Министерство образования и молодежной политики Свердловской области



ГАПОУ СО «Екатеринбургский колледж транспортного строительства»

# Отчет по программе «Практическое занятие 9.2»

Выполнил: Меньшиков Артур Андреевич

Группа: ПР-22

Преподаватель: Мирошниченко Г. В.

2023

**Задание:**

**1.** Задача на взаимодействие между классами. Разработать систему «Автобаза». Диспетчер распределяет заявки на Рейсы между Водителями и назначает для этого Автомобиль. Водитель может сделать заявку на ремонт. Диспетчер может отстранить Водителя от работы. Водитель делает отметку о выполнении Рейса и состоянии Автомобиля.

**2.** Задача на взаимодействие между классами. Разработать систему «Железнодорожная касса». Пассажир делает заявку на станцию назначения, время и дату поездки. Система регистрирует Заявку и осуществляет поиск соответствующего Поезда. Пассажир делает выбор Поезда и получает Счет на оплату. Кассир вводит номера Поездов, промежуточные и конечные станции, цены.

**Входные и выходные данные**

Задание 1:

driver1 – String/строка

driver2 – String/строка

vehicle1 – String/строка

vehicle2 – String/строка

trip1 – String/строка

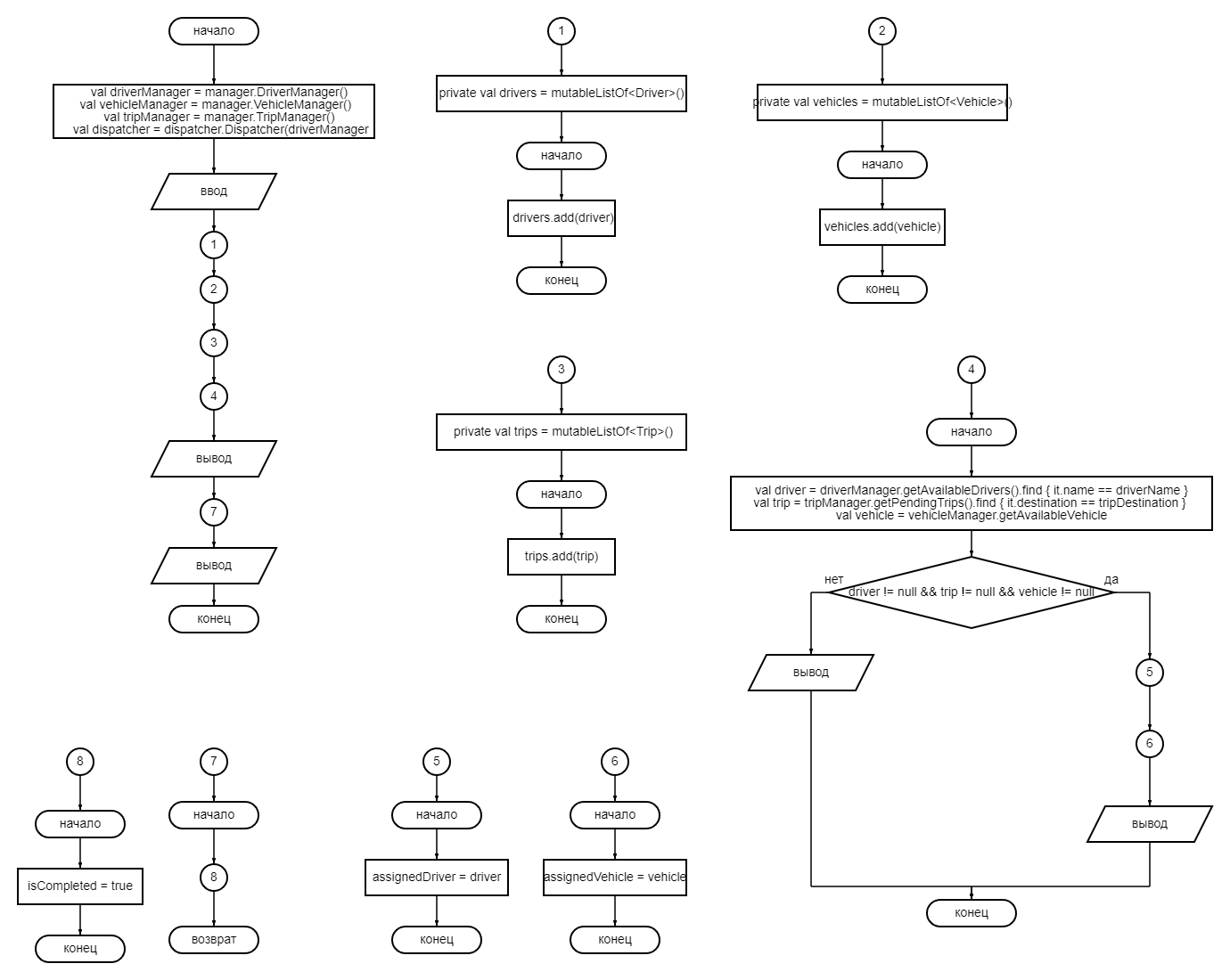
trip2 – String/строка

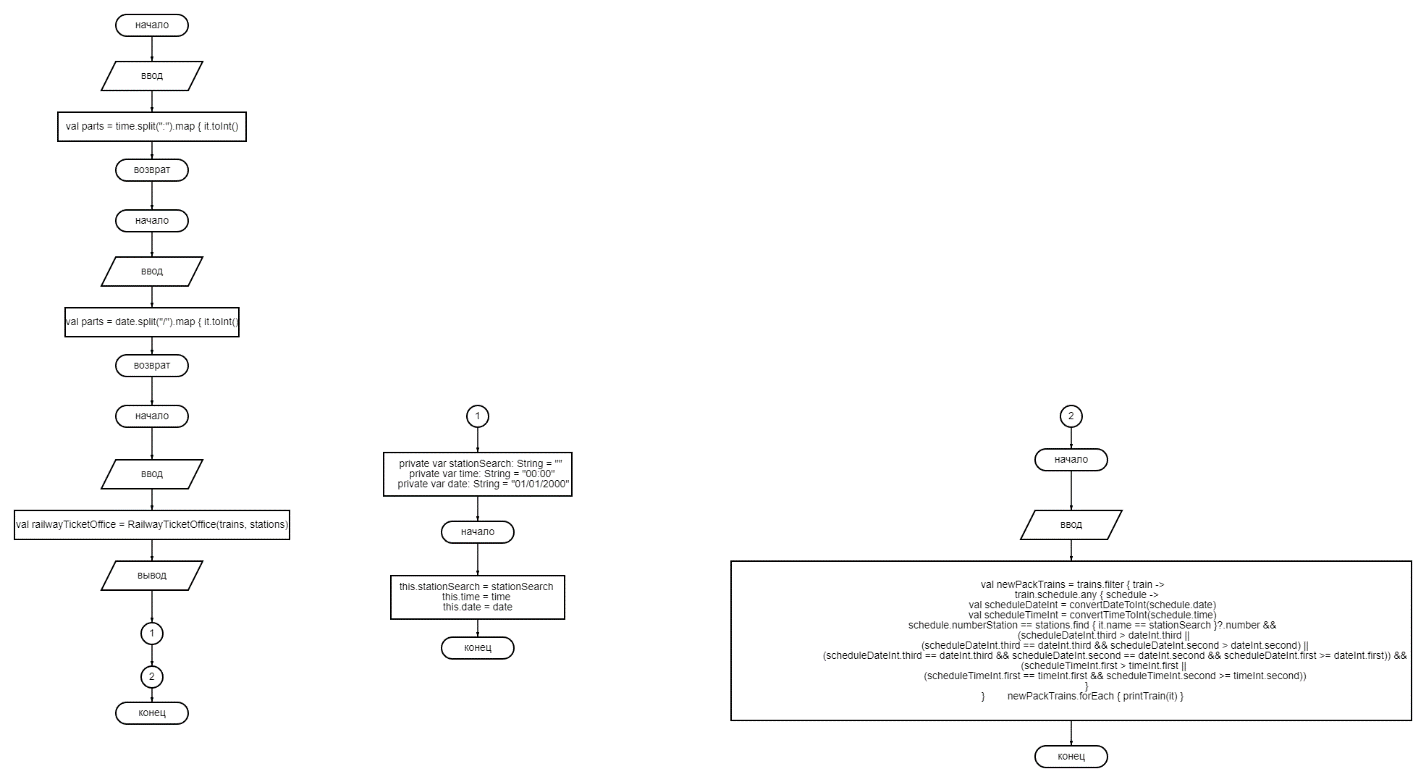
Задание 2:

stations – mutableList<String>/изменяемый список

trains – mutableList<String>/изменяемый список

**Блок-схема**

Задание 1:



**Листинг программы (если есть)**

Задание 1:

import java . time . LocalDateTime  
import java . time . format . DateTimeFormatter  
fun main() {  
 val driverManager = manager.DriverManager()  
 val vehicleManager = manager.VehicleManager()  
 val tripManager = manager.TripManager()  
 val dispatcher = dispatcher.Dispatcher(driverManager, vehicleManager, tripManager)  
  
 *println*("введите имя водителя")  
 val driver1 = driver.Driver("Водитель ${*readln*()!!.toString()}")  
 val driver2 = driver.Driver("Водитель ${*readln*()!!.toString()}")  
 driverManager.addDriver(driver1)  
 driverManager.addDriver(driver2)  
  
 *println*("введите навзание автомобиля")  
 val vehicle1 = vehicle.Vehicle("Автомобиль ${*readln*()!!.toString()}")  
 val vehicle2 = vehicle.Vehicle("Автомобиль ${*readln*()!!.toString()}")  
 vehicleManager.addVehicle(vehicle1)  
 vehicleManager.addVehicle(vehicle2)  
  
 *println*("введите пункт назначения")  
 val trip1 = trip.Trip("Пункт назначения ${*readln*()!!.toString()}")  
 val trip2 = trip.Trip("Пункт назначения ${*readln*()!!.toString()}")  
 tripManager.addTrip(trip1)  
 tripManager.addTrip(trip2)  
  
 dispatcher.assignTripToDriver(driver1.name, trip1.destination)  
 dispatcher.assignTripToDriver(driver2.name, trip2.destination)  
  
 *println*("Назначенные рейсы:")  
 tripManager.getPendingTrips().*forEach* {  
 *println*("${**it**.destination} назначен на ${**it**.assignedDriver?.name} с автомобилем ${**it**.assignedVehicle?.model}")  
 }  
  
 trip1.completeTrip()  
 fun convertTimeToInt(time: String): Pair<Int, Int> {  
 val parts = time.*split*(":").*map* { **it**.*toInt*() }  
 return Pair(parts[0], parts[1])  
 }  
  
 *println*("Завершенные рейсы:")  
 tripManager.getPendingTrips().*filter* { **it**.isCompleted }.*forEach* {  
 *println*("${**it**.destination} завершен водителем ${**it**.assignedDriver?.name}")  
 }  
 fun convertDateToInt(date: String): Triple<Int, Int, Int> {  
 val parts = date.*split*("/").*map* { **it**.*toInt*() }  
 return Triple(parts[0], parts[1], parts[2])  
 }  
  
 *println*("Еще не завершенные рейсы:")  
 tripManager.getPendingTrips().*filter* { !**it**.isCompleted }.*forEach* {  
 *println*("${**it**.destination} еще не завершен и назначен водителю ${**it**.assignedDriver?.name}")  
 }  
}

package dispatcher  
  
import manager.DriverManager  
import manager.VehicleManager  
import manager.TripManager  
class Dispatcher(  
 private val driverManager: DriverManager,  
 private val vehicleManager: VehicleManager,  
 private val tripManager: TripManager  
){  
 fun assignTripToDriver(driverName: String, tripDestination: String) {  
 val driver = driverManager.getAvailableDrivers().*find* { **it**.name == driverName }  
 val trip = tripManager.getPendingTrips().*find* { **it**.destination == tripDestination }  
 val vehicle = vehicleManager.getAvailableVehicles().*firstOrNull*()  
 if (driver != null && trip != null && vehicle != null) {  
 trip.assignDriver(driver)  
 trip.assignVehicle(vehicle)  
 *println*("Trip assigned successfully to ${driver.name} for destination ${trip.destination}.")  
 } else {  
 *println*("Failed to assign trip.")  
 }  
 }  
}

package driver  
  
class Driver(val name: String) {  
 var isAvailable: Boolean = true  
}

package manager  
  
import driver.Driver  
import trip.Trip  
import vehicle.Vehicle  
class DriverManager {  
 private val drivers = *mutableListOf*<Driver>()  
  
 fun addDriver(driver: Driver) {  
 drivers.add(driver)  
 }  
  
 fun removeDriver(driver: Driver) {  
 drivers.remove(driver)  
 }  
  
 fun getAvailableDrivers(): *List*<Driver> {  
 return drivers.*filter* { **it**.isAvailable }  
 }  
}

package trip  
  
import driver.Driver  
import vehicle.Vehicle  
  
class Trip(val destination: String) {  
 var isCompleted: Boolean = false  
 var assignedDriver: Driver? = null  
 var assignedVehicle: Vehicle? = null  
  
 fun completeTrip() {  
 isCompleted = true  
 }  
  
 fun assignDriver(driver: Driver) {  
 assignedDriver = driver  
 }  
  
 fun assignVehicle(vehicle: Vehicle) {  
 assignedVehicle = vehicle  
 }  
}

package manager  
  
import driver.Driver  
import trip.Trip  
import vehicle.Vehicle  
class TripManager {  
 private val trips = *mutableListOf*<Trip>()  
  
 fun addTrip(trip: Trip) {  
 trips.add(trip)  
 }  
  
 fun completeTrip(trip: Trip) {  
 trip.completeTrip()  
 }  
  
 fun getPendingTrips(): *List*<Trip> {  
 return trips.*filter* { !**it**.isCompleted }  
 }  
}

package vehicle  
  
class Vehicle(val model: String) {  
 var isFunctional: Boolean = true  
  
 fun repair() {  
 isFunctional = true  
 }  
}

package manager  
  
import driver.Driver  
import trip.Trip  
import vehicle.Vehicle  
class VehicleManager {  
 private val vehicles = *mutableListOf*<Vehicle>()  
  
 fun addVehicle(vehicle: Vehicle) {  
 vehicles.add(vehicle)  
 }  
  
 fun repairVehicle(vehicle: Vehicle) {  
 vehicle.repair()  
 }  
  
 fun getAvailableVehicles(): *List*<Vehicle> {  
 return vehicles.*filter* { **it**.isFunctional }  
 }  
}

Задание 2:

import java.time.LocalDateTime  
import java.time.format.DateTimeFormatter  
  
fun convertTimeToInt(time: String): Pair<Int, Int> {  
 val parts = time.*split*(":").*map* { **it**.*toInt*() }  
 return Pair(parts[0], parts[1])  
}  
  
fun convertDateToInt(date: String): Triple<Int, Int, Int> {  
 val parts = date.*split*("/").*map* { **it**.*toInt*() }  
 return Triple(parts[0], parts[1], parts[2])  
}  
  
fun main() {  
 val stations = *mutableListOf*(Station(1, "Дмитров"),)  
  
 val trains = *mutableListOf*(Train(1, *listOf*(Schedule(1, "7:00", "01/03/2023"), Schedule(6, "17:00", "02/03/2023")), 89.0))  
  
 val railwayTicketOffice = RailwayTicketOffice(trains, stations)  
 *println*("От станции Дмитров")  
 *println*("Билет до Катуар на 9:00 01/03/2023")  
 railwayTicketOffice.registration("Катуар", "9:00", "01/03/2023")  
 railwayTicketOffice.searchTrain()  
}

class Train(val number: Int, val schedule: *List*<Schedule>, val price: Double) {  
 fun getEndStation(): Schedule = schedule.*last*()  
}

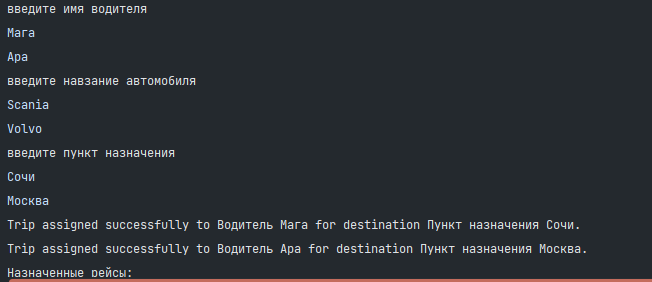
data class Station(val number: Int, val name: String)

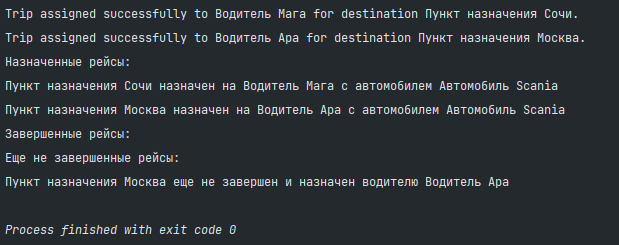
data class Schedule(val numberStation: Int, val time: String, val date: String)

class RailwayTicketOffice(private val trains: *MutableList*<Train>, private val stations: *MutableList*<Station>) {  
 private var stationSearch: String = ""  
 private var time: String = "00:00"  
 private var date: String = "01/01/2000"  
  
 fun registration(stationSearch: String, time: String, date: String) {  
 this.stationSearch = stationSearch  
 this.time = time  
 this.date = date  
 }  
  
 fun searchTrain() {  
 val timeInt = *convertTimeToInt*(time)  
 val dateInt = *convertDateToInt*(date)  
  
 val newPackTrains = trains.*filter* { train ->  
 train.schedule.*any* { schedule ->  
 val scheduleDateInt = *convertDateToInt*(schedule.date)  
 val scheduleTimeInt = *convertTimeToInt*(schedule.time)  
 schedule.numberStation == stations.*find* { **it**.name == stationSearch }?.number &&  
 (scheduleDateInt.third > dateInt.third ||  
 (scheduleDateInt.third == dateInt.third && scheduleDateInt.second > dateInt.second) ||  
 (scheduleDateInt.third == dateInt.third && scheduleDateInt.second == dateInt.second && scheduleDateInt.first >= dateInt.first)) &&  
 (scheduleTimeInt.first > timeInt.first ||  
 (scheduleTimeInt.first == timeInt.first && scheduleTimeInt.second >= timeInt.second))  
 }  
 }  
  
 newPackTrains.*forEach* { printTrain(**it**) }  
 }  
  
 fun printTrain(train: Train) {  
 *println*("\n=====================")  
 *println*("Номер поезда: ${train.number}")  
 *println*("Станции: ")  
 train.schedule.*forEach* { schedule ->  
 *println*("${schedule.numberStation}. ${stations.*find* { **it**.number == schedule.numberStation }?.name} - ${schedule.time} | ${schedule.date}")  
 }  
 *println*("Конечная станция: ${stations.*find* { **it**.number == train.getEndStation().numberStation }?.name}")  
 *println*("Дата и время: ${train.getEndStation().time} | ${train.getEndStation().date}")  
 *println*("Цена: ${train.price}")  
 *println*("=====================")  
 }  
}

**Тестовые ситуации**

Задание 1:





Задание 2:

