Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Кафедра инженерной психологии и эргономики

­­­­­

Современные языки программирования

Отчет по лабораторной работе №1

«Использование языка программирования Kotlin»

Выполнил: Кулик И.М.

Студент группы 310902

Преподаватель: Усенко Ф. В.

Минск 2024

Цель: Выполнить разработку приложения с использованием языка программирования Kotlin.

### Задание. Вариант 15

Добавить класс Гладиолус. Добавить класс Лепесток, принадлежащий классу Бутон и класс Лист, принадлежащий классу Цветок. Обеспечить создание Объектов Лепесток одновременно с объектами Бутон. Реализовать метод Засохнуть() у класса Цветок.

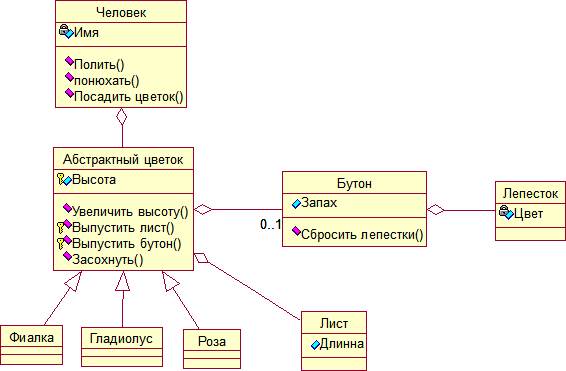


Рисунок 1 – Диаграмма классов

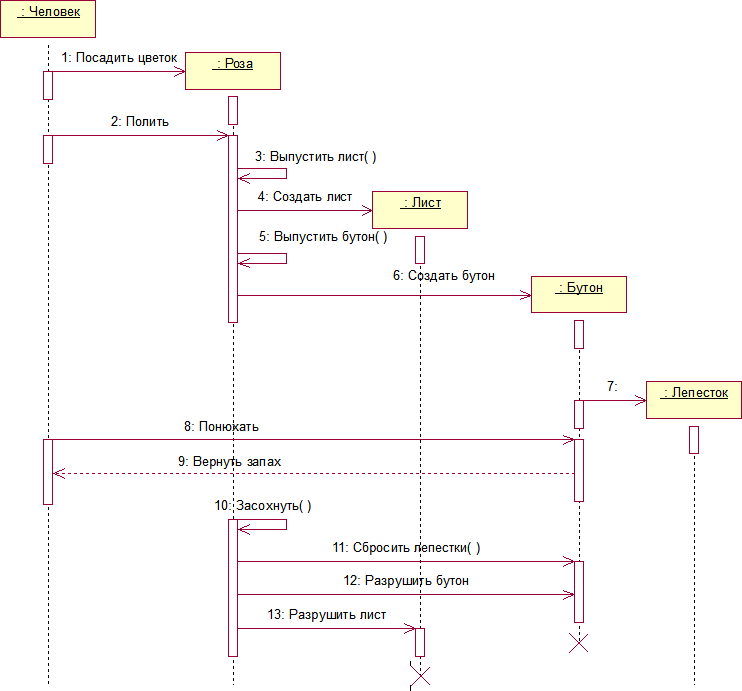


Рисунок 31 – Диаграмма последовательности

Листинг кода:

Файл Main.kt:

import libs.\*  
  
fun main() {  
 val persons: MutableList<Person> = *mutableListOf*()  
 val flowers: MutableList<Flower> = *mutableListOf*()  
  
 *println*("Enter the name")  
 val name = *readln*()  
 val person = Person(name)  
 persons.add(person)  
 val flower = person.plantFlower()  
 flowers.add(flower)  
  
 while (true) {  
 *println*("Choose the operation")  
 *println*("1. Pour the flower")  
 *println*("2. Smell the flower")  
 *println*("3. Dry up the flower")  
 *println*("4. Exit")  
 when (*readln*().*toInt*()) {  
 1 -> {  
 person.toPour(flower)  
 *println*("Bud, leaf and petal created for ${flower.getType()}")  
 }  
 2 -> {  
 if (flower.hasBudAndPetal()) {  
 person.toSmell(flower.bud!!)  
 } else {  
 *println*("The flower doesn't have a bud yet. Pour it first!")  
 }  
 }  
 3 -> {  
 flower.toDryUp()  
 *println*("${flower.getType()} has dried up and lost all its parts.")  
 }  
 4 -> break  
 else -> *println*("Invalid operation")  
 }  
 }  
}

Файл Flower.kt:

package libs  
  
abstract class Flower(protected var height: Int) {  
 var leaf: Leaf? = null  
 var bud: Bud? = null  
 var petal: Petal? = null  
  
 fun increaseHeight() {height+=3}  
 protected abstract fun createLeaf(): Leaf  
 protected abstract fun createBud(): Bud  
 protected abstract fun createPetal(): Petal  
 fun toDryUp() {  
 bud = null  
 petal = null  
 leaf = null  
 *println*("The flower has dried up.")  
 }  
 abstract fun getType(): String  
  
 fun hasBudAndPetal(): Boolean {  
 return bud != null && petal != null  
 }  
  
 fun pour() {  
 if (leaf == null) leaf = createLeaf()  
 if (bud == null) bud = createBud()  
 if (petal == null) petal = createPetal()  
 increaseHeight()  
 }  
}  
  
class Violet(height: Int) : Flower(height) {  
 override fun getType(): String {  
 return "Violet"  
 }  
  
 override fun createLeaf(): Leaf {  
 return Leaf(10)  
 }  
  
 override fun createBud(): Bud {  
 return Bud("Sweet")  
 }  
  
 override fun createPetal(): Petal {  
 return Petal("Purple")  
 }  
}  
  
class Gladiolus(height: Int) : Flower(height) {  
 override fun getType(): String {  
 return "Gladiolus"  
 }  
  
 override fun createLeaf(): Leaf {  
 return Leaf(15)  
 }  
  
 override fun createBud(): Bud {  
 return Bud("Fresh")  
 }  
  
 override fun createPetal(): Petal {  
 return Petal("Yellow")  
 }  
}  
  
class Rose(height: Int) : Flower(height) {  
 override fun getType(): String {  
 return "Rose"  
 }  
  
 override fun createLeaf(): Leaf {  
 return Leaf(12)  
 }  
  
 override fun createBud(): Bud {  
 return Bud("Fragrant")  
 }  
  
 override fun createPetal(): Petal {  
 return Petal("Red")  
 }  
}  
  
class Bud(var smell: String)  
class Petal(private var color: String)  
class Leaf(var length: Int)

Файл Person.kt:

package libs  
  
class Person(private var name: String) {  
 fun toPour(flower: Flower) {  
 *println*("You poured ${flower.getType()}")  
 flower.pour()  
 }  
  
 fun toSmell(bud: Bud) {  
 *println*("You smelled ${bud.smell} smell")  
 }  
  
 fun plantFlower(): Flower {  
 *println*("Choose the flower to plant")  
 *println*("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  
 *println*("1. Violet")  
 *println*("2. Gladiolus")  
 *println*("3. Rose")  
 val numberFlower = try {  
 *readLine*()!!.*toInt*()  
 } catch (e: Exception) {  
 *println*("Invalid input")  
 return plantFlower()  
 }  
 *println*("Enter the height")  
 val height = try {  
 *readLine*()!!.*toInt*()  
 } catch (e: Exception) {  
 *println*("Invalid input")  
 return plantFlower()  
 }  
 return when (numberFlower) {  
 1 -> Violet(height)  
 2 -> Gladiolus(height)  
 3 -> Rose(height)  
 else -> {  
 *println*("Invalid input")  
 plantFlower()  
 }  
 }  
 }  
}

Результат работы программы представлены на рисунках 1 – 5.

# 

Рисунок 1 – Результат выполнения программы

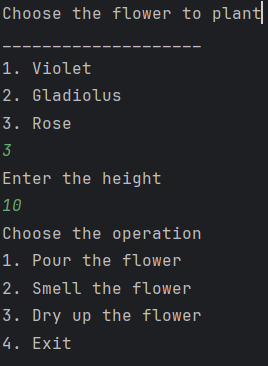


Рисунок 2 – Результат выполнения программы

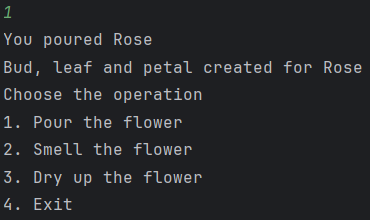


Рисунок 3 – Результат выполнения программы

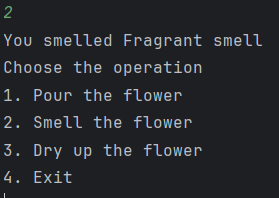


Рисунок 4 – Результат выполнения программы

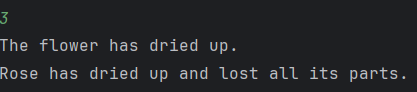


Рисунок 5 – Результат выполнения программы

Вывод: в ходе выполнения лабораторной работы номер 1 по языку программирования kotlin мы познакомились с основами языка и научились на практике применять основные его конструкции. Было потрачено время на изучение UML диаграмм для корректного понимая задания лабораторной рабыты. При выполнении возникли трудности с понииманием работы конструкторов в Kotlin, в остальном работа выполнялась гладко. Затрачено часов – 6.