Multithreading. Streams. Serializácia. IO Stream

OLIVER UDVARDI 2.APRÍL 2026

2. Odovzdanie

2. Odovzdanie

- Dokumentácia má byť aj v rámci Github repozitára

2. Odovzdanie

- Dokumentácia má byť aj v rámci Github repozitára
- Časté chyby - run-time polymorfizmus

2. Odovzdanie

- Dokumentácia má byť aj v rámci Github repozitára
- Časté chyby - run-time polymorfizmus
- Nedeklarované princípy v dokumentácii

2. Odovzdanie

- Dokumentácia má byť aj v rámci Github repozitára
- Časté chyby - run-time polymorfizmus
- Nedeklarované princípy v dokumentácii
- Kompozícia vs. Agregácia

Test

Test • 09. Apríl 2026

Test

- 09. Apríl 2026
- 1. Beh - 9:00 CPU. Dvere sa otvárajú 8:50

Test

- 09. Apríl 2026
- 1. Beh - 9:00 CPU. Dvere sa otvárajú 8:50
- 2. Beh - 10:15 CPU. Keď dopíše prvý beh

Test

- 09. Apríl 2026
- 1. Beh - 9:00 CPU. Dvere sa otvárajú 8:50
- 2. Beh - 10:15 CPU. Keď dopíše prvý beh
- Vyhodnotenie dychotomické

Test

- 09. Apríl 2026
- 1. Beh - 9:00 CPU. Dvere sa otvárajú 8:50
- 2. Beh - 10:15 CPU. Keď dopíše prvý beh
- Vyhodnotenie dychotomické
- Zoraďovanie kódu, Dopĺňanie kódu, Multichoice, Vzory, OOP princípy - identifikovať, doplniť, navrhnúť..., Streams, junit, identifikovať korektný kód, napísať kód aby vrátil požadovaný výstup

Test

Test

- Do Utorka vám pošlem mail s inštrukciami.

Test

- Do Utorka vám pošlem mail s inštrukciami.
- Dostanete informáciu o tom, v ktorom behu ste zaradení

Test

- Do Utorka vám pošlem mail s inštrukciami.
- Dostanete informáciu o tom, v ktorom behu ste zaradení
- Konzultácie k testu - oznámim po teste podľa záujmu

Test

- Do Utorka vám pošlem mail s inštrukciami.
- Dostanete informáciu o tom, v ktorom behu ste zaradení
- Konzultácie k testu - oznámim po teste podľa záujmu
- Otázky budú z bodových kategórii podľa obtiažnosti - 0.25b, 0.5b, 1b, 2b

Test

- Do Utorka vám pošlem mail s inštrukciami.
- Dostanete informáciu o tom, v ktorom behu ste zaradení
- Konzultácie k testu - oznámim po teste podľa záujmu
- Otázky budú z bodových kategórii podľa obtiažnosti - 0.25b, 0.5b, 1b, 2b
- Čistý čas testu - 60 minút

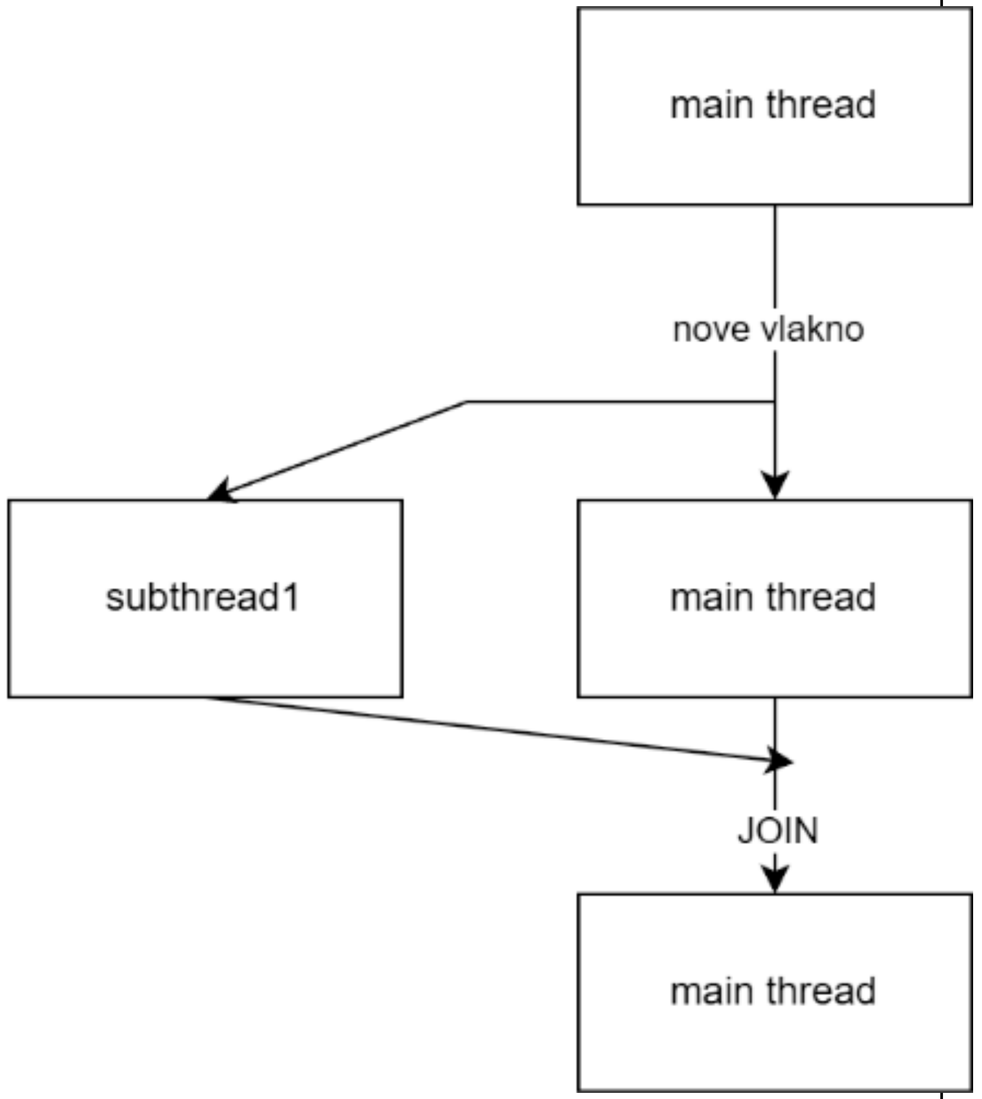
Zadanie 3

- Odovzdanie 26.04.2026
- Nezabudnite, že odovzdania v AIS a GitHub majú byť totožné. V GitHub musí byť všetko.
- Zadanie budete prezentovať na cvičeniach 9-12

Multithreading Synchronised

2 a viac vecí robíme naraz (súbežne)

- Vlákna:
- “Lightweight procesy”
- Spoločná pamäť
- Pozor na deterministickosť



Synchronized

Synchronized metódy

• V jednom čase iba jedno vlákno smie pracovať s objektom

• Každý objekt má zámok (lock), ale používajú sa iba v prípade

ak máme synchronizované časti kódu

• Synchronized na všetko?

• Deadlock

• Nevyužívame súbežnosť

• Atomic variables (AtomicInteger) – incrementAndGet,

compareAndSet

Streams
Serializácia
IO stream

Streams

Vychádzajú z funkcionálneho programovania, deklaratívne

- Možné ľahké paralelné spracovanie
- Začíname zdrojom (source),

  potom pracujeme (intermediate operations)

  a na konci je terminal expression, stream je

  možné “použiť/prejsť” iba raz
- Sú lenivé (lazy)
- Nemodifikujeme pôvodné dáta
- Collections podporujú streamy (.stream(), .parallelStream())

Source

Source

Intermediate operations - pracujú s dátami

Source

Intermediate operations - pracujú s dátami

.filter()

Source

Intermediate operations - pracujú s dátami

.filter()

.distinct()

Source

Intermediate operations - pracujú s dátami

.filter()

.distinct()

.peek()

Source

Intermediate operations - pracujú s dátami

.filter()

.distinct()

.peek()

Terminal operations - ukončujú Stream po nich sa vráti napr. List

Source

Intermediate operations - pracujú s dátami

.filter()

.distinct()

.peek()

Terminal operations - ukončujú Stream po nich sa vráti napr. List

.collect()

Source

Intermediate operations - pracujú s dátami

.filter()

.distinct()

.peek()

Terminal operations - ukončujú Stream po nich sa vráti napr. List

.collect()

.count()

Source

Intermediate operations - pracujú s dátami

.filter()

.distinct()

.peek()

Terminal operations - ukončujú Stream po nich sa vráti napr. List

.collect()

.count()

.findFirst()

Streams

Streams

Nie sú to dátové štruktúry

Streams

Nie sú to dátové štruktúry

`.filter()`

Streams

Nie sú to dátové štruktúry

`.filter()`

Berie podmienku a pustí ďalej
len tie entity, ktoré ju spĺňajú

Streams

Nie sú to dátové štruktúry

`.filter()`

Berie podmienku a pustí ďalej
len tie entity, ktoré ju spĺňajú

Streams

Nie sú to dátové štruktúry

`.filter()`

Berie podmienku a pustí ďalej
len tie entity, ktoré ju spĺňajú

`.sorted(comparing( ))`

Streams

Nie sú to dátové štruktúry

`.filter()`

Berie podmienku a pustí ďalej
len tie entity, ktoré ju spĺňajú

`.sorted(comparing( ))`

Hodnota ktorú porovnávame.
Napr. person.getName();
Zoradí podľa abecedy

Streams

Nie sú to dátové štruktúry

`.filter()`

Berie podmienku a pustí ďalej
len tie entity, ktoré ju spĺňajú

`.sorted(comparing( ))`

Hodnota ktorú porovnávame.
Napr. person.getName();
Zoradí podľa abecedy

`.map()`

Streams # Nie sú to dátové štruktúry

`.filter()`

Berie podmienku a pustí ďalej
len tie entity, ktoré ju spĺňajú

`.sorted(comparing( ))`

Hodnota ktorú porovnávame.
Napr. person.getName();
Zoradí podľa abecedy

`.map()`

Umožní nám trasformovať, s akými dátami ďalej budeme pracovať.
Napr. Som si zoradil objekty podľa mien a teraz budem pracovať len s ID
tých objektov. Namapujem Objekt k ID a ďalej v streame budú len ID

Streams # Nie sú to dátové štruktúry

`.filter()` Berie podmienku a pustí ďalej len tie entity, ktoré ju spĺňajú

`.sorted(comparing( ))` Hodnota ktorú porovnávame. Napr. person.getName(); Zoradí podľa abecedy

`.map()` Umožní nám trasformovať, s akými dátami ďalej budeme pracovať. Napr. Som si zoradil objekty podľa mien a teraz budem pracovať len s ID tých objektov. Namapujem Objekt k ID a ďalej v streame budú len ID

`.collect()`

Streams # Nie sú to dátové štruktúry

`.filter()` Berie podmienku a pustí ďalej len tie entity, ktoré ju spĺňajú

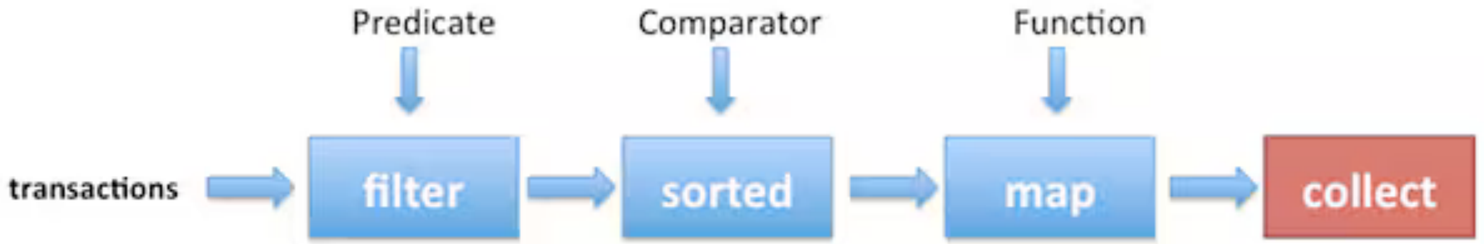
`.sorted(comparing( ))` Hodnota ktorú porovnávame. Napr. person.getName(); Zoradí podľa abecedy

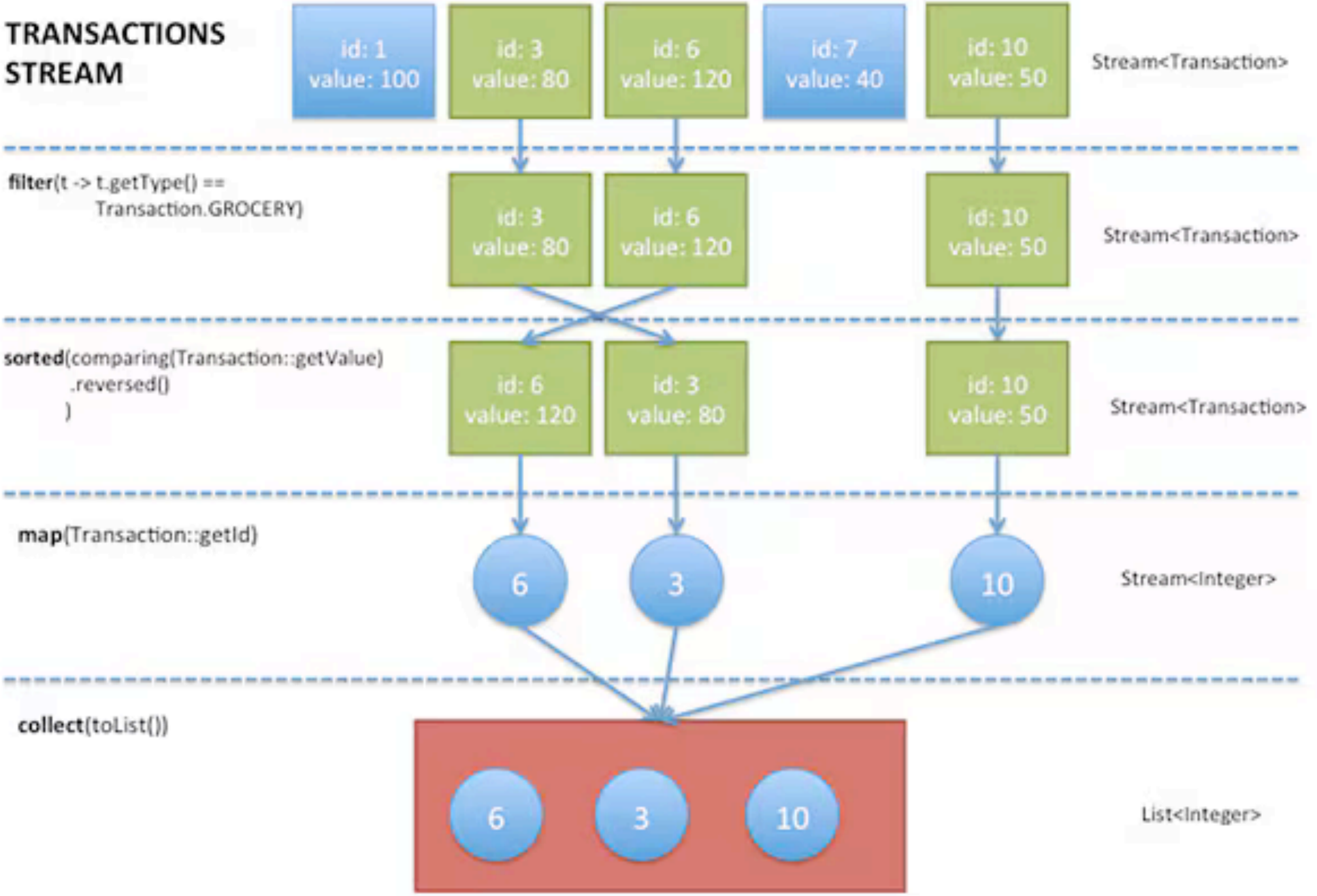
`.map()` Umožní nám trasformovať, s akými dátami ďalej budeme pracovať. Napr. Som si zoradil objekty podľa mien a teraz budem pracovať len s ID tých objektov. Namapujem Objekt k ID a ďalej v streame budú len ID

`.collect()` Spracuje nám dáta do finálnej podoby, ktorá je ako návratová hodnota

```
List<Integer> transactionsIds =
    transactions.parallelStream()
                .filter(t -> t.getType() == Transaction.GROCERY)
                .sorted(comparing(Transaction::getValue).reversed())
                .map(Transaction::getId)
                .collect(toList());
```

```
List<Integer> transactionsIds =
    transactions.parallelStream()
            .filter(t -> t.getType() == Transaction.GROCERY)
            .sorted(comparing(Transaction::getValue).reversed())
            .map(Transaction::getId)
            .collect(toList());
```




TRANSACTIONS STREAM
id: 1 value: 100
id: 3 value: 80
id: 6 value: 120
id: 7 value: 40
id: 10 value: 50
Stream<Transaction>
filter(t -> t.getType() == Transaction.GROCERY)
id: 3 value: 80
id: 6 value: 120
id: 10 value: 50
Stream<Transaction>
sorted(comparing(Transaction::getValue) .reversed())
id: 6 value: 120
id: 3 value: 80
id: 10 value: 50
Stream<Transaction>
map(Transaction::getId)
6
3
10
Stream<Integer>
collect(toList())
6
3
10
List<Integer>

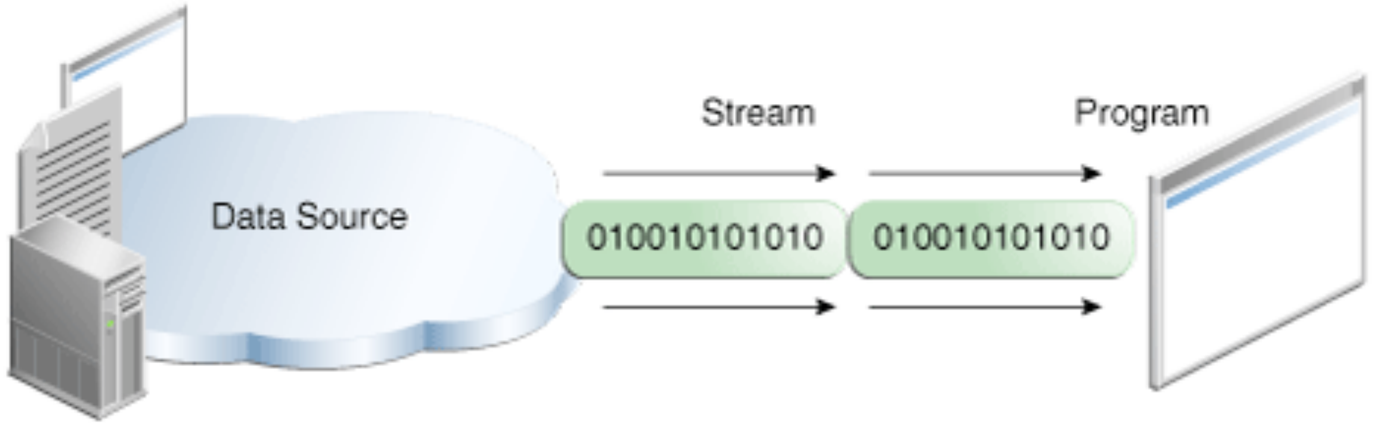
Dive deeper	https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html

Serializácia

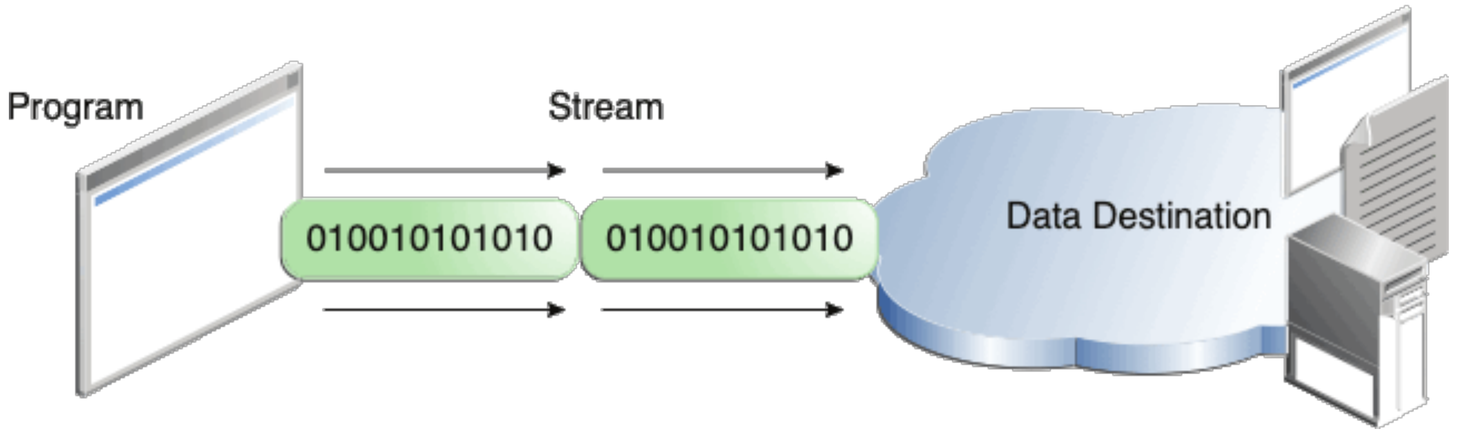
- Ukladá sa cely objektový graf, automaticky
- All or nothing
- Implements Serializable
- Pokiaľ nechcem/neviem serializovať - transient

IO stream

Stream != IO Streams



Input Stream



Output Stream

Dive deeper

https://docs.oracle.com/javase/tutorial/essential/io/streams.html

Quiz time!

OOP kvíz #6-7