

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образования «Полоцкий государственный университет»

Факультет информационных технологий
Кафедра технологий программирования

Лабораторная работа №4
по дисциплине: **«Объектно-ориентированные технологии программирования и
стандарты проектирования»**
на тему: **«ШАБЛОНЫ ФУНКЦИЙ И КЛАССОВ»**

ВЫПОЛНИЛ

студент группы 16 ИТ-3
Яблонский А.С.

ПРОВЕРИЛ

преподаватель
Ярошевич П.В.

Полоцк, 2018 г.

Вариант №10

Постановка задачи:

1. Получить практические навыки создания шаблонов и использования их в программах. Создать шаблонный класс и протестировать его с разными типами данных.

Реализация:

Реализовывать задание лабораторной работы было принято на языке программирования kotlin. Он поддерживает переопределение операторов и создание шаблонных классов, что необходимо для выполнения задания. Также он имеет гибкий удобный синтаксис, что способствует более эффективному написанию кода.

В результате работы был реализован шаблонный класс *MyList<T : Any>*. Данный класс способен работать с любыми not-null объектами языка kotlin.

Согласно заданию варианта были переопределены следующие операторы:

```
// *
operator fun times(myList: MyList<T>): MyList<T> {
    val resultList = MyList<T>()
    val bigger: MyList<T>
    val smaller: MyList<T>

    if (myList.size() > this.size()) {
        bigger = myList
        smaller = this
    } else {
        bigger = this
        smaller = myList
    }

    bigger.list.forEach { it: T
        if (smaller.list.contains(it)) {
            resultList.add(it)
        }
    }
    return resultList
}
```

оператор "" - пересечение множеств

```
operator fun compareTo(mList: MyList<T>): Int {  
    if (mList.size() == 0 || this.size() == 0) return 0  
    return if (list.containsAll(mList.list)) -1 else 0  
}
```

*оператор сравнения "<" - проверка на подмножество.

Методы были предопределены согласно официальной документации языка.

Тестирование:

Для тестирования использовалась библиотека JUnit.4. Код тестирования находится в классе MyListTest.kt.

Для тестирования корректности работы шаблонного класса MyList были реализованы следующие методы:

1. Тестирование основных операций над списком: добавление элемента, получение элемента по индексу, сравнение двух списков

```
@Test
fun addSizeTest() {
    val mList = MyList<Char>()
    mList.add('A')
    mList.add('B')
    mList.add('C')
    assertEquals( expected: 3, mList.size())
}
```

```
@Test
fun getTest() {
    val mList = MyList<Char>()
    mList.add('A')
    mList.add('B')
    mList.add('C')
    assertEquals( expected: 'A', mList[0])
    assertEquals( expected: 'B', mList[1])
    assertEquals( expected: 'C', mList[2])
}
```

```
@Test
fun equalsTest() {
    val mListA = MyList<Char>()
    mListA.add('A')
    mListA.add('B')
    mListA.add('C')

    val mListB = MyList<Char>()
    mListB.add('A')
    mListB.add('B')
    mListB.add('C')

    assertTrue( actual: mListA == mListB)
    assertFalse( actual: mListA != mListB)

    val mListC = MyList<Int>()
    mListC.add(1)
    mListC.add(2)
    mListC.add(3)
    assertFalse( actual: mListA == mListC)
}
```

2. Тестирование заданий варианта: оператор "<" - проверка на подмножество

```
@Test // < - проверка на подмножество.  
fun lowerThanListTest() {  
    val listA = MyList<Int>()  
    listA.add(111)  
    listA.add(222)  
    listA.add(333)  
    listA.add(444)  
    listA.add(555)  
  
    val listB = MyList<Int>()  
    listB.add(222)  
    listB.add(444)  
  
    assertEquals( expected: true,  actual: listA < listB)  
    assertEquals( expected: false, actual: listB < listA)  
    assertEquals( expected: false, actual: listA < MyList())  
}
```

Оператор ">" - проверка на принадлежность

```
@Test // > - проверка на принадлежность  
fun greaterThanItemTest() {  
    val listA = MyList<Int>()  
    listA.add(111)  
    listA.add(222)  
    listA.add(333)  
    listA.add(444)  
    listA.add(555)  
  
    assertEquals( expected: true,  actual: listA > 111)  
    assertEquals( expected: true,  actual: listA > 333)  
    assertEquals( expected: false, actual: listA > 1)  
    assertEquals( expected: false, actual: listA > 999)  
}
```

3. Тестирование списка со стандартными типами данных языка: Int, Boolean, Double

```
@Test
fun intMyListTest() {
    val mList = MyList<Int>()
    mList.add(2)
    mList.add(3)
    mList + 1

    assertEquals( expected: 3, mList.size())
    assertEquals( expected: 1, mList[0])
    assertEquals( expected: 2, mList[1])
    assertEquals( expected: 3, mList[2])
}
```

```
@Test
fun boolMyListTest() {
    val mList = MyList<Boolean>()
    mList.add(false)
    mList.add(false)
    mList + true

    assertEquals( expected: 3, mList.size())
    assertEquals( expected: true, mList[0])
    assertEquals( expected: false, mList[1])
    assertEquals( expected: false, mList[2])
}
```

```
@Test
fun doubleMyListTest() {
    val mList = MyList<Double>()
    mList.add(2.5)
    mList + 333.0
    mList.add(1.3)

    assertEquals( expected: 3, mList.size())
    assertEquals( expected: 333.0, mList[0])
    assertEquals( expected: 2.5, mList[1])
    assertEquals( expected: 1.3, mList[2])
}
```


4. Тестирование списка с собственными типами данных

@Test

```
fun myClassMyListTest() {  
    val mList = MyList<SomeClass>()  
    mList.add(SomeClass( id: 11))  
    mList.add(SomeClass( id: 22))  
    mList + SomeClass( id: 33)  
    mList + SomeClass( id: 44)  
  
    assertEquals( expected: 4, mList.size())  
    assertEquals(SomeClass( id: 44), mList[0])  
    assertEquals(SomeClass( id: 33), mList[1])  
    assertEquals(SomeClass( id: 11), mList[2])  
    assertEquals(SomeClass( id: 22), mList[3])  
}
```

@Test

```
fun companyClassMyListTest() {  
    val mList = MyList<Company>()  
    mList.add(Company( id: 11, name: "AAA"))  
    mList.add(Company( id: 22, name: "BBB"))  
    mList + Company( id: 33, name: "CCC")  
    mList + Company( id: 44, name: "DDD")  
  
    assertEquals( expected: 4, mList.size())  
    assertEquals(Company( id: 44, name: "DDD"), mList[0])  
    assertEquals(Company( id: 33, name: "CCC"), mList[1])  
    assertEquals(Company( id: 11, name: "AAA"), mList[2])  
    assertEquals(Company( id: 22, name: "BBB"), mList[3])  
}
```

В результате тестирования все тесты были пройдены без ошибок

▼	✓ MyListTest	56ms
✓	plusElementTest	44ms
✓	boolMyListTest	0ms
✓	addSizeTest	0ms
✓	getTest	0ms
✓	equalsTest	8ms
✓	companyClassMyListTest	1ms
✓	decTest	0ms
✓	myClassMyListTest	3ms
✓	doubleMyListTest	0ms
✓	intMyListTest	0ms

Вывод:

В результате выполнения данной лабораторной работы мне удалось получить практические навыки создания шаблонных классов и использование их в написании программ компонентов класса. Мне удалось написать программу, в которой, на основе одного класса, создается множество объектов, способных работать с разными типами данных.