LABORATORIUM

ZAAWANSOWANE PROGRAMOWANIE OBIEKTOWE

programowanie współbieżne

---

1. Wstęp

Ćwiczenie ma na celu zapoznanie z programowaniem współbieżnym

2. Przebieg ćwiczenia

- Utworzyć projekt maven lub gradle.
- Przekleić zawartość załączników do pliku Item.java, uzupełnić brakujące importy.

Wątki

- Utworzyć listę 100 obiektów klasy Item
- Utworzyć 4 wątki produkujące (wywołujące metodę produceMe()), oraz 3 wątki konsumujące (wywołujące metodę consumeMe()).
- Zaimplementować aplikację w taki sposób aby wszystkie obiekty z listy były "wyprodukowane" przez wątki produkujące i "skonsumowane" przez wątki konsumujące. Zrealizować to w taki sposób aby czas zminimalizować czas produkcji i konsumpcji wszystkich obiektów.

Pule wątków

- Utworzyć pulę wątków.
- Wykonać zadanie z poprzedniego podpunktu tak, aby wykorzystać w tym celu pulę wątków, z tym że bez podziału na wątki produkujące i konsumujące (pojedynczy wątek, w zależności od potrzeb może wykonywać obydwa zadania).
- Wykonać testy dla różnych typów puli wątków. Stream API
- Wykonać zadanie z poprzedniego punktu wykorzystując Stream API (parallelStream) Sprawozdanie

3. Sprawozdanie

- Zwięzły opis przebiegu ćwiczenia, wraz z kodami źródłowymi i czasami wykonania dla poszczególnych punktów.
- Wnioski z ćwiczeń - własne obserwacje i przemyślenia.

Item.java:

```
public class Item {
	private static final AtomicInteger COUNTER = new AtomicInteger();
	private final String name;
	private volatile boolean produced = false;
	private volatile boolean consumed = false;
	public Item() {
		this.name = "Item-" + COUNTER.getAndIncrement();
	}
	public synchronized void produceMe() {
		if (produced) {
			throw new RuntimeException(name + " already produced");
		}
		if (consumed) {
			throw new RuntimeException(name + " already consumed");
		}
		System.out.println("Producing: " + name);
		delay(2);
		produced = true;
		System.out.println("Produced: " + name);
	}
	public synchronized void consumeMe() {
		if (!produced) {
			throw new RuntimeException(name + " not produced yet");
		}
		if (consumed) {
			throw new RuntimeException(name + " already consumed");
		}
		System.out.println("Consuming: " + name);
		delay(3);
		consumed = true;
		System.out.println("Consumed: " + name);
	}
	private void delay(int seconds) {
		try {
```

```
                Thread.sleep(seconds * 1000);
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }
```