Тема: Паралельне виконання. Багатопоточність.
Мета: Ознайомлення з моделлю потоків Java. Організація паралельного виконання декількох частин програми.

1 ВИМОГИ

1.1 Розробник

Інформація про розробника:
- Когутенко Олександр Олексійович;
- КІТ-119Д;
- 11 варіант.

1.2 Загальне завдання

1. Використовуючи програми рішень попередніх задач, продемонструвати можливість паралельної обробки елементів контейнера: створити не менше трьох додаткових потоків, на яких викликати відповідні методи обробки контейнера.
2. Забезпечити можливість встановлення користувачем максимального часу виконання (таймаута) при закінченні якого обробка повинна припинятися незалежно від того знайдений кінцевий результат чи ні.
3. Для паралельної обробки використовувати алгоритми, що не змінюють початкову колекцію.
4. Кількість елементів контейнера повинна бути досить велика, складність алгоритмів обробки колекції повинна бути зіставна, а час виконання приблизно однаковий, наприклад:

- пошук мінімуму або максимуму;
- обчислення середнього значення або суми;
- підрахунок елементів, що задовольняють деякій умові;
- відбір за заданим критерієм;
- власний варіант, що відповідає обраній прикладної області.

2 ОПИС ПРОГРАМИ

2.1 Засоби ООП

Використовуеться наслідування, інтерфейс, поліморфізм.

2.2 Ієрархія та структура класів

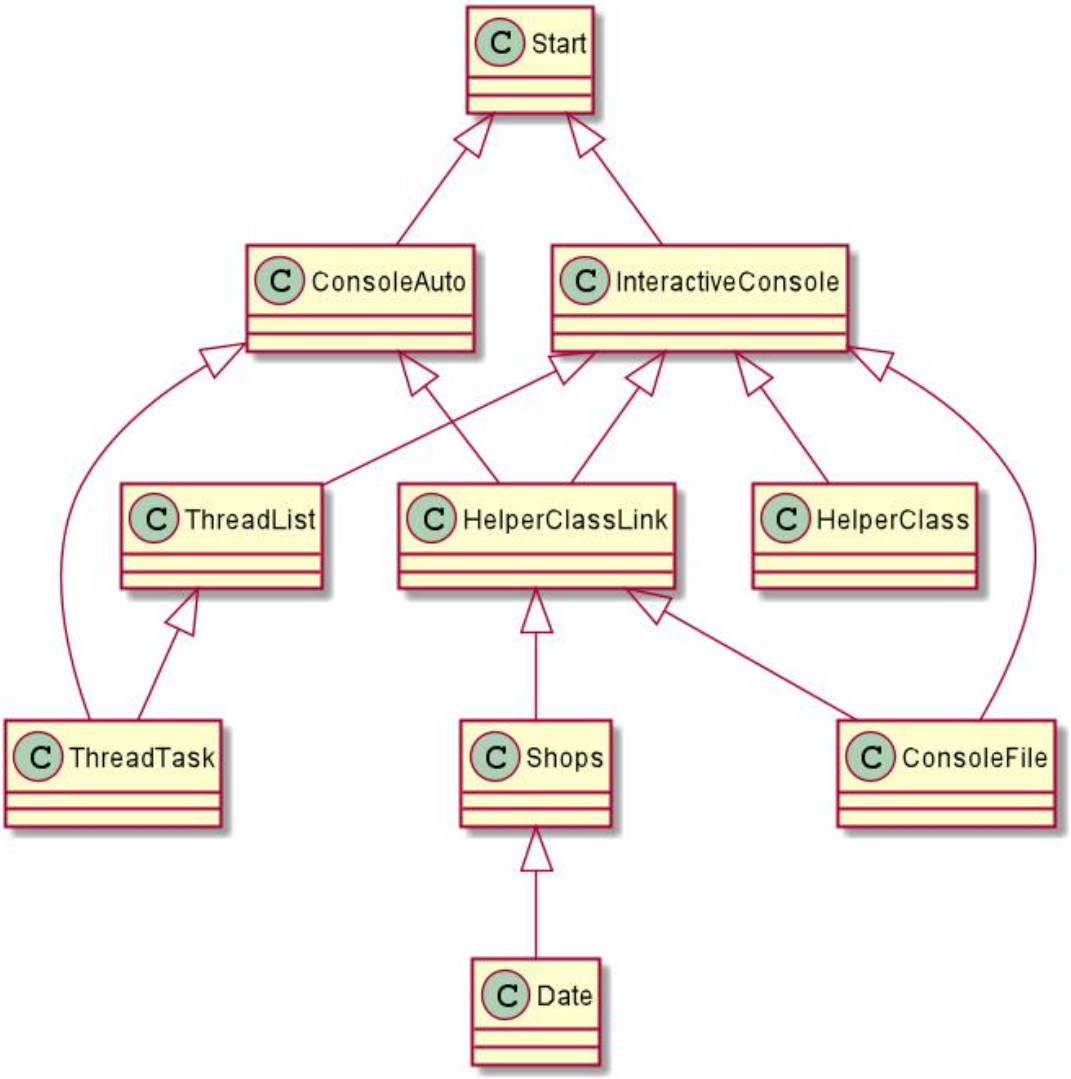

Рисунок 13.1 - иерархія класів

Використовую 7 классів: InteractiveConsole, Start, Date, Shops, ConsoleFile, HelperClassLink, ConoleAuto, ThreadTask, ThreadList.

- InteractiveConsole клас для налагодженого спілкування програми з користувачем та методами нестандартних протоколів серіалізації.
- Start клас який має точку входу у програму.
- Date клас використовуется для збереження дати.
- Shops клас приклодної галузі.
- ConoleFile клас за допомогою якого користувач може наблюдати за відображенням вмісту каталогів.
- HelperClassLink використовуеться за для реалізації зв’язного списку з методами стандартних методів серіалізації.
- ConsoleAuto класс для автоматичної роботи із списком.
- ThreadTask деяки шаблон контейнеру для
- ThreadList класс який має в собі статичні класи з яких створюється потоки для кожної дії.

2.3 Важливі фрагменти програми

На мою думку використання статичних класів і методів для створення потоків є більш зрозумілою для користувача моїми класами та на мою думку це вигідно бо моїми методами будуть користуватись по одному потоку, тобто не буде “гонки” на використання того чи іншого методу між потоками.
Фрагменти коду де є робота з потоками:

```
package ua.khpi.oop.kogutenko13;

import java.util.concurrent.Callable;
import java.util.function.Function;

public class ThreadTask extends Thread{
    protected HelperClassLink<Shops> list;

    ThreadTask() {
        list = new HelperClassLink<>();
    }

    ThreadTask(HelperClassLink<Shops> list) {
        this.list = list;
    }

    public HelperClassLink<Shops> getList(){
        return list;
    }
}
```

```
package ua.khpi.oop.kogutenko13;

import java.io.File;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class ThreadList {
    static class TaskPrintList extends ThreadTask {
        TaskPrintList(HelperClassLink<Shops> list) {
            super(list);
        }

        TaskPrintList() {
        }

        TaskPrintList(ThreadTask thread) {
            super(thread.getList());
        }

        public static void print(ThreadTask thread) throws InterruptedException {
            ThreadTask th = new TaskPrintList(thread);
```

```
            th.start();
            th.join();
            th.interrupt();
        }

        @Override
        public void run() {
            list.printList();
        }
    }

    static class TaskSerializationList extends ThreadTask {
        private static File file;
        private static Integer answ;
        ThreadTask threadTask;

        TaskSerializationList() {
        }

        TaskSerializationList(HelperClassLink<Shops> list) {
            super(list);
            threadTask.list = list;
        }

        TaskSerializationList(ThreadTask thread) {
            super(thread.getList());
            threadTask = thread;
        }


        public static void serialization(ThreadTask thread) throws InterruptedException {
            System.out.print("What save do you want? (1 - .txt; 2 - .bin; 3 - .xml)\n>>> ");
            Integer answ;
            Scanner scanner = new Scanner(System.in);
            while (true) {
                Pattern p = Pattern.compile("[123]");
                answ = scanner.nextInt();
                Matcher m = p.matcher(answ.toString());
                if (m.matches()) {
                    file = ConsoleFile.MenuFillOut();///pathname
                    ThreadTask th = new TaskPrintList(thread);
                    th.start();
                    th.join();
                    th.interrupt();
                } else {
                    System.out.println("Enter info correctly!!!");
                }
            }
        }

        @Override
        public void run() {
```

```
            threadTask.list.serialization(answ, file);
        }
    }

    static class TaskDeserializationList extends ThreadTask implements Runnable {
        private static File file;
        private static Integer answ;
        ThreadTask threadTask;
        TaskDeserializationList() { }

        TaskDeserializationList(HelperClassLink<Shops> list) {
            super(list);
            threadTask.list = list;
        }

        TaskDeserializationList(ThreadTask th) {
            super(th.getList());
            threadTask = th;
        }

        public static void deserialization(ThreadTask thread) throws InterruptedException {
            System.out.print("what deserialization do you want?\n(1 - bin, 2 - xml, 3 - txt)\n>>> ");
            Scanner scanner = new Scanner(System.in);
            while (true) {
                System.out.println("regex verification");
                Pattern p = Pattern.compile("[123]");
                answ = scanner.nextInt();
                Matcher m = p.matcher(answ.toString());
                if (m.matches()) {
                    file = ConsoleFile.MenuFillIn();///pathname
                    Thread th = new TaskDeserializationList(thread);
                    th.start();
                    th.join();
                    th.interrupt();
                    break;
                } else {
                    System.out.println("Enter info correctly!!!");
                }
            }
        }

        @Override
        public void run() {
            threadTask.getList().deserialization(answ, file);
        }
    }

    static class TaskSortList extends ThreadTask {
        private static ThreadTask thread;
        private static Integer field;
        TaskSortList() {
        }
```

```
    TaskSortList(HelperClassLink<Shops> list) {
        super(list);
        thread.list = list;
    }

    TaskSortList(ThreadTask thread) {
        super(thread.getList());
        this.thread = thread;
    }

    public static void sort(ThreadTask th) throws InterruptedException {
        ThreadTask thread = new TaskSortList(th);
        thread.start();
        thread.join();
        thread.interrupt();
    }

    @Override
    public void run() {
        System.out.println("start run sort");
        thread.list.printList();
        System.out.println("----------------------------------------------------");
        thread.list = thread.list.fromArray(thread.list.bubbleSort(thread.list.toArray() , 3));
        thread.list.printList();
    }
}

static class TaskFreshList extends ThreadTask {
    private static ThreadTask thread;
    private static Integer field;
    TaskFreshList() {
    }

    TaskFreshList(HelperClassLink<Shops> list) {
        super(list);
        thread.list = list;
    }

    TaskFreshList(ThreadTask thread) {
        super(thread.getList());
        this.thread = thread;
    }

    public static void fresh(ThreadTask th) throws InterruptedException {
        ThreadTask thread = new TaskFreshList(th);
        thread.start();
        thread.join();
        thread.interrupt();
    }

    @Override
```

```
        public void run() {
            System.out.println(thread.list.findFresh(thread.getList()));
        }
    }

}
```

```
package ua.khpi.oop.kogutenko12;

import java.io.*;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * The type Interactive console.
 */
public class InteractiveConsole {
    ...

    /**
     * Start console.
     */
    public void startConsole() {
        try {
            System.out.print("Input your nickname: ");
            p = Pattern.compile("[\\w]{2,8}");
            String nick = scanner.nextLine();
            m = p.matcher(nick);
            if (m.matches()) {
                setNickname(nick);
            } else {
                setNickname("NickName");
            }

            while (check) {
                System.out.println(
                        "1 / input   \t-\t input from file\n" +
                                "2 / show    \t-\t show information about shops\n" +
                                "3 / add     \t-\t add one shop\n" +
                                "4 / remove  \t-\t remove one shop\n" +
                                "5 / switch  \t-\t switch to another list\n" +
                                "8 / fresh   \t-\t find fresh product\n" +
                                "9 / sort    \t-\t sort linked list by fields\n" +
                                "0 / exit    \t-\t exit and save data\n");
                System.out.print(nickname + "@" + nickname + ": ");
                //regex
                input = scanner.nextLine();
```

```
            //
            switch (input) {
                case "\n": {
                    System.out.print(nickname + "@" + nickname + ": ");
                    break;
                }
                case "1": {
                    System.out.println("LinkedList\n");
                    System.out.print("what deserialization do you want?\n(1 - bin, 2 - xml, 3 -
txt)\n>>> ");
                    answerDeserialization = scanner.nextInt();
                    switch (answerDeserialization) {
                        case 1: {
                            helperL.deserializationBIN();
                            break;
                        }
                        case 2: {
                            helperL.deserializationXML();
                            break;
                        }
                        case 3: {
                            deserializationTXT();
                            break;
                        }
                        default: {
                            System.out.println("don't have this method ;((");
                            break;
                        }
                    }
                    break;
                }
                case "2": {
                    helperL.printList();
                    break;
                }
                case "3": {
                    Shops shop = new Shops();
                    shop.add();
                    helperL.add(shop);
                    break;
                }
                case "4": {
                    helperL.printList();
                    Scanner sc = new Scanner(System.in);
                    System.out.print("Enter number of id: ");
                    //regex
                    int id = sc.nextInt();
                    //
                    if (id < 0 || id > helperL.size()) {
                        throw new Exception("out of range!!!!!");
                    } else if (!helperL.remove(id)) {
                        System.out.println("NOT FOUND");
```

```
                } else {
                    helperL.printList();
                }
                break;
            }
            case "0": {
                System.out.print("What save do you want? (1 - .txt; 2 - .bin; 3 - .xml)\n>>> ");
                Integer answ;
                //regex
                while (true) {
                    p = Pattern.compile("[123]");
                    answ = scanner.nextInt();
                    m = p.matcher(answ.toString());
                    if (m.matches()) {
                        break;
                    } else {
                        System.out.println("Enter info correctly!!!");
                    }
                }

                //
                switch (answ) {
                    case 1: {
                        serializationTXT();
                        break;
                    }
                    case 2: {
                        helperL.serializationBIN();
                        break;
                    }
                    case 3: {
                        helperL.serializationXML();
                        break;
                    }
                    default: {
                        System.out.println("We dont save your array (");
                        break;
                    }
                }
                check = false;
                break;
            }
            case "9": {
                System.out.println("Entrance to 9(sorting)");
                helperL = sort(helperL);
                System.out.println("list after:\n");
                System.out.println("\n---------------------------------\n");
                helperL.printList();
                System.out.println("\n---------------------------------\n");
            }
            case "8": {
                System.out.println(findFresh());
```

```
                break;
            }
            default: {
                System.out.println("(" + input + ") I don't know this command :(");
                break;
            }
        }
    }
    System.out.println("GOOD BEY!!!");
} catch (Exception e) {
    System.out.println(e);
    check = false;
}
}

...

}
```

3 ВАРІАНТИ ВИКОРИСТАННЯ

У інтерактивному вигляді буде складно показати результати виконання паралельної обробки, тому покажу вивід результату в автоматичному режимі. Я не використовував штучну затримку, бо кількість часу обробки є превишує мілісекунду яку підтримє лічільник часу Java.

В паралельному виконанні розрахунки швидші за обробку без створення додаткових ниток, але потрібно розуміти коли потрібно використовувати потоки, бо є випадки де послідовне виконання швидше.

```
                                thread
average time of printing is        7.9375ms
average time of deserialization is 34.0ms
average time of serialization is   8.0ms
average time of sorting is         14.0ms


                                      no-thread
average time of printing is        6.9375ms
average time of deserialization is 14.0ms
average time of serialization is   50.0ms
average time of sorting is         6.0ms
```

Рисунок 13.2 - час виконання різних методів із потоками.

ВИСНОВКИ

Ознайомився з моделлю потоків Java. Організував паралельне виконання декількох частин програми.