Implémentation de classe anonyme :

```
// HelloAnonymous.java
package com.jdojo.innerclasses;
public class HelloAnonymous {
public static void main(String[] args) {
new Object() {
// An instance initializer
{
System.out.println ("Hello from an anonymous class.");
}
}; // A semicolon is necessary to end the statement
}
}
```

Programmation fonctionnelle
= passer un code d'une fonction

C : int i;
(int ()i* ;)

class C
{
void m()
{}
}
m

m();
adresse du code — opérateur de branchement

classes anonymes → syntaxe plutôt lourde
Expressions lambda
↓
simplification de la syntaxe

Annotations.

(1) La déclarer ("interface")

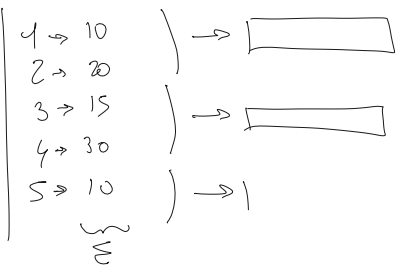
(2) L'utiliser (@<Annotation>)
↳ ne change rien

(3) Dynamiquement : si il y a une annotation, alors .....

Exercice :

- Créer l'annotation Version
- L'utiliser
- Obtenir l'instance de l'Annotation au runtime.

Paradigme Map Reduce



Evolution Classes vers LAmbdas :

```
package com.stage.annotations;


import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.*;
import java.util.function.*;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Main {

   public static int mygen()
   {
      return 0;
   }
   public static void process( Integer integer )
   {

      System.out.println("4) "+integer);
   }

   public static void main( String[] args )
   {
      List<Integer> numbers = List.of(1, 2, 4, 3, 5);

      // 1) :
      numbers.stream().peek( new Displayer());

      // 2) :
      numbers.stream().peek( new Consumer<Integer>() {
         @Override
      public void accept(Integer integer) {
         System.out.println("2) "+integer);
      }}
      ).toArray();

      // 3) :
      numbers.stream().peek( (integer) -> { System.out.println("3) "+integer); }).toArray();

      // 4) :
      numbers.stream().peek( (integer) -> Main.process(integer) ).toArray();
      // 4') :
      numbers.stream().peek( (integer) -> System.out.println("4') "+integer) ).toArray();
      numbers.stream()
            .map( i -> "&\" "+i)
            .peek( System.out::println )
            .toArray();

   }
}
```

```
package com.stage.annotations;

import java.util.function.Consumer;

public class Displayer implements Consumer<Integer>
{
   @Override
   public void accept(Integer integer) {
      System.out.println("1) "+integer);
   }
}
```