**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ**

Лабораторная работа №2

по дисциплине

«Конструирования и тестирование программного обеспечения»

Группа: АВТ-043

Факультет: АВТФ

Студент: Абазбеков М., Завьялов К.И.

Преподаватель: Шперлинг В.К.

**Содержание**

[Цель работы 3](#_Toc150547723)

[Задание 3](#_Toc150547724)

[Ход работы 3](#_Toc150547725)

[Результаты работы 5](#_Toc150547726)

[Вывод 9](#_Toc150547727)

## Цель работы

Обеспечить возможность настройки приложения на работу с выбранным из списка типов данных

## Задание

Портировать из Java разработанный в л.р.1 программный код. Сортировку

реализовать с использованием техники функционального программирования и в

императивном стиле. Алгоритм сортировки:

• Рекурсивное разделение

• Рекурсивное слияние

• Циклическое слияние

• Однократное слияние

• Лексикографическая сортировка

• Способы реализации операции сравнения в шаблоне:

• переопределенная операция сравнения в шаблоне

• параметр метода сортировки – внешняя функция сравнения

## Ход работы

Вариант 6 (Двусвязный разомкнутый список):

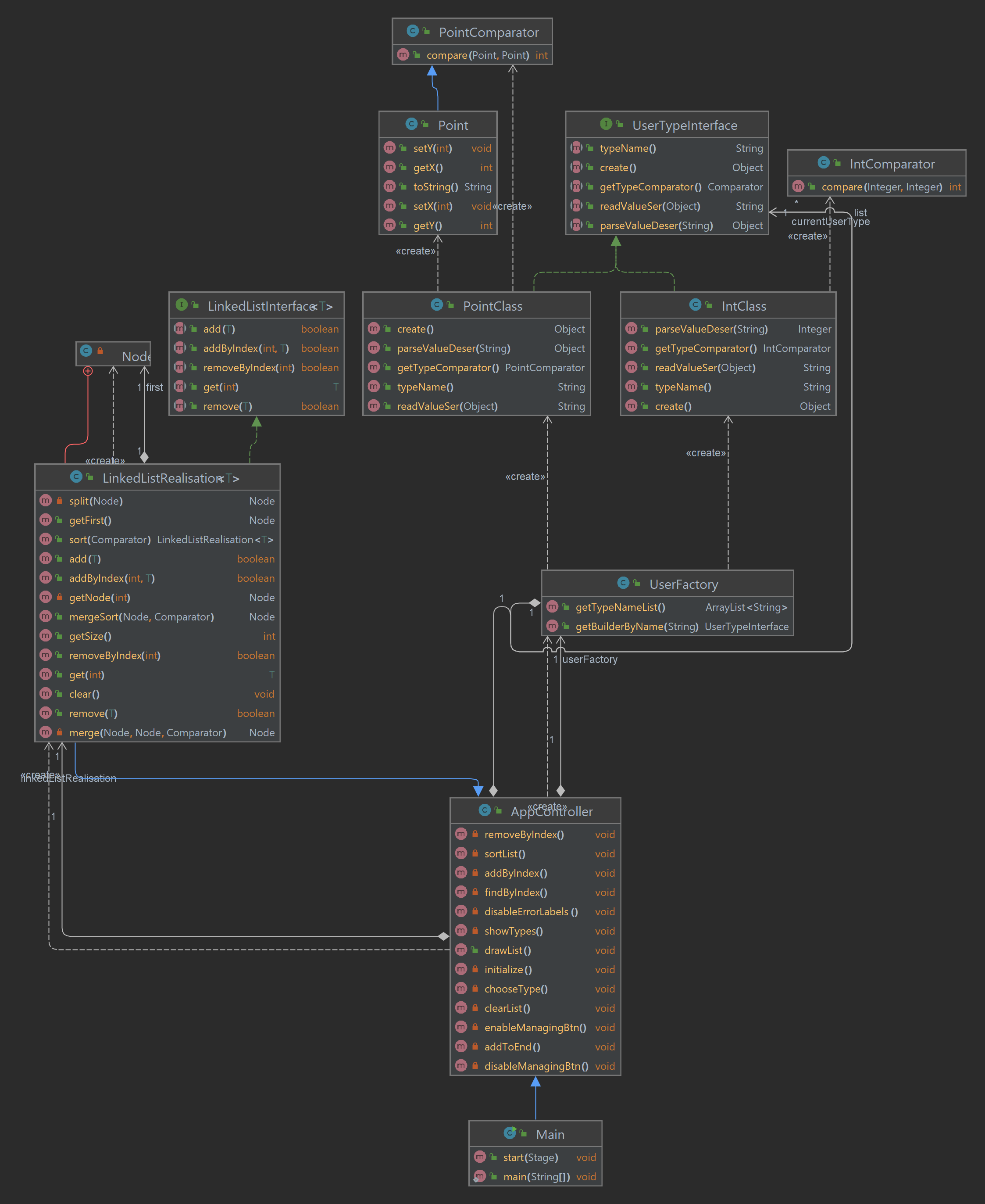


Рисунок 1 – Диаграмма классов и интерфейсов

На диаграмме классов видно, что в ходе данной лабораторной работы было разработано 2 интерфейса. Первый интерфейс содержит методы для работы с нашими моделями, такие как методы сериализации, десериализации, получения компараторов для моделей и т д. Второй интерфейс содержит методы для работы со списком. В классе списка есть методы добавления, удаления по индексу, добавления по индексу, поиска по индексу.

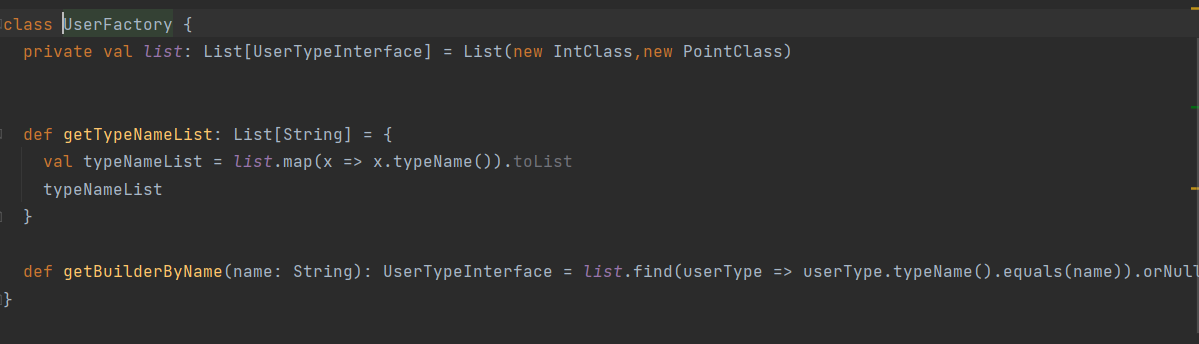


Рисунок 2 – Класс UserFactory

Для динамического отображения наших типов (Point и Int) в меню нашего приложения, был создан класс UserFactory, который обеспечивает возможность получить наши классы для работы с моделями по названию типа, а также получение всего списка строк наших типов.

В классе LinkedListRealisation, помимо методов для работы с нашей структурой данных, также есть методы для сортировки нашего двусвязного списка. Данные методы в качестве параметра получают объект класса компаратор и реализуют сортировку слиянием используя отдельные компараторы для каждого типа данных нашего варианта.

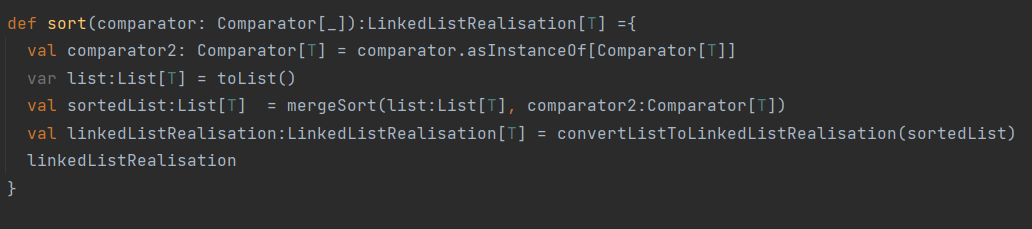


Рисунок 3 – Передача компаратора в метод сортировки.

Сортировка слияния реализована при помощи разделения нашей структуры данных на 2 части и при помощи компаратора сравнения наших типов данных. Сортировка выполнена в функциональном стиле программирования. Данные иммутабельны.

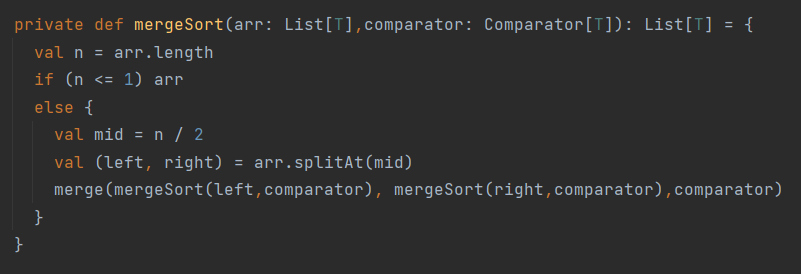


Рисунок 4 – Разбитие списка на 2 половины.

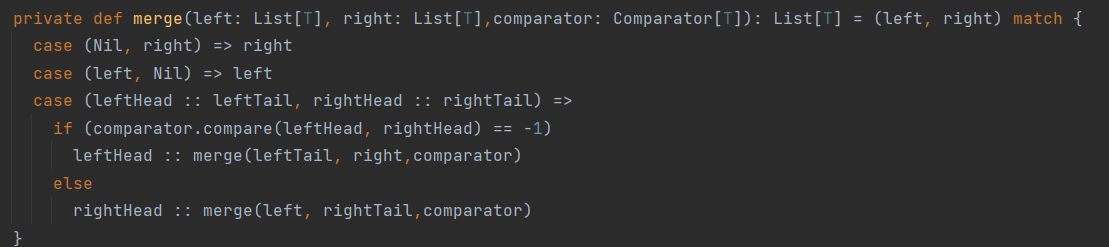


Рисунок 5 – Часть алгоритма сравнения при помощи компаратора.

Также была реализована загрузка данных из текстового файла, а также запись в этот файл. Файл содержит тип данных, хранимых в данной структуре, а также сами данные. Когда данные читаются из файла, первым действием происходит чтение типа данных. В зависимости от него происходит чтение определенных данных.

## Результаты работы

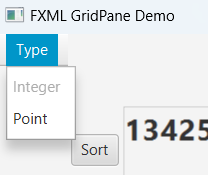


Рисунок 6 – Выбор типа данных из списка.

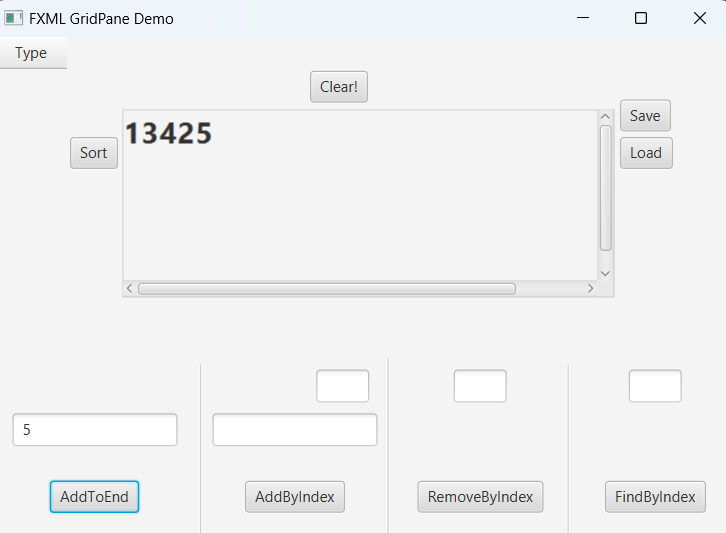


Рисунок 7 – Создание структуры данных для типа Int.

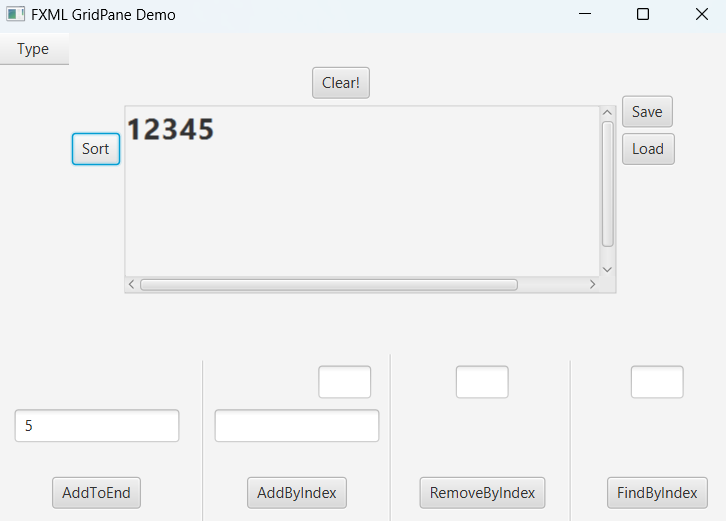


Рисунок 8 – Сортировка данных.

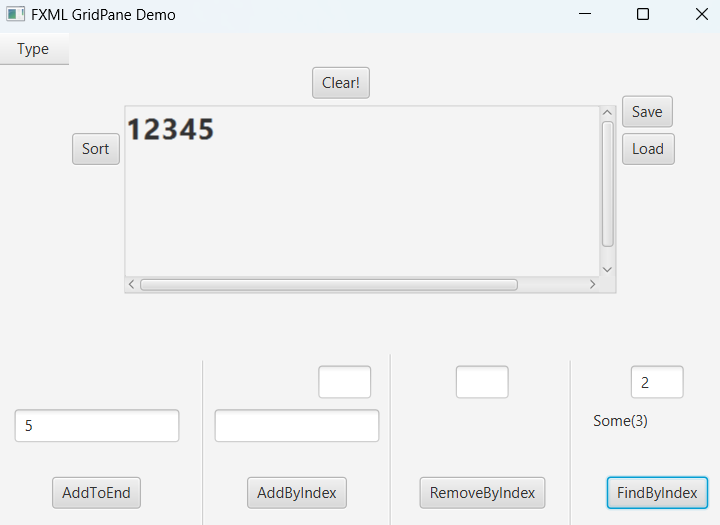


Рисунок 9 – Поиск по индексу.

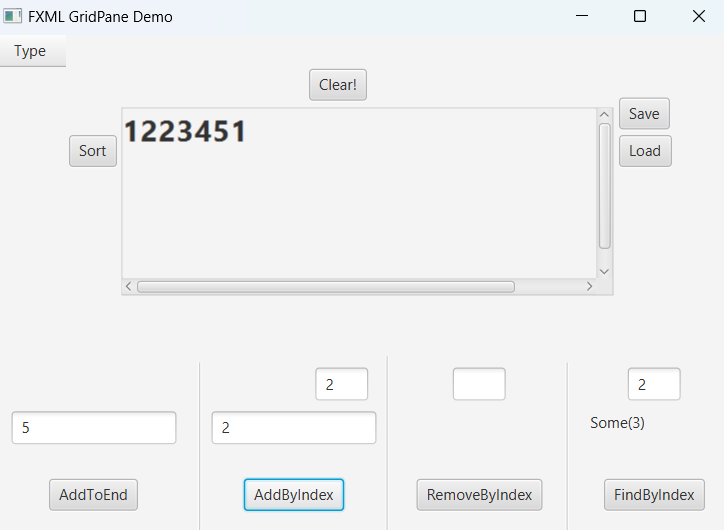


Рисунок 10 – Добавление по индексу.

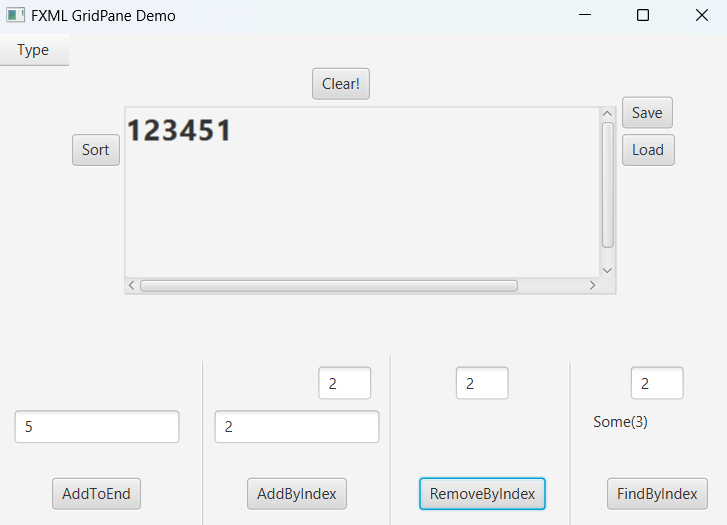


Рисунок 11 – Удаление по индексу.

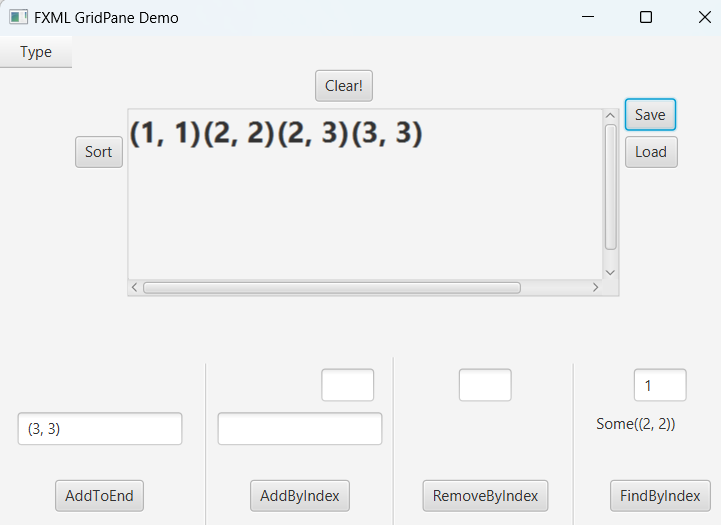


Рисунок 12 – Работа с типом Point.

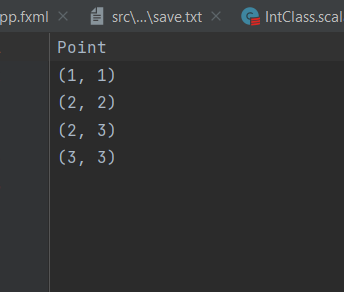


Рисунок 13 – Сохраненные данные в файл.

## Вывод

В ходе данной лабораторной работы мы познакомились с языком программирования Scala. Разработали структуру данных, работающую при помощи шаблона с разными типами данных, реализовали методы сортировки для сортировки нашего списка. Реализовали сохранение в текстовый файл и чтение оттуда. Также было реализовано GUI для работы с нашим двусвязным списком.

**Приложение:**

**IntComparator**

package org.example.linkedListRealisation.comparators  
  
import java.util.Comparator  
  
class IntComparator extends Comparator[Int] {  
 override def compare(o1: Int, o2: Int): Int = o1.compare(o2)  
}

**PointComparator**

package org.example.linkedListRealisation.comparators  
  
import org.example.linkedListRealisation.models.Point  
  
import java.util.Comparator  
  
class PointComparator extends Comparator[Point] {  
  
 override def compare(o1: Point, o2: Point): Int = {  
 val doubleValue1: Double = Math.*sqrt*(Math.*pow*(o1.getX, 2).toInt + Math.*pow*(o1.getY, 2).toInt);  
 val doubleValue2: Double = Math.*sqrt*(Math.*pow*(o2.getX, 2).toInt + Math.*pow*(o2.getY, 2).toInt);  
 val intValue: Integer = doubleValue1.compare(doubleValue2);  
 intValue  
 };  
}

**View**

package org.example.linkedListRealisation.views  
import org.example.linkedListRealisation.Interfaces.UserTypeInterface  
import org.example.linkedListRealisation.UserFactory  
import org.example.linkedListRealisation.dataStructures.LinkedListRealisation  
import scalafx.Includes.\_  
import scalafx.event.ActionEvent  
import scalafx.scene.control.{Button, Label, Menu, MenuBar, MenuItem, ScrollPane, Separator, TextField}  
import scalafx.scene.layout.{AnchorPane, HBox, VBox}  
import scalafx.scene.text.{Font, FontWeight}  
  
import java.beans.EventHandler  
import java.io.{File, FileInputStream, FileOutputStream, FileWriter, ObjectInputStream, ObjectOutputStream}  
import java.lang.Exception  
import scala.collection.mutable.ArrayBuffer  
import scala.io.Source  
import scala.util.control.Breaks.break  
import scala.util.control.Exception  
  
class View() extends AnchorPane {  
  
 val *userFactory*:UserFactory = new UserFactory  
 var *currentUserType*:UserTypeInterface = null  
 var *linkedListRealisation*: LinkedListRealisation[AnyRef] = null  
 var *loadedType*:String = null  
  
 val *menuBar* = new MenuBar  
 val *typeMenu* = new Menu("Type")  
  
  
 *menuBar*.getMenus.add(*typeMenu*)  
 // Buttons  
 val *saveBtn* = new Button("Save") {  
 layoutX = 500  
 layoutY = 50  
 }  
 val *loadBtn* = new Button("Load") {  
 layoutX = 500  
 layoutY = 80  
 }  
 val *sortBtn* = new Button("Sort"){  
 layoutX = 60  
 layoutY = 80  
 }  
 val *addToEndBtn* = new Button("AddToEnd") {  
 layoutX = 44  
 layoutY = 355  
 }  
 val *addByIndexBtn* = new Button("AddByIndex") {  
 layoutX = 200  
 layoutY = 355  
 }  
 val *removeByIndexBtn* = new Button("RemoveByIndex") {  
 layoutX = 338  
 layoutY = 355  
 }  
 val *findByIndexBtn* = new Button("FindByIndex") {  
 layoutX = 488  
 layoutY = 355  
 }  
  
 // Text Fields  
 val *addToEndInput* = new TextField {  
 layoutX = 14  
 layoutY = 301  
 prefHeight = 26  
 prefWidth = 132  
 }  
 val *addByIndexInput* = new TextField {  
 layoutX = 174  
 layoutY = 301  
 prefHeight = 26  
 prefWidth = 132  
 }  
 val *addId* = new TextField {  
 layoutX = 257  
 layoutY = 266  
 prefHeight = 26  
 prefWidth = 42  
 }  
 val *removeId* = new TextField {  
 layoutX = 367  
 layoutY = 266  
 prefHeight = 26  
 prefWidth = 42  
 }  
 val *findId* = new TextField {  
 layoutX = 507  
 layoutY = 266  
 prefHeight = 26  
 prefWidth = 42  
 }  
  
 // Labels  
 val *addToEndErrorLbl* = new Label {  
 layoutX = 20  
 layoutY = 332  
 prefHeight = 18  
 prefWidth = 118  
 }  
 val *addByIndexErrorLbl* = new Label {  
 layoutX = 181  
 layoutY = 332  
 prefHeight = 18  
 prefWidth = 118  
 }  
 val *removeByIndexErrorLbl* = new Label {  
 layoutX = 329  
 layoutY = 332  
 prefHeight = 18  
 prefWidth = 118  
 }  
 val *errorLoading* = new Label("Невозможно прочитать данные из файла!") {  
 layoutX = 180  
 layoutY = 210  
 prefHeight = 18  
 prefWidth = 250  
 }  
 val *findByIndexErrorLbl* = new Label {  
 layoutX = 469  
 layoutY = 332  
 prefHeight = 18  
 prefWidth = 118  
 }  
 val *valueForFind* = new Label {  
 layoutX = 477  
 layoutY = 301  
 prefHeight = 18  
 prefWidth = 101  
 visible = false  
 }  
 val *vBox* = new VBox {  
 prefHeight = 150  
 prefWidth = 444  
 }  
  
 children = List(*vBox*)  
 // Scroll Pane  
 val *scrollPane* = new ScrollPane {  
 layoutX = 102  
 layoutY = 58  
 prefHeight = 150  
 prefWidth = 393  
  
 val *contentPane* = new AnchorPane {  
 prefHeight = 150  
 prefWidth = 376  
  
 children = List(*vBox*)  
 }  
 content = *contentPane* }  
  
 val *separator1* = new Separator {  
 layoutX = 163  
 layoutY = 262  
 orientation = scalafx.geometry.Orientation.*VERTICAL* prefHeight = 140  
 prefWidth = 4  
 }  
 val *separator2* = new Separator {  
 layoutX = 313  
 layoutY = 257  
 orientation = scalafx.geometry.Orientation.*VERTICAL* prefHeight = 140  
 prefWidth = 4  
 }  
 val *separator3* = new Separator {  
 layoutX = 457  
 layoutY = 262  
 orientation = scalafx.geometry.Orientation.*VERTICAL* prefHeight = 140  
 prefWidth = 4  
 }  
  
  
 val *clearBtn* = new Button("Clear!") {  
 layoutX = 252  
 layoutY = 27  
 }  
  
  
 *addToEndBtn*.onAction = (e: ActionEvent) => {  
 disableErrorLabels()  
 val valueToAdd = *addToEndInput*.getText  
 if (valueToAdd.isEmpty) {  
 *addToEndErrorLbl*.setVisible(true)  
 *addToEndErrorLbl*.setText("Wrong input!")  
 }  
 else {  
 try{  
 val obj = *currentUserType*.parseValueDeser(valueToAdd)  
  
 if (obj != null) {  
 *linkedListRealisation*.add(obj)  
 drawList()  
 }  
 }  
 catch{  
 case x: Exception => {  
 *addToEndErrorLbl*.setVisible(true)  
 *addToEndErrorLbl*.setText("Неверный формат!")  
  
 *println*("Wrong Type Format")  
 }  
  
  
 }  
  
 }  
 }  
  
 *addByIndexBtn*.onAction = (e: ActionEvent) => {  
 disableErrorLabels()  
 val valueToAdd = *addByIndexInput*.getText  
 val index = *addId*.getText  
 if (valueToAdd.isEmpty || index.isEmpty) {  
 *addByIndexErrorLbl*.setVisible(true)  
 *addByIndexErrorLbl*.setText("Empty input!")  
 }  
 else {  
 val obj = *currentUserType*.parseValueDeser(valueToAdd)  
 *linkedListRealisation*.addAtIndex(obj,index.toInt)  
 drawList()  
 *addByIndexErrorLbl*.setVisible(false)  
 }  
 }  
  
 *removeByIndexBtn*.onAction = (e: ActionEvent) => {  
 disableErrorLabels()  
 val indexToRemove = *removeId*.getText  
 if (indexToRemove.isEmpty) {  
 *removeByIndexErrorLbl*.setVisible(true)  
 *removeByIndexErrorLbl*.setText("Empty input!")  
 }  
 else {  
 *linkedListRealisation*.removeAtIndex(indexToRemove.toInt)  
 drawList()  
 *addToEndErrorLbl*.setVisible(false)  
 }  
 }  
  
 *findByIndexBtn*.onAction = (e: ActionEvent) => {  
 disableErrorLabels()  
 val indexToFind = *findId*.getText  
 if (indexToFind.isEmpty) {  
 *findByIndexErrorLbl*.setVisible(true)  
 *findByIndexErrorLbl*.setText("Empty input!")  
 }  
 else {  
 val str = *linkedListRealisation*.findById(indexToFind.toInt).toString  
 *valueForFind*.setVisible(true)  
 *valueForFind*.setText(str)  
 *findByIndexErrorLbl*.setVisible(false)  
 }  
 }  
  
 *clearBtn*.onAction = (e: ActionEvent) => {  
 this.*linkedListRealisation*.clear()  
 drawList()  
 }  
  
 *saveBtn*.onAction = (e: ActionEvent) => {  
 saveListToFile()  
 }  
  
 *loadBtn*.onAction = (e: ActionEvent) => {  
 loadListFromFile()  
 }  
  
 *sortBtn*.onAction = (e: ActionEvent) => {  
 this.*linkedListRealisation* = *linkedListRealisation*.sort(*currentUserType*.getTypeComparator);  
 drawList()  
 }  
  
 children = List(  
 *menuBar*,  
 *addToEndBtn*,  
 *addByIndexBtn*,  
 *removeByIndexBtn*,  
 *findByIndexBtn*,  
 *addToEndInput*,  
 *addByIndexInput*,  
 *addId*,  
 *removeId*,  
 *findId*,  
 *addToEndErrorLbl*,  
 *addByIndexErrorLbl*,  
 *removeByIndexErrorLbl*,  
 *findByIndexErrorLbl*,  
 *valueForFind*,  
 *scrollPane*,  
 *separator1*,  
 *separator2*,  
 *separator3*,  
 *clearBtn*,  
 *saveBtn*,  
 *loadBtn*,  
 *errorLoading*,  
 *sortBtn* )  
  
 def initialize(): Unit = {  
 disableManagingBtn()  
 showTypes()  
 disableErrorLbl()  
  
 }  
  
 def drawList(): Unit = {  
 *vBox*.getChildren.clear()  
 val size = *linkedListRealisation*.getSize  
 /\*for (i <- 0 until size) {  
 val str = currentUserType.readValueSer(linkedListRealisation.findById(i).get)+" "  
  
 val label = new Label(str)  
 label.setFont(Font.font("Roboto", FontWeight.EXTRA\_BOLD, 24))  
 vBox.getChildren.add(label)  
 }\*/  
  
 *linkedListRealisation*.forEach(elem =>{  
 val str = *currentUserType*.readValueSer(elem) + " "  
  
 val label = new Label(str)  
 label.setFont(Font.*font*("Roboto", FontWeight.*EXTRA\_BOLD*, 24))  
 *vBox*.getChildren.add(label)  
 })  
 }  
  
  
  
  
  
  
  
 private def showTypes(): Unit = {  
 var i = 0  
  
 for (s <- *userFactory*.getTypeNameList) {  
 val itemToAdd = new MenuItem(s)  
 itemToAdd.setOnAction((event:ActionEvent)=>{  
 *linkedListRealisation* = new LinkedListRealisation[AnyRef]  
 //drawList()  
 *currentUserType* = *userFactory*.getBuilderByName(itemToAdd.getText)  
  
  
 *typeMenu*.getItems.stream().forEach(menuItem => menuItem.setDisable(false))  
  
 itemToAdd.setDisable(true)  
 enableManagingBtn()  
 })  
 itemToAdd.setId("item" + i)  
 *typeMenu*.getItems.add(itemToAdd)  
 i += 1  
 }  
 }  
  
 def disableManagingBtn(): Unit = {  
 *addToEndBtn*.setDisable(true)  
 *addByIndexBtn*.setDisable(true)  
 *removeByIndexBtn*.setDisable(true)  
 *findByIndexBtn*.setDisable(true)  
 }  
  
 private def enableManagingBtn(): Unit = {  
 *addToEndBtn*.setDisable(false)  
 *addByIndexBtn*.setDisable(false)  
 *removeByIndexBtn*.setDisable(false)  
 *findByIndexBtn*.setDisable(false)  
 }  
 private def disableErrorLbl():Unit={  
 *errorLoading*.setVisible(false)  
 }  
  
 private def disableErrorLabels(): Unit = {  
 *addToEndErrorLbl*.setVisible(false)  
 *addByIndexErrorLbl*.setVisible(false)  
 *removeByIndexErrorLbl*.setVisible(false)  
 *findByIndexErrorLbl*.setVisible(false)  
 }  
  
  
  
 def saveListToFile(): Unit = {  
  
 val fileWriter = new FileWriter(new File("save.txt"))  
 fileWriter.write(*currentUserType*.typeName()+"\n")  
 for (i <- 0 until *linkedListRealisation*.getSize) {  
 fileWriter.write(*linkedListRealisation*.findById(i).get.toString +"\n")  
 }  
 fileWriter.close()  
  
  
 }  
  
 def loadListFromFile(): Unit = {  
 *linkedListRealisation*.clear();  
 *errorLoading*.setVisible(false)  
 val fileName = "save.txt"  
  
 var i: Integer = 0  
 var userType:UserTypeInterface = null  
 for (line <- Source.*fromFile*(fileName).getLines()) {  
 if(i==0) {  
 *loadedType* = line  
 if(*loadedType* != *currentUserType*.typeName()){  
 *errorLoading*.setVisible(true)  
 *linkedListRealisation*.clear()  
 break()  
 }  
 userType = *userFactory*.getBuilderByName(*loadedType*)  
 }  
 else {  
 *linkedListRealisation*.add(userType.parseValueDeser(line))  
 }  
 i += 1  
 }  
  
 drawList()  
 }  
  
 initialize()  
}

**LinkedListRealisation**

package org.example.linkedListRealisation.dataStructures  
import org.example.linkedListRealisation.Interfaces.LinkedListInterface  
import scalafx.scene.input.KeyCombination.ModifierValue.Any  
  
import java.util.Comparator  
import java.util.function.Consumer  
import scala.reflect.ClassTag  
  
class LinkedListRealisation[T:ClassTag] extends LinkedListInterface[T] with Serializable {  
 private class Node(var prev: Node, var next: Node, val value: T) extends Serializable  
  
 private var head: Node = null  
 private var tail: Node = null  
 private var size:Int = 0  
  
  
  
 def clear():Unit = {  
 head = null;  
 tail = null;  
 size = 0;  
 }  
 def getSize:Int = {  
 size  
 }  
 def add(item: T): Boolean = {  
 val newNode = new Node(null, null, item)  
 if (head == null) {  
 head = newNode  
 tail = newNode  
 } else {  
 tail.next = newNode  
 newNode.prev = tail  
 tail = newNode  
 }  
 size+=1;  
 true  
 }  
  
 def addAtIndex(item: T, index: Int): Boolean = {  
 val newNode = new Node(null, null, item)  
 if (index == 0) {  
 newNode.next = head  
 head.prev = newNode  
 head = newNode  
 } else {  
 var current = head  
 var i = 0  
 while (i < index && current != null) {  
 current = current.next  
 i += 1  
 }  
 if (current != null) {  
 newNode.next = current  
 newNode.prev = current.prev  
 current.prev.next = newNode  
 current.prev = newNode  
 } else {  
 tail.next = newNode  
 newNode.prev = tail  
 tail = newNode  
 }  
 }  
 size +=1  
 true  
 }  
  
 def remove(item: T): Boolean = {  
 var current = head  
 while (current != null) {  
 if (current.value == item) {  
 if (current == head) {  
 head = current.next  
 if (head != null) head.prev = null  
 }  
 if (current == tail) {  
 tail = current.prev  
 if (tail != null) tail.next = null  
 }  
 if (current.prev != null) current.prev.next = current.next  
 if (current.next != null) current.next.prev = current.prev  
 size -=1  
 return true  
 }  
 current = current.next  
 }  
 false  
 }  
  
 def removeAtIndex(index: Int): Boolean = {  
 if (index == 0) {  
 if (head != null) {  
 head = head.next  
 if (head != null) head.prev = null  
 }  
 size -=1  
 true  
 } else {  
 var current = head  
 var i = 0  
 while (i < index && current != null) {  
 current = current.next  
 i += 1  
 }  
 if (current != null) {  
 if (current == tail) {  
 tail = current.prev  
 if (tail != null) tail.next = null  
 } else {  
 current.prev.next = current.next  
 current.next.prev = current.prev  
 }  
 size -=1  
 true  
 } else {  
 false  
 }  
 }  
 }  
  
 def findById(id: Int): Option[T] = {  
 var current = head  
 var i = 0  
 while (i < id && current != null) {  
 current = current.next  
 i += 1  
 }  
 if (current != null) {  
 Some(current.value)  
 } else {  
 None  
 }  
 }  
  
 def printList(): Unit = {  
 var current = head  
 while (current != null) {  
 println(current.value)  
 current = current.next  
 }  
 }  
  
 def forEach(consumer: Consumer[T]): Unit = {  
 var current = this.head  
 while (current != null) {  
 consumer.accept(current.value)  
 current = current.next  
 }  
 }  
  
 def foo[T:ClassTag](count: Int, value: T): Array[T] = Array.fill[T](count)(value)  
  
 def toList(): List[T] = {  
 var current = head  
 var array:Array[T] = Array[T]()  
  
 while (current != null) {  
 array :+= current.value  
  
 current = current.next  
 }  
 val list:List[T] = array.toList  
 list  
 }  
  
 def convertListToLinkedListRealisation(list: List[T]):LinkedListRealisation[T] ={  
 val linkedListRealisation:LinkedListRealisation[T] = new LinkedListRealisation[T]  
 for(i<-list.indices){  
 linkedListRealisation.add(list(i))  
 }  
 linkedListRealisation  
 }  
  
  
 def sort(comparator: Comparator[\_]):LinkedListRealisation[T] ={  
 val comparator2: Comparator[T] = comparator.asInstanceOf[Comparator[T]]  
 var list:List[T] = toList()  
 val sortedList:List[T] = mergeSort(list:List[T], comparator2:Comparator[T])  
 val linkedListRealisation:LinkedListRealisation[T] = convertListToLinkedListRealisation(sortedList)  
 linkedListRealisation  
 }  
  
  
 private def merge(left: List[T], right: List[T],comparator: Comparator[T]): List[T] = (left, right) match {  
 case (Nil, right) => right  
 case (left, Nil) => left  
 case (leftHead :: leftTail, rightHead :: rightTail) =>  
 if (comparator.compare(leftHead, rightHead) == -1)  
 leftHead :: merge(leftTail, right,comparator)  
 else  
 rightHead :: merge(left, rightTail,comparator)  
 }  
  
 private def mergeSort(arr: List[T],comparator: Comparator[T]): List[T] = {  
 val n = arr.length  
 if (n <= 1) arr  
 else {  
 val mid = n / 2  
 val (left, right) = arr.splitAt(mid)  
 merge(mergeSort(left,comparator), mergeSort(right,comparator),comparator)  
 }  
 }  
}

**LinkedListInterface**

package org.example.linkedListRealisation.Interfaces  
  
import scala.reflect.ClassTag  
  
trait LinkedListInterface[T] {  
 def add(value: T): Boolean  
 def addAtIndex(value: T, id: Int): Boolean  
 def remove(value: T): Boolean  
 def removeAtIndex(id: Int): Boolean  
 def findById(id: Int): Option[T]  
 def printList(): Unit  
}

**UserTypeInterface**

package org.example.linkedListRealisation.Interfaces  
  
import java.util.Comparator  
  
trait UserTypeInterface {  
 def typeName(): String  
  
 def readValueSer(obj: AnyRef): String  
  
 def parseValueDeser(ss: String): AnyRef  
  
 def getTypeComparator: Comparator[\_]  
}

**Point**

package org.example.linkedListRealisation.models  
  
class Point(var x:Integer, var y:Integer) extends Serializable {  
  
  
  
 def setX(\_x: Int) {  
 x = \_x;  
 }  
  
 def getX(): Int = x  
  
 def setY(\_y: Int) {  
 y = \_y;  
 }  
  
 def getY(): Int = y  
 override def toString: String = "("+ x + ", " + y+")"  
  
}

**IntClass**

package org.example.linkedListRealisation.userTypes  
  
import org.example.linkedListRealisation.Interfaces.UserTypeInterface  
import org.example.linkedListRealisation.comparators.IntComparator  
  
import java.util.Comparator  
  
class IntClass extends UserTypeInterface {  
 override def typeName = "Integer"  
  
 //override def create: Int = (Math.random \* 100).toInt - 50  
  
 override def clone: AnyRef = try super.clone.asInstanceOf[IntClass]  
 catch {  
 case e: CloneNotSupportedException =>  
 throw new RuntimeException(e)  
 }  
  
 override def readValueSer(intValue: AnyRef): String = intValue.toString  
  
 override def parseValueDeser(str: String): Integer = str.toInt  
  
 override def getTypeComparator():Comparator[\_] = new IntComparator  
  
  
}

**PointClass**

package org.example.linkedListRealisation.userTypes  
  
import org.example.linkedListRealisation.Interfaces.UserTypeInterface  
import org.example.linkedListRealisation.comparators.{IntComparator, PointComparator}  
import org.example.linkedListRealisation.models.Point  
  
import java.util.Comparator  
  
class PointClass extends UserTypeInterface with Cloneable {  
 override def typeName = "Point"  
  
 // override def create = new Point((Math.random \* 100).toInt - 50, (Math.random \* 100).toInt - 50)  
  
 override def clone: AnyRef = try super.clone.asInstanceOf[PointClass]  
 catch {  
 case e: CloneNotSupportedException =>  
 throw new RuntimeException(e)  
 }  
  
 override def readValueSer(point: AnyRef): String = {  
 val pointToSer = point.asInstanceOf[Point]  
 pointToSer.toString  
 }  
  
 override def parseValueDeser(str: String): AnyRef = {  
 val len:Integer = str.length  
 val substr:String = str.substring(1,len-1)  
 val parameterOfPoint = substr.split(", ")  
 if (parameterOfPoint.length == 2) {  
  
 val x = parameterOfPoint(0).toInt  
 val y = parameterOfPoint(1).toInt  
  
 new Point(x, y)  
 }  
 else null  
 }  
  
 override def getTypeComparator():Comparator[\_] = new PointComparator  
}

**Main**

import org.example.linkedListRealisation.views.View  
import scalafx.application.JFXApp3.PrimaryStage  
  
import scalafx.application.{JFXApp3, Platform}  
import scalafx.scene.Scene  
  
object Main extends JFXApp3 {  
  
 def start(): Unit = {  
 *stage* = new PrimaryStage() {  
 title = "LinkedListRealisation"  
 scene = new Scene(new View)  
 }  
 }  
}

**UserFactory** package org.example.linkedListRealisation  
  
import org.example.linkedListRealisation.Interfaces.UserTypeInterface  
import org.example.linkedListRealisation.userTypes.{IntClass, PointClass}  
  
import scala.Console.*in*class UserFactory {  
 private val *list*: List[UserTypeInterface] = List(new IntClass,new PointClass)  
  
  
 def getTypeNameList: List[String] = {  
 val typeNameList = *list*.map(x => x.typeName()).toList  
 typeNameList  
 }  
  
 def getBuilderByName(name: String): UserTypeInterface = *list*.find(userType => userType.typeName().equals(name)).orNull  
}