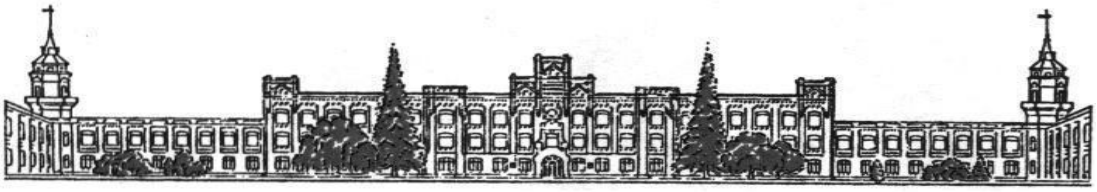


Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра системного проектування

Лабораторна робота №5

з дисципліни “Паралельні обчислення”

на тему: “**Отримання навичок використання механізму Future-Promise для взаємодії між потоками**”

Виконала:

студентка III курсу, групи ДА-01

Балуматкіна О. В.

Прийняла(в):

Київ-2023

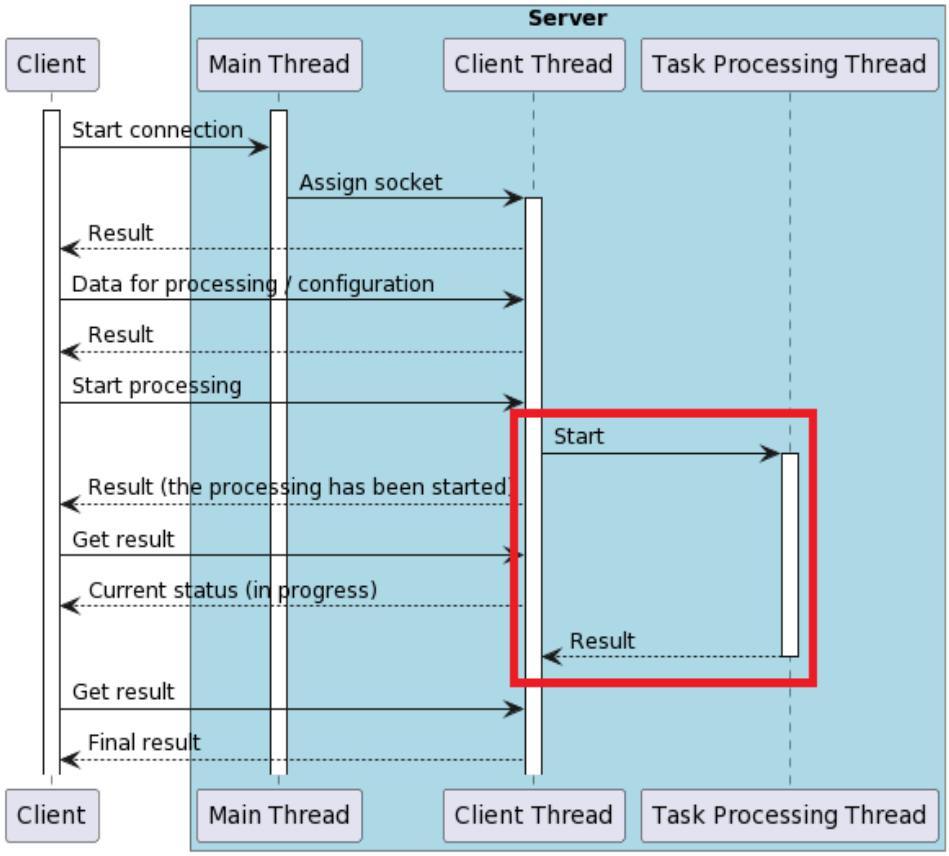
Зміст

Мета роботи

Розглянути механізм Future-Promise для асинхронної роботи з потоком обробки даних, навчитись імплементувати даний механізм на обраній мові програмування в рамках передачі інформації між клієнтом та сервером.

Завдання

1. В клієнт-серверному додатку з четвертої лабораторної роботи використати механізм future-promise для роботи з потоком обробки даних.
2. Дозволити клієнту запитувати поточний статус обробки без блокування потоку, який відповідає за комунікацію.



Хід роботи

Теоретична складова

Щоб використати механізм Future-Promise у простій клієнт-серверній програмі, необхідно виконати такі дії:

- Визначити функцію, яка буде виконуватися асинхронно на сервері. Ця функція має повертати значення, яке буде надіслано клієнту в результаті операції. У лабораторній роботі, наприклад, значенням може бути ціле число – кількість мілісекунд, які сервер виконував задачу, або – складніша структура даних.
- Створити об’єкт Promise в коді сервера та використати його для створення об’єкта Future.
- Передати майбутній об’єкт асинхронній функції, щоб вона могла встановити значення Promise об’єкта після виконання завдання.
- Зачекати, поки об'єкт буде виконаний, викликавши його метод "get". Це заблокує потік, поки результат не буде готовий.

В рамках мови програмування Java буде використано спеціальний клас для розумної роботи з логікою об’єктів типу Future – CompletableFuture.

Як зазначено вище, одразу потрібно виконати метод, який на сервері буде виконуватись асинхронно. Сама логіка та код був реалізований ще у четвертій лабораторній роботі, тож в даному протоколі буде описано безпосередньо використання Future для отримання клієнтом проміжних та фінальних результатів підрахунку моди та медіани переданого масиву випадкових чисел.

Future в обміні інформацією між клієнтом та сервером

Відповідно до завдання попередньої лабораторної роботи, необхідно було реалізувати можливість для клієнта отримувати результати розрахунків, що

проводяться на сервері, в будь який момент часу, навіть коли ще не весь масив було оброблено.

Для ефективного виконання такої вимоги без залучення складних структур з використанням додаткових потоків, внаслідок чого можна було б легко заплутатись, була використана логіка об’єкту Future та відповідний Java клас CompletableFuture.

ClientHandler.java:

```
CompletableFuture<Double[]> future = new CompletableFuture<>();
…
case "3" -> {
    serverResponse(dos, "Option 3. Starting processing.\n");

    // Future starting
    int finalNumbThreads = nThreads;
    future = CompletableFuture.supplyAsync(() ->
            processVector(array, finalNumbThreads));
}
case "4" -> {
    serverResponse(dos, "Option 4. Getting results.");

    if (future != null && future.isDone()) {
        Double[] result = future.getNow(null);
        serverResponse(dos, "Result:\nMode: " + result[0] +
                "\nMedian: " + result[1] + "\n");
        future = null;
    } else {
        serverResponse(dos, "At the moment results are not ready.\n");
    }
}
```

У вищенаведеному коді, зокрема case “3” можна побачити, що застосовується метод .supplyAsync() для ініціалізації асинхронного обчислення та створення нового об’єкта CompletableFuture, що міститиме у собі результат поточного обчислення.

У case “4”, в свою чергу, виконується функціонал передачі клієнту проміжного та фінального результатів, з обробкою випадку, коли клієнт запитує результати обчислень занадто рано і жоден з виокремлених потоків ще не встиг провести підрахунки (At the moment results are not ready.), внаслідок чого клієнт може трохи зачекати та запитати результати обчислення знову.

Висновок

У ході виконання даної лабораторної роботи було розглянуто та досліджено механізм Future-Promise, що було також імплементовано у коді обміну даними між певним сервером та декількома клієнтами, а саме механізм було застосовано при отриманні клієнтами результатів обрахунків моди та медіани масиву чисел з конкретного клієнта в будь-які моменти часу, навіть до того, як весь масив було оброблено та підраховано.

Посилання на гіт: https://github.com/balumatkina/parallel-computing-kpi.git

Додаток А

Server.java

```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {

    public Server(int portNumb) {
        try {
            ServerSocket serverSocket = new ServerSocket(portNumb);
            System.out.println("Server started: " + serverSocket);
            System.out.println("Waiting for client..\n");

            while (true) {
                Socket clientSocket = null;
                try {
                    clientSocket = serverSocket.accept();
                    System.out.println("Client connected: " + clientSocket +
"\n");

                    // Multi client handle
                    ClientHandler clientHandler = new
ClientHandler(clientSocket);
                    Thread thread = new Thread(clientHandler);
                    thread.start();

                } catch (Exception e) {
                    clientSocket.close();
                    e.printStackTrace();
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        new Server(6060);
    }
}
```

Додаток Б

ClientHandler.java

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.*;
import java.util.concurrent.*;

public class ClientHandler implements Runnable {
    private Socket clSocket;
    private ExecutorService executor;
    private DataOutputStream dos;
    private DataInputStream dis;

    public ClientHandler(Socket clSocket) throws IOException {
        this.clSocket = clSocket;
        dis = new DataInputStream(clSocket.getInputStream());
        dos = new DataOutputStream(clSocket.getOutputStream());
    }

        @Override
        public void run () {
        try {
            CompletableFuture<Double[]> future = new CompletableFuture<>();
            int length = 0, nThreads = 0;
            boolean isConnected = true;
            String clientCommand;
            ArrayList<Double> array = new ArrayList<>();

            String serverCommands = """
                    This server provides calculations of mode
                    and median of a given vector of numbers.
                    Options:
                    1. Setting configurations (vector length, number of
threads);
                    2. Sending a vector to the server;
                    3. Starting processing;
                    4. Getting results of counted mode and median;
                    5. Disconnecting from the server.
                    """;
            dos.writeUTF(serverCommands);

            while (isConnected) {
                clientCommand = dis.readUTF();
                switch (clientCommand) {
                    case "1" -> {
                        serverResponse(dos, "Option 1. Getting
configurations.");

                        String[] configs = dis.readUTF().split(" ");
                        length = Integer.parseInt(configs[0]);
                        nThreads = Integer.parseInt(configs[1]);

                        serverResponse(dos, "Received data:\nArray length: "
                                + length + "\nThreads amount: " + nThreads +
```

```
"\n");
                    }
                    case "2" -> {
                        array.clear();
                        serverResponse(dos, "Option 2. Getting array.");

                        for (int i = 0; i < length; i++) {
                            array.add(dis.readDouble());
                        }

                        serverResponse(dos, "Array successfully received.\n");
                    }
                    case "3" -> {
                        serverResponse(dos, "Option 3. Starting processing.\n");

                        // Future starting
                        int finalNumbThreads = nThreads;
                        future = CompletableFuture.supplyAsync(() ->
                                processVector(array, finalNumbThreads));
                    }
                    case "4" -> {
                        serverResponse(dos, "Option 4. Getting results.");

                        if (future != null && future.isDone()) {
                            Double[] result = future.getNow(null);
                            serverResponse(dos, "Result:\nMode: " + result[0] +
                                    "\nMedian: " + result[1] + "\n");
                            future = null;
                        } else {
                            serverResponse(dos, "At the moment results are not
ready.\n");
                        }
                    }
                    case "5" -> {
                        serverResponse(dos, "Option 5. Breaking connection.");

                        isConnected = false;
                        if (executor != null) {
                            executor.shutdown();
                        }

                        serverResponse(dos, "Connection stopped.\n");
                    }
                    default -> System.err.println("Unknown operation.");
                }
            }

            try {
                if (dis != null) {
                    dis.close();
                }
                if (dos != null) {
                    dos.close();
                }
                if (clSocket != null) {
                    clSocket.close();
                }
                System.out.println("Client closed: " + clSocket);
            } catch (IOException e) {
```

```
                e.printStackTrace();
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public Double[] processVector(ArrayList<Double> array, int nThreads) {
        Double mode = 0.0, counter = 0.0, finalMedian = 0.0;
        int length = array.size();

        executor = Executors.newFixedThreadPool(nThreads);

        ArrayList<Double> medianArray = new ArrayList<>();
        // Task list for different threads
        List<Calculation> tasks = new ArrayList<>();
        // For Future
        CompletionService<Double[]> completionService = new
ExecutorCompletionService<>(executor);

        for (int i = 0; i < nThreads; i++) {
            int startIndex = length / nThreads * i;
            int endIndex = (i == (nThreads - 1)) ? length : (length / nThreads *
(i + 1));

            endIndex = Math.min(endIndex, length);

            tasks.add(new Calculation(array, startIndex, endIndex));
        }

        try {
            // Submitting tasks for processing
            for (int i = 0; i < nThreads; i++) {
                completionService.submit(tasks.get(i));
            }

            mode = array.get(0);
            finalMedian = array.get(0);
            for (int i = 0; i < nThreads; i++) {
                Future<Double[]> completedFuture = completionService.take();
                Double[] res = completedFuture.get();

                if (res[2] > counter) {
                    mode = res[1];
                    counter = res[2];
                }

                medianArray.add(res[0]);
                Collections.sort(medianArray);
                int middle = medianArray.size() / 2;
                if (medianArray.size() % 2 == 0) {
                    finalMedian = (medianArray.get(middle - 1) +
medianArray.get(middle)) / 2.0;
                } else {
                    finalMedian = medianArray.get(middle);
                }

            }
```

```
        } catch (InterruptedException | ExecutionException e) {
            e.printStackTrace();
        }

        return new Double[]{mode, finalMedian};
    }

    private static void serverResponse(DataOutputStream dos, String message)
throws IOException {
        dos.writeUTF(message);
        System.out.println(message);
    }
}
```

Додаток В

Calculation.java

```
import java.util.*;
import java.util.concurrent.*;

public class Calculation implements Callable<Double[]> {

    private final ArrayList<Double> vector;
    private final int startIndex;
    private final int endIndex;
    private double mode;
    private double median;
    private Double counter = 0.0;

    public Calculation(ArrayList<Double> vector, int startIndex, int endIndex) {
        this.vector = vector;
        this.startIndex = startIndex;
        this.endIndex = endIndex;
    }

    @Override
    public Double[] call() {
        return getModeMedian();
    }

    private void calcMedian() {
        List<Double> sublist = vector.subList(startIndex, endIndex);
        Double[] vectorChunk = sublist.toArray(new Double[0]);

        Arrays.sort(vectorChunk);
        int middle = vectorChunk.length / 2;
        if (vectorChunk.length % 2 == 0) {
            median = (vectorChunk[middle - 1] + vectorChunk[middle]) / 2.0;
        } else {
            median = vectorChunk[middle];
        }
    }

    private void calcMode() {
        List<Double> sublist = vector.subList(startIndex, endIndex); // Add 1 to
include endIndex
        Double[] vectorChunk = sublist.toArray(new Double[0]);

        Map<Double, Double> freqMap = new HashMap<>();
        // Iterate through the array and update the frequency of each number in
the HashMap
        for (double num : vectorChunk) {
            if (freqMap.containsKey(num)) {
                freqMap.put(num, freqMap.get(num) + 1.0);
            } else {
                freqMap.put(num, 1.0);
            }
        }

        // Find the mode and its frequency
        for (Map.Entry<Double, Double> entry : freqMap.entrySet()) {
```

```
            if (entry.getValue() > counter) {
                mode = entry.getKey();
                counter = entry.getValue();
            }
        }
    }

    private Double[] getModeMedian() {
        calcMode();
        calcMedian();
        return new Double[] {median, mode, counter};
    }
}
```

Додаток Г

Client.java

```
import java.io.*;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.Scanner;

public class Client {
    private Socket socket;
    private DataOutputStream dos;
    private DataInputStream dis;

    public Client (String address, int port) {
        Scanner scanner = new Scanner(System.in);
        int arrayLength = 0, nThreads = 0;
        try {
            socket = new Socket(address, port);
            System.out.println("The client is connected\n" + socket +"\n");

            dos = new DataOutputStream(socket.getOutputStream());
            dis = new DataInputStream(socket.getInputStream());
        } catch (IOException u) {
            u.printStackTrace();
        }

        try {
            serverRead(dis);

            while (true) {
                System.out.println("Enter command number (1-5): ");
                String command = scanner.nextLine();

                dos.writeUTF(command);

                switch (command) {
                    case "1" -> {
                        serverRead(dis);

                        while (!(arrayLength > 0)) {
                            System.out.println("Print array length:");
                            arrayLength = scanner.nextInt();
                        }

                        while (!(nThreads > 0)) {
                            System.out.println("Print threads amount:");
                            nThreads = scanner.nextInt();
                        }
                        scanner.nextLine();

                        dos.writeUTF(arrayLength + " " + nThreads);

                        serverRead(dis);
                    }
                    case "2" -> {
```

```
                        List<Double> array =
generateRandDoubleArrList(arrayLength);

                        serverRead(dis);

                        for (double num : array) {
                            dos.writeDouble(num);
                        }

                        serverRead(dis);
                    }
                    case "3" ->
                            serverRead(dis);
                    case "4" -> {
                        serverRead(dis);
                        serverRead(dis);
                    }
                    case "5" -> {
                        serverRead(dis);
                        serverRead(dis);

                        try {
                            if (dis != null) {
                                dis.close();
                            }
                            if (dos != null) {
                                dos.close();
                            }
                            if (socket != null) {
                                socket.close();
                            }
                            if (scanner != null) {
                                scanner.close();
                            }
                        } catch (IOException e) {
                            e.printStackTrace();
                        }

                        return;
                    }
                    default -> System.out.println("Invalid operation number.
Please try again.");
                }
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
//        Client client = new Client("localhost", 6060);
        try {
            new Client("localhost", 6060);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
    private static List<Double> generateRandDoubleArrList (int arrLength) {
        Random random = new Random();
        List<Double> array = new ArrayList<>();

        for (int i = 0; i < arrLength; i++) {
            array.add(random.nextDouble(100));
        }
        return array;
    }

    private static void serverRead(DataInputStream dis) throws IOException {
        String serverResponse = dis.readUTF();
        System.out.println("SERVER: " + serverResponse);
    }

}
```

Контрольні питання:

1. *Яка різниця між об'єктами Future та Promise?*
   Future є інтерфейсом, який представляє обіцянку отримання результату певної асинхронної операції та надає методи для перевірки стану завершення операції й отримання результату. **Promise** – інтерфейс або клас, який дозволяє створювати асинхронні операції та керувати ними, дає можливість обіцяти результат, який буде доступний у майбутньому.
2. *Як отримати об'єкт Future?*
   Через використання **ExecutorService**, його методами submit або invokeAll, щоб виконати асинхронну задачу і отримати об'єкт Future; через використання **CompletableFuture**, що є розширеним варіантом звичайного Future, можна використовувати методи supplyAsync або runAsync, щоб виконати асинхронну задачу і отримати об'єкт CompletableFuture, який можна преобразувати в об'єкт Future; через використання розширених бібліотек або фреймворків (Java Concurrency API або Spring Framework).
3. *Яка різниця між методом get() і методом wait() об'єкта Future?*
   get(): виклик блокує виконання поточного потоку допоки результат операції не буде доступний, повертає результат асинхронної операції або кидає TimeoutException або CancellationException, якщо операція ще не завершена, то метод буде очікувати на її завершення. **wait():** призупиняє виконання поточного потоку та чекає доступного результату, використовується для реалізації механізмів спілкування між потоками, не повертає результат та не кидає винятків, а просто чекає.
4. *Яка різниця між методом valid() та методом wait_for() об'єкта Future?*
   valid(): перевіряє, чи об'якт Future є дійсним, повертає true якщо Future створений та пов'язаний з певною асинхронною операцією, false в іншому випадку, метод не блокує виконання поточного потоку та не очікує результатів. **wait_for():** очікує результату асинхронної операції протягом

певного періоду часу, вказаного як параметр методу, якщо результат за цей час стає доступним – повертає його, якщо результат не доступний – повертає null, метод блокує виконання поточного потоку на встановлений час очікування.

5. *Як обробляти exceptions, які можуть виникнути всередині асинхронної операції?*

   В рамках даної лабораторної роботи усі винятки, які можуть кидатись кодом у ході виконання асинхронної операції, оброблюються через try{…} catch (InterruptedException | ExecutionException e) {e.printStackTrace();}. Іншим варіантом може бути створення методу по типу CompletableFuture<String> handleFuture для обробки результату асинхронної операції, або ж винятку, якщо такий був кинутий (в першому варіанті метод просто повертає значення, в другому – оброблює виняток).