diginamic
FORMATION

« Votre passeport pour l'emploi numérique »

Formation Java 11

Expression Lambda

Sommaire

- Approche Impérative vs Approche Fonctionnelle
- Expression Lambda
- Méthode Référence

Un exemple

- Soit une liste de comptes courants.
- Notre objectif est de calculer la moyenne des soldes.

```
public class CompteCourant {
    String numero;
    String intitule;
    double solde;
    double montDecouvertAutorise;
}

List<CompteCourant> comptes = new ArrayList<>();
```

Avec une approche impérative

```
double somme = 0.0;
double moyenne = 0.0;

for(CompteCourant c : list) {
    somme += c.getSolde();
}

if(!list.isEmpty()) {
    moyenne = somme / list.size();
}
```

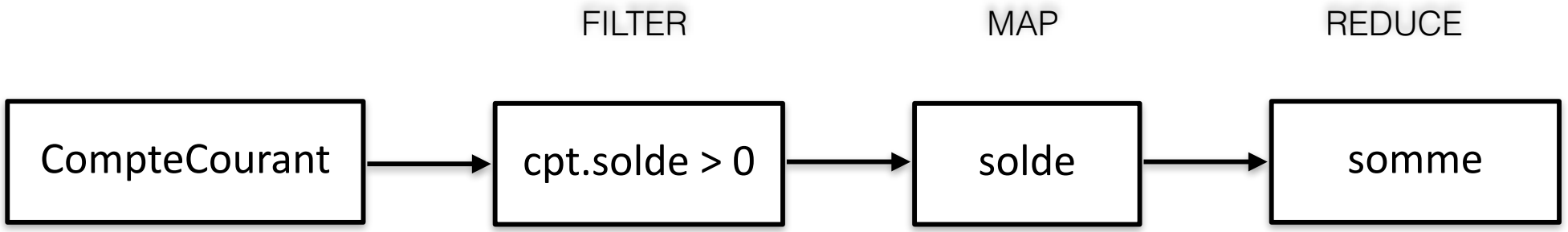
Avec uniquement les soldes > 0

```
double somme = 0.0;
double moyenne = 0.0;
int nbComptes = 0;

for(CompteCourant c : list) {
    if(c.getSolde() > 0.0) {
        somme += c.getSolde();
        nbComptes++;
    }
}
if(!list.isEmpty()) {
    moyenne = somme / nbComptes;
}
```

Approche fonctionnelle (SQL)

```sql
SELECT AVG (solde)
FROM COMPTE_COURANT
WHERE SOLDE > 0
```

Approche fonctionnelle



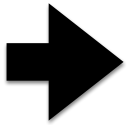
Approche fonctionnelle (JDK 7)

```java
public interface Mapper<T, V> {
    public V map(T t);
}

public interface Predicate<T> {
    public boolean filter(T t);
}

public interface Reducer<T> {
    public T reduce(T t1, T t2);
}
```

```java
list.map(new Mapper<CompteCourant,Double>() {
    @Override
    public Double map(CompteCourant t) {
        return t.getSolde();
    }
}).filter(new Predicate<Double>() {
    @Override
    public boolean filter(Double t) {
        return t > 0;
    }
}).reduce(new Reducer<Double>() {
    @Override
    public Double reduce(Double t1, Double t2) {
        return t1+t2;
    }
});
```

Approche fonctionnelle (JDK 8)

```
new Mapper<CompteCourant,Double>() {
    @Override
    public Double map(CompteCourant t) {
        return t.getSolde();
    }
}
```



```
(CompteCourant t) -> t.getSolde()

ou

t -> t.getSolde()
```

Approche fonctionnelle (JDK 8)

```
new Predicate<Double>() {
    @Override
    public boolean filter(Double t) {
        return t > 0;
    }
}
```

```
(Double t) -> t > 0

ou

t -> t > 0
```

Approche fonctionnelle (JDK 8)

```
new Reducer<Double>() {
    @Override
    public Double reduce(Double t1, Double t2) {
        return t1+t2;
    }
}
```

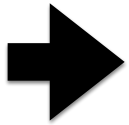
```
(Double t1, Double t2) -> t1 + t2
```

ou

```
(t1,t2) -> t1 + t2
```

Approche fonctionnelle (JDK 8)

```
list.map(new Mapper<CompteCourant,Double>() {
    @Override
    public Double map(CompteCourant t) {
        return t.getSolde();
    }
}).filter(new Predicate<Double>() {
    @Override
    public boolean filter(Double t) {
        return t > 0;
    }
}).reduce(new Reducer<Double>() {
    @Override
    public Double reduce(Double t1, Double t2) {
        return t1+t2;
    }
});
```



```
list.map(t -> t.getSolde())
    .filter(t -> t > 0)
    .reduce((t1, t2) -> t1+t2)
```

Si la lambda nécessite plusieurs lignes de code ?

```
list()
    .map(t -> {
        System.out.println(t);
        return t.getSolde();
    })
    .filter(t -> t > 0)
    .reduce((t1, t2) -> t1+t2);
```

Si aucun paramètre

```
log.err("message", () -> "erreur")
```

Quelles conditions pour utiliser une lambda ?

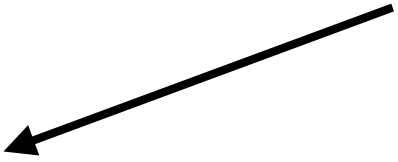
- Une seule méthode abstraite dans l'interface. On parle alors d'**interface fonctionnelle**.
- Les types de paramètres de l'unique méthode doivent être compatible avec les types de l'expression lambda.

Une lambda dans une variable ?

```
Mapper<CompteCourant, Double> mapper = t -> t.getSolde();
Predicate<Double> filter = t -> t > 0;
Reducer<Double> reduce = (t1, t2) -> t1+t2;

list.stream()
        .map(mapper)
        .filter(filter)
        .reduce(reduce);
```

Référence de méthode

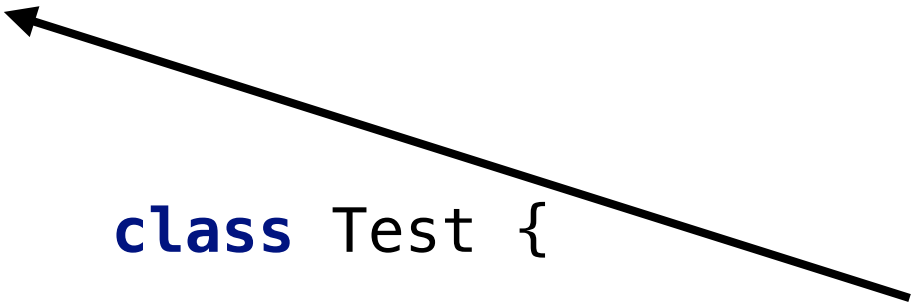
```
class Run {

    Runnable runField = this::run;

    public void run() {
        // fait quelque chose
    }
}
```



Référence de méthode

```
class Compte {
    double getSolde() {
        // un traitement
    }
}

interface A {
    double get(Compte c);
}
```

```
class Test {

    A a = Compte::getSolde;
}
```



Travaux Pratiques