PROGRAMOWANIE OBIEKTOWE I GUI

dr inż. Michał Tomaszewski

katedra Metod Programowania
Polsko-Japońska Akademia Technik Komputerowych

WYKŁAD 04

Poruszyliśmy zagadnienia:

- Kolekcje

WYKŁAD 04

Poruszyliśmy zagadnienia:

- Kolekcje
- Strumienie

Plan wykładu:

- Strumienie cd

Plan wykładu:

- Strumienie cd
- Współbieżność

STRUMIENIE

`Stream API` nie ma nic wspólnego z strumieniami plikowymi!

STRUMIENIE

`Stream API` nie ma nic wspólnego z strumieniami plikowymi!

Strumienie są tworem pochodzącym z programowania funkcyjnego.

`Stream API` nie ma nic wspólnego z strumieniami plikowymi!

Strumienie są tworem pochodzącym z programowania funkcyjnego.

Implementują wzorzec programistyczny Monad, który przedstawia mechanizm powiązania metod w ten sposób że rezultat jednej staje się wejściem drugiej.

IDEA STREAM API

Stream reprezentują zbiór danych, na którym wykonuje się sekwencje operacji rozpoczynających się od wypełnienia, przez szereg modyfikacji, a kończącą się na dostarczeniu wyniku.

Stream reprezentują zbiór danych, na którym wykonuje się sekwencje operacji rozpoczynających się od wypełnienia, przez szereg modyfikacji, a kończącą się na dostarczeniu wyniku.

Operacje na strumienach są:

Stream reprezentują zbiór danych, na którym wykonuje się sekwencje operacji rozpoczynających się od wypełnienia, przez szereg modyfikacji, a kończącą się na dostarczeniu wyniku.

Operacje na strumienach są:

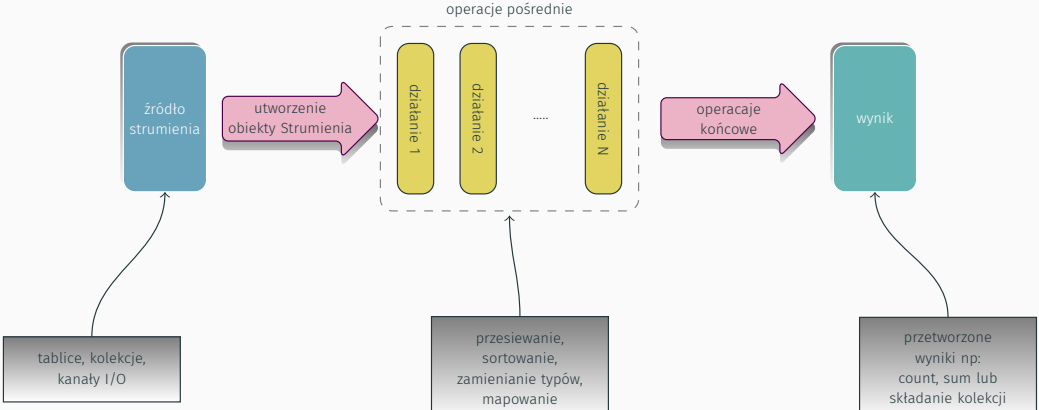
- pośrednie,

Stream reprezentują zbiór danych, na którym wykonuje się sekwencje operacji rozpoczynających się od wypełnienia, przez szereg modyfikacji, a kończącą się na dostarczeniu wyniku.

Operacje na strumienach są:

- pośrednie,
- końcowe.

Stream reprezentują zbiór danych, na którym wykonuje się sekwencje operacji rozpoczynających się od wypełnienia, przez szereg modyfikacji, a kończącą się na dostarczeniu wyniku.



WŁAŚCIWOŚCI

- Strumień nie przechowuje elementów na których działa.

WŁAŚCIWOŚCI

- Strumień nie przechowuje elementów na których działa.
- Operacje strumieniowe nie modyfikują swojego źródła.

WŁAŚCIWOŚCI

- Strumień nie przechowuje elementów na których działa.
- Operacje strumieniowe nie modyfikują swojego źródła.
- Operacje strumieniowe są „leniwe” (lazy) kiedy tylko jest to możliwe.

TWORZENIE STRUMIENI

Strumienie tworzy się:

TWORZENIE STRUMIENI

Strumienie tworzy się:

- na podstawie istniejących kolekcji – metoda `of`,

Strumienie tworzy się:

- na podstawie istniejących kolekcji – metoda `of`,

```
Stream<String> ryhme = Stream.of(
    "Ene", "due", "rabe",
    "chinczyk", "polknal" "zabe",
    "a", "zaba", "chinczyka",
    "co", "z", "tego", "wynika"
);
```

TWORZENIE STRUMIENI

Strumienie tworzy się:

- na podstawie istniejących kolekcji – metoda `of`,
- bez elementów – metoda `empty`,

TWORZENIE STRUMIENI

Strumienie tworzy się:

- na podstawie istniejących kolekcji – metoda `of`,
- bez elementów – metoda `empty`,

```
Stream<Integer> ints = Stream.empty();
```

TWORZENIE STRUMIENI

Strumienie tworzy się:

- na podstawie istniejących kolekcji – metoda `of`,
- bez elementów – metoda `empty`,
- produkując zestaw elementów – metoda `generate`,
- produkując zakres elementów – metoda `iterate`.

TWORZENIE STRUMIENI

Strumienie tworzy się:

- na podstawie istniejących kolekcji – metoda `of`,
- bez elementów – metoda `empty`,
- produkując zestaw elementów – metoda `generate`,
- produkując zakres elementów – metoda `iterate`.

```
Stream<Double> doubles = Stream.generate(Math::random);
```

TWORZENIE STRUMIENI

Strumienie tworzy się:

- na podstawie istniejących kolekcji – metoda `of`,
- bez elementów – metoda `empty`,
- produkując zestaw elementów – metoda `generate`,
- produkując zakres elementów – metoda `iterate`.

TWORZENIE STRUMIENI

Strumienie tworzy się:

- na podstawie istniejących kolekcji – metoda `of`,
- bez elementów – metoda `empty`,
- produkując zestaw elementów – metoda `generate`,
- produkując zakres elementów – metoda `iterate`.

```
Stream<BigInteger> integers = Stream.iterate(
    BigInteger.ZERO, n->n.add(BigInteger.ONE)
);
```

Przykład

Strumienie można zawęzić:

Strumienie można zawęzić:

- filter,

Strumienie można zawęzić:

- filter,
- map,

Strumienie można zawęzić:

- filter,
- map,
- flatMap.

Przykład

ZBIERANIE WYNIKÓW

Aby zebrać efekt pracy poszczególnych operacji strumienia można użyć interfejsu Collector.

ZBIERANIE WYNIKÓW

Aby zebrać efekt pracy poszczególnych operacji strumienia można użyć interfejsu Collector.

- filter,

ZBIERANIE WYNIKÓW

Aby zebrać efekt pracy poszczególnych operacji strumienia można użyć interfejsu Collector.

- filter,
- map,

ZBIERANIE WYNIKÓW

Aby zebrać efekt pracy poszczególnych operacji strumienia można użyć interfejsu Collector.

- filter,
- map,
- flatMap.

Przykład

PROGRAMOWANIE WSPÓŁBIEŻNE

WSPÓŁBIEŻNOŚĆ

Program jest **współbieżny**, jeśli jego wykonanie wiąże się z więcej niż jednym **przepływem strowania**.

WSPÓŁBIEŻNOŚĆ

Program jest **współbieżny**, jeśli jego wykonanie wiąże się z więcej niż jednym **przepływem strowania**.
Każdy przepływ sterowania jest realizowany przez odrębny **wątek**.

WSPÓŁBIEŻNOŚĆ

Program jest **współbieżny**, jeśli jego wykonanie wiąże się z więcej niż jednym **przepływem strowania**.
Każdy przepływ sterowania jest realizowany przez odrębny **wątek**.

Początkowo wykonuje się tylko wątek **główny** i wątki **systemowe**.

WSPÓŁBIEŻNOŚĆ

Program jest **współbieżny**, jeśli jego wykonanie wiąże się z więcej niż jednym **przepływem strowania**.

Każdy przepływ sterowania jest realizowany przez odrębny **wątek**.

Początkowo wykonuje się tylko wątek **główny** i wątki **systemowe**.

Niezależnie od liczby procesorów, program wielowątkowy zachowuje się tak, jakby każdy przepływ sterowania był realizowany przez oddzielny procesor.

Użycie klasy pochodnej od `Thread`:

```
class Runner
    extends Thread {

    public void run(){
        // ...
    }
}

// ...

public static void main(String[] args){
    new Runner().start();
}
```

Użycie klasy `Thread` przez implementację interfejsu `Runnable`:

```
class Program
    implements Runnable {

    public static void main(String[] args){
        new Thread(this).start();
    }

    public void run(){
        // ...
    }
}
```

Dziękuję