**Content**

[1. Terminology 2](#_Toc4578705)

[1. Macros 2](#_Toc4578706)

[2. Java Streams 2](#_Toc4578707)

[3. Dynamic proxies 2](#_Toc4578708)

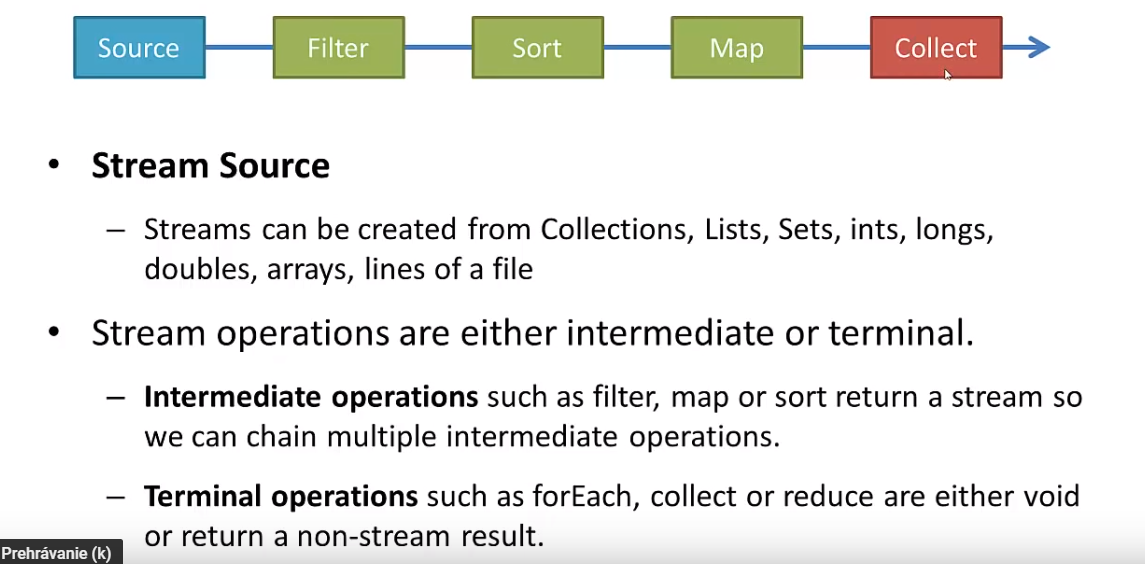
1. Terminology
   1. Macros
   2. Java Streams

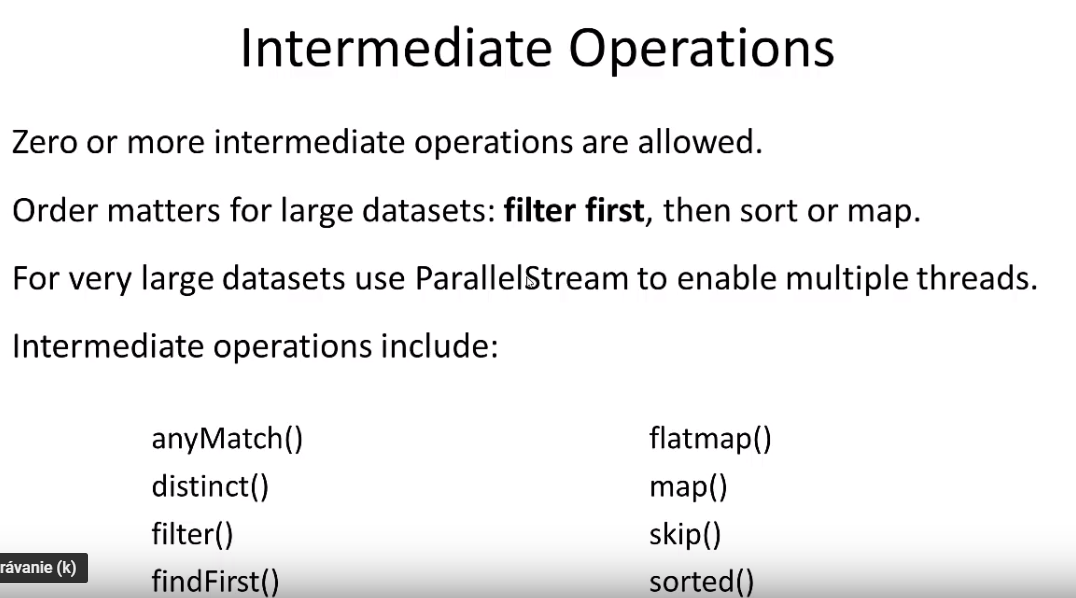
Streams bring functional programming to Java. Java 8+

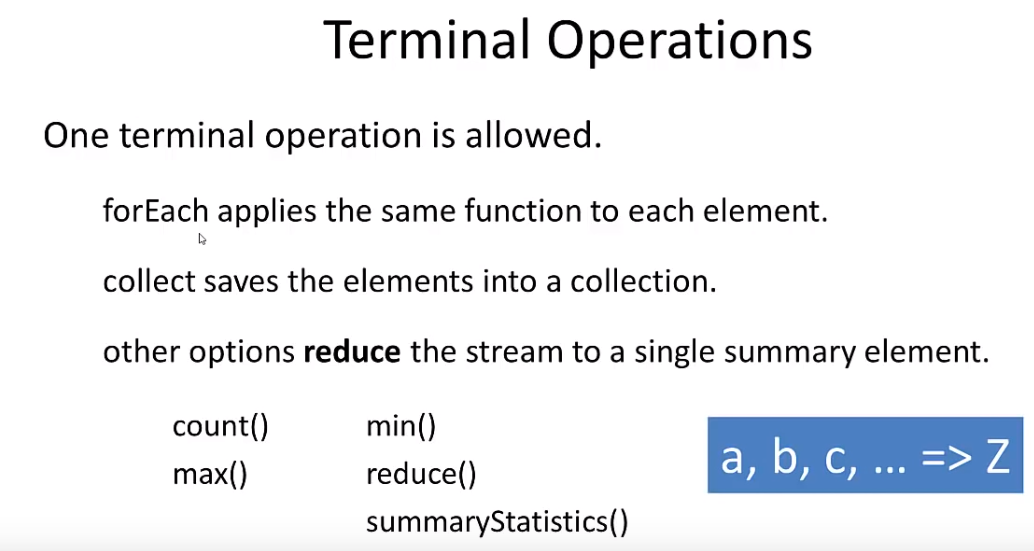
* **Effective**
* **Lambda expressions**
* **ParallelStreams** make it very easy to **multi-thread operations**

Consists of

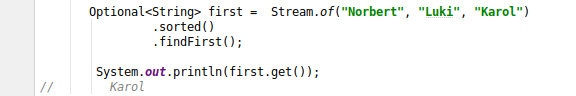
* Source
* 0 or intermadiate operations
* Terminal operation







* + 1. Functional programming in Java
    2. Method reference – „*System.out::println“*
  1. Dynamic proxies
  2. Whats new in Java EE
  3. Optional



1. AOP – Java
   1. Atomic

Simply put, shared state very easily leads to problems when concurrency is involved. If access to **shared mutable objects** is not managed properly, applications can quickly become prone to some hard-to-detect concurrency errors.

In this article, we’ll revisit the use of locks to handle concurrent access, explore some of the disadvantages associated with **locks** and finally, introduce **atomic** variables as an alternative.

There is a branch of research focused on creating non-blocking algorithms for **concurrent** environments: **compare-and-swap (CAS).** A typical CAS operation works on three operands:

1. The memory location on which to operate (M)
2. The existing expected value (A) of the variable
3. The new value (B) which needs to be set

**The CAS operation updates atomically the value in M to B, but only if the existing value in M matches A, otherwise no action is taken.**

* 1. Transaction

*Bundle multiple database changes into atomic operations, Undo!*

A transaction, in the context of a database, is a logical unit that is independently executed for data retrieval or updates. In relational databases, database transactions must be atomic, consistent, isolated and durable--summarized as the ACID acronym.

Transactions are completed by **COMMIT** or **ROLLBACK SQL statements**, which indicate a transaction’s beginning or end. The ACID acronym defines the properties of a database transaction, as follows:

1. **Atomicity**: A transaction must be fully complete, saved (committed) or completely undone (rolled back). A sale in a retail store database illustrates a scenario which explains atomicity, e.g., the sale consists of an inventory reduction and a record of incoming cash. Both either happen together or do not happen - it's all or nothing.
2. **Consistency**: The transaction must be fully compliant with the state of the database as it was prior to the transaction. In other words, the transaction cannot break the database’s constraints. For example, if a database table’s Phone Number column can only contain numerals, then consistency dictates that any transaction attempting to enter an alphabetical letter may not commit.
3. **Isolation**: Transaction data must not be available to other transactions until the original transaction is committed or rolled back.
4. **Durability**: Transaction data changes must be available, even in the event of database failure.

Operations that alter the state of the database – select is not such an operation!

Atomic Operations: Database operations that update the state are atomic. All or nothing. Insert either works completely or it doesnt. Update too. You never get half an operation.

Reasons why operations may not work:

One or more contraints violated. Datatype mismatch, Syntax error.

e.g. : money transaction: 2 updates - -50e, +50e – i want to bundle multiple operations into a single, atomic transaction. Wether both of the updates works or none of them.

