МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №4

Выполнил

Василюк П. О.,

студент группы ПО-5

Проверил

Крощенко А.А.,

ст. преп. Кафедры ИИТ,

«\_\_» \_\_\_\_\_\_\_\_ 2021 г.

Брест, 2021

**Цель работы:** приобрести навыки применения паттернов проектирования при решении практических задач с использованием языка Kotlin

**Вариант 2**

**Первая группа заданий.** Заводы по производству автомобилей. Реализовать возможность создавать автомобили различных типов на различных заводах.

**Вторая группа заданий.** Проект «Универсальная электронная карта». В проекте должна быть реализована универсальная электронная карта, в которой есть функции паспорта, страхового полиса, банковской карты и т. д.

**Третья группа заданий.** Проект «Принтеры». В проекте должны быть реализованы разные модели принтеров, которые выполняют разные виды печати.

**Выполнение:**

**Код программы:**

**Task\_1.kt**

fun main(args: Array<String>) {  
 val pclient = Client(PetroleFactory())  
 pclient.TestingRun()  
 val dclient = Client(DieselFactory())  
 dclient.TestingRun()  
 }  
  
abstract class AbstractFactory {  
 abstract fun CreatePassengerCar(): PassengerCar  
 abstract fun CreateTruckCar(): TruckCar  
}  
  
class DieselFactory : AbstractFactory() {  
 override fun CreatePassengerCar(): PassengerCar {  
 return DieselPassengerCar()  
 }  
  
 override fun CreateTruckCar(): TruckCar {  
 return DieselTruckCar()  
 }  
}  
  
class PetroleFactory : AbstractFactory() {  
 override fun CreatePassengerCar(): PassengerCar {  
 return PetrolePassengerCar()  
 }  
  
 override fun CreateTruckCar(): TruckCar {  
 return PetroleTruckCar()  
 }  
}  
  
abstract class PassengerCar {  
 abstract fun GetInfo()  
}  
  
abstract class TruckCar {  
 abstract fun GetInfo()  
}  
  
class DieselPassengerCar : PassengerCar() {  
 override fun GetInfo() {  
 *println*("Diesel passenger car is working")  
 }  
}  
  
class PetrolePassengerCar : PassengerCar() {  
 override fun GetInfo() {  
 *println*("Petrole passenger car is working")  
 }  
}  
  
class DieselTruckCar : TruckCar() {  
 override fun GetInfo() {  
 *println*("Diesel truck car is working")  
 }  
}  
  
class PetroleTruckCar : TruckCar() {  
 override fun GetInfo() {  
 *println*("Petrole truck car is working")  
 }  
}  
  
class Client(private val \_factory: AbstractFactory) {  
 fun GetPassengerCar(): PassengerCar {  
 return \_factory.CreatePassengerCar()  
 }  
  
 fun GetTruckCar(): TruckCar {  
 return \_factory.CreateTruckCar()  
 }  
  
 fun TestingRun() {  
 GetPassengerCar().GetInfo()  
 GetTruckCar().GetInfo()  
 }  
}

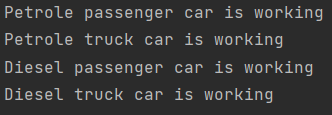
**Task\_2.kt**

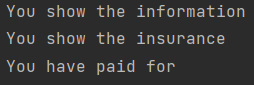
fun main(args: Array<String>) {  
 val myCard = ElectronicCard(Passport(), InsurancePolicy(), BankCard())  
 myCard.ShowInfo()  
 myCard.ShowInsurance()  
 myCard.PayFor()  
}  
  
class Passport {  
 fun ShowInfo() {  
 *println*("You show the information")  
 }  
}  
  
class InsurancePolicy {  
 fun ShowInsurance() {  
 *println*("You show the insurance")  
 }  
}  
  
class BankCard {  
 fun PayFor() {  
 *println*("You have paid for")  
 }  
}  
  
class ElectronicCard(  
 private val passport: Passport,  
 private val insurancePolic: InsurancePolicy,  
 private val bankCard: BankCard  
) {  
 fun ShowInfo() {  
 passport.ShowInfo()  
 }  
  
 fun ShowInsurance() {  
 insurancePolic.ShowInsurance()  
 }  
  
 fun PayFor() {  
 bankCard.PayFor()  
 }  
}

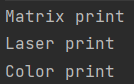
**Task\_3.kt**

fun main(args: Array<String>) {  
 val matrixPrinter = Context(MatrixPrinter())  
 matrixPrinter.ExecuteAlgorithm()  
 val laserPrinter = Context(LaserPrinter())  
 laserPrinter.ExecuteAlgorithm()  
 val colorPrinter = Context(ColorPrinter())  
 colorPrinter.ExecuteAlgorithm()  
 }  
  
interface IPrinter {  
 fun Print()  
}  
  
class ColorPrinter : IPrinter {  
 override fun Print() {  
 *println*("Color print")  
 }  
}  
  
class MatrixPrinter : IPrinter {  
 override fun Print() {  
 *println*("Matrix print")  
 }  
}  
  
class LaserPrinter : IPrinter {  
 override fun Print() {  
 *println*("Laser print")  
 }  
}  
  
class Context(type: IPrinter?) {  
 var contextPrinter: IPrinter? = null  
  
 fun ExecuteAlgorithm() {  
 contextPrinter!!.Print()  
 }  
  
 init {  
 contextPrinter = type  
 }  
}

**Результаты:**

****

****

****

**Вывод:** приобрести навыки применения паттернов проектирования при решении практических задач с использованием языка Kotlin