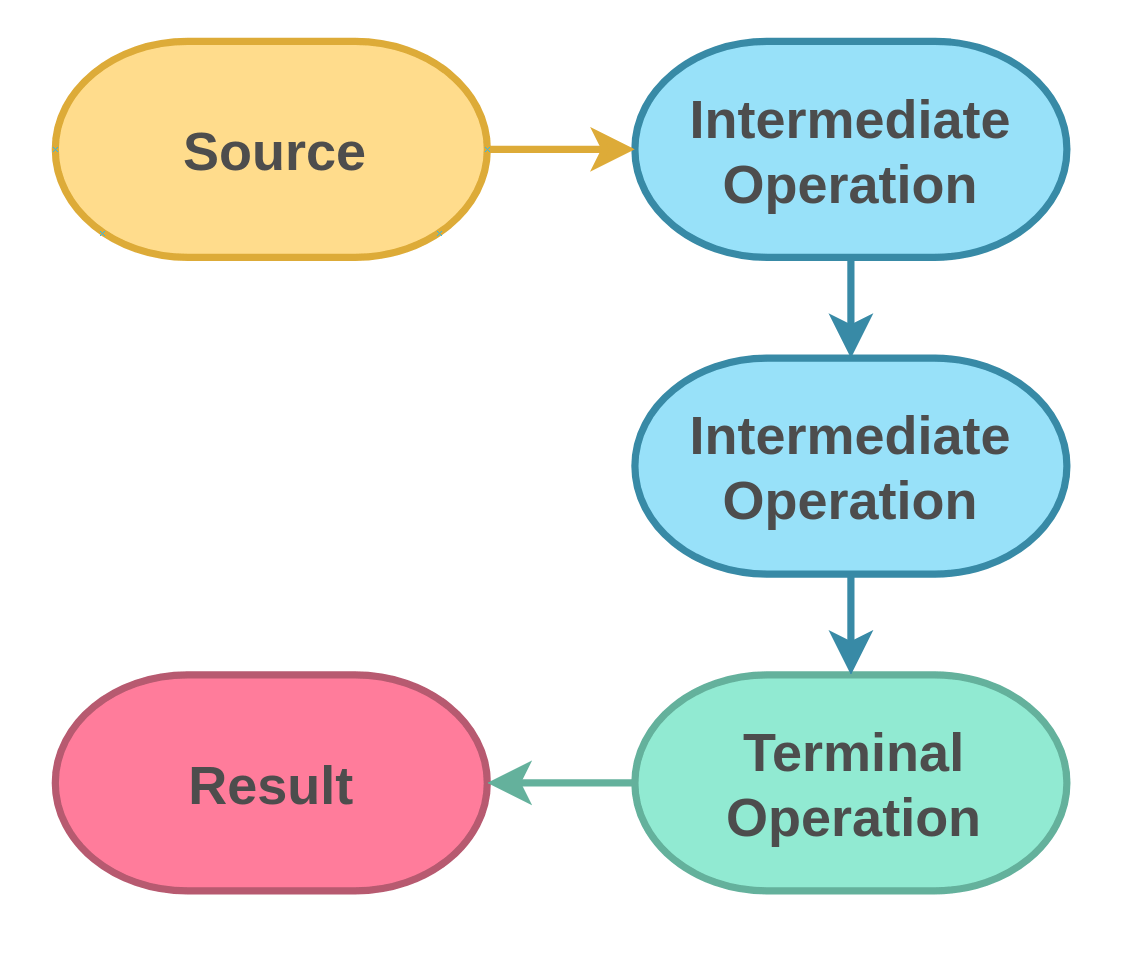
**Java Streams**

* Stream API is basically a lazy-sequential data pipeline of functional blocks. It isn’t implemented as a data structure or by changing its elements directly
* The basic concept behind stream is simple:
  + We got a data source (Source)
  + We perform zero or more intermediate operations on it (Intermediate operations)
  + We got a result after the operations (Terminal operation)



* Performing operations on the stream elements is great. But at some point, we want to get a result back from our data pipeline. Terminal operations are initiating the lazy pipeline to do the actual work and don’t return a new stream

* Laziness here means until we call a terminal operation on a stream, no work is done
* One of the main pillars of functional programming is an immutable state. Most intermediate operations are stateless, except for
  + distinct( ), sorted( ), limit( ), skip( )
* Stream characteristics are
  + Laziness
  + Stateless (mostly)
  + Optimization included (redundant operations removed, short-circuited)
  + Non-reusable
  + Less boilerplate code (easy to read and comprehend, emphasizes ‘what’)
  + Easy parallelization (but should be done with caution)
  + Primitive handling

* Lambdas can be simple one-liners or huge code blocks if wrapped in curly braces. To retain the simplicity and conciseness, we should restrict ourselves to these two use cases for operations:
  + One-line expressions e.g. .filter(album -> album.getYear > 4)
  + Method references e.g. .filter(this::myFilterCriteria)

The bytecode between a lambda and a method reference differs slightly – with the method reference generating less

* By putting each pipeline step into a new line, we can improve readability
* If an intermediate operation throws a checked exception, we should refactor it to a method and handle its exception accordingly
* For stream debugging, use the peek( ) method
* Take extra care for the ordering of operations. That would really improve performance

**Functional programming**

* Functional programming is a programming paradigm evolved from the invention of Lambda calculus in 1936

* A pure function has two elemental properties:
  + The same input generates the same output
  + No side effects e.g. affecting global state or changing argument values
* Due to the predictable result for non-changing input arguments, we could replace a once-run function with its return value. This is called Referential Transparency