Programare avansata pe obiecte – laborator 9

Alina Puscasu *alina.puscasu@endava.com*

Lambdas

(lista parametrilor) -> {expresie sau instructiuni}

- Concept din programarea functionala care reprezinta o functie anonima
- Sunt o alternativa mai eleganta decat utilizarea unor clase anonime (mai putin cod, mai usor de urmarit, mai scalabil)

```
interface Engine {
    int getFuelCapacity();
}

public class Car {
    public Engine getEngine(int fuelCapacity) {

        // apel anonim
        return new Engine() {
            @Override
            public int getFuelCapacity() {
                return fuelCapacity;
            }
        };
    }


    public static void main(String[] args) {
        Car car = new Car();
        System.out.println(car.getEngine( fuelCapacity: 11).getFuelCapacity());
    }
}
```

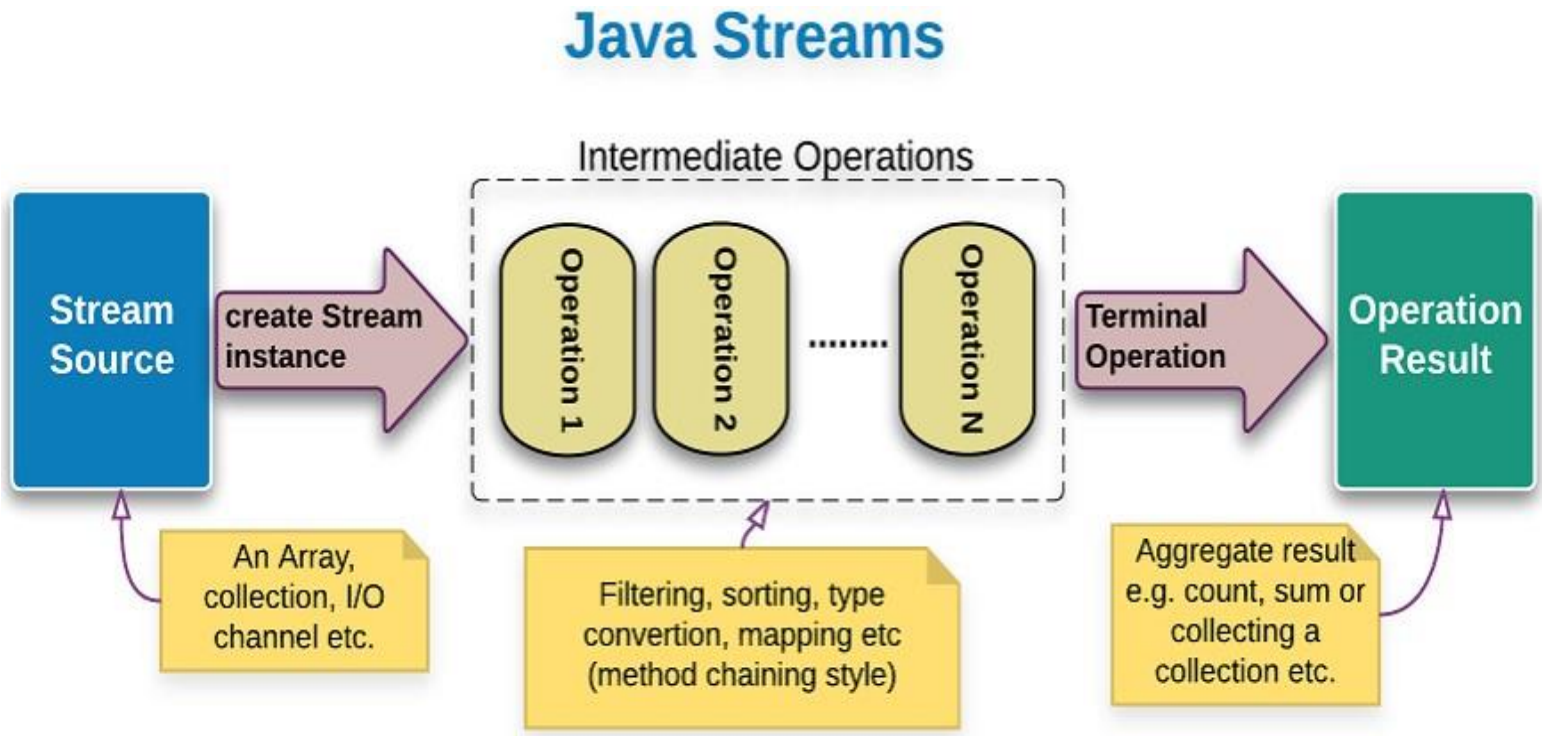
```
interface Engine {
    int getFuelCapacity();
}

public class Car {
    public Engine getEngine(int fuelCapacity) {
        // expresie lamda
        return () -> fuelCapacity;
    }


    public static void main(String[] args) {
        Car car = new Car();
        System.out.println(car.getEngine( fuelCapacity: 11).getFuelCapacity());
    }
}
```

```
public static void main(String[] args) {
    Student[] students = {new Student( name: "Maria",  age: 20)};

    Arrays.sort(students, new Comparator<Student>() {
        @Override
        public int compare(Student o1, Student o2) {
            return o1.getAge() - o2.getAge();
        }
    });

    Arrays.sort(students, (o1, o2) -> o1.getAge() - o2.getAge());
}
}
```

- Alta situatie des intalnita de folosire a lor este pentru transmiterea de functii ca parametru
  - Pentru asta au fost introduse interfetele functionale (interfata functionala = interfata cu o singura metoda abstracta, adnotata cu @FunctionalInterface)
    - Le gasim in pachetul java.util.function (vezi exemple cod laborator din pachet interfete.functionale)

Streams

- Flux de date care suporta operatii de procesare
- Realizeaza operatii pe colectii intr-un mod elegant si eficient
- Majoritatea operatiilor pe un stream vor genera alt stream, ceea ce permite crearea unui lant de prelucrari. Metodele utilizate pentru prelucrare sunt in java.util.stream.



- Putem crea stream-uri prin mai multe modalitati:

```
public class CreationOfStreams {
    public static void main(String[] args) {
        String[] arr = new String[]{"a", "b", "c"};
        Stream<String> stringStream = Arrays.stream(arr);

        List<Student> studentList = new ArrayList<>();
        Stream<Student> studentStream = studentList.stream();

        Stream<String> secondStringStream = Stream.of("a", "b");

        Stream<Car> carStream = Stream.empty();

        Stream<Integer> integerStream = Stream.iterate( seed: 0, x -> x + 1); // 0 1 2 3...

        Stream<Integer> secondIntegerStream = Stream.generate(() -> 1); // 1 1 1...
    }
}
```

- Operatiile intermediare efectuate asupra unui stream sunt efectuate abia in momentul in care este invocata o operatie de inchidere

- Exemple de operatii intermediare:
    - Stream<T> filter (Predicate<? super T> predicate)
    - Stream<T> sorted (Comparator<? super T> comparator)
    - Stream<T> limit (long maxSize)
    - Stream<T> distinct()
    - Stream<R> map(Function<T, R> mapper)

- Exemple de operatii de inchidere:
    - void forEach(Consumer<T> action)
    - Optional<T> max(Comparator<T> comparator)
    - Optional<T> min(Comparator<T> comparator)
    - R collect(Collector<T,A,R> collector)
        - ✦ Clasa Collector contine o serie de metode statice: toList(), toSet(), toMap(), joining(String delimitator), counting(), averagingDouble/Int/Long(), summingDouble/Int/Long()/groupingBy()
- Exemple de folosire in pachetul streams din laborator