

Лабораторна робота №10

Обробка параметризованих контейнерів

Мета: розширення функціональності параметризованих класів.

1 Вимоги

Використовуючи програму рішення завдання лабораторної роботи №9:

1. Розробити параметризовані методи (Generic Methods) для обробки колекцій об'єктів згідно прикладної задачі.
2. Продемонструвати розроблену функціональність (створення, управління та обробку власних контейнерів) в діалоговому та автоматичному режимах.
 - Автоматичний режим виконання програми задається параметром командного рядка -auto. Наприклад, java ClassName -auto.
 - В автоматичному режимі діалог з користувачем відсутній, необхідні данні генеруються, або зчитуються з файлу.
3. Забороняється використання алгоритмів з Java Collections Framework.

Варіант 3:

Адресна книга. Сортування за прізвищем, за ім'ям, за датою народження, за датою редагування.

1.1 Розробник

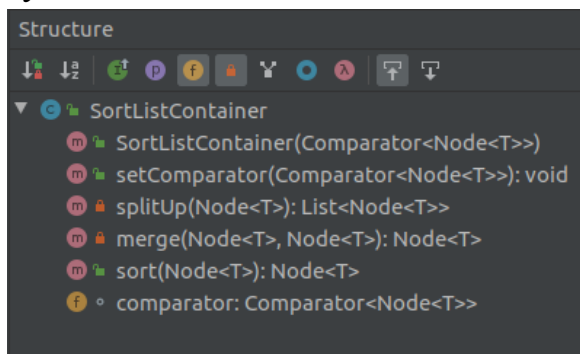
Інформація про розробника:

- Безпальний М.Л.;
- КІТ - 118Г;
- варіант №3.

ОПИС ПРОГРАМИ

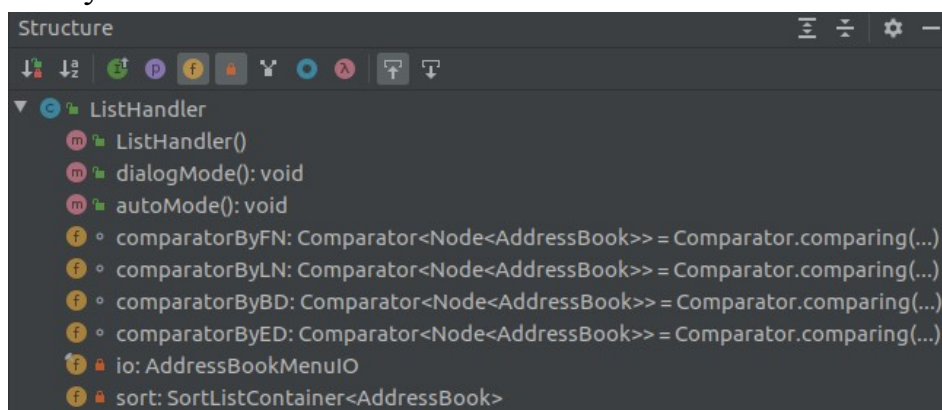
Дана програма містить клас для сортування контейнеру ("*SortListContainer*"), клас для обробки контейнеру в діалоговому та автоматичному режимах ("*ListHandler*"), клас для управління вводом-виводом для обробки контейнеру ("*ListMenuIO*"), який реалізує інтерфейс ("*AddressBookMenuIO*"), а також клас для демонстрації роботи розроблених класів ("*DemoListApp*"), який містить метод *main*.

Структура класу *SortListContainer*:



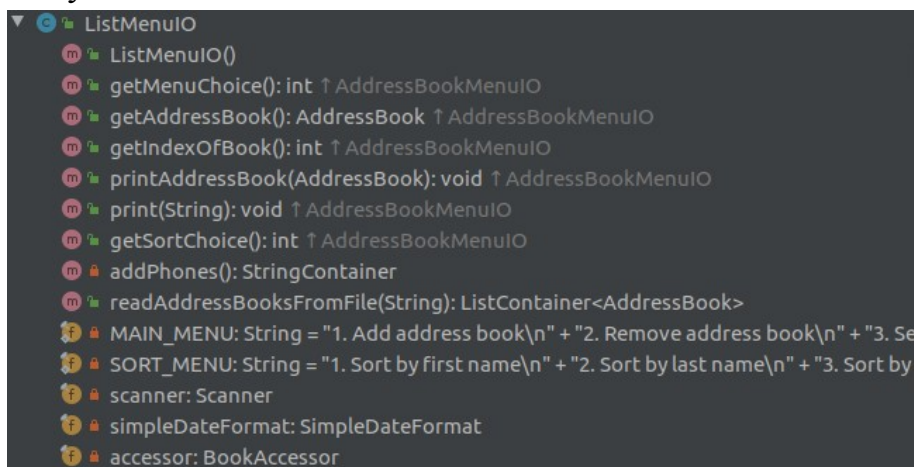
Клас виконує сортування зв'язного списку за алгоритмом зливання. "Merge sort"

Структура класу *ListHandler*:



Клас містить конструктор, що ініціалізує змінну інтерфейсу вводу-виводу "іо", та змінну для сортування "sort". Також клас містить компаратори для сортування по різним критеріям і методи обробки контейнеру (в діалоговому та автоматичному режимах).

Структура класу *ListMenuIO*



Клас містить конструктор, що ініціалізує приватні змінні класу типів *Scanner*, *SimpleDateFormat* та *BookAccessor* (для лабораторної роботи №11). Клас займається обробкою даних, що вводяться за клавіатури та з файлу.

Текст програми

SortListContainer.java

```
package ua.khpi.oop.bezpalyi10;

import ua.khpi.oop.bezpalyi09.ListContainer;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

public class SortListContainer<T> {
    Comparator<ListContainer.Node<T>> comparator;

    public SortListContainer(Comparator<ListContainer.Node<T>> comparator) {
        this.comparator = comparator;
    }

    public void setComparator(Comparator<ListContainer.Node<T>> comparator) {
        this.comparator = comparator;
    }

    private List<ListContainer.Node<T>> splitUp(ListContainer.Node<T> inHead) {
        List<ListContainer.Node<T>> list = new ArrayList<>();

        if (inHead == null || inHead.getNext() == null) {
            list.add(inHead);
            list.add(null);
            return list;
        }
        ListContainer.Node<T> slow = inHead;
        ListContainer.Node<T> fast = inHead.getNext();

        while (fast != null) {
            fast = fast.getNext();
            if (fast != null) {
                slow = slow.getNext();
                fast = fast.getNext();
            }
        }

        list.add(inHead);
        list.add(slow.getNext());
        slow.setNext(null);
        return list;
    }

    private ListContainer.Node<T> merge(ListContainer.Node<T> first,
ListContainer.Node<T> second) {
        if (first == null) {
            return second;
        }

        if (second == null) {
            return first;
        }

        ListContainer.Node<T> result;

        if (comparator.compare(first, second) <= 0) {
            result = first;
            result.setNext(merge(first.getNext(), second));
        } else {
            result = second;
            result.setNext(merge(first, second.getNext()));
        }
    }
}
```

```

    }

    return result;
}

public ListContainer.Node<T> sort(ListContainer.Node<T> inHead) {
    if (inHead == null || inHead.getNext() == null) {
        return inHead;
    }
    List<ListContainer.Node<T>> array = splitUp(inHead);
    ListContainer.Node<T> firstPart = array.get(0);
    ListContainer.Node<T> secondPart = array.get(1);

    firstPart = sort(firstPart);
    secondPart = sort(secondPart);

    return merge(firstPart, secondPart);
}
}

```

ListHandler

```

package ua.khpi.oop.bezpalyi10;

import ua.khpi.oop.bezpalyi07.AddressBook;
import ua.khpi.oop.bezpalyi09.ListContainer;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDateTime;
import java.util.Comparator;
import java.util.Date;

public class ListHandler {

    Comparator<ListContainer.Node<AddressBook>> comparatorByFN =
        Comparator.comparing(addressBookNode ->
addressBookNode.getValue().getFirstName());

    Comparator<ListContainer.Node<AddressBook>> comparatorByLN =
        Comparator.comparing(addressBookNode ->
addressBookNode.getValue().getLastName());

    Comparator<ListContainer.Node<AddressBook>> comparatorByBD =
        Comparator.comparing(addressBookNode ->
addressBookNode.getValue().getDateOfBirth());

    Comparator<ListContainer.Node<AddressBook>> comparatorByED =
        Comparator.comparing(addressBookNode ->
addressBookNode.getValue().getEditDateAndTime());

    private final AddressBookMenuIO io;

    private SortListContainer<AddressBook> sort;

    public ListHandler() {
        io = new ListMenuIO();
        sort = null;
    }

    public void dialogMode() {
        ListContainer<AddressBook> list = new ListContainer<>();
        System.out.println("List is ready!");
    }
}

```

```

while (true) {
    int choice = io.getMenuChoice();

    if (choice == 8) {
        System.out.println("Goodbye");
        break;
    }

    if (choice < 1 || choice > 7) {
        System.out.println("Invalid input data!");
        continue;
    }

    switch (choice) {
        case 1:
            try {
                AddressBook addressBook = io.getAddressBook();
                if (addressBook != null) {
                    list.add(addressBook);
                }
            } catch (ParseException e) {
                System.out.println("Failed to parse data!");
            }
            break;
        case 2:
            list.remove(io.getIndexOfBook());
            break;
        case 3:
            io.printAddressBook(list.get(io.getIndexOfBook()));
            break;
        case 4:
            io.print(list.toString());
            break;
        case 5:
            list.clear();
            break;
        case 6:
            io.print("Size: " + list.size());
            break;
        case 7:
            int sortChoice = io.getSortChoice();
            if (sortChoice == 0) {
                System.out.println("Invalid input data!");
                break;
            }
            switch (sortChoice) {
                case 1:
                    sort = new SortListContainer<>(comparatorByFN);
                    list.setHead(sort.sort(list.getHead()));
                    break;
                case 2:
                    sort = new SortListContainer<>(comparatorByLN);
                    list.setHead(sort.sort(list.getHead()));
                    break;
                case 3:
                    sort = new SortListContainer<>(comparatorByBD);
                    list.setHead(sort.sort(list.getHead()));
                    break;
                case 4:
                    sort = new SortListContainer<>(comparatorByED);
                    list.setHead(sort.sort(list.getHead()));
                    break;
            }
            break;
    }
}

```

```

    }
}

public void autoMode() throws ParseException {
    ListContainer<AddressBook> list = new ListContainer<>();
    AddressBook addressBook1;
    AddressBook addressBook2;
    AddressBook addressBook3;
    AddressBook addressBook4;
    LocalDateTime now = LocalDateTime.now();

    SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");
    Date date1 = simpleDateFormat.parse("2009-12-31");

    addressBook1 = new AddressBook.Builder()
        .setFirstName("Bob")
        .setLastName("First surname")
        .setSecondName("First second name")
        .setAddress("Address1")
        .setDateOfBirth(date1)
        .setEditTime(now.toString())
        .build();

    addressBook1.addPhoneNumber("9825792");
    addressBook1.addPhoneNumber("3928729");

    Date date2 = simpleDateFormat.parse("1980-10-20");
    addressBook2 = new AddressBook.Builder()
        .setFirstName("Alex")
        .setLastName("Second surname")
        .setSecondName("Second second name")
        .setAddress("Address2")
        .setDateOfBirth(date2)
        .setEditTime(now.plusDays(3).toString())
        .build();

    addressBook2.addPhoneNumber("290302");
    addressBook2.addPhoneNumber("0978431");

    Date date3 = simpleDateFormat.parse("2000-09-23");
    addressBook3 = new AddressBook.Builder()
        .setFirstName("John")
        .setLastName("Biter")
        .setSecondName("Alfred")
        .setAddress("Address3")
        .setDateOfBirth(date3)
        .setEditTime(now.minusHours(72).toString())
        .build();
    addressBook3.addPhoneNumber("74744798");

    Date date4 = simpleDateFormat.parse("2000-10-20");
    addressBook4 = new AddressBook.Builder()
        .setFirstName("Mark")
        .setLastName("Bezpalyi")
        .setSecondName("Leonidovich")
        .setAddress("Address4")
        .setDateOfBirth(date4)
        .setEditTime(now.plusMonths(2).toString())
        .build();
    addressBook4.addPhoneNumber("0371630238743");

    list.add(addressBook1);
}

```

```

        list.add(addressBook2);
        list.add(addressBook3);
        list.add(addressBook4);

        io.print(list.toString());
    }
}

```

ListMenuIO

```

package ua.khpi.oop.bezpalyi10;

import ua.khpi.oop.bezpalyi05.StringContainer;
import ua.khpi.oop.bezpalyi07.AddressBook;
import ua.khpi.oop.bezpalyi09.ListContainer;
import ua.khpi.oop.bezpalyi11.AddressBookAccessor;
import ua.khpi.oop.bezpalyi11.BookAccessor;

import java.io.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDateTime;
import java.util.Date;
import java.util.Scanner;

public class ListMenuIO implements AddressBookMenuIO {
    private final static String MAIN_MENU =
        "1. Add address book\n" +
        "2. Remove address book\n" +
        "3. Search address book\n" +
        "4. Show all address books\n" +
        "5. Remove all address book\n" +
        "6. Show size\n" +
        "7. Sort address books\n" +
        "8. Exit";

    private final static String SORT_MENU =
        "1. Sort by first name\n" +
        "2. Sort by last name\n" +
        "3. Sort by birth date\n" +
        "4. Sort by edit date";

    private final Scanner scanner;
    private final SimpleDateFormat simpleDateFormat;
    private final BookAccessor accessor;

    public ListMenuIO() {
        scanner = new Scanner(System.in);
        simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");
        accessor = new AddressBookAccessor();
    }

    @Override
    public int getMenuChoice() {
        System.out.println(MAIN_MENU);
        return scanner.nextInt();
    }

    @Override
    public AddressBook getAddressBook() throws ParseException {
        AddressBook.Builder bookBuilder = new AddressBook.Builder();

        System.out.print("Enter first name: ");
        String data = scanner.next();
    }
}

```

```

        if (accessor.assertNames(data)) {
            bookBuilder.setFirstName(data);
        } else {
            System.out.println("Invalid input first name!");
            return null;
        }

        System.out.print("Enter last name: ");
        data = scanner.next();
        if (accessor.assertNames(data)) {
            bookBuilder.setLastName(data);
        } else {
            System.out.println("Invalid input last name!");
            return null;
        }

        System.out.print("Enter second name: ");
        data = scanner.next();
        if (accessor.assertNames(data)) {
            bookBuilder.setSecondName(data);
        } else {
            System.out.println("Invalid input second name!");
            return null;
        }

        System.out.print("Enter date of birth in format yyyy-MM-dd: ");
        Date date = simpleDateFormat.parse(scanner.next());
        bookBuilder.setDateOfBirth(date);

        System.out.print("Enter address: ");
        bookBuilder.setAddress(scanner.next());

        bookBuilder.setEditTime(LocalDateTime.now().toString());
        bookBuilder.setPhoneNumbers(addPhones());
        return bookBuilder.build();
    }

    @Override
    public int getIndexOfBook() {
        System.out.print("Enter index of book: ");
        return scanner.nextInt();
    }

    @Override
    public void printAddressBook(AddressBook book) {
        System.out.println(book);
    }

    @Override
    public void print(String string) {
        System.out.println(string);
    }

    @Override
    public int getSortChoice() {
        System.out.println("Choose number of sort");
        System.out.println(SORT_MENU);
        int choice = scanner.nextInt();
        if (choice < 1 || choice > 4) {
            return 0;
        }
        return choice;
    }
}

```



```

private StringContainer addPhones() {
    StringContainer container = new StringContainer();
    System.out.print("How many phone numbers book will contain? (max: 5)
");
    int size = scanner.nextInt();

    if (size < 0) {
        return container;
    } else if (size > AddressBookAccessor.MAX_PHONE_NUMBERS_SIZE) {
        size = AddressBookAccessor.MAX_PHONE_NUMBERS_SIZE;
    }

    int maxTriesSize = 5;
    String data;
    for (int i = 0; i < size && maxTriesSize > 0; i++) {
        System.out.print("Enter phone number:");
        data = scanner.next();
        if (accessor.assertPhoneNumber(data)) {
            container.add(data);
        } else {
            i--;
            System.out.println("Invalid input phone number!");
            maxTriesSize--;
        }
    }
    return container;
}

public ListContainer<AddressBook> readAddressBooksFromFile(String path) {
    ListContainer<AddressBook> list = new ListContainer<>();
    try(FileReader reader = new FileReader(path)) {
        BufferedReader br = new BufferedReader(reader);
        AddressBook.Builder bookBuilder = new AddressBook.Builder();

        String data = br.readLine();
        while(data != null) {
            if (accessor.assertNames(data)) {
                bookBuilder.setFirstName(data);
            } else {
                System.out.println("Invalid input first name!");
                return list;
            }

            data = br.readLine();
            if (accessor.assertNames(data)) {
                bookBuilder.setLastName(data);
            } else {
                System.out.println("Invalid input last name!");
                return list;
            }

            data = br.readLine();
            if (accessor.assertNames(data)) {
                bookBuilder.setSecondName(data);
            } else {
                System.out.println("Invalid input second name!");
                return list;
            }

            data = br.readLine();
            Date date = SimpleDateFormat.parse(data);
            bookBuilder.setDateOfBirth(date);

            data = br.readLine();

```

```

        bookBuilder.setAddress(data);

        bookBuilder.setEditTime(LocalDateTime.now().toString());

        data = br.readLine();
        String[] phoneNumbers = data.split(", ");
        StringContainer sc = new StringContainer();
        for(String s : phoneNumbers) {
            sc.add(s);
        }
        bookBuilder.setPhoneNumbers(sc);
        list.add(bookBuilder.build());

        br.readLine();
    }
} catch (FileNotFoundException e) {
    System.out.println("No such file " + path);
} catch (IOException e) {
    throw new UncheckedIOException(e);
} catch (ParseException e) {
    System.out.println("Invalid birth date format!");
}
return list;
}
}

```

AddressBookMenuIO

```

package ua.khpi.oop.bezpalyi10;

import ua.khpi.oop.bezpalyi07.AddressBook;
import java.text.ParseException;

public interface AddressBookMenuIO {
    int getMenuChoice();

    AddressBook getAddressBook() throws ParseException;

    int getIndexOfBook();

    void printAddressBook(AddressBook book);

    void print(String string);

    int getSortChoice();
}

```

DemoListApp.java

```

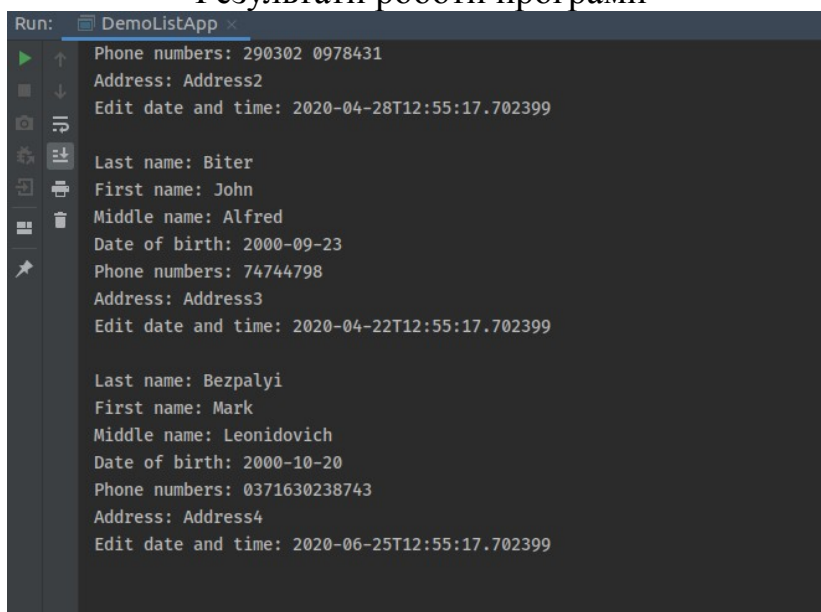
package ua.khpi.oop.bezpalyi10;

import java.text.ParseException;

public class DemoListApp {
    public static void main(String[] args) throws ParseException {
        ListHandler listHandler = new ListHandler();
        if (args.length > 0 && args[0].equals("-auto")) {
            listHandler.autoMode();
        } else {
            listHandler.dialogMode();
        }
    }
}

```

Результати роботи програми



```
Run: DemoListApp <
Phone numbers: 290302 0978431
Address: Address2
Edit date and time: 2020-04-28T12:55:17.702399

Last name: Biter
First name: John
Middle name: Alfred
Date of birth: 2000-09-23
Phone numbers: 74744798
Address: Address3
Edit date and time: 2020-04-22T12:55:17.702399

Last name: Bezpalyi
First name: Mark
Middle name: Leonidovich
Date of birth: 2000-10-20
Phone numbers: 0371630238743
Address: Address4
Edit date and time: 2020-06-25T12:55:17.702399
```

Рисунок 1 — результати роботи програми в автоматичному режимі

До уваги: для перевірки правильності роботи програми написані тести, що знаходяться в папці test в однойменному пакеті.

Висновок: зробив розширення функціональності параметризованих класів, додаванням сортировки та оброки в автоматичному та діалоговому режимах.