

Principales nouveauautés des versions 8 à 21 du langage JAVA

Table des matières

I - Evolution du langage java (introduction).....	4
1. Historique et évolution du langage java.....	4
II - Lambda expressions (prog. Fonctionnelle).....	6
1. Java multi-paradigme "objet + fonctionnel"	6

2. Lambda expressions et prog. fonctionnelle.....	8
III - Streams (prog. fonctionnelle).....	16
1. Streams (java 8 et plus).....	16
IV - Optional , DateTime ,20	20
1. Optional , DateTime ,20	20
V - Modules (à partir de java 9).....25	25
1. Modules java 9+ (essentiel).....25	25
VI - ForkJoin, Concurrent, CompletableFuture,42	42
1. Concurrent api.....42	42
2. ForkJoin.....53	53
3. CompletableFuture , streams asynchrones.....62	62
VII - Reactive-Streams (depuis java 9).....70	70
1. Reactive Streams (depuis java 9).....70	70
VIII - JShell (RPEL java).....78	78
1. JShell (RPEL java).....78	78
IX - Process api , Http2 api ,83	83
1. Process Api.....83	83
2. Http2 Client.....87	87
3. Diverses nouveautés de java 9+.....92	92
X - var et ajouts/restrictions de java 10 et 11.....101	101
1. var (inférence de type pour variables locales).....101	101
2. Ajouts divers de java 10 et 11.....101	101
3. Restrictions/Restructurations de java 11.....104	104
XI - Switch expressions et ajouts java 12,...,17.....106	106
1. Switch expressions.....106	106
2. Pattern Matching instanceof.....108	108
3. TextBloc.....109	109
4. Record (classe de données simplifiée pour DTO).....111	111
5. Sealed classes.....117	117

6. Pattern matching sur switch/case of object.....	119
7. utilitaire jpackage.....	120
8. Autres apports des versions récentes (12,...,17).....	121
9. Nouveau type de licence depuis java 17.....	121

XII - Virtual Threads et ajouts java 18,...,21.....122

1. Pattern and record matching.....	122
2. Virtual Thread.....	124
3. Structured Concurrency (preview java 21).....	128
4. Foreign Function and Memory (preview java 21).....	131
5. Vector api (incubator java 21).....	132

XIII - Annexe – new io.....135

1. NIO2 (new IO).....	135
2. Principales classes et interfaces de nio2.....	135
3. Gestion des chemins (Path).....	137
4. Classe utilitaire Files (" <i>helper</i> " avec 50 méthodes statiques).....	141
5. Parcours des éléments d'un répertoire.....	143
6. Parcours d'une hiérarchie de répertoires (visiteur).....	145
7. FileSystem (par défaut et "personnalisé").....	145
8. Lecture et écriture dans un fichier.....	146

XIV - Annexe – Lombok.....148

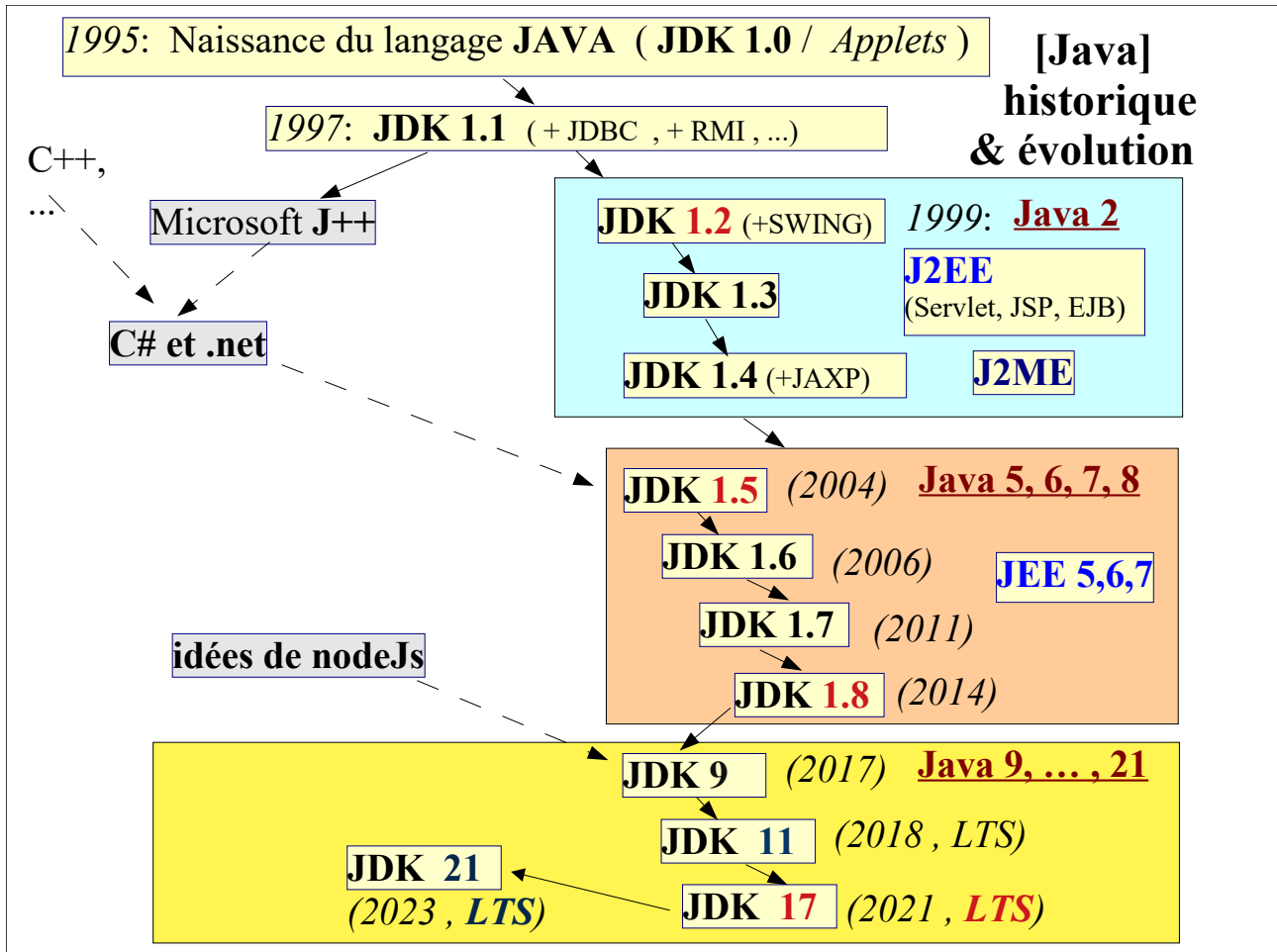
1. Lombok.....	148
----------------	-----

XV - Annexe – Bibliographie, Liens WEB + TP.....150

1. énoncés de TP.....	150
-----------------------	-----

I - Evolution du langage java (introduction)

1. Historique et évolution du langage java



JDK signifie *Java Development Kit*.

Les JDK 1.2 et 1.5 ont apportées de grandes nouveautés qui ont modifié le langage en profondeur.

Le terme plate-forme **Java 2** désigne toutes les versions de *Java* à partir du *JDK 1.2* et englobe également une extension pour les serveurs d'applications : *J2EE* (*Java 2 Enterprise Edition*).

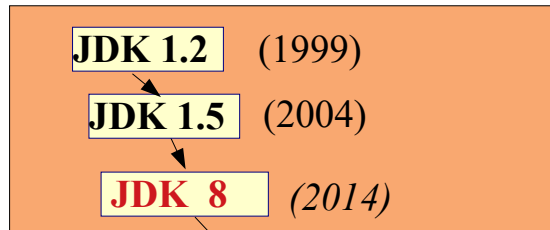
Bien que différents et incompatibles les langages Java et C# comportent beaucoup de points communs (syntaxe et architecture assez proches).

La société **SUN MicroSystem** qui a inventé le langage **JAVA** est le propriétaire officiel du langage et décide de son évolution (en tenant compte des avis de ses partenaires).

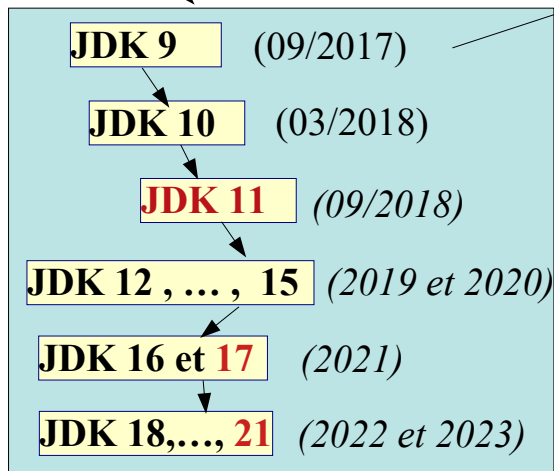
L'entreprise "SUN" a été rachetée par "Oracle".

Oracle est maintenant le **nouveau propriétaire** de "**Java**".

La version **8** du **jdk** a apporté quelques grandes nouveautés syntaxiques (**lambda expressions**, **streams**, ...) et propose **java-fx** à la place de **swing**.



Evolution récente du langage java



Depuis V9 (2017) , **une nouvelle version tous les 6 mois** avec quelquefois très peu de nouveautés !

NB: Les versions **11, 17** et **21** (stables) sont **LTS** (Long Time Support) et donc conseillées .

Attention : le mode "**--enable-preview**" est **éphémère** (remis en cause tous les 6 mois et exploitable qu'avec une version bien trop précise du JDK et/ou d'un IDE (eclipse ou autre)).

NB : à partir du jdk 11 , java-fx est devenu une extension facultative (désormais plus incluse dans le jdk) .

NB : a coté du jdk officiel (de l'entreprise Oracle) , il existe également le "**openJdk**" entièrement "open source" (un peu moins complet que le jdk d'Oracle mais sans partie potentiellement payante en production) .

NB : en v11 , le openJdk était gratuit et le "Oracle jdk" payant pour une utilisation commerciale.

À partir de la version 17 , les deux versions "openJdk" et "Oracle JDK" sont gratuites aussi bien en mode développement que pour des utilisations commerciales (licence « Oracle No-Fee Terms and Conditions » (NFTC).)

II - Lambda expressions (prog. Fonctionnelle)

1. Java multi-paradigme "objet + fonctionnel"

1.1. Evolution importante dès java 8

Les premières versions de java (1.0 à 1.7) étaient centrées sur le paradigme orienté objet .

A partir de la version 8 , le langage java s'est fortement inspiré du langage scala et est devenu un langage multi-paradigme "objet" et "fonctionnel" .

Sans renier ses fondements orientés objets, le langage java a introduit à partir de la version 8 un **nouveau style complémentaire de programmation** appelée **"programmation fonctionnelle"** .

1.2. Programmation fonctionnelle

Au sein d'un contexte de programmation purement orienté objet , les objets comportent des données dont les valeurs (à l'instant t) représentent un certain état .

En appelant une méthode sur un objet , il y a souvent un changement de valeur(s) effectué et donc un changement d'état .

Exemple :

```
p1 = new Personne("jean", "Bon" , 40) ;// où 40 signifie ici 40 ans
p1.incrementerAge() ; //premier changement d'état (40 ans → 41 ans)
p1.incrementerAge() ;//second changement d'état (41 ans → 42 ans)
System.out.println("nouvel age de p1=" + p1.getAge()) ; //affiche 42
```

A l'inverse en appliquant le style de programmation fonctionnelle , on manipule avant tout des fonctions pures (sans contexte , sans état contextuel) qui permettent d'appliquer des transformations (ou des calculs) .

Exemple :

```
Personne p1 = new Personne("jean","Bon",40);
System.out.println("p1="+p1);
Function<Personne,Personne> avecUnAnDePlus ;
avecUnAnDePlus = (Personne p) -> new Personne(p.getPrenom(),
                                     p.getNom(),
                                     p.getAge() + 1);
Personne p1Bis = avecUnAnDePlus.apply(avecUnAnDePlus.apply(p1));
//NB: l'objet (de données) p1 n'est pas modifié.
//la double application de la fonction de transformation avecUnAnDePlus
//a permis de créer un nouvel objet p1Bis ayant 2 ans de plus que p1.
System.out.println("p1Bis="+p1Bis);//42 ans
```

NB : La syntaxe java suivante

```
avecUnAnDePlus = (Personne p) -> new Personne(p.getPrenom(),  
                                                p.getNom(),  
                                                p.getAge() + 1);
```

est nouvelle depuis java 8 et correspond à une *lambda expression* .

Ça correspond ici à une fonction admettant en entrée un objet de type *Personne* et retournant en retour un autre objet *Personne* .

1.3. Programmation fonctionnelle accrochée à un squelette objet

Etant donné que java est un langage avant tout orienté objet , les nouveaux éléments ayant un style "fonctionnel" doivent s' accrocher à un squelette de code un peu orienté objet pour pouvoir exister , être compilé et être exécuté par une machine virtuelle java.

2. Lambda expressions et prog. fonctionnelle

2.1. Interfaces fonctionnelles , "lambda expression" et références

Une **interface fonctionnelle** (que l'on peut facultativement explicitement marquer avec la nouvelle annotation `@FunctionalInterface` du `jdk >= 1.8`) est **une interface qui ne comporte qu'une seule méthode ordinaire** (sans "static" ni "default").

Son rôle est de permettre une gestion simple d'une certaine méthode/fonction abstraite .

(*SAM*= Single Abstract Method → interface de type "SAM")

Exemple (V1) :

```
package tp.langage.v8.sam;
/* L'annotation @FunctionalInterface (du jdk >= 1.8) est facultative
   elle permet au compilateur de vérifier que l'interface comporte une seule méthode (ordinaire) */
@FunctionalInterface
public interface SamPredicate<E> {
    boolean test(E e); //retourne true si l'entité e (de type E) satisfait certains critères
}
```

Exemple (V2) :

```
package tp.langage.v8.sam;

@FunctionalInterface
public interface SamPredicate<E> {
    boolean test(E e); //retourne true si l'entité e (de type E) satisfait certains critères
    default String getAuthor() { return "developpeur fou"; } //méthode par défaut
    // (alias "extension method" alias "defender method" ). Les méthodes par défaut ont permis
    // d'ajouter de nouvelles fonctionnalités à java8 sans remettre en question les interfaces java7
    static void methodeStatiqueAutoriseeSurInterfaceDepuisJava8(String msg){
        System.out.println(msg);
    }
    //les méthodes statiques au sein des interfaces permettent d'éviter la programmation de
    // nombreuses mini classes utilitaires périphériques
}
```

NB : `java.util.function.Predicate` (depuis java 8) correspond à un **équivalent prédéfini** de la version 1 de l'interface fonctionnelle `SamPredicate` .

Classe "Person" (pour la compréhension de l'exemple) :

```
...
public class Person {
    String firstname; //+get/set
    String lastname; //+get/set
    int age; //+get/set

    public Person(String firstname, String lastname, int age) { ...}
    public Person() {}
    public String toString() {return "Person (" +  firstname + "," + lastname + ", age=" + age + ")"; }

    public static int compareP1P2ByAge(Person p1, Person p2) {
        if (p1.getAge() > p2.getAge()) {
            return 1;
        } else if (p1.getAge() < p2.getAge()) {
            return -1;
        } else {
            return 0;
        }
    }

    public int compareThisP2ByAge(Person p2) {
        if (this.getAge() > p2.getAge()) {
            return 1;
        } else if (this.getAge() < p2.getAge()) {
            return -1;
        } else {
            return 0;
        }
    }
}
}
```

Syntaxe lourde (java 7) avec classes anonymes imbriquées :

```
...
public class FilterSortListJava7TestApp {
    ...
    public static List<Person> extractSubListBySamPredicate(List<Person> pList,
                                                            final SamPredicate<Person> samPredicate) {
        final List<Person> sublist = new ArrayList<Person>();
        for (Person p : pList) {
            if (samPredicate.test(p)) {
                sublist.add(p);
            }
        }
        return sublist;
    }
    ...
    public static void mainWithInnerAnonymousSamPredicateImplementations() {
        List<Person> pList = StaticPersonList.initList();

        System.out.println("person sublist with age from 20 to 25:");
        List<Person> subList1 = extractSubListBySamPredicate(pList,
            new SamPredicate<Person>() {
                @Override
                public boolean test(Person p) {
                    return p.getAge() >= 20 && p.getAge() <= 25;
                }
            }
        );
    }
}
```

```

    });
    System.out.println(subList1);

    System.out.println("person sublist with lastName starting with letter p");
    List<Person> subList2 = extractSubListBySamPredicate(pList,
        new SamPredicate<Person>(){
            @Override
            public boolean test(Person p) {
                return p.getLastName().startsWith("p");
            }
        });
    System.out.println(subList2);
}
...
}

```

Syntaxe (très allégée) avec lambda expression depuis de jdk 1.8 :

```

package tp.langage.v8.lambda;
....
import java.util.function.Predicate;

public class FilterSortListJava8TestApp {
    ...
    public static List<Person> extractSubListByJava8Predicate(List<Person> pList, final
    Predicate<Person> predicate) {
        final List<Person> sublist = new ArrayList<Person>();
        for (Person p : pList) {
            if (predicate.test(p)) {
                sublist.add(p);
            }
        }
        return sublist;
    }
    public static void mainWithLambdaExpressions() {
        List<Person> pList = StaticPersonList.initList();

        System.out.println("person sublist with age from 20 to 25:");
        List<Person> subList1 = extractSubListByJava8Predicate(pList,
            (person) -> person.getAge() >= 14 && person.getAge() <= 25 );
        System.out.println(subList1);

        System.out.println("person sublist with lastName starting with letter p");
        List<Person> subList2 = extractSubListByJava8Predicate(pList,
            (person) -> person.getLastName().startsWith("p") );
        System.out.println(subList2);
    }
}

```

2.2. Lambda expressions

La nouvelle syntaxe **(arguments) -> corps de la fonction** apportée par le jdk 1.8 est appelée "**lambda expression**".

Derrière cette syntaxe très concise se cache un gros travail de déduction / résolution de la part du compilateur :

- fabrication automatique d'une implémentation respectant l'interface uni-fonctionnelle précisée par le contexte (ex : fonction englobante invoquée)
- mise en rapprochement des arguments de la "lambda expression" avec les paramètres d'entrées de la méthode abstraite (de l'interface fonctionnelle)
- utilisation du corps de la fonction précisé au sein de la "lambda expression" avec éventuelle interprétation contextuelle (lexicale) du mot clef this .
- ...

→ les types des arguments peuvent ainsi être déduits (et leurs compatibilités vérifiées) par le compilateur .

Exemples (variations syntaxiques) :

```
Collections.sort(pList,
(Person p1, Person p2) -> { return Integer.compare(p1.getAge() , p2.getAge()) ; })
```

ou bien avec return implicite (si une seule instruction dans { }) :

```
Collections.sort(pList, (Person p1, Person p2) -> Integer.compare(p1.getAge() , p2.getAge()) )
```

ou bien (avec déduction/inférence des types de p1 et p2 selon le contexte) :

```
Collections.sort(pList, (p1, p2) -> Integer.compare(p1.getAge() , p2.getAge()) )
```

Exemple dans son contexte :

```
public static void sortListComparatorInnerAnonymousImplementation() {
    List<Person> pList = StaticPersonList.initList();
    Collections.sort(pList, new java.util.Comparator<Person>(){
        @Override
        public int compare(Person p1, Person p2) {
            if (p1.getAge() > p2.getAge()) { return 1; }
            else if (p1.getAge() < p2.getAge()) { return -1; }
            else { return 0; }
        }
    });
    System.out.println(pList);
}
```

simplifié en

```
public static void sortListWithLambdaExpression() {
    List<Person> pList = StaticPersonList.initList();
    Collections.sort(pList,
        /* (Person p1, Person p2) -> p1.getAge() == p2.getAge() ? 0 : (p1.getAge() < p2.getAge() ? -1 : 1) */
        (Person p1, Person p2) -> Integer.compare(p1.getAge(), p2.getAge())
        /* (p1,p2) -> p1.getLastname().compareTo(p2.getLastname()) */
    );
    System.out.println(pList);
}
```

NB:

- D'un point de vue "typage de données / signature" , une lambda expression peut être vue par le compilateur comme un équivalent d'une implémentation d'une classe anonyme imbriquée implémentant une interface unifonctionnelle.
- D'un point de vue "performance/implémentation réelle" , une lambda expression est une construction spécifique du compilateur (qui est plus légère et plus performante qu'une classe anonyme imbriquée) . Conceptuellement, on est pas très loin d'un code "inline" du c++ ou d'une référence de fonction fléchée de javascript.

2.3. Référence de fonctions

NB : la (nouvelle) syntaxe **NomClasse::nomFonction** correspond à une **référence de fonction** .
→ les arguments et le corps de la fonction référencée sont alors utilisés de la même façon que ceux d'une "lambda expression" .

NB2 : une référence de fonction peut éventuellement **référer un "constructeur"** tel que le montre cet exemple :

```
....map(person -> new Student(person));
```

pouvant être ré-écrit en

```
...map(Student::new);
```

NB :

A la place d'une lambda expression à n paramètres en entrée, on pourra placer :

- soit une référence de fonction statique avec n paramètres en entrée
- soit une référence de fonction ordinaire (pas statique) avec n-1 paramètres en entrée sachant que lorsque cette méthode sera exécutée , le mot clef **this** sera utilisé en faisant office de premier paramètre d'un équivalent en forme lambda .

Exemple :

```
...
public class Person {
    ...
    public static int compareP1P2ByAge(Person p1, Person p2) {
        return p1.getAge() - p2.getAge() ;
    }

    public int compareThisP2ByAge(Person p2) {
        return this.getAge() - p2.getAge() ;
    }
}
```

```
public static void sortListWithFunctionReference() {
    List<Person> pList = StaticPersonList.initList();
    Collections.sort(pList, Person::compareP1P2ByAge);
    //Person::compareP1P2ByAge() have same code as java.util.Comparator.compare()
    //but with different name and without explicit interface implementation
    System.out.println("via static function reference, pList=" + pList);

    pList = StaticPersonList.initList();
    Collections.sort(pList, Person::compareThisP2ByAge);
    System.out.println("via no static function reference, pList=" + pList);

    System.out.println("liste triée par nom puis par prénom (with function reference:");
    pList.sort(Comparator.comparing(Person::getLastName)
                .thenComparing(Person::getFirstName));
    System.out.println(pList);
}
```

2.4. Scope visibility in lambda expression :

```

public class Java8TestApp {
    private String message="Hello World (V8)";
    public static void main(String[] args) {
        (new Java8TestApp()).testRun();
    }
    public void testRun(){
        //lambda expression (compatible run()) utilisant "message" du scope parent:
        /* Runnable r1 = () -> System.out.println(this.message); */
        Runnable r1 = () -> System.out.println(message);
        Thread t1 = new Thread(r1); t1.start();
    }
}

```

NB : En langage **java** , une lambda expression est formulée par $(p) \rightarrow p.getXy() \geq this.seuil$

en langage **javascript** , une fonction fléchée est formulée par $(p) \Rightarrow p.xy \geq this.seuil$

Dans les 2 cas le mot clef **this** (utilisé dans une lambda) fait référence au contexte objet englobant.

2.5. Function, Predicate, Supplier, Consumer, ...

Avec un peu plus de recul et plus formellement :

Une **interface fonctionnelle** correspond à un **type (de traitement fonctionnel) abstrait**.

Une **"lambda expression"** (ou bien une référence de méthode) correspond à une **implémentation concrète d'une interface fonctionnelle**.

→ on peut écrire

```
Runnable r1 = () -> System.out.println("message");
```

D'autre part, le package **java.util.function** comporte un paquet d'**interfaces uni-fonctionnelles génériques** :

Function<T,R> - takes an object of type T and returns R.

Supplier<T> - just returns an object of type T.

Predicate<T> - returns a boolean value based on input of type T.

Consumer<T> - performs an action with given object of type T (no return)

BiFunction - like Function but with two parameters.

BiConsumer - like Consumer but with two parameters

Interface Fonctionnelle	signature de la fonction	exemple
UnaryOperator <T>	T apply(T t)	String::toLowerCase
BinaryOperator <T>	T apply(T t1, T, t2)	BigInteger::add
Predicate <T>	boolean test(T t)	Collection::isEmpty
Function <T, R>	R apply(T t)	Arrays::asList
BiFunction <T, U, R>	R apply(T t, U u)	(x, y) -> x * y
Supplier <T>	T get()	Instant::now
Consumer <T>	void accept(T t)	System.out::println
autres (variantes , ...)

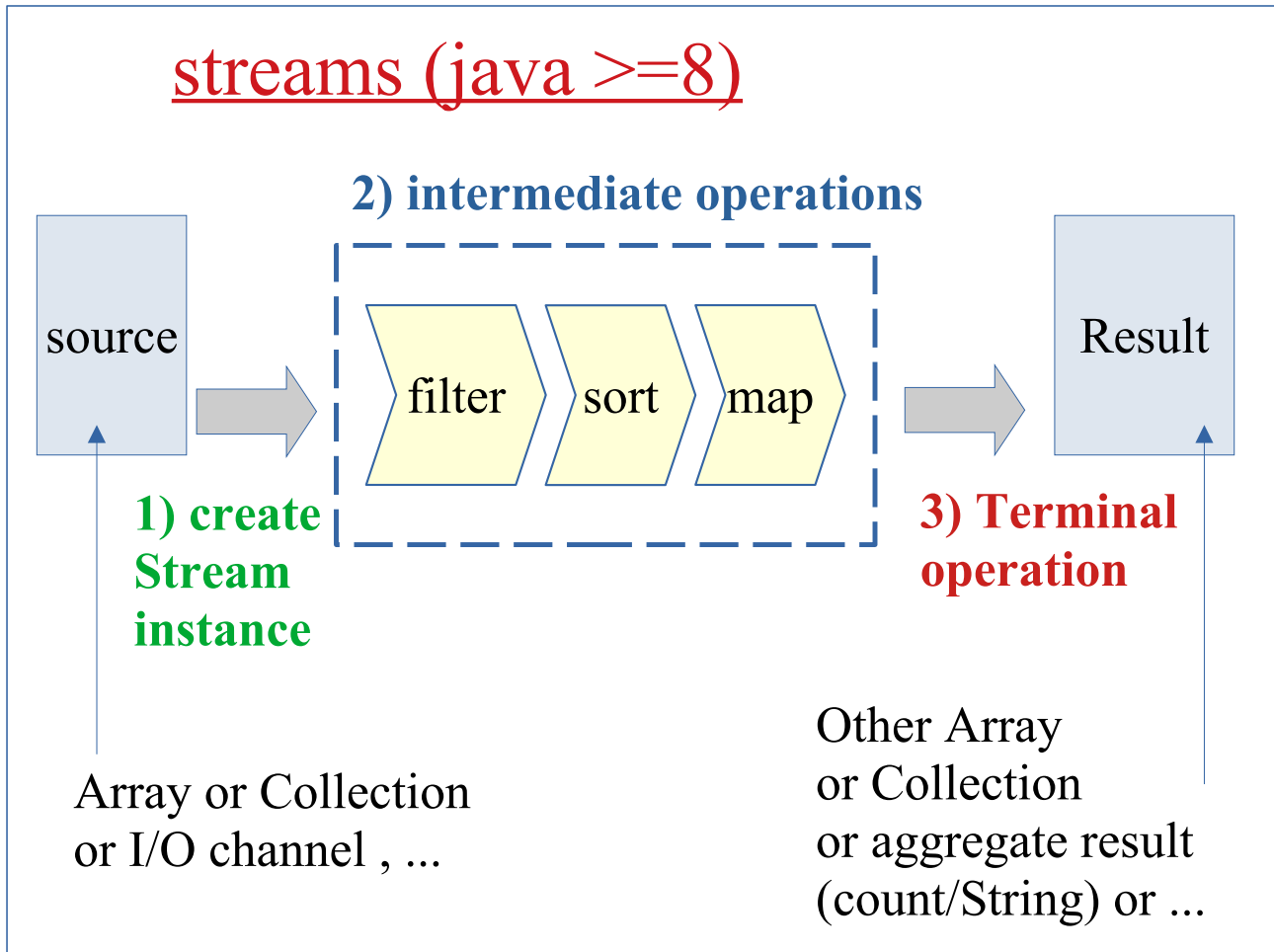
Exemple d'utilisation (explicite et directe) :

```
List<String> liste = initListOfString();
Function<String, String> atrFct = (name) -> {return "@" + name;} ;
Function<String, Integer> lengthFct = (name) -> name.length() ;
//ou bien Function<String, Integer> lengthFct = String::length ;
for (String s : liste) {
    System.out.println( atrFct.apply(s) + " , length=" + lengthFct.apply(s));
}
```

L'utilisation des types fonctionnels abstraits et génériques (de **java.util.function**) est souvent effectuée indirectement en tant que paramètres des opérations (sort, filter, map, ...) sur les "Streams".

III - Streams (prog. fonctionnelle)

1. Streams (java 8 et plus)



1.1. Stream et opérations (sort , filter , map , reduce , collect , ...)

`java.util.stream.Stream` (depuis le jdk 1.8) permet d'effectuer **des opérations de tri , filtrage , ... avec une syntaxique très concises sur des flux de données en vrac** (*bulk data operations in english*).

Exemple:

```
public static void mapWithStream(){ // map() is for "transformation"

    List<Person> persons = StaticPersonList.initList();
    Stream<Student> studentsStream = null;
    /*
    //v1 (explicit):
    studentsStream = persons.stream().filter(p -> p.getAge() <= 30)
        .map(new Function<Person, Student>() {
            @Override
            public Student apply(Person person) {
                return new Student(person);
            }
        });

    //v2 (with lambda expression):
    studentsStream = persons.stream().filter(p -> p.getAge() <=30 )
        .map(person -> new Student(person));
    */

    //v3 (with function/constructor reference):
    studentsStream = persons.stream().filter(p -> p.getAge() <=30 )
        .map(Student::new);

    System.out.println("liste of students:");
    studentsStream.forEach(System.out::println);

    //with collect() terminal operation (at end of operation stream) :
    List<Student> students = persons.stream().filter(p -> p.getAge() <=25 )
        .map(Student::new)
        .collect(Collectors.toList()); //or .collect(Collectors.toCollection(ArrayList::new));

    System.out.println("liste of (youngs) students:" + students);
}
```

```
List<Person> listePersonnesFiltreesTrieesEtTransformees =
    persons.stream()
    .filter( (p)->p.getAge()>=18 )
    .sorted( (p1,p2)->Integer.compare(p1.getAge(), p2.getAge()))
    .map( (p) -> { p.setLastname(p.getLastname().toUpperCase()); return p; } )
    .collect(Collectors.toList());
```

```
public static void skipAndLimitOnStream(){
    final List<Integer> demoValues = Arrays.asList(1,2,3,4,5,6,7,8,9,10);
    //limit the input -> [1, 2, 3, 4] :
    System.out.println(demoValues.stream().limit(4).collect(Collectors.toList()));
    //jumping over the first 4 elements -> [5, 6, 7, 8, 9, 10] :
    System.out.println(demoValues.stream().skip(4).collect(Collectors.toList()));
}
```

```
public static void reduceStreamToASingleResult(){
    final List<String> demoValues = Arrays.asList("1_2_3","-4_5_6","-7_8_9");
    System.out.println("stream reduced to as single optional result: "
        + demoValues.stream().reduce(String::concat));

    final List<Integer> demoValues2 = Arrays.asList(5,8,12);
    System.out.println("stream reduced to as single result (somme des valeurs): "
        + demoValues2.stream().reduce(0 , (x,y) -> x+y));
    //.reduce(identity, accumulator) where identity is initialValue
    // (or default result if stream is empty)
}
```

```
public static void filteringWithStreamFilterAndLambda(){
    List<Person> pList = StaticPersonList.initList();
    pList.stream().filter( (p) -> p.getAge() >= 32 ).forEach( (p) -> System.out.println(p) );

    List<String> liste = initListOfString();
    //liste.stream().map((s)->s.toUpperCase()).forEach(System.out::println);
    liste.stream().map(String::toUpperCase).forEach(System.out::println);

    //.stream() for sequential operations , .parallelStream() for parallel operations
    //source of stream can be:
    //Stream.of(val1,val2,val3...) , Stream.of(array) and list.stream().
    //or final Stream<String> splitOf = Stream.of("A,B,C".split(", "));
```

```
}
```

Autres exemples :

```
String s = IntStream.iterate(1, i -> i * 2) //generation d'une sequence a priori infinie 1 , 2 , 4 , ...
    .limit (10) //limitation a 10 iterations
    .mapToObj(String :: valueOf) //transformation de chaque élément en String
    .collect(Collectors.joining(" ; ")); //collector rassemblant tout en une seule grande chaine
    //via des concatenations (en boucle) avec le separateur/jointeur spécifié
System.out.println(s);           //affiche 1 ; 2 ; 4 ; 8 ; 16 ; 32 ; 64 ; 128 ; 256 ; 512
```

```
List<Product> listProd = ProductUtil.initSampleProductList();
Optional<Product> ofp=listProd.stream().filter(p -> p.getPrice() >= 200).findFirst();
System.out.println("premier produit trouve avec prix >= 200 : " + ofp.orElse(null));

boolean auMoinsUnProduitExistantQuiCommenceParPrinter =
    listProd.stream().anyMatch(p -> p.getLabel().startsWith("printer"));
if(auMoinsUnProduitExistantQuiCommenceParPrinter)
    System.out.println("au moins un des produits existants commence par printer");
```

```
double prixMaxi =listProd.stream()
    .map(p -> p.getPrice())
    .reduce(Double::max).orElse(0.0);
System.out.println("prixMaxi="+prixMaxi);
```

```
List<String> liste1 = Arrays.asList("rouge", "vert" , "bleu");
List<String> liste2 = Arrays.asList("jaune", "orange" , "bleu");
List<String> listeCouleurs = Stream.concat(liste1.stream(), liste2.stream())
    .distinct()
    .collect(Collectors.toList());
System.out.println("liste des couleurs (sans doublon)="+listeCouleurs);
```

→ liste des couleurs (sans doublon)=[rouge, vert, bleu, jaune, orange]

```
List <Point> points = Arrays.asList(new Point(1, 2),new Point(2, 4));
System.out.println("liste de points =" + points);
points.stream().forEach(p -> p.translate(10, 5)); //forEach() = operation terminale (void)
    // p.translate() : void/ne retourne rien et modifie p
System.out.println("Apres translation(10,5) , liste de points =" + points);
```

liste de points =[java.awt.Point[x=1, y=2], java.awt.Point[x=2, y=4]]
 Apres translation(10,5) , liste de points =[java.awt.Point[x=11,y=7], java.awt.Point[x=12,y=9]]

IV - Optional , DateTime , ...

1. Optional , DateTime , ...

1.1. Optional<T>

`java.util.Optional` (depuis le jdk 1.8) correspond à un **conteneur (jamais "null") de valeur "objet" optionnelle** . Ceci un sémantiquement plus précis qu'une valeur "null" .

Depuis java 8 , le **type de retour** `Optional<T>` plutôt que T éventuellement null ne fait qu'**expliquer clairement** (via le type , sans autre commentaire ou documentation supplémentaire) **l'aspect optionnel d'une référence vers un objet en valeur de retour**

Cela pourrait correspondre à la **multiplicité 0..1** en UML mais malheureusement `Optional<T>` n'est prévu que pour n'être utilisé qu'au niveau des types de retour (pas sur un type d'attribut/propriété).
Principale raison : `Optional<T>` n'est pas bien gérée par la sérialisation JSON (valeur null préférée) ni par la persistance automatique de JPA/Hibernate (valeur null préférée).

Autrement dit, **`Optional<T>` est surtout utile pour éviter un `NullPointerException` en plein milieu d'un enchaînement de traitements via les streams de java >=8 .**

Exemple :

```
...
public static List<String> initWinnerList(){
    String[] winnerArray = {"bob", "anna", "alice"};
    List<String> winnerList = Arrays.asList(winnerArray);
    return winnerList;
}
public static Optional<String> getWinnerByName(String name){
    for(String s : initWinnerList()){
        if(s.equals(name))
            return Optional.of(s);
    } /*else*/
    return Optional.empty(); //instead of return null
}
public static void testOptional(){
    Optional<String> opStr = getWinnerByName("alice");
    System.out.println("Optional<String> opStr = " + opStr );
    if(opStr.isPresent())
        System.out.println(opStr.get());
    //with lambda expression:
    opStr.ifPresent( (s) -> System.out.println(s) );
}
```

```

opStr = getWinnerByName("looser");
System.out.println("Optional<String> opStr = " + opStr );
//System.out.println(opStr.get());
//java.util.NoSuchElementException instead of NullPointerException
opStr.ifPresent((s)->System.out.println(s));

//String str = "abc";
String str = null;
Optional<String> opS = Optional.ofNullable(str); //build .empty() if null
System.out.println("Optional<String> opS = " + opS );

String msg = opS.map((notNullStr)-> "not null string value : " + notNullStr)
                .orElse("empty optional");
System.out.println(msg);
}

```

NB :

optional.get(); retourne la valeur interne (non nulle) ou bien retourne une **exception** si vide/empty.

optional.orElse(null); retourne la valeur interne (non nulle) ou bien retourne **null** si vide/empty.

```

public static void testDiversAvecNouvellesMethodesDeOptionalDepuisJava9() {
    Optional<String> opS1 = Optional.of("s1");
    Optional<String> opS2 = /*Optional.ofNullable(null); */ Optional.empty();

    //op.or(...) permet de construire et fournir un autre optional (valeur par défaut
    // ou plan B) si l'optionnel original est null/empty :
    System.out.println(opS1.or(()->Optional.of("default_string"))); //affiche Optional[s1]
    System.out.println(opS2.or(()->Optional.of("default_string")));
    //affiche Optional[default_string]

    //op.ifPresentOrElse(nomEmptyConsumer_as_lambda , emptyActionLambda)
    //permet de déclencher alternativement une lambda ou une autre en fonction d'un
    // contenu vide ou pas , pas d'enchaînement après (opération terminale en "void")
    opS1.ifPresentOrElse( (value) -> System.out.println("opS1="+value),
                        () -> System.out.println("value of opS1 is empty"));
    //affiche opS1=s1
    opS2.ifPresentOrElse( (value) -> System.out.println("opS2="+value),
                        () -> System.out.println("value of opS2 is empty"));
    //affiche value of opS2 is empty
}

```

1.2. LocalTime , LocalDate (de java.time)

java.time. **LocalTime** , **LocalDate** (d'inspiration JodaTime) sont plus simples à utiliser que **Date** et **Calendar**

Exemple :

```
public static void testLocalTimeAndLocalDate(){
LocalTime now = LocalTime.now();      System.out.println("now is " + now);
LocalTime later = now.plus(8, ChronoUnit.HOURS);
System.out.println("later (now+8h) is " + later);

LocalDate today = LocalDate.now();      System.out.println("today is " + today);
LocalDate thirtyDaysFromNow = today.plusDays(30);
System.out.println("thirtyDaysFromNow is " + thirtyDaysFromNow);
LocalDate nextMonth = today.plusMonths(1); System.out.println("nextMonth is " + nextMonth);
LocalDate aMonthAgo = today.minusMonths(1); System.out.println("aMonthAgo is " + aMonthAgo);

//LocalDate date14July2015 = LocalDate.of(2015, 7, 14);
LocalDate date14July2015 = LocalDate.of(2015, Month.JULY, 14);

LocalTime time = LocalTime.of(14 /*h*/, 15 /*m*/, 0 /*s*/);
LocalDateTime datetime = date14July2015.atTime(time);
System.out.println("le 14 juillet 2015 à 14h15 : " + datetime);

LocalDate date1=today , date2=nextMonth;
Period p1 = Period.between(date1, date2) ;   System.out.println("periode p1:" + p1);

LocalTime time1= time;
LocalTime time2= LocalTime.of(14 /*h*/, 30 /*m*/, 0 /*s*/);
Duration d = Duration.between(time1, time2);   System.out.println("durée d:" + d);

Duration twoHours = Duration.ofHours(2); System.out.println("durée de 2 heures:" + twoHours);
Duration tenMinutes = Duration.ofMinutes(10); System.out.println("durée de 10 minutes:" + tenMinutes);
Duration thirtySecs = Duration.ofSeconds(30); System.out.println("durée de 30 secondes:" + thirtySecs

LocalTime t2 = time.plus(twoHours);      System.out.println("14h15 plus 2 heures:" + t2);

//.with(temporalAdjuster) :
```

```

LocalDate premierJourDeCetteAnnee=
    LocalDate.now().with(TemporalAdjusters.firstDayOfYear());
System.out.println("premierJourDeCetteAnnee="+premierJourDeCetteAnnee);

LocalDate dernierJourDuMois= LocalDate.now().with(TemporalAdjusters.lastDayOfMonth());
System.out.println("dernierJourDuMois="+dernierJourDuMois);

ZoneId myZone = ZoneId.systemDefault();
System.out.println("my (local) zoneId is:" + myZone);

//lien entre java.util.Date et java.time... :
Date date = new Date();
Instant nowInstant = date.toInstant();
LocalDateTime dateTime = LocalDateTime.ofInstant(nowInstant, myZone);
System.out.println("today/now from Date:" + dateTime);
}

```

```

LocalDateTime now = LocalDateTime.now();
System.out.println("basic/default display of LocalDateTime.now() :"+ now);
//ex :2021-11-23T09:28:12.289915600

Instant instantT = now.atZone(ZoneId.systemDefault()).toInstant();
long nbMsSinceFirstJanuary1970GMT = instantT.toEpochMilli();
System.out.println("instantT.toEpochMilli() , timestamp , nb ms since 1970-01-01 00:00:00
GMT="+ nbMsSinceFirstJanuary1970GMT); //ex : 1637656092289

```

```

LocalDate nowDate = LocalDate.now();
System.out.println("basic/default display : today (nowDate local) is " + nowDate); //ex :2021-11-23
String sdate_fr_2=nowDate.format(DateTimeFormatter.ofPattern("EEEE, dd MMMM
yyyy",Locale.FRENCH)); // EEEE means fullname of weekday
System.out.println("sdate_fr_2="+sdate_fr_2); //exemple: dimanche, 13 septembre 2020

```

1.3. Encodage base64 (java.util.Base64)

```

public static void testBase64(){
    try {
        // Encode using basic encoder :
        String base64encodedString =
            Base64.getEncoder().encodeToString("Myjava8String".getBytes("utf-8"));
        System.out.println("Base64 Encoded String (Basic) :"+ base64encodedString);
    }
}

```

```
// Decode :  
byte[] base64decodedBytes = Base64.getDecoder().decode(base64encodedString);  
System.out.println("Original String: " + new String(base64decodedBytes, "utf-8"));  
} catch (UnsupportedEncodingException e) {  
    e.printStackTrace();  
}  
}
```

1.4. Diverses autres nouveautés de java 8

Amélioration de l'introspection si option "-arguments" au lancement du compilateur "javac"

→ `parameter.getName()` retourne véritable nom du paramètre (stocké dans `byteCode`) plutôt que "arg0", "arg1", ...

`@Repeatable` ([java.lang.annotation](#)) , ...

V - Modules (à partir de java 9)

1. Modules java 9+ (essentiel)

1.1. Problèmes et limitations en java 8

En Java 8 et antérieur, à l'exécution d'une application, la JVM recherche les classes utilisées par l'application dans :

- Les classes prédéfinies de la plate-forme Java : stockées dans l'**énorme** fichier **monolithique** `rt.jar` (*jusqu'à 53 Mo en Java 8 !*).
- Le **classpath** paramétré au lancement (un ensemble de chemins relatifs ou absolus vers des `.jars` ou des répertoires contenant des arborescences en package contenant des fichiers `".class"`)

Selon d'éventuelles différences de contextes (ordinateurs différents , "jdk" différents, ...) , **le classpath n'est pas forcément le même à la compilation et à l'exécution** et cela peut engendrer `"java.lang.NoClassDefFoundError"` !!!!

En Java 8 et antérieur, à l'exécution d'une application, le **"classloader"** de la JVM a le comportement suivant :

- il charge les classes sur demande lors d'un `new` ou d'un appel à une méthode statique.
- il charge les classes linéairement (selon l'ordre des jars dans le classpath qui est donc très important) et s'arrête à la première classe correspondant au nom complet demandé.
- aucune vérification n'est faite à l'exécution sur l'existence de plusieurs occurrences d'une même classe.
- aucune vérification n'est faite au démarrage de l'application sur la présence de tous les jars / classes nécessaires au bon fonctionnement de l'application.
- le classpath est plat sans aucune notion de dépendances entre jar.

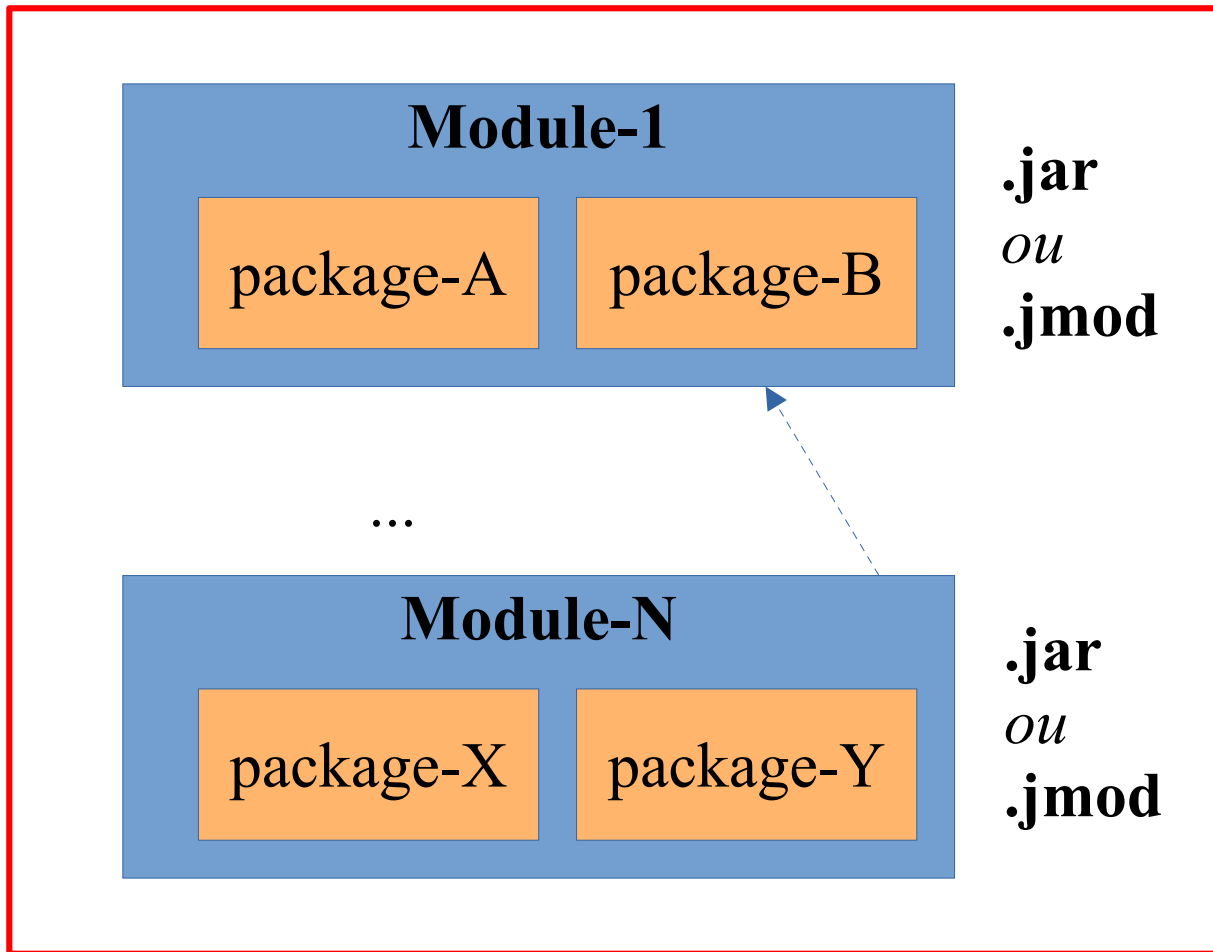
1.2. Projet JIGSAW (modularisation de la jvm depuis java >=9)

Principaux **apports** de la **modularisation** de la JVM (depuis V9) :

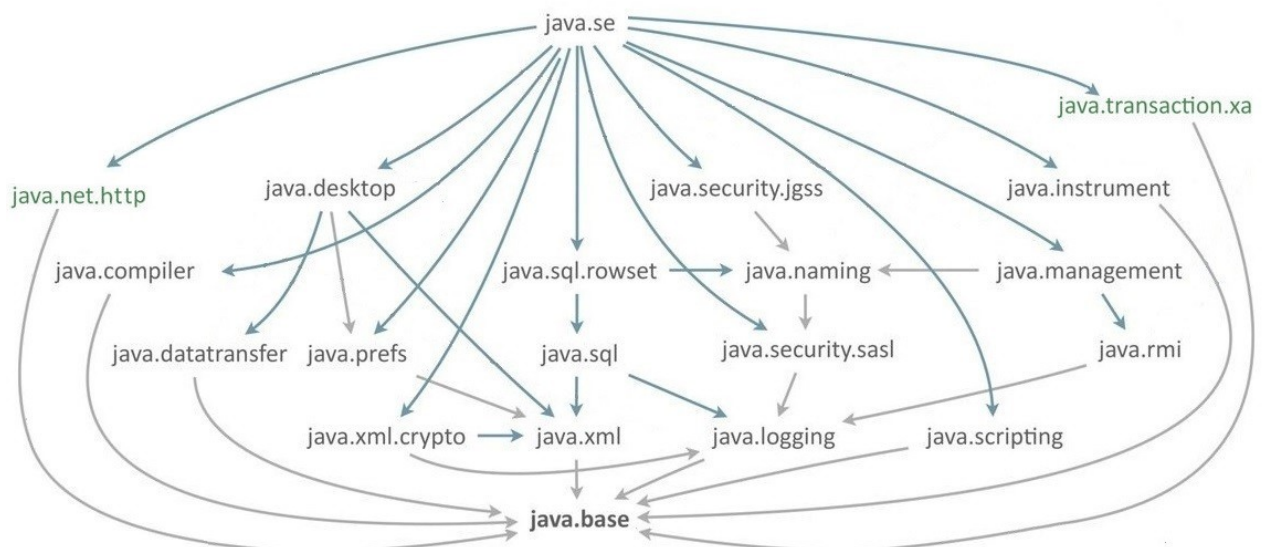
- **un classpath sous forme d'arbre de dépendances**
- **une vérification de la présence de tous les modules nécessaires à notre application au démarrage**, sans quoi l'application ne démarre pas.
- **un renforcement de la sécurité**, seuls les packages exportés explicitement par un module sont visibles par un autre
- la JVM elle-même est **modulaire** (*l'énorme fichier `rt.jar` n'existe plus depuis java 9*)
(NB: la taille du JRE a sensiblement diminué à partir de java 11)

NB: Un module **"java >=9"** est un **.jar** comportant le fichier ***module-info.java*** (compilé en `module-info.class`) à la racine du module .

Appli ou JVM modulaire depuis java 9










































































Exemple de graphe de dépendances entre modules d'un jdk récent :



Modules du jdk 11 :

Java > jdk-11.0.4 > jmods

Nom	Modifié le	Type	Taille
 java.base.jmod	19/08/2019 20:57	Fichier JMOD	18 021 Ko
 java.compiler.jmod	19/08/2019 20:57	Fichier JMOD	109 Ko
 java.datatransfer.jmod	19/08/2019 20:57	Fichier JMOD	50 Ko
 java.desktop.jmod	19/08/2019 20:57	Fichier JMOD	11 799 Ko
 java.instrument.jmod	19/08/2019 20:57	Fichier JMOD	116 Ko
 java.logging.jmod	19/08/2019 20:57	Fichier JMOD	110 Ko
 java.management.jmod	19/08/2019 20:57	Fichier JMOD	867 Ko
 java.management.rmi.jmod	19/08/2019 20:57	Fichier JMOD	87 Ko
 java.naming.jmod	19/08/2019 20:57	Fichier JMOD	436 Ko
 java.net.http.jmod	19/08/2019 20:57	Fichier JMOD	680 Ko
 java.prefs.jmod	19/08/2019 20:57	Fichier JMOD	53 Ko
 java.rmi.jmod	19/08/2019 20:57	Fichier JMOD	369 Ko
 java.scripting.jmod	19/08/2019 20:57	Fichier JMOD	44 Ko
 java.se.jmod	19/08/2019 20:57	Fichier JMOD	3 Ko
 java.security.jgss.jmod	19/08/2019 20:57	Fichier JMOD	638 Ko
 java.security.sasl.jmod	19/08/2019 20:57	Fichier JMOD	79 Ko
 java.smartcardio.jmod	19/08/2019 20:57	Fichier JMOD	57 Ko
 java.sql.jmod	19/08/2019 20:57	Fichier JMOD	74 Ko
 java.sql.rowset.jmod	19/08/2019 20:57	Fichier JMOD	184 Ko
 java.transaction.xa.jmod	19/08/2019 20:57	Fichier JMOD	4 Ko
 java.xml.crypto.jmod	19/08/2019 20:57	Fichier JMOD	635 Ko
 java.xml.jmod	19/08/2019 20:57	Fichier JMOD	4 380 Ko
 jdk.accessibility.jmod	19/08/2019 20:57	Fichier JMOD	484 Ko
 jdk.aot.jmod	19/08/2019 20:57	Fichier JMOD	274 Ko
 jdk.attach.jmod	19/08/2019 20:57	Fichier JMOD	38 Ko
 jdk.charsets.jmod	19/08/2019 20:57	Fichier JMOD	1 451 Ko
 jdk.compiler.jmod	19/08/2019 20:57	Fichier JMOD	6 904 Ko
⋮			
 jdk.crypto.cryptoki.jmod	19/08/2019 20:57	Fichier JMOD	305 Ko
 jdk.crypto.ec.jmod	19/08/2019 20:57	Fichier JMOD	148 Ko
 jdk.crypto.mscapi.jmod	19/08/2019 20:57	Fichier JMOD	61 Ko
 jdk.dynalink.jmod	19/08/2019 20:57	Fichier JMOD	159 Ko
 jdk.editpad.jmod	19/08/2019 20:57	Fichier JMOD	7 Ko
 jdk.hotspot.agent.jmod	19/08/2019 20:57	Fichier JMOD	2 298 Ko
 jdk.httpserver.jmod	19/08/2019 20:57	Fichier JMOD	98 Ko
 jdk.internal.ed.jmod	19/08/2019 20:57	Fichier JMOD	8 Ko
 jdk.internal.jvmstat.jmod	19/08/2019 20:57	Fichier JMOD	88 Ko
 jdk.internal.le.jmod	19/08/2019 20:57	Fichier JMOD	181 Ko
 jdk.internal.opt.jmod	19/08/2019 20:57	Fichier JMOD	80 Ko
 jdk.internal.vm.ci.jmod	19/08/2019 20:57	Fichier JMOD	399 Ko
 jdk.internal.vm.compiler.jmod	19/08/2019 20:57	Fichier JMOD	6 025 Ko
 jdk.internal.vm.compiler.manag...	19/08/2019 20:57	Fichier JMOD	12 Ko
 jdk.jartool.jmod	19/08/2019 20:57	Fichier JMOD	195 Ko
 jdk.javadoc.jmod	19/08/2019 20:57	Fichier JMOD	1 594 Ko
 jdk.jcmd.jmod	19/08/2019 20:57	Fichier JMOD	143 Ko
 jdk.jconsole.jmod	19/08/2019 20:57	Fichier JMOD	456 Ko
 jdk.jdeps.jmod	19/08/2019 20:57	Fichier JMOD	710 Ko
 jdk.jdi.jmod	19/08/2019 20:57	Fichier JMOD	841 Ko
 jdk.jdp.agent.jmod	19/08/2019 20:57	Fichier JMOD	121 Ko
 jdk.jfr.jmod	19/08/2019 20:57	Fichier JMOD	414 Ko
 jdk.jlink.jmod	19/08/2019 20:57	Fichier JMOD	388 Ko
 jdk.jshell.jmod	19/08/2019 20:57	Fichier JMOD	626 Ko
 jdk.jsobject.jmod	19/08/2019 20:57	Fichier JMOD	6 Ko
 jdk.jstatd.jmod	19/08/2019 20:57	Fichier JMOD	33 Ko
⋮			
 jdk.localedata.jmod	19/08/2019 20:57	Fichier JMOD	9 327 Ko
 jdk.management.agent.jmod	19/08/2019 20:57	Fichier JMOD	80 Ko
 jdk.management.jfr.jmod	19/08/2019 20:57	Fichier JMOD	34 Ko
 jdk.management.jmod	19/08/2019 20:57	Fichier JMOD	68 Ko
 jdk.naming.dns.jmod	19/08/2019 20:57	Fichier JMOD	57 Ko
 jdk.naming.rmi.jmod	19/08/2019 20:57	Fichier JMOD	19 Ko
 jdk.net.jmod	19/08/2019 20:57	Fichier JMOD	21 Ko
 jdk.pack.jmod	19/08/2019 20:57	Fichier JMOD	131 Ko
 jdk.rmic.jmod	19/08/2019 20:57	Fichier JMOD	516 Ko
 jdk.scripting.nashorn.jmod	19/08/2019 20:57	Fichier JMOD	2 143 Ko
 jdk.scripting.nashorn.shell.jmod	19/08/2019 20:57	Fichier JMOD	56 Ko
 jdk.sctp.jmod	19/08/2019 20:57	Fichier JMOD	23 Ko
 jdk.security.auth.jmod	19/08/2019 20:57	Fichier JMOD	73 Ko
 jdk.security.jgss.jmod	19/08/2019 20:57	Fichier JMOD	24 Ko
 jdk.unsupported.desktop.jmod	19/08/2019 20:57	Fichier JMOD	14 Ko
 jdk.unsupported.jmod	19/08/2019 20:57	Fichier JMOD	18 Ko
 jdk.xml.dom.jmod	19/08/2019 20:57	Fichier JMOD	42 Ko
 jdk.zipfs.jmod	19/08/2019 20:57	Fichier JMOD	87 Ko

1.3. option "--list-modules" de java

print_list_modules.bat

```
set JAVA_HOME=D:\Prog\java\jdk-14.0.1_windows-x64_bin\jdk-14.0.1
"%JAVA_HOME%\bin\java" --list-modules
pause
```

--> affiche la liste des modules systèmes de la JRE (et du JDK) , soit par exemple :

```
java.base@14.0.1
java.compiler@14.0.1
java.datatransfer@14.0.1
java.desktop@14.0.1
...
jdk.xml.dom@14.0.1
jdk.zipfs@14.0.1
```

1.4. option "--module-path" de java

L'option **--module-path** d'une ligne de commande java permet de spécifier le "module path". C'est une **liste de un ou plusieurs répertoires qui comportent des modules** .

Exemple :

prepare_modules.bat

```
cd /d "%~dp0"
REM cd ..
REM mvn package
REM cd /scripts
copy ..\mod_xx\target\mod_xx.jar .\my_modules
copy ..\mod_yy\target\mod_yy.jar .\my_modules
pause
```

launch_app_withmodules.bat

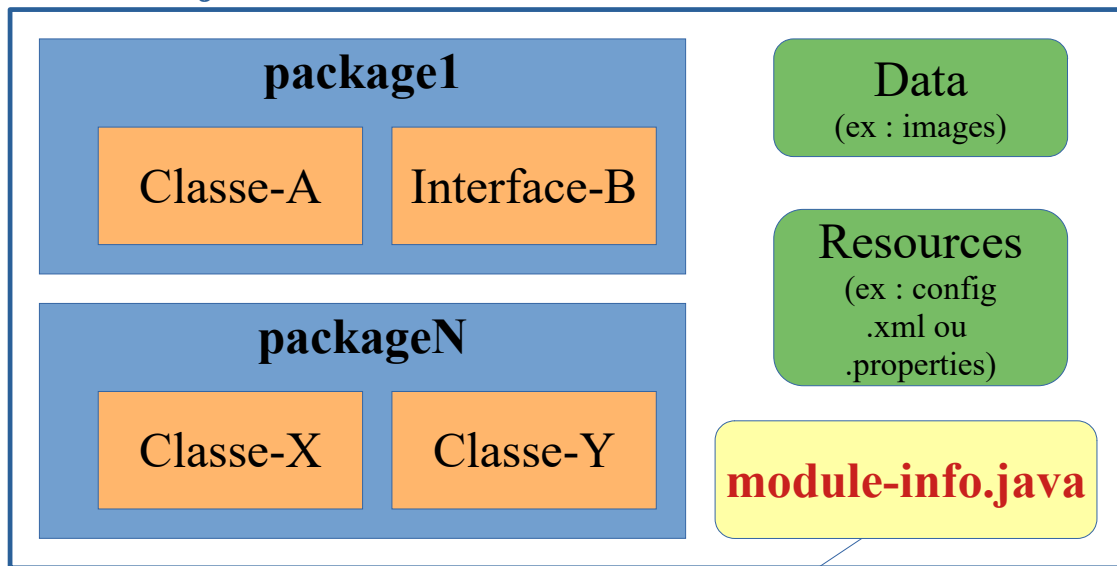
```
cd /d "%~dp0"
REM java --module-path modules_directory -m mainModuleName/mainClassFullName
java --module-path .\my_modules -m tp.module.modyy/tp.mod.mod_yy.app.MyApp
pause
```

1.5. Structure d'un module

Un module java (packagé en un ".jar" éventuellement renommé) est **l'association** de :

- un **paquet de packages java** (un sous package précis "xxx.yyy.zzz" ne peut être rangé que dans un seul module)
- un **paquet d'éventuelles ressources** (fichiers annexes : configuration_xml , images, ...)
- un fichier de configuration **module-info.java**

Module java 9+



*NB: si module-info.java pas spécifié , alors module automatique (exports *, requires *)*

```
module m2 {
    exports package1;
    requires m1;
}
```

1.6. Types de modules

Types de modules	Caractéristiques
System Modules	modules prédéfinis livrés avec jre/jdk (listés via java --list-modules)
Application Modules	modules applicatifs (.jar fabriqués avec module-info.class)
Automatic Modules	unofficial module (.jar fabriqués sans module-info.class) (with names based on ".jar" name) chargés depuis le --module-path . <i>Attention : sans module-info.class , ces modules ont par défaut un accès complet en lecture vers tous les autre modules chargés .</i>
Unnamed Module	" module fourre tout sans nom " comportant toutes les classes et packages chargés depuis des ".jar" via le --classpath (et pas le nouveau --module-path) -> pour récupérer tout les éléments codés en java <= 8 .

1.7. Descripteur d'un module (module-info.java)

name	nom de notre module (avec d'éventuel(s) "." mais pas de "-")
dependencies (<i>requires</i>)	liste des modules dont on dépend
public packages (<i>exports</i>)	liste des packages que l'on rend accessibles aux autres modules. Normalement, un bon module ne doit exposer publiquement qu'une petite partie de son API (interfaces , pas l'implémentation)
services offered (<i>provides</i>)	implémentation de services (à injecter/utiliser dans d'autres modules)
services consumed (<i>uses</i>)	autoriser le module courant à utiliser des services d'autres modules implémentant une certaine interface ou classe abstraite
reflection permissions (<i>opens</i>)	autoriser ou pas d'autres classes à utiliser la "Reflection java" pour accéder aux éléments privés d'un de nos packages

1.8. Variantes de dépendances (requires)

requires	module my.module { requires module.name; }	dépendance à la fois à la compilation et à l'exécution ("runtime")
requires static	module my.module { requires static module.name; }	dépendance à la compilation seulement , facultatif à l'exécution ("runtime")
requires transitive	module my.module { requires transitive module.name; }	dépendance automatique transitive (au niveau du futur module client qui n'aura donc pas besoin d'explicitement les dépendances indirectes)

comportement "requires transitive"

Si yy utilise xx utilisant lui même zz et que xx expose une interface de zz
alors yy est alors obligé d'utiliser zz
Si le module xx comporte "**requires transitive** zz"
alors yy aura juste besoin d'exprimer "requires xx" (pas besoin de requires zz")

comportement "requires static"

Le (rare) "**requires static** moduleOptionnelPasToujoursUtilise"
signifie "obligatoire à la compilation" et "facultatif à l'exécution"
avec éventuelle NoClassDefFoundError à gérer au runtime .

1.9. Variantes d'exportations / expositions

exports	module my.module {	exportation explicite
----------------	--------------------	------------------------------

	<code>exports com.my.package.name;</code> <code>}</code>	
exports ... to ...	<code>module my.module {</code> <code> export com.my.package.name to com.specific.package;</code> <code>}</code>	(rare) exportation limitée/restreinte à certain(s) autres package(s) ami(s).

Comportement des "exports"

NB: **exports p1 n'exporte pas les sous packages p1.aa et p1.bb**

Un même nom complet de package ne peut pas être placé dans plusieurs modules (mais dans un unique module)

1.10. interfaces et implémentations de Services

uses	<code>module my.module {</code> <code> uses interfaceOrAbstractClass.name;</code> <code>}</code>	besoin de consommer/injecter un certain type de service
provides ... with ...	<code>module my.module {</code> <code> provides MyInterface with MyImpl;</code> <code>}</code>	Notre module fournit l'implémentation d'un certain type de service

provides <type> **with** <type> et **uses** <type> ont été conçus pour mettre en oeuvre une sorte d'injection de dépendances (selon packages d'interfaces) et avec implémentations internes cachées mais cependant chargées au runtime (selon le paramétrage de `--module-path`)

Exemple de module abstrait (packagé en `my_modules/mod_aa_itf.jar`) :

```
module tp.module.modaa.itf {
    exports tp.mod.mod_aa.itf;
}
```

et

```
package tp.mod.mod_aa.itf;

public interface MyDisplayApi {
    void display(String msg);
    void displayEx(String msg);
}
```

Exemple de module d'implémentation (packagé en `my_modules/mod_aa_impl.jar`) :

```
module tp.module.modaa.aa.impl1 {
```

```

requires tp.module.modaa.itf;
provides tp.mod.mod_aa.itf.MyDisplayApi with tp.mod.mod_aa.impl1.MyDisplayImplV1;
}

```

et

```

package tp.mod.mod_aa.impl1;
import tp.mod.mod_aa.itf.MyDisplayApi;

public class MyDisplayImplV1 implements MyDisplayApi {
    public void display(String msg) { System.out.println(">>>" + msg); }
    public void displayEx(String msg) { System.out.println(":::" + msg); }
}

```

Exemple d'utilisation de service :

```

module tp.module.modxx {
    ...
    exports tp.mod.mod_xx_ext;
    requires transitive tp.module.modaa.itf;
    uses tp.mod.mod_aa.itf.MyDisplayApi;
}

// pour uses tp.mod.mod_aa.itf.MyDisplayApi;
// besoin de requires transitive tp.module.modaa.itf;
// mais pas besoin de requires transitive tp.module.modaa.impl1 ou impl2;

```

```

package tp.mod.mod_xx_ext;
import java.util.Iterator;
import java.util.ServiceLoader;
import tp.mod.mod_aa.itf.MyDisplayApi;

public class CxExt {
    private static MyDisplayApi displayService = null;

    public static void displayViaMyDisplayApi(String msg) {
        if(displayService==null) {
            Iterable<MyDisplayApi> myDisplayServices = ServiceLoader.load(MyDisplayApi.class);
            //list of found implementations
            Iterator<MyDisplayApi> myDisplayServicesIterator = myDisplayServices.iterator();
            if(myDisplayServicesIterator.hasNext()) {
                displayService = myDisplayServicesIterator.next(); //first implementation is choosen
            }
        }
        if(displayService!=null) {
            displayService.display(msg);
        } else{

```



```

    }
    }
}
System.out.println("no MyDisplayApi implementation found !!!");

```

----> comportement d'un appel à `CxExt.displayViaMyDisplayApi("my top secret message");`

- si aucune implémentation n'est présente dans le `-module-path` ça affiche "*no MyDisplayApi implementation found !!!*"
- si au moins une implémentation (ex : `mod_aa_impl1.jar` ou `mod_aa_impl2.jar`) est présente dans le `-module-path` la première trouvée est alors utilisée et ça affiche ">>> *my top secret message*"

1.11. Variantes d'ouvertures à l'introspection :

open	<code>open module my.module { }</code>	<i>on autorise tous les autres packages à effectuer de l'introspection sur des éléments privés de l'ensemble des éléments de notre module</i>
opens	<code>module my.module { opens com.my.package; }</code>	Seul(s) un ou plusieurs(s) de nos packages sont ouvert à de l'introspection avec "accès aux éléments privés"
opens ... to	<code>module my.module { opens com.my.package to com.specific.package; }</code>	introspection ouverte mais limitée/restreinte à certain(s) autres package(s) ami(s).

opens <package> (à ajouter à côté de exports <package>) permet d'autoriser une introspection évoluée (ex: avec `setAccessible(true)`) .

L'autorisation doit être explicitement donnée (ce qui renforce la sécurité)

Exemple :

Du côté "module ...modxx" :

```

package tp.mod.mod_xx.pub;

public class Cx {
    private String secret="007";
    ...
}

```

et

```

module tp.module.modxx {
    ...
    exports tp.mod.mod_xx.pub;
    opens tp.mod.mod_xx.pub;
}

```

Du côté module client ...modyy :

```
module tp.module.modyy {
    requires tp.module.modxx;
    ...
}
```

et

```
package tp.mod.mod_yy.app;
import java.lang.reflect.Field;
import tp.mod.mod_xx.pub.Cx;

public class MyApp {
    private static void testOpenReflection(Cx objX) {
        Class classX = objX.getClass();
        for(Field f : classX.getDeclaredFields()) {
            f.setAccessible(true);
            String sVal="?";
            try {
                sVal = f.get(objX).toString();
            } catch (IllegalAccessException e) { e.printStackTrace();
                //Exception in thread "main" java.lang.reflect.InaccessibleObjectException:
                //Unable to make field private java.lang.String tp.mod.mod_xx.pub.Cx.secret
                //accessible: module tp.module.modxx does not "opens tp.mod.mod_xx.pub" to module tp.module.modyy
            }
            System.out.println( f.getType().getSimpleName()+ " " + f.getName() + "=" + sVal);
        }
    }

    public static void main(String[] args) {
        Cx objX = new Cx();        testOpenReflection(objX);
    }
}
```

--> affiche une exception de type **java.lang.reflect.InaccessibleObjectException** si pas de opens tp.mod.mod_xx.pub; dansmodxx

--> affiche "**String secret=007**" si **opens** tp.mod.mod_xx.pub; dansmodxx

1.12. Exemple (partiel) de quelques modules java 9+ :

NB: l'aspect "**multi-modules maven**" n'est pas indispensable mais cela introduit une cohérence

structurelle :

pom.xml (multi-modules)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>tp.mod</groupId>
  <artifactId>mod</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>pom</packaging>
  <description>modules=nouveautes de java 9 </description>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <java.version>11</java.version>
    <maven.compiler.release>${java.version}</maven.compiler.release>
  </properties>

  <modules>
    <module>mod_xx</module>
    <module>mod_yy</module>
  </modules>

  <build>
    <pluginManagement>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.8.0</version>
          <configuration>
            <release>${java.version}</release>
            <source>${java.version}</source> <target>${java.version}</target>
          </configuration>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

module "mod_xx" (sera utilisé par "mod_yy")

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>tp.mod</groupId>
```

```

<artifactId>mod</artifactId>
<version>0.0.1-SNAPSHOT</version>
</parent>
<artifactId>mod_xx</artifactId>
<description>mod_xx java>=9 sera utilisé par mod_yy</description>
<build>
  <finalName>${project.artifactId}</finalName>
</build>
</project>

```

module-info.java (à la racine de src/main/java)

```

module tp.module.modxx {
    exports tp.mod.mod_xx.pub;
    exports tp.mod.mod_xx_ext;
}

```

```

package tp.mod.mod_xx.pub;

import tp.mod.mod_xx.internal.InternalCx;

public class Cx {
    public void fl(String s) {
        InternalCx icx = new InternalCx();
        icx.fli("***" + s);
    }
}

```

```

package tp.mod.mod_xx.internal;

public class InternalCx {
    public void fli(String s) { System.out.println("fl:"+s); }
}

```

```

package tp.mod.mod_xx_ext;

import tp.mod.mod_xx_ext.internal.InternalCxExt;

public class CxExt {
    public static double add(double x, double y) { return InternalCxExt.addition(x, y); }
}

```

```

package tp.mod.mod_xx_ext.internal;

public class InternalCxExt {
    public static double addition(double x, double y) { return x+y; }
}

```

module "mod_yy" (utilisant "mod_xx")**pom.xml**

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
  4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>tp.mod</groupId>
    <artifactId>mod</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <artifactId>mod_yy</artifactId>
  <description>mod_yy (java=>9) utilisant mod_xx</description>

  <dependencies>
    <dependency>
      <groupId>tp.mod</groupId>
      <artifactId>mod_xx</artifactId>
      <version>0.0.1-SNAPSHOT</version>
    </dependency>
  </dependencies>

  <build>
    <finalName>${project.artifactId}</finalName>
  </build>
</project>

```

module-info.java (à la racine de src/main/java)

```

module tp.module.modyy {
  requires tp.module.modxx;
  requires java.desktop;
  exports tp.mod.mod_yy.pub;
}

```

```

package tp.mod.mod_yy.pub;
import tp.mod.mod_yy.internal.InternalCy;

```

```

public class Cy {
  public void f2(String s) {
    InternalCy icy = new InternalCy();
    icy.f2i("###" + s);
  }
}

```

```

package tp.mod.mod_yy.internal;
//import tp.mod.mod_xx.internal.InternalCx; //pas accessible car package pas exporté
//import tp.mod.mod_xx_ext.internal.InternalCxExt; //pas accessible car package pas exporté

```

```
import tp.mod.mod_xx.pub.Cx;
import tp.mod.mod_xx_ext.CxExt;

public class InternalCy {
    public void f2i(String s) { System.out.println("f2:"+s); }
    public void f1i(String s) { Cx cx =new Cx(); cx.f1(s); }
    public static double ajouter(double a,double b) { return CxExt.add(a, b); }
    //code impossible (basé sur classe InternalCx pas accessible / pas exportée ):
    //public void forbidden_f1(String s) { InternalCx icx =new InternalCx(); icx.f1(s); }
}
```

```
package tp.mod.mod_yy.app;
import javax.swing.JOptionPane;
import tp.mod.mod_yy.internal.InternalCy;
public class MyApp {
    public static void main(String[] args) {
        InternalCy icy = new InternalCy();
        icy.f2i("java");icy.f1i("java");
        JOptionPane.showMessageDialog(null, "ok with requires java.desktop");
    }
}
```

1.13. Modules explicites , automatiques et "unnamed"

Modules explicites gérés rigoureusement :

accès contrôlés si `module.info.java` est présent

Si dans le module `mod_yy` (utilisant `mod_xx`) , le fichier **`module.info.java`** est présent
il faut alors tout rendre cohérent (exports , requires) :
et alors **seuls les éléments bien exportés par `mod_xx`**
et bien importés (via requires) dans `mod_yy` seront accessibles dans `mod_yy`.

Modules automatiques sans gestion rigoureuse :

accès libres **mais pas rigoureux** si pas de `module.info.java` (modules "automatiques")

NB: si jamais de `module.info.java` (pour chaque module)
alors tout ce qui est publique est accessible --> comme ancien comportement de java <=8 .

requires * par défaut si `module.info.java` est absent (du coté module utilisateur)

Si le module `mod_xx` comporte `module.info.java` (avec certains éléments exportés),
alors le module `mod_yy` (utilisant `mod_xx`) pourra alors éventuellement ne pas comporter de
fichier `module.info.info` et dans ce cas le comportement semble équivalent à
requires * (vérifié)
*et (à priori) exports **

NB : On appel **module automatique** un *.jar* ne comportement pas de fichier `module-info.class` et pourtant **placé** dans un des répertoires du `--module-path` .

Un tel module se voit alors attribué un nom automatiquement basé sur le nom du *.jar* et quelques ajustements (ex : `libs-legacy/xxx-yyy-legacy-1.0-SNAPSHOT.jar` est automatiquement nommé `xxx.yyy.legacy`) et un `requires xxx.yyy.legacy` est donc envisageable depuis un autre module .

Les modules automatiques voient tous leurs types publics exportés ("`exports *`" automatique). Un module utilisant un module automatique peut donc utiliser n'importe quel type public

Attention, maven et eclipse ne gèrent pas très bien les modules automatiques lors de l'exécution d'une application java.

Modules "unnamed" pour les *.jar* chargés depuis le `--class-path` plutôt que `--module-path`:

NB : On appelle **module "unnamed"** le méga module constitué de tous les **.jar** qui ne sont pas chargés via le `--module-path` mais qui sont (à l'ancienne) chargés via l'option `--class-path` ayant `-classpath` et `-cp` comme *alias* .

Cet "unnamed module" a la particularité d'avoir accès à :

- tous les packages exportés par tous les autres modules disponibles dans le module-path
- tous les jars du classpath (ie: tous les autres types présents dans cet unnamed module)

Le module par défaut (sans nom) "unnamed module" exporte tous ses packages.

Cependant les classes de cet "unnamed module" ne sont accessibles que depuis ce même "unnamed module".

NB : Par défaut, aucun "named module" ne peut accéder aux classes du "unnamed module".

(that was done on purpose to prevent modular JARs from depending on "the chaos of the class path".)

Eventuel ajout de modules (chargés via `--module-path`) dans la sphere de visibilité du "unnamed module"

Il est possible d'ajouter un fichier ".jar" (hors jdk récent mais chargé via le `--module-path`) au "unnamed / default module" via l'option de `--add-modules` de java :

Exemple :

```
java .... --add-modules java.xml.bind
```

ou bien(via maven) :

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.8.1</version>
  <configuration>
    <source>11</source><target>11</target>
    <release>11</release>
    <compilerArgs>
      <arg>--add-modules</arg>
      <arg>java.xml.bind</arg>
    </compilerArgs>
  </configuration>
</plugin>
```

Ceci est essentiellement utile pour l'ancien l'écosystème JEE .

1.14. Options (très pointues) disponibles en ligne de commande pour ré-ajuster la configuration des modules

--add-reads <module>=<target-module>(,<target-module>)*

met à jour <module> pour lire <target-module>, sans tenir compte de la déclaration de module.

<target-module> peut être ALL-UNNAMED pour lire tous les modules sans nom.

--add-exports <module>/<package>=<target-module>(,<target-module>)*

met à jour <module> pour exporter <package> vers <target-module>, sans tenir compte de la déclaration de module. <target-module> peut être ALL-UNNAMED pour effectuer un export vers tous les modules sans nom.

--add-opens <module>/<package>=<target-module>(,<target-module>)*

met à jour <module> pour ouvrir <package> vers <target-module>, sans tenir compte de la déclaration de module

--patch-module <module>=<file>(;<file>)*

Remplacement ou augmentation d'un module avec des classes et des ressources dans des fichiers ou des répertoires JAR.

VI - ForkJoin, Concurrent, CompletableFuture, ...

1. Concurrent api

1.1. interface Runnable (java.lang)

Depuis java 8 , l'interface fondamentale Runnable est devenue une interface (uni-)fonctionnelle.

```
public interface Runnable {  
    public void run();  
}
```

```
Runnable r = () -> { /*code de la tâche*/ }
```

1.2. interface Callable<T> (java.util.concurrent)

```
public interface Callable<T> {  
    public T call();  
}
```

```
Callable<T> c = () -> { /*corps de la tâche*/ return t; }
```

Callable<T> existe depuis le jdk 1.5 et est depuis java 1.8 vue comme une interface (uni-)fonctionnelle .

1.3. interface Future<T> (java.util.concurrent)

Disponible depuis le jdk 1.5, un objet **Future<T>** en **java** correspond partiellement à la notion de **Promise** en *javascript* .

Un objet technique de ce type (*recupéré immédiatement lors d'un lancement de traitement long*) **permettra de récupérer un résultat en différé** (dans le futur) .

boolean isDone()	teste la terminaison du thread pour savoir si la donnée résultante de son exécution est disponible. Retourne true si ce thread s'est bien terminé ou s'il a malheureusement levé une exception pendant son exécution, ou enfin s'il a été suspendu.
T get()	retourne le résultat (instance de T) de la tâche exécutée par le thread. Si le thread n'a pas terminé son exécution, alors l'appel de cette méthode est bloqué en attente active jusqu'à ce qu'il termine.
boolean cancel(true)	Demande à arrêter la tâche si elle n'est pas finie , le paramètre d'entrée "mayInterruptIfRunning" est souvent fixé à true . La valeur de retour (souvent ignorée) est à true si tâche interrompue ou false si tâche déjà terminée .
boolean isCancelled()	renvoi true si tâche "interrompue" avant la fin .
T get(long timeout, TimeUnit unit)	variante avec timeout de la méthode .get() Si par exemple après un timeout de 1500 TimeUnit.MILLISECONDS la tâche n'est toujours pas finie , cette méthode remonte une exception de type TimeoutException à rattraper via un try/catch pour nous signaler que cette tâche n'est pas terminée .

1.4. ExecutorService

Au coeur du package `java.util.concurrent` l'interface **ExecutorService** comporte les principales méthodes suivantes :

<code>void execute(Runnable command)</code>	lance l'exécution (via un thread créé ou disponible) d'une tâche de type <code>Runnable</code> , ne retourne rien.
<code>Future<T> submit(Callable<T> task)</code>	lance l'exécution d'une tâche de type <code>Callable<T></code> , retourne un objet de type <code>Future<T></code> permettant de récupérer ultérieurement (en différé) un résultat de type <code>T</code> .
<code>Future<?> submit(Runnable task)</code>	Comme <code>execute()</code> mais retournant <code>Future<?></code> pour attente du résultat ou de la fin
<code>T invokeAny(Collection<? extends Callable<T>> tasks)</code> ou bien <code>T invokeAny(... tasks, long timeout, ...)</code>	Lance (via des threads) une liste de tâches et attend <u>en mode bloquant</u> la première réponse , les autres tâches moins rapides sont automatiquement annulées/stoppées.
<code>List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks)</code> <i>existe en version avec timeout</i>	lance <u>en mode bloquant</u> plein de tâches et récupère une liste de résultats encapsulés dans des <code>Future<T></code> quand tout est prêt/fini .
Autres méthodes	<code>shutdown()</code> , <code>awaitTermination(timeout,...)</code> , ...

Une instance d'une classe implémentant l'interface `ExecutorService` pourra être créée via

```
ExecutorService executor = Executors.newSingleThreadExecutor();
```

ou bien

```
ExecutorService executor = Executors.newFixedThreadPool(3/*nThreads*/)
```

ou bien

```
ExecutorService executor = Executors.newCachedThreadPool();
```

ou bien d'autres façon encore .

<code>newSingleThreadExecutor()</code> ;	Un seul nouveau thread pouvant lancer plusieurs tâches alors exécutées séquentiellement les unes après les autres .
<code>newFixedThreadPool(3/*nThreads*/)</code>	via un pool de threads (en //) de taille maxi à paramétrer (si plus de tâches à exécuter que de threads dispos --> attente automatique via queue) . chaque thread ne sera arrêté que si appel explicite à <code>.shutdown()</code> .
<code>newCachedThreadPool()</code>;	via un pool de threads dont la taille est automatiquement ajustée en fonction des besoins (à la hausse ou à la baisse si rien à faire durant 60s)
autres	voir javadoc Executors

1.5. Exemple "EssaiExecutors"

MyRunnableCode.java

```
package tp.langage.thread;

public class MyRunnableCode implements Runnable {
    private String prefix;

    public MyRunnableCode() {super(); this.prefix = ""; }
    public MyRunnableCode(String prefix) { super(); this.prefix = prefix; }

    @Override
    public void run() {
        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {}
        System.out.println(prefix + Thread.currentThread().getName());
    }
}
```

EssaiExecutors.java

```
package tp.langage.thread;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

//ExecutorService depuis jdk 1.5
public class EssaiExecutors {

    // Un "ExecutorService" (à fabriquer via Executors.new....Executor()) démarre automatiquement des
    // Threads (rangés dans des "pools") pour exécuter des instances de Callable<T> ou de Runnable
    public static void main(String[] args) {

        MyRunnableCode myRunnableCode = new MyRunnableCode("");

        ExecutorService singleThreadExecutor = Executors.newSingleThreadExecutor();
                                                    //exécution séquentielle en background

        singleThreadExecutor.submit(myRunnableCode);
        singleThreadExecutor.submit(myRunnableCode);
        singleThreadExecutor.submit(myRunnableCode);
        singleThreadExecutor.shutdown();//automatiquement différé
    }
}
```

```

MyRunnableCode myRunnableCode2 = new MyRunnableCode("#");

ExecutorService multiThreadExecutor = Executors.newFixedThreadPool(3);
//exécutions multiples (en //) en background
multiThreadExecutor.submit(myRunnableCode2);
multiThreadExecutor.submit(myRunnableCode2);
multiThreadExecutor.submit(myRunnableCode2);
multiThreadExecutor.shutdown();//automatiquement différé , .shutdownNow() pour arrêt immédiat

MyRunnableCode myRunnableCode3 = new MyRunnableCode("@");

ScheduledExecutorService scheduleExecutor =
    Executors.newSingleThreadScheduledExecutor();
System.out.println("Lancement d'un thread/tâche (@) en différé (2000ms)");
scheduleExecutor.schedule(myRunnableCode3, 2000, TimeUnit.MILLISECONDS);
scheduleExecutor.shutdown();//automatiquement différé , .shutdownNow() pour arrêt immédiat
}
}

```

Résultats :

```

Lancement d'un thread (@) en différé (2000ms)
#pool-2-thread-3
#pool-2-thread-2
*pool-1-thread-1
#pool-2-thread-1
*pool-1-thread-1
*pool-1-thread-1
@pool-3-thread-1

```

1.6. Exemple "TestFuture" (avec Executors et Callable<T>)

LongTask.java

```
package tp.langage.thread;

public class LongTask {

    public static void printThread() {
        System.out.println(Thread.currentThread().getName());
    }

    public static void simulateLongTask(String msg, long nbMs) {
        try {
            System.out.println(">>(begin)" + msg + " / by " + Thread.currentThread().getName());
            Thread.sleep(nbMs);
            System.out.println("<<(end)" + msg + " / by " + Thread.currentThread().getName());
        } catch (InterruptedException e) {
            //e.printStackTrace();
            System.out.println("*** interrupted ***");
        }
    }
}
```

CallableComputing.java

```
package tp.langage.thread;
import java.util.concurrent.Callable;

public class CallableComputing implements Callable<String> {
    private double x;

    @Override
    public String call() throws Exception {
        LongTask.simulateLongTask("long computing task (in background) ...", 5000);
        return String.valueOf(Math.sqrt(x));
    }

    public CallableComputing() {super(); this.x = 0; }
    public CallableComputing(double x) {super(); this.x = x; }

    public double getX() { return x; }
    public void setX(double x) {this.x = x; }
}
```

EssaiFuture.java

```

package tp.langage.thread;

import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class EssaisFuture {

    public static void main(String[] args) {
        //NB: Callable<T>/call() ressemble un peu à l'interface Running/run()
        //mais permet de récupérer (ultérieurement) un résultat via Future<T> .
        Callable<String> c = new CallableComputing(9);
        String result=null;

        ExecutorService executor = Executors.newSingleThreadExecutor();

        Future<String> futureRes = executor.submit(c);
        while(result==null){
            LongTask.simulateLongTask("other works ...",2000);
            if(futureRes.isDone()){
                try {
                    result = futureRes.get();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                } catch (ExecutionException e) {
                    e.printStackTrace();
                }
            }
        }
        System.out.println("result=" + result);

        System.out.println("-----");
        Future<String> futureRes2 = executor.submit(c);
        LongTask.simulateLongTask("other works ...",2000);
        if(futureRes2.isDone()){
            try {
                result = futureRes2.get();
                System.out.println("result2=" + result);
            } catch (InterruptedException e) {
                e.printStackTrace();
            } catch (ExecutionException e) {
                e.printStackTrace();
            }
        }
    }
    else {

```



```

    futureRes2.cancel(true);
    if(futureRes2.isCancelled()){
        System.out.println("background computing was cancelled");
    }
}
System.out.println("-----");
Future<String> futureRes3 = executor.submit(c);
LongTask.simulateLongTask("other works ...",2000);
try {
    result = futureRes3.get(1500,TimeUnit.MILLISECONDS);
    System.out.println("result3=" + result);
} catch (InterruptedException e) {
    e.printStackTrace();
} catch (ExecutionException e) {
    e.printStackTrace();
} catch (TimeoutException e) {
    System.err.println("tâche 3 toujours pas terminée au bout de 1500ms");
    System.out.println("after 1500ms,futureRes3.isDone()="+futureRes3.isDone());
    System.out.println("after 1500ms,futureRes3.isCancelled()="+futureRes3.isCancelled());
    //e.printStackTrace(); }
}
    executor.shutdown();//automatiquement différé , .shutdownNow() pour arrêt immédiat
}

```

Résultats :

```

>>(begin)other works ... / by main
>>(begin)long computing task (in background) ... / by pool-1-thread-1
<<(end)other works ... / by main
>>(begin)other works ... / by main
<<(end)other works ... / by main
>>(begin)other works ... / by main
<<(end)long computing task (in background) ... / by pool-1-thread-1
<<(end)other works ... / by main
result=3.0
-----
>>(begin)other works ... / by main
>>(begin)long computing task (in background) ... / by pool-1-thread-1
<<(end)other works ... / by main
** interrupted **
background computing was cancelled
-----
>>(begin)other works ... / by main
>>(begin)long computing task (in background) ... / by pool-1-thread-1
<<(end)other works ... / by main
after 1500ms,futureRes3.isDone()==false
after 1500ms,futureRes3.isCancelled()==false
tâche 3 toujours pas terminée au bout de 1500ms
<<(end)long computing task (in background) ... / by pool-1-thread-1

```

1.7. Semaphore (Synchronisation de Threads)

Un **Semaphore** est (en java depuis le jdk 1.5) un objet technique de synchronisation entre différents threads .

Un **sémaphore** correspond conceptuellement à un **ensemble de jetons disponible** .

Un thread doit **acquérir un jeton disponible** (via un **appel bloquant** à semaphore.acquire() ou bien semaphore.tryAcquire(timeout...) *pour pouvoir ensuite travailler seul sur une ressource partagée* .

En appelant la méthode symétrique semaphore.release() , un thread peut rendre un jeton dans le semaphore (souvent après avoir terminé un certain travail en mode "exclusivité") . Cette action va immédiatement débloquer d'éventuelles attentes exprimées via semaphore.acquire() .

Plus précisément, un **sémaphore** encapsule un entier, avec une contrainte de positivité, et deux opérations atomiques d'incrément et de décrémentation :

- via le constructeur : variable entière (toujours positive ou nulle) ;
- opération -- (**acquire()**) : décrémente le compteur s'il est strictement positif ; bloque s'il est nul en attendant de pouvoir le décrémentation ;
- opération ++ (**release()**) : incrémente le compteur.

NB : Depuis java 1.5 , les **Semaphores** constituent un mécanisme de synchronisation **plus souple et plus fiable** que le mécanisme .wait()/notify() disponible dès java 1.0 sur la classe Object .

EssaiSemaphore.java

```
package tp.langage.thread;
import java.util.concurrent.Semaphore;
import java.util.concurrent.TimeUnit;

public class EssaiSemaphore {
    public static void main(String[] args) {
        Semaphore semaphore = new Semaphore(0);

        Thread t = new Thread(()-> { LongTask.simulateLongTask("background thread work ...", 3000);
                                semaphore.release()});

        t.start();
        System.out.println("... faire autre chose ...");

        //attendre la disponibilité du sémaphore:
        try {
            //semaphore.acquire();
            if(semaphore.tryAcquire(5, TimeUnit.MINUTES)){
                System.out.println("semaphore acquis");
            }else{
                System.out.println("after 5mn (semaphore toujours pas disponible)");
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

}

Résultats :

```

... faire autre chose ...
>>(begin)background thread work ... / by Thread-0
<<(end)background thread work ... / by Thread-0
semaphore acquis

```

La notion de "**Mutex**" (mutuelle exclusion) peut s'implémenter en java avec **un sémaphore à un seul jeton** .

1.8. CompletableFuture (attendre et consommer résultats produits)

EssaiCompletableFuture.java

```

package tp.langage.thread;

import java.util.concurrent.Callable;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutorCompletionService;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

//CompletableFuture depuis jdk1.6 avec méthode take() retournant un Future
//logique producteur/consommateur

public class EssaiCompletableFuture {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(4); //jusqu'à 4 threads en //
        CompletableFuture<String> completionService =
            new ExecutorCompletionService<String>(executor);

        double[] tabVal = { 4, 9, 16, 25, 36, 49, 64, 81, 100 };
        int taille = tabVal.length;
        for(int i=0;i<taille;i++){
            Callable<String> c = new CallableComputing(tabVal[i]);
            //CompletableFuture<String> encapsule l'executor et est typé comme Future<T>
            completionService.submit(c); //lancement asynchrone d'un "producteur"
        }
        for(int i=0;i<taille;i++){
            try {
                Future<String> futureRes = completionService.take(); //attente du PREMIER TERMINE
                System.out.println(futureRes.get()); //consommateur simple
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

```

```

    }
    }
    System.out.println("fin-main");
    //arrêter l'executor (si besoin en différé) :
    executor.shutdown();
}
}

```

Résultats :

```

>>(begin)long computing task (in background) ... / by pool-1-thread-2
>>(begin)long computing task (in background) ... / by pool-1-thread-3
>>(begin)long computing task (in background) ... / by pool-1-thread-4
>>(begin)long computing task (in background) ... / by pool-1-thread-1
<<(end)long computing task (in background) ... / by pool-1-thread-2
<<(end)long computing task (in background) ... / by pool-1-thread-4
<<(end)long computing task (in background) ... / by pool-1-thread-3
<<(end)long computing task (in background) ... / by pool-1-thread-1
>>(begin)long computing task (in background) ... / by pool-1-thread-3
3.0
4.0
2.0
>>(begin)long computing task (in background) ... / by pool-1-thread-1
>>(begin)long computing task (in background) ... / by pool-1-thread-2
>>(begin)long computing task (in background) ... / by pool-1-thread-4
5.0
<<(end)long computing task (in background) ... / by pool-1-thread-3
<<(end)long computing task (in background) ... / by pool-1-thread-1
<<(end)long computing task (in background) ... / by pool-1-thread-2
>>(begin)long computing task (in background) ... / by pool-1-thread-1
6.0
<<(end)long computing task (in background) ... / by pool-1-thread-4
8.0
7.0
9.0
<<(end)long computing task (in background) ... / by pool-1-thread-1
10.0
fin-main

```

2. ForkJoin

2.1. basic fork/join

```
public class EssaiBasicForkJoin {
    public static void main(String[] args) {
        System.out.println("debut - main");
        Thread t = new Thread(new MyRunnableCode());
        t.start(); // basic sort of fork()
        System.out.println(("suite - main / avant join"));
        try {
            t.join(); //attente de la fin de l'exécution du thread
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("fin main - apres join");
    }
}
```

-->

```
debut - main
suite - main / avant join
Thread-0
fin main - apres join
```

2.2. fork/join (depuis java 1.7)

Le framework **fork / join** en Java (depuis le jdk 1.7) est idéal pour **un problème qui peut être divisé en parties plus petites et résolu en parallèle**.

Les étapes fondamentales d'un problème fork / join sont les suivantes:

- **Diviser le problème en plusieurs morceaux**
- **Résoudre chacune des pièces en parallèle**
- **Combinez chacune des sous-solutions en une solution globale**

Une [ForkJoinTask](#) est l'interface qui définit un tel problème. On s'attend généralement à ce que vous sous-classiez l'une de ses implémentations abstraites (généralement la [RecursiveTask](#)) plutôt que d'implémenter l'interface directement.

Détail technique :

Le *ForkJoinPool* est le coeur du framework. Il s'agit d'une implémentation de [ExecutorService](#) qui gère les threads de travail et nous fournit des outils pour obtenir des informations sur l'état et les performances du pool de threads.

Les threads de travail ne peuvent exécuter qu'une tâche à la fois, mais *ForkJoinPool* ne crée pas de thread séparé pour chaque sous-tâche. Au lieu de cela, chaque thread du pool a sa propre file d'attente à deux extrémités (double ended *queue[deque]*) qui stocke les tâches.

Cette architecture est essentielle pour équilibrer la charge de travail du thread à l'aide de l'algorithme "work-stealing".

2.3. fork/join appliqué sur un quickSort en mode "multi-processeurs"

MyQuickSortAlgo.java (version sans fork/join)

```
package tp.langage.thread;
public class MyQuickSortAlgo {

    static void echanger(double[] tableau ,int indice1 ,int indice2){
        double temp = tableau[indice1];
        tableau[indice1] = tableau[indice2];
        tableau[indice2] = temp;
    }

    static int partition(double[] tableau,int deb,int fin){
        int indicePivot=deb; //au sens indice initial qui va évoluer
        double valeurPivot=tableau[deb]; //valeur du pivot (= arbitrairement valeur en première position du tableau)
        //via une future permutation , cette valeur sera à une future autre position

        for(int i=deb+1;i<=fin;i++){
            if (tableau[i]<valeurPivot){
                indicePivot++; //nouvelle valeur pour le futur indice du pivot (qui peut encore évoluer selon boucle en cours)
                echanger(tableau,indicePivot,i); //pour placer à la "future gauche" de l'indice provisoire du pivot
                //tous les éléments plus petits que le pivot
            }
        }
        echanger(tableau,deb,indicePivot); //permutation pour que la valeur du pivot soit rangée à sa place (précédemment calculée)
        //et pour qu'un des éléments plus petits soit placé au début (à gauche )

        return indicePivot;
    }

    //version ordinaire (sans optimisation multi-proc):
    static void tri_rapide(double[] tableau,int deb,int fin){
        if(deb<fin){
            //partitionner le tableau en 2 parties partiellement ré-arrangées .
            //d'un coté tous les éléments plus petits que le pivot , de l'autre coté tous les éléments plus grands:
            int positionPivot=partition(tableau,deb,fin);
            tri_rapide(tableau,deb,positionPivot-1); //trier le sous tableau des plus petits éléments que le pivot
            tri_rapide(tableau,positionPivot+1,fin); //trier le sous tableau des plus grands éléments que le pivot
        }
    }

    static void quick_sort(double[] tableau){
        tri_rapide(tableau, 0, tableau.length - 1 );
    }
}
```

MyQuickSortMultiProc.java (avec fork/join)

```
package tp.langage.thread;

import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveAction;
```

//ForkJoin (RecursiveAction) / divide & conquer since jdk 1.7

```
public class MyQuickSortMultiProc extends RecursiveAction {

    private static final long serialVersionUID = 1L;
    private static final int FORK_JOIN_MIN_SIZE=1024;

    private double[] tab;
    private int start,end;

    public MyQuickSortMultiProc(double[] tableau,int deb,int fin){
        this.tab = tableau;
        this.start = deb;
        this.end = fin;
    }

    static void quick_sort_multiProc(double[] tableau){
        MyQuickSortMultiProc myQuickSortMultiProc=
            new MyQuickSortMultiProc(tableau, 0, tableau.length - 1);
        ForkJoinPool threadPool = new ForkJoinPool();
        threadPool.invoke(myQuickSortMultiProc);
    }

    @Override //RecursiveAction
    protected void compute() {
        // pas de paramètre et donc les tab et indices
        // doivent être renseignés en tant qu'attributs + constructeurs
        MyQuickSortMultiProc sousTriGaucheViaForkJoin=null;
        MyQuickSortMultiProc sousTriDroitViaForkJoin = null;
        //System.out.println("MyQuickSortMultiProc.compute() executé par "+Thread.currentThread().getName() );

        if(start<end){
            //partitionner le tableau en 2 parties partiellement ré-arrangées .
            //d'un coté tous les éléments plus petits que le pivot , de l'autre coté tous les éléments plus grands:
            int positionPivot=MyQuickSortAlgo.partition(tab,start,end);

            //NB: étant donné que la version forkJoin ajoute une complexité au niveau du code
            // (instance à créer , thread à gérer) , cette version ne sera activée/utilisée
            //que pour trier des sous tableaux dont la taille minimum est supérieure à
            //FORK_JOIN_MIN_SIZE=1024

            if(positionPivot - start > FORK_JOIN_MIN_SIZE){
                sousTriGaucheViaForkJoin= new MyQuickSortMultiProc(tab, start, positionPivot-1);
                sousTriGaucheViaForkJoin.fork(); //déléguer le tri du sous tableau des plus petits éléments que le pivot
            } else MyQuickSortAlgo.tri_rapide(tab, start, positionPivot-1);

            if(end - positionPivot > FORK_JOIN_MIN_SIZE){
                sousTriDroitViaForkJoin= new MyQuickSortMultiProc(tab, positionPivot+1,end);
                sousTriDroitViaForkJoin.fork(); //déléguer le tri du tableau des plus grands éléments que le pivot
            } else MyQuickSortAlgo.tri_rapide(tab, positionPivot+1, end);
        }
    }
}
```

```

if(sousTriGaucheViaForkJoin!=null) sousTriGaucheViaForkJoin.join(); //attendre
if(sousTriDroitViaForkJoin!=null) sousTriDroitViaForkJoin.join(); //attendre

//NB: il existe invokeAll(recursiveAction1 , recursiveAction2) qui declenche en // .fork() et .join()
    }
}
}

```

TestMultiProcesseurs.java (tests avec rapidités comparées)

```

package tp.langage.thread;

public class TestMultiProcesseurs {
    public static void main(String[] args) {
        System.out.println("nb processors:" + Runtime.getRuntime().availableProcessors());
        double[] t1 = produce_init_tab();
        double[] copyOfT1 = t1.clone();
        display_tab(t1); //display_tab(copyOfT1);
        System.out.println("tri ordinaire (quick-sort) ");
        test_tri(t1);
        System.out.println("tri (quick-sort) optimisé pour machine multi-processeurs ");
        test_tri_multiProc(copyOfT1);
    }

    static double[] produce_init_tab() {
        //double[] t = { 5,2,1,9,3,4,12,8,16,6 };
        //double[] t = { 26,7,5,2,1,9,3,4,34,12,8,16,6,78,10,89,33,23,90,123,72,3,48 };
        //final int taille=10;
        final int taille=1024*1024*8;
        double[] t = new double[taille];
        for(int i=0;i<taille;i++){
            t[i]=Math.random()*taille;
        }
        return t;
    }

    static void display_tab(double[] tab){

```



```

        if(tab.length <= 30) {
            for(double x : tab)
                System.out.print(x + " ");
            System.out.print("\n");
        } else{ System.out.println("tableau de taille = " + tab.length);
        }
    }

    static void test_tri(double[] tab){
        long td = System.nanoTime();
        MyQuickSortAlgo.quick_sort(tab);
        long tf = System.nanoTime();    display_tab(tab);
        System.out.println("## " + (tf-td)/ 1000000 + " ms");
    }

    static void test_tri_multiProc(double[] tab){
        long td = System.nanoTime();
        MyQuickSortMultiProc.quick_sort_multiProc(tab);
        long tf = System.nanoTime();    display_tab(tab);
        System.out.println("** " + (tf-td) / 1000000 + " ms");
    }
}

```

Résultats (avec i7):

```

nb processors:4
tableau de taille = 8388608
## tri ordinaire (quick-sort)
tableau de taille = 8388608
## 872 ms
** tri (quick-sort) optimisé pour machine multi-processeurs
tableau de taille = 8388608
** 538 ms

```

Autre exemple de fork/join (calcul de moyenne ou d'écartType).

SequentialComputing.java

```
package tp.thread.sam;
//interface d'une référence de fonction pour calcul ordinaire de somme ou moyenne ou ...
public interface SequentialComputing {
    //start et end sont les indices sur la plage du tableau à manipuler .
    //arg = null ou éventuel argument nécessaire à un calcul (ex: arg=moyenne pour calcul de variance)
    double basicCompute(double[] numbers, int start, int end, Double arg); //sum or average or ....
}
```

ResultAggregate.java

```
package tp.thread.sam;
//interface d'une référence de fonction pour recombinaison 2 sous sommes ou 2 sous moyennes
public interface ResultAggregate {
    Double composeTotalRes(double res_tab1, int taille_tab1, double res_tab2, int taille_tab2);
}
```

MyStatsAlgo.java

```
package tp.thread;

import tp.thread.sam.ResultAggregate;
import tp.thread.sam.SequentialComputing;

//calcul de somme , moyenne ou variance sur grands tableaux
public class MyStatsAlgo {
    public static final int DECOMP_MIN_SIZE=1024*50;

    public static final Double composeMoyenneTotale(double moyenne_tab1, int taille_tab1,
                                                    double moyenne_tab2, int taille_tab2) {
        return (moyenne_tab1*taille_tab1 + moyenne_tab2*taille_tab2) / (taille_tab1 + taille_tab2);
    }

    //versions ordinaires (sans decomposition en 2 sous parties):

    static Double moyenne_ordinaire_subPart(double[] tableau, int deb, int fin, Double notUsedArg){
        Double somme = 0.0;
        for(int i=deb; i<=fin ;i++) {
            somme += tableau[i];
        }
        int sizeSubPart = (fin - deb)+1;
        return somme / sizeSubPart;
    }

    static Double variance_ordinaire_subPart(double[] tableau, int deb, int fin, Double moyenne){
```

```

    Double varianceFoisN = 0.0;
    for(int i=deb; i<=fin ;i++) {
        varianceFoisN += (tableau[i] - moyenne) * (tableau[i] - moyenne);
    }
    int sizeSubPart = (fin - deb)+1;
    return varianceFoisN/sizeSubPart;
}

```

//versions avec decomposition en 2 sous parties:

```

static double moyenne_decomp_subPart(double[] tableau,int deb,int fin){
    return compute_decomp_subPart(tableau,deb,fin,null,
        MyStatsAlgo::moyenne_ordinaire_subPart,MyStatsAlgo::composeMoyenneTotale);
}

```

```

static double variance_decomp_subPart(double[] tableau,int deb,int fin,double moyenne){
    return compute_decomp_subPart(tableau,deb,fin,moyenne,
        MyStatsAlgo::variance_ordinaire_subPart,MyStatsAlgo::composeMoyenneTotale);
}

```

```

static double compute_decomp_subPart(double[] tableau,int deb,int fin,Double arg,
    SequentialComputing seqComputing , ResultAggregate resAggregate){
    double resSp;
    int sizeSubPart = (fin - deb)+1;
    if(sizeSubPart >= DECOMP_MIN_SIZE){
        int indiceMilieu = deb + (sizeSubPart / 2);
        double resCalculSousPartie1 = compute_decomp_subPart(tableau,deb,indiceMilieu-1,arg,seqComputing,resAggregate);
        double resCalculSousPartie2 = compute_decomp_subPart(tableau,indiceMilieu,fin,arg,seqComputing,resAggregate);
        resSp = resAggregate.composeTotalRes(resCalculSousPartie1 , (indiceMilieu - deb) ,
            resCalculSousPartie2 , (fin - indiceMilieu +1) );
    }
    else
        resSp = seqComputing.basicCompute(tableau,deb,fin,arg);
    return resSp;
}

```

//fonctions de niveau principal (appels simples , niveau global):

```

static double moyenne_ordinaire(double[] tableau){
    return moyenne_ordinaire_subPart(tableau, 0, tableau.length - 1 , null );
}

```

```

static double ecartType_ordinaire(double[] tableau){
    double moyenne = moyenne_ordinaire_subPart(tableau, 0, tableau.length - 1 , null );
    double variance = variance_ordinaire_subPart(tableau, 0, tableau.length - 1 , moyenne );
    return Math.sqrt(variance);
}

```

```

static double moyenne_decomp(double[] tableau){
    return moyenne_decomp_subPart(tableau, 0, tableau.length - 1 );
}

```

```

static double ecartType_decomp(double[] tableau){
    double moyenne = moyenne_decomp_subPart(tableau, 0, tableau.length - 1 );
    double variance = variance_decomp_subPart(tableau, 0, tableau.length - 1 , moyenne);
    return Math.sqrt(variance);
}

```

MyStatsMultiProc.java

```

package tp.thread;

import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;

import tp.thread.sam.ResultAggregate;
import tp.thread.sam.SequentialComputing;

public class MyStatsMultiProc extends RecursiveTask<Double> {
    private static final int FORK_JOIN_MIN_SIZE=1024*50;
        // THRESHOLD/SEUIL à éventuellement ajuster selon complexité du calcul

    private double[] tab;
    private int start,end;
    private SequentialComputing seqComputing; //référence de fonction pour calcul ordinaire de somme ou moyenne ou ...
    private ResultAggregate resAggregate; //référence de fonction pour recombinaison 2 sous sommes ou 2 sous moyennes
    private Double arg; //null ou éventuel argument nécessaire à un calcul (ex: arg=moyenne pour calcul de variance)

    public MyStatsMultiProc(double[] tableau,int deb,int fin , Double arg,
        SequentialComputing seqComputing , ResultAggregate resAggregate){
        this.tab = tableau; this.start = deb; this.end = fin; this.arg = arg;
        this.seqComputing = seqComputing; this.resAggregate = resAggregate;
    }

    static double moyenne_multiProc(double[] tableau){
        MyStatsMultiProc myStatsMultiProc= new MyStatsMultiProc(tableau, 0, tableau.length - 1,
            null,MyStatsAlgo::moyenne_ordinaire_subPart,MyStatsAlgo::composeMoyenneTotale);
        ForkJoinPool threadPool = new ForkJoinPool();
        return threadPool.invoke(myStatsMultiProc);
    }

    static double ecartType_multiProc(double[] tableau){
        double moyenne = moyenne_multiProc(tableau);
        MyStatsMultiProc myStatsMultiProc= new MyStatsMultiProc(tableau, 0, tableau.length - 1,
            moyenne,MyStatsAlgo::variance_ordinaire_subPart,MyStatsAlgo::composeMoyenneTotale);
        ForkJoinPool threadPool = new ForkJoinPool();
        double variance = threadPool.invoke(myStatsMultiProc);
        return Math.sqrt(variance);
    }

    @Override
    protected Double compute() {
        Double res =0.0;
        //System.out.println("MyStatsMultiProc.compute() executé par "+Thread.currentThread().getName() );

        //NB: etant donné que la version forkJoin ajoute une complexité au niveau du code
        // (instance à créer , thread à gérer) , cette version ne sera activée/utilisée
        //que pour traiter des sous tableaux dont la taille minimum est supérieure à FORK_JOIN_MIN_SIZE

        int sizeSubPart = (end - start)+1;

        if(sizeSubPart >= FORK_JOIN_MIN_SIZE) {
            int indiceMilieu = this.start + (sizeSubPart / 2);
            // pas de parametre dans .compute() et donc les tab et indices doivent être renseignés
            //en tant qu'attributs + constructeurs
            MyStatsMultiProc sousCalculGaucheViaForkJoin=
                new MyStatsMultiProc(tab,start,indiceMilieu-1,arg,seqComputing,resAggregate);

```

```
MyStatsMultiProc sousCalculDroitViaForkJoin=  
    new MyStatsMultiProc(tab,indiceMilieu,end,arg,seqComputing,resAggregate);  
  
sousCalculGaucheViaForkJoin.fork();//déléguer (via potentiel autre thread)  
  
//sous solution A (.compute() ) :  
//Double resCalculSousPartie2 = sousCalculDroitViaForkJoin.compute();//faire soit même (via meme thread)  
  
//sous solution B (.fork/join ) :  
sousCalculDroitViaForkJoin.fork();//déléguer (via potentiel autre thread)  
Double resCalculSousPartie2 = sousCalculDroitViaForkJoin.join(); //attendre  
  
Double resCalculSousPartie1 = sousCalculGaucheViaForkJoin.join(); //attendre  
  
res = resAggregate.composeTotalRes(resCalculSousPartie1 , (indiceMilieu - start) ,  
                                   resCalculSousPartie2 , (end - indiceMilieu +1) );  
}else {  
    res = seqComputing.basicCompute(this.tab,this.start,this.end,this.arg);  
}  
return res;  
}
```

Exemple de performances comparées (sur un petit i7) :

nb processors:4

tableau de taille = 134217728

moyenne ordinaire (sans decomposition)

##MOYENNE=500.0102357180581

601 ms

\$\$ moyenne avec decomposition

\$\$MOYENNE=500.01023571807764

\$\$ **174 ms**

** moyenne avec decomposition optimise pour machine multi-processeurs (fork/join)

**MOYENNE=500.01023571807764

** **64 ms**

3. CompletableFuture , streams asynchrones

CompletableFuture<T> disponible depuis le *jdk 1.8* est une version améliorée de **Future<T>** pour **enchaîner certains traitements asynchrones** et qui comporte *quelques similitudes avec les "callback" et "Promise.then()" de javascript* .

```
public class CompletableFuture<T>
extends Object
implements Future<T>, CompletionStage<T>
```

dans le package **java.util.concurrent** .

Dans les grandes lignes , la **principale valeur ajoutée de CompletableFuture<T>** vis à vis de **Future<T>** réside dans l'**implémentation de l'interface CompletionStage<T>** de manière à **bien resynchroniser / ordonner différentes tâches asynchrones qui seront par nature exécutées en différé** (dans le futur) .

Autrement dit , un bloc d'instructions de ce type :

```
CompletableFuture.supplyAsync(traitementAsynchrone1 )
    .thenApply( traitementAsynchrone2 )
    .thenApply(traitementAsynchrone3)
    .thenAccept( traitementAsynchroneFinal );
```

correspond à enregistrement de tâches asynchrones à effectuer en différé dès que possible .

Le future résultat du *traitementAsynchrone1* constituera l'entrée de la tâche *traitementAsynchrone2* qui ne pourra alors être démarrée que lorsque la tâche *traitementAsynchrone1* sera terminée et ainsi de suite via cet enchaînement de `.thenApply()` `.then...()`

Rappel :

- l'interface (uni-)fonctionnelle **Supplier<T>** comporte l'unique méthode **T .get()** et correspond à un producteur ou fournisseur de valeur de Type *T* .
- l'interface (uni-)fonctionnelle **Function<T,R>** comporte la méthode **R .apply(T t)** et correspond à une fonction appliquée à un unique paramètre de type *T* et retournant *R* .
- l'interface (uni-)fonctionnelle **Consumer<T>** comporte la méthode **void .accept(T t)** et correspond à un consommateur d'élément de type *T* (sans valeur produite) .

Principales méthodes de CompletableFuture<T> et CompletionStage<T> :

NB1 : l'abréviation **CF<T>** est à comprendre comme CompletableFuture<T>

NB2 : Sans précision de l'executor (souvent en tant que dernier paramètre facultatif des méthodes surchargées) , la classe CompletableFuture lance en interne les tâches asynchrones via **ForkJoinPool.commonPool()** par défaut .

NB3 : Au sein de ce tableau , les méthodes en *italiques* seront à considérées comme "**static**" .
et <T,R> est généralement à interpréter comme < ? super T , ? extends R>

CF<Void> runAsync (Runnable r)	retourne un CF<T> correspondant à la future exécution asynchrone de la tâche r (Runnable)
CF<U> <i>supplyAsync</i> (Supplier<U> s)	retourne un CF<U> correspondant à la future exécution asynchrone de la tâche s (fournissant une valeur de type U)
CF<R> <i>thenApply</i> (Function<T,R> f)	Souvent en milieu d'enchaînement, après la fin du CF<T> précédent (vu comme préfixe (this)) , retourne un CF<R> correspondant à la future exécution asynchrone de la tâche fonctionnelle f
CF<R> thenCompose (Function<T,CF<R>> f)	Un peu comme thenApply() mais en précisant une fonction qui (immédiatement, sans attente) construit elle même une instance de CF <R> qui sera consommée dans la suite de l'enchaînement
CF<T> <i>exceptionally</i> (Function<Throwable ex,T> f)	Souvent en milieu d'enchaînement, transforme une exception potentielle en élément de type T pour le bon déroulement de la suite des enchaînements, retransmet la valeur inchangée si pas d'exception .
CF<R> <i>handle</i> (BiFunction<T, Throwable,R> f)	Combine le comportement de thenApply() et exceptionally() via une tâche asynchrone prenant 2 argument en entrée : value et exception .
CF<Void> <i>thenAccept</i> (Consumer<T> c)	En terminaison, après la fin du CF<T> précédent (vu comme préfixe (this)) , retourne un CF<Void> correspondant à la future exécution asynchrone de la tâche terminale c (Consumer avec argument en entrée)
CF<Void> thenRun (Runnable r)	Souvent en terminaison, après la fin du CF<T> précédent (vu comme préfixe (this)) , retourne un CF<Void> correspondant à la future exécution asynchrone de la tâche terminale r (Runnable sans argument en entrée)
boolean complete (T value) et boolean completeExceptionally (Throwable ex)	si tâche pas terminée , fixe le résultat de cette tâche qui sera ultérieurement récupérée par .get().

Remarque : si cf.*thenApplyAsync*(...); à la place de cf.*thenApply*(...); alors tâche quelquefois exécutée par encore un autre thread.

3.1. Premiers exemples de CompletableFuture

EssaiAsyncJava8.java

```
package tp.langage.thread;

import java.util.concurrent.CompletableFuture; import java.util.function.Supplier;

public class EssaiAsyncJava8 {
    public static void main(String[] args) {
        System.out.println("debut main / interpreted by " + Thread.currentThread().getName());

        //initialisation asynchrone:
        //CompletableFuture<Void> cf = CompletableFuture.runAsync(aRunnableObject); //with no return result !!!
        CompletableFuture<Double> completableFuture1 =
            /*CompletableFuture.supplyAsync( new Supplier<Double>(){
                public Double get(){ LongTask.simulateLongTask("long computing - p1" , 2000);
                return 2.0; }
            });*/
        CompletableFuture.supplyAsync( ()-> { LongTask.simulateLongTask("long ...- p1" , 2000);
            /*throw new RuntimeException("exceptionXY");*/ return 2.0; } );

        //En cas d'exception en asynchrone/tâche de fond:
        CompletableFuture<Double> safecompletableFuture1 =
            completableFuture1.exceptionally(ex -> { System.out.println("problem: " +
                ex.getMessage()); return 0.0; } );
        /*CompletableFuture<Double> safecompletableFuture1 =
            completableFuture1.handle((resOk,ex) -> {
                if(resOk!=null) return resOk;
                else { System.out.println("problem: " + ex.getMessage()); return 0.0; } } );*/

        System.out.println("suite main A / interpreted by " + Thread.currentThread().getName());

        // continuations asynchrones (avec Function<T1,T2>):
        CompletableFuture<Double> completableFuture2 =
            safecompletableFuture1.thenApply((x) -> { LongTask.simulateLongTask("long computing -
                p2" , 2000); return x*x; } );

        CompletableFuture<String> completableFuture3 =
            completableFuture2.thenApply((x) -> { LongTask.simulateLongTask("long computing - p3" ,
                2000); return String.valueOf(x); } );

        System.out.println("suite main B / interpreted by " + Thread.currentThread().getName());
        //fin/terminaison asynchrone:
        completableFuture3.thenAccept((x) -> System.out.println(x) );
    }
}
```



```
//completableFuture.thenRun(()->System.out.println("ok")); //with no input !!!

LongTask.simulateLongTask("pause pour eviter arrêt complet du programme " +
    " avant la fin des taches de fond" , 8000);
System.out.println("fin main / interpreted by " + Thread.currentThread().getName());
}
}
```

Résultats :

```
debut main / interpreted by main
suite main A / interpreted by main
>>(begin)long computing - p1 / by ForkJoinPool.commonPool-worker-3
suite main B / interpreted by main
>>(begin)pause pour eviter arrêt complet du programme avant la fin des taches de fond / by main
<<(end)long computing - p1 / by ForkJoinPool.commonPool-worker-3
>>(begin)long computing - p2 / by ForkJoinPool.commonPool-worker-3
<<(end)long computing - p2 / by ForkJoinPool.commonPool-worker-3
>>(begin)long computing - p3 / by ForkJoinPool.commonPool-worker-3
<<(end)long computing - p3 / by ForkJoinPool.commonPool-worker-3
4.0
<<(end)pause pour eviter arrêt complet du programme avant la fin des taches de fond / by main
fin main / interpreted by main
```

Variation syntaxique (générant le même résultat) :

EssaiAsyncJava8V2.java

```
package tp.langage.thread;

import java.util.concurrent.CompletableFuture;

public class EssaiAsyncJava8V2 {

    public static Double extractInitValue() {
        LongTask.simulateLongTask("long computing - p1" , 2000);
        /*throw new RuntimeException("exceptionXY");*/ return 2.0;
    }

    public static Double auCarre(Double x){
        LongTask.simulateLongTask("long computing - p2" , 2000); return x*x;
    }

    public static String convertAsString(Double x){
        LongTask.simulateLongTask("long computing - p3" , 2000); return String.valueOf(x);
    }
}
```

```

public static void displayString(String s){
    System.out.println(s) ;
}

public static void main(String[] args) {
    System.out.println("debut main / interpreted by " + Thread.currentThread().getName());

    CompletableFuture.supplyAsync(EssaiAsyncJava8V2::extractInitValue )
        .exceptionally(ex -> { System.out.println("problem: " +
            ex.getMessage()); return 0.0; } )
        .thenApply(EssaiAsyncJava8V2::auCarre )
        .thenApply(EssaiAsyncJava8V2::convertAsString)
        .thenAccept(EssaiAsyncJava8V2::displayString );

    System.out.println("suite main / interpreted by " + Thread.currentThread().getName());
    LongTask.simulateLongTask("pause pour eviter arrêt complet du programme" +
        " avant la fin des taches de fond" , 8000);
    System.out.println("fin main / interpreted by " + Thread.currentThread().getName());
}
}

```

Autre variation syntaxique avec this et lambda expressions :

EssaiAsyncJava8V2Bis.java

```

package tp.langage.thread;
import java.util.concurrent.CompletableFuture;
public class EssaiAsyncJava8V2Bis {
    public static void simuLong(String number,long nbMs) {
        LongTask.simulateLongTask("long computing - p"+number , nbMs);
    }
    private double initialXValue;
    private String result;

    public void noStaticMethodWithLambda() {
        this.initialXValue=2;
        //code am"lirable avec synchronized(this){ this... }
        CompletableFuture.supplyAsync(()->{ simuLong("p1",2000);
            return this.initialXValue; } )
            .thenApply( (x)->{ simuLong("p2",2000); return x*x; } )
            .thenApply( (x)->{ simuLong("p3",2000); return String.valueOf(x); } )
            .thenAccept( (s)->{System.out.println(s); this.result=s});
        LongTask.simulateLongTask("pause pour eviter arrêt complet du programme"
            + " avant la fin des taches de fond" , 8000);
    }
}

```

```

        System.out.println("result="+this.result);
    }

    public static void main(String[] args) {
        EssaiAsyncJava8V2Bis thisApp = new EssaiAsyncJava8V2Bis();
        thisApp.noStaticMethodWithLambda();
    }
}

```

result=4

3.2. Combinaisons avec CompletableFuture

Exemple partiel : *EssaiAsyncJava8V3.java*

```

package tp.langage.thread;

import java.util.concurrent.CompletableFuture;
import java.util.concurrent.CompletionStage;

public class EssaiAsyncJava8V3 {

    //plein de code commun avec EssaiAsyncJava8V2 (pas répété ici)

    public static Double plus(Double x, Double y){
        LongTask.simulateLongTask("long computing - plus" , 2000); return x+y;
    }

    public static CompletionStage<Double> cflnitVal() {
        return CompletableFuture.supplyAsync(EssaiAsyncJava8V3::extractInitValue );
    }

    //méthode static pour pour then.compose(...) :
    public static CompletionStage<String> cfAsString(Double x){
        CompletableFuture<Double> cfDouble = new CompletableFuture<Double>();
        cfDouble.complete(x);
        System.out.println("**** tout le debut de cfAsString sans attente ****");
        return cfDouble.thenApplyAsync(EssaiAsyncJava8V3::convertAsString);
    }

    public static void main(String[] args) {
        System.out.println("debut main de EssaiAsyncJava8V3");
        //thenCompose register another future to apply/compose to thisFuture (without waiting)
        // to produce a new future :
        cflnitVal().thenCompose(EssaiAsyncJava8V3::cfAsString)
            .thenAccept(EssaiAsyncJava8V3::displayString );

        CompletableFuture<Double> cfD1 = new CompletableFuture<Double>();
    }
}

```

```

cfD1.complete(3.0); //COMPLETE BY MAIN --> NEED .thenApplyAsync() to be continued by other thread

CompletableFuture<Double> cfD2 = new CompletableFuture<Double>();
cfD2.complete(2.0); //COMPLETE BY MAIN --> NEED .thenApplyAsync() to be continued by other thread
System.out.println("suite du main de EssaiAsyncJava8V3");
CompletableFuture<Double> cfD3 = cfD1.thenApplyAsync(EssaiAsyncJava8V3::auCarre);//3*3=9
CompletableFuture<Double> cfD4 = cfD2.thenApplyAsync(EssaiAsyncJava8V3::auCarre);//2*2=4

//thenCombine apply a biFunction from 2 futures (this & other) to produce a new future

CompletableFuture<Double> cfD5 =
    cfD3.thenCombine(cfD4, EssaiAsyncJava8V3::plus);//9+4=13
cfD5.thenAccept((x)->System.out.println(x));

CompletableFuture<Double> cfD6 = cfD1.thenApplyAsync(EssaiAsyncJava8V3::auCarre);//3*3=9
CompletableFuture<Double> cfD7 = cfD2.thenApplyAsync(EssaiAsyncJava8V3::auCarre);//2*2=4
cfD6.thenAcceptBoth(cfD7, (x,y) -> System.out.println(x*y));//9*4 = 36

LongTask.simulateLongTask("pause pour éviter arrêt complet du programme "
    + "avant la fin des taches de fond", 8000);
System.out.println("fin main de EssaiAsyncJava8V3 / interpreted by " + Thread.currentThread().getName());
}

```

Autres combinaisons :

Variantes proches de thenCombine :

.thenAcceptBoth(otherFuture, biConsumer) --> action terminale (sans valeur calculée ni retournée) avec valeurs en entrées lorsque les 2 futures sont terminés

.runAfterBoth(otherFuture, runnable) --> action sans valeur en entrée lorsque les 2 futures sont terminés

Variantes proches de thenAcceptBoth/runAfterBoth :

.thenAcceptEither(otherFuture, consumer) --> action avec valeur en entrées lorsque le premier des 2 futures est terminé

.runAfterEither(otherFuture, runnable) --> action sans valeur en entrée lorsque le premier des 2 futures est terminé

Variante proche de thenAcceptEither :

thenApplyEither(otherFuture , function) --> génère un nouveau future (avec valeur) pour poursuivre un enchaînement avec thenApply() ou autre.

Variantes avec nombres d'arguments variables:

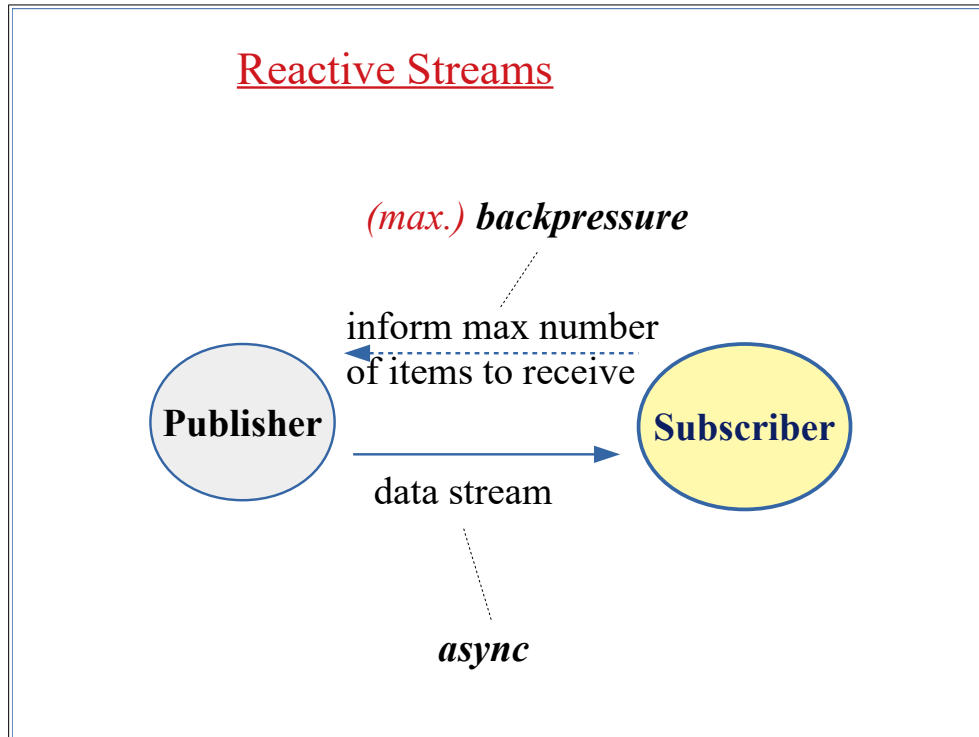
.allOf(...) pour attendre la fin de n(3 ou plus) futures

.anyOf(...) pour attendre la fin du plus rapide parmi n

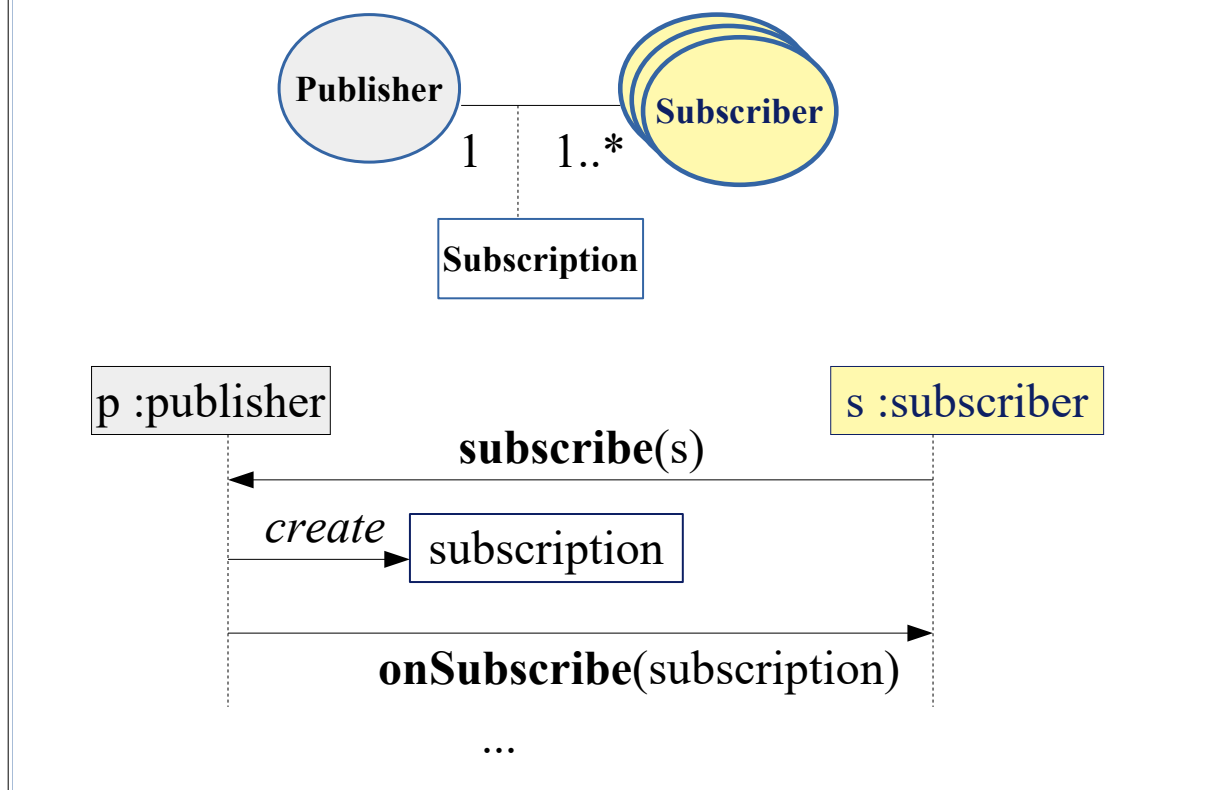
VII - Reactive-Streams (depuis java 9)

1. Reactive Streams (depuis java 9)

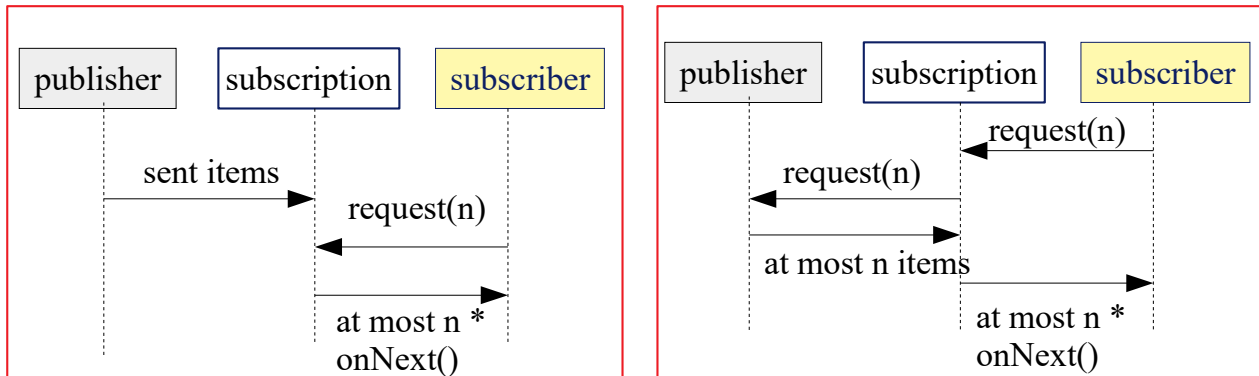
1.1. Concepts des "reactive streams"



Reactive Streams



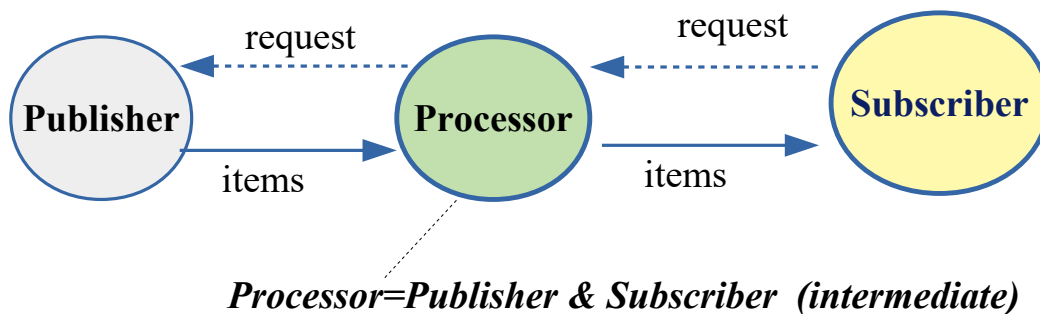
Reactive Streams



queue approach/scenario

Or

generate onDemand scenario



1.2. Abstractions standards du langage java (Flow)

La pseudo classe **Flow** du package **java.util.concurrent** ne fait que regrouper un ensemble d'interfaces complémentaires :

```
public final class Flow {
    @FunctionalInterface
    public static interface Publisher<T> {
        public void subscribe(Subscriber<? super T> subscriber);
    }
    public static interface Subscriber<T> {
        public void onSubscribe(Subscription subscription); //notification of subscription
        public void onNext(T item); //data/item reception/notification
        public void onError(Throwable throwable); //exception reception/notification
        public void onComplete(); //end of stream notification (no more item to receive)
    }
}
```



```

public static interface Subscription {
    public void request(long n); //ask to receive at most n more items , may be call several times
    public void cancel();
}

public static interface Processor<T,R> extends Subscriber<T>, Publisher<R> {}

```

--> utilisation avec implémentations de type

*class MySubscriber<T> implements **Flow.Subscriber**<T> { ... } , etc*

1.3. Mise en oeuvre élémentaire via **SubmissionPublisher**

NB : La classe `java.util.concurrent.SubmissionPublisher<T>` est une implémentation prédéfinie simple de `Flow.Publisher<T>` qui comporte une méthode `.submit()` à généralement appeler plusieurs fois pour envoyer des données aux abonnés (*subscribers*) .

Exemple :

MySimplePrintSubscriber.java

```

package tp.langage.flow;

import java.util.concurrent.Flow;
import java.util.concurrent.Flow.Subscription;

public class MySimplePrintSubscriber implements Flow.Subscriber<Object> {
    private Subscription subscription;
    private String subscriberName;
    private int nbTotalReceived = 0;
    private int nbTotalRequest = 0;
    private static final int NB_OTHER_ITEMS=4;

    public MySimplePrintSubscriber() {
        this.subscriberName="mySimplePrintSubscriber";
    }
    public MySimplePrintSubscriber(String subscriberName){
        this.subscriberName=subscriberName;
    }

    private void requestSomeItems() {
        subscription.request(NB_OTHER_ITEMS);
        this.nbTotalRequest+=NB_OTHER_ITEMS;
    }

    @Override
    public void onSubscribe(Subscription subscription) {

```

```

    this.subscription = subscription;
    System.out.println(this.subscriberName + ">> Received subscription notification");
    requestSomeItems();
}

@Override
public void onNext(Object item) {
    this.nbTotalReceived++;
    System.out.println(this.subscriberName + ">> Received item: " + item.toString() + " thread:"
        + Thread.currentThread().getName() + " nbTotalRequest="
        + this.nbTotalRequest + " nbTotalReceived=" + this.nbTotalReceived);
    if(nbTotalReceived==nbTotalRequest) {
        requestSomeItems();
    }
}

@Override
public void onError(Throwable error) {
    System.out.println(this.subscriberName + ">> Error occurred: " + error.getMessage());
}

@Override
public void onComplete() {
    System.out.println(this.subscriberName + ">> complete");
}
}

```

MySubmissionPublisherApp.java

```

package tp.langage.flow;

import java.util.concurrent.SubmissionPublisher;

public class MySubmissionPublisherApp {

    public static void pause(long nbMs) {
        try { Thread.sleep(nbMs); } catch (InterruptedException e) { e.printStackTrace(); }
    }

    public static void main(String[] args) {
        sansProcesseur();
    }

    public static void sansProcesseur() {
        SubmissionPublisher<Object> publisher = new SubmissionPublisher<>();
        publisher.subscribe(new MySimplePrintSubscriber("s1"));
        System.out.println("Submitting first items...");
        for (int i = 0; i < 10; i++) {
            publisher.submit(i); //onNext() may be called on subscriber(s) if request(...)
            if(i==5) {
                pause(1000);
            }
        }
    }
}

```

```

        publisher.subscribe(new MySimplePrintSubscriber("s2"));
        System.out.println("Submitting others items...");
    }
}
pause(1000);
publisher.close(); //onComplete() will be called on subscriber(s)
}
}

```

Résultats :

```

Submitting first items...
s1>> Received subscription notification thread:ForkJoinPool.commonPool-worker-3
** debut de pause de 1000 ms , thread:main
s1>> Received item: 0 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=4 nbTotalReceived=1
s1>> Received item: 1 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=4 nbTotalReceived=2
s1>> Received item: 2 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=4 nbTotalReceived=3
s1>> Received item: 3 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=4 nbTotalReceived=4
s1>> Received item: 4 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=8 nbTotalReceived=5
s1>> Received item: 5 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=8 nbTotalReceived=6
** fin de pause de 1000 ms , thread:main
Submitting others items...
s2>> Received subscription notification thread:ForkJoinPool.commonPool-worker-3
** debut de pause de 1000 ms , thread:main
s2>> Received item: 6 thread:ForkJoinPool.commonPool-worker-5 nbTotalRequest=4 nbTotalReceived=1
s1>> Received item: 6 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=8 nbTotalReceived=7
s2>> Received item: 7 thread:ForkJoinPool.commonPool-worker-5 nbTotalRequest=4 nbTotalReceived=2
s1>> Received item: 7 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=8 nbTotalReceived=8
s2>> Received item: 8 thread:ForkJoinPool.commonPool-worker-5 nbTotalRequest=4 nbTotalReceived=3
s1>> Received item: 8 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=12 nbTotalReceived=9
s2>> Received item: 9 thread:ForkJoinPool.commonPool-worker-5 nbTotalRequest=4 nbTotalReceived=4
s1>> Received item: 9 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=12 nbTotalReceived=10
** fin de pause de 1000 ms , thread:main
s2>> complete thread:ForkJoinPool.commonPool-worker-5
s1>> complete thread:ForkJoinPool.commonPool-worker-3

```

AuCarreProcessor.java

```

package tp.langage.flow;
import java.util.concurrent.Flow.Subscriber; import java.util.concurrent.Flow.Subscription;
import java.util.concurrent.SubmissionPublisher;

public class AuCarreProcessor extends SubmissionPublisher<Object>
    implements Subscriber<Object> {
    private Subscription subscription;

    @Override
    public void onSubscribe(Subscription subscription) {
        this.subscription = subscription;
        subscription.request(1);
    }

    @Override
    public void onNext(Object item) {
        double val = Double.parseDouble(item.toString());
        submit(String.valueOf(val*val));
        subscription.request(1);
    }
}

```

```

}

@Override
public void onError(Throwable error) {
    error.printStackTrace();
    closeExceptionally(error);
}

@Override
public void onComplete() {
    System.out.println("AuCarreProcessor completed");
    close();
}
}

```

MySubmissionPublisherApp.java (V2)

```

package tp.langage.flow;

import java.util.concurrent.SubmissionPublisher;

public class MySubmissionPublisherApp {

    //.... pause() et sansProcessor()

    public static void main(String[] args) {
        //sansProcesseur();
        avecProcesseur();
    }

    public static void avecProcesseur() {
        //publisher <--> processor <--> subscriber
        AuCarreProcessor processor = new AuCarreProcessor();
        processor.subscribe(new MySimplePrintSubscriber("s1"));
        SubmissionPublisher<Object> publisher = new SubmissionPublisher<>();
        publisher.subscribe(processor);
        System.out.println("Submitting items...");
        for (int i = 0; i < 10; i++) {
            publisher.submit(i); //onNext() may be called on subscriber(s) if request(...)
        }
        pause(1000);
        publisher.close(); //onComplete() will be called on subscriber(s)
    }
}

```

Résultats :

```

Submitting items...
s1>> Received subscription notification thread:ForkJoinPool.commonPool-worker-3
** debut de pause de 1000 ms , thread:main
s1>> Received item: 0.0 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=4 nbTotalReceived=1
s1>> Received item: 1.0 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=4 nbTotalReceived=2
s1>> Received item: 4.0 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=4 nbTotalReceived=3
s1>> Received item: 9.0 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=4 nbTotalReceived=4
s1>> Received item: 16.0 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=8 nbTotalReceived=5
s1>> Received item: 25.0 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=8 nbTotalReceived=6

```

```
s1>> Received item: 36.0 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=8 nbTotalReceived=7
s1>> Received item: 49.0 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=8 nbTotalReceived=8
s1>> Received item: 64.0 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=12 nbTotalReceived=9
s1>> Received item: 81.0 thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=12 nbTotalReceived=10
** fin de pause de 1000 ms , thread:main
AuCarreProcessor completed
s1>> complete thread:ForkJoinPool.commonPool-worker-5
```

1.4. "Reactive Streams" avec RxJava

RxJava (proche de **RxJs**) est une des implémentations proches des "Reactive Streams" de java>=9 .

RxJava existe depuis longtemps et fait partie de la famille des technologies créées par Netflix . Il existe des versions 1.x et 2.x et 3.x .

RxJava comporte plein de similitude avec la version javascript "RxJs" dont l'abstraction "**Observable**" .

RxJava est souvent utilisé au sein de développement Android pour effectuer des appels Http en asynchrone (un peu comme RxJs au sein d'une application "Angular") .

Certaines versions de RxJava peuvent être utilisées avec le jdk 1.6 .

La version 3.x nécessite au moins java 1.8 .

URL officielle de "RxJava" : <https://github.com/ReactiveX/RxJava>

1.5. "ReactiveStreams" avec Reactor

Reactor est une autre implémentation proche des "Reactive Streams" de java>=9 .

La classe reactor.adapter.JdkFlowAdapter comporte des méthodes .publisherToFlowPublisher() et .flowPublisherToFlux() permettant de convertir des **Flux** de **Reactor** en Flow java9 et vice versa .

Le projet "Reactor" est en partie géré par la communauté "Spring" et est intégré dans le framework "Spring 5" avec les "WebFlux" .

Reactor est un projet plus récent que RxJava . Il nécessite au minimum le jdk 1.8 (comme Spring5)

URL officielle de Reactor : <https://projectreactor.io>

VIII - JShell (RPEL java)

1. JShell (RPEL java)

1.1. Principe RPEL

REPL (**R**ead-**E**val-**P**rint **L**oop)

Un langage interprété (ex : "lisp" , "basic" ou "javascript") peut (à un haut niveau , quelque fois proche d'une interaction utilisateur) effectuer une boucle dite "REPL" de ce type :

```
loop({
- read expression (ex : read "2*4+5" in variable expr )
- eval result (ex : res=eval(expr) = 13)
- print result (ex : console.log("res="+res) ; )
})
```

vision fonctionnelle :

```
(loop (print (eval (read()))))
```

```
expression: 2+3
result=5
expression: 4+4
result=8
expression: 2*6+1
result=13
expression: fin
end
```

1.2. Présentation de JShell java

Depuis java9 , JShell permet d'interpréter et exécuter du code à la volée (sans devoir absolument écrire du code dans une classe java) .

(exemple : C:\Program Files\Java\jdk-11.0.4\bin\jshell.exe)

```
Ctrl Invite de commandes - jshell
Microsoft Windows [version 10.0.18362.900]
(c) 2019 Microsoft Corporation. Tous droits réservés.

C:\Users\didier>jshell
| Welcome to JShell -- Version 11.0.4
| For an introduction type: /help intro

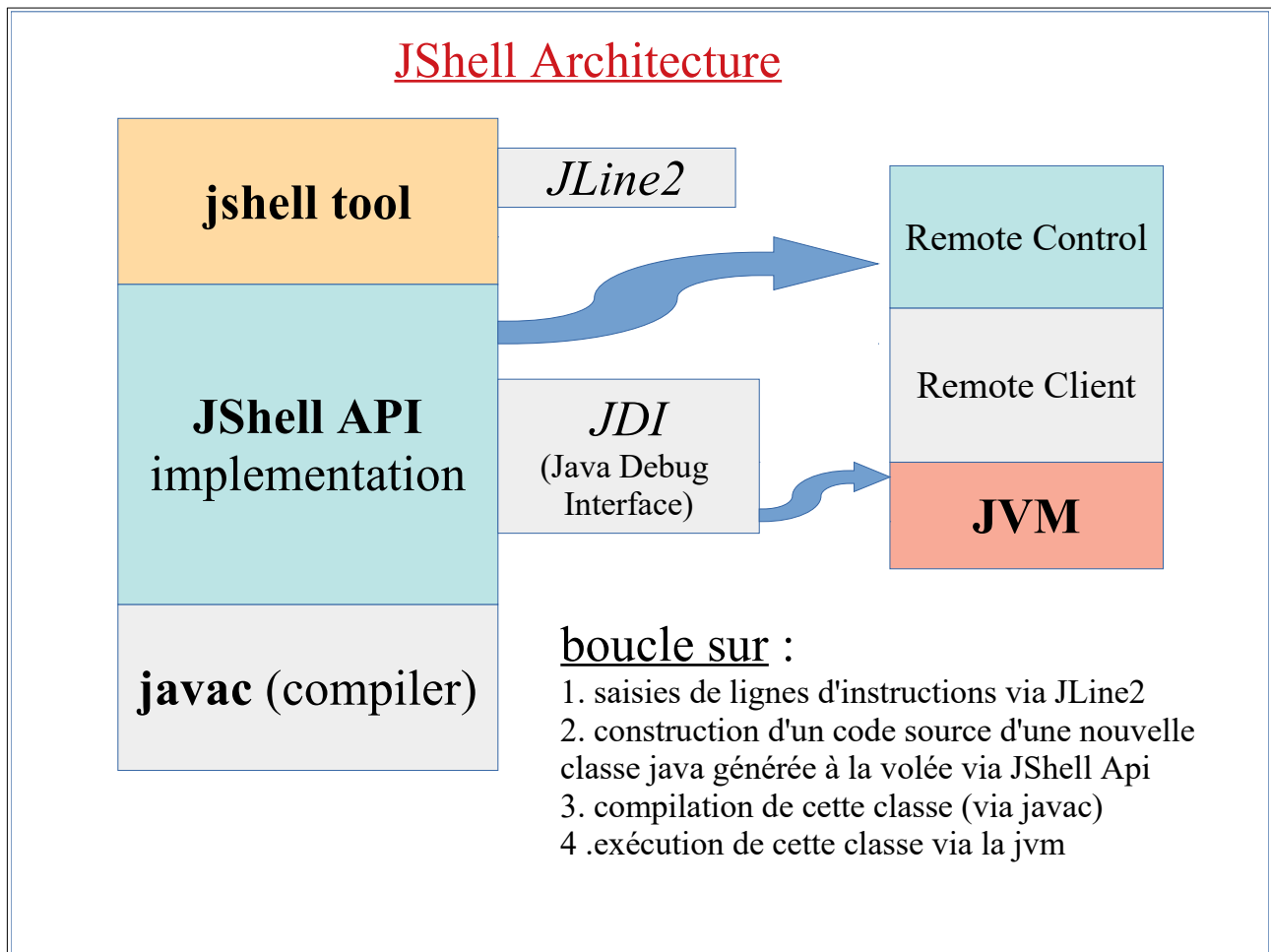
jshell> 2+2
$1 ==> 4

jshell> Math.sqrt(16)
$2 ==> 4.0

jshell> "fic1"+".txt"
$3 ==> "fic1.txt"

jshell>
```

1.3. Fonctionnement interne de JShell



1.4. Utilisation de JShell (exemples)

```
jshell> System.out.println("ok");  
ok
```

```
jshell> 1+  
...> 3  
$2 ==> 4
```

```
jshell> for(int i=0;i<3;i++){  
...> System.out.println("i="+i);  
...> }  
i=0  
i=1  
i=2
```

```
jshell> 2+2  
$4 ==> 4
```

```
jshell> $4*3  
$5 ==> 12
```

```
jshell> "abc"+"def"  
$6 ==> "abcdef"  
jshell> $6.toUpperCase()  
$7 ==> "ABCDEF"
```

```
jshell> double x=5.5  
x ==> 5.5  
  
jshell> double y=6.6;  
y ==> 6.6  
  
jshell> x+y  
$10 ==> 12.1
```

```
jshell> /vars  
| int $2 = 4  
| int $4 = 4  
| int $5 = 12  
| String $6 = "abcdef"  
| String $7 = "ABCDEF"  
| double x = 5.5  
| double y = 6.6  
| double $10 = 12.1
```

/vars affiche la liste des variables .

```
jshell> public int add(int a,int b){
...> return a+b;
...> }
| created method add(int,int)

jshell> add(5,6)
$12 ==> 11
```

```
jshell> /imports
| import java.io.*
| import java.math.*
| import java.net.*
| import java.nio.file.*
| import java.util.*
| import java.util.concurrent.*
| import java.util.function.*
| import java.util.prefs.*
| import java.util.regex.*
| import java.util.stream.*
```

```
jshell> class User{
...> String nom;
...> public User(){ nom="default name"; }
...> public void sayHello(){ System.out.println("Hello, my name is " + nom); }
...> }
| created class User

jshell> User u1=new User();
u1 ==> User@6e1567f1

jshell> u1.nom="toto"
$15 ==> "toto"

jshell> u1.sayHello()
Hello, my name is toto
```

```
jshell> /types
| class User
```

1.5. Pour approfondir le sujet jshell

https://blog.soat.fr/2018/03/java9_decouverte_jshell_1

...

IX - Process api , Http2 api , ...

1. Process Api

1.1. Lancement d'un processus depuis une application java

```
public static void startNodePad() {
    ProcessBuilder builder = new ProcessBuilder("notepad.exe"); //de java.lang depuis java 1.5
    try {
        Process process = builder.start();
        System.out.println("pid of process=" + process.pid());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

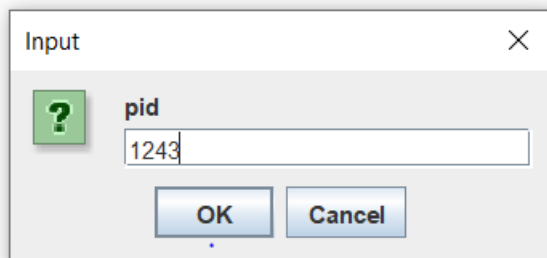
1.2. L'interface ProcessHandle (java.lang) depuis java 9

return type	method	fonctionnalités
static Stream < ProcessHandle >	allProcesses()	Returns a snapshot of all processes visible to the current process.
Stream < ProcessHandle >	children()	Returns a snapshot of the current direct children of the process.
int	compareTo (ProcessHandle other)	Compares this ProcessHandle with the specified ProcessHandle for order.
static ProcessHandle	current()	Returns a ProcessHandle for the current process.
Stream < ProcessHandle >	descendants()	Returns a snapshot of the descendants of the process.
boolean	destroy()	Requests the process to be killed.
boolean	destroyForcibly()	Requests the process to be killed forcibly.
boolean	equals (Object other)	Returns true if other object is non-null, is of the same implementation, and represents the same system process; otherwise it returns false.
int	hashCode()	Returns a hash code value for this ProcessHandle.
ProcessHandle.Info	info()	Returns a snapshot of information about the process.
boolean	isAlive()	Tests whether the process represented by this ProcessHandle is alive.
static Optional < ProcessHandle >	of (long pid)	Returns an Optional < ProcessHandle > for an existing native process.

CompletableFuture<ProcessHandle>	onExit()	Returns a CompletableFuture<ProcessHandle> for the termination of the process.
Optional<ProcessHandle>	parent()	Returns an Optional<ProcessHandle> for the parent process.
long	pid()	Returns the native process ID of the process.
boolean	supportsNormalTermination()	Returns true if the implementation of destroy() normally terminates the process.

1.3. Exemples de gestion de processus

```
import java.io.IOException;
import java.util.Optional;
import javax.swing.JOptionPane;
....
public static void destroyProcessById() {
    long pid=Long.parseLong(JOptionPane.showInputDialog(null, "pid"));
    Optional<ProcessHandle> optionalProcessHandle = ProcessHandle.of(pid);
    optionalProcessHandle.ifPresent(processHandle -> processHandle.destroy());
}
```



```
public static void startAndDestroyProcess() {
    //ProcessBuilder builder = new ProcessBuilder("C:\\Program Files\\Mozilla Firefox\\firefox.exe");
    ProcessBuilder builder = new ProcessBuilder("notepad.exe");
    try {
        Process process = builder.start();
        System.out.println("pid of process=" + process.pid());
        Thread.sleep(5000); //5000ms
        process.destroy();
        if (process.isAlive()) {
            System.out.println("process still alive ...");
            process.destroyForcibly(); //a utiliser que si process.destroy() ne suffit pas
        }else {
            System.out.println("process is no more alive ...");
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
}
}
```

```
...
public static String processHandleAsString(ProcessHandle ph) {
    return "pid="+ph.pid()+" - "+ph.info().toString();
}

public static void displayingAllProcess() {
    ProcessHandle.allProcesses().forEach(ph -> System.out.println(processHandleAsString(ph)));
}
-->
```

```
...
pid=21964 - [user: Optional[LAPTOP-DDC\didier], cmd: C:\Program Files (x86)\Google\Chrome\Application\chrome.exe,
startTime: Optional[2020-07-02T14:04:20.296Z], totalTime: Optional[PT0.078125S]]
pid=4840 - []
pid=5696 - [user: Optional[LAPTOP-DDC\didier], cmd: C:\Program Files\Java\jdk-11.0.4\bin\javaw.exe, startTime:
Optional[2020-07-02T14:47:38.376Z], totalTime: Optional[PT0.34375S]]
```

```
package org.mycontrib.tp;
public class MySubProcess {
    public static void main(String[] args) {
        System.out.println("MySubProcess ...");
        try { Thread.sleep(5000); /*5000ms*/ } catch (InterruptedException e) { e.printStackTrace(); }
        System.exit(0);
    }
}
```

```
public static void startAndWaitingProcess() {
    ProcessBuilder builder = new ProcessBuilder();
    /*
    final String javaHome = "C:\\Program Files\\Java\\jdk-11.0.4"; // " " need around directory name with space
    final String javaCmdeWithQuote = "\"" + javaHome + "\\bin\\java.exe" + "\"";
    //final String cmde = javaCmdeWithQuote + " -version";
    final String cmde = javaCmdeWithQuote + "-cp .\\target\\myProcessApp.jar org.mycontrib.tp.MySubProcess";
    System.out.println("cmde="+cmde);
    //builder.command("cmd.exe", "/c", cmde); // ok, "/c" = terminate after this run
    */
    //NB: .\\target\\myProcessApp.jar will be ready after mvn install or ...
    builder.command("java", "-cp", ".\\target\\myProcessApp.jar",
        "org.mycontrib.tp.MySubProcess"); //ok with java in PATH
    boolean isStopped = false;
    try {
        Process process = builder.start();
        System.out.println("pid of process=" + process.pid());
        BufferedReader outputReaderOfSubProcess = new BufferedReader(new
            InputStreamReader(process.getInputStream()));
        System.out.println("output of process (first line) =" +
            outputReaderOfSubProcess.readLine());
        isStopped=process.waitFor(3, TimeUnit.SECONDS);
        if(isStopped) {
```

```
        System.out.println("process is terminated");
    } else {
        System.out.println("process not terminated after 3s");
        isStopped=process.waitFor(3, TimeUnit.SECONDS);//re-wait
    }
    if (isStopped) {
        System.out.println("process is stopped with exit value="
            +process.exitValue());
    }
} catch (Exception e) {
    e.printStackTrace();
}
}
```

2. Http2 Client

2.1. Préambule sur le protocole HTTP/2 :

La version **1.1** du protocole Http date de **1997**.

La version 2 du protocole Http a été standardisée en **2015** (en s'inspirant du protocole expérimental SPDY de google commencé en 2009)

La plupart des navigateurs web sont aujourd'hui compatible avec **http/2** .

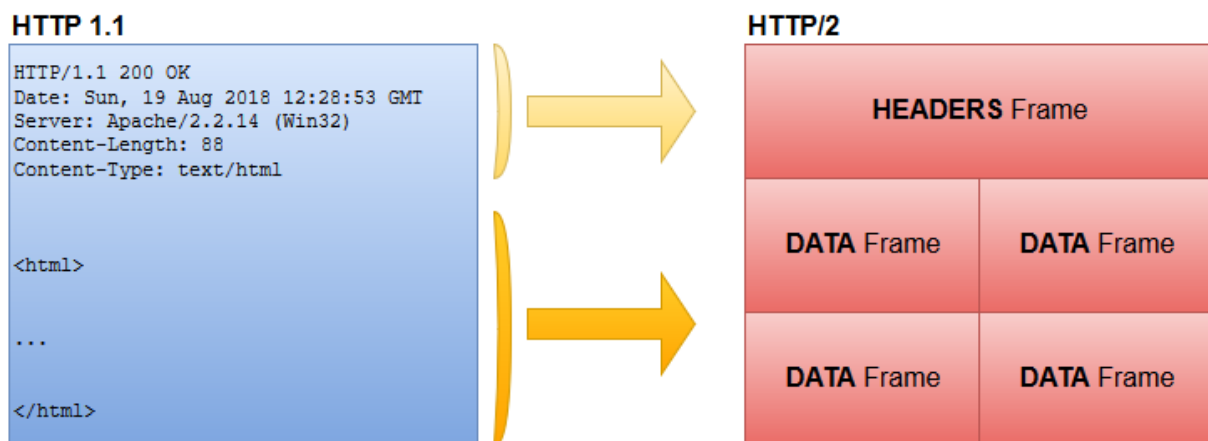
HTTP/2 conserve une rétrocompatibilité complète avec HTTP 1.1 . Mêmes syntaxes .

http/1 échange la plupart des données en mode texte.

http/2 échange (en interne) les données en mode binaire (même si du côté "client" et "serveur" une vision "mode texte" est toujours de mise pour les types MIMES "text/html" , ...)

Une connexion http/1 ne pouvait récupérer qu'une seule ressource à la fois . Malgré la réutilisation de connexion (keep-alive) , et l'éventuelle parallélisation des connexions tcp/ip vers un même domaine (limitées à 6) , un chargement d'un grand nombre de ressources via http est plutôt séquentiel en http/1 et peut donc prendre beaucoup de temps.

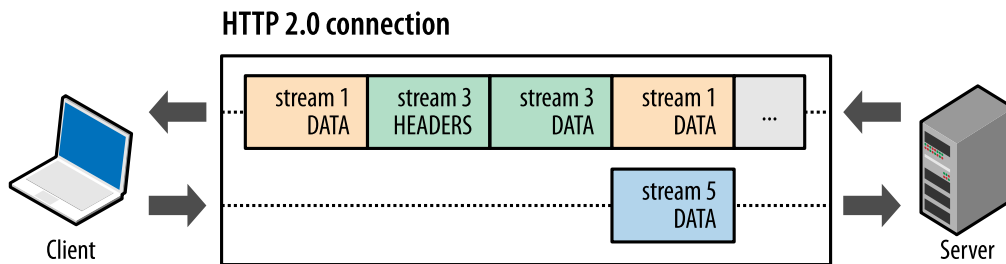
Une connexion **http/2** peut être utilisée en mode "**multiplexage de flux**" et on peut ainsi récupérer plusieurs ressources en parallèle . Ceci permet de beaucoup améliorer la vitesse de certains téléchargement .



En interne, le protocole **http/2** véhicule des données dans des blocs binaires appelées "**Frames**" (avec parties "**header**" et "**data**") .

Plusieurs "frames" peuvent ainsi transitées (éventuellement dans les 2 sens en mode "bi-directionnel" au sein d'une même connexion http/2 . Les "frames" peuvent éventuellement être

priorisées .



NB : au sein d'une même connection HTTP2 établie, le client et le serveur peuvent échanger (dans les 2 sens) des frames (HEADERS + DATA) comme s'il s'agissait de requêtes indépendantes .

Chaque frame (HEADERS ou DATA) comporte un attribut important **.streamId** .

Ceci permet au client de comprendre qu'une frame de réponse est associée à telle frame de requête (par analyse de corrélation du stream ID) .

Le protocole **http/2** autorise également un mode **push** (où le serveur peut prendre l'initiative de renvoyer spontanément certaines pièces attachées par encore demandées) .

Le protocole **http/2** est capable d'effectuer automatiquement **une compression** (en binaire optimisé) des entêtes HTTP ce qui permet d'**optimiser la bande passante** .

Pour approfondir --> <https://oxiane.developpez.com/tutoriels/java/http2-java/>

2.2. Présentation de l'api java HttpClient

La nouvelle API **HTTP Client** (en pré-version "incubator" en v9 puis standard depuis v 11) propose **un support des versions 1.1 et 2 du protocole HTTP** ainsi que les WebSockets côté client.

Cette **api** (beaucoup plus moderne que l'ancienne classe `HttpURLConnection` datant du jdk 1.1) **peut fonctionner en mode asynchrone** et permet (si nécessaire) l'utilisation de l'API Flow (reactive streams) pour fournir les données du body d'une requête (`Flow.Publisher`) et consommer le body d'une réponse (`Flow.Subscriber`).

L'API est contenue dans le package **java.net.http** du module `java.net.http`. Elle contient plusieurs types dont les principaux sont **`HttpClient`** , **`HttpClient.Builder`** , **`HttpRequest`** , **`HttpRequest.Builder`** , **`HttpRequest.BodyPublisher`** , **`HttpRequest.BodyPublishers`** , **`HttpResponse`** , `HttpResponse.ResponseInfo` , **`HttpResponse.BodySubscriber`** , **`HttpResponse.BodySubscribers`** , `HttpResponse.BodyHandler` , `HttpResponse.BodyHandlers` .

Généralités : Un **`XxxBuilder`** sert à construire une instance immuable de type **`Xxx`**

Les **`XxxYyyys`** sont des fabriques de **`XxxYyy`**

Un "**`Handler`**" sert à décortiquer la réponse (et la récupérer dans un certain type) en s'aidant des informations d'entêtes trouvées dans `HttpResponse.ResponseInfo`

2.3. Exemples avec api HttpClient

Exemple en mode synchrone :

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.net.http.HttpResponse.BodyHandlers;
...
public static void test_new_http2_client_since_java9_standard_since_java11() {
    HttpClient client = HttpClient.newHttpClient();

    HttpRequest req =
        HttpRequest.newBuilder(URI.create("http://www.google.com"))
            .header("User-Agent", "Java")
            .GET()
            .build();

    try {
        HttpResponse<String> resp = client.send(req, BodyHandlers.ofString());
        System.out.println("reponse status:" + resp.statusCode()); //200 for ok
        System.out.println("reponse uri:" + resp.uri().toString()); //http://www.google.com
        System.out.println("reponse type:" + resp.headers().map().get("Content-Type"));
            //--> [text/html; charset=ISO-8859-1]
        System.out.println("reponse text:" + resp.body()); //texte html
    } catch (IOException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

```
HttpClient client = HttpClient.newHttpClient(); //Http2 par défaut si possible
```

ou bien

```
HttpClient httpClient = HttpClient.newBuilder()
    .version(HttpClient.Version.HTTP_1_1)
    /.proxy(ProxySelector.of(new InetSocketAddress("proxy.xyz.com", 80)))
    .connectTimeout(Duration.ofSeconds(10))
    .build();
```

Préparation d'une requête à envoyer en mode POST :

```
HttpRequest requetePost = HttpRequest.newBuilder()
```

```
.uri(URI.create("https://www.xyz.com/api-xyz"))
.setHeader("Content-Type", "application/json")
.POST(BodyPublishers.ofString(dataAsJsonString))
.build();
```

Quelques Variantes :

```
.POST(HttpRequest.BodyProcessor.fromByteArray(sampleData))
```

ou encore

```
.POST(HttpRequest.BodyProcessor.fromFile(
    Paths.get("src/test/resources/sample.json")))
```

Variante du premier exemple en **mode asynchrone** :

```
..
//En mode asynchrone , sendAsync retournant un CompletableFuture :
client.sendAsync(req, BodyHandlers.ofString())
    .thenAccept(resp -> {
        System.out.println("recuperation réponse asynchrone / interpreted by "
            + Thread.currentThread().getName());
        System.out.println("reponse status:" + resp.statusCode());
        System.out.println("reponse uri:" + resp.uri().toString());
        System.out.println("reponse type:" + resp.headers().map().get("Content-Type"));
        System.out.println("reponse text size:" + resp.body().length());
    });

System.out.println("suite synchrone interpreted by " + Thread.currentThread().getName());
try { Thread.sleep(2000); //pause ici pour eviter arrêt complet du programme
    // avant la fin des taches de fond asynchrones
} catch (InterruptedException e) { e.printStackTrace(); }
System.out.println("fin synchrone / interpreted by " + Thread.currentThread().getName());
```

Résultats :

```
suite synchrone interpreted by main
recuperation réponse asynchrone / interpreted by ForkJoinPool.commonPool-worker-3
reponse status:200
reponse uri:http://www.google.com
reponse type:[text/html; charset=ISO-8859-1]
reponse text size:12791
fin synchrone / interpreted by main
```

Avec subscriber/reactive stream (pour gérer réponse) :

```

public static void test_new_httpClient_withSubscriber() {
    HttpClient client = HttpClient.newHttpClient();
    HttpRequest req =
        HttpRequest.newBuilder(URI.create("http://www.google.com"))
            .header("User-Agent", "Java").GET().build();
    MySimplePrintSubscriber subscriber1 = new MySimplePrintSubscriber();
    //NB ; MySimplePrintSubscriber est une implémentation simple de Flow.Subscriber<Object>

    // BodyHandlers.fromLineSubscriber(s) for registering a subscriber that will receive response text line by line
    // the return CompletableFuture contains no body but status and other infos
    client.sendAsync(req, BodyHandlers.fromLineSubscriber(subscriber1))
        .thenApply(HttpResponse::statusCode) //extract statusCode from httpResponse
        .thenAccept((status) -> {
            if (status != 200) { System.err.printf("ERROR: %d status %n", status); }
        });
    System.out.println("suite synchrone interpreted by " + Thread.currentThread().getName());
    try {
        Thread.sleep(2000); //pause ici pour éviter arrêt complet du programme
        // avant la fin des taches de fond asynchrones
    } catch (InterruptedException e) { e.printStackTrace(); }
    System.out.println("fin synchrone / interpreted by " + Thread.currentThread().getName());
}

```

Résultat de type :

```

mySimplePrintSubscriber>> Received onNext(item) with item=ligne1
mySimplePrintSubscriber>> Received onNext(item) with item=ligne2
mySimplePrintSubscriber>> Received onNext(item) with item=ligne3

```

3. Diverses nouveautés de java 9+

3.1. Nouvelles méthodes sur Optional<T>

Rappel essentiel java8 :

optional.get(); retourne la valeur interne (non nulle) ou bien retourne une **exception** si vide/empty.

optional.orElse(null); retourne la valeur interne (non nulle) ou bien retourne **null** si vide/empty.

Nouvelles méthodes depuis java9 :

```
public static void testDiversAvecNouvellesMethodesDeOptionalDepuisJava9() {
    Optional<String> opS1 = Optional.of("s1");
    Optional<String> opS2 = /*Optional.ofNullable(null); */ Optional.empty();

    //op.or(...) permet de construire et fournir un autre optional (valeur par défaut
    // ou plan B) si l'optionnel original est null/empty :
    System.out.println(opS1.or(()->Optional.of("default_string"))); //affiche Optional[s1]
    System.out.println(opS2.or(()->Optional.of("default_string")));
    //affiche Optional[default_string]

    //op.ifPresentOrElse(nomEmptyConsumer_as_lambda , emptyActionLambda)
    //permet de déclencher alternativement une lambda ou une autre en fonction d'un
    // contenu vide ou pas , pas d'enchaînement après (opération terminale en "void")
    opS1.ifPresentOrElse( (value) -> System.out.println("opS1="+value),
        () -> System.out.println("value of opS1 is empty"));
    //affiche opS1=s1
    opS2.ifPresentOrElse( (value) -> System.out.println("opS2="+value),
        () -> System.out.println("value of opS2 is empty"));
    //affiche value of opS2 is empty

    List<Optional<String>> listOfOptionals = new ArrayList<>();
    listOfOptionals.add(Optional.of("janvier"));
    listOfOptionals.add(Optional.empty());
    listOfOptionals.add(Optional.of("mars"));
    listOfOptionals.add(Optional.empty());
    listOfOptionals.add(Optional.of("mai"));
    List<String> filteredList = listOfOptionals.stream()
        .flatMap(Optional::stream)//Optional.stream() depuis java 9
        .collect(Collectors.toList());
    System.out.println("filteredList="+filteredList);// [janvier, mars, mai]
```

}

3.2. private method in interface (since java 9)

```
package tp;
public interface MyInterface {
    public int f1(int a); //normal method
    default int f2() { return private_submethod(); } //default method in interface since java 8
    default int f3() { return private_submethod(); }
    private int private_submethod() { return 9; } // private method in interface since java 9
}
```

```
package tp;
public class MyItfImpl implements MyInterface {
    @Override
    public int f1(int a) {
        return a+1;
    }
}
```

```
...
public static void test_private_interface_method_since_java9() {
    MyInterface obj = new MyItfImpl();
    System.out.println("obj.f2()="+obj.f2());
    System.out.println("obj.f3()="+obj.f3()); //same private sub method (for defaultt)
}
...
```

3.3. of(...) et immutable collections

```
...
import java.util.stream.IntStream;

public class MyApp {

    public static void test_stream_improvement_since_java9() {
```

```

        IntStream.iterate(1, i -> i < 5, i -> i + 1)
            .forEach(System.out::println);
    }

    public static void test_collection_factory_method_of_since_java9() {
        List<String> ls2 = List.of("acb" , "def" , "ghi"); //List.of(...) since java 9
        //ls2.add("jkl"); //java.lang.UnsupportedOperationException on
        // java.util.ImmutableCollections
        ls2.stream().forEach(System.out::println);

        //IMMUTABLE COLLECTIONS --> .of(...) = BOF if immutable is not wished !!!

        Set<Integer> integers = Set.of(1,2,3,4); //List.of(...) since java 9
        //integers.add(5); //java.lang.UnsupportedOperationException / IMMUTABLE !
        integers.stream()
            .map(i->i+1)
            .forEach(System.out::println);
    }
}

```

NB: il existe également

`nonemptyImmutableMap = Map.of(1, "one", 2, "two", 3, "three") ;`

3.4. Amélioration de try-with-resources (java 7 , java 9)

```

public static void tryWithResourceStyleSinceJava9() throws IOException {
    final Scanner scanner = new Scanner(new File("files/test.txt"));
    //scanner can now be declared as final before try(...){} since java9
    try (scanner) {
        while (scanner.hasNext()) {
            System.out.println(scanner.nextLine());
        }
    }
    //Automatic close (even without finally) because of
    //Scanner implementing java.lang.AutoCloseable of java7
}

```

3.5. Api StackWalker

L'api "*StackWalker*" a été introduite au sein de java 9 pour permettre de **parcourir la pile des appels de méthodes effectuées par le thread courant** .

Exemple :

```
package tp.j9_10_11;
import java.lang.StackWalker.StackFrame;
import java.util.List;

public class StackWalkingDemoApp {

    public static void main(String[] args) {
        StackWalkingDemoApp thisApp = new StackWalkingDemoApp();
        thisApp.m1();//calling m2 calling m3
        thisApp.mx();//calling mxy
        thisApp.my();//calling mxy
    }

    public void m1() {
        m2();
    }

    public void m2() {
        m3a();
        m3b();
        m3b2();
        m3b3();
    }

    public void mx() {
        System.out.println("mx() calling mxy()");
        mxy();
    }

    public void my() {
        System.out.println("my() calling mxy()");
        mxy();
    }

    public void mxy() {
        //may be called by mx() or my()
        StackWalker stackWalker = StackWalker.getInstance();
        if(MyStackWalkerUtil.wasCalledByMethod(stackWalker, "mx"))
            System.out.println("mxy() was called by mx()");
        else
            System.out.println("mxy() was not called by mx()");

        if(MyStackWalkerUtil.wasCalledByMethod(stackWalker, "my"))
            System.out.println("mxy() was called by my()");
        else
            System.out.println("mxy() was not called by my()");
    }

    public void m3a() {
        List<StackFrame> stackTrace = MyStackWalkerUtil.getFramesList(StackWalker.getInstance());
        System.out.println("stackTrace as List<StackFrame> : " + stackTrace + "\n");
        /*[tp.j9_10_11.StackWalkingDemoApp.m3a(StackWalkingDemoApp.java:26),
```

```

tp.j9_10_11.StackWalkingDemoApp.m2(StackWalkingDemoApp.java:20),
tp.j9_10_11.StackWalkingDemoApp.m1(StackWalkingDemoApp.java:16),
tp.j9_10_11.StackWalkingDemoApp.main(StackWalkingDemoApp.java:12)] */
}

public void m3b() {
    StackWalker stackWalker = StackWalker.getInstance();
    //stackWalker.forEach(System.out::println);
    stackWalker.walk(stackFrameStream->{stackFrameStream.forEach(System.out::println); return null;});
    /*
        tp.j9_10_11.StackWalkingDemoApp.m3b(StackWalkingDemoApp.java:33)
        tp.j9_10_11.StackWalkingDemoApp.m2(StackWalkingDemoApp.java:21)
        tp.j9_10_11.StackWalkingDemoApp.m1(StackWalkingDemoApp.java:16)
        tp.j9_10_11.StackWalkingDemoApp.main(StackWalkingDemoApp.java:12)    */
}

public void m3b2() {
    StackWalker stackWalker = StackWalker.getInstance();
    stackWalker.forEach(MyStackWalkerUtil::displayFrameDetails);
    /* StackWalker stackWalkerEx = StackWalker.getInstance(StackWalker.Option.RETAIN_CLASS_REFERENCE);
    stackWalkerEx.forEach(MyStackWalkerUtil::displayFrameDetailsEx); */
}

public void m3b3() {
    Class<?> caller = StackWalker
        .getInstance(StackWalker.Option.RETAIN_CLASS_REFERENCE)
        .getCallerClass();
    System.out.println("getClassCaller()=" + caller.getCanonicalName());

    StackFrame lastCallFrame = MyStackWalkerUtil.getLastCallFrame(StackWalker.getInstance());
    System.out.println("last call: " + lastCallFrame.getMethodName() + " on " + lastCallFrame.getClassName());
    //last call: m3b3 on tp.j9_10_11.StackWalkingDemoApp
    StackFrame firstCallFrame = MyStackWalkerUtil.getFirstCallFrame(StackWalker.getInstance());
    System.out.println("first call:" + firstCallFrame.getMethodName() + " on " + firstCallFrame.getClassName());
    //first call: main on tp.j9_10_11.StackWalkingDemoApp
}
}

```

```

package tp.j9_10_11;
import java.lang.StackWalker.StackFrame;
import java.util.List;                import java.util.Optional;
import java.util.stream.Collectors;   import java.util.stream.Stream;

public class MyStackWalkerUtil {

    //for walker.walk(MyStackWalkerUtil::stackFrameAsList);
    public static List<StackFrame> stackFrameAsList(Stream<StackFrame> stackFrameStream) {
        String thisUtilityCallName = MyStackWalkerUtil.class.getName();
        return stackFrameStream
            .filter(f -> !f.getClassName().equals(thisUtilityCallName))
            //remove side effect of this static utility class
            .collect(Collectors.toList());
    }

    public static List<StackFrame> getFramesList(StackWalker walker) {
        List<StackFrame> frameList = walker.walk(MyStackWalkerUtil::stackFrameAsList);
        return frameList;
    }

    public static boolean wasCalledByMethod(StackWalker walker, String methodName) {

```



```

        List<StackFrame> frameList = walker.walk(
            stackFrameStream->{
                return stackFrameStream.filter(f -> f.getMethodName().equals(methodName))
                    .collect(Collectors.toList());
            });
        return !frameList.isEmpty();
    }

    public static StackFrame getLastCallFrame(StackWalker walker) {
        List<StackFrame> frames = getFramesList(walker);
        return frames.get(0);
    }

    public static StackFrame getFirstCallFrame(StackWalker walker) {
        List<StackFrame> frames = getFramesList(walker);
        return frames.get(frames.size()-1);
    }

    public static void displayFrameDetails( StackFrame stackFrame) {
        System.out.println("\n*****\n");
        System.out.println(" Class name : " + stackFrame.getClassName());
        System.out.println(" File name : " + stackFrame.getFileName());
        System.out.println(" Bytecode index : " + stackFrame.getByteCodeIndex());
        System.out.println(" Line number : " + stackFrame.getLineNumber());
        System.out.println(" Method name : " + stackFrame.getMethodName());
        System.out.println(" Is method native or not? : " + stackFrame.isNativeMethod());
    }

    public static void displayFrameDetailsEx( StackFrame stackFrame) {
        displayFrameDetails(stackFrame);
        System.out.println(" Declaring Class name : " + stackFrame.getDeclaringClass());
    }
}

```

NB:

- L'option StackWalker.Option.**RETAIN_CLASS_REFERENCE** permet d'effectuer un appel ultérieur à StackWalker.**getCallerClass()** et StackFrame.**getDeclaringClass()**
- l'option StackWalker.Option.SHOW_REFLECT_FRAMES permet de voir des "frames" spéciales (par défaut pas conservées) liées à des invocations indirectes via l'api java.lang.reflect (ex: Method.invoke() , Constructor.newInstance())
- l'option StackWalker.Option.SHOW_HIDDEN_FRAMES permet de voir des "frames" spéciales liées aux mécanismes internes de la JVM

Exemple d'affichage détaillé :

Class name : tp.j9_10_11.StackWalkingDemoApp
 File name : StackWalkingDemoApp.java
 Bytecode index : 1
 Line number : 18
 Method name : m1
 Is method native or not? : false

3.6. Api VarHandle

L'api très pointue "*VarHandle*" permet de **manipuler indirectement des variables**.

Cette api peut quelquefois être considérée comme une alternative à l'api *java.lang.reflect* (en ce qui concerne la manipulation des variables d'une classe) .

Exemple :

```
package tp.j9_10_11;

import java.lang.invoke.MethodHandles;
import java.lang.invoke.VarHandle;

public class VarHandleExample {

    //varHandle = référence sur une variable pour pouvoir la manipuler par la suite
    private VarHandle sizeFieldOfAnySubObject = null;
    public enum SubObjectType { machine, animal };
    Object subObject ; //anything with a ".size"

    public VarHandleExample(SubObjectType subObjectType) {
        switch(subObjectType) {
            case machine: this.subObject = new MachineWithSize(); break;
            case animal: this.subObject = new AnimalWithSize(); break;
        }
        try {
            //initialisation du "varHandle" :
            sizeFieldOfAnySubObject = MethodHandles.lookup()
                .in(this.subObject.getClass())
                .findVarHandle(this.subObject.getClass(), "size", Integer.class);
            //ON A AFFAIRE ICI A UNE SORTE D'ATERNATIVE A L'INTROSPECTION/REFLECTION
            //DE MANIERE A ACCEDER ET MANIPULER un attribut(variable de classe) d'un objet quasi quelconque
        } catch (NoSuchFieldException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
    }

    //récupération indirecte de la valeur de la taille
    public Integer getSize() {
        if(sizeFieldOfAnySubObject==null) return null;
        return (Integer) sizeFieldOfAnySubObject.get(subObject);
    }

    //mise à jour indirecte de la valeur de la taille
    public void setSize(Integer size) {
        if(sizeFieldOfAnySubObject!=null)
            sizeFieldOfAnySubObject.set(subObject, size);
    }

    //affichage indirect de la valeur de la taille :
    public void printSize() {
        System.out.println("size="+this.getSize());
    }
}
```

```
private class AnimalWithSize {
    private String name="strange animal";
    private Integer size = 50;

    @Override
    public String toString() {
        return "AnimalWithSize [name=" + name + ", size=" + size + "]";
    }
}

private class MachineWithSize {
    private String label="good machine";
    private Integer size = 80;

    @Override
    public String toString() {
        return "MachineWithSize [label=" + label + ", size=" + size + "]";
    }
}

public static void main(String[] args) {
    System.out.println("For animal:");
    VarHandleExample varHandleEx = new VarHandleExample(SubObjectType.animal);
    varHandleEx.printSize();//50
    varHandleEx.setSize(2);
    varHandleEx.printSize();//2

    System.out.println("For machine:");
    varHandleEx = new VarHandleExample(SubObjectType.machine);
    varHandleEx.printSize();//80
    varHandleEx.setSize(2);
    varHandleEx.printSize();//2
}
```

3.7. Annotation @SafeVarargs améliorée

@SafeVarargs est un peu comme @SuppressWarnings et a été amélioré en java 9

Exemple :

```
//bad V1 with warnings (MyNew9_10_11TestApp.java uses unchecked or unsafe operations ,
// Type safety: Potential heap pollution via varargs parameter elements)
public static <T> T[] unsafeAsArray(T... elements) {
    return elements; // unsafe! don't ever return a parameterized varargs array just returning Object[]
}

//V2 (without warning because of @SafeVarargs on a function with <T> and ...)
//NB: @SafeVarargs exists since java 7
//since java 9, @SafeVarargs can be used on private methods
@SafeVarargs
private static <T> Object[] safeAsArray(T... elements) {
    return elements; // safe as just returning Object[]
}

public static <T> T[] badArrayOfThree(T elt) {
    T[] arrayOf3Elts = unsafeAsArray(elt, elt, elt); // broken! This will be an Object[] no matter what T is
    return arrayOf3Elts;
}

public static <T> Object[] goodArrayOfThree(T elt) {
    Object[] arrayOf3Elts = safeAsArray(elt, elt, elt); // explicitly view as Object[] no matter what T is
    return arrayOf3Elts;
}

public static void threeBadAndGoodXxx() {
    try {
        String[] threeXxxAsStringArray = badArrayOfThree("xxx"); //cast exception
        System.out.println("threeXxxAsStringArray:"+threeXxxAsStringArray);
    } catch (Exception e) {
        System.err.println(e.getMessage()); //cast exception
    }

    Object[] threeXxxAsObjectArray = goodArrayOfThree("xxx");
    System.out.println("threeXxxAsObjectArray:");
    for(Object obj : threeXxxAsObjectArray)
        System.out.println("\t"+obj); //ok , affiche 3 fois xxx
}

//called by main()
public static void testImprovedSafeVarargs() {
    threeBadAndGoodXxx();
}
```

X - var et ajouts/restrictions de java 10 et 11

1. var (inférence de type pour variables locales)

```
public static void test_var_since_java10(){
    List<String> ls1 = new ArrayList<String>(); //since java 5
    List<String> ls2 = new ArrayList<>(); //since java 7 (diamond)
    var ls3 = new ArrayList<String>(); //type inference since java 10 with new keyword "var"

    ls3.add("abc"); ls3.add("def");
    ls3.stream().forEach(System.out::println);

    //NB "var" (since java 10) is not a reserved word.
    //an old variable may be named "var" :
    int var=12; //possible (but NOT ADVISED) .
    System.out.println("var="+var);
}
```

```
public class ExperimentalClass {
    var text="text1"; //not allowed here

    public static void main(String[] args) {
        var localVar = "texte"; //ok seulement sur variables locales
        System.out.println("localVar="+localVar);
    }

    public static void m1(var p/*not allowed here*/){
        System.out.println("p="+p);
    }
}
```

2. Ajouts divers de java 10 et 11

2.1. Ajouts divers de java 10

* Unmodifiable collection improvement:

```
Collections.unmodifiableList(...);
.collect(Collectors.toUnmodifiableList());
```

**internal JVM improvements*

2.2. Nouvelles méthodes pour les bibliothèque String, Collections et Files

Java 11 ajoute des méthodes pratiques aux API String, Collections et Files.

API String

L'API String ajoute quatre nouvelles méthodes pratiques.

- **String.repeat(Integer)** : Il répète simplement une chaîne n fois.
- **String.isBlank()** : qui indique si une chaîne est vide ou ne contient que des espaces.
- **String.strip()** : prend soin de supprimer les espaces blancs de début et de fin.
- **String.stripLeading()** ne supprime que les espaces blancs de début .
- **String.stripTrailing()** ne supprime que les espaces blancs de fin.
- **String.lines()** : vous permet de traiter des textes multilignes en tant que flux. Un flux représente une séquence d'éléments pouvant être traités séquentiellement sans avoir à charger tous les éléments en mémoire en une seule fois.

API Collections

Java 11 facilite la conversion une collection en tableau via la méthode **Collection.toArray (Function <T []>)**.

Exemple :

```
final Set<String> saisons = Set.of("Hiver", "Printemps", "Ete", "Automne");
out.println(Arrays.toString(saisons.toArray(String[]::new)));
```

API Files

Java 11 ajoute les méthodes **Files.readString(Path)** et **Files.writeString(Path, CharSequence, OpenOption)** avec diverses surcharges, ce qui facilite beaucoup la lecture et l'écriture de fichiers.

2.3. Autres nouvelles fonctionnalités diverses

2.3.a. Optional et Predicate

Sur un Optional on peut maintenant utiliser la méthode **isEmpty()** .

Java 11 ajoute la méthode statique **Predicate.not(Predicate)**, qui renvoie une instance Predicate prédéfinie qui annule le prédicat donné.

2.3.b. Utiliser le mot clé var dans lambda

Depuis Java 11, on peut désormais utiliser le mot-clé **var** pour les paramètres d'une fonction

Lambda. L'utilisation de **var** pour les paramètres de **Lambda** présente un avantage majeur: vous pouvez annoter la variable. Par exemple, vous pouvez indiquer que la valeur de la variable ne peut pas être nulle en utilisant l'annotation **@NotNull** (*nb : @NotNull ne fonctionne pas sans type*)

Exemples :

```
(var s1, var s2) -> s1 + s2
```

```
(@NotNull var s1, @NotNull var s2) -> s1 + s2
```

2.3.c. Adoption d'UNICODE 10

Java 11 utilise maintenant la dernière version : Unicode 10.

La mise à jour d'Unicode 8 à 10 inclut **16 018** nouveaux caractères et **10** nouveaux scripts

2.3.d. Quelques liens pour aller plus loin (encore approfondir)

<https://developpement-informatique.com/article/232/nouveautes-de-java-11>

3. Restrictions/Restructurations de java 11

3.1. Jdk 11 allégé

Comparé au jdk8 , le jdk 11 (ou bien la version openjdk 11) avec fait une grande cure d'amaigrissement .

Les modules JEE, JavaFX et CORBA considérés soit comme "pointus / rarement utilisés" , soit comme "obsolètes" ne sont plus fournis dans le cœur de la jvm .

C'est maintenant au développeur de configurer son projet pour intégrer des dépendances additionnelles facultatives telles que "java.xml.bind" , ...

Principal impact :

Lorsque l'on doit migrer un projet existant de java 8 (LTS) vers java 11 (ou openjdk 11) (LTS) , on a souvent besoin d'ajouter de nouvelles dépendances au sein de pom.xml

Exemple partiel :

```
<!-- since java11 , jaxb2 (indirect dependency of jackson) is no more provided in jdk/jre
      theses dependencies are now required: -->
  <dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.1</version>
  </dependency>
  <dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-runtime</artifactId>
    <version>2.3.1</version>
    <scope>runtime</scope>
  </dependency>
```

3.2. Jdk 11 avec changement de license

jdk-8	BLC / généralement gratuit
jdk-11	OTN (Oracle Technologie Network ...) payant en usage commercial
openjdk-11, ...	BLC ou ... / généralement gratuit

Avant java 11

Avant la version 11 Java SE (Standard Edition) est sous licence [Oracle BCL](#) (Binary Code License Agreement for the Java SE Platform Products).

Cette licence permet d'**utiliser certaines fonctionnalités gratuitement et d'autres en payant** (pour un usage commercial, tout est gratuit pour un usage interne).

Ces fonctionnalités sont regroupées dans [3 éditions différentes](#) :

1. Java SE (gratuit)

Contient les fonctionnalités suivantes :

- Java Development Kit (JDK) : JRE + JavaFX SDK + outils permettant aux développeurs de compiler et de déboguer.
- Java Runtime Environment (JRE) : JVM (Java Virtual Machine) + bibliothèques de classes standard + composants permettant l'exécution d'applications Java.
- Java FX Runtime
- JRockit JDK : JDK basé sur une JVM haute performance (présent jusqu'à Java 6)

2. Java SE Advanced et Java SE Advanced Desktop (payant)

Identique à Java SE (gratuit) mais avec des outils de diagnostic et de profilage de la JVM en plus (– Java Mission Control – Flight Recorder)

3. Java SE Suite avec "JRockit Real Time" (payant)

A partir de java 11 :

Depuis la version 11 la licence Oracle n'est plus la même. Oracle propose deux versions, une sous licence open source (appelée version OpenJDK d'Oracle), l'autre sous licence commerciale (appelée version JDK d'Oracle).

La **version open source** est sous [licence GPLv2+CPE](#) (GNU General Public License v2 with Classpath Exception).

Elle permet de distribuer une application (gratuitement ou non) et oblige de rendre accessible le code source de l'application (l'application peut être copiée, modifiée et distribuée).

Les versions open source existent en fait depuis la **version 9** de Java et sont disponibles ici :

<https://jdk.java.net/>

La **version commerciale** est maintenant [sous licence OTN](#) (Oracle Technology Network License Agreement for Oracle Java SE).

Elle permet de commercialiser une application sans quelle soit sous licence open source.

Au fur et à mesure les fonctionnalités payantes du JDK d'Oracle sont intégrées aux versions OpenJDK d'Oracle (Java Flight Recorder, Java Mission Control, Application Class-Data Sharing, ZGC).

On se retrouve donc avec une [version 11 du JDK et d'OpenJDK identique fonctionnellement parlant](#), le but est que les versions JDK et OpenJDK soient facilement interchangeables.

XI - Switch expressions et ajouts java 12,...,17

1. Switch expressions

La nouvelle forme du **switch()** avec des "**lambda**" était apparue dès les jdk 1.12 et 1.13 en mode "preview". C'est enfin devenu une nouvelle fonctionnalité "standard" à partir de la version **1.14**

1.1. expression d'un switch/case avec des "lambda"

```
int dayOfWeek = 4;
System.out.print("dayOfWeek="+dayOfWeek + " -> ");
switch(dayOfWeek) {
    case 1 -> System.out.println("monday");
    case 2 -> System.out.println("tuesday");
    case 3 -> System.out.println("wednesday");
    case 4 -> System.out.println("thursday");
    case 5 -> System.out.println("friday");
    case 6 -> System.out.println("saturday");
    case 0 , 7 -> System.out.println("sunday");
    default -> System.out.println("unknown");
}
```

Résultat:

```
dayOfWeek=4 -> thursday
```

1.2. switch en tant qu'expression retournant un résultat

```
String dayName="vendredi";
System.out.print("dayName="+dayName + " -> ");
int dayNumber =
    switch(dayName) {
        case "lundi" , "monday" -> 1;
        case "mardi" , "tuesday" -> 2;
        case "mercredi" , "wednesday" -> 3;
        case "jeudi" , "thursday" -> 4;
        case "vendredi" , "friday" -> 5;
        case "samedi" , "saturday" -> 6;
        case "dimanche" , "sunday" -> 7;
        default -> 0;
    };
System.out.println("dayNumber="+dayNumber);
```

Résultat :

```
dayName=vendredi -> dayNumber=5
```

1.3. switch avec expressions complexes et mot clef yield

Lorsqu'une expression doit nécessiter plusieurs instructions pour être calculée, on peut englober celles-ci par un bloc d'accolades et préciser la valeur calculée/retournée via le mot clef "yield" (ressemblant à un "return" mais de niveau switch et pas de niveau fonction) .

```
int value = (int) (Math.random() * 14);
String result = switch( value ) {
    case 0, 2, 4, 6, 8 -> {
        double racine = Math.sqrt( value );
        yield "chiffre pair dont la racine carré vaut " + racine;
    }
    case 1, 3, 5, 7, 9 -> {
        double carre = value * value;
        yield "chiffre impair dont le carré vaut " + carre;
    }
    default -> "ce n'est plus un chiffre, mais un nombre";
};
System.out.println("value=" + value + "->" + result );
```

Résultats :

value=2->chiffre pair dont la racine carré vaut 1.4142135623730951

value=13->ce n'est plus un chiffre, mais un nombre

value=3->chiffre impair dont le carré vaut 9.0

...

2. Pattern Matching instanceof

En preview en version 14, 15 et standardisé en version 16 et 17/LTS .

```
public class TestPatternMatchingInstanceOfApp {

    public static void main(String[] args) {
        String s = "abc";
        oldStyleWithoutPatternMatchingInstanceOf(s);
        newStyleWithPatternMatchingInstanceOf(s);
    }

    public static void oldStyleWithoutPatternMatchingInstanceOf(Object obj) {
        if (obj instanceof String) {
        String str = (String) obj;
        int len = str.length();
        System.out.println("obj is as string of length="+len);
        }
    }

    public static void newStyleWithPatternMatchingInstanceOf(Object obj) {
        if (obj instanceof String str) {
        int len = str.length();
        System.out.println("obj is as string of length="+len);
        }
    }
}
```

C'est un **sucre syntaxique** (simplification de syntaxe) **bien pratique** .

3. TextBloc

Un "TextBloc" (depuis java 15) est une chaîne de caractère formulée comme un **bloc de texte sur plusieurs lignes**. Encadré par des `"""` (triple guillemet), un *textBloc* peut comporter des caractères quelconques qui n'ont plus besoin d'échappement.

Ceci est surtout utile pour le format JSON (nécessitant des `"` et pas de `'`) à l'intérieur d'une chaîne java délimitée par des `"`.

écriture sans textbloc et très peu lisible :

```
public static void oldStyleWithoutTextBloc() {
    String myJsonString
        = "{\r\n" + "\"username\" : \"JeanBon\", \r\n" + "\"country\" : \"France\" \r\n" + "}";
    System.out.println("myJsonString="+myJsonString);
}
```

équivalent bien plus lisible avec textBloc :

```
public static void newStyleWithTextBloc() {
    String myJsonStringAsTextBloc = """
        {
            "username" : "AlexTherieur",
            "country" : "Belgique"
        }
        """;
    System.out.println("myJsonStringAsTextBloc="+myJsonStringAsTextBloc);
}
```

→ affiche

```
myJsonStringAsTextBloc={
"username" : "AlexTherieur",
"country" : "Belgique"
}
```

```
public static void someTextBlocs() {
    //au minimum 2 lignes de code :
    //  """ suivi de newligne pour commencer
    //  texte quelconque au milieu
    //  """; pour finir
    String simpleTextBloc0 = """
        blabla""";
    System.out.println("simpleTextBloc0="+simpleTextBloc0);
    //affiche simpleTextBloc0=blabla
}
```

```

//pour bien controller l'intentation , il faut savoir que lorsque le compilateur va analyser
//le code source, il va supprimer le nombre minimum de tabulations qu'il y a
//au niveau de toutes les lignes de code (du textBloc sans tenir compte du tout début = "")
//Bonne pratique : au sens IDE (eclipse/IntelliJ/...) placer des tabulations pour indenter le code
//          au sens contenu final du textBloc , utiliser plutôt des espaces et sauts de lignes
//          placer la fin "" sur une ligne séparée
//          (comportant le nombre de tabulations qui seront explicitement supprimées du texte)
String textBloc3AvecIndentationControlee = ""
    {
        "username" : "AlexTherieur",
        "country" : "Belgique"
    }
    "";

System.out.println("textBloc3AvecIndentationControlee="+textBloc3AvecIndentationControlee);

//par défaut les espaces en fin de ligne sont considérés inutiles et ne sont pas conservés
//le caractère d'échappement \s permet d'explicitement demander à conserver les espaces en fin de ligne
String textBloc4AvecEspacesConserveEnFinDeLignes = ""
    blabla  \s"";

System.out.println("textBloc4AvecEspacesConserveEnFinDeLignes="
    +textBloc4AvecEspacesConserveEnFinDeLignes + "suiteApres");

}

```

```

public static void textBlocsWithFormattedParams() {
    String username="toto";
    int age=40;
    double height = 1.73;
    boolean ok=true;
    String jsonString = ""
        {
            "username" : "%s",
            "age" : %d,
            "height" : %s,
            "ok" : %b
        }
        ""formatted(username,age,String.valueOf(height),ok);
        //same syntax of String.format
    System.out.println("jsonString="+jsonString);
}

```

affiche

```

{ "username" : "toto",
  "age" : 40,
  "height" : 1.73,
  "ok" : true
}

```

4. Record (classe de données simplifiée pour DTO)

"Preview" en version 14 et 15, les "record" sont standardisés depuis java 16 (et 17 LTS) .

//exemple de classe POJO/JavaBean classique (pour comparaison) :

```
public class OldCustomer{

    private Integer id;
    private String firstName;
    private String lastName;

    public OldCustomer() {        super();
    }

    public OldCustomer(Integer id, String firstName, String lastName) {
        super();
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Override
    public String toString() {
        return "OldCustomer [id=" + id + ", firstName=" + firstName
            + ", lastName=" + lastName + "]";
    }

    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }

    //+ autres get/set
    //+ hashCode()
    //+ equals()
}
```

classe équivalente avec annotations de lombok :

```
import lombok.*;

@Getter @Setter @ToString
@NoArgsConstructor @AllArgsConstructor
@EqualsAndHashCode
public class LombokCustomer {
    private Integer id;
    private String firstName;
    private String lastName;
}
```

Utilisation classique d'un javaBean codé avec ou sans lombok:

```
OldCustomer oldC1 = new OldCustomer(1,"jean","Bon");
System.out.println("oldC1 as POJO =" + oldC1.toString());
System.out.println("id of oldC1 =" + oldC1.getId());
oldC1.setFirstName("luc");
System.out.println("firstName of oldC1 =" + oldC1.getFirstName());
```

```
OldCustomer oldC2 = new OldCustomer();
System.out.println("oldC2 =" + oldC2.toString());
```

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

```
LombokCustomer lC1 = new LombokCustomer(1,"jean","Bon");
System.out.println("olC1 as Lombok POJO =" + lC1.toString());
System.out.println("id of lC1 =" + lC1.getId());
lC1.setFirstName("luc");
System.out.println("firstName of lC1 =" + lC1.getFirstName());
```

```
LombokCustomer lC2 = new LombokCustomer();
System.out.println("lC2 =" + lC2.toString());
```

Nouveau mot clef "record"

Le nouveau mot clef **record** permet de demander au compilateur de construire automatiquement une sorte de pseudo **POJO IMMUTABLE** (sans Setter possible !) et avec des accesseurs de type .xxx() ne respectant même pas les conventions .getXxx() !!!

Dans les détails : classe construite "final" héritant de **java.lang.Record** et avec des champs privés "final" et avec .equals()/hashCode()/toString() et des accesseurs de type .xxx() et allArgsConstructor .

NB :

- record ne peut pas être vu comme un remplacement de lombok car IMMUTABLE/final/read-only et car .xxx() plutôt que .getXxx()
- record doit plutôt être vu comme une sorte de petit DTO/VO très léger

Minimalist v1 (with no default constructor):

```
public record CustomerRecord(Integer id,String firstName,String lastName) {
}
```

Utilisation :

```
CustomerRecord c1 = new CustomerRecord(1,"jean","Bon");
System.out.println("c1 as record =" + c1.toString());
System.out.println("id of c1 =" + c1.id()); //mais pas .getId() ni .id
c1.setFirstName("luc");
System.out.println("firstName of c1 =" + c1.firstName()); //mais pas .getFirstName() ni .firstName
```


public void **setFirstName**(String firstName) { this.firstName = firstName; }
 est **interdit/impossible** car **this.firstName** est **final/immutable** !!!

// V2 (with explicit default constructor) :

```
public record CustomerRecord(Integer id,String firstName,String lastName) {
    public CustomerRecord() { this(0,null,null);}
    //possible mais pas souvent utile si final/immutable !!!
}
```

Utilisation :

```
CustomerRecord c2 = new CustomerRecord();
System.out.println("c2="+c2.toString());
//affiche ici CustomerRecord[id=0, firstName=null, lastName=null]
```

V3 (V2 + quelques *redefinitions* & *ajout de methodes*)

```
public record CustomerRecord(Integer id,String firstName,String lastName) {
    public CustomerRecord() { this(0,null,null);}
    public String fullName() { return firstName + " " + lastName;}
    public String lastName() { return lastName.toUpperCase(); }
}
```

V4 (V3 + new record constructor checking syntax)

```
public record CustomerRecord(Integer id,String firstName,String lastName) {
    //not useful default constructor:
    public CustomerRecord() { this(0,"unknown","unknown");}

    //new method:
    public String fullName() { return firstName + " " + lastName;}

    //old pojo style getter as a new complementary method :
    public String getFirstName() { return firstName; }

    //accessor redefinition :
    public String lastName() { return lastName.toUpperCase(); }

    //specific record new syntax for allArgsConstructor
    //partial redefinition with value(s) checking:
    public CustomerRecord {
        if (lastName == null || lastName.isBlank()) {
            throw new IllegalArgumentException("lastName is required");
        }
    }
}
```

```

public class TestRecordApp {

    public static void main(String[] args) {
        testUsefulRecordV1();
        testUsefulRecordV2();
        testUsefulRecordV3();
    }

    //private or public locally RECORD seems to be a good use case for "record" new concept :

    private record AddressV1(Integer number,String street,String zipCode,String town) {
        public String toJsonString() {
            return ""
                {
                    "number" : %d,
                    "street" : "%s",
                    "zipCode" : "%s",
                    "town" : "%s"
                }
                ""formatted(number,street,zipCode,town);
        }
    };

    public static void testUsefulRecordV1() {
        AddressV1 a1 = new AddressV1(12,"rueElle","75000" ,"Paris");
        System.out.println("a1="+a1.toString());
        System.out.println("a1="+a1.toJsonString());
    }

    //V2 avec compatibilité avec api jackson-databind (souvent utilisé par JEE , Spring, ...)
    //pratique pour définition de DTO/VO (à la volée)
    public record AddressV2(Integer number,String street,String zipCode,String town) {
    };

    public static void testUsefulRecordV2() {
        AddressV2 a1 = new AddressV2(12,"rueElle","75000" ,"Paris");
        System.out.println("a1="+a1.toString());
        try {
            ObjectMapper jacksonObjectMapper = new ObjectMapper();
            String a1JsonString = jacksonObjectMapper.writeValueAsString(a1);
            System.out.println("via jackson, a1JsonString="+a1JsonString);
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
    }
}

```

Résultats :

```

a1=AddressV1[number=12, street=rueElle, zipCode=75000, town=Paris]
a1={

```

```
"number" : 12,
"street" : "rueElle",
"zipCode" : "75000",
"town" : "Paris"
}
```

```
a1=AddressV2[number=12, street=rueElle, zipCode=75000, town=Paris]
via jackson, a1JsonString={"number":12,"street":"rueElle","zipCode":"75000","town":"Paris"}
```

*//NB : Les records de java ne sont bien gérés qu'avec des versions récentes de jackson-databind .
 // la version 2.12.5 de **jackson-databind** supporte bien les records de java 17 et est compatible
 //avec une version récente de SpringBoot et SpringMVC (sur partie @RestController)*

```
public class Dto {
    public record Address(Integer number,String street,String zipCode,String town) {
    };

    public record Person(Integer id,String firstName,String lastName,Address address) {
        public Person(Integer id,String firstName,String lastName) {
            this(id,firstName,lastName,null);
        }
    };
}
```

```
public static void testUsefulRecordV3() {
    Dto.Person p1Dto = new Dto.Person(1,"jean","Bon" ,
                                     new Dto.Address(12,"rueElle","75000" ,"Paris"));
    System.out.println("\np1Dto="+p1Dto.toString());

    try {
        ObjectMapper jacksonObjectMapper = new ObjectMapper();
        String p1JsonString = jacksonObjectMapper.writeValueAsString(p1Dto);
        System.out.println("via jackson, p1JsonString="+p1JsonString);
        //-----
        Dto.Person p1BisDto=
            jacksonObjectMapper.readValue(p1JsonString,Dto.Person.class);
        System.out.println("via jackson, p1BisDto=clone de p1Dto="+p1BisDto.toString());
    } catch (JsonProcessingException e) {
        e.printStackTrace();
    }
}
```

Résultats :

```
p1Dto=Person[id=1, firstName=jean, lastName=Bon, address=Address[number=12,
street=rueElle, zipCode=75000, town=Paris]]
```

```
via jackson, p1JsonString={"id":1,"firstName":"jean","lastName":"Bon","address":
{"number":12,"street":"rueElle","zipCode":"75000","town":"Paris"}}
```

```
via jackson, p1BisDto=clone de p1Dto=Person[id=1, firstName=jean, lastName=Bon,
address=Address[number=12, street=rueElle, zipCode=75000, town=Paris]]
```

Attention (restriction importante)

En interne les "record" sont codés comme des classes spéciales héritant de la classe abstraite *java.lang.Record* .

Etant donné que java ne supporte que l'héritage simple (via mot clef extends), **il n'est pas possible d'utiliser le mot clef extends de manière à ce qu'un record hérite d'un autre record** .

On peut cependant utiliser le mot clef **implements** entre un **record** et une (ou plusieurs) **interface(s)**.

Exemple :

```
public class MyRecords {  
    public interface Product {Long id(); String label(); Double price();}  
    public record ProductL0(Long id,String label,Double price) implements Product {  
    }  
  
    public interface ProductI1 extends Product {String features();}  
    public record ProductL1(Long id,String label,Double price,String features)  
        implements ProductI1{  
    }  
}
```

```
MyRecords.ProductL0 p0 = new MyRecords.ProductL0(1L,"productA", 12.5);
```

```
MyRecords.Product pRef = p0;
```

```
System.out.println(pRef.toString());
```

```
System.out.println("id=" + pRef.id() + " label=" + pRef.label() + " price=" + pRef.price());
```

Autre détails :

La méta classe **java.lang.Class<T>** comporte maintenant une nouvelle méthode booléenne **isRecord()** .

5. Sealed classes

"Preview" en versions 15,16 et standardisé en version 17 .

```
//NB: "SEALED" signifiant scellé en francais
//est une nouveauté du langage java permettant d'indiquer que seules certaines classes
//(précisées par permits ... , ... )
//auront le droit d'hériter de la classe ou interface actuelle
//cela ressemble à final class qui interdit carrément tout héritage

//sealed keyword can be used if option "enabled preview features for java 16"
public sealed interface AnimalDomestique permits Chat, Chien {
    public void sayHello();
    //...
}
// effet : seules les classes Chat et Chien auront le droit d'hériter de AnimalDomestique .

/*
//HERITAGE via extends ou implements INTERDIT via sealed sur AnimalDomestique :
final class Ce implements AnimalDomestique{
//...
}
*/
```

```
//déclarée final , la classe Chat ne peut pas avoir de sous classe
public final class Chat implements AnimalDomestique{

    @Override
    public void sayHello() {
        System.out.print("chat miaou - ");
    }

    public void ronronner() {
        System.out.println(" ronron ");
    }

}
```

```
//déclarée non-sealed , la classe Chien peut pas avoir des sous classes
public non-sealed class Chien implements AnimalDomestique{
    @Override
    public void sayHello() {
        System.out.print("chien wouf wouf - ");
    }

    public void aLaNiche() {
        System.out.println(" dodo niche ");
    }

}
```

```

public class ChienFou extends Chien {
    @Override
    public void sayHello() {
        System.out.print("chienFou wouf wouf grrr wouf grrr - ");
    }

    public void tournerEnRond() {
        System.out.println(" tourne pas rond dans sa tete ");
    }
}

```

```

public static void operationOnSealed(AnimalDomestique a) {
    if(a instanceof Chien chien) {
        chien.aLaNiche();
        if(chien instanceof ChienFou chienFou) {
            chienFou.tournerEnRond();
        }
    }
    if(a instanceof Chat chat) {
        chat.ronronner();
    }
}

public static void testSealed() {

    var animauxDomestiques = new ArrayList<AnimalDomestique>();
    animauxDomestiques.add(new Chat());
    animauxDomestiques.add(new Chien());
    animauxDomestiques.add(new ChienFou());

    animauxDomestiques.stream().forEach(a-> { a.sayHello(); operationOnSealed(a);});
}

```

affiche

```

chat miaou - ronron
chien wouf wouf - dodo niche
chienFou wouf wouf grrr wouf grrr - dodo niche
tourne pas rond dans sa tete

```

NB : Une interface scellée est implémentable comme un paquets de "record" .

Exemple :

```

public sealed interface Vivant {
    record Vertebre(String name, String p1 , String p2) implements Vivant {}
    record Invertebre(String name, String p3) implements Vivant {}
}

```

NB: Le principal intérêt des classes scellées de java 16/17 tient dans la possibilité d'utiliser un pattern matching au sein d'un switch/case de type d'objet sans partie "default" .

6. Pattern matching sur switch/case of object

//NB: still in "preview mode" in java 17-LTS !!!

//java --enable-preview --source 17

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.8.1</version>
  <configuration>
    <release>${java.version}</release>
    <source>${java.version}</source>
    <target>${java.version}</target>
    <compilerArgs>--enable-preview</compilerArgs>
  </configuration>
</plugin>
```

Exemple :

```
package tp.j15_16_17;

public class TestPatternMatchingSwitchPreviewApp {

  static double getDoubleUsingSwitch(Object o) {
    return switch (o) {
      case Integer i -> i.doubleValue();
      case Float f -> f.doubleValue();
      case Double d -> d.doubleValue();
      case String s -> Double.parseDouble(s);
      case null -> 0d;
      default -> 0d;
    };
  }

  static String getTypeAnimalDomesticAsString(AnimalDomestique a) {
    return switch (a) {
      case Chat chat -> "chat";
      case Chien chien -> "chien";
      //default -> "ni chat , ni chien";
    };
  }

  /*
```

Remarque importante :

*Ce switch/case avec pattern-matching de type nécessite absolument une partie "default" si AnimalDomestique n'est pas scellé (sans le mot clef sealed et ...)
et ne nécessite pas de partie "default" si AnimalDomestique est scellé*

Autrement dit l'intérêt principal des classes scellées tient dans la possibilité d'utiliser directement les différents types de classes concrètes (dérivant d'un même type abstrait scellé) au sein d'un switch/case sans default sans avoir besoin de gérer en parallèle une énumération à valeurs possibles fixes/finies.

```

*/
}

static String getTypeVivantFromRecord(Vivant v) {
    return switch (v) {
        case Vivant.Invertebre ive -> "invertebre";
        case Vivant.Vertebre ve -> "vertebre";
        //with default if Vivant is not sealed
    };
}

public static void main(String[] args) {
    System.out.println(getDoubleUsingSwitch("12.5")); //12.5
    System.out.println(getDoubleUsingSwitch(12.6)); //12.6
    System.out.println(getDoubleUsingSwitch(12.7f)); //12.699999809265137
    System.out.println(getDoubleUsingSwitch(12)); //12.0
    System.out.println(getDoubleUsingSwitch(null)); //0.0

    System.out.println(getTypeAnimalDomesticAsString(new Chat())); //chat
    System.out.println(getTypeAnimalDomesticAsString(new Chien())); //chien
    System.out.println(getTypeAnimalDomesticAsString(new ChienFou())); //chien

    System.out.println(getTypeVivantFromRecord(
        new Vivant.Invertebre("limace","pas rapide"))); // invertebre
    System.out.println(getTypeVivantFromRecord(
        new Vivant.Vertebre("homme","intelligent","pas toujours sage"))); // vertebre
    }
}

```

7. utilitaire jpackage

L'utilitaire **jpackage** sert à construire des installateurs d'applications pour windows, linux ou mac.

Ça peut construire des fichier ".msi" pour windows.

package_with_jpackage.bat

```

cd /d "%~dp0"
REM this script should be run after mvn package or ...

set JAVA_HOME=C:\Program Files\Java\jdk-17
REM NB: on windows10 , with jdk17 , wix tools set is a required dependency of jpackage
REM wix311-binaries.zip can be download from https://github.com/wixtoolset
set WIX_HOME=C:\Prog\wix311
set PATH=%PATH%;%JAVA_HOME%\bin;%WIX_HOME%
REM type=msi or exe for windows , deb or rpm for windows , dmg or pkg on mac
set TYPE=msi
set NAME=JPackagedemoApp

```



```

set MAIN_CLASS=tp.MainClassForJPackageTestApp
set INPUT_JAR_DIR=./target
set MAIN_JAR=tp_java_8_14-0.0.1-SNAPSHOT.jar
set OPTIONS=--win-console --java-options '--enable-preview'

jpackage --input %INPUT_JAR_DIR% --name %NAME% --main-jar %MAIN_JAR%
--main-class %MAIN_CLASS% --type %TYPE% %OPTIONS%
REM fichier construit : JPackageDemoApp.msi

REM apres installation de JPackageDemoApp.msi
REM C:\Program Files\JPackageDemoApp\JPackageDemoApp.exe et runtime java

REM https://www.baeldung.com/java14-jpackage
REM https://www.devdungeon.com/content/use-jpackage-create-native-java-app-installers

```

8. Autres apports des versions récentes (12,...,17)

8.1. Helpful *NullPointerException* (JEP 358)

Depuis le java 14, les traces générées autour de *NullPointerException* sont beaucoup plus claires/explicites .

Exemple de code avec bug:

```

int[] arr = null;
arr[0] = 1;

```

Avec les jdk précédent (ex : v11) :

```

Exception in thread "main" java.lang.NullPointerException
at tp.MyClass.main(MyClass.java:27)

```

Maintenant, la log d'erreur indique plus clairement la source du problème:

```

java.lang.NullPointerException: Cannot store to int array because "arr" is null

```

9. Nouveau type de licence depuis java 17

Le JDK 17 D'Oracle De Nouveau Gratuit Pour Une Utilisation Commerciale

Le JDK Oracle est à nouveau disponible gratuitement pour une utilisation en production - sous la nouvelle "[Oracle No-Fee Terms et conditions](#)" (NFTC). Cette décision annule une [décision de 2018](#) de facturer l'utilisation en production du JDK d'Oracle et n'affecte pas la [distribution OpenJDK d'Oracle](#). La NFTC s'applique à la version 17 récemment publiée d'Oracle JDK et aux versions futures.

Autre raison de passer à java 17 : **java 17 est plus rapide/performant que java 11 .**

XII - Virtual Threads et ajouts java 18,...,21

1. Pattern and record matching

Rappel du pattern matching "if/instanceof" (déjà possible en java 16,17)

```
if (obj instanceof String str) {
    ...
}
```

1.1. Pattern matching switch (preview java 17 , finalisé java 21)

```
static double getDoubleUsingSwitch(Object o) {
    return switch (o) {
        case Integer i -> i.doubleValue();
        case Float f -> f.doubleValue();
        case Double d -> d.doubleValue();
        case String s -> Double.parseDouble(s);
        case null -> 0d;
        default -> 0d;
    };
}
```

```
static String getTypeAnimalDomesticAsString(AnimalDomestique a) {
    return switch (a) {
        case Chat chat -> "chat";
        case Chien chien -> "chien";
        //default -> "ni chat , ni chien"; //default not mandatory if AnimalDomestique is "sealed"
    };
}
```

1.2. Destructuration via "record pattern"

```
public class MyRecords {
    public record Point(int x,int y) {};
    public record GPS_Point(double latitude,double longitude) {};
    public record Location(String name,GPS_Point position) {};
}
```

```

import tp.java_new_21plus.record_pattern.MyRecords.*;

public class RecordDeconstructApp {

    public static void main(String[] args) {
        Double d1=23.5;
        print(d1);

        Point pt1 = new Point(20,30);
        print(pt1);
        print_destructuring(pt1);
        switch_destructuring(pt1);

        Location loc1 = new Location("Paris" , new GPS_Point(48.866667, 2.333333));
        print(loc1);
        print_destructuring(loc1);
        switch_destructuring(loc1);
    }

    public static void print(Object o) {
        //Pattern matching for instance of (since java 16/17):
        if(o instanceof Double d) {System.out.println("double d="+d);}
        if(o instanceof Point pt) { System.out.println("Point pt="+pt);}
        if(o instanceof Location loc) { System.out.println("Location loc="+loc);}
    }

    public static void print_destructuring(Object o) {
        //record pattern = destructure (like javascript) in Pattern matching for instance of
        if(o instanceof Point (int x,int y)) {
            System.out.println("Destructuring Point x="+x + " y="+y);
        }
        if(o instanceof Location (String name,GPS_Point(double latitude,double longitude))) {
            System.out.println("Destructuring Location name="+name
                               + " latitude="+latitude + " longitude="+longitude );
        }
    }

    public static void switch_destructuring(Object o) {
        //record pattern = destructure (like javascript) in switch expression
        double maxi = switch(o) {
            case Point (int x,int y) -> (double) Math.max(x, y);
            case Location (String name,GPS_Point(double latitude,double longitude))
                -> Math.max(latitude, longitude);
            default -> 0;
        };
        System.out.println("switch_destructuring, maxi="+maxi);
    }
}

```

2. Virtual Thread

2.1. Threads natifs et virtuels

Un **thread natif** est géré par le système d'exploitation (ex : linux ou windows)

Un **thread virtuel** n'est géré que par la machine virtuelle java . C'est possible depuis les versions 18,19,20,21 de java. En étant plus léger, un thread virtuel peut démarrer plus rapidement et le changement de contexte d'un thread à l'autre est également plus rapide.

2.2. Démarrage d'un thread virtuel

La classe **java.lang.Thread** comporte maintenant une méthode **.ofVirtual()** permettant de créer un nouveau thread virtuel .

Thread.ofVirtual()

```
.name(virtualThreadName)
//.start(runnable); //code automatiquement démarré
.unstarted(runnable); //code à démarrer via appel à .start() ou équivalent
```

Exemple :

```
import java.time.Duration;

public class MyThreadUtil {

    static void log(String message) {
        System.out.println(message + " by " + Thread.currentThread());
    }

    static void sleep(Duration duration) {
        try { Thread.sleep(duration); } catch (InterruptedException e) { e.printStackTrace(); }
    }

    static void sleep(long nbMs) {
        try { Thread.sleep(nbMs); } catch (InterruptedException e) { e.printStackTrace(); }
    }
}
```

```
public class MyRunnableUtil {
    public static Runnable prepareCoffeeRunnable() {
        return ()->{
            MyThreadUtil.log("prepare coffee");
            MyThreadUtil.sleep(Duration.ofMillis(1000));
            MyThreadUtil.log("coffee is ready");
        };
    }
}
```

```

public static void testSimpleVirtualThread() throws Exception {
    Thread vt1 = Thread.ofVirtual().name("vt1")
                .unstarted(MyRunnableUtil.prepareCoffeeRunnable());
    vt1.start(); // démarrage de vt1
    vt1.join(); // attente de la fin de vt1 depuis main Thread
}

```

==>

prepare coffee by VirtualThread[#21,vt1]/runnable@ForkJoinPool-1-worker-1

coffee is ready by VirtualThread[#21,vt1]/[runnable@ForkJoinPool-1-worker-1](#)

2.3. Threads virtuels via Executors

```

//one unnamed virtualThread for each concurrent task submitted to the executor
try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {
    ....
}

```

```

public static void testViaExecutors() throws Exception {
    //one named virtualThread (generated by factory)
    //for each concurrent task submitted to the executor
    final ThreadFactory virtualThreadFactory =
        Thread.ofVirtual().name("routine-", 0).factory();
    //names of virtual threads will be "routine-0", "routine-1", ...
    try (var executor = Executors.newThreadPerTaskExecutor(virtualThreadFactory)) {
        Future<?> f1 = executor.submit(MyRunnableUtil.prepareJokeRunnable());
        Future<?> f2 = executor.submit(MyRunnableUtil.prepareCoffeeRunnable());

        Future<Integer> f3 = executor.submit(MyRunnableUtil.longComputationCallable());
        f1.get(); // attente fin de f1
        f2.get(); // attente fin de
        Integer resF3 = f3.get(); // attente resultat de f3
        System.out.println("resF3="+resF3);
    }
}

```

Résultat :

prepare a joke by VirtualThread[#21,routine-0]/runnable@ForkJoinPool-1-worker-1

prepare coffee by VirtualThread[#23,routine-1]/runnable@ForkJoinPool-1-worker-2

start computing by VirtualThread[#25,routine-2]/runnable@ForkJoinPool-1-worker-3

joke is ready by VirtualThread[#21,routine-0]/runnable@ForkJoinPool-1-worker-1

coffee is ready by VirtualThread[#23,routine-1]/runnable@ForkJoinPool-1-worker-1

cumputing is ready by VirtualThread[#25,routine-2]/runnable@ForkJoinPool-1-worker-3

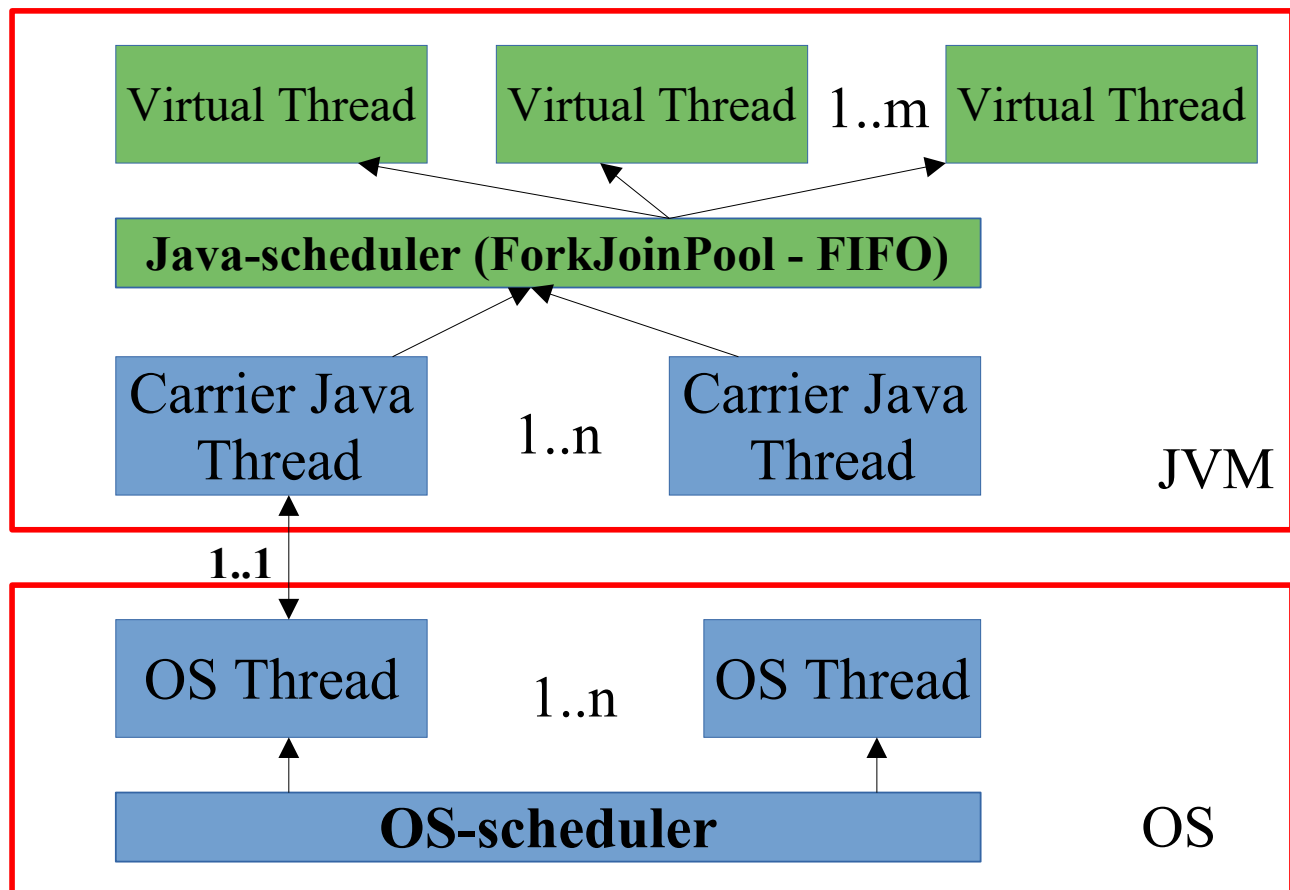
resF3=5

2.4. CarrierPlatformThread

```
//[#virtualThreadId,virtualThreadName]/... carrierPlatformThread
```

```
//exemple : VirtualThread[#37,routine-8]/runnable@ForkJoinPool-1-worker-4
```

VirtualThreads and CarrierPlatformThread



La machine virtuelle java s'appuie en interne sur un pool de "carrier java thread" (mappé chacun avec un thread natif de l'OS) pour prendre en charge un paquet de threads virtuels.

Lorsqu'un thread virtuel est bloqué (en attente, en sommeil, ...), celui-ci peut être temporairement dissocié du "carrier java thread" porteur de manière à ce que ce "carrier java thread" puisse exécuter d'autres tâches plus utiles (autre "virtual thread").

Le lien entre un "virtual_thread" et un "carrier java thread" peut évoluer dans le temps.

Exemple :

```
VirtualThread[#24,routine-2]/runnable@ForkJoinPool-1-worker-7
```

```
VirtualThread[#24,routine-2]/runnable@ForkJoinPool-1-worker-6
```

Par défaut sur une machine à n processeurs/coeurs, le "ForkJoinPool" comportera n "carrier java

thread" (ex : *ForkJoinPool-1-worker-1* , ... , *ForkJoinPool-1-worker-8*)

Cas d'utilisation :

Les threads virtuels apportent un plus dans le cas où beaucoup d'opérations concurrentes sont potentiellement bloquées (en attentes , exemple = appel synchrone vers api REST).

Par contre , les `VirtualThreads` n'apportent rien pour un usage intensif du CPU . Pour cela l'algorithme du `ForkJoin` et les `ParallelStream` sont déjà assez bien optimisés.

3. Structured Concurrency (preview java 21)

L'api "**Structured Concurrency**" (toujours en mode "*preview*" en java 21) sert à déclencher des sous tâches en parallèle (qui seront exécutées par des threads virtuels) tout en ayant la possibilité d'attendre (en mode "bloquant avec timeout") selon les besoins :

- soit la fin de toutes les sous-tâches
- soit le résultat de la sous-tâches la plus rapide

Exemple :

```
public class FakeLongTaskService {
    public static FakeLongTaskService INSTANCE = new FakeLongTaskService();

    public double slowSquareRoot(double x,int nbMs) throws InterruptedException{
        try {
            Thread.currentThread().sleep(nbMs);
        } catch (InterruptedException e) {
            System.err.println(Thread.currentThread()+"was interrupted before end of sleep="+nbMs);
            throw e;
        }
        if(nbMs<50) throw new RuntimeException("lunatic exception because nbMs<50");
        double res = Math.sqrt(x);
        System.out.println("slowSquareRoot(x="+x + ") returning res="+res
            + " after sleep=" + nbMs + " "+ Thread.currentThread());
        return res;
    }
}
```

En mode attente du résultat de la sous-tâches la plus rapide :

```
...
import java.time.Instant;
import java.util.concurrent.StructuredTaskScope;
import java.util.concurrent.TimeoutException;

public class WithFirstFinishedSubTask {

    public static WithFirstFinishedSubTask INSTANCE = new WithFirstFinishedSubTask();
    /*
     ShutdownOnSuccess captures the first result and shuts down the task scope
     to interrupt unfinished threads and wakeup the owner.
     This class is intended for cases where the result of any subtask will do ("invoke any")
     and where there is no need to wait for results of other unfinished tasks.
     It defines methods to get the first result or throw an exception if all subtasks fail.
     */

    public Double quickerOfSlowSquareRoot(double x) {
        Double firstFinishedResult=0.0;
        try (var scope = new StructuredTaskScope.ShutdownOnSuccess<Double>()) {
            //int extra = -200; //causing all subtaskToFail because xxx + extra < 50
            //int extra = 1000; //causing timeoutException
            int extra = 10; //no global error
            StructuredTaskScope.Subtask<Double> subtask1 =
                scope.fork(() -> FakeLongTaskService.INSTANCE.slowSquareRoot(x, 200+extra));//200ms
            StructuredTaskScope.Subtask<Double> subtask2 =
                scope.fork(() -> FakeLongTaskService.INSTANCE.slowSquareRoot(x, 100+extra));
            StructuredTaskScope.Subtask<Double> subtask3 =
                scope.fork(() -> FakeLongTaskService.INSTANCE.slowSquareRoot(x, 50+extra));//50ms
        }
```



```

        StructuredTaskScope.Subtask<Double> subtask4 =
            scope.fork(() -> FakeLongTaskService.INSTANCE.slowSquareRoot(x, 40));
            //if less than 50 with negative extra ->lunatic Exception
        //NB: sometimes futurRes4 throws a RuntimeException , it fails ,
        //but "no problem" if at least another // subtask succeed

        //scope.join();
        scope.joinUntil(Instant.now().plusSeconds(1)); //with TimeoutException if all is not finish 1s later now

        //firstFinishedResult = scope.result(); //without specific exception
        firstFinishedResult = scope.result(e -> new RuntimeException("cannot compute squareRoot"));
        //with specific exception if all failed (but not timeout)
        System.out.println("firstFinishedResult="+firstFinishedResult);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        } catch (TimeoutException e) {
            throw new RuntimeException(e);
        } catch (/*ExecutionException*/ Exception e) {
            throw new RuntimeException(e);
        }
        return firstFinishedResult;
    }
}

public static void testQuickerSubTask() {
    try {
        double res = WithFirstFinishedSubTask.INSTANCE.quickerOfSlowSquareRoot(9.0);
        System.out.println("quickerOfSlowSquareRoot() returning "+res);
    } catch (Exception e) {
        System.err.println("quickerOfSlowSquareRoot() throwing "+e.getMessage());
    }
}

```

Résultats :

slowSquareRoot(x=9.0) returning res=3.0 after sleep=60

VirtualThread[#200]/runnable@ForkJoinPool-1-worker-1

VirtualThread[#199]/runnable@ForkJoinPool-1-worker-8was interrupted before end of sleep=110

VirtualThread[#198]/runnable@ForkJoinPool-1-worker-7was interrupted before end of sleep=210

firstFinishedResult=3.0

quickerOfSlowSquareRoot() returning 3.0

En mode "attente de la fin de toutes les sous-tâches parallèles"

```

...
import java.util.concurrent.StructuredTaskScope;
import java.util.concurrent.TimeoutException;

public class AllFinishedSubTasks {

    public static AllFinishedSubTasks INSTANCE = new AllFinishedSubTasks();
    /*
    ShutdownOnSuccess captures the first result and shuts down the task scope
    to interrupt unfinished threads and wakeup the owner.
    This class is intended for cases where the result of any subtask will do ("invoke any")
    and where there is no need to wait for results of other unfinished tasks.

```

It defines methods to get the first result or throw an exception if all subtasks fail.
 */

```
public List<Double> doubleListSlowSquareRoot(List<Double> doubleList) {
    List<Double> doubleResultList = new ArrayList<>();
    try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
        List<StructuredTaskScope.Subtask<Double>> subtaskList =
            new ArrayList<StructuredTaskScope.Subtask<Double>>();
        //int extra = 1000; //causing timeoutException
        int extra = 10; //no global error
        for(Double val : doubleList) {
            StructuredTaskScope.Subtask<Double> subtask = scope.fork(
                () -> (Double) FakeLongTaskService.INSTANCE.slowSquareRoot(val, 100+extra)
            );
            subtaskList.add(subtask);
        }
        //scope.join();
        scope.joinUntil(Instant.now().plusSeconds(1)); //with timeout

        for(StructuredTaskScope.Subtask<Double> subtask : subtaskList ) {
            doubleResultList.add(subtask.get());
        }
    } catch (InterruptedException e) { throw new RuntimeException(e);
    } catch (TimeoutException e) { throw new RuntimeException(e);
    } catch (/*ExecutionException*/ Exception e) { throw new RuntimeException(e); }
    return doubleResultList;
}

public static void testAllFinishedParallelSubTask() {
    try {
        List<Double> doubleList = Arrays.asList(9.0 , 4.0 , 25.0 , 16.0);
        System.out.println("doubleList="+doubleList);
        List<Double> resultList =
            AllFinishedSubTasks.INSTANCE.doubleListSlowSquareRoot(doubleList);
        System.out.println("doubleListSlowSquareRoot() returning "+resultList);
    } catch (Exception e) {
        System.err.println("doubleListSlowSquareRoot() throwing "+e.getMessage());
    }
}
}
```

Résultats :

doubleList=[9.0, 4.0, 25.0, 16.0]

*slowSquareRoot(x=4.0) returning res=2.0 after sleep=110
 VirtualThread[#203]/runnable@ForkJoinPool-1-worker-5
 slowSquareRoot(x=16.0) returning res=4.0 after sleep=110
 VirtualThread[#205]/runnable@ForkJoinPool-1-worker-1
 slowSquareRoot(x=25.0) returning res=5.0 after sleep=110
 VirtualThread[#204]/runnable@ForkJoinPool-1-worker-8
 slowSquareRoot(x=9.0) returning res=3.0 after sleep=110
 VirtualThread[#202]/runnable@ForkJoinPool-1-worker-7
doubleListSlowSquareRoot() returning [3.0, 2.0, 5.0, 4.0]*

4. Foreign Function and Memory (preview java 21)

L'api FFM (Foreign Function and Memory) est toujours en mode **preview** en java 21.

Cette api permet d'accéder à des fonctions et blocs mémoires externes à la JVM (ex : fonction native du langage "C") de manière efficace et rigoureuse (mieux que JNI ou JNA).

Attention:

sans arrêt , de petits changements au niveau de l'api FFM (Foreign Function and Memory) entre les previews des versions 18,19,20,21 (PAS STABLE DU TOUT)

Le code ci-dessous est pour le jdk 21

<https://openjdk.org/jeps/442> pour java 21 (third preview of FFM)

```
...
import java.lang.foreign.Arena;
import java.lang.foreign.FunctionDescriptor;
import java.lang.foreign.Linker;
import java.lang.foreign.MemorySegment;
import java.lang.foreign.SymbolLookup;
import java.lang.foreign.ValueLayout;
import java.lang.invoke.MethodHandle;

public class FFMTestApp {
    public static void main(String[] args) throws Throwable {
        // 1. Get a lookup object for commonly used libraries
        SymbolLookup stdlib = Linker.nativeLinker().defaultLookup();

        // 2. Get a handle to the "strlen" function in the C standard library
        MethodHandle strlen = Linker.nativeLinker().downcallHandle(
            stdlib.find("strlen").orElseThrow(),
            FunctionDescriptor.of(ValueLayout.JAVA_LONG, ValueLayout.ADDRESS));

        try (Arena offHeap = Arena.ofConfined()) { //version java 21

            // 3. Convert Java String to C string and store it in off-heap memory
            MemorySegment str = offHeap.allocateUtf8String("Happy Coding!");
                                                    //13 caracteres

            // 4. Invoke the foreign function
            long len = (long) strlen.invoke(str);

            System.out.println("len = " + len); //len=13 (ok)
        }
    }
}
```

5. Vector api (incubator java 21)

"**Vector api**" (toujours en mode "incubator" en java 21) sert à effectuer d'un seul coup des opérations mathématiques (ex : addition, multiplication,...) sur des tableaux de valeurs en utilisant certaines fonctionnalités pointues de certains micro-processeurs récents.

```
package tp.java_new_21plus.vector_api;
import jdk.incubator.vector.IntVector;

public class TestVectorApi {
    //in run configuration : VM arguments
    //--add-modules=jdk.incubator.vector

    public static void main(String[] args) {
        smallVectorAddition();
        largeVectorAdditionV1(); //with IntVector.SPECIES_512
        largeVectorAdditionV2(); //with IntVector.SPECIES_128 , 4 iterations
    }

    public static void display_intTab(String tabName, int[] values) {
        System.out.print(tabName+"={");
        for(int v : values)
            System.out.print(" " + v);
        System.out.print("}\n");
    }
}
//...
}
```

```
public static void smallVectorAddition() {
    int[] a = {1, 2, 3, 4};
    int[] b = {17, 18, 19, 20};
    int[] c = new int[4]; //4fois 32bits = 128bits

    // Scalar (ordinary) computation:
    //c[0] = a[0] + b[0];
    //...
    //c[15] = a[15] + b[15];
    for (int i = 0; i < c.length;i++)
        c[i] = a[i] + b[i];
    display_intTab("c",c);

    // Vector computation:
    IntVector va = IntVector.fromArray(IntVector.SPECIES_128, a, 0);
    IntVector vb = IntVector.fromArray(IntVector.SPECIES_128, b, 0);
    IntVector vc = va.add(vb); //add operation between 2 vector

    int[] c2 = new int[4];
    vc.toArray(c2, 0); //store vc in ordinary array of scalar
    display_intTab("c2",c2);
}
```

variante sans boucle :

```
public static void largeVectorAdditionV1() {
    int[] a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};
    int[] b = {17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32};

    // Vector computation:
    int[] c2 = new int[16];
    IntVector va = IntVector.fromArray(IntVector.SPECIES_512, a, 0);
    IntVector vb = IntVector.fromArray(IntVector.SPECIES_512, b, 0);
    IntVector vc = va.add(vb); //add operation between 2 vector
    vc.toArray(c2, 0); //store vc in ordinary array of scalar
    display_intTab("c2", c2);
}
```

16 fois 32 = 512 (tableaux de 16 cases de int (32bits))

variante avec boucle et offset:

```
public static void largeVectorAdditionV2() {
    int[] a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};
    int[] b = {17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32};

    // Vector computation:
    int[] c2 = new int[16];
    for (int offset = 0; offset < c2.length; offset += IntVector.SPECIES_128.length()) {
        IntVector va = IntVector.fromArray(IntVector.SPECIES_128, a, offset);
        IntVector vb = IntVector.fromArray(IntVector.SPECIES_128, b, offset);
        IntVector vc = va.add(vb); //add operation between 2 vector
        vc.toArray(c2, offset); //store vc in ordinary array of scalar
    }
    display_intTab("c2", c2);
}
```

Résultats :

WARNING: Using incubator modules: jdk.incubator.vector

c={ 18 20 22 24}

c2={ 18 20 22 24}

c={ 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48}

c2={ 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48}

ANNEXES

XIII - Annexe – new io

1. NIO2 (new IO)

1.1. Introduction à nio , nio2

Dès l'origine du langage java , la classe File du package java.io était disponible pour accéder aux éléments du "file system" (fichier , répertoire, ...) et pour effectuer des opérations dessus (tester l'existence d'un fichier , supprimer des fichiers , ...).

Bien qu'opérationnel , le package java.io (et la classe File) ont quelques limitations importantes :

- La classe File manque de fonctionnalité importante telle qu'une méthode "copy".
- Beaucoup de méthode retourne des booléens plutôt que des exceptions . Ce qui rend assez délicate l'analyse des problèmes (raison exacte de l'erreur?).
- Pas de bonne gestion des liens symboliques .
- Un ensemble très limité d'attributs sur les fichiers sont disponibles sur java.io.File.

Pour surmonter certaines lacunes , un nouveau package "java.nio" a été introduit dès java4 de façon à apporter quelques améliorations très techniques telles que les suivantes :

- *Channels et Selectors*: un "channel" est une abstraction d'une caractéristiques de bas niveau telle qu'un fichier mappé en mémoire .
- *Buffers*: Buffering pour toutes les classes wrapper/primitives (sauf pour Boolean).
- *Jeu de caractères*: **Charset** (*java.nio.charset*), avec encodeurs et décodeurs entre bytes[] et symboles "Unicode".

La version 7 de java a introduit un nouveau (gros) package "**java.nio.file**" (alias nio2) .

Ce package apporte à la fois de nouvelles fonctionnalités techniques telles que :

- gestion des liens symboliques
- bonne gestions des "Path" (aspect bien séparé).
- meilleur gestion des "File attributes"

et de nouvelles syntaxes (plus concises , plus "orienté objet") telles que les remontées d'exceptions.

Attention : Entre java1, java4 , java7 , des ajouts , mais pas de suppression (pour garder une compatibilité avec les anciennes versions).

Bien que la classe "java.io.File" n'est pas officiellement considérée comme obsolète / "deprecated" , il est très conseillé d'utiliser les nouvelles classes de nio2 (sachant qu'il existe des passerelles entre "java.io" et "java.nio.file") .

2. Principales classes et interfaces de nio2

NIO 2 (associé aux packages "**java.nio.file**" et "**java.nio.file.attribute**") repose sur plusieurs classes et interfaces dont les principales sont :

- **Path** : encapsule un chemin dans le système de fichiers

- **Files** : avec méthodes statiques pour manipuler les éléments du système de fichiers
- *FileSystemProvider* : fournisseur technique (implémentation) de FS
- **FileSystem** : encapsule un système de fichiers
- *FileSystems* : fabrique permettant entre autres de créer une instance de *FileSystem*
- **FileStorage** : système de stockage (ex : partition / volume logique ,)

Quelques équivalences/transpositions de fonctionnalités basiques entre "java.io" et "nio2" :

Fonctionnalité	java.io	NIO 2
Encapsuler un chemin	java.io.File	java.nio.file.Path
Vérifier les permissions	File.canRead(), File.canCrite() et File.canExecute()	Files.isReadable(), Files.isWritable() et Files.isExecutable().
Vérifier le type d'élément	File.isDirectory(), File.isFile()	Files.isDirectory(Path, LinkOption...), Files.isRegularFile(Path, LinkOption...),
Taille d'un fichier	File.length()	Files.size(Path)
Obtenir ou modifier la date de dernière mise à jour	File.lastModified() , File.setLastModified(long)	Files.getLastModifiedTime(Path, LinkOption...), Files.setLastModifiedTime(Path, FileTime)
Modifier les attributs	File.setExecutable(), File.setReadable(), File.setReadOnly(), File.setWritable()	Files.setAttribute(Path, String, Object, LinkOption...)
Déplacer un fichier	File.renameTo()	Files.move()
Supprimer un fichier	File.delete()	Files.delete()
Créer un fichier	File.createNewFile()	Files.createFile()
Créer un fichier temporaire	File.createTempFile()	Files.createTempFile(Path, String, FileAttributes<?>), Files.createTempFile(Path, String, String, FileAttributes<?>)
Tester l'existence d'un fichier	File.exists	Files.exists() ou Files.notExists()
Obtenir le chemin absolu	File.getAbsolutePath() ou File.getAbsolutePath()	Path.toAbsolutePath()
Chemin canonique (sans ".", "..")	File.getCanonicalPath() ou File.getCanonicalFile()	Path.toRealPath() ou Path.normalize()
Convertir en URI	File.toURI()	Path.toURI()
L'élément est-il caché	File.isHidden()	Files.isHidden()
Obtenir le contenu d'un répertoire	File.list() ou File.listFiles()	Path.newDirectoryStream()
Créer un répertoire	File.mkdir() ou File.mkdirs()	Path.createDirectory()
Obtenir le contenu du répertoire racine	File.listRoots()	FileSystem.getRootDirectories()
Place totale , libre ,	File.getTotalSpace()	FileStore.getTotalSpace()

... sur FS	File.getFreeSpace()	FileStore.getUnallocatedSpace()
------------	---------------------	---------------------------------

3. Gestion des chemins (Path)

L'interface "java.nio.file.**Path**" est une représentation abstraite d'un chemin (relatif ou absolu) au sein d'un système de fichiers.

Un chemin peut référencer un fichier, un répertoire, un lien symbolique, un sous-chemin, ...

Les instances de "Path" sont "immutables" et peuvent être utilisées dans un contexte multi-threads.

3.1. Obtention d'une instance de "Path"

Récupération d'un chemin en appelant explicitement **getPath()** sur une instance de "**FileSystem**" :

```
Path chemin = FileSystems.getDefault().getPath(
    "C:/Users/powerUser/Temp/monfichier.txt");
```

équivalent indirect via la méthode (utilitaire / "helper") statique **Paths.get()** :

```
Path chemin = Paths.get("C:/Users/powerUser/Temp/monfichier.txt");
Path chemin2 = Paths.get(URI.create("file:///C:/Users/powerUser/Temp/monfichier.txt"));
Path chemin3 = Paths.get(System.getProperty("java.io.tmpdir"), "monfichier.txt");
```

NB : **Path.of()** existe depuis java 11 et se comporte comme **Paths.get()**

NB : bien que (sous windows) on puisse utiliser le séparateur "\\" (ex : [C:\\RepXy\\monfichier.txt](#)), il vaut mieux utiliser le séparateur "/" (plus portable et plus simple).

Passerelle "io / nio2" (depuis java 7) :

En partant d'une instance de l'ancienne classe **java.io.File**, on peut appeler la nouvelle méthode **".toPath()** pour récupérer une instance de "java.nio.file.Path".

3.2. Obtention des éléments d'un chemin

Méthode	Rôle
String getFileName()	Retourne le nom du dernier élément du chemin. Si le chemin concerne un fichier alors c'est le nom du fichier qui est retourné
Path getName(int index)	Retourne l'élément du chemin dont l'index est fourni en paramètre. Le premier élément (hors racine) possède l'index 0 et correspond à la première partie sous la racine (ex : "windows" sous racine " c:/ ").
int getNameCount()	Retourne le nombre d'éléments du chemin (hors racine)
Path getParent()	Retourne le chemin parent ou null s'il n'existe pas
Path getRoot()	Retourne la racine d'un chemin absolu (par exemple C:\ sous windows ou / sous Unix) ou null pour un chemin relatif
String toString()	Retourne le chemin sous la forme d'une chaîne de caractères

Path subPath (int <i>beginInclusiveIndex</i> , int <i>endExclusiveIndex</i>)	Retourne un sous-chemin (hors racine) [beginIndex,endIndex[
--	---

Exemple avec un chemin absolu :	Exemple avec un chemin relatif :
path=C:\Windows\Fonts\arial.ttf toString() = C:\Windows\Fonts\arial.ttf getFileName() = arial.ttf getRoot() = C:\ getName(0) = Windows getNameCount() = 3 getParent() = C:\Windows\Fonts subpath(0,2) = Windows\Fonts	path=Fonts\arial.ttf toString() = Fonts\arial.ttf getFileName() = arial.ttf getRoot() = null getName(0) = Fonts getNameCount() = 2 getParent() = Fonts subpath(0,2) = Fonts\arial.ttf

3.3. Manipulation , combinaison et conversions de chemins

Méthode	Rôle
Path normalize ()	Normaliser (ou rendre "canonique") un chemin en supprimant les éléments non indispensables « . » et « .. » qu'il contient.
Path relativize (Path other)	Retourner le chemin relatif permettant d'aller du "path courant" vers celui fourni en paramètres .
Path resolve (Path relative)	Combiner deux chemins (courant + relatif_en_arg) pour former global
Path toAbsolutePath ()	Retourne un chemin absolu (en <i>tenant compte du répertoire courant du contexte d'exécution</i> (idem à "pwd")).
Path toRealPath (LinkOption...)	Retourner le chemin physique du chemin notamment en résolvant les liens symboliques selon les options fournies . Peut lever une exception si le fichier au bout du chemin courant (absolu ou relatif) n'existe pas .
URI toUri ()	Retourner le chemin sous la forme d'une URI (file:///)

Exemple1 :

```
path.toString() = C:\Utilisateurs\..\Windows\..\Fonts\arial.ttf
path.normalize() = C:\Windows\Fonts\arial.ttf
```

```
path.toString() = ..\..\Fonts\..\arial.ttf
path.normalize() = ..\..\Fonts\arial.ttf      (seul le ../ inutile supprimé)
```

Exemple2 :

```
Path windowsFontPath = Paths.get("c:/Windows/Fonts");
Path usersPath = Paths.get("c:/Utilisateurs");
//relative path from current to arg
Path relativePathFromUsersToWindowsFonts=
usersPath.relativize(windowsFontPath);
System.out.println("relative path=" + relativePathFromUsersToWindowsFonts);
```

```
relative path=..\Windows\Fonts
```

Exemple3 :

```
Path windowsPath = Paths.get("c:/Windows");
Path arialRelativePath = Paths.get("Fonts/arial.ttf");
Path globalPath = windowsPath.resolve(arialRelativePath);
System.out.println("globalPath="+globalPath);

globalPath=c:\Windows\Fonts\arial.ttf
```

```
Path windowsFontsPath = Paths.get("C:/Windows/Fonts");
System.out.println("uri=" + windowsFontsPath.toUri()); //java.net.URI

uri=file:///C:/Windows/Fonts/
```

```
Path relativePath = Paths.get(".");
Path absolutePath = relativePath.toAbsolutePath(); //selon rep courant (pwd)
System.out.println("absolutePath=" + absolutePath);

absolutePath=C:\Users\didier\workspace\test_j7_j8\.
```

Rappel : sous linux , un lien symbolique se construit via la commande
\$ ln -s /nom_du_dossier_source nom_du_lien

Attention, sous windows , un lien symbolique ne doit pas être confondu avec un raccourci .
 A l'époque de windows XP , un lien symbolique se construisait avec la commande "*junction.exe*" (à installer).

Depuis "Vista" , et encore aujourd'hui avec windows 7, 8 et 10, un lien symbolique se construit avec la commande **mklink** suivante :

MKLINK [[/D] | [/H] | [/J]] Lien Cible

/D : Crée un lien symbolique vers un répertoire. Par défaut, il s'agit d'un lien symbolique vers un fichier.

/H : Crée un lien réel à la place d'un lien symbolique.

/J : Crée une jonction de répertoires.

Lien : Spécifie le nom du nouveau lien symbolique.

Cible : Spécifie le chemin d'accès (relatif ou absolu) auquel le nouveau lien fait référence.

Exemple (à lancer avec des droits "administrateur"):

```
cd c:\tmp\bb
```

```
mklink /J lien_vers_aa c:\tmp\aa
```

jonction créée pour lien_vers_aa <==> c:\tmp\aa

```
Path path = Paths.get("C:/tmp/bb/raccourci_vers_aa/f1.txt");
→ java.nio.file.NoSuchFileException: C:\tmp\bb\raccourci_vers_aa\f1.txt
```

```
Path path = Paths.get("C:/tmp/bb/lien_vers_aa/f1.txt");
try {
    System.out.println("chemin indirect sans suivre résolution lien symbolique="
        + path.toRealPath(LinkOption.NOFOLLOW_LINKS));
    System.out.println("chemin direct suivant résolution lien symbolique="
        + path.toRealPath());
} catch (IOException e) {
    e.printStackTrace();
}
```

```
chemin indirect sans suivre résol lien symbolique=C:\tmp\bb\lien_vers_aa\fl.txt
chemin direct suivant résolution lien symbolique=C:\tmp\aa\fl.txt
```

3.4. Comparaisons de chemins

La méthode `.equals()` permet de tester si deux chemins ont des valeurs identiques.

L'interface **Path** hérite de **Comparable**. Les chemins peuvent donc être automatiquement triés.

L'interface **Path** comporte en outre les méthodes de comparaison spécifiques suivantes :

Méthode	Rôle
int compareTo (Path other) (Comparable<Path>)	Comparer le chemin avec celui fourni en paramètre et retourne 0 si identiques, <0 avant, >0 si après
boolean endsWith (Path other) boolean endsWith (String other)	Comparer la fin du chemin avec celui fourni en paramètre
boolean startsWith (Path other) boolean startsWith (String other)	Comparer le début du chemin avec celui fourni en paramètre

3.5. interface PathMatcher avec méthode matches(path) et "glob"

Un "glob" est une expression basée sur certains méta-caractères qui seront comparés à des parties de "path" .

```
Path path = Paths.get("C:/tmp/aa/fl.txt");
PathMatcher txtMatcher = FileSystems.getDefault().getPathMatcher("glob:*.txt");
if (txtMatcher.matches(path.getFileName())) {
    System.out.println(path + " reference un fichier texte");
}
```

Motif (dans glob)	Rôle associé / correspondance
*	Aucun ou plusieurs caractères
**	Aucun ou plusieurs sous-répertoires
?	Un caractère quelconque
{ }	Un ensemble de motifs exemple : {htm, html}
[]	Un ensemble de caractères. Exemple : [A-Z] : toutes les lettres majuscules [0-9] : tous les chiffres [a-z,A-Z] : toutes les lettres indépendamment de la casse Chaque élément de l'ensemble est séparé par un caractère virgule

	Le caractère - permet de définir une plage de caractères A l'intérieur des crochets, les caractères *, ? et / ne sont pas interprétés
\	Il permet d'échapper des caractères pour éviter qu'ils ne soient interprétés. Il sert notamment à échapper le caractère \ lui-même
Les autres caractères	Ils se représentent eux-mêmes sans être interprétés

Quelques exemples :

<i>Glob</i>	<i>Correspondances</i>
*.html	tous les fichiers ayant l'extension .html
???	trois caractères quelconques
[0-9]	tous les fichiers qui contiennent au moins un chiffre
*.{htm, html}	tous les fichiers dont l'extension est htm ou html
Test*.java	tous les fichiers dont le nom commence par un Test et possède une extension .java

NB : L'interface **PathMatcher** (existant depuis java 7) comporte une unique méthode "*Boolean matches(Path path)*". Elle peut donc être utilisée au sein de "lambda expression" depuis java 8.

Exemple (fonctionnant depuis java8) :

...
....

4. Classe utilitaire Files (*"helper" avec 50 méthodes statiques*)

4.1. Vérifications sur fichiers ou répertoires

Méthode	Rôle
boolean exists (Path)	vérifier l'existence sur le système de fichiers de l'élément dont le chemin est encapsulé dans le paramètre de type Path fourni
boolean notExists (Path)	
boolean isReadable (Path path)	peut être lu (droits en lecture) ?
boolean isWritable (Path path)	peut être modifié (droits en écriture) ?
boolean isHidden (Path path)	est caché ?
boolean isExecutable (Path path)	est exécutable ?
boolean isRegularFile (Path path)	est un fichier ?
boolean isDirectory (Path path)	est un répertoire ?
boolean isSymbolicLink (Path path)	est un lien symbolique ?

String probeContentType (Path path)	retourne type MIME d'un fichier (ex: "text/plain") (ou null si indéterminé). S'appuie par défaut sur l'OS sous jacent.
--	--

Exemple:

```
Path path = Paths.get("C:/tmp/aa/fl.txt");
if(Files.isRegularFile(path)){
    System.out.println(path + " est un chemin vers un fichier");
}
```

4.2. Création d'un fichier ou d'un répertoire

Méthode	Rôle ou bien exemple
Path createFile (Path path, FileAttribute<?>... attrs)	Créer un fichier dont le chemin est encapsulé par l'instance de type Path fournie en paramètre
Path createDirectory (Path dir, FileAttribute<?>... attrs)	Path monRepertoire = Paths.get("C:/temp/mon_repertoire"); Files.createDirectory (monRepertoire);
Path createDirectories (Path dir, FileAttribute<?>... attrs)	Créer d'un seul coup plusieurs niveaux de sous répertoire selon le chemin exprimé "C:/temp/ niveau1/niveau2/niv3 "
Path createTempDirectory (Path dir, String prefix , FileAttribute<?>... attrs) Path createTempDirectory (String prefix , FileAttribute<?>... attrs)	Créer un répertoire temporaire de nom " prefixe indiqué " + numéro_calculé_par_syst (ex : rep_1245643) au sein du répertoire indiqué ou (à défaut) au sein du répertoire système (par défaut) prévu pour les temporaires.
Path createTempFile (Path dir, String prefix , String suffix , FileAttribute<?>... attrs) Path createTempFile (String prefix , String suffix , FileAttribute<?>... attrs)	Créer un fichier temporaire de nom " prefixe indiqué " + numéro_calculé_par_syst + " suffixe indiqué " (ex : fic_1245643.txt) au sein du répertoire indiqué ou (à défaut) au sein du répertoire système (par défaut) prévu pour les temporaires. Le suffix peut éventuellement être à null .

Les **attributs** (de type *FileAttribute*) sont **facultatifs** (des valeurs par défaut existent).

Les attributs possibles seront étudiés ultérieurement .

NB : **createDirectory**() créer un seul niveau de sous répertoire à la fois (*contrairement à createDirectories*) .

Si le répertoire existe déjà , l'exception *FileAlreadyExists* est remontée.

Si le répertoire parent n'existe pas , *NoSuchFileException* est remontée.

4.3. Copie d'un fichier ou d'un répertoire

Méthode	Rôle / fonctionnalité
Path copy (Path source, Path target,	Copier un élément avec les options précisées (

CopyOption... options)	StandardCopyOption.REPLACE_EXISTING , StandardCopyOption.COPY_ATTRIBUTES)
long copy (<i>InputStream</i> in, <i>Path</i> target, CopyOption... options)	Copier tous les octets d'un flux de type <i>InputStream</i> vers un fichier
long copy (<i>Path</i> source, <i>OutputStream</i> out)	Copier tous les octets d'un fichier dans un flux de type <i>OutputStream</i>

NB : L'option pointue `LinkOption.NOFOLLOW_LINKS` permet de recopier si besoin un lien symbolique au bout du path indiqué (plutôt que de suivre le lien).

NB2: il est possible d'utiliser la méthode `copy()` sur un répertoire . Cependant, le répertoire sera créé sans que les fichiers contenus et .. ne soient eux aussi copiés .

Quoi que contienne le répertoire, la méthode `copy` ne crée qu'un répertoire vide. Pour copier le contenu du répertoire, il faut parcourir son contenu et copier chacun des éléments un par un.

4.4. Déplacement et suppression d'un fichier ou d'un répertoire

Méthode	Rôle
move (<i>Path</i> source, <i>Path</i> target, CopyOption... options)	Déplacer ou renommer un élément avec les options précisées (<code>StandardCopyOption.REPLACE_EXISTING</code> , <code>StandardCopyOption.ATOMIC_MOVE</code>)
void delete (<i>Path</i> path)	Supprimer un élément du système de fichiers (avec exception s'il n'existe pas ou si répertoire pas vide)
boolean deleteIfExists (<i>Path</i> path)	Supprimer un élément du système de fichiers s'il existe (sans exception s'il existe pas)

Si un déplacement efficace/performant demandé en mode "ATOMIC_MOVE" est impossible (par exemple déplacement de "`c:/repXy`" vers "`d:/repXy`") , une exception est alors remontée. On peut alors éventuellement ré-essayer sans l'option .

Des exceptions peuvent potentiellement remonter si un répertoire à déplacer n'est pas vide et que certains fichiers contenus sont en cours d'utilisation.

5. Parcours des éléments d'un répertoire

La solution de parcours proposée par NIO2 est plus performante que `java.io.File.list(...)` .

La méthode `newDirectoryStream()` de la classe utilitaire **Files** attend en paramètre un objet de type *Path* qui correspond au répertoire à parcourir et permet d'obtenir une instance de "**stream**" de type `DirectoryStream<Path>` (à parcourir avec un itérateur ou autre) .

Attention: il est très important d'invoquer la méthode `close()` de l'instance de type **DirectoryStream** pour libérer les ressources utilisées.

Exemple :

```
Path tmpAaPath = Paths.get("C:/tmp/aa");
DirectoryStream<Path> stream = null;
```

```
try {
    stream=Files.newDirectoryStream(tmpAaPath) ;
    Iterator<Path> iterator = stream.iterator() ;
    while(iterator.hasNext()) {
        Path p = iterator.next() ;
        System.out.println(p) ;
    }
}
catch(IOException ex){    ex.printStackTrace();
}
finally {
    try {stream.close();
    } catch (IOException e) {e.printStackTrace();
    }
}
}
```

Exemple amélioré et simplifié :

```
Path tmpAaPath = Paths.get("C:/tmp/aa");

//NB1 : le second paramètre facultatif de Files.newDirectoryStream()
// sert à filtrer les éléments à parcourir (sans besoin de préfixe glob:)

//NB2: L'interface DirectoryStream implémente hérite de Closable et
//le try(avec_ressource_implémentant_interface Closable)
//sera automatiquement associé à un finally implicite déclenchant .close()

try (DirectoryStream<Path> stream = Files.newDirectoryStream(tmpAaPath,"*.txt")){
    for(Path p : stream){
        System.out.println(p);
    }
}
catch(IOException ex){
    ex.printStackTrace();
}
}
```

On peut également paramétrer et utiliser un filtre spécifique lors du parcours :

```
Path tmpAaPath = Paths.get("C:/tmp/aa");
DirectoryStream.Filter<Path> littleSizeFilter = new DirectoryStream.Filter<Path>() {
    public static final long MEGABYTE = 1024*1024;
    @Override
    public boolean accept(Path element) throws IOException {
        return (Files.size(element) <= MEGABYTE );
    }
}; //fin de classe anonyme imbriquée implémentant DirectoryStream.Filter<Path>

try (DirectoryStream<Path> stream = Files.newDirectoryStream(tmpAaPath,littleSizeFilter)){
    for(Path p : stream){
        System.out.println(p);
    }
}
}
```


NB : ce code (java7) pourra être amélioré/simplifié via une lambda expression de java8 .

6. Parcours d'une hiérarchie de répertoires (visiteur)

La méthode **Files.walkFileTree()** permet de parcourir une (sous-)arborescence de répertoires en utilisant le **design pattern "visiteur"**. Ce type de parcours peut être utilisé pour rechercher, copier, déplacer, supprimer, ... des éléments de la hiérarchie parcourue.

Il faut écrire une classe qui implémente l'interface **java.nio.file.FileVisitor<T>**. Cette interface définit des méthodes qui seront des callbacks appelées lors du parcours de la hiérarchie.

Méthode	Rôle / fonctionnalité
FileVisitResult postVisitDirectory (T dir, IOException exc)	Le parcours sort d'un répertoire qui vient d'être parcouru ou une exception est survenue durant le parcours
FileVisitResult preVisitDirectory (T dir, BasicFileAttributes attrs)	Le parcours rencontre un répertoire, cette méthode est invoquée avant de parcourir son contenu
FileVisitResult visitFile (T file, BasicFileAttributes attrs)	Le parcours rencontre un fichier
FileVisitResult visitFileFailed (T file, IOException exc)	La visite d'un des fichiers durant le parcours n'est pas possible et une exception a été levée

Les méthodes de l'interface **FileVisitor** renvoient toutes une valeur qui appartient à l'énumération **FileVisitResult**. Cette valeur permet de contrôler le processus de parcours de l'arborescence :

- **CONTINUE** : poursuite du parcours
- **TERMINATE** : arrêt immédiat du parcours
- **SKIP_SUBTREE** : inhibe le parcours de la sous-arborescence.
- **SKIP_SIBLING** : inhibe le parcours des répertoires frères.

Exemple(s) à ajouter ici plus tard .

7. FileSystem (par défaut et "personnalisé")

7.1. Fabrique FileSystems

FileSystems est une **fabrique** (avec **méthodes statiques**) pour obtenir des objets **FileSystem**.

.getDefault() renvoie l'instance de type **FileSystem** qui encapsule le F.S. de la JVM.

.getFileSystem(fsUri) renvoie un **FileSystem** selon l'URI est fourni en paramètre.

.newFileSystem() surchargée permet de créer une instance spécifique de type **FileSystem** (cas pointu)

7.2. FileSystem

String separator = FileSystems.getDefault().**getSeparator()**; // / sous linux ou \ sous windows

```
Iterable<Path> dirs = FileSystems.getDefault().getRootDirectories();
for (Path name: dirs) {
    System.err.println(name); // C:\ , D:\ , ...
}
```

7.3. FileSystem spécifique

...

8. Lecture et écriture dans un fichier

8.1. Vue d'ensemble

Principaux apports de NIO et NIO2 :

IO	NIO	NIO2
Depuis Java 1.0 et 1.1	Depuis Java 1.4 (JSR 151)	Depuis Java 7 (JSR 203)
Synchrone bloquant	Synchrone non bloquant	Asynchrone non bloquant
File InputStream OutputStream Reader (Java 1.1) Writer (Java 1.1) Socket RandomAccessFile	FileChannel SocketChannel ServerSocketChannel (Charset, Selector, ByteBuffer)	Path AsynchronousFileChannel AsynchronousByteChannel AsynchronousSocketChannel AsynchronousServerSocketChannel SeekableByteChannel

8.2. Options sur l'ouverture d'un fichier

L'énumération **StandardOpenOption** implémente l'interface *OpenOption* et définit les options d'ouverture standard d'un fichier :

Valeur	Signification
APPEND	Si le fichier est ouvert en écriture alors les données sont ajoutées au fichier. Cette option doit être utilisée avec les options CREATE ou WRITE

CREATE	Créer un nouveau fichier s'il n'existe pas sinon le fichier est ouvert
CREATE_NEW	Créer un nouveau fichier : si le fichier existe déjà alors une exception est levée
DELETE_ON_CLOSE	Supprimer le fichier lorsque son flux associé est fermé : cette option est utile pour des fichiers temporaires
DSYNC	Demander l'écriture synchronisée des données dans le système de stockage sous-jacent (pas d'utilisation des tampons du système) ???
READ	Ouvrir le fichier en lecture
SPARSE	Indiquer au système que le fichier est clairsemé ce qui peut lui permettre de réaliser certaines optimisations si l'option est supportée par le système de fichiers (c'est notamment le cas avec NTFS)
SYNC	Demander l'écriture synchronisée des données et des métadonnées dans le système de stockage sous-jacent
TRUNCATE_EXISTING	Si le fichier existe et qu'il est ouvert en écriture alors il est vidé. Cette option doit être utilisée avec l'option WRITE
WRITE	Ouvrir le fichier en écriture

8.3. Lecture de l'intégralité d'un fichier / Files.readAllLines()

Lecture (en boucle) de toutes les lignes d'un fichier texte :

```
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
...
List<String> lignes = Files.readAllLines(
    FileSystems.getDefault().getPath("c:/tmp/aa/fl.txt"), StandardCharsets.UTF_8);
for (String ligne : lignes)
    System.out.println(ligne);
```

Lecture d'un bloc de tout un (petit) fichier binaire :

```
byte[] binaryContent = Files.readAllBytes(binaryFilePath);
```

XIV - Annexe – Lombok

1. Lombok

1.1. Fonctionnalités de lombok et principes de fonctionnement

Beaucoup de frameworks java nécessitent que l'on programme des classes java avec tout un tas de choses classiques mais répétitives (getter/setter , constructeurs , méthode toString() ,).

De façon à gagner du temps dans la programmation et à aérer le code source on peut éventuellement s'appuyer sur des annotations telles que @Getter , @Setter de la technologie lombok de façon à ce que les méthodes getXxx() et setXxx() soient automatiquement générées avec le code basique par défaut . Il est heureusement toujours possible de personnaliser quelques getters/setters .

Depuis le jdk 1.6 (et Pluggable Annotation Processing API (JSR 269)) , les annotations de rétention "SOURCE" , telles de celles de lombok sont détectées dans le code source à compiler et traitées automatiquement durant la phase de compilation à partir du moment où le classpath comporte les ".jar" des traitements associés aux annotations. Il n'est donc plus nécessaire de lancer explicitement l'utilitaire apt du jdk 1.5 .

Lorsque la compilation du code est gérée par la technologie maven, les annotations de lombok (@Getter , @Setter , ...) sont interprétées et traitées automatiquement (si la dépendance maven "lombok" est présente dans pom.xml) .

1.2. Dépendances "maven" pour lombok

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.16.10</version> <!-- or via spring boot version -->
</dependency>
```

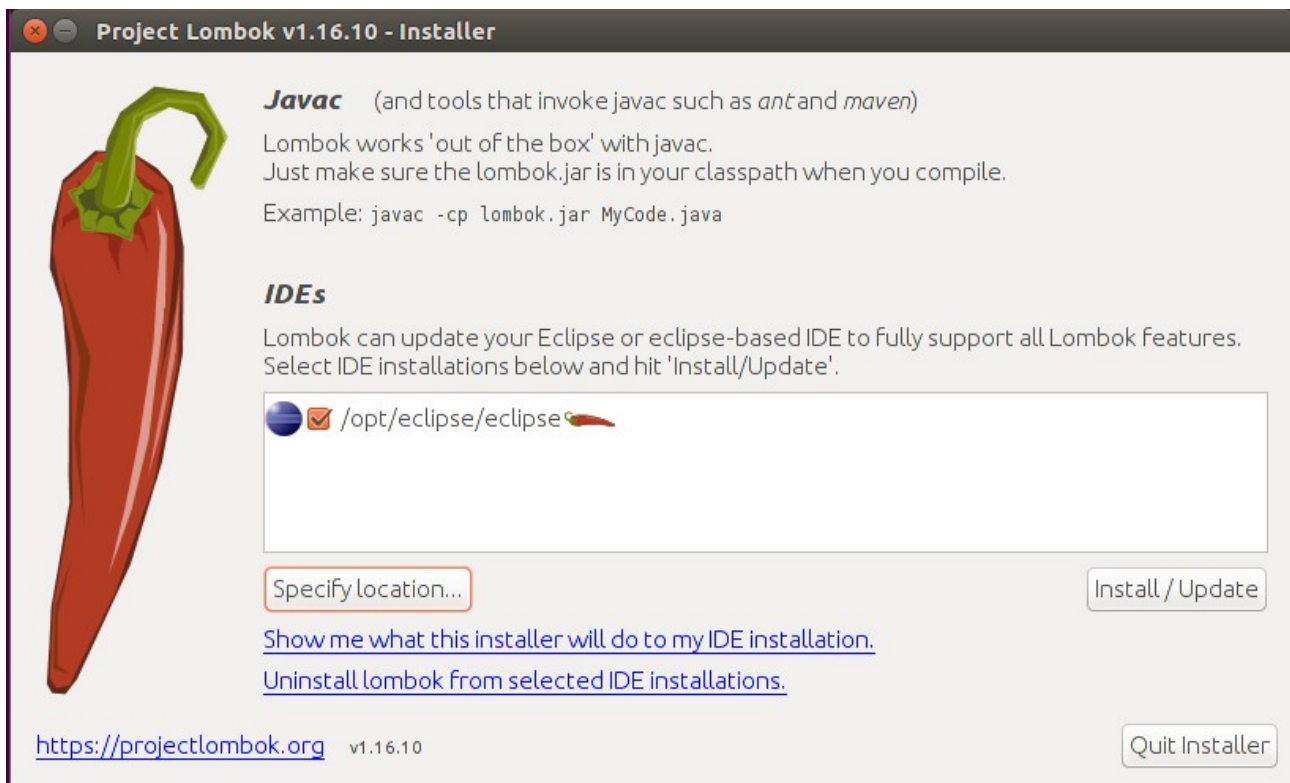
1.3. Installation de l'agent lombok dans un IDE (ex : eclipse)

Dans le cas particulier d'une compilation lancée par eclipse , une synchronisation doit être établie entre le code source et le code compilé (dès chaque enregistrement du code source modifié).

De façon à ce que les annotations de lombok soient bien prises en compte par l'IDE eclipse (ou autre), une configuration doit être effectuée. Celle-ci peut s'effectuer de la façon suivante :

~/m2/repository/org/projectlombok/lombok/1.16.10

\$ java -jar lombok-1.16.10.jar (ou double click sous windows)



→ un redémarrage de l'IDE (ex : eclipse) est nécessaire.

1.4. Paramétrages (@Getter, @Setter) et utilisation

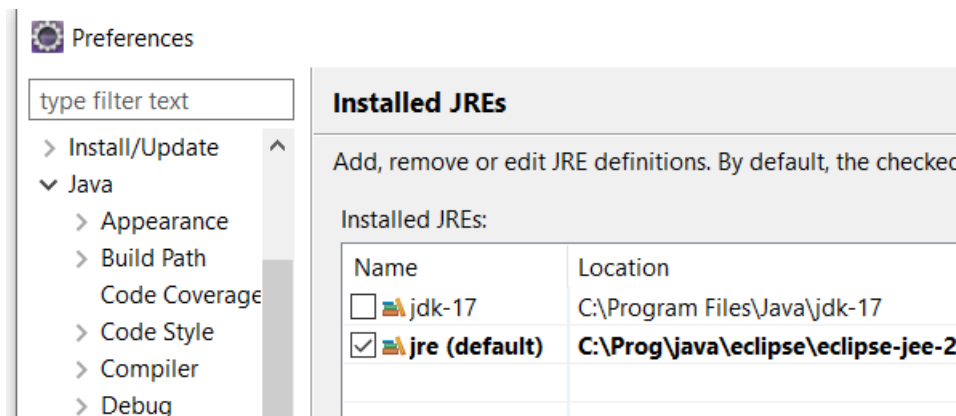
```
import lombok.AllArgsConstructor; import lombok.NoArgsConstructor;
import lombok.Getter; import lombok.Setter; import lombok.ToString;
//----- lombok generation code annotations -----
@Getter @Setter
@ToString
//@EqualsAndHashCode
@NoArgsConstructor @AllArgsConstructor
//-----
@Entity @Table(name="Customer")
public class Customer {
    @Id
    private Long id;
    private String name;
}
```

XV - Annexe – Bibliographie, Liens WEB + TP

1. énoncés de TP

1.1. Tp 0 : éléments logiciels à installer pour les Tps

- **jdk 17/TLS** (exemple : `jdk-17_windows-x64_bin.exe`
ou `jre17` intégré à `eclipse-jee-2021-12` ou à `intelliJ`)
- **eclipse jee 2021-12** (avec `jre17` et `maven` intégré et supportant `jdk 17`) ou bien équivalent (`intelliJ`)
exemple : https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2021-12/R/eclipse-jee-2021-12-R-win32-x86_64.zip
- **git** (exemple : `git-for-windows` + `tortoise-git`)
<https://github.com/git-for-windows/git/releases/download/v2.35.1.windows.2/Git-2.35.1.2-64-bit.exe> et <https://download.tortoisegit.org/tgit/2.13.0.0/TortoiseGit-2.13.0.1-64bit.msi>



URL (indicative , sujette à éventuelle changement) d'un référentiel git avec des "points de départ" pour les Tps :

git clone https://github.com/didier-tp/java_new_8_14

Puis sous eclipse "**file import ... / maven / import existing maven project**" et sélectionner `java_new_8_14/tp_java_8_14` comportant `pom.xml` .

Eventuels réglages de l'IDE (ex : eclipse) vis à vis de lombok

Se placer dans le répertoire `C:\Users\xyz\.m2\repository\org\projectlombok\lombok\1.18.22`

Double cliquer sur `lombok.1.18.22.jar`

Spécifier le répertoire d'installation d'eclipse et cliquer sur install

Redémarrer eclipse pour les modifications de `eclipse.ini` soient prises en compte.

NB :

- pour une formation "nouveau de java 8 à 17" on pourra éventuellement effectuer tous les Tps.
- pour une formation "**nouveautés de java 9 à 17**" on pourra ne pas effectuer les Tps n° 1 , 2, 3, 4 et 6, 7, 8.

1.2. Tp 1 : simplification et ré-écriture de code avec des "lambda"

1. Analyser et faire fonctionner l'exemple `tp.sans_lambda.AppGestionProduitsSansLambda` s'appuyant partiellement en interne sur `tp.util.ProductUtil`
2. Coder et faire fonctionner un équivalent avec "lambda" expressions `tp.avec_lambda.AppGestionProduitsAvecLambda`

NB : décommenter les affichages `System.out.println(...)` au fur et à mesure du code écrit pour tester

1.3. Tp 2 : prog. fonctionnelle synchrone avec des "streams"

- Compléter le code de la classe `tp.avec_lambda.AppGestionProduitsAvecStream` pour tester tri , filtrage , transformation via des streams et expérimenter divers collecteurs.
- Bien tester
- Analyser l'exemple `tp.avec_lambda.AppExemplesStreams` et le faire fonctionner
- Effectuer éventuellement d'autres expérimentations libres sur les streams

1.4. Tp 3 : utilisation de Optional<T>

- Compléter le code de la classe `tp.avec_lambda.AppWithOptional` en utilisant `Optional<T>`.
- 1 : faire fonctionner le point de départ sans `Optional`
2 : coder la méthode `displayWithOptional()` selon les commentaires et relancer le `main()` pour tester
3 : coder la méthode `displayWithOptionalAndLambda()` selon les commentaires et relancer le `main()`

1.5. Tp 4 : exploitation de l'api LocalDate , LocalTime, ...

1. Analyser et faire fonctionner l'exemple `tp.date.WithoutLocalDate`
2. Coder et faire fonctionner un équivalent avec "LocalDate," dans `tp.date.WithLocalDate`
3. Coder de nouvelles fonctionnalités (selon les commentaires) dans `tp.date.WithLocalDate`

1.6. Tp 5 : restructuration de projet avec des modules

1. Charger le projet java/maven multi-modules "**mod_mvsn**" de manière à analyser la structure complète du code en exemple dans le support de cours .

Analyser tous les fichiers *module-info.java*

Exécuter le main de la classe principale *tp.mod.mod_yy.app.MyApp*

Lancer les scripts de mod_mvn/scripts avec un "maven install" sur le projet "mod_mvn"

2. Charger le projet java/maven multi-modules "**my_mods**" (de structure similaire).
Exécuter le main de la classe principale *tp.mod.main.MyApp*
Restructurer les packages (en séparant les sous parties .impl)
Déplacer certaines parties du code vers les autres sous projets
et configurer les fichiers *module-info.java* (à créer/ajouter)
Relancer le main de la classe principale *tp.mod.main.MyApp*
Compléter et lancer les scripts de my_mods/scripts avec un "maven install" sur le projet
"my_mods"
Autres éventuelles restructurations libres (ex : services ou ...)

1.7. Tp 6 : bases de concurrent api (ExecutorService , ...)

Analyser et lancer dans l'ordre suivant les différents tests suivants pour accompagner la compréhension des éléments techniques du package java.util.concurrent et de ExecutorService :

- tp.concurrent.basic.EssaiExecutors
- tp.concurrent.basic.EssaiFuture
- tp.concurrent.basic.EssaiSemaphore
- tp.concurrent.basic.EssaiCompletionService

1.8. Tp 7 : application de "fork and join"

1. Au sein du package "tp.thread" , Analyser l'exemple constitué par les interfaces SequentialComputing , ResultAggregate) et les classes (MyStatsAlgo , MyStatsMultiProc , TestMultiProcesseurs).
Lancer le main() de TestMultiProcesseurs .
2. **En s'inspirant de l'exemple sophistiqué précédent , coder (plus simplement) un calcul de maximum via fork/join** . On pourra partir du code partiel des classes (MyBasicAlgo , MyBasicMultiProc , BasicTestMultiProcesseurs) et le compléter .
Lancer ensuite le main() de BasicTestMultiProcesseurs .

1.9. Tp 8 : CompletableFuture et prog. fonctionnelle asynchrone

Créer si besoin le sous répertoire vide "**input**" sous le répertoire "**files**" du projet .

Analyser le code de la classe **tp.util.MyFileConsumer** .

Analyser le code de la méthode *initSampleProductListByCategory(...)* de **tp.util.ProductUtil** .

Analyser et compléter le code de la classe
tp.concurrent.completable.**ProductAsyncWithCompletableFuture** .

Effectuer si besoin des "refresh" eclipse sur le répertoire **files/input** entre 2 essais .

Comportement attendu en recopiant **printer.txt** de files/to_copy vers files/input :

```
<<(end)waiting 2000ms before (re-)trying fetch category file in files/input / by
ForkJoinPool.commonPool-worker-3
.files\input\printer.txt
category=printer
Product [id=1, label=printer z1, price=156.8]
Product [id=6, label=printer z2, price=357.9]
Product [id=7, label=printer zz3, price=251.3]
<<(end)pause pour eviter arrêt complet du programme avant la fin des taches de fond / by main
fin main / interpreted by main
```

Comportement attendu en recopiant **computer.txt** de files/to_copy vers files/input :

```
.files\input\computer.txt
category=computer
Product [id=3, label=computer x2, price=976.5]
Product [id=5, label=computer y6, price=896.83]
Product [id=8, label=computer z8, price=351.3]
```

Comportement attendu en ne recopiant aucun fichier vers files/input :

```
<<(end)waiting 2000ms before (re-)trying fetch category file in files/input / by
ForkJoinPool.commonPool-worker-3
category=other
Product [id=2, label=usb wire, price=6.8]
Product [id=4, label=webcam, price=96.83]
```

car other = category par défaut si pas choisie après 20 essais de 2s .

1.10. Tp 9 : manipulations de "Reactive Streams"

Analyser le code des classes du package **tp.langage.flow** :
(exemples du cours + JsonProcessor)

Analyser le code de la classe **tp.langage.flow.FromFilePublisherApp** .

Effectuer un premier test en faisant en sorte que **main()** appelle **sansProcesseur()** ;

Effectuer si besoin des "refresh" eclipse sur le répertoire **files/input** entre 2 essais .
Mode opératoire (après avoir lancer **FromFilePublisherApp.main()**) :
recopier le fichier **p1.json** ou bien **p2.json** ou bien **p3.json**
de **files/to_copy** vers **files/input**

Comportement attendu :

```
s1>> Received subscription notification thread:ForkJoinPool.commonPool-worker-5
s2>> Received subscription notification thread:ForkJoinPool.commonPool-worker-3
.files\input\p1.json
```

```
s2>> Received item: { "id" : 999 , "label" : "produit999" , "price" : 99.99 , "features" : "aime les oeufs" }
thread:ForkJoinPool.commonPool-worker-5 nbTotalRequest=4 nbTotalReceived=1
s1>> Received item: { "id" : 999 , "label" : "produit999" , "price" : 99.99 , "features" : "aime les oeufs" }
thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=4 nbTotalReceived=1
.\files\input\p2.json
s2>> Received item: { "id" : 888 , "label" : "produit888" , "price" : 88.88 , "features" : "aime les huit" }
thread:ForkJoinPool.commonPool-worker-3 nbTotalRequest=4 nbTotalReceived=2
s1>> Received item: { "id" : 888 , "label" : "produit888" , "price" : 88.88 , "features" : "aime les huit" }
thread:ForkJoinPool.commonPool-worker-5 nbTotalRequest=4 nbTotalReceived=2
s2>> complete thread:ForkJoinPool.commonPool-worker-3
s1>> complete thread:ForkJoinPool.commonPool-worker-7
```

Modifier FromFilePublisherApp en faisant en sorte que main() appelle maintenant avecEtSansProcesseur();

Compléter le code de la méthode avecEtSansProcesseur()

Comportement attendu :

Un des abonnés "s_json" récupère et affiche les personnes au format json string

L'autre abonné "s_java" récupère et affiche les personnes transformées en objets java via le JsonProcessor .

```
s_java>> Received subscription notification thread:ForkJoinPool.commonPool-worker-3
s_json>> Received subscription notification thread:ForkJoinPool.commonPool-worker-7
.\files\input\p1.json
s_json>> Received item: { "id" : 999 , "label" : "produit999" , "price" : 99.99 , "features" : "aime les oeufs" }
thread:ForkJoinPool.commonPool-worker-9 nbTotalRequest=4 nbTotalReceived=1
s_java>> Received item: Product [id=999, label=produit999, price=99.99] thread:ForkJoinPool.commonPool-
worker-9 nbTotalRequest=4 nbTotalReceived=1
.\files\input\p3.json
s_json>> Received item: { "id" : 777 , "label" : "produit777" , "price" : 77.77 , "features" : "sept sept sept" }
thread:ForkJoinPool.commonPool-worker-7 nbTotalRequest=4 nbTotalReceived=2
s_java>> Received item: Product [id=777, label=produit777, price=77.77] thread:ForkJoinPool.commonPool-
worker-9 nbTotalRequest=4 nbTotalReceived=2
JsonProcessor completed
s_json>> complete thread:ForkJoinPool.commonPool-worker-3
s_java>> complete thread:ForkJoinPool.commonPool-worker-7
```

1.11. Tp 10 : quelques manipulations de JShell

Effectuer quelques manipulation libres (selon l'inspiration du moment) avec la console interactive jshell en s'inspirant des exemples syntaxiques du support de cours (chapitre "JShell") .

1.12. Tp 11: utilisation de l'api Process

Cloner si besoin via git le référentiel https://github.com/didier-mycontrib/java_8_9_1x

Charger le projet maven "myProcessApp" dans eclipse (ou un autre IDE java).

Faire fonctionner les exemples du cours en switchant de sous fonction appelée par ProcessApiApp.main() .

Autres expérimentations libres selon idées et envies ...

1.13. Tp 12: utilisation de l'api HttpClient

Au sein du projet principal "tp_java_8_14", se placer dans le package `tp.j9_10_11`.
Ajuster la méthode `MyNew9_10_11TestApp.main()` de façon à ce qu'elle n'appelle que `test_new_http2_client_since_java9_standard_since_java11()`;

Analyser le code et faire fonctionner l'exemple .

Analyser et tester ensuite `test_new_httpClient_withSubscriber()`;

1.14. Tp 13: utilisation de var et d'autres apports de java 10 et 11

Analyser et tester les autres parties de la classe exemple `tp.j9_10_11.MyNew9_10_11TestApp`

1.15. Tp 14: nouvelles fonctionnalités de java 12 /13/14 (switch, ...)

Analyser et tester le code de la classe exemple `tp.j14.TestNewSwitchApp`

1.16. Tp 15: nouvelles fonctionnalités de java 15 /16 /17

Analyser et tester le code des classes exemples suivantes :

`tp.j15_16_17.TestPatternMatchingInstanceOfApp`

`tp.j15_16_17.TestTextBlocApp`

`tp.j15_16_17.TestRecordApp`

1.17. Tp 16: synthèse des principales nouvelles syntaxes

Compléter le code de `tp.j15_16_17.SyntheseTestApp` en utilisant au maximum les nouvelles syntaxes des versions 8,9,10,11,12,13,14,15,16,17 de java

Phase 1 :

switch/case avec lambda expression au sein de `tp.j15_16_17.SyntheseTestApp`

Résultat attendu :

LIGNE

RECTANGLE

CERCLE

LIGNE
TYPE DE FIGURE INCONNU

Phase2 :

switch/case avec lambda expression et yield au sein de `tp.j15_16_17.SyntheseTestApp`

Résultat attendu :

LIGNE
resTransformation=Line from Figure(type=line, x1=10, y1=10, x2=100, y2=100, color=red)
RECTANGLE
resTransformation=Rectangle from Figure(type=rectangle, x1=5, y1=5, x2=120, y2=120, color=blue)
CERCLE
resTransformation=Circle from Figure(type=circle, x1=40, y1=40, x2=80, y2=80, color=green)
LIGNE
resTransformation=Line from Figure(type=line, x1=10, y1=10, x2=100, y2=10, color=black)
TYPE DE FIGURE INCONNU
resTransformation=Unknown from Figure(type=ellipse, x1=10, y1=10, x2=100, y2=30, color=orange)

Phase3 :

1. lire les **indications** dans les méthodes figureTo...() de `tp.util.FigureUtil`
2. **coder** les **...Record** nécessaires au sein de `tp.j15_16_17.Dto`
3. **coder les méthodes figureTo...() de tp.util.FigureUtil** en utilisant le mot clef var si besoin de variables locales
4. relancer `tp.j15_16_17.SyntheseTestApp.main()`

Résultat attendu :

LIGNE
resTransformation=LineRecord[x1=10, y1=10, x2=100, y2=100, lineColor=red]
RECTANGLE
resTransformation=RectangleRecord[x1=5, y1=5, largeur=115, hauteur=115, fillColor=blue]
CERCLE
resTransformation=CircleRecord[xc=60, yc=60, radius=28, fillColor=green]
LIGNE
resTransformation=LineRecord[x1=10, y1=10, x2=100, y2=10, lineColor=black]
TYPE DE FIGURE INCONNU
resTransformation=FigureRecord[type=ellipse, x1=10, y1=10, x2=100, y2=30, color=orange]

Phase4 :

- lire les **indications** dans les méthodes figureTo...() de `tp.j15_16_17.Dto`
- au niveau de chaque record , redéfinir la méthode toString() en s'appuyant sur les **TextBlocs** et de manière à générer des chaînes de caractères au format JSON.
- **relancer** `tp.j15_16_17.SyntheseTestApp.main()`

résultat attendu :

LIGNE

```
resTransformation={  
  "x1" : 10,  
  "y1" : 10,  
  "x2" : 100,  
  "y2" : 100,  
  "lineColor" : "red"  
}
```

RECTANGLE

```
resTransformation={  
  "x1" : 5,  
  "y1" : 5,  
  "largeur" : 115,  
  "hauteur" : 115,  
  "fillColor" : "blue"  
}
```

CERCLE

```
resTransformation={  
  "xc" : 60,  
  "yc" : 60,  
  "radius" : 28,  
  "fillColor" : "green"  
}
```

LIGNE

```
resTransformation={  
  "x1" : 10,  
  "y1" : 10,  
  "x2" : 100,  
  "y2" : 10,  
  "lineColor" : "black"  
}
```

TYPE DE FIGURE INCONNU

```
resTransformation={  
  "type" : "ellipse",  
  "x1" : 10,  
  "y1" : 10,  
  "x2" : 100,  
  "y2" : 30,  
  "color" : "orange"  
}
```

1.18. TP 17: expérimentation "sealed" et "pattern matching"

1. Effectuer temporairement des basculement de commentaires en haut des classes "AnimalDomestique" et "Chien" de manière à ce que ces classes ne soient plus scellées.
2. Etudier l'impact de ce changement au niveau de la méthode ***getTypeAnimalDomesticAsString()*** de la classe ***TestPatternMatchingSwitchPreviewApp***.
3. Ajouter la partie default manquante ou bien re-sceller les classes "AnimalDomestique" et "Chien" .
4. Eventuels autres expérimentations libres (selon l'inspiration du moment).

1.19. TP 18: Exo synthese (HttpClient + Record + Json/Jackson)

Compléter le code de la classe `tp.j15_16_17.HttpJsonRecordTestApp` selon les indications suivantes :

Phase 1 :

Via l'api **HttpClient** , appeler en mode **GET** le web service REST dont l'URL est ***https://catfact.ninja/fact***

Ce WS facile à appeler (sans api_key) retourne une réponse au format **JSON** de type ***{ "fact" : "un fait sur les chats" , "length" : 21_ou_autre }***

On pourra effectuer l'appel en mode synchrone ou bien en asynchrone via `client.sendAsync(req, BodyHandlers.ofString()).thenAccept(resp -> ...);` si mode asynchrone choisie prévoir pause d'attente en fin du `main()`

Phase 2 :

Via une structure de code de ce type :

```
client.sendAsync(req, BodyHandlers.ofString())
    .thenApply(resp -> {
        System.out.println(.....);
        System.out.println(.....);
        return resp.body();
    })
    .thenApply(*****lambda transformant jsonString en Dto.CatFact*****)
    .thenAccept(*****lambda affichant le record de type Dto.CatFact*****);
```

on pourra:

- transformer la réponse du format "jsonString" vers le format instance de **DTO.CatFact** (record à coder avec propriétés `.fact` et `.length`) en s'appuyant sur l'api jackson databind en version `>= 2.12.5` gérant bien les "record"
catFact = jacksonObjectMapper.readValue(catFactAsJsonString, CatFact.class);
- afficher finalement les valeurs du DTO/record java construit via `... .toString()`

Exemple de résultats affichés dans la console:

suite synchrone interpreted by main

recuperation reponse asynchrone / interpreted by ForkJoinPool.commonPool-worker-1

reponse status:200

reponse uri:https://catfact.ninja/fact

reponse type:[application/json]

reponse text:{"fact":"Approximately 1/3 of cat owners think their pets are able to read their minds.,"length":78}

catFact as java record:CatFact[fact=Approximately 1/3 of cat owners think their pets are able to read their minds., length=78]

fin synchrone / interpreted by main