Java 8

Interface

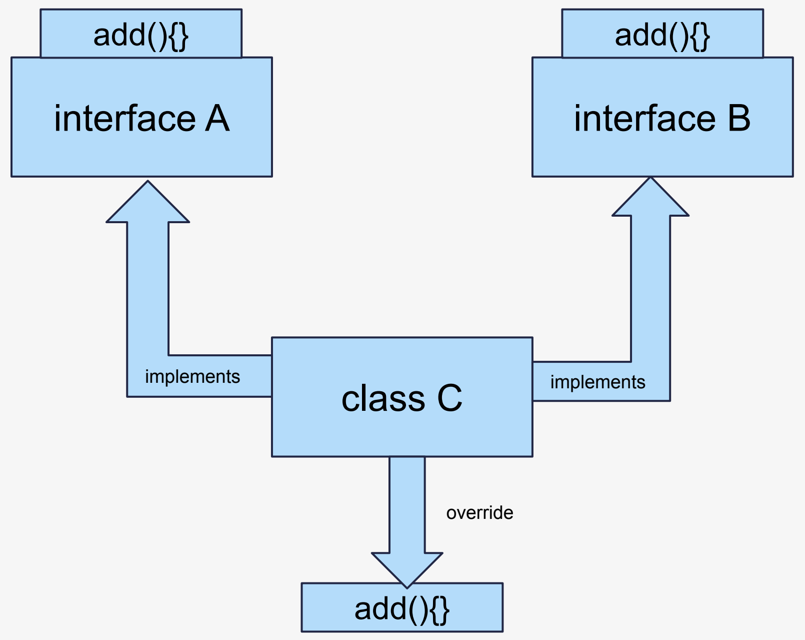
Before Java 8

- We can only declare a method in Interface
- By default it is “public abstract”

From Java 8

- We can only declare and define a method in Interface
- By this they achieved backward compatibility
- Interface can now take static functions

Diamond Problem in Java 8 Interface



- Two interface having same function
- Class extend both the interface
- Ambiguity arises here
- To overcome this we need to override the methods in the implemented class

default method in interface

- To declare a method inside the interface default keyword is used

```
interface A {
        default void add(){
                system.out.println(“added”)
        }
}
```

static method in interface

- From Java 8 we can also declare static method inside the interface call them directly.

```
interface A {
        static void add(){
                system.out.println(“added”)
        }
}
```

For loop

Traditional For loop

```
List<Integer> lst = Arrays.asList(1,2,3,4,5,6);
for(int i = 0; i < lst.size(); i++){
        System.out.println(lst.get(i));
}
```

Enhanced For loop

```
List<Integer> lst = Arrays.asList(1,2,3,4,5,6);
for(Integer i : lst){
        System.out.println(i);
}
```

Foreach loop

- It is default method in iterable interface
- It pass each element and perform actions on each element

```
List<Integer> lst = Arrays.asList(1,2,3,4,5,6);
lst.forEach(i -> System.out.println(i));
```

Lambda

Code in oops

- ➢ Everything is an object
- ➢ All code blocks are associated with classes and objects

Functions as values

- ➢ Inline values :
  - ○ String name = “foo”;
  - ○ double pi = 3.14;
- ➢ aBlockOfCode = {

  …

  }

Why Lambda

- ➢ Enables Functional programming
- ➢ Parallel programming

Syntax

```
parameter -> expression body
```

Important characteristics of a lambda expression

- ➢ **Optional type declaration** − No need to declare the type of a parameter. The compiler can inference the same from the value of the parameter.
- ➢ **Optional parentheses around parameter** − No need to declare a single parameter in parenthesis. For multiple parameters, parentheses are required.
- ➢ **Optional curly braces** − No need to use curly braces in expression body if the body contains a single statement.
- ➢ **Optional return keyword** − The compiler automatically returns the value if the body has a single expression to return the value. Curly braces are required to indicate that expression returns a value.

Example:

No Parameter

```
() -> System.out.println("hello");
```

With parameter

```
(a,b) -> a*b
```

Multiline

```
(a,b) -> {
        if(b==0) return 0;
        return a/b;
};
```

Inbuilt Interfaces

- Java 8 has some inbuilt interfaces to address some of these common scenarios
- Package : java.util.functions
- Some of the commonly user interfaces are
    - Predicate
        - Takes input argument and return boolean value
    - Consumer
        - Takes input argument and return nothing
    - Supplier
        - Takes nothing and return a object

Method Reference

- ➢ Method reference is used to refer method of functional interface.
- ➢ It is compact and easy form of lambda expression.
- ➢ Each time when you are using lambda expression to just referring a method, you can replace your lambda expression with method reference

Types of Method References

- ➢ Reference to a static method.

```
ContainingClass::staticMethodName
```

- ➢ Reference to an instance method.

```
containingObject::instanceMethodName
```

- ➢ Reference to a constructor.

```
ClassName::new
```

Optional

Optional is a public final class and used to deal with NullPointerException in Java application.

Methods

- ➢ isPresent()
- ➢ empty()
- ➢ orElse()
- ➢ isEmpty()
- ➢ ifPresent()

Streams

Streams

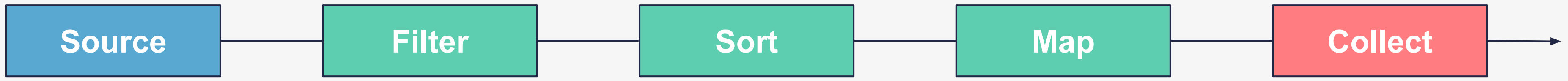
A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result

Features of Java stream

- ➢ A stream is not a data structure instead it takes input from the Collections, Arrays, or I/O channels.
- ➢ Streams don't change the original data structure, they only provide the result as per the pipelined methods

Streams Pipeline

A Stream pipeline consists of a source, followed by intermediate operations and a terminal operation.



Stream Source

- ➢ Stream can be created from Collections and Arrays.

Intermediate operations

- ➢ Intermediate operations such has filter, map or sort return a stream, so we can chain multiple operations.

Terminal operations

- ➢ Terminal operations such as forEach, collect or reduce either return void or returns a non stream result.

Intermediate Operations

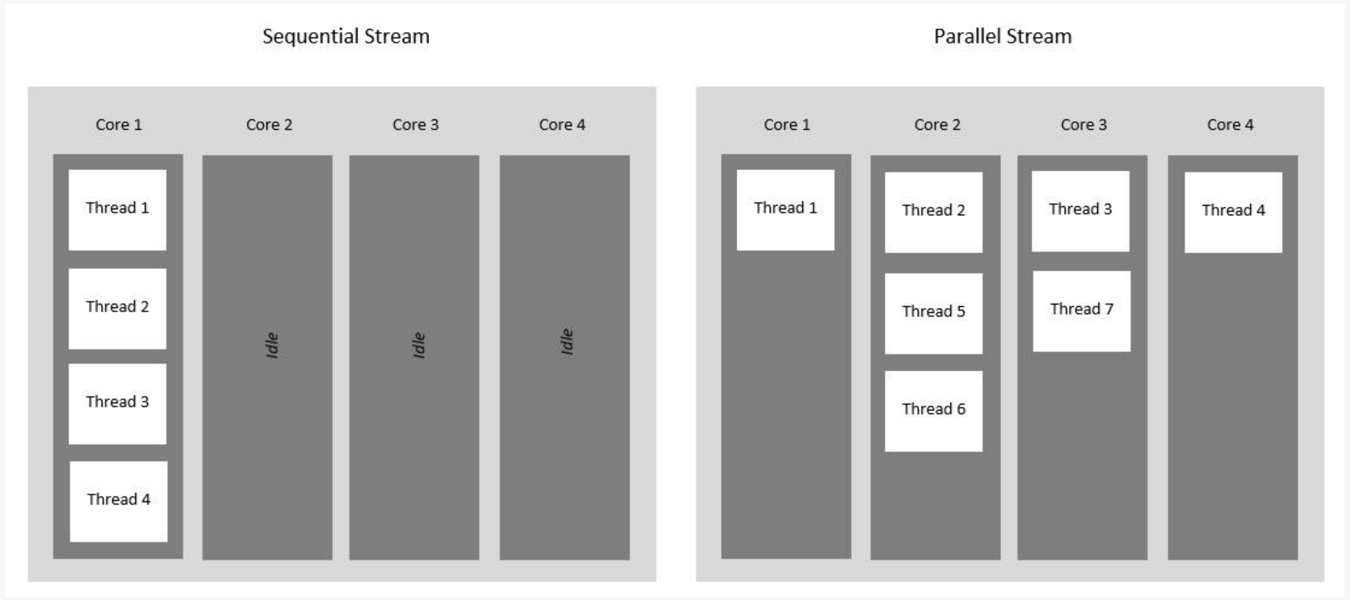
- Zero or more intermediate operations allowed
- Order Matters for large dataset, filter first then sort or map
- Intermediate operation include:
    - filter()
    - map()
    - sorted()
    - anyMatch()
    - distinct()
    - findFirst()
    - skip()
    - flatMap()
    - mapToInt()
    - limit()

Terminal Operations

- One operation is allowed
- forEach applies function to each element
- Collect saves the element into collection
- Other options reduces the stream to a single element
    - count()
    - reduce()
    - sum()
    - average()

Parallel streams

- ➢ It is meant for utilizing multiple cores of the processor
- ➢ The elements in the stream processed parallely, the order of execution, how ever is not under our control



Ways to implement Streams

- ➢ Using parallel() method on a stream
- ➢ Using parallelStream() on a Collection

Problems

First Problem

Given a list of integers, find out all the numbers starting with 1 using Stream functions.
Input = [10,15,8,49,25,98,32]
Output = [10,15]

Second Problem

You are given an array prices where prices[i] is the price of a given stock on the ith day.

You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.