import org.apache.poi.ss.usermodel.Cell  
import org.apache.poi.ss.usermodel.CellType  
import org.apache.poi.ss.usermodel.WorkbookFactory  
import java.io.File  
  
class Student(val group: String, // группа  
 val name: String, // ФИО  
 val ex: Int, // доп баллы  
 val lecAttendance: List<Boolean>, // посещаемость лекций  
 val labAttendance: List<Boolean>, // посещаемость лаб  
 val labWorks: List<Int>, // оценки за ЛР  
 val exams: List<Int>, // оценки за ЭП  
 val tests: List<Int>, // оценки за КР  
 val attest: String) { // аттестация  
 // Вывод студента  
 fun write() {  
 *println*("Студент: $name\n" +  
 "Доп баллы: $ex\n" +  
 "Посещаемость: ${attendanceGrade()}\n" +  
 "Оценки за:\n" +  
 "\tЛабораторные работы: $labWorks\n" +  
 "\tЭкзаменационный проект: $exams\n" +  
 "\tКонтрольные работы: $tests\n" +  
 "Итого баллов: ${getTotalScore() + ex}\n")  
 }  
 // Подсчёт оценки посещаемости  
 fun attendanceGrade(): Int {  
 val lecTrueCount = lecAttendance.*count* **{ it }** val labTrueCount = labAttendance.*count* **{ it }** val attendScore = lecTrueCount.toDouble() / lecAttendance.size \* 5 + labTrueCount.toDouble() / labAttendance.size \* 5  
 return when {  
 attendScore < 4 -> 0  
 attendScore < 6 -> 4  
 attendScore < 7.5 -> 6  
 attendScore < 9 -> 8  
 else -> 10  
 }  
 }  
 // Подсчёт всех баллов для допуска (без ЭП и доп баллов)  
 fun getTotalScore(): Int {  
 return attendanceGrade() + labWorks.*sum*() + tests.*sum*() \* 2  
 }  
 // Получение данных об экзамене  
 fun evalPreGrade(): String {  
 return when {  
 getTotalScore() < 30 -> "недопуск"  
 (getTotalScore() + ex) < 60 -> "допуск"  
 (getTotalScore() + ex) < 75 -> "САМОКАТ"  
 (getTotalScore() + ex) < 90 -> "ХОРОШО"  
 else -> "ОТЛИЧНО"  
 }  
 }  
}  
  
data class GroupStats(  
 val avgAttendance: Double,  
 val avgLabWorks: List<Double>,  
 val avgExams: List<Double>,  
 val avgTests: List<Double>,  
 val commonAttest: String,  
 val avgCommonLabWorks: Double,  
 val avgCommonExams: Double,  
 val avgCommonTests: Double,  
 val avgCommonAccess: Double,  
 val avgCommonTotal: Double,  
 val avgPercentLabWorks: Double,  
 val avgPercentExams: Double,  
 val avgPercentTests: Double,  
 val avgPercentAccess: Double,  
 val avgPercentTotal: Double  
)  
// Список средних значений из Int  
fun averageIntList(lists: List<List<Int>>): List<Double> {  
 if (lists.isEmpty()) return *emptyList*()  
  
 val size = lists.*first*().size  
 return *List*(size) **{** index **->** lists.*map* **{ it**[index] **}**.*average*()  
 **}**}  
  
// Выбор ат ил н для группы в зависимости от того, чего больше  
fun mostCommonAttest(attests: List<String>): String {  
 return attests.*groupBy* **{ it }**.*maxByOrNull* **{ it**.value.size **}**?.key ?: ""  
}  
  
// Получение оценки из ячейки. Если там число, то возвращаем его, если строка, то преобразуем в баллы  
fun getCellGrade(cell: Cell): Int{  
 return when (cell.*cellType*) {  
 CellType.*STRING* -> {  
 val stringValue = cell.*stringCellValue* when (stringValue) {  
 "д" -> 2  
 "", "-" -> 0  
 else -> stringValue.*toIntOrNull*()  
 ?: 0 // Попробуем преобразовать строку в число, если это не удастся, то 0  
 }  
 }  
 else -> cell.*numericCellValue*.toInt()  
 }  
}  
  
// Чтение файла  
fun fileRead(): List<Student> {  
 val students = *mutableListOf*<Student>()  
  
 val file = File("students.xlsx")  
 val workbook = WorkbookFactory.create(file.*inputStream*())  
 // Проход по всем листам  
 for (sheetNum in 0 *until* workbook.*numberOfSheets*) {  
 val sheet = workbook.getSheetAt(sheetNum)  
 var rowC = 0  
 // Выбор листов групп  
 if (("36" in sheet.*sheetName*) or ("39" in sheet.*sheetName*))  
 for (row in sheet) {  
 // Пропуск двух строк  
 if (rowC > 1)  
 if (row.getCell(0) != null) {  
 val name = row.getCell(1).*stringCellValue* // получение ФИО  
 val ex = row.getCell(2).*numericCellValue*.toInt() // получение доп баллов  
  
 val lecAttendanceList = *mutableListOf*<Boolean>()  
 val labAttendanceList = *mutableListOf*<Boolean>()  
 var cellIndex = 4  
 // Чтение данных о посещаемости отдельно для лекций и для лаб  
 var leclab = true  
 while (sheet.getRow(0).getCell(cellIndex).*cellType* != CellType.*STRING*) {  
 // переключение счёта лекций и лаб  
 if (leclab and (sheet.getRow(0).getCell(cellIndex).*numericCellValue* == 1.0))  
 leclab = false  
 else if (!leclab and (sheet.getRow(0).getCell(cellIndex).*numericCellValue* == 1.0))  
 leclab = true  
 var value = ""  
 if (row.getCell(cellIndex) != null)  
 value = row.getCell(cellIndex).*stringCellValue* if (leclab)  
 lecAttendanceList.add(value == "+")  
 else  
 labAttendanceList.add(value == "+")  
 cellIndex += 1  
 }  
  
 // Чтение оценок за лабораторные работы  
 val labWorks = *mutableListOf*<Int>()  
 for (i in cellIndex..(cellIndex + 6)) { // Колонки с ЛР1 по ЛР7  
 val value = *getCellGrade*(row.getCell(i))  
 labWorks.add(value)  
 }  
 cellIndex += 7  
  
 // Чтение оценок за экзаменационные работы  
 val exams = *mutableListOf*<Int>()  
 for (i in cellIndex..(cellIndex + 4)) { // Колонки с ЭП1 по ЭП5  
 val value = *getCellGrade*(row.getCell(i))  
 exams.add(value)  
 }  
 cellIndex += 5  
  
 // Чтение оценок за контрольные работы  
 val tests = *mutableListOf*<Int>()  
 for (i in cellIndex..(cellIndex + 2)) { // Колонки с КР1 по КР3  
 val value = *getCellGrade*(row.getCell(i))  
 tests.add(value)  
 }  
 cellIndex += 32  
  
 // Создание объекта Student и добавление его в список  
 val student = Student(sheet.*sheetName*, name, ex, lecAttendanceList, labAttendanceList, labWorks, exams, tests, row.getCell(cellIndex).*stringCellValue*)  
 students.add(student)  
 } else break  
 rowC += 1  
 }  
 }  
  
 workbook.close()  
  
 return students  
}  
  
// Вывод в таблицу групп значений ЛР, ЭП и КР  
fun printFormatted(value: Double): String {  
 return when {  
 value < 0.5 -> "-".*padEnd*(4)  
 value < 1 -> "+".*padEnd*(4)  
 value < 1.5 -> "++".*padEnd*(4)  
 value < 2.5 -> "д".*padEnd*(4)  
 else -> "${String.*format*("%.1f", value).*padEnd*(3)} "  
 }  
}  
  
fun main() {  
 */\*\*  
 \* Дана таблица с результатами обучения Вашей группы в семестре. Необходимо  
 прочитать все данные о посещаемости и все оценки за КР, ЛР и ЭП. Создать структуру классов  
 для хранения информации.  
 1 Прочитать суммарные баллы и оценки. Найти лабораторную работу, которую  
 выполнили наибольшее количество человек на любую оценку среди неполучивших допуск.  
 Найти КР, которую выполнили наименьшее количество человек хотя бы на допуск среди не  
 поливших допуск.  
 \*/* val students = *fileRead*()  
  
 val stoopid = students.*filter* **{ it**.getTotalScore() < 30 **}** var max = 0  
 var min = 60  
 var maxL = 1  
 var minK = 1  
 for (i in 0..6) {  
 var count = 0  
 stoopid.*forEach***{** count += if (**it**.labWorks.get(i) > 0) 1 else 0  
 **}** if (count > max) {  
 max = count  
 maxL = i + 1  
 }  
 }  
 for (i in 0..2) {  
 var count = 0  
 stoopid.*forEach***{** count += if (**it**.tests.get(i) > 0) 1 else 0  
 **}** if (count < min) {  
 min = count  
 minK = i + 1  
 }  
 }  
 *println*("Лабораторная работа №$maxL была сдана наибольшим количеством студентов без допуска")  
 *println*("Контрольная работа №$minK была сдана наименьшим количеством студентов без допуска\n")  
  
 */\*\*  
 \* 2 Рассчитать самостоятельно в программе баллы за посещаемость, КР, ЛР и ЭП так же,  
 как в документе. Найти сумму баллов и вывести количество баллов за посещаемость, ЛР, ЭП,  
 КР, ИТОГО для каждого студента.  
 \*/* students.*forEach* **{** *println*("Студент: ${**it**.name}\n" +  
 "Доп баллы: ${**it**.ex}\n" +  
 "Посещаемость: ${**it**.attendanceGrade()}\n" +  
 "Оценки за:\n" +  
 "\tЛабораторные работы: ${**it**.labWorks}\tВсего: ${**it**.labWorks.*sum*()}\n" +  
 "\tЭкзаменационный проект: ${**it**.exams}\tВсего: ${**it**.exams.*sum*()}\n" +  
 "\tКонтрольные работы: ${**it**.tests}\tВсего: ${**it**.tests.*sum*() \* 2}\n" +  
 "Итого баллов: ${**it**.getTotalScore() + **it**.ex + **it**.exams.*sum*()}\n")  
 **}** */\*\*  
 \* 3 Сформировать рейтинг так же, как и в разделе рейтинг программно. Вывести 5  
 худших среди получивших допуск, сгруппировав их по группам. Вывести 5 лучших среди не  
 получивших допуск, сгруппировав их по группам в том же формате, как и в разделе рейтинг,  
 то есть выводить проценты сделанного.  
 \*/* val rating = students.*sortedBy* **{ it**.name **}**.*sortedByDescending* **{ it**.getTotalScore() + **it**.ex **}** val studOnlyAccess = rating.*filter* **{ it**.evalPreGrade() == "допуск" **}**.*takeLast*(5)  
 val studNoAccess = rating.*filter* **{ it**.evalPreGrade() == "недопуск" **}**.*take*(5)  
  
 *println*("№ ${"Группа".*padEnd*(6)} ${"Фамилия И О".*padEnd*(20)} ${"ат".*padEnd*(2)} " +  
 "${"Экзамен".*padEnd*(8)} ${"Пос".*padEnd*(3)} " +  
 "${"ЛР".*padEnd*(5)} " +  
 "${"ЭП".*padEnd*(5)} " +  
 "${"КР".*padEnd*(5)} " +  
 "${"Допуск".*padEnd*(6)} " +  
 "${"Итого".*padEnd*(5)}")  
 *println*()  
 studOnlyAccess.*forEachIndexed* **{** index, it **->** *println*("${(index + 1).toString().*padEnd*(2)} ${it.group.*padEnd*(6)} ${it.name.*padEnd*(20)} ${it.attest.*padEnd*(2)} " +  
 "${it.evalPreGrade().*padEnd*(8)} ${(it.attendanceGrade() \* 10).toString().*padEnd*(3)} " +  
 "${String.*format*("%.1f", it.labWorks.*sum*().toDouble() / (it.labWorks.size \* 5) \* 100).*padEnd*(5)} " +  
 "${String.*format*("%.1f", it.exams.*sum*().toDouble() / (it.exams.size \* 5) \* 100).*padEnd*(5)} " +  
 "${String.*format*("%.1f", it.tests.*sum*().toDouble() / (it.tests.size \* 5) \* 100).*padEnd*(5)} " +  
 "${String.*format*("%.1f", it.getTotalScore().toDouble() / (it.attendanceGrade() + it.labWorks.size \* 5 + it.tests.size \* 10) \* 100).*padEnd*(6)} " +  
 "${(it.getTotalScore() + it.ex + it.exams.*sum*()).toString().*padEnd*(5)}") **}** *println*()  
 studNoAccess.*forEachIndexed* **{** index, it **->** *println*("${(index + 1).toString().*padEnd*(2)} ${it.group.*padEnd*(6)} ${it.name.*padEnd*(20)} ${it.attest.*padEnd*(2)} " +  
 "${it.evalPreGrade().*padEnd*(10)} ${(it.attendanceGrade() \* 10).toString().*padEnd*(3)} " +  
 "${String.*format*("%.1f", it.labWorks.*sum*().toDouble() / (it.labWorks.size \* 5) \* 100).*padEnd*(5)} " +  
 "${String.*format*("%.1f", it.exams.*sum*().toDouble() / (it.exams.size \* 5) \* 100).*padEnd*(5)} " +  
 "${String.*format*("%.1f", it.tests.*sum*().toDouble() / (it.tests.size \* 5) \* 100).*padEnd*(5)} " +  
 "${String.*format*("%.1f", it.getTotalScore().toDouble() / (it.attendanceGrade() + it.labWorks.size \* 5 + it.tests.size \* 10) \* 100).*padEnd*(6)} " +  
 "${(it.getTotalScore() + it.ex + it.exams.*sum*()).toString().*padEnd*(5)}") **}** */\*\*  
 \* 4 Сформировать программно раздел группы так же, как сформировано в таблице и  
 вывести на экран.  
 \*/* val groupByGroup = students.*groupBy* **{ it**.group **}** val results = groupByGroup.*mapValues* **{** entry **->** val groupStudents = entry.value  
  
 val avgAttendance = groupStudents.*map* **{ it**.attendanceGrade() **}**.*average*() // средняя посещаемость  
 val avgLabWorks = *averageIntList*(groupStudents.*map* **{ it**.labWorks **}**) // средний балл за отдельные ЛР  
 val avgExams = *averageIntList*(groupStudents.*map* **{ it**.exams **}**) // средний балл за отдельные ЭП  
 val avgTests = *averageIntList*(groupStudents.*map* **{ it**.tests **}**) // средний балл за отдельные КР  
 val commonAttest = *mostCommonAttest*(groupStudents.*map* **{ it**.attest **}**) // аттестация по группе  
  
 val avgCommonLabWorks = groupStudents.*map* **{ it**.labWorks.*sum*() **}**.*average*() // средний балл за все ЛР  
 val avgCommonExams = groupStudents.*map* **{ it**.exams.*sum*() **}**.*average*() // средний балл за все ЭП  
 val avgCommonTests = groupStudents.*map* **{ it**.tests.*sum*() \* 2 **}**.*average*() // средний балл за все КР  
 val avgCommonAccess = groupStudents.*map* **{ it**.attendanceGrade() + **it**.labWorks.*sum*() + **it**.tests.*sum*() \* 2 **}**.*average*() // средний балл на допуск  
 val avgCommonTotal = groupStudents.*map* **{ it**.attendanceGrade() + **it**.labWorks.*sum*() + **it**.tests.*sum*() \* 2 + **it**.ex**}**.*average*() // средний балл итого  
  
 val avgPercentLabWorks = groupStudents.*map* **{ it**.labWorks.*sum*().toDouble() / (**it**.labWorks.size \* 5) \* 100 **}**.*average*() // процент ЛР  
 val avgPercentExams = groupStudents.*map* **{ it**.exams.*sum*().toDouble() / (**it**.exams.size \* 5) \* 100 **}**.*average*() // процент ЭП  
 val avgPercentTests = groupStudents.*map* **{ it**.tests.*sum*().toDouble() \* 2 / (**it**.tests.size \* 10) \* 100 **}**.*average*() // процент КР  
 val avgPercentAccess = groupStudents.*map* **{** (**it**.attendanceGrade() + **it**.labWorks.*sum*() + **it**.tests.*sum*() \* 2).toDouble() / 75 \* 100 **}**.*average*() // процент допуск  
 val avgPercentTotal = groupStudents.*map* **{ it**.attendanceGrade() + **it**.labWorks.*sum*() + **it**.tests.*sum*() \* 2 + **it**.ex **}**.*average*() // процент итого  
  
 GroupStats(avgAttendance, avgLabWorks, avgExams, avgTests, commonAttest,  
 avgCommonLabWorks, avgCommonExams, avgCommonTests, avgCommonAccess, avgCommonTotal,  
 avgPercentLabWorks, avgPercentExams, avgPercentTests, avgPercentAccess, avgPercentTotal)  
 **}** // Таблица групп  
 *println*("${"Группа".*padEnd*(6)} ${"ЛР1".*padEnd*(3)} ${"ЛР2".*padEnd*(3)} ${"ЛР3".*padEnd*(3)} " +  
 "${"ЛР4".*padEnd*(3)} ${"ЛР5".*padEnd*(3)} ${"ЛР6".*padEnd*(3)} ${"ЛР7".*padEnd*(3)} " +  
 "${"ЭП1".*padEnd*(3)} ${"ЭП2".*padEnd*(3)} ${"ЭП3".*padEnd*(3)} ${"ЭП4".*padEnd*(3)} ${"ЭП5".*padEnd*(3)} " +  
 "${"КР1".*padEnd*(3)} ${"КР2".*padEnd*(3)} ${"КР3".*padEnd*(3)} " +  
 "${"Пос".*padEnd*(4)} ${"ЛР".*padEnd*(4)} ${"ЭП".*padEnd*(4)} ${"КР".*padEnd*(4)} " +  
 "${"Допуск".*padEnd*(6)} ${"Итого".*padEnd*(5)} ${"Ат".*padEnd*(2)} ${"Оценка".*padEnd*(8)} " +  
 "${"Пос".*padEnd*(4)} ${"ЛР".*padEnd*(4)} ${"ЭП".*padEnd*(4)} ${"КР".*padEnd*(4)} " +  
 "${"Допуск".*padEnd*(6)} ${"Итого".*padEnd*(5)}")  
 results.*forEach* **{** (group, stats) **->** *print*("${group.*padEnd*(6)} ")  
 for (i in 0..6)  
 *print*(*printFormatted*(stats.avgLabWorks[i]))  
 for (i in 0..4)  
 *print*(*printFormatted*(stats.avgExams[i]))  
 for (i in 0..2)  
 *print*(*printFormatted*(stats.avgTests[i]))  
 *print*("${String.*format*("%.1f", stats.avgAttendance).*padEnd*(4)} " +  
 "${String.*format*("%.1f", stats.avgCommonLabWorks).*padEnd*(4)} " +  
 "${String.*format*("%.1f", stats.avgCommonExams).*padEnd*(4)} " +  
 "${String.*format*("%.1f", stats.avgCommonTests).*padEnd*(4)} " +  
 "${String.*format*("%.1f", stats.avgCommonAccess).*padEnd*(6)} " +  
 "${String.*format*("%.1f", stats.avgCommonTotal).*padEnd*(5)} " +  
 "${stats.commonAttest.*padEnd*(2)} ")  
 *print*((when {  
 stats.avgCommonAccess < 30 -> "недопуск"  
 stats.avgCommonTotal < 60 -> "допуск"  
 stats.avgCommonTotal < 75 -> "САМОКАТ"  
 stats.avgCommonTotal < 90 -> "ХОРОШО"  
 else -> "ОТЛИЧНО"  
 }).*padEnd*(8))  
 *print*(" ${(String.*format*("%.0f", stats.avgAttendance \* 10) + "%").*padEnd*(4)} " +  
 "${(String.*format*("%.0f", stats.avgPercentLabWorks) + "%").*padEnd*(4)} " +  
 "${(String.*format*("%.0f", stats.avgPercentExams) + "%").*padEnd*(4)} " +  
 "${(String.*format*("%.0f", stats.avgPercentTests) + "%").*padEnd*(4)} " +  
 "${(String.*format*("%.0f", stats.avgPercentAccess) + "%").*padEnd*(6)} " +  
 "${(String.*format*("%.0f", stats.avgPercentTotal) + "%").*padEnd*(4)} ")  
 *println*()  
 **}** // Средние значения по каждому столбцу  
 *print*("3 курс ")  
 for (i in 0..6)  
 *print*(*printFormatted*(results.*map* **{** (\_, stats) **->** stats.avgLabWorks[i] **}**.*average*()))  
 for (i in 0..4)  
 *print*(*printFormatted*(results.*map* **{** (\_, stats) **->** stats.avgExams[i] **}**.*average*()))  
 for (i in 0..2)  
 *print*(*printFormatted*(results.*map* **{** (\_, stats) **->** stats.avgTests[i] **}**.*average*()))  
 *print*("${String.*format*("%.1f", results.*map* **{** (\_, stats) **->** stats.avgAttendance **}**.*average*()).*padEnd*(4)} " +  
 "${String.*format*("%.1f", results.*map* **{** (\_, stats) **->** stats.avgCommonLabWorks **}**.*average*()).*padEnd*(4)} " +  
 "${String.*format*("%.1f", results.*map* **{** (\_, stats) **->** stats.avgCommonExams **}**.*average*()).*padEnd*(4)} " +  
 "${String.*format*("%.1f", results.*map* **{** (\_, stats) **->** stats.avgCommonTests **}**.*average*()).*padEnd*(4)} " +  
 "${String.*format*("%.1f", results.*map* **{** (\_, stats) **->** stats.avgCommonAccess **}**.*average*()).*padEnd*(6)} " +  
 "${String.*format*("%.1f", results.*map* **{** (\_, stats) **->** stats.avgCommonTotal **}**.*average*()).*padEnd*(5)} " +  
 "ат ")  
 *print*((when {  
 results.*map* **{** (\_, stats) **->** stats.avgCommonAccess **}**.*average*() <= 75 \* 0.4 -> "недопуск"  
 results.*map* **{** (\_, stats) **->** stats.avgCommonTotal **}**.*average*() < 60 -> "допуск"  
 results.*map* **{** (\_, stats) **->** stats.avgCommonTotal **}**.*average*() < 75 -> "САМОКАТ"  
 results.*map* **{** (\_, stats) **->** stats.avgCommonTotal **}**.*average*() < 90 -> "ХОРОШО"  
 else -> "ОТЛИЧНО"  
 }).*padEnd*(8))  
 *print*(" ${(String.*format*("%.0f", results.*map* **{** (\_, stats) **->** stats.avgAttendance \* 10 **}**.*average*()) + "%").*padEnd*(4)} " +  
 "${(String.*format*("%.0f", results.*map* **{** (\_, stats) **->** stats.avgPercentLabWorks **}**.*average*()) + "%").*padEnd*(4)} " +  
 "${(String.*format*("%.0f", results.*map* **{** (\_, stats) **->** stats.avgPercentExams **}**.*average*()) + "%").*padEnd*(4)} " +  
 "${(String.*format*("%.0f", results.*map* **{** (\_, stats) **->** stats.avgPercentTests **}**.*average*()) + "%").*padEnd*(4)} " +  
 "${(String.*format*("%.0f", results.*map* **{** (\_, stats) **->** stats.avgPercentAccess **}**.*average*()) + "%").*padEnd*(6)} " +  
 "${(String.*format*("%.0f", results.*map* **{** (\_, stats) **->** stats.avgPercentTotal **}**.*average*()) + "%").*padEnd*(5)} ")  
 */\*\*  
 \* 5 Найти группу, в которой наименьший средний балл на допуск среди тех, кто получил  
 оценки 3, 4, 5  
 \*/* val autoExam = students.*filter* **{ it**.evalPreGrade() in *listOf*("ОТЛИЧНО", "ХОРОШО", "САМОКАТ") **}** val groupByGroupWithEx = autoExam.*groupBy* **{ it**.group **}** val avgInGroup = groupByGroupWithEx.*mapValues* **{** (\_, group) **->** group.*map* **{ it**.getTotalScore().toDouble() / (**it**.attendanceGrade() + **it**.labWorks.size \* 5 + **it**.tests.size \* 10) \* 100 **}**.*average*() **}** *println*("В группе ${avgInGroup.*minByOrNull* **{ it**.value **}**?.key} наименьший средний балл на допуск среди тех, кто получил 3, 4 или 5")  
}

Объявление класса Student, который хранит информацию о студенте: группа, ФИО, дополнительные баллы, посещаемость лекций и лабораторных занятий, оценки за лабораторные работы, экзаменационные проекты, контрольные работы и аттестацию.

Данные передаются через первичный конструктор

class Student(val group: String, // группа

val name: String, // ФИО

val ex: Int, // доп баллы

val lecAttendance: List<Boolean>, // посещаемость лекций

val labAttendance: List<Boolean>, // посещаемость лаб

val labWorks: List<Int>, // оценки за ЛР

val exams: List<Int>, // оценки за ЭП

val tests: List<Int>, // оценки за КР

val attest: String) { // аттестация

Давайте разберем функцию attendanceGrade подробно. Эта функция вычисляет оценку студента за посещаемость лекций и лабораторных занятий.

kotlin

Копировать код

fun attendanceGrade(): Int {

* Определяем функцию attendanceGrade, которая возвращает целое число (Int), представляющее оценку за посещаемость.

kotlin

Копировать код

val lecTrueCount = lecAttendance.count { it }

val labTrueCount = labAttendance.count { it }

* lecTrueCount и labTrueCount вычисляют количество true значений в списках lecAttendance и labAttendance, соответственно.
  + lecAttendance.count { it } считает количество посещений лекций.
  + labAttendance.count { it } считает количество посещений лабораторных.

kotlin

Копировать код

val attendScore = lecTrueCount.toDouble() / lecAttendance.size \* 5 + labTrueCount.toDouble() / labAttendance.size \* 5

* attendScore вычисляет общий балл за посещаемость.
  + lecTrueCount.toDouble() / lecAttendance.size вычисляет долю посещенных лекций.
  + labTrueCount.toDouble() / labAttendance.size вычисляет долю посещенных лабораторных.
  + Обе доли умножаются на 5, чтобы получить оценку за лекции и лабораторные, и затем складываются.

kotlin

Копировать код

return when {

attendScore < 4 -> 0

attendScore < 6 -> 4

attendScore < 7.5 -> 6

attendScore < 9 -> 8

else -> 10

}

}

* Возвращаем оценку на основе значения attendScore с использованием конструкции when.
  + Если attendScore меньше 4, возвращаем 0.
  + Если attendScore меньше 6, возвращаем 4.
  + Если attendScore меньше 7.5, возвращаем 6.
  + Если attendScore меньше 9, возвращаем 8.
  + Во всех остальных случаях возвращаем 10.

**Пример**

Предположим, что у нас есть следующие данные:

* lecAttendance = listOf(true, true, false, true, false)
* labAttendance = listOf(true, false, true, true)

Тогда:

1. lecTrueCount = 3 (3 посещения лекций)
2. labTrueCount = 3 (3 посещения лабораторных)
3. attendScore = 3.0 / 5 \* 5 + 3.0 / 4 \* 5 = 3 + 3.75 = 6.75

Поскольку 6.75 находится в диапазоне < 7.5, функция вернет 6.

Эта функция averageIntList принимает список списков целых чисел и возвращает список средних значений для каждого индекса во вложенных списках. Давайте разберем её построчно:

kotlin

Копировать код

fun averageIntList(lists: List<List<Int>>): List<Double> {

* Объявляется функция averageIntList, которая принимает параметр lists типа List<List<Int>> (список списков целых чисел) и возвращает List<Double> (список дробных чисел).

kotlin

Копировать код

if (lists.isEmpty()) return emptyList()

* Если входной список lists пуст, функция сразу возвращает пустой список emptyList().

kotlin

Копировать код

val size = lists.first().size

* Определяется переменная size, которая принимает значение размера первого списка во входном списке списков. Предполагается, что все внутренние списки имеют одинаковый размер.

kotlin

Копировать код

return List(size) { index ->

* Возвращается новый список, размер которого равен size. Для каждого индекса от 0 до size-1 выполняется блок кода:

kotlin

Копировать код

lists.map { it[index] }.average()

* Для каждого индекса создается новый список, состоящий из элементов на этой позиции во всех вложенных списках (lists.map { it[index] }).
* Затем для этого списка вычисляется среднее значение (.average()). Среднее значение является результатом для текущего индекса.

Полная функция:

fun averageIntList(lists: List<List<Int>>): List<Double> {

if (lists.isEmpty()) return emptyList()

val size = lists.first().size

return List(size) { index ->

lists.map { it[index] }.average()

}

}

**Пример**

Рассмотрим пример с данными:

kotlin

Копировать код

val data = listOf(

listOf(1, 2, 3),

listOf(4, 5, 6),

listOf(7, 8, 9)

)

1. size будет равен 3 (размер первого вложенного списка).
2. Создается новый список из 3 элементов, где для каждого индекса выполняется вычисление среднего значения:
   * Для index = 0: список [1, 4, 7], среднее = (1 + 4 + 7) / 3 = 4.0
   * Для index = 1: список [2, 5, 8], среднее = (2 + 5 + 8) / 3 = 5.0
   * Для index = 2: список [3, 6, 9], среднее = (3 + 6 + 9) / 3 = 6.0

Таким образом, результат будет:

kotlin

Копировать код

val result = averageIntList(data)

// result: [4.0, 5.0, 6.0]

**Заключение**

Функция averageIntList вычисляет и возвращает список средних значений для каждого индекса из входного списка списков целых чисел. Она полезна для обработки данных, где необходимо вычислить средние значения по каждому столбцу в матрице чисел.

Конечно, давайте разберем эту строку максимально подробно:

kotlin

Копировать код

return List(size) { index -> lists.map { it[index] }.average() }

**Контекст**

Эта строка кода встречается в функции averageIntList, которая принимает список списков целых чисел и возвращает список, содержащий средние значения для каждого индекса из этих списков.

**Разбор строки**

1. **List(size) { index -> ... }**:
   * Это создание нового списка типа List<Double> с заданным размером size.
   * **{ index -> ... }**: Это лямбда-выражение, которое будет вызываться для каждого индекса списка от 0 до size - 1.
     + **index**: Переменная, представляющая текущий индекс списка.
2. **lists.map { it[index] }.average()**:
   * Это вычисление среднего значения для каждого индекса во вложенных списках.
     + **lists**: Это список списков целых чисел, переданный в функцию.
     + **it[index]**: Для каждого вложенного списка it берется элемент с индексом index.
     + **.map { it[index] }**: Создает список, содержащий элементы с индексом index из каждого вложенного списка.
     + **.average()**: Вычисляет среднее значение элементов в полученном списке.

**Итоговое значение**

Этот фрагмент кода создает список, где каждый элемент представляет собой среднее значение элементов с одинаковым индексом из всех вложенных списков. Таким образом, каждый элемент нового списка - это среднее значение столбца из списков lists.

**Пример**

Предположим, у нас есть следующий список списков целых чисел:

kotlin

Копировать код

val lists = listOf(

listOf(1, 2, 3),

listOf(4, 5, 6),

listOf(7, 8, 9)

)

Если мы вызовем функцию averageIntList(lists), то она вернет список средних значений для каждого столбца:

kotlin

Копировать код

val result = averageIntList(lists)

// result будет равен [4.0, 5.0, 6.0], так как средние значения для каждого столбца: (1+4+7)/3=4.0, (2+5+8)/3=5.0, (3+6+9)/3=6.0

fun mostCommonAttest(attests: List<String>): String {

return attests.groupBy { it }.maxByOrNull { it.value.size }?.key ?: ""

}

**Назначение функции**

Цель этой функции - определить наиболее часто встречающийся элемент в списке строк.

**Параметры функции**

* **attests: List<String>**: Это список строк, в котором нужно найти наиболее часто встречающийся элемент.

**Шаги функции**

1. **Группировка элементов по их значению**:

kotlin

Копировать код

attests.groupBy { it }

* + Эта строка использует функцию groupBy, чтобы сгруппировать элементы списка attests по их значению. Таким образом, каждый уникальный элемент будет ключом в результирующем словаре, а значениями будут списки строк с одинаковым значением.

1. **Определение наиболее часто встречающегося элемента**:

kotlin

Копировать код

.maxByOrNull { it.value.size }

* + После группировки мы используем maxByOrNull, чтобы найти элемент с наибольшим количеством значений (строк) в списке.
  + **it** здесь представляет пару ключ-значение в словаре. it.value.size возвращает размер списка (значений), соответствующего ключу.
  + maxByOrNull возвращает элемент с максимальным значением, основываясь на результате функции, переданной в качестве аргумента. В данном случае, это размер списка значений.

1. **Возвращение наиболее часто встречающегося элемента**:

kotlin

Копировать код

?.key ?: ""

* + Если список attests не пустой и был найден наиболее часто встречающийся элемент, то мы возвращаем этот элемент (ключ словаря). В противном случае, мы возвращаем пустую строку.
  + ?.key извлекает ключ из результата операции maxByOrNull. Если результат равен null (например, если attests пустой список), оператор Elvis ?: возвращает пустую строку.

**Итог**

Эта функция принимает список строк и возвращает наиболее часто встречающийся элемент в этом списке.

// Получение оценки из ячейки. Если там число, то возвращаем его, если строка, то преобразуем в баллы

fun getCellGrade(cell: Cell): Int{

return when (cell.cellType) {

CellType.STRING -> {

val stringValue = cell.stringCellValue

when (stringValue) {

"д" -> 2

"", "-" -> 0

else -> stringValue.toIntOrNull() ?: 0 // Попробуем преобразовать строку в число, если это не удастся, то 0

}

}

else -> cell.numericCellValue.toInt()

}

}

**Назначение функции**

Эта функция используется для извлечения оценки из ячейки таблицы Excel. Если значение ячейки является числом, функция возвращает это число. Если значение ячейки является строкой, функция пытается преобразовать эту строку в числовое значение в соответствии с определенными правилами.

**Параметры функции**

* **cell: Cell**: Это объект ячейки (из библиотеки Apache POI), из которой нужно извлечь оценку.

**Шаги функции**

1. **Определение типа ячейки**:

kotlin

Копировать код

when (cell.cellType) {

* + Этот оператор when проверяет тип ячейки.

1. **Обработка случая, когда ячейка содержит строку**:

kotlin

Копировать код

CellType.STRING -> {

val stringValue = cell.stringCellValue

when (stringValue) {

"д" -> 2

"", "-" -> 0

else -> stringValue.toIntOrNull() ?: 0

}

}

* + Если тип ячейки - строка, мы извлекаем строковое значение из ячейки (cell.stringCellValue).
  + Затем используется оператор when для определения соответствующего числового значения в зависимости от содержимого строки:
    - Если значение равно "д", функция возвращает 2.
    - Если значение пустое ("") или содержит "-" (означает отсутствие оценки), функция возвращает 0.
    - В остальных случаях мы пытаемся преобразовать строку в число с помощью toIntOrNull(). Если преобразование не удалось (например, если строка содержит нечисловое значение), возвращается 0.

1. **Обработка случая, когда ячейка содержит число**:

kotlin

Копировать код

else -> cell.numericCellValue.toInt()

* + Если тип ячейки не является строкой, предполагается, что это числовое значение, и оно извлекается с помощью cell.numericCellValue. Затем это значение преобразуется в целое число с помощью метода toInt().

**Итог**

Эта функция обрабатывает различные типы ячеек Excel и возвращает соответствующие оценки: числовые значения извлекаются напрямую, а строки интерпретируются в соответствии с определенными правилами (например, "д" соответствует оценке 2, отсутствующие оценки - 0, а любые другие значения строк преобразуются в числа, если это возможно, или считаются нулем).

Хорошо, давайте разберем эту функцию подробно:

kotlin

Копировать код

// Чтение файла

fun fileRead(): List<Student> {

val students = mutableListOf<Student>()

val file = File("students.xlsx")

val workbook = WorkbookFactory.create(file.inputStream())

// Проход по всем листам

for (sheetNum in 0 until workbook.numberOfSheets) {

val sheet = workbook.getSheetAt(sheetNum)

var rowC = 0

// Выбор листов групп

if (("36" in sheet.sheetName) or ("39" in sheet.sheetName))

for (row in sheet) {

// Пропуск двух строк

if (rowC > 1)

if (row.getCell(0) != null) {

val name = row.getCell(1).stringCellValue // получение ФИО

val ex = row.getCell(2).numericCellValue.toInt() // получение доп баллов

val lecAttendanceList = mutableListOf<Boolean>()

val labAttendanceList = mutableListOf<Boolean>()

var cellIndex = 4

// Чтение данных о посещаемости отдельно для лекций и для лаб

var leclab = true

while (sheet.getRow(0).getCell(cellIndex).cellType != CellType.STRING) {

// переключение счёта лекций и лаб

if (leclab and (sheet.getRow(0).getCell(cellIndex).numericCellValue == 1.0))

leclab = false

else if (!leclab and (sheet.getRow(0).getCell(cellIndex).numericCellValue == 1.0))

leclab = true

var value = ""

if (row.getCell(cellIndex) != null)

value = row.getCell(cellIndex).stringCellValue

if (leclab)

lecAttendanceList.add(value == "+")

else

labAttendanceList.add(value == "+")

cellIndex += 1

}

// Чтение оценок за лабораторные работы

val labWorks = mutableListOf<Int>()

for (i in cellIndex..(cellIndex + 6)) { // Колонки с ЛР1 по ЛР7

val value = getCellGrade(row.getCell(i))

labWorks.add(value)

}

cellIndex += 7

// Чтение оценок за экзаменационные работы

val exams = mutableListOf<Int>()

for (i in cellIndex..(cellIndex + 4)) { // Колонки с ЭП1 по ЭП5

val value = getCellGrade(row.getCell(i))

exams.add(value)

}

cellIndex += 5

// Чтение оценок за контрольные работы

val tests = mutableListOf<Int>()

for (i in cellIndex..(cellIndex + 2)) { // Колонки с КР1 по КР3

val value = getCellGrade(row.getCell(i))

tests.add(value)

}

cellIndex += 32

// Создание объекта Student и добавление его в список

val student = Student(sheet.sheetName, name, ex, lecAttendanceList, labAttendanceList, labWorks, exams, tests, row.getCell(cellIndex).stringCellValue)

students.add(student)

} else break

rowC += 1

}

}

workbook.close()

return students

}

**Назначение функции**

Эта функция читает данные из файла Excel и возвращает список объектов Student, содержащих информацию о студентах.

**Шаги функции**

1. **Открытие файла**:

kotlin

Копировать код

val file = File("students.xlsx")

val workbook = WorkbookFactory.create(file.inputStream())

* + Файл Excel считывается с диска.

1. **Итерация по листам в файле**:

kotlin

Копировать код

for (sheetNum in 0 until workbook.numberOfSheets) {

val sheet = workbook.getSheetAt(sheetNum)

* + Функция перебирает все листы в файле Excel.

1. **Выбор нужных листов**:

kotlin

Копировать код

if (("36" in sheet.sheetName) or ("39" in sheet.sheetName))

* + Эта часть кода фильтрует листы по их названию, оставляя только те, которые содержат "36" или "39" в названии.

1. **Чтение данных из каждой строки**:
   * Каждая строка представляет одного студента.
   * Для каждой строки выполняются следующие шаги:
     + Пропуск первых двух строк (предполагается, что первые две строки содержат заголовки или другую мета-информацию).
     + Извлечение данных о студенте, таких как имя, дополнительные баллы и прочее.
     + Чтение данных о посещаемости лекций и лабораторных работ.
     + Чтение оценок за лабораторные, экзаменационные и контрольные работы.
     + Создание объекта Student и добавление его в список students.
2. **Закрытие рабочей книги**:

kotlin

Копировать код

workbook.close()

* + После завершения чтения данных из файла рабочая книга закрывается для освобождения ресурсов.

1. **Возвращение списка студентов**:

kotlin

Копировать код

return students

* + Возвращается список объектов Student, содержащих информацию о студентах.

Эта функция printFormatted используется для форматированного вывода значений в таблицу. Вот как она работает:

1. Принимает на вход числовое значение типа Double.
2. Использует конструкцию when для определения формата вывода в зависимости от значения value.
3. Если value меньше 0.5, возвращается строка "-", выровненная по ширине в 4 символа пробелами справа (padEnd(4)).
4. Если value меньше 1, возвращается строка "+", также выровненная по ширине в 4 символа пробелами справа.
5. Если value меньше 1.5, возвращается строка "++", также выровненная по ширине в 4 символа пробелами справа.
6. Если value меньше 2.5, возвращается строка "д", также выровненная по ширине в 4 символа пробелами справа.
7. Иначе (если value больше или равно 2.5), значение форматируется с одной десятичной знаком с помощью String.format("%.1f", value), затем полученная строка выравнивается по ширине в 3 символа пробелами справа.

Эта функция применяется, вероятно, для вывода оценок в таблицу в заданном формате для лучшей читаемости.

Эта функция main() выполняет следующие действия:

1. **Чтение данных из файла:**

kotlin

Копировать код

val students = fileRead()

* + Считывает данные о студентах из файла Excel и сохраняет их в переменной students.

1. **Анализ результатов обучения:**

kotlin

Копировать код

val stoopid = students.filter { it.getTotalScore() < 30 }

var max = 0

var min = 60

var maxL = 1

var minK = 1

* + Создает список студентов, получивших общий балл менее 30.
  + Инициализирует переменные для нахождения наиболее и наименее популярных лабораторных работ и контрольных работ.

1. **Анализ лабораторных работ:**

kotlin

Копировать код

for (i in 0..6) {

// Цикл по всем лабораторным работам (от 1 до 7)

var count = 0

stoopid.forEach {

// Подсчитывает количество студентов, сдавших каждую лабораторную работу

count += if (it.labWorks.get(i) > 0) 1 else 0

}

// Находит лабораторную работу, которую сдало наибольшее количество студентов

if (count > max) {

max = count

maxL = i + 1

}

}

1. **Анализ контрольных работ:**

kotlin

Копировать код

for (i in 0..2) {

// Цикл по всем контрольным работам (от 1 до 3)

var count = 0

stoopid.forEach {

// Подсчитывает количество студентов, сдавших каждую контрольную работу

count += if (it.tests.get(i) > 0) 1 else 0

}

// Находит контрольную работу, которую сдало наименьшее количество студентов

if (count < min) {

min = count

minK = i + 1

}

}

1. **Вывод результатов:**

kotlin

Копировать код

println("Лабораторная работа №$maxL была сдана наибольшим количеством студентов без допуска")

println("Контрольная работа №$minK была сдана наименьшим количеством студентов без допуска\n")

* + Выводит информацию о лабораторной и контрольной работах, сданных наибольшим и наименьшим количеством студентов без допуска.

Этот код анализирует данные о студентах и выводит информацию о популярности различных видов работ и результаты анализа.

Этот блок кода выполняет следующие действия:

1. **Перебор студентов:**

kotlin

Копировать код

students.forEach {

* + Проходит по каждому студенту в списке students.

1. **Вывод информации:**

kotlin

Копировать код

println("Студент: ${it.name}\n" +

"Доп баллы: ${it.ex}\n" +

"Посещаемость: ${it.attendanceGrade()}\n" +

"Оценки за:\n" +

"\tЛабораторные работы: ${it.labWorks}\tВсего: ${it.labWorks.sum()}\n" +

"\tЭкзаменационный проект: ${it.exams}\tВсего: ${it.exams.sum()}\n" +

"\tКонтрольные работы: ${it.tests}\tВсего: ${it.tests.sum() \* 2}\n" +

"Итого баллов: ${it.getTotalScore() + it.ex + it.exams.sum()}\n")

* + Выводит информацию о каждом студенте:
    - Имя студента.
    - Дополнительные баллы (доп баллы).
    - Баллы за посещаемость (рассчитываются с помощью метода attendanceGrade()).
    - Баллы за лабораторные работы, их общее количество и оценки.
    - Баллы за экзаменационный проект, его общее количество и оценки.
    - Баллы за контрольные работы, их общее количество и оценки.
    - Итоговые баллы, рассчитанные с учетом общего балла, дополнительных баллов и баллов за экзамены.

Этот блок кода выводит подробную информацию о каждом студенте, включая его общий балл и баллы за различные виды работ.

Этот блок кода выполняет следующие действия:

1. **Формирование рейтинга:**

kotlin

Копировать код

val rating = students.sortedBy { it.name }.sortedByDescending { it.getTotalScore() + it.ex }

* + Формирует список студентов (rating), отсортированный по имени (в алфавитном порядке) и общему баллу в убывающем порядке.

1. **Фильтрация студентов по допуску:**

kotlin

Копировать код

val studOnlyAccess = rating.filter { it.evalPreGrade() == "допуск" }.takeLast(5)

val studNoAccess = rating.filter { it.evalPreGrade() == "недопуск" }.take(5)

* + Создает два списка студентов: studOnlyAccess - студенты с допуском и studNoAccess - студенты без допуска. Они фильтруются из списка rating.

1. **Вывод информации:**
   * Выводит информацию о студентах с допуском и без допуска.

kotlin

Копировать код

println("№ ${"Группа".padEnd(6)} ${"Фамилия И О".padEnd(20)} ${"ат".padEnd(2)} " +

"${"Экзамен".padEnd(8)} ${"Пос".padEnd(3)} " +

"${"ЛР".padEnd(5)} " +

"${"ЭП".padEnd(5)} " +

"${"КР".padEnd(5)} " +

"${"Допуск".padEnd(6)} " +

"${"Итого".padEnd(5)}")

println()

studOnlyAccess.forEachIndexed { index, it -> println("${(index + 1).toString().padEnd(2)} ${it.group.padEnd(6)} ${it.name.padEnd(20)} ${it.attest.padEnd(2)} " +

"${it.evalPreGrade().padEnd(8)} ${(it.attendanceGrade() \* 10).toString().padEnd(3)} " +

"${String.format("%.1f", it.labWorks.sum().toDouble() / (it.labWorks.size \* 5) \* 100).padEnd(5)} " +

"${String.format("%.1f", it.exams.sum().toDouble() / (it.exams.size \* 5) \* 100).padEnd(5)} " +

"${String.format("%.1f", it.tests.sum().toDouble() / (it.tests.size \* 5) \* 100).padEnd(5)} " +

"${String.format("%.1f", it.getTotalScore().toDouble() / (it.attendanceGrade() + it.labWorks.size \* 5 + it.tests.size \* 10) \* 100).padEnd(6)} " +

"${(it.getTotalScore() + it.ex + it.exams.sum()).toString().padEnd(5)}") }

println()

studNoAccess.forEachIndexed { index, it -> println("${(index + 1).toString().padEnd(2)} ${it.group.padEnd(6)} ${it.name.padEnd(20)} ${it.attest.padEnd(2)} " +

"${it.evalPreGrade().padEnd(10)} ${(it.attendanceGrade() \* 10).toString().padEnd(3)} " +

"${String.format("%.1f", it.labWorks.sum().toDouble() / (it.labWorks.size \* 5) \* 100).padEnd(5)} " +

"${String.format("%.1f", it.exams.sum().toDouble() / (it.exams.size \* 5) \* 100).padEnd(5)} " +

"${String.format("%.1f", it.tests.sum().toDouble() / (it.tests.size \* 5) \* 100).padEnd(5)} " +

"${String.format("%.1f", it.getTotalScore().toDouble() / (it.attendanceGrade() + it.labWorks.size \* 5 + it.tests.size \* 10) \* 100).padEnd(6)} " +

"${(it.getTotalScore() + it.ex + it.exams.sum()).toString().padEnd(5)}") }

* + Выводит информацию о студентах, отсортированных по допуску и не допуску. Выводятся номер, группа, имя, аттестация, оценки за экзамен и посещаемость, баллы за лабораторные, экзаменационные и контрольные работы, общий балл и допуск.

Эта часть кода использует лямбда-выражение в методе forEachIndexed, который принимает два параметра: индекс элемента списка и сам элемент.

Вот как это связано с лямбда-выражениями:

1. **forEachIndexed**: Это функция высшего порядка, которая принимает лямбда-выражение в качестве аргумента. Лямбда-выражение определяет, что будет выполняться для каждого элемента списка.
2. **Лямбда-выражение**: В данном случае, это анонимная функция, которая принимает два аргумента: index и it (элемент списка). В теле этой функции указан код, который будет выполнен для каждого элемента списка.
3. **Вывод информации о студентах**: Внутри лямбда-выражения на каждой итерации выполняется код, который выводит информацию о текущем студенте с использованием println(). Лямбда-выражение дает доступ к текущему элементу и его индексу, что позволяет легко работать с каждым элементом списка.

Использование лямбда-выражений в методе forEachIndexed делает код более компактным и позволяет лаконично описывать действия, которые должны выполняться для каждого элемента списка.

Этот код выполняет следующие действия:

1. **Группировка студентов по группам**:
   * groupByGroup группирует студентов по номеру группы.
2. **Вычисление статистики для каждой группы**:
   * Для каждой группы студентов вычисляются следующие статистические параметры:
     + Средняя посещаемость (avgAttendance).
     + Средний балл за лабораторные работы (avgLabWorks).
     + Средний балл за экзаменационные проекты (avgExams).
     + Средний балл за контрольные работы (avgTests).
     + Самый часто встречающийся тип аттестации в группе (commonAttest).
     + Средний балл за все лабораторные работы (avgCommonLabWorks).
     + Средний балл за все экзаменационные проекты (avgCommonExams).
     + Средний балл за все контрольные работы (avgCommonTests).
     + Средний балл на допуск (avgCommonAccess).
     + Средний итоговый балл (avgCommonTotal).
     + Процент выполнения лабораторных работ от максимального (avgPercentLabWorks).
     + Процент выполнения экзаменационных проектов от максимального (avgPercentExams).
     + Процент выполнения контрольных работ от максимального (avgPercentTests).
     + Процент получения допуска от максимального (avgPercentAccess).
     + Процент итогового балла от максимального (avgPercentTotal).
3. **Вывод результатов**:
   * Для каждой группы выводится таблица, содержащая вычисленные статистические параметры. Каждый столбец таблицы представляет собой конкретный параметр статистики (например, средний балл за лабораторные работы).
   * Каждая строка таблицы представляет собой одну группу студентов.
   * Выводится также среднее значение каждого параметра для всех групп.
   * Оценки, определенные на основе среднего итогового балла, такие как "допуск", "недопуск", "САМОКАТ", "ХОРОШО" или "ОТЛИЧНО", также выводятся для каждой группы.

Этот код выполняет следующие действия:

1. **Отбор студентов, получивших оценки 3, 4 или 5**:
   * Сначала создается список autoExam, содержащий только тех студентов, которые получили оценки "ОТЛИЧНО", "ХОРОШО" или "САМОКАТ".
2. **Группировка студентов по группам**:
   * Далее происходит группировка отфильтрованных студентов по номеру группы в списке groupByGroupWithEx.
3. **Вычисление среднего балла на допуск для каждой группы**:
   * Для каждой группы студентов вычисляется средний балл на допуск (avgInGroup). Для этого берется средний балл каждого студента, но только тех, кто получил оценки "ОТЛИЧНО", "ХОРОШО" или "САМОКАТ", и делят его на сумму баллов за посещаемость, лабораторные работы и контрольные работы.
4. **Нахождение группы с наименьшим средним баллом на допуск**:
   * Затем из полученных средних баллов находится минимальное значение с помощью minByOrNull.
   * Выводится информация о группе с наименьшим средним баллом на допуск среди тех, кто получил оценки 3, 4 или 5.

Эта запись означает следующее:

* results.map { (\_, stats) -> stats.avgAttendance \* 10 } - это часть кода, которая преобразует коллекцию results (которая является картой, сопоставляющей каждой группе GroupStats) в список значений. Каждое значение в этом списке представляет среднюю посещаемость, умноженную на 10, для каждой группы студентов.
* .average() - после преобразования коллекции в список происходит вычисление среднего значения этого списка, то есть среднего значения средних посещаемостей, умноженных на 10, для всех групп.
* + "%").padEnd(4)} - здесь среднее значение посещаемости, умноженное на 10, преобразуется в процентное значение, добавляется символ процента %, а затем дополняется пробелами слева до 4 символов. Таким образом, это выражение форматирует среднюю посещаемость как процент с выравниванием по правому краю на 4 символа.

что значит (\_,stats)->

(\_, stats) -> в данном контексте означает деструктуризацию элемента пары, представленной в коллекции results.

* \_ - это здесь заполнитель для первого элемента пары, который в данном случае не используется. Обычно его используют, когда не нужно обращаться к первому элементу пары.
* stats - это второй элемент пары, который содержит объект типа GroupStats.

Таким образом, (\_, stats) позволяет получить доступ только ко второму элементу пары (объекту GroupStats), пропуская первый элемент, который в данном случае не используется.