*("Поступаем в ${school.getInstitutionName()}")  
 val passedSchool = school.enrollStudent(pupil)  
 if(passedSchool == true) {  
 while(true) {  
 *println*()  
 *println*("1. Учиться")  
 *println*("2. Закончить школу")  
 *println*("3. Назад в меню")  
 var choic: Int  
 try {  
 choic = *readln*().*toInt*()  
 } catch (e: Exception) {  
 *println*("Некорректный ввод")  
 continue  
 }  
 when(choic) {  
 1-> {  
 school.pupilStudy(pupil)  
 }  
 2-> {  
 val ftrstudent = pupil.graduate()  
 if (ftrstudent is Student) {  
 school.expelStudent(pupil)  
 *println*()  
 *println*("Желаете сдать вступительный экзамен?")  
 *println*("1. Да")  
 *println*("2. Нет")  
 var chh: Int = *readln*().*toInt*()  
 if(chh == 1) {  
 if (pupil.takeExam()) {  
 val newStudent = pupil.graduate()  
 if (newStudent is Student) {  
 university.enrollStudent(newStudent)  
 // Логика для университета  
 *universityRoutine*(newStudent, university)  
 }  
 } else {  
 *println*("Вы не сдали экзамен.")  
 continue  
 }  
 } else continue  
  
 } else continue  
 }  
 3 -> break // Возврат в главное меню  
 } } }  
 else continue  
 } } } }  
  
fun universityRoutine(student: Student, university: University): Boolean {  
 while (true) {  
 *println*()  
 *println*("1. Учиться")  
 *println*("2. Сходить на лекцию")  
 *println*("3. Сходить на семинар")  
 *println*("4. Сходить лабораторный практикум")  
 *println*("5. Закончить университет")  
 *println*("6. Назад в меню")  
 var cho: Int  
 try {  
 cho = *readln*().*toInt*()  
 } catch (e: Exception) {  
 *println*("Некорректный ввод")  
 continue  
 }  
 when (cho) {  
 1 -> {  
 student.study()  
 university.assignGrades(student)  
 }  
 2 -> university.publicConductLecture()  
 3 -> university.publicConductSeminar()  
 4 -> university.publicConductLab()  
 5 -> {  
 val graduate = student.graduate()  
 if (graduate != null && student.isReadyForGraduation()) {  
 university.expelStudent(student)  
 *exitProcess*(0) // Возврат значения, чтобы выйти из программы  
 } else {  
 continue  
 }  
 }  
 6 -> return false // Возврат в главное меню  
 } } }

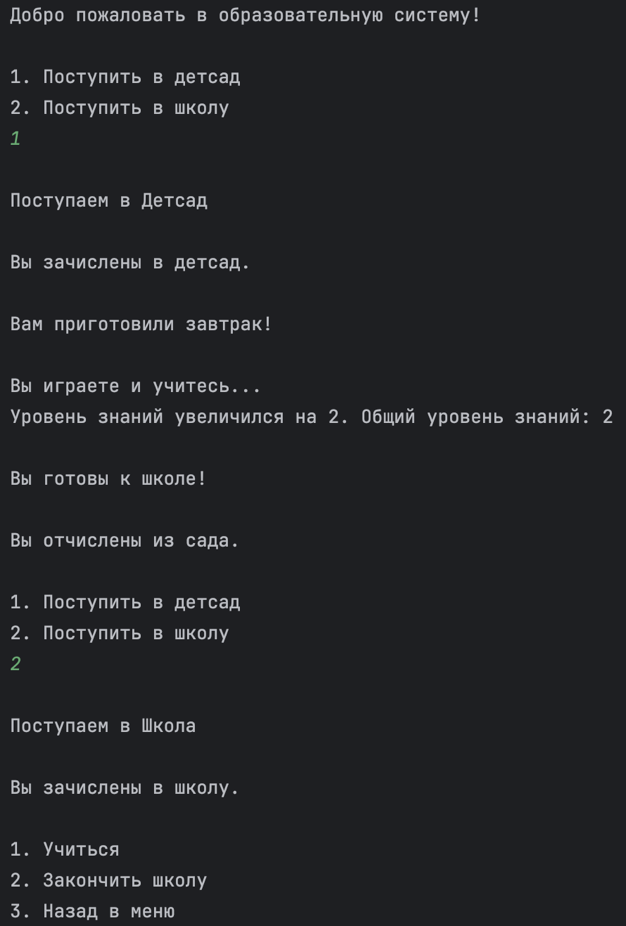
Файл Student.kt:

import kotlin.system.exitProcess  
  
abstract class AbstractStudent() {  
 var grades: MutableList<Int> = *mutableListOf*() //объявляем изменяемый список оценок студента  
 protected var knowledge: Int = 0  
  
 abstract fun study()  
 abstract fun increaseKnowledge(amount: Int)  
 abstract fun graduate(): AbstractStudent?  
 fun isReadyForGraduation(): Boolean {  
 val averageGrade = grades.*average*() //вычисляем среднюю оценку из списка оценок  
 return knowledge >= 25 && averageGrade >= 7 // Выпуск возможен при достаточном уровне знаний и оценках  
 }  
}  
  
class Kindergartener: AbstractStudent() {  
  
 override fun study() {  
 *println*()  
 *println*("Вы играете и учитесь...")  
 increaseKnowledge(2)  
 }  
  
 override fun increaseKnowledge(amount: Int) {  
 knowledge += amount  
 *println*("Уровень знаний увеличился на $amount. Общий уровень знаний: $knowledge")  
 }  
  
 override fun graduate(): AbstractStudent? {  
 return if (knowledge >= 2) {  
 *println*()  
 *println*("Вы готовы к школе!")  
 Pupil() // Возвращаем объект Pupil, так как это следующий этап.  
 } else {  
 *println*()  
 *println*("Вы пока не готовы к школе.")  
 null  
 }  
 }  
}  
  
class Pupil: AbstractStudent() {  
 fun takeExam(): Boolean {  
 //имитация сдачи вступительного экзамена  
 val success = (1..10).*random*() > 5 // Проходит экзамен с вероятностью 50%  
 if (success) {  
 *println*("Школьник сдал вступительный экзамен!")  
 }  
 else {  
 *println*("Школьник не сдал вступительный экзамен.")  
 }  
 return success  
 }  
 override fun study() {  
 *println*("Школьник учится...")  
 increaseKnowledge (2)  
 }  
  
 override fun increaseKnowledge(amount: Int) {  
 knowledge += amount  
 *println*("Уровень знаний увеличился на $amount. Общий уровень знаний: $knowledge")  
 }  
 override fun graduate(): AbstractStudent? {  
 return if (knowledge >= 10) {  
 *println*("Школьник готов стать студентом!")  
 Student() // Переход на следующий этап  
 } else {  
 *println*("Школьник пока не готов стать студентом.")  
 null  
 }  
 }  
}  
  
class Student: AbstractStudent() {  
 override fun study() {  
 *println*("Студент учится...")  
 increaseKnowledge (3)  
 }  
  
 override fun increaseKnowledge(amount: Int) {  
 knowledge += amount  
 *println*("Уровень знаний увеличился на $amount. Общий уровень знаний: $knowledge")  
 }  
 override fun graduate(): AbstractStudent? {  
 return if (knowledge >= 25) {  
 *println*("Студент готов работать!")  
 *exitProcess*(0)  
 } else {  
 *println*("Студент не готов работать.")  
 null  
 }  
 }  
}

Файл EducationInstitution.kt:

import kotlin.random.Random // Импортируем Random для генерации случайных чисел  
  
abstract class AbstractEducationalInstitution {  
 protected abstract val name: String  
 fun getInstitutionName(): String {  
 return name  
 }  
 abstract fun enrollStudent(student: AbstractStudent): Boolean  
 abstract fun expelStudent(student: AbstractStudent)  
 abstract fun assignGrades(student: AbstractStudent)  
}  
  
class University: AbstractEducationalInstitution() {  
 private var student: Student? = null  
 override val name = "Университет"  
 override fun enrollStudent(student: AbstractStudent): Boolean {  
 return if (student is Student) {  
 this.student = student //ссылка на текущий объект класса  
 *println*("Студент зачислен в университет.")  
 true  
 }  
 else {return false}  
 }  
 override fun expelStudent(student: AbstractStudent) {  
 if (this.student == student) {  
 this.student = null  
 *println*("Студент отчислен из университета и готов работать.")  
 }  
 }  
 override fun assignGrades(student: AbstractStudent) {  
 if (this.student == student) {  
 // Генерируем случайную оценку от 1 до 10  
 val grades = *List*(1) **{** Random.nextInt(1, 11) **}** student.grades.addAll(grades)  
 *println*("Оценки выставлены студенту: $grades")  
 }  
 }  
  
 fun publicConductLecture() {  
 conductLecture()  
 }  
  
 fun publicConductSeminar() {  
 conductSeminar()  
 }  
  
 fun publicConductLab() {  
 conductLab()  
 }  
  
 private fun conductLecture () {  
 *println*("Для студента проводится лекция.")  
 student?.study();  
 assignGrades(student!!) // Вызываем метод с текущим студентом  
 }  
 private fun conductSeminar() {  
 *println*("Проводится семинар.")  
 student?.study(); //оператор безопасного вызова (только для случаев не с null)  
 assignGrades(student!!) // Вызываем метод с текущим студентом  
 }  
 private fun conductLab() {  
 *println*("Проводится лабораторный практикум.")  
 student?.study();  
 assignGrades(student!!)  
 }  
}  
  
class School: AbstractEducationalInstitution() {  
 private var pupil: Pupil? = null  
 override val name = "Школа"  
 override fun enrollStudent(pupil: AbstractStudent): Boolean {  
 return if (pupil is Pupil) {  
 this.pupil = pupil  
 *println*()  
 *println*("Вы зачислены в школу.")  
 true  
 }  
 else {return false}  
 }  
 override fun expelStudent(pupil: AbstractStudent) {  
 if (this.pupil == pupil) {  
 this.pupil = null  
 *println*()  
 *println*("Школьник отчислен из школы.")  
 conductRuler()  
 }  
 }  
 override fun assignGrades(pupil: AbstractStudent) {  
 if (this.pupil == pupil) {  
 // Генерируем 3 случайные оценки от 1 до 10  
 val grades = *List*(1) **{** Random.nextInt(1, 11) **}** pupil.grades.addAll(grades)  
 *println*("Оценки выставлены школьнику: $grades")  
 }  
 }  
  
 fun pupilStudy(pupil: AbstractStudent) {  
 conductLesson ()  
 assignGrades(pupil)  
 }  
  
 private fun conductLesson () {  
 *println*()  
 *println*("Для школьников проводится урок.")  
 pupil?.study()  
 }  
 private fun conductRuler() {  
 *println*("Проводится линейка.")  
 }  
}  
  
class Kindergarten: AbstractEducationalInstitution() {  
 private var kindergartener: Kindergartener? = null  
 override val name = "Детсад"  
 private fun takeBreakfast() {  
 *println*()  
 *println*("Вам приготовили завтрак!")  
 }  
 override fun enrollStudent(kindergartener: AbstractStudent):Boolean {  
 return if (kindergartener is Kindergartener) {  
 this.kindergartener = kindergartener  
 *println*()  
 *println*("Вы зачислены в детсад.")  
 takeBreakfast()  
 true  
 }  
 else {return false}  
 }  
 override fun expelStudent(kindergartener: AbstractStudent) {  
 if (this.kindergartener == kindergartener) {  
 this.kindergartener = null  
 *println*()  
 *println*("Вы отчислены из сада.")  
 }  
 }  
 override fun assignGrades(kindergartener: AbstractStudent) {  
 if (this.kindergartener == kindergartener) {  
 // Генерируем случайную оценку от 1 до 10  
 val grades = *List*(1) **{** Random.nextInt(1, 11) **}** kindergartener.grades.addAll(grades)  
 *println*("Оценки выставлены детсадовцу.")  
 }  
 }  
}

Результаты работы программы представлены на рисунках 3 – 6.



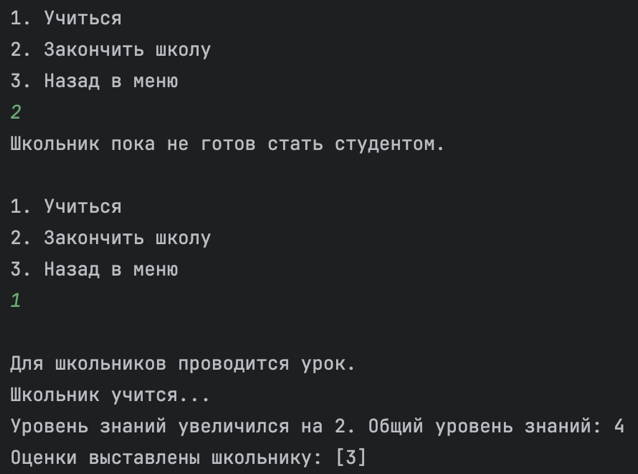


Рисунок 3 – Результат работы программы.

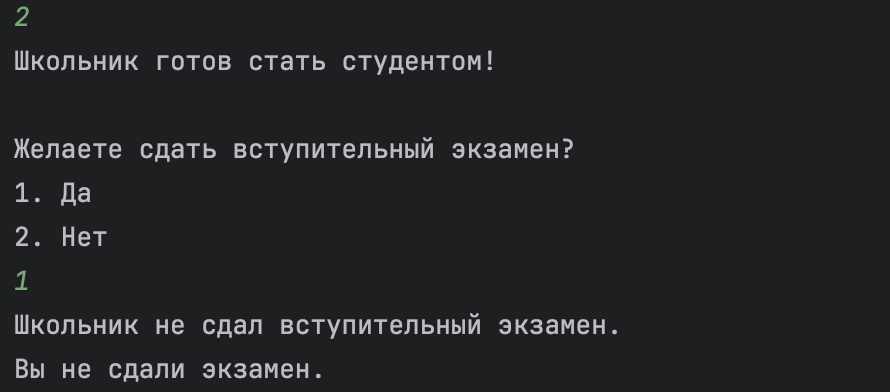


Рисунок 4 – Результат работы программы.

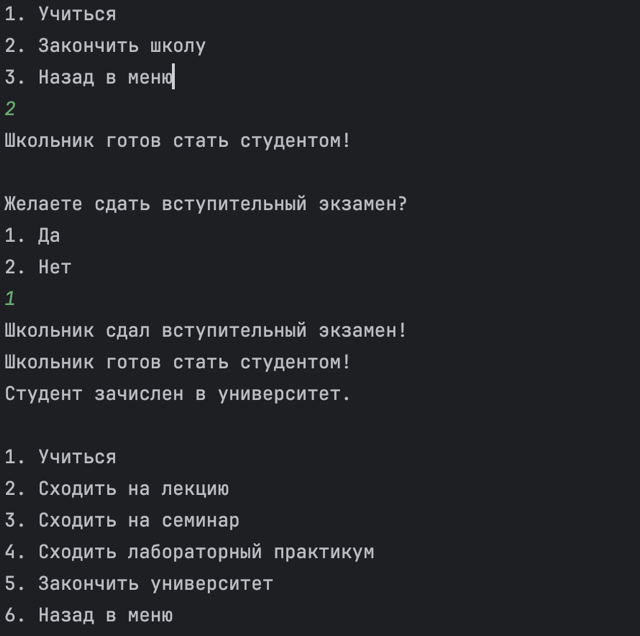


Рисунок 5 – Результат работы программы.

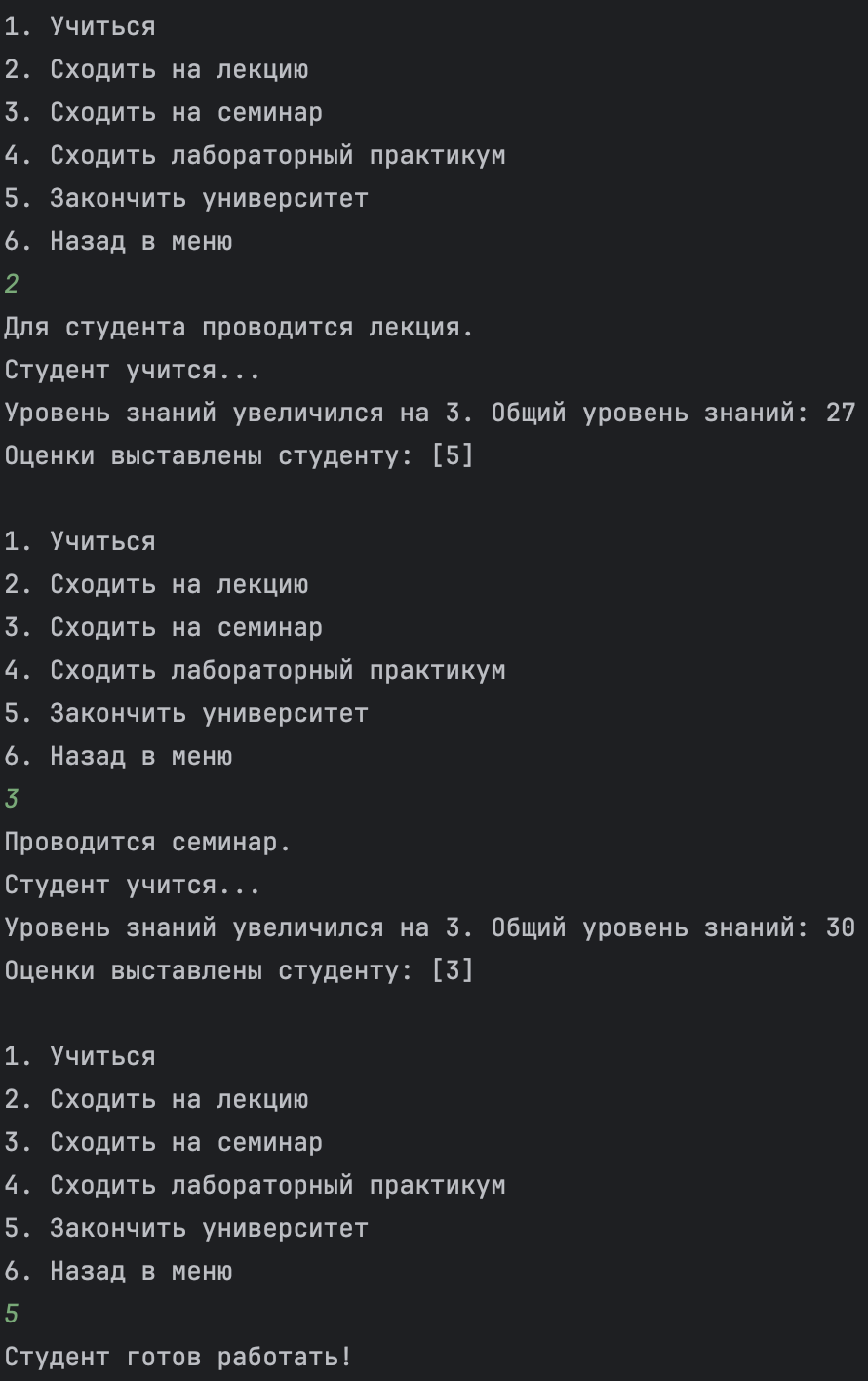


Рисунок 6 – Результат работы программы.

Вывод:В ходе выполнения лабораторной работы разработали приложение с использованием языка программирования Kotlin, познакомились с основами языка, научились анализировать диаграммы классов и последовательностей и преобразовывать их в код.