Laboratorium 12

rozwinięcie laboratorium 9

Cel ćwiczenia:

Zrównoleglenie zapisu i odczytu danych pracowników do/z plików archiwalnych.

Materiały:

1. https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html
2. https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/util/concurrent/CompletableFuture.html
3. https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/util/concurrent/Executors.html
4. https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/util/concurrent/Executor.html
5. https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/util/concurrent/ExecutorService.html

Zagadnienia:

1. CompletableFuture
2. ExecutorService

Uwaga:

Dane każdego pracownika przechowuj w osobnym pliku!

Polecenie:

1) Zmodyfikuj istniejący kod zapisu danych pracownika w następujący sposób:
   1. Wydziel zapis do pliku danych pojedynczego pracownika do odrębnej metody zwracającej typ `void`.
   2. Zadeklaruj executor z pulą (`FixedThreadPool`) o zadanej liczbie wątków, np. 10.
   3. Zadeklaruj tablicę obiektów `CompletableFuture<Void>` o rozmiarze równym liczbie pracowników.
   4. W pętli dla każdego pracownika wywołaj metodę `runAsync()` z klasy `CompletableFuture` (w parametrach przekaż `callback` do metody zapisującej pracownika do pliku oraz executor) budując w ten sposób obiekty `CompletableFuture<Void>`. Wpisz je do kolejnych elementów tablicy.
   5. W kolejnej pętli, wywołując metodę `get()` lub `join()` dla każdego obiektu z tablicy, odczytaj czy każdy z wątków zakończył się poprawnie. Powinieneś zbudować obsługę wyjątków `InterruptedException`, `ExecutionException`.
   6. Możesz również wykorzystać metodę `allOf()` z klasy `CompletableFuture` i przekształcić tablicę obiektów w jeden obiekt `CompletableFuture`, a następnie dla niego wywołać pojedynczą metodę `get()` lub `join()`.

2) Zmodyfikuj istniejący kod odczytu danych pracownika z pliku do pamięci w poniższy sposób:
   1. Wydziel odczyt z pliku danych pojedynczego pracownika do metody zwracającej typ `Pracownik`.
   2. Zadeklaruj executor z pulą (`FixedThreadPool`) o zadanej liczbie wątków, np. 10.
   3. Zadeklaruj listę obiektów `CompletableFuture<Pracownik>`.
   4. Odczytaj listę plików ze wskazanej lokalizacji.
   5. W pętli dla każdego pliku wywołaj metodę `supplyAsync()` z klasy `CompletableFuture` (w parametrach przekaż callback do metody odczytującej pracownika z pliku oraz executor) budując w ten sposób obiekty `CompletableFuture<Pracownik>` i przypisz je do kolejnych elementów listy.
   6. Wywołując metodę `get()` lub `join()` dla każdego obiektu z listy odczytaj czy wątek zakończył się poprawnie (powinieneś zbudować obsługę wyjątków `InterruptedException`, `ExecutionException`). Zapisz w pamięci obiekty `Pracownik` zwrócone przez prawidłowo zakończone wątki.

■

Uwaga! Projekt zawierający kody źródłowe programu po sprawdzeniu na zajęciach laboratoryjnych musi zostać skompresowany w jeden plik archiwum ZIP lub GZIP i przesłany na adres kkraska@zut.edu.pl:

1) plik z kodem źródłowym musi mieć nazwę:

   L12_{Forma i stopień studiów}_{Grupa z planu zajęć}_{NAZWISKO}_{IMIĘ}.(zip|gz)

   np.: L12_S1_PO1_J.Java_L_A_KRASKA_KRZYSZTOF.zip

2) plik musi zostać wysłany z konta poczty uczelni (zut.edu.pl),
3) nagłówek maila musi mieć postać:

   L12_{Forma i stopień studiów}_{Grupa z planu zajęć}_{NAZWISKO}_{IMIĘ}

   np.: L12_S1_PO1_J.Java_L_A_KRASKA_KRZYSZTOF

4) email nie powinien zawierać żadnej treści (tylko załącznik).