13. Паралельне виконання. Багатопоточність

- **Мета:** Ознайомлення з моделлю потоків *Java*.
- Організація паралельного виконання декількох частин програми.
-

1 ВИМОГИ

1.1 Розробник

Інформація про розробника:

- Шарма Олександр Раджнішович
- НТУ “ХПІ” КІТ-119б
- Варіант 2(25)

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.

1.3 Задача

Кадрове агентство. Знайти всі вакансії, де потрібні викладачі (педагоги, вчителі) зі стажем не менше 10 років, які знають англійську мову та володіють автомобілем.

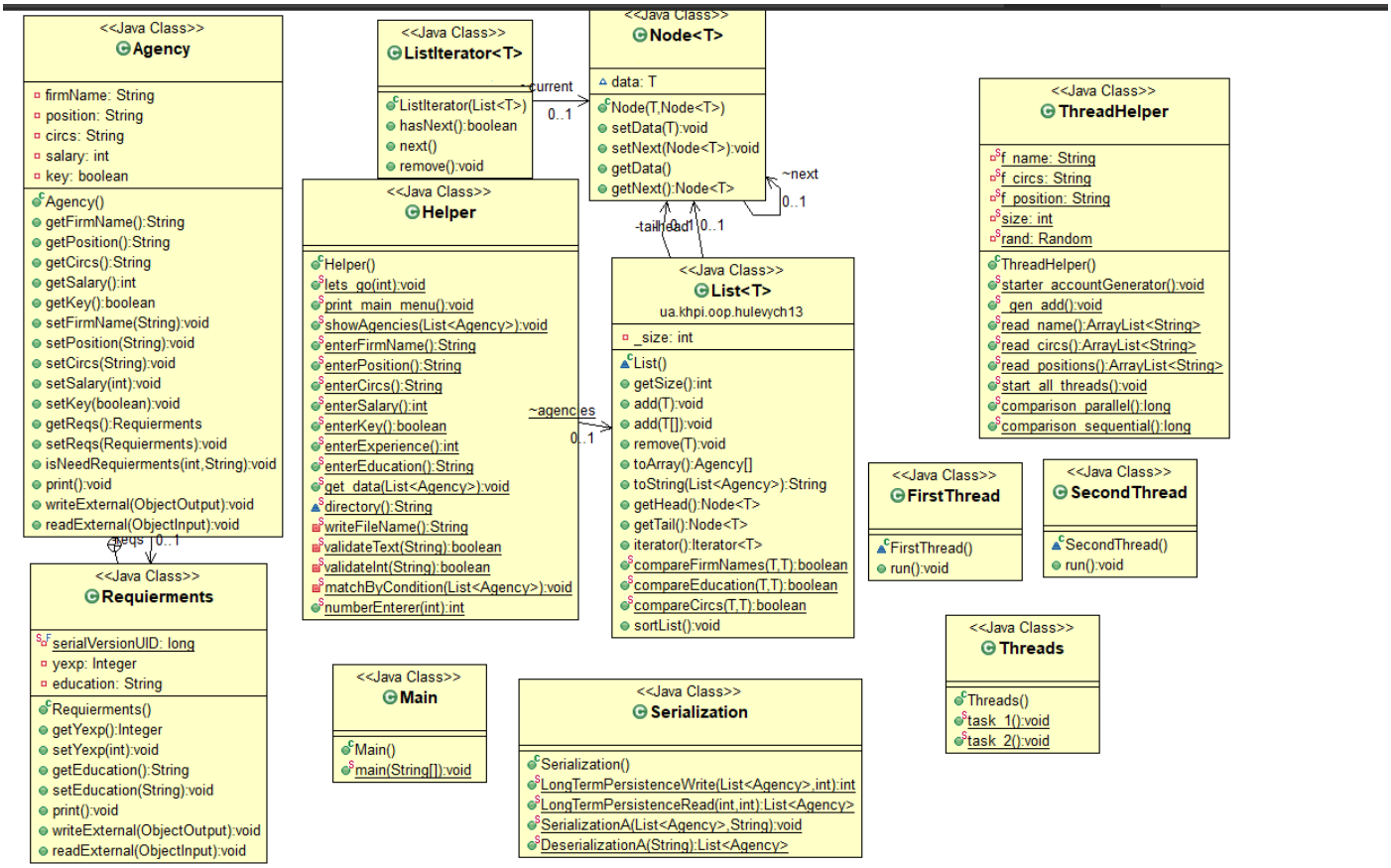
2 ОПИС ПРОГРАМИ

2.1 Засоби ООП

Композиція, інкапсуляція.

2.2 Ієрархія та структура даних

Рисунок 1 – діаграма класів



2.3 Важливі фрагменти програми

```
    }
    public static void start_all_threads() {
        System.out.println("Set the timer [0 - 100 000 ms]: ");
        int timer_num = Helper.numberEnterer(100000);
        System.out.println("Starting all threads...");

        FirstThread first = new FirstThread();
        Thread t1 = new Thread(first,"FirstThread");

        SecondThread second = new SecondThread();
        Thread t2 = new Thread(second,"SecondThread");


        t1.start();
        t2.start();
        Timer timer = new Timer(timer_num, new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent event) {
                System.out.println("Interrupting thread...");
                t1.interrupt();
                t2.interrupt();
            }
        });
        timer.setRepeats(false);
        timer.start();
        try {
            t1.join();
            t2.join();
            timer.stop();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Finishing all threads...");
    }
```

Рисунок 2 – Генерація даних та виконання завдання

3 ВАРІАНТИ ВИКОРИСТАННЯ

Програма створена для роботи з прикладною задачею. Для коректної роботи були реалізовані методи введення та отримання даних, також дані приховані від користувача, щоб не порушувати суттєвість об’єкту.

```
11.Thread count
12.Parallel and sequental comparsion
10
Enter the amount of accounts to be generated [0 - 100 000 000]
100
Starting generation...

Finished
Set the timer [0 - 100 000 ms]:
10000
Starting all threads...
First Thread started
Second Thread started
Second Thread finished. Vacanties with add conditions : 51
First Thread finished. Firms with name Daxx : 0
Finishing all threads...
```

Рисунок 3 – Генерація даних та виконання завдання

ВИСНОВКИ

В даній лабораторній роботі було розроблено методи паралельної обробки даних. Набуто навичок об’єктно-орієнтованого підходу.