Тема 13. Паралельне виконання. Багатопоточність
Мета:

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

1 ВИМОГИ

1.1 Розробник

Інформація про розробника:
- Гряник Георгій Володимирович
- КІТ-119Д;
- 6 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

2 ОПИС ПРОГРАМИ

2.1 Засоби ООП

Thread- це потік виконання в програмі. Віртуальна машина Java дозволяє додатку мати кілька потоків виконання, що працюють одночасно.

2.2 Ієрархія та структура класів

Клас “PoliceFile ” – описує поліцейську картотеку з можливістю додавати та виводити дані класу. Клас описує дані про злочинця відповідно до завданн. Клас «Date” – опису формат часу :день, місяць, рік . Створений для ергономічного запису дат відомостей про злочинця. Клас « Console_program» - клас керування програми , створений щоб надавати користувачеві можливість керувати програмою. Клас «ContainerList» - клас-контейнер створений для зберігання даних у список. Реалізовано додавання, видалення та інші можливості для керування даними. Клас «Serializator» - клас розроблений для збереження даних контейнеру у файл. При цьому зберігання прохоже у звичайний файл та файл типу .xml. При цьому в класі реалізовано методи для відновлення даних як із звичайного файлу так із .xml файлу. Клас Console_File розроблений для роботи із файлами розміщені в директоріях. Цей клас забезпечує можливість користувачеві обирати файл та перемінятися між директоріями. Клас «Helper» - реалізація допоміжних методів які реалізують

допоміжні дії в основній програмі. Клас Obshchak – загальна область пам’яті. ProcessProcesses- клас контролювання процесів: виклик процесів та завершення.

2.3 Важливі фрагменти програми

```
/**
 * @author <Георгій>
 *
 */

public class Obshchak {

	public static ContainerList<PoliceFile> List=new ContainerList<PoliceFile>();
	public static Boolean stop ;
	public static int TimeOut=0;
	public static int Choice=0;

}

private static final double DIVIDER = 1_000_000;

	ProcessProcesses()
	{
		long start = System.nanoTime();
		Thread.currentThread().setName("ProcessProcesses-"+Obshchak.Choice);
		start();
		System.out.print("\n"+Thread.currentThread().getName()+" - закінчив роботу;
"+"Тривалість роботи: "+ (System.nanoTime() - start) / DIVIDER*1000+"\n");

	}

	@SuppressWarnings("deprecation")
	public void run() {

		try {

		 Serializator<ContainerList<PoliceFile>> serializator=new
Serializator<ContainerList<PoliceFile>>();
		switch(Obshchak.Choice)///пошук введеної команди
		{

		case 6: //System.out.print("\n\nЗбереження даних:"+
serializator.serialization());
			Obshchak.stop = true;
			Thread t =new Thread ( serializator::serialization);
			 t.start();

			for(;;)
			{
				Thread.sleep(10);
				if(Obshchak.stop==false)break;
			}
		  if(Obshchak.TimeOut!=0)
		  {
			Thread.sleep(Obshchak.TimeOut);
			t.stop();
			System.out.print("примусово зупинено");
		  }
		break;
		case 8:Thread sort =  new Thread ( Helper::Sort);
```

```
		Obshchak.stop=true;
		for(;;)
		{
			Thread.sleep(10);
			if(Obshchak.stop==false)break;

		}
	if(Obshchak.TimeOut!=0)
	{
		Thread.sleep(Obshchak.TimeOut);
		sort.stop();
		System.out.print("примусово зупинено");

	}

	break;
		case 9:Thread reg=  new Thread (Helper::AppliedTask);///time here
		long startReg = System.nanoTime();
		Thread.currentThread().setName("reg");
		reg.start();

	if(Obshchak.TimeOut!=0)
	{
		Thread.sleep(Obshchak.TimeOut);
		reg.stop();
		System.out.print("примусово зупинено");
	}else reg.join();
		System.out.print("\n"+Thread.currentThread().getName()+" - закінчив роботу;
"+"Тривалість роботи: "+ (System.nanoTime() - startReg) / DIVIDER*1000+"\n");

		break;

		case 10:
			Thread r=new Thread(Helper::romove);

			Obshchak.stop=true;
			r.start();
			for(;;)
			{
				Thread.sleep(10);
				if(Obshchak.stop==false)break;

			}
		  if(Obshchak.TimeOut!=0)
		  {
			Thread.sleep(Obshchak.TimeOut);
			r.stop();
			System.out.print("примусово зупинено");

		  }


		case 11: Thread clear= new Thread( Obshchak.List::clear );///time here??
				clear.start();
			  if(Obshchak.TimeOut!=0)
			  {
				Thread.sleep(Obshchak.TimeOut);
				clear.stop();
				System.out.print("примусово зупинено");

			  }
		break;
```

```
                case 13:
                        Thread d=new Thread(serializator::deserializtionXML) ;
                        d.start();
                        Obshchak.stop=true;
                        for(;;)
                        {
                                Thread.sleep(10);
                                if(Obshchak.stop==false)break;
                        }
                  if(Obshchak.TimeOut!=0)
                  {
                        Thread.sleep(Obshchak.TimeOut);
                        d.stop();
                        System.out.print("примусово зупинено");
                  }
                  break;


                }
                }catch(Exception e)  {    //for (int i=0;i<10;i++)System.out.println(" EROR
EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR
EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR
EROR EROR EROR EROR EROR EROR ");
                        System.out.print("\n\n\nТрапилась помилка. Але тепер все
добре!!\n\n");
                        System.out.println(e);}


        }


}
 ////////////////////////////////////
```

3 ВАРІАНТИ ВИКОРИСТАННЯ

Паралельна обробка дозволяє не чекати завершення довгого процесу, наприклад збереження даних, і продовжити основну роботу програми.

Використання парольної обробки даних зменшить час роботи користувача з програмою, користувач може запросити від програми декілька завдань, і по мірі виконання поверне результат в незалежності від роботи основної програми. Єдина умова роботи це робота основної програми, тобто неповинна бути закритою.

Ім'я	По-батькові	Прізвище	дата народження	дата ОПВ	дата останнього звільнення	дати судимостей
Оксана	Вікторівна	Анапенко	03.01.1995	14.12.2016	04.05.2019	02.12.2005
Пурпел	Олегович	Чорний	14.12.1993	22.03.2019	12.12.2020	05.03.2009
Павло	Олександрович	Панапенко	12.12.1920	12.12.1980	12.12.1990	02.12.1940 03.12.1950

Рисунок 1. Вивід даних (частина великого списку)

```
 ваша команда:
 deserializtionXML: true8

ProcessProcesses-8 - закінчив роботу;
Сортувати:
  1. Ім`я
  2. по-Батькові
  3. Прізвище
  4. датою народження
  5. датою останнього позбавлення волі
  6. датою останнього звільнення
in: 1
clear - почав роботу
clear - закінчив роботу


 Оберіть команду:
*1 - Вивести поточні записані данні
*2 - Додати данні
*3 - Оновити дані
*4 - пошук
*5 - Кількість злочинців у базі
*6 - зберегти дані(save)
*7 - перетворення у масив, перетворення у рядок
*8 - Сортувати
*9 - Знайти всіх ув'язнених не молодше 20 років з прізвищем, що починається з голосної та міст
*10 - Видалити за номером
*11 - очистити дані
*12 - встановити Time out
*13 (exit)-вийти

 ваша команда:  *сортування завершено
```

Рисунок 2. Результат роботи паралельного виконання процесів

ВИСНОВКИ

Під час виконання лабораторної роботи було набуто навички роботи з багатопоточностю, та паралельною обробкою даних.