Оглавление

Лабораторная работа №1. Введение в Kotlin

Kotlin – это статически типизированный объектно-ориентированный язык, совместимый с Java и предназначенный для промышленной разработки приложений. Поддерживает функциональный стиль программирования. Работает поверх JVM. Есть онлайн площадка для написания кода и запуска программ на Kotlin по адресу https://try.kotlinlang.org/. Онлайн документация по языку находится по адресу https://kotlinlang.org/docs/reference/.

Удобнее всего работать с Kotlin в среде программирования IntelliJ IDEA. Для этого после ее запуска выбираем Create New Project (если среда запускается в первый раз или предыдущий проект был закрыт командой Close Project из меню File) или в меню File выбираем New – Project… (рис. 1).

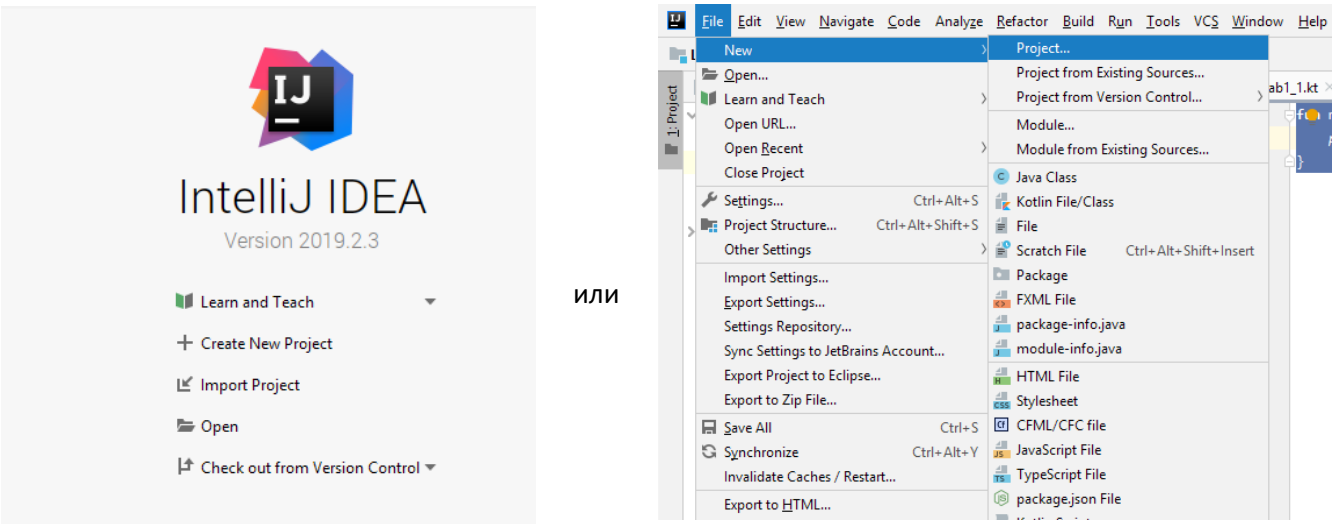


Рис. 1. Варианты создания Kotlin-проекта в IntelliJ IDEA.

Появится окно с вариантами проектов. Выбираем раздел Kotlin, в нем JVM | IDEA (рис. 2а). Нажимаем Next. Во втором окне вводим название проекта и жмем Finish (рис. 2б).

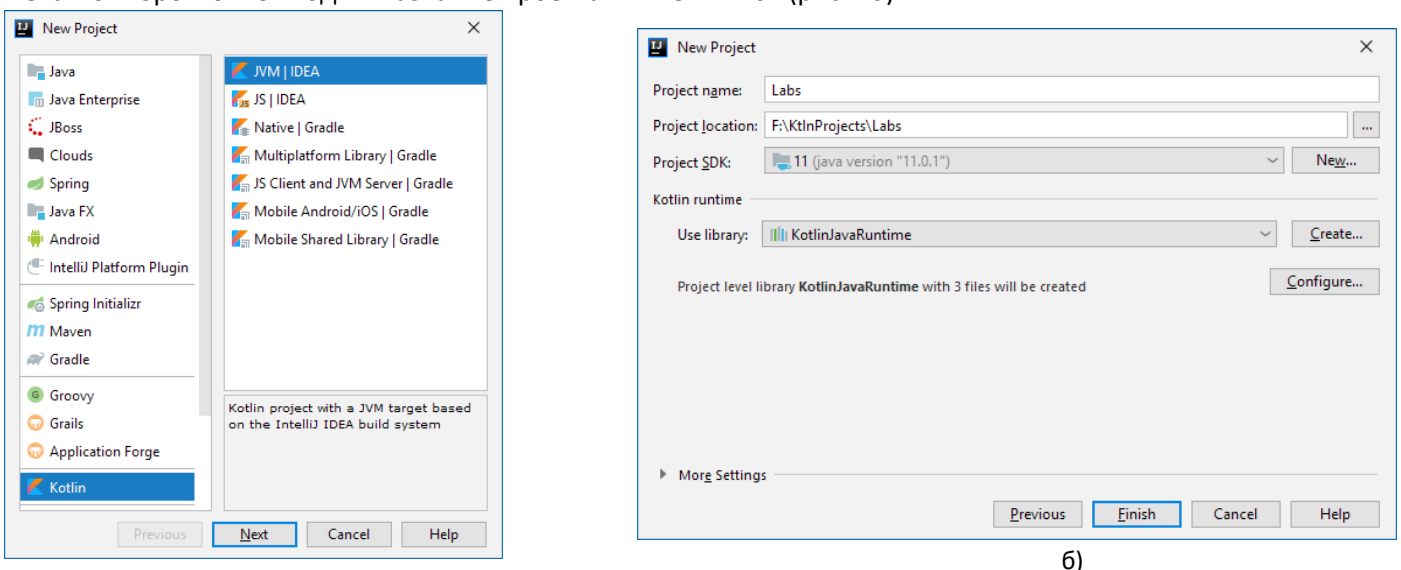


Рис. 2. Выбор проекта и задание его параметров.

В появившемся проекте в левой части будет структура папок и файлов проекта. Вызываем контекстное меню папки src и выбираем New –> Kotlin File/Class. В появившемся окне в поле Name вводим название файла (например, lab1_1) и жмем Enter (рис. 3).

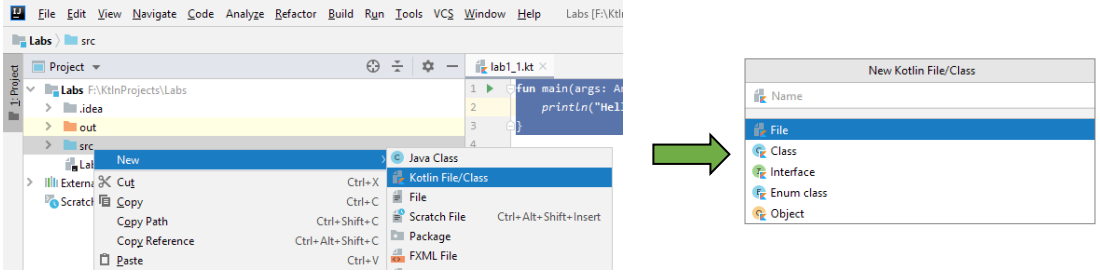


Рис. 3. Добавление файла в проект.

Созданный файл сразу появится в центральном окне.

Первая программа на Kotlin:

```
fun main() {
    println("Hello, world!")
}
```

При запуске в консоль выведется hello, world!

Разберем этот код:

- `fun` – зарезервированное слово для обозначения функции;
- `fun main()` – создание функции;
- `println("Hello, world!")` – вывод данных в консоль.

Именно функция main является запускной в программах на языке Kotlin.

Базовые типы данных в Kotlin:

Название типа	Размер (в битах)	Принимаемые значения
Byte	8	-128 .. 127
Short	16	-32768 .. 32767
Int	32	-2,147,483,648 (-2^{31}) .. 2,147,483,647 (2^{31} - 1)
Long	64	-9,223,372,036,854,775,808 (-2^{63}) .. 9,223,372,036,854,775,807 (2^{63} - 1)
Float	32	1.40129846432481707e-45 .. 3.40282346638528860e+38
Double	64	4.94065645841246544e-324 .. 1.79769313486231570e+308

Объявление переменных осуществляется с помощью ключевого слова ***var*** в любом месте программы, причем тип данных указывать не обязательно, если его можно выяснить из самого значения, но можно указать тип данных через двоеточие от переменной:

```
var x = 10;    //объявление целого
println("x=$x")
var y = 6.5    //объявление вещественного
println("y=$y")
var z = 5f     //объявление вещественного
println("z=$z")
var a:Int = 25        //объявление целого с явным указанием типа
println("a=$a")
```

В консоль будет выведено

x=10

y=6.5

z=5.0

a=25

В операторе println обозначение выводимой переменной происходит внутри кавычек после символа $. Также после этого символа можно в фигурных скобках указывать небольшие выражения, результат которых будет подставлен в выводимую строку.

```
var y = 6.5
var z = 5f
println("z+y=${z+y}")
```

В консоль будет выведено

z+y=11.5

Для объявления констант можно использовать ключевое слово ***val***:

```
val x = 10;    //неизменяемая величина
println("x=$x")
val y = 2
println("y=$y")
```

Такие переменные нельзя изменять после первого присвоения значения.

Строки в Kotlin объявляются как обычные переменные, значения им присваиваются в двойных кавычках. Есть тип String. Примеры объявлений строк:

```
var str = "Hello, world!"
println(str)
```

```
val str2: String = "Kotlin - это как Java, только Kotlin ☺."
println(str2)
```

Дополнительно есть специальный вид строк в тройных кавычках (так называемые «необработанные строки»). Пример:

```
val s = """
    О сколько нам открытий чудных
    Готовят просвещенья дух
    И опыт, сын ошибок трудных,
    И гений, парадоксов друг,
    И случай, бог изобретатель..."""
println(s)
```

В консоль будет выведено

О сколько нам открытий чудных
Готовят просвещенья дух
И опыт, сын ошибок трудных,
И гений, парадоксов друг,
И случай, бог изобретатель...

Для ввода данных с консоли можно использовать функцию readline(), она возвращает строку, и эту строку можно сразу перевести в нужный тип данных, при этом нужно использовать два восклицательных знака в конце этой функции. Пример ввода данных с клавиатуры:

```
print("Input a: ")
val a = readLine()!!.toInt()
print("Input b: ")
val b = readLine()!!.toInt()
println("a=$a b=$b")
```

В консоль будет выведено

Input a: 6
Input b: 3
a=6 b=3

Оператор !! делает возможным действия в условиях вероятного получения null, т.к. обычно Kotlin запрещает действия с объектом, если он может быть null. Это сделано для большей безопасности создаваемых приложений. В нашем случае readLine() может вернуть null, поэтому впрямую нельзя вызывать toInt(), но можно поставить !! и тогда компилятор разрешит преобразование. **Но пользоваться этим нужно осторожно, только в тех случаях, когда точно известно, что объект не null !!!**

Операция присваивания (=) и арифметические операции (+, –, *, /, %) заимствованы из языка C (C++/Java).

Оператор if также заимствован из C-подобных языков, но при этом еще может возвращать значение, как тернарный оператор ? : . Например:

```
val a = 5
val b = 3
var x = if (a>b) a else b
println(x)
```

В консоль будет выведено 5.

Также в Kotlin есть оператор when, заменивший привычный switch … case. Например, код для определения четности введенного числа:

```
val x1:Int = readLine()!!.toInt()
when (x1 % 2) {
    1 -> print("нечетное")
    0 -> print("четное")
}
```

Можно проверять вхождение в диапазон:

```
when (x1) {
    in 0..9 -> println("x - от 0 до 9")
    in 10..99 -> println("x - от 11 до 99")
    else -> println("другое")
}
```

Можно использовать без параметра и просто проверять выполнение условий:

```
when {
    x1 % 3 == 0 -> println("число делится на 3")
    x1 % 5 == 0 -> println("число делится на 5")
    else -> println("другое")
}
```

Но в данном случае нужно помнить, что при выполнении одного из условий **остальные не проверяются**.

Цикл for представлен в Kotlin в немного отличном от Java виде. Например:

```
for (i in 1..3) {
    println(i)
}
```

Данный цикл выведет в консоль числа от 1 до 3. Другой пример:

```
for (i in 6 downTo 0 step 2) {
    println(i)
}
```

Этот цикл выведет в консоль числа от 6 до 0 с шагом 2 (6 4 2 0).

Циклы while и do … while работают как в C/C++/Java:

```
var x = 10
while (x > 0) {
    print("x=$x ")
    x--
}
```

В консоль будет выведена строка

x=10 x=9 x=8 x=7 x=6 x=5 x=4 x=3 x=2 x=1

```
var y = 0
do{
    println("${y++}")
} while (y < 10)
```

В консоль будут выведены числа от 0 до 9 включительно.

Задания

1. Написать программу, выясняющую кол-во четных цифр во введенном пользователем числе.
2. Написать программу, выясняющую кол-во нечетных цифр во введенном пользователем числе.
3. Разложить заданное число на простые множители.
4. Число, равное сумме всех своих делителей, включая единицу, называется совершенным. Найти и напечатать все совершенные числа в интервале от 2 до x.
5. Среди всех n-значных чисел указать те, сумма цифр которых равна данному числу k (пользователь вводит n и k).
6. Среди всех n-значных чисел указать те, произведение цифр которых равна данному числу k (пользователь вводит n и k).
7. Найти наибольшую и наименьшую цифры в записи данного натурального числа n.
8. Дано натуральное число N. Найти и вывести все числа в интервале от 1 до N–1, у которых сумма всех цифр совпадает с суммой цифр данного числа. Если таких чисел нет, то вывести слово «нет». Пример. N = 44. Числа: 17, 26, 35.
9. Дано натуральное число N. Найти и вывести все числа в интервале от 1 до N–1, у которых произведение всех цифр совпадает с произведением цифр данного числа. Если таких чисел нет, то вывести слово «нет». Пример. N = 32. Числа: 6, 16, 23.
10. Дано натуральное число N. Найти и вывести все числа в интервале от 1 до N–1, у которых произведение всех цифр совпадает с суммой цифр данного числа. Если таких чисел нет, то вывести слово «нет». Пример. N = 44. Числа: 18, 24.
11. Дано натуральное число N. Найти и вывести все числа в интервале от 1 до N–1, у которых сумма всех цифр совпадает с произведением цифр данного числа. Если таких чисел нет, то вывести слово «нет». Пример. N = 32. Числа: 24, 15, 6.
12. Составить программу, которая выводит на экран таблицу умножения от 1 до 9 в десятичной системе счисления.
13. С клавиатуры вводится двузначное число, среди всех четырехзначных чисел вывести на экран те, которые начинаются или заканчиваются этим числом.
14. Среди четырехзначных чисел из интервала, заданного пользователем, найти все, у которых сумма первых двух цифр равна сумме последних двух.

15. Среди четырехзначных чисел из интервала, заданного пользователем, найти все, у которых произведение первых двух цифр равна сумме последних двух.
16. Среди четырехзначных чисел из интервала, заданного пользователем, найти все, у которых произведение первых двух цифр равна произведению последних двух.
17. Среди четырехзначных чисел из интервала, заданного пользователем, найти все, у которых сумма первых двух цифр равна произведению последних двух.
18. Пользователь вводит числа x, a, b. Из промежутка от a до b найти все числа, сумма цифр которых дает x.
19. Пользователь вводит числа x, a, b. Из промежутка от a до b найти все числа, разность цифр которых по модулю дает x.
20. Пользователь вводит x, a, b. Из промежутка от a до b найти все числа, произведение цифр которых по модулю дает x.

Лабораторная работа №2. Массивы в Kotlin

Для создания массивов в Kotlin есть несколько вариантов.

Для примитивных типов данных существуют специальные классы ByteArray, ShortArray, IntArray и т.д. Например, создадим массив из 3 целых чисел:

```
val x: IntArray = intArrayOf(5, 2, 8)
x.forEach { print("$it ") } //выведет на экран 5 2 8
```

Переменная x представляет собой массив. Во второй строчке для массива вызывается цикл forEach (есть для каждого класса массивов и списков), в теле которого можно прописать действие, которое будет выполняться с каждым элементом массива, причем сам элемент будет доступен по ссылке it (на самом деле здесь используется подход, основанный на понятии лямбда, о нем в следующей лабораторной). То же самое можно написать и более стандартным способом:

```
for (i in x) {
    print("$i ")
}
```

Но предпочтительнее первый способ как менее подверженный потенциальным ошибкам.

Можно создать массив без заранее определенных элементов и задать их через генератор случайных значений:

```
var arr = IntArray(5)        //создаем массив из 5 целых чисел
for (i in arr.indices){      //делаем цикл по индексам массива
    arr[i] = Random.nextInt(10)  //и присваиваем каждому элементу рандомное значение
    print("${arr[i]} ")
}
```

Можно совместить объявление массива и присвоение каждому элементу случайного значения:

```
var mas = IntArray(5) { Random.nextInt(10) } //или var mas = IntArray(5, { Random.nextInt(10) })
mas.forEach { print("$it ") }
```

В первой строчке мы задаем размер массива и функцию, которая будет вызываться для каждого элемента массива.

В языке Kotlin есть много встроенных методов для работы с массивом:

- *reversedArray*() – переворачивает массив и возвращает новый массив, старый массив остается без изменений

```
val numbers: IntArray = intArrayOf(1, 2, 3, 4)
var numbers2 = numbers.reversedArray()
println(Arrays.toString(numbers2)) //выведет на экран [4, 3, 2, 1]
```

- *reverse*() – переворачивает массив без создания нового

```
val numbers: IntArray = intArrayOf(1, 2, 3, 4)
numbers.reverse()
println(Arrays.toString(numbers)) //выведет на экран [4, 3, 2, 1]
```

- *sort*() – сортирует массив от меньшего к большему

```
val numbers: IntArray = intArrayOf(7, 5, 8, 4, 9, 6)
numbers.sort()
println(Arrays.toString(numbers)) //выведет на экран [4, 5, 6, 7, 8, 9]
```

Сортировать можно не весь массив, а только определённый диапазон. Для этого в методе sort() нужно указать начало и размер диапазона. Например, для сортировки только первых трёх элементов из предыдущего примера нужно написать `numbers.sort(0, 3)`. Тогда будут отсортированы только первые 3 элемента, остальные останутся в прежнем виде.

- `sortDescending()` – сортирует массив от большего к меньшему
- `sortedArray()`, `sortedArrayDescending()` – сортируют и возвращают элементы в новый массив, старый остается без изменений.
- `contains(element)` – проверяет, содержится ли element в массиве, возвращает true или false

```
val numbers: IntArray = intArrayOf(7, 5, 8, 4, 9, 6)
println("Число 4 ${if (!numbers.contains(4)) "не" else ""} содержится в массиве")
```

В данном примере в консоль будет выведено «Число 4 содержится в массиве». Строка

```
println("Число 4 ${if (!numbers.contains(4)) "не" else ""} содержится в массиве")
```

использует оператор if для формирования строки, т.е. если не содержится элемент 4, то в строку добавляется частица «не», если содержится – то ничего не добавляется. В данном случае необходимо использовать именно полный формат if … else, этого требует синтаксис встраиваемых выражений внутри println.

- `average()` – среднее арифметическое элементов массива

```
val array = arrayOf(2, 3, 2)
println(array.average()) //2.3333333333333335
```

Чтобы вывести число с нужным количеством знаков в дробной части, можно использовать форматирование в стиле языка Си:

```
println("%.2f".format(array.average())) // 2,33
```

- `sum()` – сумма элементов массива
- `min()` – минимальный элемент массива
- `max()` – максимальный элемент массива
- `intersect()` – пересечение массивов

```
val firstArray = intArrayOf(1, 2, 3, 4, 5)
val secondArray = intArrayOf(3, 5, 6, 7, 8)
val intersectedArray = firstArray.intersect(secondArray.asIterable())
intersectedArray.forEach{print("$it ")}    // 3 5
```

- `distinct()` – возвращает набор только уникальных элементов массива

```
var ArrNums = intArrayOf(5, 4, 3, 2, 3, 5, 1)
println("Уникальные эл-ты в ArrNum: " + ArrNums.distinct()) //5, 4, 3, 2, 1
```

Задания

1. Дан массив. Удалить из него нули и после каждого числа, оканчивающего на 5, вставить 1.
2. Случайным образом генерируется массив чисел. Пользователь вводит числа a и b. Заменить элемент массива на сумму его соседей, если элемент массива четный и номер его лежит в промежутке от a до b.
3. В одномерном массиве удалить промежуток элементов от максимального до минимального.
4. Дан одномерный массив. Переставить элементы массива задом-наперед.
5. Сформировать одномерный массив случайным образом. Определить количество четных элементов массива, стоящих на четных местах.
6. Задается массив. Определить порядковые номера элементов массива, значения которых содержат последнюю цифру первого элемента массива 2 раза (т.е. в массиве должны быть не только однозначные числа).
7. Сформировать одномерный массив из целых чисел. Вывести на экран индексы тех элементов, которые кратны трем и пяти.
8. Задается массив. Написать программу, которая вычисляет, сколько раз введенная с клавиатуры цифра встречается в массиве.
9. Задается массив. Узнать, какие элементы встречаются в массиве больше одного раза.
10. Даны целые числа $a_1, a_2, \dots, a_n$. Вывести на печать только те числа, для которых $a_i \geq i$.
11. Дан целочисленный массив с количеством элементов n. Напечатать те его элементы, индексы которых являются степенями двойки.
12. Задана последовательность из N чисел. Определить, сколько среди них чисел меньших К, равных К и больших К.
13. Задан массив действительных чисел. Определить, сколько раз меняется знак в данной последовательности чисел, напечатать номера позиций, в которых происходит смена знака.
14. Задана последовательность N чисел. Вычислить сумму чисел, порядковые номера которых являются простыми числами.
15. Дан массив чисел. Указать те его элементы, которые принадлежат отрезку [c, d].

16. Массив состоит из нулей и единиц. Поставить в начало массива нули, а затем единицы.
17. Дан массив целые положительных чисел. Найти среди них те, которые являются квадратами некоторого числа x.
18. В массиве целых чисел найти наиболее часто встречающееся число. Если таких чисел несколько, то определить наименьшее из них.
19. Дан целочисленный массив с количеством элементов n. Сжать массив, выбросив из него каждый второй элемент.
20. Дан массив, состоящий из n натуральных чисел. Образовать новый массив, элементами которого будут элементы исходного, оканчивающиеся на цифру k.
21. Даны действительное число x и массив A[n]. В массиве найти два члена, среднее арифметическое которых ближе всего к x.
22. Даны два массива A и B. Найти, сколько элементов массива A совпадает с элементами массива B.

Лабораторная работа №3. ООП в Kotlin

Для создания класса с нужными полями и конструктором в языке Kotlin достаточно всего одной строчки:

```
open class Auto(var firm: String = "Без названия", var maxSpeed: Int = 0)
```

В данном случае создается класс Auto с двумя полями (firm и maxSpeed) и с конструктором, который по умолчанию присваивает этим полям нужные значения. Для проверки используем следующий код:

```
fun main(){
    val myAuto: Auto = Auto("Ford", 400)
    println("${myAuto.firm} ${myAuto.maxSpeed}")
    val myAuto2 = Auto()
    println("${myAuto2.firm} ${myAuto2.maxSpeed}")
    val myAuto3 = Auto(maxSpeed = 200)
    println("${myAuto3.firm} ${myAuto3.maxSpeed}")
}
```

В консоль будет выведено

Ford 400
Без названия 0
Без названия 200

Т.е. мы можем вызывать конструктор с нужными параметрами, без параметров и с частичными параметрами, указав имя параметра.

Теперь можно наследоваться от созданного класса и расширять его функциональность:

```
class Car (firm: String = "No",
           maxSpeed: Int = 0,
           var model: String = "Нет",
           var numDoors: Int = 4,
           var fullTime: Boolean = false): Auto(firm, maxSpeed) {

    override fun toString() = "Легковая машина (фирма=$firm, максСкорость=$maxSpeed, модель=$model, " +
            "кол-во дверей=$numDoors, полноприводный=${if (fullTime) "да" else "нет"} )"
}
```

В данном случае мы создали класс Car, который унаследован от класса Auto. При этом мы указываем все поля создаваемого класса, в том числе и те, которые наследуются из класса Auto, и задаем им значения по умолчанию. Те параметры, которые наследуются из класса Auto (это firm и maxSpeed), мы передаем в класс Auto.

Также в новом классе мы переопределяем метод toString (это стандартный метод, он определен для всех объектов по умолчанию, когда мы выводим объект в консоль или в строку, но информация, которую он выводит, не очень полезная, например, println(myAuto3) выведет на экран Lab3.Auto@2812cbfa). Для перегрузки (или переопределения) используется ключевое слово override перед именем перегружаемой функции. В самой функции мы просто написали формат вывода информации об объекте.

Теперь можем проверить, как выводятся на экран объекты класса Car:

```
val myCar1 = Car(firm = "Mers", numDoors = 2)
println(myCar1)
```

В консоль будет введено

Легковая машина (фирма=Mers, максСкорость=0, модель=Нет, кол-во дверей=2, полноприводный=нет)

Мы создали объект класса Car, использовав только 2 параметра, остальные были взяты из конструктора класса, в котором мы указали значения по умолчанию.

Создадим еще один класс-наследник от Auto – Truck (грузовой автомобиль).

```
class Truck (firm: String = "No",
             maxSpeed: Int = 0,
             var model: String = "Нет",
             var power: Int = 0,
             var trailer: Boolean = false) : Auto(firm, maxSpeed){

    fun setAllInfo() {
         print("Введите фирму-производитель грузового авто: ")
         firm = readLine()!!
         print("Введите максимальную скорость грузового авто: ")
         maxSpeed = readLine()!!.toInt();
         print("Введите модель грузового авто: ")
         model = readLine()!!
         print("Введите мощность грузового авто: ");
         power = readLine()!!.toInt()
         print("Введите признак прицепа грузового авто (true/false): ");
         trailer = readLine()!!.toBoolean()
         println()
    }

    override fun toString() = "\n\tГрузовик"+"\n\t"+" Фирма: "+firm+"\n\t"+
       " Максимальная скорость: " +maxSpeed+"\n\t"+" Модель: "+model+"\n\t"+
       " Мощность: "+power+"\n\t"+" Признак прицепа: " +trailer+"\n"
}
```

В этом классе добавлена функция `setAllInfo`, которая позволяет ввести всю нужную информацию с клавиатуры и присвоить ее полям создаваемого объекта класса Truck. Можно проверить работу с классом:

```
val myTruck = Truck()
myTruck.setAllInfo()
println(myTruck)
```

Пример ввода с консоли и вывода полной информации об объекте:

Введите фирму-производитель грузового авто: Dove
Введите максимальную скорость грузового авто: 300
Введите модель грузового авто: k3
Введите мощность грузового авто: 500
Введите признак прицепа грузового авто (true/false): true

Грузовик
Фирма: Dove
Максимальная скорость: 300
Модель: k3
Мощность: 500
Признак прицепа: true

Теперь можно создать класс-агрегатор (гараж) для хранения наших объектов:

```
class GarageAuto {
    private var masAuto = ArrayList<Auto>();
    fun addAuto (auto: Auto) { //для добавления объектов в массив
        masAuto.add(auto)
    }
    fun findAuto(a: Auto) : Boolean{ //для поиска объектов в массиве
        return masAuto.contains(a)
    }
    override fun toString(): String { //для вывода в консоль
        var str = "В гараже:\n "
        for (a in masAuto) {
            str = str+("\t" + a+"\n")
        }
        return str
    }
}
```

Объекты в этом классе хранятся в массиве типа ArrayList. Это шаблонный класс, мы указали в угловых скобках тип объектов, которые будут в нем храниться (Auto). Т.к. мы указали базовый класс, то и все объекты классов наследников смогут храниться в этом массиве.

Для проверки работы нового класса можно использовать следующий код:

```
val myAuto: Auto = Auto("Ford", 400)
val myCar1 = Car(firm = "Mers", numDoors = 2)
val myTruck = Truck("Kamaz", 200, "Masters", 500, false)
```

```
var myGarage = GarageAuto()
myGarage.addAuto(myAuto)
myGarage.addAuto(myCar1)
myGarage.addAuto(myTruck)
println(myGarage)
if (myGarage.findAuto(myCar1)) println("Есть машина $myCar1")
```

Вывод программы:

В гараже:

Lab3.Auto@96532d6
Легковая машина (фирма=Mers, максСкорость=0, модель=Нет, кол-во дверей=2, полноприводный=нет)

Грузовик
Фирма: Kamaz
Максимальная скорость: 200
Модель: Masters
Мощность: 500
Признак прицепа: false

Есть машина Легковая машина (фирма=Mers, максСкорость=0, модель=Нет, кол-во дверей=2, полноприводный=нет)

Также в Kotlin есть специальный оператор is, который позволяет определить принадлежность объекта к определенному классу. Например, при выполнении следующего кода:

```
for (a in masAuto) {
    if (a is Car) println("Легковая машина")
}
```

текст `Легковая машина` будет выведен столько раз, сколько легковых машин содержится в гараже.

Задания (создать классы, в них предусмотреть различные поля классов и методы для работы):

1. Базовый класс – учащийся. Производные – школьник и студент. Создать класс Конференция, который может содержать оба вида учащихся. Предусмотреть метод подсчета участников конференции отдельно по школьникам и по студентам (использовать оператор is).
2. Базовый класс – работник. Производные – работник на почасовой оплате и на окладе. Создать класс Предприятие, который может содержать оба вида работников. Предусмотреть метод подсчета работников отдельно на почасовой оплате и на окладе (использовать оператор is).
3. Базовый класс – компьютер. Производные – ноутбук и смартфон. Создать класс РемонтСервис, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно ремонтируемых ноутбуков и смартфонов (использовать оператор is).
4. Базовый класс – печатные издания. Производные – книги и журналы. Создать класс КнижныйМагазин, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно книг и журналов (использовать оператор is).
5. Базовый класс – помещения. Производные – квартира и офис. Создать класс Дом, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно квартир и офисов (использовать оператор is).
6. Базовый класс – файл. Производные – звуковой файл и видео-файл. Создать класс Каталог, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно звуковых и видео-файлов (использовать оператор is).
7. Базовый класс – летательный аппарат. Производные – самолет и вертолет. Создать класс Авиакомпания, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно самолетов и вертолетов (использовать оператор is).
8. Базовый класс – соревнование. Производные – командные соревнования и личные. Создать класс Чемпионат, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно командных соревнований и личных (использовать оператор is).
9. Базовый класс – мебель. Производные – диван и шкаф. Создать класс Комната, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно диванов и шкафов (использовать оператор is).
10. Базовый класс – оружие. Производные – огнестрельное и холодное. Создать класс ОружейнаяПалата, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно огнестрельного и холодного оружия (использовать оператор is).
11. Базовый класс – оргтехника. Производные – принтер и сканер. Создать класс Офис, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно принтеров и сканеров (использовать оператор is).
12. Базовый класс – СМИ. Производные – телеканал и газета. Создать класс Холдинг, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно телеканалов и газет (использовать оператор is).

Лабораторная работа №4. Работа с графическим интерфейсом в Kotlin и лямбда-функции

В качестве библиотеки для создания графического интерфейса средствами Kotlin можно использовать любые Java-пакеты. Рассмотрим на пример пакета Swing. Соответственно, будем использовать все те же классы, что и в Java (см. Лабораторные по языку Java), только на Kotlin. Например, код для создания окна:

```
fun main(){
    var form: JFrame = JFrame("Окно")
    form.defaultCloseOperation = JFrame.EXIT_ON_CLOSE
    form.size = Dimension(200,200)
    form.setLocationRelativeTo(null)
    form.isVisible = true
}
```

Сделаем интерфейс для отображения объектов из предыдущей лабораторной (Auto, Car, Truck, GarageAuto) в виде списка JList, с возможностью создания объектов, выбора объектов в списке и изменения полей объектов:

```
var northBox = Box(BoxLayout.Y_AXIS)    // верхняя панель формы
var northBoxUp = Box(BoxLayout.X_AXIS)  //верхняя часть верхней панели
var northBoxDown = Box(BoxLayout.X_AXIS) //нижняя часть верхней панели
var listOfAuto = JList<Auto>()          //список для отображения авто
var listModel = DefaultListModel<Auto>()  //модель данных для списка
var garage = GarageAuto()               //гараж
var fieldsMap = HashMap<String, JComponent>()  //HashMap для хранения названий полей ввода данных и самих полей
val butCreate = JButton("Create")   //кнопка для показа полей ввода данных

fun main() {
    var form: JFrame = JFrame("Окно")
    form.defaultCloseOperation = JFrame.EXIT_ON_CLOSE
    form.setLocationRelativeTo(null)  // чтоб форма появилась в центре экрана
    buildNorth(form)        //вызываем функцию для заполнения верхней части формы
    buildCenter(form)        //вызываем функцию для заполнения центральной части формы
    form.pack()         //упаковываем форму
    form.minimumSize = form.size    //задаем минимальный размер
    form.isVisible = true           //показываем форму
}

fun buildNorth(form: JFrame) {  //функция для заполнения верхней части формы
    var combo = JComboBox<String>()  //комбо-бокс для выбора между Car и Truck
    combo.addItem("Car")        //добавляем элементы в комбо-бокс
    combo.addItem("Truck")
    northBoxUp.add(combo)   //добавляем комбо-бокс на верхнюю часть верхней панели
    northBoxUp.add(Box.createHorizontalStrut(10))  //добавляем промежуток после комбо-бокса
    northBoxUp.add(butCreate)          //добавляем кнопку "Create"
    northBoxUp.add(Box.createHorizontalGlue())  //добавляем пружину после кнопки
    var auto = Auto()  //объект класса авто
    val addButton = JButton("Add")  //кнопка для добавления данных в объект и в JList
    butCreate.addActionListener{ //слушатель нажатия на кнопку
        northBoxDown.removeAll()  //удаляем всё с нижней части верхней панели
        northBoxDown.isVisible = true // показываем нижнюю часть верхней панели
        butCreate.isEnabled = false  //делаем неактивной кнопку "Create"
   //создаем HashMap с полями и типами полей либо класса Car, либо класса Truck в зависимости от выбора в комбо-боксе
        var fields = if (combo.selectedItem == "Car") hashMapOf("firm" to "String", "maxSpeed" to "Int",
                                                "model" to "String", "numDoors" to "Int", "fullTime" to "Boolean")
                     else hashMapOf("firm" to "String", "maxSpeed" to "Int", "model" to "String", "power" to "Int",
                                    "trailer" to "Boolean")
//вызываем функцию, которой передаем созданный HashMap и получаем новый HashMap с названиями полей ввода данных
//и самими полями, а сама функция создает поля ввода данных в зависимости от типа поля
        fieldsMap=fillNorthBoxDown(fields) //и добавляет их в нижнюю часть верхней панели
        northBoxDown.add(addButton) //добавляем кнопку для добавления данных в объект и в JList
        form.pack() //упаковываем форму
    }
    addButton.addActionListener { //слушатель нажатия на кнопку для добавления данных в объект и в JList
        butCreate.isEnabled = true  //активируем кнопку "Create"
        northBoxDown.isVisible = false // убираем нижнюю часть верхней панели
        //в зависимости от выбранного значения комбо-бокса присваиваем объекту auto результат работы нужной функции
        if (combo.selectedItem == "Car")  auto = createCar()  else auto = createTruck()
        println(auto) //вывод в консоль для отладки
        listModel.addElement(auto) //в модель данных для списка добавляем созданный объект
        garage.addAuto(auto) //и в гараж тоже добавляем
        form.pack() //упаковываем форму
    }
    northBoxDown.isVisible = false // убираем нижнюю часть верхней панели
    northBox.add(northBoxUp) //добавляем верхнюю часть верхней панели
    northBox.add(northBoxDown)  //добавляем нижнюю часть верхней панели
    form.add(northBox, BorderLayout.NORTH) //добавляем верхнюю панель на форму
}
//функция для создания полей ввода данных в зависимости от типа поля, возвращает хэш-map с компонентами формы
fun fillNorthBoxDown(fields: HashMap<String, String>): HashMap<String, JComponent> {
    var fieldsHash = HashMap<String, JComponent>() //создаем хэш-map, ключи – названия полей, значения - сами поля
    fields.forEach { k, v -> //цикл по всем парам ключ-значение хэш-мапа, переданного в функцию
// k, v -> - это лямбда-функция, про неё написано после рис. 4
        var box = Box(BoxLayout.Y_AXIS) //создаем вертикальную коробку, сверху будет название поля, внизу - само поле
```

```
        box.border = BorderFactory.createLineBorder(Color.black) //делаем ей границу
        box.add(JLabel(k)) //добавляем название поля
        when (v) { //в зависимости от типа поля будем создавать разные поля ввода данных
            "String" -> { //если строка - то обычное текстовое поле
                val textField = JTextField(10)
                box.add(textField)
                northBoxDown.add(box)
                fieldsHash.put(k, textField) //добавляем название поля и само поле в хэш-мап в виде ключ-значение
            }
            "Int" -> { //если целое - то форматированное текстовое поле с числовыми значениями
                var numberField = JFormattedTextField(NumberFormat.getNumberInstance())
                numberField.columns = 10
                box.add(numberField)
                northBoxDown.add(box)
                fieldsHash.put(k, numberField) //добавляем название поля и само поле в хэш-мап в виде ключ-значение
            }
            "Boolean" -> { //если логическое - то комбо-бокс с выбором true или false
                var comboBoolean = JComboBox<Boolean>()
                comboBoolean.addItem(true)
                comboBoolean.addItem(false)
                comboBoolean.preferredSize = Dimension(100, 10) //делаем размер комбо-бокса побольше
                box.add(comboBoolean)
                northBoxDown.add(box)
                fieldsHash.put(k, comboBoolean) //добавляем название поля и само поле в хэш-мап в виде ключ-значение
            }
        }
    }
    return fieldsHash  //возвращаем созданный хэш-мап с названиями полей и полями
}

fun createCar() : Car{  //функция для создания объекта класса Car из значений, вводимых в поля ввода данных
    //данные берутся из fieldsMap
    val firmField = fieldsMap.get("firm") as JTextField
    val maxSpeedField = fieldsMap.get("maxSpeed") as JFormattedTextField
    val modelField = fieldsMap.get("model") as JTextField
    val numDoorsField = fieldsMap.get("numDoors") as JFormattedTextField
    val fullTimeField = fieldsMap.get("fullTime") as JComboBox<Boolean>
    return Car(firmField.text, maxSpeedField.text.toInt(), modelField.text, numDoorsField.text.toInt(),
                    fullTimeField.selectedItem as Boolean)
}

fun createTruck() : Truck{  //функция для создания объекта класса Truck из значений, вводимых в поля ввода данных
    val firmField = fieldsMap.get("firm") as JTextField
    val maxSpeedField = fieldsMap.get("maxSpeed") as JFormattedTextField
    val modelField = fieldsMap.get("model") as JTextField
    val powerField = fieldsMap.get("power") as JFormattedTextField
    val trailerField = fieldsMap.get("trailer") as JComboBox<Boolean>
    return Truck(firmField.text, maxSpeedField.text.toInt(), modelField.text, powerField.text.toInt(),
                    trailerField.selectedItem as Boolean)
}

fun buildCenter(form: JFrame) {  //функция для заполнения центральной части формы
    listOfAuto.model = listModel  //устанавливаем модель данных для списка на форме
    form.add(listOfAuto, BorderLayout.CENTER) //добавляем список на форму в центр
//добавим возможность выбора объекта, чтоб его поля грузились в соответствующие поля в нижней части верхней панели
//и чтоб можно было менять данные в объекте или отменять изменения
    listOfAuto.addMouseListener(object : MouseAdapter() { //создаем слушатель событий мыши для списка
        override fun mousePressed(e: MouseEvent?) { //перегружаем метод нажатия на клавишу мыши
            super.mousePressed(e)   //вызываем метод суперкласса
            val tempList= e!!.source as JList<Auto>  //получаем объект-источник события (сам список)
            println(tempList.selectedValue) //выводим в консоль выбранный в списке эл-нт для контроля
            var tempAuto = tempList.selectedValue //получаем выбранный в списке эл-нт (по которому щелкнули мышью)
            northBoxDown.removeAll()  //убираем всё с нижней части верхней панели
            northBoxDown.isVisible = true //показываем нижнюю часть верхней панели
//создаем HashMap с полями и типами полей либо класса Car, либо класса Truck в зависимости от класса объекта tempAuto
            var fields = if (tempAuto is Car) hashMapOf("firm" to "String", "maxSpeed" to "Int",
                                                        "model" to "String", "numDoors" to "Int", "fullTime" to "Boolean")
            else hashMapOf("firm" to "String", "maxSpeed" to "Int", "model" to "String",
                           "power" to "Int", "trailer" to "Boolean")
            fieldsMap=fillNorthBoxDown(fields) //вызываем функцию fillNorthBoxDown и передаем ей созданный выше хэш-мап
//с полями класса, сама функция создает поля ввода данных в зависимости от типа поля и добавляет их в нижнюю часть
//верхней панели и возвращает новый HashMap с названиями полей ввода данных и самими полями
            if (tempAuto is Car) { //если объект, по которому щелкнули, принадлежит классу Car
            //то заполняем созданные поля данными из объекта, вызывая соответствующие методы класса Car
                (fieldsMap.get("firm") as JTextField).text = tempAuto.firm
                (fieldsMap.get("maxSpeed") as JFormattedTextField).value = tempAuto.maxSpeed
                (fieldsMap.get("model") as JTextField).text = tempAuto.model
                (fieldsMap.get("numDoors") as JFormattedTextField).value = tempAuto.numDoors
                (fieldsMap.get("fullTime") as JComboBox<Boolean>).selectedItem = tempAuto.fullTime
            }
            else {//иначе у нас объект, по которому щелкнули, принадлежит классу Truck
                //и заполняем созданные поля данными из объекта, вызывая соответствующие методы класса Truck
                (fieldsMap.get("firm") as JTextField).text = tempAuto.firm
                (fieldsMap.get("maxSpeed") as JFormattedTextField).value = tempAuto.maxSpeed
```

```
            (fieldsMap.get("model") as JTextField).text = (tempAuto as Truck).model
            (fieldsMap.get("power") as JFormattedTextField).value = (tempAuto as Truck).power
            (fieldsMap.get("trailer") as JComboBox<Boolean>).selectedItem = (tempAuto as Truck).trailer
        }

        val changeButton = JButton("Change") //создаем кнопку для подтверждения изменения данных в объекте
        northBoxDown.add(changeButton) //добавляем кнопку
        northBoxDown.add(Box.createHorizontalStrut(10)) //добавляем отступ
        val cancelButton = JButton("Cancel") //создаем кнопку для отмены изменений данных в объекте
        northBoxDown.add(cancelButton) //добавляем кнопку
        form.pack() //упаковываем форму
        val i = listModel.indexOf(tempAuto) //получаем из модели данных списка номер выбранного объекта
        changeButton.addActionListener{ //добавляем к кнопке Change слушатель нажатия
            northBoxDown.isVisible = false  //скрываем нижнюю часть верхней панели
            butCreate.isEnabled = true  //активируем кнопку Create
            //присваиваем объекту tempAuto новое значение, либо через функцию createCar, либо через createTruck()
            if (tempAuto is Car) tempAuto = createCar() else tempAuto = createTruck()
            listModel.set(i, tempAuto) //меняем выбранный в списке объект на новый с измененными данными
           garage.changeAuto(i, tempAuto) //меняем объект в гараже
        }
        cancelButton.addActionListener { //слушатель для кнопки отмены изменений данных
            northBoxDown.isVisible = false //скрываем нижнюю часть верхней панели
            butCreate.isEnabled = true  //активируем кнопку Create
        }
        butCreate.isEnabled = false //делаем неактивной кнопку "Create"
    }))
}
```

Вот что должно получиться в итоге:

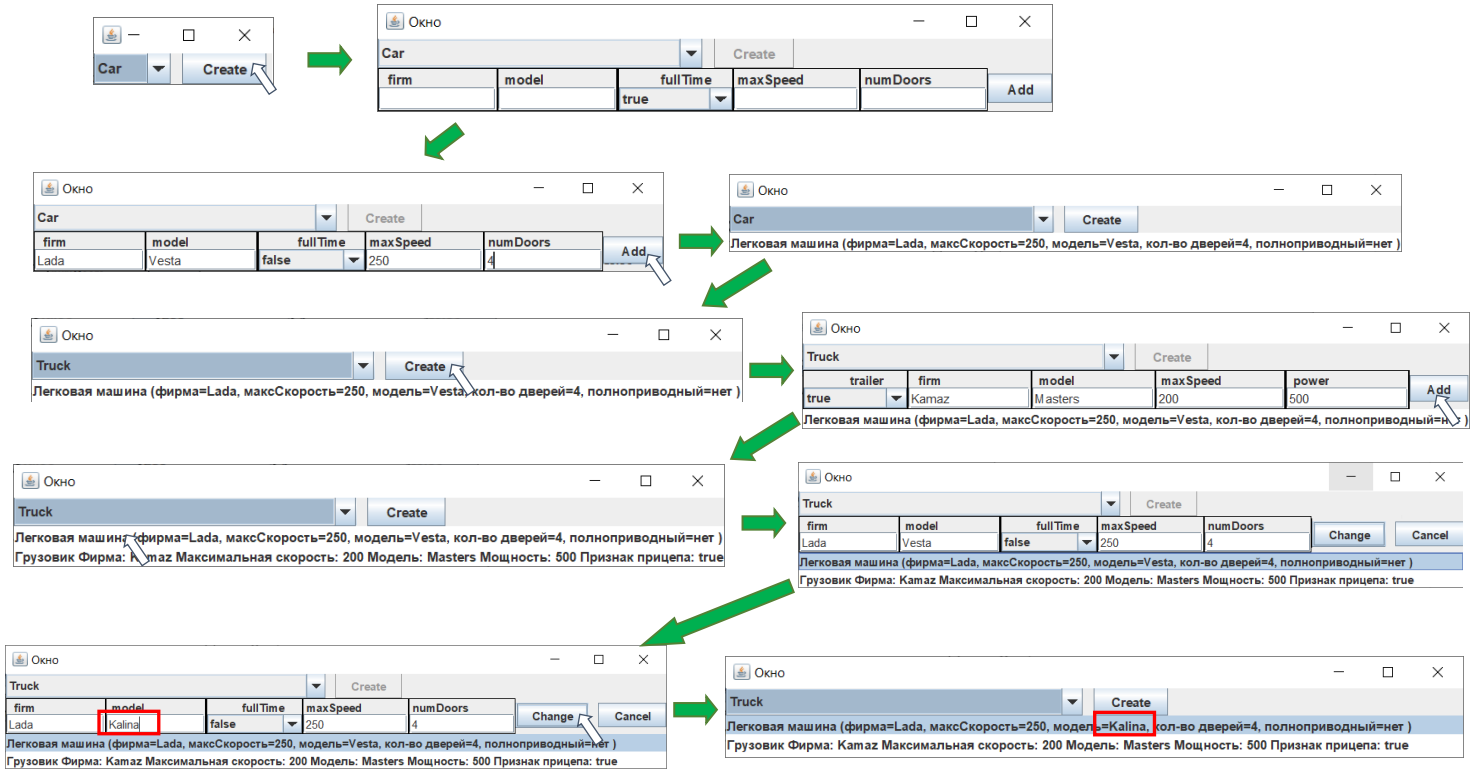


Рис. 4. Пример работы с программой

В приведенном выше участке программы `fields.forEach { k, v ->` в функции `fillNorthBoxDown` используется подход, основанный на лямбда-функциях. Разберем его.

Лямбда-функция – это вид анонимной функции. Состоит как правило из двух частей: до знака –> и после него, но можно и без первой части и знака –>. В нашем случае, у объекта fields (это хэш-мап) вызывается метод forEach (это цикл Для Каждого по всем парам ключ-значение, хранящимся в хэш-мапе). И дальше в фигурных скобках идет лямбда-функция. Ей в качестве параметров даются переменные k и v (ключ и значение каждой пары соответственно), они указаны до стрелки –>. После стрелки указываются действия, которые нужно выполнить с участием переменных k и v. Например, код

```
var fields = HashMap<String, Int>()
fields.set("Иванов", 25)
fields.set("Петров", 30)
fields.set("Сидоров", 35)
fields.forEach { k, v ->
    println("$k  -  $v")
}
```

выведет в консоль

Иванов - 25
Сидоров - 35
Петров - 30

Задание

Сделать графический интерфейс для работы с классами, созданными в ЛР №3. Добавить возможность вывода в консоль текущего содержимого с массивом объектов и возможность удаления выделенного объекта.

Лабораторная работа №5. Работа с файлами в Kotlin. Сериализация объектов

Добавим к программе, созданной в ЛР №4, возможность записывать данные в файл и считывать данные из файла. Т.к. данные являются объектами классов, то необходимо разобраться с таким понятием как сериализация. Это процесс, который переводит объект в последовательность байтов, по которой затем его можно полностью восстановить, и эту последовательность можно записать в файл. Соответственно, десериализация – это обратный процесс, когда из бинарного файла мы загружаем объекты.

Для начала внесем небольшие изменения в класс Auto:

```
open class Auto(var firm: String = "Без названия", var maxSpeed: Int = 0) : Serializable {
    companion object { //специальный дополнительный объект-компаньон для сериализации
        private const val serialVersionUID: Long = 123
    }
}
```

Мы сделали этот класс реализующим интерфейс `Serializable`, который и позволит записывать объекты данного класса и всех его наследников в файл. Дополнительно мы добавляем к нашему классу специальное поле (в виде отдельного объекта-компаньона), которое позволит сериализовать и десериализовать объекты независимо от того, на каком компьютере и с какой версией JVM это происходит.

Также добавим в класс GarageAuto следующий код:

```
fun writeToFile(fileName: String) { //метод для записи массива объектов в файл
    try { //секция для небезопасной работы с файлами
        val fos = FileOutputStream(fileName) //поток вывода данных в файл
        val oos = ObjectOutputStream(fos) //поток передачи объектов в другой поток
        oos.writeObject(masAuto) //записываем весь массив сразу
        println("Done writing")
        oos.close() //закрываем все потоки
        fos.close()
    } catch (e: IOException) { //ловим исключения
        e.printStackTrace()
    }
}

fun readFromFile(fileName: String) { //метод для чтения массива объектов из файла
    try {
        val fin = FileInputStream(fileName) //поток ввода данных из файла
        val ois = ObjectInputStream(fin)  //поток получения объектов из другого потока
        masAuto.clear()  //очищаем массив
        masAuto = ois.readObject() as ArrayList<Auto> //получаем массив из потока объектов
        println("From file: ")
        masAuto.forEach { println("$it") } //выводим в консоль полученный массив для проверки
        fin.close()  //закрываем все потоки
        ois.close()
    } catch (ex: Exception) {  //ловим исключения
        ex.printStackTrace()
    }
}

fun getAll(): ArrayList<Auto> { //для получения массива объектов из гаража
    return masAuto
}
```

И добавим заполнение нижней части для нашей программы с визуальным интерфейсом:

```
fun buildSouth(form: JFrame) { //функция для заполнения нижней части формы
    var southBox = Box(BoxLayout.X_AXIS) //коробка
    val butSave = JButton("Save")  //кнопка для сохранения данных в файл
    val butLoad = JButton("Load")  //кнопка для загрузки данных из файла
    southBox.add(butSave)            //загружаем кнопки в коробку, между ними - пружина
```

```
    southBox.add(Box.createHorizontalGlue())
    southBox.add(butLoad)
    var fileDialog = FileDialog(form)  //диалог для работы с файлами
    butSave.addActionListener {  //обработчик для кнопки сохранения
        fileDialog.mode = FileDialog.SAVE  //для диалога устанавливаем режим Сохранения
        fileDialog.title = "Сохранить файл"  //делаем ему заголовок
        fileDialog.isVisible = true  //делаем его видимым
        val fileName = fileDialog.directory+fileDialog.file  //получаем из него файл с полным путем
        println(fileName)  //выводим имя файла в консоль для отладки
        garage.writeToFile(fileName) //вызываем у объекта-гаража метод для записи данных в файл с нужным именем
    }
    butLoad.addActionListener { //обработчик для кнопки загрузки
        fileDialog.mode = FileDialog.LOAD  //для диалога устанавливаем режим Загрузки
        fileDialog.title = "Открыть файл"  //делаем ему заголовок
        fileDialog.isVisible = true  //делаем его видимым
        val fileName = fileDialog.directory+fileDialog.file  //получаем из него файл с полным путем
        println(fileName)  //выводим имя файла в консоль для отладки
        garage.readFromFile(fileName) //вызываем у объекта-гаража метод для чтения данных из файла с нужным именем
        listModel.clear()  //очищаем данные в списке
        listModel.addAll(garage.getAll())  //загружаем в список все данные из гаража (а там они из файла)
        form.pack()  //упаковываем форму
    }
    form.add(southBox, BorderLayout.SOUTH)  //добавляем нижнюю панель на форму
}
```

И добавляем строчку *buildSouth*(form) в функцию main. Если все сделано правильно, то в целом работа с сохранением и загрузкой данных будет примерно как на рис. 5.

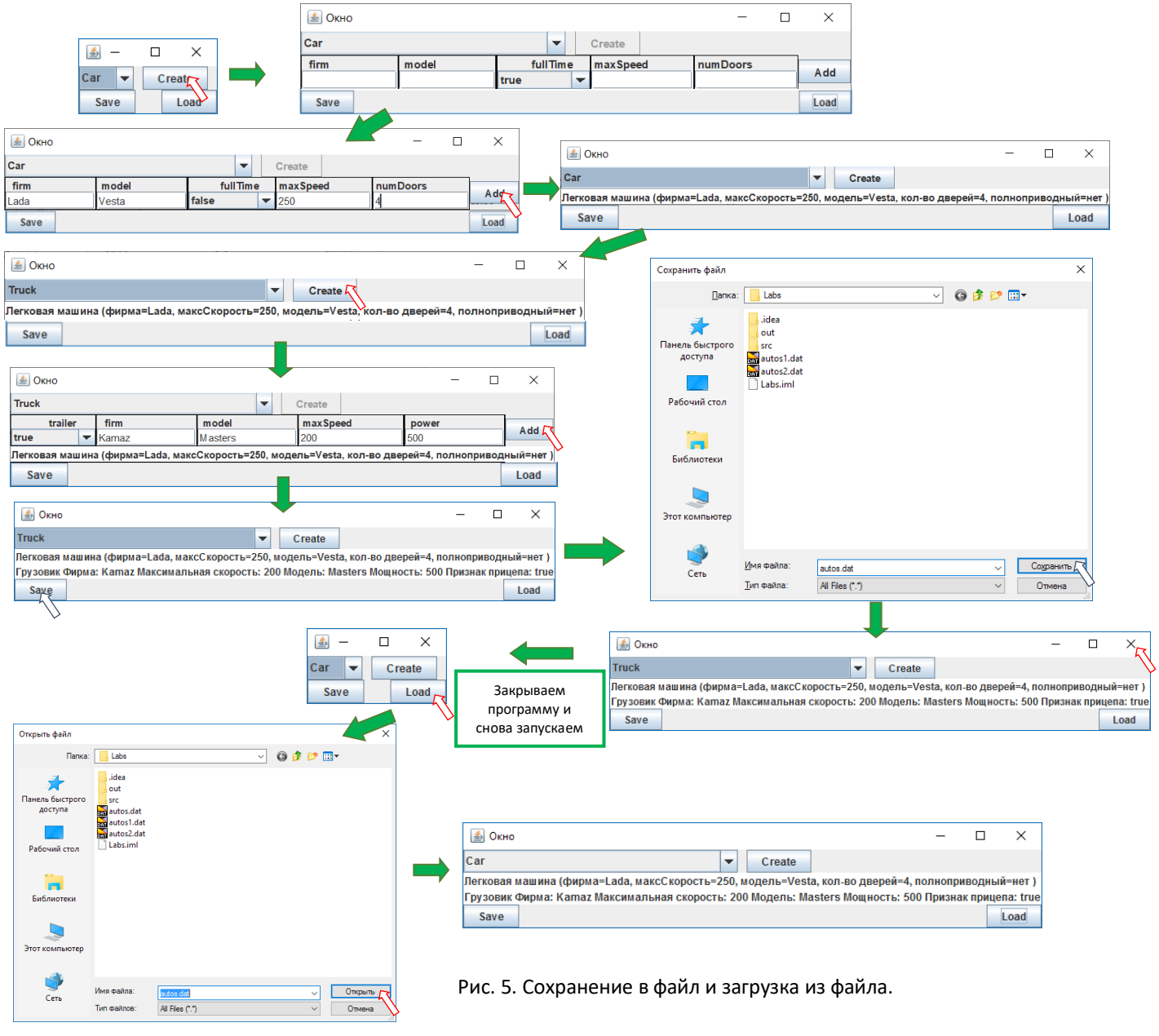


Рис. 5. Сохранение в файл и загрузка из файла.

Т.е. мы можем сохранить в файл все объекты, которые были созданы в программе, закрыть программу, запустить снова и загрузить все данные из файла.

Задание

Добавить к программе, сделанной по своему варианту в ЛР №4, возможности записи данных в файл и загрузки данных из файла.

Лабораторная работа №6. Создание pdf-документов средствами Kotlin

PDF — межплатформенный открытый формат электронных документов, изначально разработанный фирмой Adobe Systems с использованием ряда возможностей языка PostScript. В первую очередь предназначен для представления полиграфической продукции в электронном виде. Удобно использовать для создания электронных копий информации, представленной в программе, для последующего распространения или печати.

Добавим к своему проекту возможность сохранения данных в формате PDF. Существует несколько свободно распространяемых библиотек, позволяющих без особых проблем сделать подобное в любом Java-проекте, а, следовательно, и в Kotlin-проекте. Например, библиотека Apache PDFBox. Пример кода с использованием данной библиотеки:

```
fun main() {
    val doc = PDDocument() //создаем pdf-документ
    val page = PDPage()  //создаем страницу для документа
    doc.addPage(page)  //добавляем страницу в документ

    val contentStream = PDPageContentStream(doc, page) //создаем поток для вывода данных в документ
    contentStream.beginText()   //начинаем работу с потоком

    val font = PDType0Font.load( doc, File("arial.ttf")) //загружаем нужный шрифт с поддержкой кириллицы
//этот файл должен лежать в корневой папке проекта
    contentStream.setFont(font, 16f) //задаем размер шрифта
    contentStream.setLeading(24.5f)  //устанавливаем межстрочный интервал
    contentStream.newLineAtOffset(25f, 725f) //задаем позицию для первой строки,
                                             //относительно нижнего левого края документа
//создаем строки с текстом, который поместим в pdf-файл
    val text1 = "Пример добавления документа в pdf-файл. Hello, world "
    val text2 = "in pdf style! ☺ lol"

    contentStream.showText(text1) //добавляем текст из первой переменной в поток вывода
    contentStream.newLine()     //добавляем переход на новую строку
    contentStream.showText(text2) //добавляем текст из второй переменной в поток вывода
    contentStream.endText()     //окончание работы с текстом
    println("Content added")  //вывод в консоль для отладки

    contentStream.close()   //закрываем поток вывода
    doc.save("new1.pdf")  //сохраняем документ
    doc.close()    //закрываем документ
}
```

Для правильной работы этого кода в проект необходимо добавить библиотеку pdfbox-app-2.0.17.jar (есть на сервере): копируем этот файл в корень своего проекта, в IDEA выбираем пункт меню File – Project Structure. В появившемся окне выбираем Libraries (рис. 6).

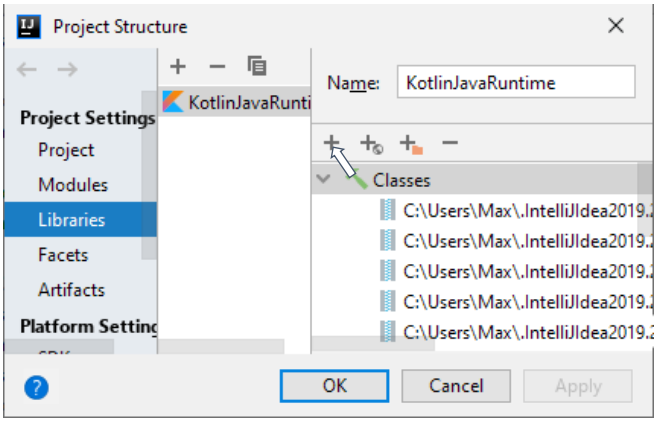


Рис. 6. Окно для добавления внешней библиотеки в проект.

Далее нажимаем на + (на рис. 6 на него указывает стрелка), появится окно выбора файла, указываем папку своего проекта и в ней нужный файл (рис. 7). Нажимаем OK и файл появится в разделе Classes в левой части окна Project Structure.

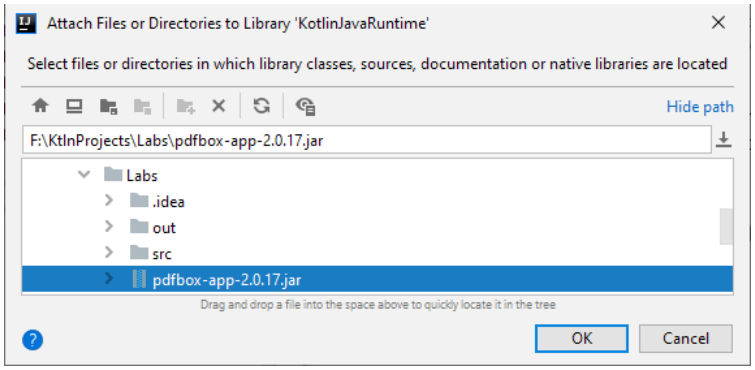


Рис. 7. Выбор файла с библиотекой.

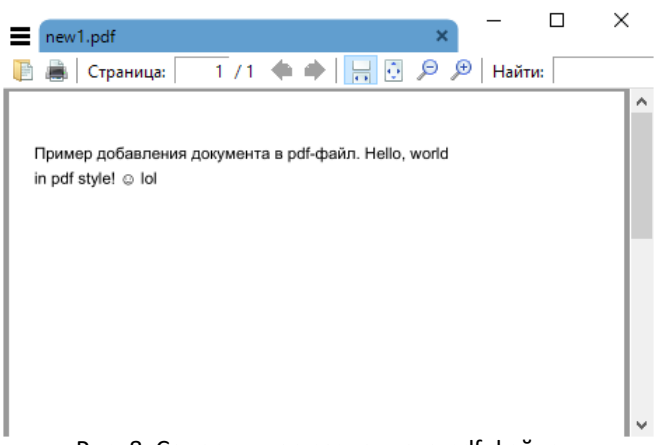


Рис. 8. Содержимое созданного pdf-файла.

Теперь можно запустить проект с примером. В результате запуска должен появится файл **new1.pdf** в корне проекта. Если открыть этот файл в любой программе для просмотра PDF, то там будет всего 2 строки с текстом (рис. 8).

Добавим возможность вывода данных в pdf-файл к проекту из лаб.раб.№5. Для этого подключим библиотеку pdfbox-app-2.0.17.jar как описано выше к нашему проекту. Далее, вносим изменения в функцию `buildSouth` – добавляем кнопку с надписью toPDF:

```
fun buildSouth(form: JFrame) {
    var southBox = Box(BoxLayout.X_AXIS)
    val butSave = JButton("Save")
    val butLoad = JButton("Load")
    val butToPDF = JButton("toPDF") //создаем новую кнопку
    southBox.add(butSave)
    southBox.add(Box.createHorizontalGlue())
    southBox.add(butLoad)
    southBox.add(Box.createHorizontalGlue()) // пружина
    southBox.add(butToPDF)    //добавляем новую кнопку в бокс
// тут идет код без изменения до butLoad.addActionListener {   …   }, после него вставляем следующий код
    butToPDF.addActionListener {   //слушатель нажатия кнопки
        fileDialog.mode = FileDialog.SAVE   //диалог в режим сохранения
        fileDialog.title = "Сохранить в PDF"  //заголовок диалога сохранения
        fileDialog.setFile("*.pdf")  //фильтр для файлов
        fileDialog.isVisible = true  //показываем диалог сохранения
    //если пользователь выбрал каталог и файл, т.е. они не содержат null
    //это нужно, чтоб обработать отказ от сохранения, иначе будет ошибка
        if (!(fileDialog.directory+fileDialog.file).contains("null")) {
            val fileName = fileDialog.directory+fileDialog.file //записываем путь к файлу
            println("fileName=$fileName") //выводим полное имя файла в консоль

            val doc = PDDocument()  //создаем pdf-документ
            val page = PDPage()      //создаем страницу для документа
            doc.addPage(page)        //добавляем страницу в документ

            val contentStream = PDPageContentStream(doc, page) //создаем поток для вывода в документ
            contentStream.beginText()                   //начинаем работу с потоком
    //устанавливаем шрифт для документа, этот файл должен быть в корне проекта (есть на сервере)
        val font = PDType0Font.load(doc, File("arial.ttf"))
        contentStream.setFont(font, 10f)   //задаем размер шрифта
        contentStream.setLeading(24.5f)   //устанавливаем межстрочный интервал
        contentStream.newLineAtOffset(25f, 725f) //задаем позицию для первой строки,
                                                 //относительно нижнего левого края документа
            garage.getAll().forEach {  //цикл по содержимому гаража
    //записываем в переменную text содержимое каждого элемента гаража, причем заменяем переводы строки на
    //пробелы, а табы – на пустой символ, это нужно для корректного вывода в pdf-файл
                val text = it.toString().replace("\n", " ", true).replace("\t", "", true)
                println("text = $text") //выводим текст в консоль для проверки
                contentStream.showText(text) //добавляем текст из переменной в поток вывода
                contentStream.newLine()       //добавляем переход на новую строку
            }
            contentStream.endText()  //окончание работы с текстом
            println("Content to pdf added")
            contentStream.close()    //закрываем поток вывод
            doc.save(fileName)       //сохраняем документ
            doc.close()     //закрываем документ
            println("end pdf")
        }
    } //после этого оставшийся код из прошлой лабораторной
```

В итоге последовательность действий для сохранения данных в pdf-файл должна выглядеть примерно как на рис. 9:

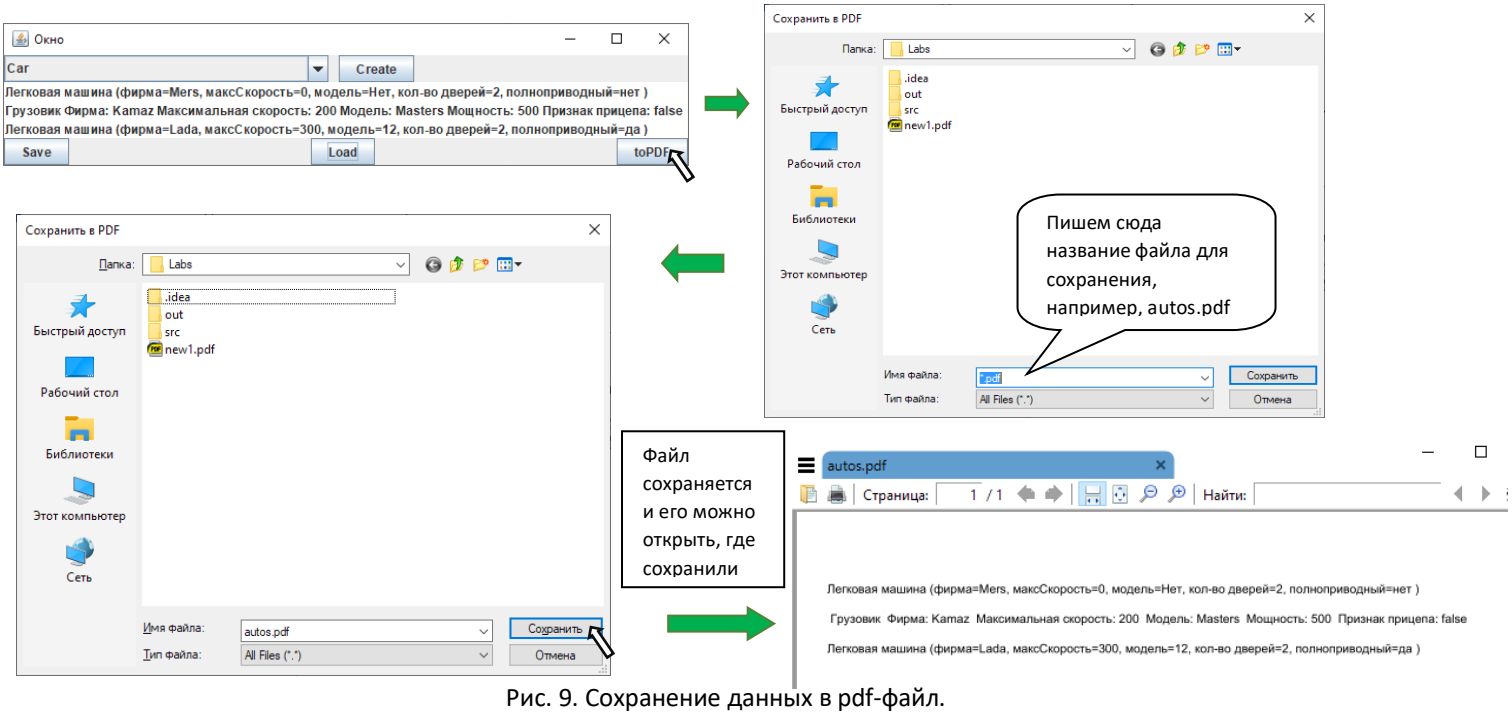


Рис. 9. Сохранение данных в pdf-файл.

Задание

Добавить к программе, сделанной по своему варианту в ЛР №5, возможность передачи данных в pdf-файл.

Лабораторная работа №7. Работа с изображениями

Реализуем в своем проекте возможность добавления изображения к объекту в списке. Для показа изображения можно использовать объект класса JLabel с объектом класса ImageIcon внутри. Например, сделаем возможность добавления картинки к созданному объекту. Для этого добавим к базовому классу Auto строковое поле image, установив значение по умолчанию "no_picture.png":

```
open class Auto(var firm: String = "Без названия", var maxSpeed: Int = 0, var image: String = "no_picture.png")
```

Также поменяем классы-наследники, чтобы они тоже могли содержать путь к файлу с изображением:

```
class Car(firm: String = "No", maxSpeed: Int = 0, image: String = "no_picture.png", var model: String = "Нет",
          var numDoors: Int = 4, var fullTime: Boolean = false): Auto(firm, maxSpeed, image)
class Truck (firm: String = "No",
             maxSpeed: Int = 0,
             image: String = "no_picture.png",
             var model: String = "Нет",
             var power: Int = 0,
             var trailer: Boolean = false) : Auto(firm, maxSpeed, image){
```

Далее внесем изменения в файл с созданием интерфейса:

- добавим поле

```
var currentImageFile = "no_picture.png"
```

перед функцией main;

- в обработчик нажатия butCreate.addActionListener в самое начало вставляем строку

```
currentImageFile = "no_picture.png"
```

- в функцию fillNorthBoxDown перед return вставим следующие строки:

```
var jLabel = JLabel() //создаем лейбл
loadImage(jLabel,"no_picture.png") //вызываем свою функцию для загрузки картинки в лейбл
northBoxDown.add(jLabel) //добавляем лейбл на форму
```

```
//далее делаем слушатель нажатия на лейбл, чтобы можно было поменять картинку
//при щелчке мыши по лейблу появляется диалог открытия файла, можно выбрать нужную картинку
jLabel.addMouseListener(object : MouseAdapter() {
    override fun mousePressed(e: MouseEvent?) {
        super.mousePressed(e)
// northBoxDown.parent.parent.parent.parent.parent – это основное окно
        var fileDialog = FileDialog(northBoxDown.parent.parent.parent.parent.parent as JFrame)
        fileDialog.mode = FileDialog.LOAD
        fileDialog.title = "Открыть изображение"
        fileDialog.isVisible = true
        val fileName = fileDialog.directory+fileDialog.file
        println(fileName)
        loadImage(jLabel, fileName) //загружаем картинку на лейбл
        currentImageFile = fileName
    }
})
```

- добавляем новую функцию для загрузки изображения в лейбл

```
fun loadImage(label: JLabel, imagePath: String) { // параметры – сам лейбл и путь к файлу с изображением
    val icon = ImageIcon(imagePath) //создаем объект класса ImageIcon с файлом внутри
    val width  = icon.iconWidth //получаем размеры изображения
    val height = icon.iconHeight
    val imageScale = width / height.toFloat()  //вычисляем соотношение ширины к высоте
    val newWidth = 70 //устанавливаем нужное значение ширины картинки
    val newHeight = (newWidth / imageScale).toInt() //вычисляем значение высоты с учетом соотношения
//задаем параметр icon лейбла – картинки внутри лейбла
    label.icon = ImageIcon(icon.image.getScaledInstance(newWidth, newHeight, Image.SCALE_SMOOTH))
}
```

- в метод buildCenter в обработчик нажатия listOfAuto.addMouseListener после строки

```
var tempAuto = tempList.selectedValue
```

добавим строчку `currentImageFile = tempAuto.image`

- метод `butToPDF.addActionListener` переписываем в следующем виде, чтоб добавлять картинки в pdf-файл (новый код выделен желтым, но удаленный текст не видно ☺):

```
butToPDF.addActionListener { //слушатель нажатия кнопки
    fileDialog.mode = FileDialog.SAVE //диалог в режим сохранения
    fileDialog.title = "Сохранить в PDF" //заголовок диалога сохранения
    fileDialog.setFile("*.pdf") //фильтр для файлов
    fileDialog.isVisible = true //показываем диалог сохранения
    if (!(fileDialog.directory+fileDialog.file).contains("null")) {
        val fileName = fileDialog.directory+fileDialog.file //записываем путь к файлу
        println("fileName=$fileName") //выводим полное имя файла в консоль
        val doc = PDDocument() //создаем pdf-документ
        val page = PDPage() //создаем страницу для документа
        doc.addPage(page) //добавляем страницу в документ
        val contentStream = PDPageContentStream(doc, page) //создаем поток для вывода в документ
        contentStream.beginText() //начинаем работу с потоком
//устанавливаем шрифт для документа, этот файл должен быть в корне проекта (есть на сервере)
        val font = PDType0Font.load(doc, File("arial.ttf"))
        contentStream.setFont(font, 10f) //задаем размер шрифта
        contentStream.setLeading(24.5f) //устанавливаем межстрочный интервал
        contentStream.endText() // заканчиваем работу с текстом
        var i : Float = 0f // счетчик, понадобится ниже
        garage.getAll().forEach { //цикл по содержимому гаража
            val text = it.toString().replace("\n", " ", true).replace("\t", "", true)
            println("text = $text") //выводим текст в консоль для проверки
            contentStream.beginText() // начинаем работу с текстом
            val y: Float = 725f-150f*(i++) //высчитываем смещение для текста от нижнего края страницы
                                           //с учетом номера текущей записи
            contentStream.newLineAtOffset(25f, y) //задаем позицию для строки
            contentStream.showText(text) //добавляем текст из переменной в поток вывода
            contentStream.newLine() //добавляем переход на новую строку
            contentStream.endText() //заканчиваем работу с текстом
            val pdImage = PDImageXObject.createFromFile(it.image, doc) //объект для картинки в pdf
            val width  = pdImage.width  //далее код для сохранения пропорции картинки при изменении
            val height = pdImage.height  //размера
            val imageScale = width / height.toFloat()
            val newHeight = 100f
            val newWidth = (newHeight * imageScale)
//вставляем картинку в pdf-документ, второй и третий параметр – отступ от нижнего левого края страницы
//по x и y соответственно, отступ по y высчитываем относительно предыдущего текста
```

```
                contentStream.drawImage(pdImage, 150F, y-130, newWidth, newHeight)
            }
            println("Content to pdf added")
            contentStream.close() //закрываем поток вывод
            doc.save(fileName) //сохраняем документ
            doc.close() //закрываем документ
            println("end pdf")
        }
    }
```

После проведенных изменений взаимодействие с программой может выглядеть следующим образом:

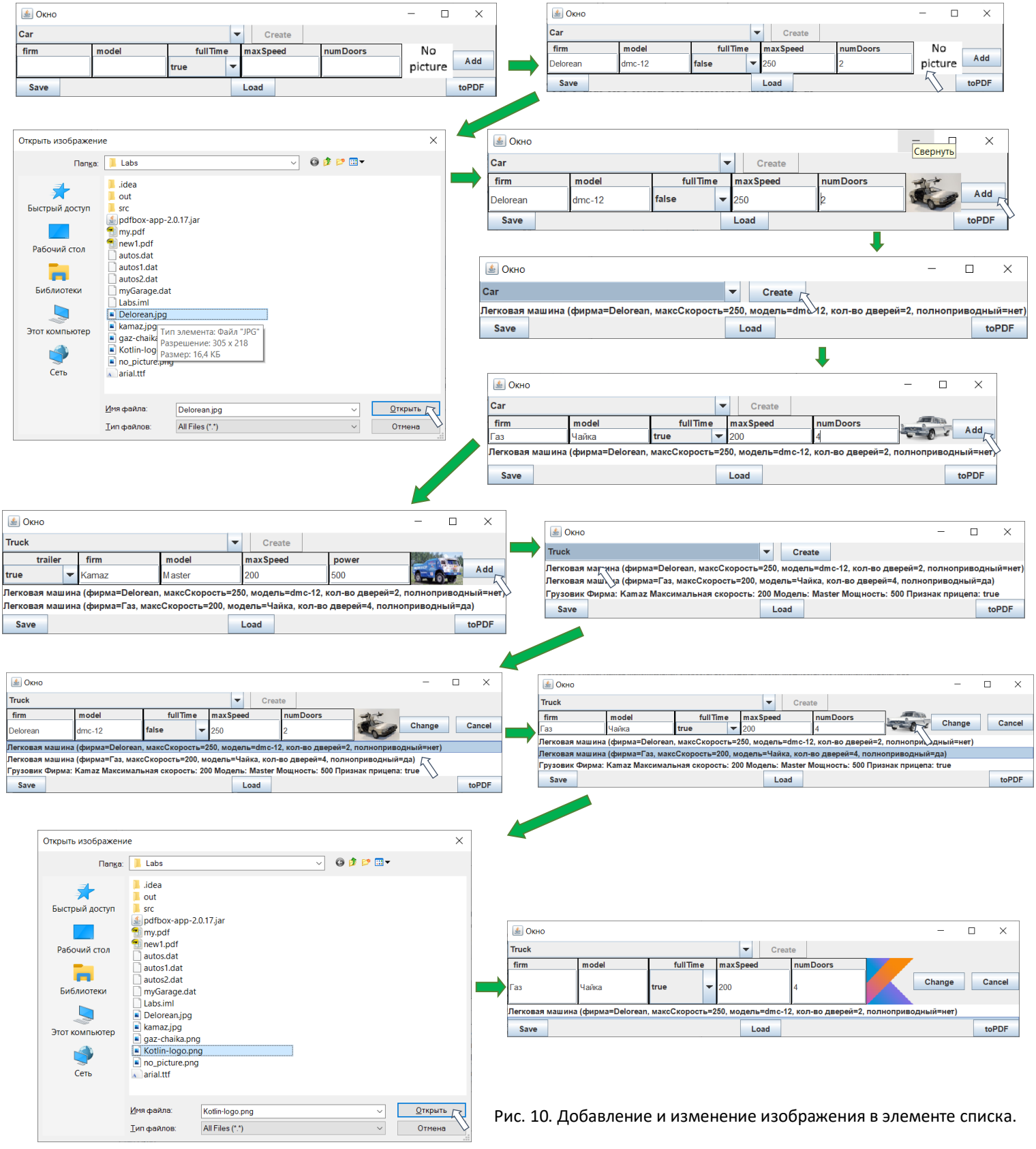


Рис. 10. Добавление и изменение изображения в элементе списка.

А если нажать на кнопку сохранения в pdf, то получится примерно следующее содержание в pdf:

Легковая машина (фирма=DeLorean, максСкорость=250, модель=DMC-12, кол-во дверей=2, полноприводный=нет)

Легковая машина (фирма=Газ, максСкорость=200, модель=Чайка, кол-во дверей=4, полноприводный=да)

Задание

Добавить к своему проекту изображения с возможностью загружать из файла и менять на другие. Исправить сохранение в pdf, чтоб сохранялись и изображения тоже.

Лабораторная работа №8. Kotlin и MS Office

Еще одна полезная возможность любого проекта – сохранение данных в формате MS Office, например, в формате MS Word или MS Excel. Добавим к своему проекту возможность сохранения данных в файл формата MS Word. Для этого можно воспользоваться библиотекой Apache POI. Актуальная версия библиотеки находится по адресу http://apache-mirror.rbc.ru/pub/apache/poi/release/bin/ в виде архива. Минимальным набором для реализации базовых функций являются следующие файлы (с учетом версии):

poi-3.14-20160307.jar
poi-examples-3.14-20160307.jar
poi-excelant-3.14-20160307.jar
poi-ooxml-3.14-20160307.jar
poi-ooxml-schemas-3.14-20160307.jar
poi-scratchpad-3.14-20160307.jar
xmlbeans-2.6.0.jar

В архиве эти файлы должны быть в корне и в каталогах lib и ooxml-lib, переписываем их оттуда в отдельную папку в свой проект. Чтобы библиотеку можно было использовать в проекте, необходимо подключить созданную папку с указанными файлами к проекту (как в лаб.раб. 6).

Разберем сохранение данных в формате docx на примере:

```
fun toWordDocx(str: String){ //функция для записи переданной строки в docx-файл
    val document = XWPFDocument() //создаем документ Word в формате docx
    val paragraph = document.createParagraph() //создаем абзац в документе
    val run = paragraph.createRun() //создаем объект для записи в полученный ранее абзац
    run.setText(str) //пишем текст
    run.addBreak() //переход на новую строку
    val icon = ImageIcon("Kotlin-logoe.png") //Объект ImageIcon для временной загрузки изображения
    val width  = icon.iconWidth  //чтоб получить реальные размеры картинки для вычисления
    val height = icon.iconHeight  //соотношения сторон
    val imageScale = width / height.toFloat()
    val newHeight = 100.0
    val newWidth = (newHeight * imageScale)
    val inStream = FileInputStream("Kotlin-logo.png") //поток для загрузки картинки в docx
    //вставляем картинку в документ
    run.addPicture(inStream, XWPFDocument.PICTURE_TYPE_PNG, "Kotlin-logo.png", Units.toEMU(newWidth),
Units.toEMU(newHeight))
    inStream.close() //закрываем поток
```

```
    val file = File("garageToWord.docx") //создаем файл
    val out = FileOutputStream(file) //создаем файловый поток вывода с новым файлом
    document.write(out) //пишем в файл из созданного объекта
    out.close() // закрываем поток вывода
    document.close() //закрываем документ
    println("docx written successfully")
}
```

И если вызвать данную функцию и передать ей в качестве параметра “Hello world”, то будет создан файл garageToWord.docx со следующим содержимым:

Теперь создадим функцию, которая будет принимать в качестве параметров объект класса GarageAuto и путь к файлу в формате docx и записывать в этот файл данные из переданного объекта:

```
fun garageToWord(garage: GarageAuto, fileName: String){
    val document = XWPFDocument() //создаем документ Word
    val paragraph = document.createParagraph() //создаем абзац в документе
    val run = paragraph.createRun() //создаем объект для записи в полученный ранее абзац
    garage.getAll().forEach{ //цикл по всем объектам массива в гараже
        run.setText(it.toString()) //записываем строковое представление объекта в документ
        run.addBreak() //переход на новую строку
        val icon = ImageIcon(it.image) //Объект ImageIcon для временной загрузки изображения из объекта
        val width  = icon.iconWidth //чтоб получить реальные размеры картинки для вычисления
        val height = icon.iconHeight //соотношения сторон
        val imageScale = width / height.toFloat()
        val newHeight = 200.0 //задаем желаемую высоту картинки
        val newWidth = (newHeight * imageScale)  //получаем соответствующую ширину картинки
        val inStream = FileInputStream(it.image) //поток для загрузки картинки в docx
        //вставляем картинку в документ
        run.addPicture(inStream, XWPFDocument.PICTURE_TYPE_PNG, it.image, Units.toEMU(newWidth),
Units.toEMU(newHeight))
        inStream.close() //закрываем поток с картинкой
        run.addBreak() //переход на новую строку
    }
    val file = File(fileName) //создаем файл
    val out = FileOutputStream(file) //создаем файловый поток вывода с новым файлом
    document.write(out) //пишем в файл из созданного объекта
    out.close() // закрываем потоки вывода
    document.close() //закрываем документ
    println("garage has been written to word file successfully")
}
```

Далее в основной файл проекта с интерфейсом добавляем в нижнюю панель кнопку с текстом toWord и прописываем обработчик нажатия на эту кнопку:

```
    butToWord.addActionListener{
        fileDialog.mode = FileDialog.SAVE //диалог в режим сохранения
        fileDialog.title = "Сохранить в word file" //заголовок диалога сохранения
        fileDialog.setFile("*.docx") //фильтр для файлов
        fileDialog.isVisible = true //показываем диалог сохранения
//проверяем, что пользователь выбрал каталог и файл, т.е. они не содержат null, это нужно,
//чтоб обработать отказ от сохранения, иначе будет ошибка
        if (!(fileDialog.directory+fileDialog.file).contains("null")) {
            var fileName = fileDialog.directory+fileDialog.file
  //проверяем, что имя файла содержит docx, и добавляем это расширение, если его нет в файле
            if (!fileName.contains(".docx")) fileName = fileName.plus(".docx")
            garageToWord(garage, fileName) //вызываем ранее созданную функцию с нужными параметрами
        }
    }
```

В итоге, если все сделано правильно, получится файл с расширением docx с примерно следующим содержимым (в зависимости от созданных в программе объектов):

Легковая машина (фирма=Delorean, максСкорость=250, модель=DMC-12, кол-во дверей=2, полноприводный=да)

Грузовик Фирма: Kamaz Максимальная скорость: 200 Модель: Master Мощность: 500 Признак прицепа: true

Теперь рассмотрим возможность создания файла презентации в формате MS PowerPoint (pptx). Создадим функцию для вывода данных из массива машин в презентацию:

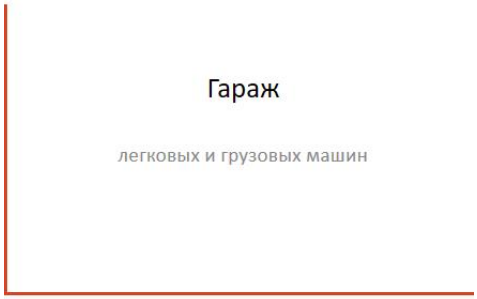
```
fun garageToPPTX(garage: GarageAuto, fileName: String){
    val ppt = XMLSlideShow() //создаем документ с презентацией
    val slideMaster = ppt.slideMasters[0] //объект для видов компоновок
    val titleLayout = slideMaster.getLayout(SlideLayout.TITLE) //компоновка титульного слайда

    val slide0 = ppt.createSlide(titleLayout) //создаем титульный слайд с выбранной компоновкой
    val title1 = slide0.getPlaceholder(0) //получаем доступ к основному заголовку титульного слайда
    title1.text = "Гараж"; //делаем свой заголовок
    val title2 = slide0.getPlaceholder(1) //получаем доступ к подзаголовку титульного слайда
    title2.clearText() //очищаем его
    title2.text = "легковых и грузовых машин" //делаем свой подзаголовок
    //далее в цикле по объектам массива гаража будем создавать отдельный слайд для каждого объекта
    garage.getAll().forEach {
        val tempAuto = it //сохраняем ссылку на очередной объект массива
        //получаем объект для компоновки слайда с заголовком и содержимым
        val slideLayout = slideMaster.getLayout(SlideLayout.TITLE_AND_CONTENT);
        val slide1 = ppt.createSlide(slideLayout) //создаем слайд с выбранной компоновкой
        val title = slide1.getPlaceholder(0) //получаем доступ к заголовку слайда
        val jClassStr = tempAuto.javaClass.kotlin //выясняем класс текущего объекта
        println("class = ${jClassStr}") //выводим название класса в консоль для проверки
//обычно название класса представлено в виде "class Пакет.Имя", нам нужно только имя, поэтому нужно взять
        val objClass = jClassStr.toString().split('.').last() //слово после последней точки
        title.text = objClass //задаем полученное имя заголовку слайда
        val content: XSLFTextShape = slide1.getPlaceholder(1) //получаем доступ к основному телу слайда
        content.clearText() //очищаем содержимое основного тела слайда
    //получаем список полей класса текущего объекта и делаем цикл по списку
        jClassStr.memberProperties.forEach {
            println("\t${it.name}: ${it.get(tempAuto)}") //выводим имя текущего поля и его значение для
                                                        //текущего объекта
            if (it.name!="image") {  //если поле - не изображение
                val p1 = content.addNewTextParagraph() //создаем текстовый объект в виде списка
                p1.indentLevel = 0 //задаем параметры списка
                p1.isBullet = true //чтобы он был маркированный
                val r1 = p1.addNewTextRun() //берем у списка объект для добавления нового элемента
                r1.setText("${it.name}: ${it.get(tempAuto)}")  //вставляем текст в элемент списка
            } else {  //иначе, если поле - изображение
//создаем байтовый массив, в него загружаем изображение из файла, который берем по значению в поле image
                val picture: ByteArray = IOUtils.toByteArray(FileInputStream("${it.get(tempAuto)}"))
        //создаем объект для добавления изображения на слайд
```

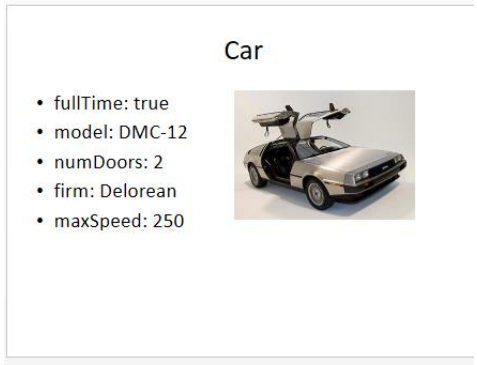
```
                val idx = ppt.addPicture(picture, PictureData.PictureType.PNG)
                var pic = slide1.createPicture(idx) //вставляем объект с изображением на слайд
//создаем объект ImageIcon для временной загрузки изображения из поля объекта
                val icon = ImageIcon("${it.get(tempAuto)}")
                val width  = icon.iconWidth //чтоб получить реальные размеры картинки для вычисления
                val height = icon.iconHeight //соотношения сторон
                val imageScale = width / height.toFloat()
                val newHeight = 200 //задаем желаемую высоту картинки
                val newWidth = (newHeight * imageScale)  //получаем соответствующую ширину картинки
         // задаем параметры изображения на слайде: положение по x и y, ширина, высота
                pic.anchor = Rectangle(350, 130, newWidth.toInt(), newHeight);
            }
        }
    }
    val file = File(fileName) //создаем файл, в который будет сохранена презентация
    val out = FileOutputStream(file)  //создаем файловый поток вывода с новым файлом
    ppt.write(out)  //пишем в файл созданную презентацию
    out.close()  //закрываем поток
    ppt.close() //закрываем документ
    println("Done writing pptx")
}
```

Для вызова данной функции добавим к интерфейсу программы новую кнопку и обработчик нажатия:

```
    butToPPTX.addActionListener{
        fileDialog.mode = FileDialog.SAVE //диалог в режим сохранения
        fileDialog.title = "Сохранить в pptx" //заголовок диалога сохранения
        fileDialog.setFile("*.pptx") //фильтр для файлов
        fileDialog.isVisible = true //показываем диалог сохранения
//если пользователь выбрал каталог и файл, т.е. они не содержат null
//это нужно, чтоб обработать отказ от сохранения, иначе будет ошибка
        if (!(fileDialog.directory+fileDialog.file).contains("null")) {
            var fileName = fileDialog.directory+fileDialog.file
            if (!fileName.contains(".pptx")) fileName = fileName.plus(".pptx")
            garageToPPTX(garage, fileName)
        }
    }
```

И в итоге должна получиться презентация с примерно следующими слайдами:





Задание

Добавить к своему проекту возможность вывода данных в файлы Word и PowerPoint.

Лабораторная работа №9. Технология Drag&Drop

Википедия говорит следующее:

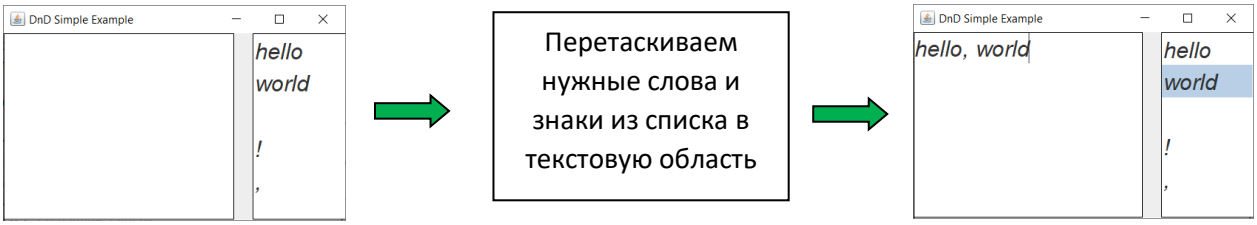
Drag-and-drop (в переводе с английского означает буквально тащи-и-бросай; Бери-и-Брось) — способ оперирования элементами интерфейса в интерфейсах пользователя (как графическим, так и текстовым, где элементы GUI реализованы при помощи псевдографики) при помощи манипулятора «мышь» или сенсорного экрана.

Способ реализуется путём «захвата» (нажатием и удержанием главной (первой, чаще левой) кнопки мыши) отображаемого на экране компьютера объекта, программно доступного для подобной операции, и перемещении его в другое место (для изменения расположения) либо «бросания» его на другой элемент (для вызова соответствующего, предусмотренного программой, действия).

Попробуем сделать интерфейс с возможностью drag&drop:

```
fun main(){
    val window = JFrame("DnD Simple Example") //создаем окно
    window.defaultCloseOperation = JFrame.EXIT_ON_CLOSE
    window.size = Dimension(500, 300) //размер окна

    val tArea = JTextArea()  //создаем текстовую область
    val blackLine = BorderFactory.createLineBorder(Color.black) //создаем объект для черной границы
    tArea.border = blackLine //задаем текстовой области границу
    val myFont = Font("Times", Font.ITALIC, 25) //создаем объект для шрифта
    tArea.font = myFont //устанавливаем шрифт для текстовой области
    tArea.dragEnabled = true  //разрешаем использование drag&drop для текстовой области
    val list = JList<String>() //создаем список
    list.dragEnabled = true //разрешаем использование drag&drop для списка
    list.preferredSize = Dimension(100, 100)  //задаем предпочитаемый размер для списка
    list.font = myFont  //устанавливаем шрифт для списка
    list.border = blackLine  //задаем списку границу
    val lModel = DefaultListModel<String>()  //создаем модель данных для списка
    list.model = lModel  //устанавливаем модель для списка
    lModel.addAll(arrayListOf("hello", "world", " ", "!", ","))  //вставляем данные в модель
    //создаем компоновку BorderLayout с нужными параметрами отступа между объектами
    val borderLay = BorderLayout(20,20)
    window.layout = borderLay //устанавливаем компоновку на окно
    window.add(tArea, BorderLayout.CENTER)  //вставляем объекты в окно
    window.add(list, BorderLayout.EAST)
    window.setLocationRelativeTo(null)
    window.isVisible = true
}
```

В итоге должно получиться приложение с примерно следующим функционалом и внешним видом:

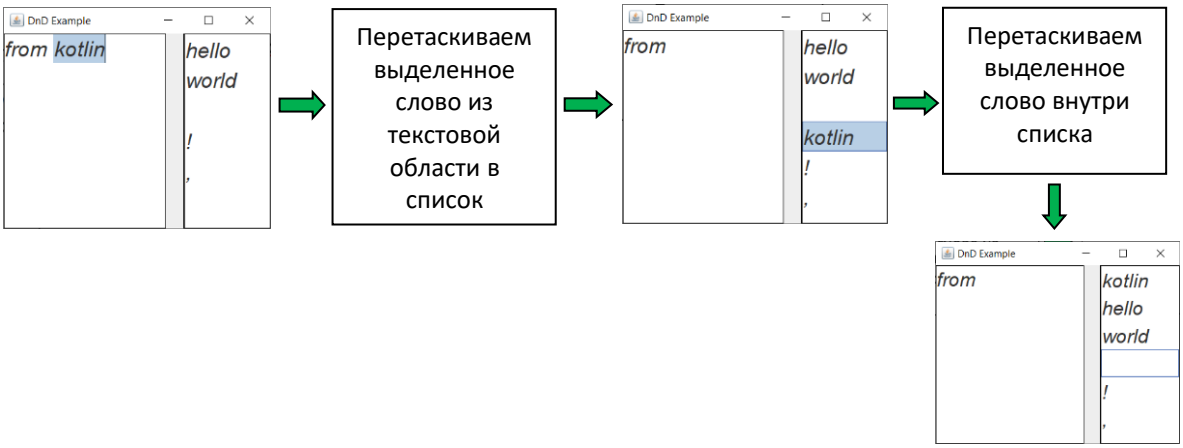


Но в обратную сторону пока что работать не будет: мы не сможем перетаскивать текст из текстовой области в список. Чтобы это исправить, понадобится вставить следующий код:

```
list.dropMode = DropMode.INSERT  //устанавливаем режим приема данных при d&d - вставка эл-та в список
list.setTransferHandler(object : TransferHandler() { //задаем обработчик приема/передачи объекта при d&d
    private var index = 0 //индекс текущего элемента
    private var beforeIndex = false //переменная, отвечающая за признак перемещения объекта - перед или
                                    //после текущего выбора

//перегружаем метод, определяющий действие при d&d (копирование или перемещение)
    override fun getSourceActions(comp: JComponent): Int {
```

```
            return MOVE  //выбираем Перемещение
        }

    //перегружаем метод для выбора эл-та для передачи при d&d
        override fun createTransferable(comp: JComponent): Transferable {
            index = list.getSelectedIndex() //получаем индекс выделенного эл-та списка (текущий номер)
            return StringSelection(list.getSelectedValue()) //возвращаем сам выбранный при d&d объект списка
        }
    //перегружаем метод для действия после окончания вставки объекта, чтобы удалить старый объект
        override fun exportDone(comp: JComponent, trans: Transferable, action: Int) {
            if (action == MOVE) { //если выбран способ Перемещение
            //то смотрим на признак индекса перемещения, если новый объект появится перед текущим,
              if (beforeIndex) lModel.remove(index + 1) //то удаляем объект с номером на 1 больше от текущего
              else lModel.remove(index) // иначе удаляем объект с текущим номером
            }
             beforeIndex = false //сбрасываем признак перемещения объекта
        }

    //перегружаем метод для разрешения импорта объекта
        override fun canImport(support: TransferSupport): Boolean {
            return support.isDataFlavorSupported(DataFlavor.stringFlavor)
        }

        override fun importData(support: TransferSupport): Boolean { //перегружаем метод для вставки объекта
            //получаем объект для вставки в список
            val s = support.transferable.getTransferData(DataFlavor.stringFlavor) as String
            //получаем объект, представляющий место вставки в список
            val dl = support.dropLocation as JList.DropLocation
            lModel.add(dl.index, s) //вставляем объект в список
            //устанавливаем новое значение для признака перемещения объекта - перед или после текущего выбора
            beforeIndex = if (dl.index < index) true else false
            return true
        }
    })
```

В результате получаем пример полноценного Drag&Drop приложения:



Задание

Сделать приложение с возможностью Drag&Drop по вариантам:

1. 2 списка – 1 с числами, другой – с буквами.
2. 2 списка – 1 с гласными, другой – с согласными.
3. Текстовое поле и список.
4. Панель и список, элементы при перетаскивании из списка превращаются в чек-боксы.
5. Панель и список, элементы при перетаскивании из списка превращаются в радиокнопки.