
50
nouvelles choses que l'on peut faire avec
Java
8
@JosePaumard

José PAUMARD

MCF Um. Paris 13

PhD Appl C.S.

Open source dev.

Independant

José PAUMARD

Open source dev.

Independant

José PAUMARD

Paris JUG

Devoxx FRANCE

Questions ?

#50new8

@JosePaumard

Date

Date : Instant

Un instant est un point de la ligne du temps

```
Instant start = Instant.now() ;

Instant end = Instant.now() ;
```

Date : Duration

Une « duration » est une durée

```
Instant start = Instant.now() ;

Instant end = Instant.now() ;
```

```
Duration elapsed = Duration.between(start, end) ;
long millis = elapsed.toMillis() ;
```

Date : Duration

On peut faire des calculs sur les « durations »

```
Instant start = Instant.now() ;

Instant end = Instant.now() ;
```

```
Duration elapsed = Duration.between(start, end) ;
long millis = elapsed.toMillis() ;
```

```
elapsed.plus(2L, TemporalUnit.SECONDS) ;
```

Date : LocalDate

Une LocalDate est une date empirique

```
LocalDate now = LocalDate.now() ;

LocalDate shakespeareDoB =
   LocaleDate.of(1564, Month.APRIL, 23) ;
```

Date : Period

Une Period est une durée entre LocalDate

```
LocalDate now = LocalDate.now() ;

LocalDate shakespeareDoB =
   LocalDate.of(1564, Month.APRIL, 23) ;

Period p = shakespeareDoB.until(now) ;
System.out.println("# years = " + p.getYears()) ;
```

```
> # years = 449
```

Date : Period

Une Period est une durée entre LocalDate

```
LocalDate now = LocalDate.now() ;

LocalDate shakespeareDoB =
   LocalDate.of(1564, Month.APRIL, 23) ;

Period p = shakespeareDoB.until(now) ;
System.out.println("# years = " + p.getYears()) ;
```

```
long days = shakespeareDoB.until(now, ChronoUnit.DAYS) ;
System.out.println("# days = " + days) ; // 164_354
```

Date : TemporalAdjuster

Permet de trouver une date à partir d'une autre

```
LocalDate now = LocalDate.now() ;

LocalDate nextSunday =
   now.with(TemporalAdjuster.next(DayOfWeek.SUNDAY)) ;
```

Date : TemporalAdjuster

Permet de trouver une date à partir d'une autre

```
LocalDate now = LocalDate.now() ;

LocalDate nextSunday =
   now.with(TemporalAdjuster.next(DayOfWeek.SUNDAY)) ;
```

14 méthodes statiques dans la boite à outils

firstDayOfMonth(), lastDayOfYear()

firstDayOfNextMonth()

Date : TemporalAdjuster

Permet de trouver une date à partir d'une autre

```
LocalDate now = LocalDate.now() ;

LocalDate nextSunday =
   now.with(TemporalAdjuster.next(DayOfWeek.SUNDAY)) ;
```

14 méthodes statiques dans la boite à outils

firstInMonth(DayOfWeek.MONDAY)

next(DayOfWeek.FRIDAY)

Date : LocalTime

Permet de coder une heure empirique : 10h20

```
LocalTime now = LocalTime.now() ;

LocalTime time = LocalTime.of(10, 20) ; // 10h20
```

Date : LocalTime

Permet de coder une heure empirique : 10h20

```
LocalTime now = LocalTime.now() ;

LocalTime time = LocalTime.of(10, 20) ; // 10h20
```

```
LocalTime lunchTime = LocalTime.of(12, 30) ;

LocalTime coffeeTime = lunchTime.plusHours(2) ; // 14h20
```

Date : ZonedTime

Permet de coder des heures localisées

```
Set<String> allZonesIds = ZoneId.getAvailableZoneIds() ;

String ukTZ = ZoneId.of("Europe/London") ;
```

Date : ZonedTime

Permet de coder des heures localisées

```
System.out.println(
   ZonedDateTime.of(
      1564, Month.APRIL.getValue(), 23, // year / month / day
      10, 0, 0, 0,                      // h / mn / s / nanos
      ZoneId.of("Europe/London"))
); // prints 1564-04-23T10:00-00:01:15[Europe/London]
```

Date : ZonedTime

On peut faire des calculs sur les heures localisées

```
ZonedDateTime currentMeeting =
   ZonedDateTime.of(
      LocalDate.of(2014, Month.APRIL, 18), // LocalDate
      LocalTime.of(9, 30),                 // LocalTime
      ZoneId.of("Europe/London")
) ;

ZonedDateTime nextMeeting =
   currentMeeting.plus(Period.ofMonth(1)) ;
```

Date : ZonedTime

On peut faire des calculs sur les heures localisées

```
ZonedDateTime currentMeeting =
   ZonedDateTime.of(
      LocalDate.of(2014, Month.APRIL, 18), // LocalDate
      LocalTime.of(9, 30),                 // LocalTime
      ZoneId.of("Europe/London")
) ;

ZonedDateTime nextMeeting =
   currentMeeting.plus(Period.ofMonth(1)) ;
ZonedDateTime nextMeetingUS =
   nextMeeting.withZoneSameInstant(ZoneId.of("US/Central")) ;
```

Date : Formattage

Classe utilitaire : DateTimeFormatter

```
ZonedDateTime nextMeetingUS =
   nextMeeting.withZoneSameInstant(ZoneId.of("US/Central"));

System.out.println(
   DateTimeFormatter.ISO_DATE_TIME.format(nextMeetingUS)
);
// prints 2014-04-12T03:30:00-05:00[US/Central]

System.out.println(
   DateTimeFormatter.RFC_1123_DATE_TIME.format(nextMeetingUS)
);
// prints Sat, 12 Apr 2014 03:30:00 -0500
```

Date : liens avec java.util.Date

Classe utilitaire : DateTimeFormatter

```
Date date = Date.from(instant);               // legacy -> new API
Instant instant = date.toInstant();           // API -> legacy
```

Date : liens avec java.util.Date

Classe utilitaire : DateTimeFormatter

```
Date date = Date.from(instant);             // legacy -> new API
Instant instant = date.toInstant();         // API -> legacy
```

```
TimeStamp time = TimeStamp.from(instant); // legacy -> new API
Instant instant = time.toInstant();       // API -> legacy
```

Date : liens avec java.util.Date

Classe utilitaire : DateTimeFormatter

```
Date date = Date.from(instant);              // legacy -> new API
Instant instant = date.toInstant();          // API -> legacy
```

```
TimeStamp time = TimeStamp.from(instant); // legacy -> new API
Instant instant = time.toInstant();       // API -> legacy
```

```
Date date = Date.from(localDate);            // legacy -> new API
LocalDate localDate = date.toLocalDate(); // API -> legacy
```

Date : liens avec java.util.Date

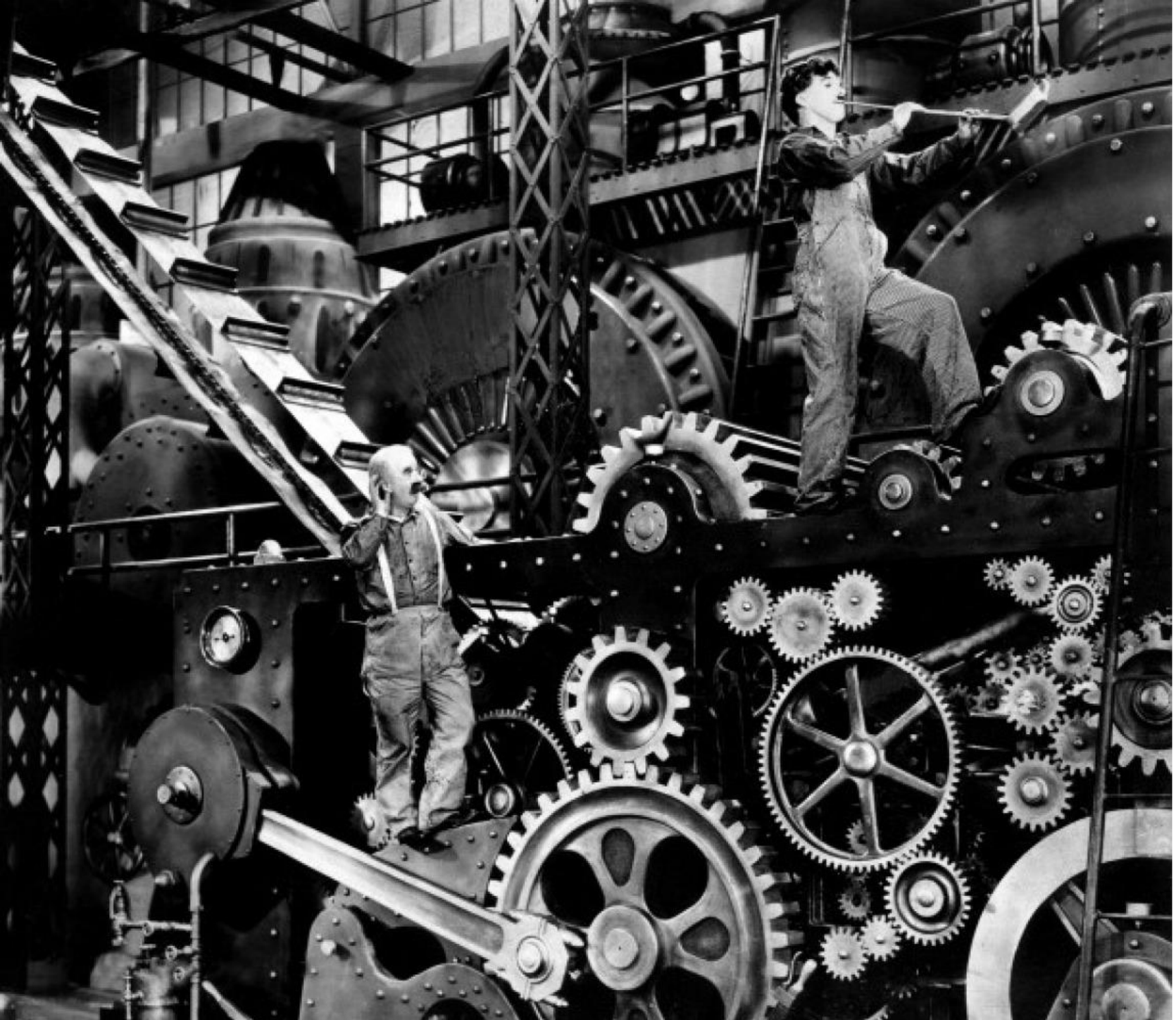
Classe utilitaire : DateTimeFormatter

```
Date date = Date.from(instant);               // legacy -> new API
Instant instant = date.toInstant();           // API -> legacy
```

```
TimeStamp time = TimeStamp.from(instant); // legacy -> new API
Instant instant = time.toInstant();       // API -> legacy
```

```
Date date = Date.from(localDate);          // legacy -> new API
LocalDate localDate = date.toLocalDate(); // API -> legacy
```

```
Time time = Time.from(localTime);          // legacy -> new API
LocalTime localTime = time.toLocalTime(); // API -> legacy
```



String

String : Stream

Un stream sur les lettres qui composent une String

```
String s = "bonjour" ;
IntStream stream = s.chars() ;

stream.forEach(Sytem.out::println) ;
```

String : Stream

Un stream sur les lettres qui composent une String

```
String s = "bonjour" ;
IntStream stream = s.chars() ;

stream
    .map(String::toUpperCase)
    .forEach(Sytem.out::print) ;
```

Affiche :

```
> BONJOUR
```

String : expressions régulières

Construction de Stream à partir d'une regexp

```
// book est une grrrande chaîne
Stream<String> words =
    Pattern
        .compile("[^\\p{javaLetter}]")
        .splitAsStream(book) ;
```

String : concaténation

Le naïf écrit :

```
String s1 = "bonjour" ;
String s2 = "le monde" ;

String s3 = s1 + " " + s2 ;
```

String : concaténation

L’ignorant lui dit d’écrire :

```
StringBuilder sb1 = new StringBuilder("bonjour") ;
sb1.append(" le monde") ;

String s3 = sb1.toString() ;
```

String : concaténation

L'ignorant lui dit d'écrire :

```
String s1 = "bonjour" ;
String s2 = "le monde" ;
```

```
LINENUMBER 10 L2
   NEW java/lang/StringBuilder
   DUP
   ALOAD 1
   INVOKESTATIC java/lang/String.valueOf(Ljava/lang/Object;)Ljava/lang/String;
   INVOKESPECIAL java/lang/StringBuilder.<init>(Ljava/lang/String;)V
   LDC " "
   INVOKEVIRTUAL java/lang/StringBuilder.append(Ljava/lang/String;)Ljava/lang/StringBuilder;
   ALOAD 2
   INVOKEVIRTUAL java/lang/StringBuilder.append(Ljava/lang/String;)Ljava/lang/StringBuilder;
   INVOKEVIRTUAL java/lang/StringBuilder.toString()Ljava/lang/String;
   ASTORE 3
```

String : concaténation

Le spécialiste Java 8 écrit

```
// The JDK 8 way
StringJoiner sj = new StringJoiner(", ") ;
sj.add("one").add("two").add("three") ;
String s = sj.toString() ;
System.out.println(s) ;
```

String : concaténation

Le spécialiste Java 8 écrit

```
// The JDK 8 way
StringJoiner sj = new StringJoiner(", ") ;
sj.add("one").add("two").add("three") ;
String s = sj.toString() ;
System.out.println(s) ;
```

Ce qui affiche

```
> one, two, three
```

String : concaténation

Le spécialiste Java 8 écrit

```
// The JDK 8 way
StringJoiner sj = new StringJoiner(", ", "{", "}") ;
sj.add("one").add("two").add("three") ;
String s = sj.toString() ;
System.out.println(s) ;
```

Ce qui affiche

```
> {one, two, three}
```

String : concaténation

Le spécialiste Java 8 écrit

```
// The JDK 8 way
StringJoiner sj = new StringJoiner(", ", "{", "}") ;
// on ne met rien dedans
String s = sj.toString() ;
System.out.println(s) ;
```

Ce qui affiche

```
> {}
```

String : concaténation

S’utilise aussi à partir de String directement

```
// From the String class, with a vararg
String s = String.join(", ", "one", "two", "three");
System.out.println(s);
```

Ce qui affiche

```
> one, two, three
```

String : concaténation

S’utilise aussi à partir de String directement

```
// From the String class, with an Iterable
String [] tab = {"one", "two", "three"} ;
String s = String.join(", ", tab) ;
System.out.println(s) ;
```

Ce qui affiche

```
> one, two, three
```

I/O

I/O : lecture de fichiers texte

Stream implémente AutoCloseable

```
// Java 7 : try with resources and use of Paths
Path path = Paths.get("d:", "tmp", "debug.log");
try (Stream<String> stream = Files.lines(path)) {

   stream.filter(line -> line.contains("ERROR"))
         .findFirst()
         .ifPresent(System.out::println);

} catch (IOException ioe) {
   // handle the exception
}
```

I/O : lecture d'un répertoire

Files.list retourne les fichiers du répertoire

```
// Java 7 : try with resources and use of Paths
Path path = Paths.get("c:", "windows");
try (Stream<Path> stream = Files.list(path)) {

   stream.filter(path -> path.toFile().isDirectory())
         .forEach(System.out::println);

} catch (IOException ioe) {
   // handle the exception
}
```

I/O : lecture d'une arborescence

Files.walk retourne les fichiers du sous-arbre

```
// Java 7 : try with resources and use of Paths
Path path = Paths.get("c:", "windows");
try (Stream<Path> stream = Files.walk(path)) {

   stream.filter(path -> path.toFile().isDirectory())
         .forEach(System.out::println);

} catch (IOException ioe) {
   // handle the exception
}
```

I/O : lecture d'une arborescence

Files.walk retourne les fichiers du sous-arbre, profondeur

```
// Java 7 : try with resources and use of Paths
Path path = Paths.get("c:", "windows");
try (Stream<Path> stream = Files.walk(path, 2)) {

   stream.filter(path -> path.toFile().isDirectory())
         .forEach(System.out::println);

} catch (IOException ioe) {
   // handle the exception
}
```

List

Iterable : forEach

ForEach : itère sur tous les éléments, prend un consumer

```
// méthode forEach sur Iterable
List<String> strings =
   Arrays.asList("one", "two", "three") ;

strings.forEach(System.out::println) ;
```

Ne marche pas sur les tableaux

```
> one, two, three
```

Collection : removeIf

Retire un objet : prend un prédicat

```
// removes an element on a predicate
Collection<String> strings =
   Arrays.asList("one", "two", "three", "four");

// works « in place », no Collections.unmodifiable...
Collection<String> list = new ArrayList<>(strings);

// returns true if the list has been modified
boolean b = list.removeIf(s -> s.length() > 4);
```

```
> one, two, four
```

List : replaceAll

Remplace un objet par sa transformée

```
// removes an element on a predicate
Collection<String> strings =
   Arrays.asList("one", "two", "three", "four");

// works « in place », no Collections.unmodifiable...
Collection<String> list = new ArrayList<>(strings);

// returns nothing
list.replaceAll(String::toUpperCase);
```

```
> ONE, TWO, THREE, FOUR
```

List : sort

Tri une liste en place, prend un comparateur

```
// removes an element on a predicate
Collection<String> strings =
   Arrays.asList("one", "two", "three", "four");

// works « in place », no Collections.unmodifiable...
Collection<String> list = new ArrayList<>(strings);

// returns nothing
list.sort(Comparator.naturalOrder()) ;
```

```
> four, one, three, two
```

Parallel Arrays

Parallel Arrays

Arrays.parallelSetAll

```
long [] array = new long [...] ;

Arrays.parallelSetAll(array, index -> index % 3) ;

System.out.println(Arrays.toString(array)) ;
```

Parallel Arrays

Arrays.parallelPrefix : fold right

```
long [] array = new long [...] ;

Arrays.parallelPrefix(array, (l1, l2) -> l1 + l2) ;

System.out.println(Arrays.toString(array)) ;
```

```
long [] array = {1L, 1L, 1L, 1L} ;

> [1, 2, 3, 4]
```

Parallel Arrays

Arrays.sort : tri en place

```
long [] array = new long [...] ;

Arrays.parallelSort(array) ;

System.out.println(Arrays.toString(array)) ;
```

Comparator

Comparator !

Que dire de plus ?

```
Comparator.naturalOrder()
```

Comparator !

Que dire de plus ?

```
Comparator.naturalOrder()

public static
   <T extends Comparable<? super T>> Comparator<T> naturalOrder() {
      return (Comparator<T>)
          Comparators.NaturalOrderComparator.INSTANCE;
    }
```

Comparator !

```
enum NaturalOrderComparator
implements Comparator<Comparable<Object>> {

    INSTANCE;









}
```

Comparator !

```
enum NaturalOrderComparator
implements Comparator<Comparable<Object>> {

   INSTANCE;

   public int compare(Comparable<Object> c1, Comparable<Object> c2) {
      return c1.compareTo(c2);
   }




}
```

Comparator !

```
enum NaturalOrderComparator
implements Comparator<Comparable<Object>> {

   INSTANCE;

   public int compare(Comparable<Object> c1, Comparable<Object> c2) {
      return c1.compareTo(c2);
   }

   public Comparator<Comparable<Object>> reversed() {
      return Comparator.reverseOrder();
   }
}
```

Comparator !

Que dire de plus ?

```
Comparator.comparingBy(Person::getLastName)
          .thenComparing(Person::getFirstName)
          .thenComparing(Person::getAge)
```

Map

Map : forEach

Prend un BiConsumer

```
// the existing map
Map<String, Person> map = ... ;

map.forEach(
   (key, value) -> System.out.println(key + " -> " + value)
) ;
```

Map : replace

Remplace une valeur avec sa clé

```
// the existing map
Map<String, Person> map = ... ;

// key, newValue
map.replace("six", john) ;

// key, oldValue, newValue
map.replace("six", peter, john) ;
```

Map : replaceAll

Transforme toutes les valeurs

```
// the existing map
Map<String, Person> map = ... ;

// key, oldValue
map.replaceAll(
    (key, value) -> key + " -> " + value ;
) ;
```

Map : remove

Retire les paires clés / valeurs

```
// the existing map
Map<String, Person> map = ... ;

// key, oldValue
map.remove("six", john) ;
```

Map : compute

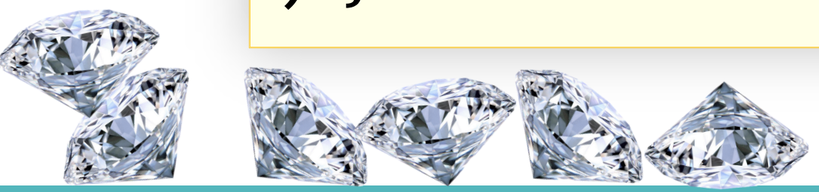
Calcule une valeur à partir de la clé et la valeur existante, et d'une fonction qui fusionne la paire clé / valeur

```
// the existing map
Map<String, Person> map = ... ;

// key, oldValue
map.compute(
    key,
    (key, value) -> key + "::" + value
) ;
```

Map : merge

Calcule une valeur à partir de la clé, de l'actuelle valeur si elle existe, et d'une fonction qui fusionne les valeurs

```
// the existing map
Map<String, Person> map = ... ;

// key, otherValue
map.merge(
    key,
    otherValue,
    (value, otherValue) -> value.concat(", ").concat(otherValue)
) ;
```



Map : putIfAbsent

Ajoute une paire clé / valeur si la clé n’est pas déjà dans la table

```
// the existing map
Map<String, Person> map = ... ;

// key, newValue
map.putIfAbsent("un", john) ;
```

Map : computeIfAbsent

Si la clé est absente : associe une valeur calculée par exécution de la fonction. Dans tous les cas : retourne la valeur (nouvelle ou ancienne)

```
// the existing map
Map<String, Map<String, Person>> map = ... ;

// key, newValue
map.computeIfAbsent(
    "un",
    key -> new HashMap<>()
)
.put(".un", john) ;
```

Map : computeIfPresent

Si la clé est présente : associe une valeur calculée par exécution de la fonction. Retourne la valeur.

```
// the existing map
Map<String, Map<String, Person>> map = ... ;

// key, newValue
map.computeIfPresent(
    "un", map,
    (key, value) -> ... // la nouvelle valeur
)
.put(".un", john) ;
```

Completable Future

CompletableFuture

Extension de Future

```
CompletableFuture<String> page =
    CompletableFuture.supplyAsync(



    ) ;
```

CompletableFuture

Extension de Future

```
CompletableFuture<String> page =
    CompletableFuture.supplyAsync(
        () ->
        readWebPage(url)
    ) ;
```

CompletableFuture

Permet de créer des pipelines

```
CompletableFuture.supplyAsync(
    () ->
    readWebPage(url)
)
.thenApply(content -> getImages(content)) ;
```

CompletableFuture

Permet de créer des pipelines

```
CompletableFuture<List<Image>> images =
   CompletableFuture.supplyAsync(
      () ->
      readWebPage(url)
   )
   .thenApply(content -> getImages(content)) ; // function
```

CompletableFuture

thenCompose : composition de tâches dans le futur

```
CompletableFuture<List<Image>> cf =
   CompletableFuture.supplyAsync(
      () ->
      readWebPage(url)
   )
   .thenCompose(content -> getImages(content))
```

CompletableFuture

thenCompose : composition de tâches dans le futur

```
CompletableFuture.supplyAsync(
    () ->
    readWebPage(url)
)
.thenCompose(content -> getImages(content))
.thenApply(image -> writeToDisk(image)) ; // retourne CF<Boolean>
```

CompletableFuture

thenCompose : composition de tâches dans le futur

```
List<CompletableFuture<Boolean>> result =
   CompletableFuture.supplyAsync(
      () ->
      readWebPage(url)
   )
   .thenCompose(content -> getImages(content))
   .thenApply(image -> writeToDisk(image)) ; // retourne CF<Boolean>
```

CompletableFuture

allOf : composition de tâches dans le futur (anyOf existe)

```
CompletableFuture.allOf(
    CompletableFuture.supplyAsync(
        () ->
        readWebPage(url)
    )
    .thenCompose(content -> getImages(content))
    .thenApply(image -> writeToDisk(image))
)
.join() ;
```

CompletableFuture

thenCombine : combine plusieurs CF

```
CompletableFuture cf1 = ... ;
CompletableFuture cf2 = ... ;

cf1.thenCombine(cf2, (b1, b2) -> b1 & b2) ; // retourne la combinaison
                                            // des résultats des CF
```

Applique la fonction une fois les deux CF exécutés

CompletableFuture

thenCombine : combine plusieurs CF

```
CompletableFuture cf1 = ... ;
CompletableFuture cf2 = ... ;

cf1.thenCombine(cf2, (b1, b2) -> b1 & b2) ; // retourne la combinaison
                                            // des résultats des CF
```

Applique la fonction une fois les deux CF exécutés

thenAcceptBoth, runAfterBoth

CompletableFuture

applyToEither : utilise le premier résultat disponible

```
CompletableFuture cf1 = ... ;
CompletableFuture cf2 = ... ;

cf1.applyToEither(cf2, (b) -> ...) ; // s'applique au résultat
                                     // du premier CF dispo
```

CompletableFuture

applyToEither : utilise le premier résultat disponible

```
CompletableFuture cf1 = ... ;
CompletableFuture cf2 = ... ;

cf1.applyToEither(cf2, (b) -> ...) ; // s'applique au résultat
                                     // du premier CF dispo
```

acceptEither, runAfterEither

Concurrence

Variables atomiques

On avait :

```
AtomicLong atomic = new AtomicLong() ;
long l1 = atomic.incrementAndGet() ;
```

Variables atomiques

On a :

```
AtomicLong atomic = new AtomicLong() ;
long l1 = atomic.incrementAndGet() ;

long l2 = atomic.updateAndGet(l -> l*2 + 1) ;
```

Variables atomiques

On a :

```
AtomicLong atomic = new AtomicLong() ;
long l1 = atomic.incrementAndGet() ;

long l2 = atomic.updateAndGet(l -> l*2 + 1) ;

long l3 = atomic.accumulateAndGet(12L, (l1, l2) -> l1 % l2) ;
```

LongAdder

On a :

```
LongAdded adder = new LongAdder() ;

adder.increment() ; // dans un thread
adder.increment() ; // dans un autre thread
adder.increment() ; // encore dans un autre thread

long sum = adder.sum() ;
```

LongAccumulator

Même chose, mais on généralise :

```
LongAccumulator accu =
    new LongAccumulator((l1, l2) -> Long.max(l1, l2), 0L) ;

accu.accumulate(value1) ; // dans un thread
accu.accumulate(value2) ; // dans un autre thread
accu.accumulate(value2) ; // encore dans un autre thread

long sum = accu.longValue() ;
```

StampedLock

Un lock avec lecture optimiste

```
StampedLock sl= new StampedLock() ;
```

```
long stamp = sl.writeLock() ;
try {
   ...
} finally {
   sl.unlockWrite(stamp) ;
}
```

```
long stamp = sl.readLock() ;
try {
   ...
} finally {
   sl.unlockRead(stamp) ;
}
```

StampedLock

Un lock avec lecture optimiste

```
StampedLock sl= new StampedLock() ;
```

```
long stamp = sl.writeLock() ;
try {
   ...
} finally {
   sl.unlockWrite(stamp) ;
}
```

```
long stamp = sl.readLock() ;
try {
   ...
} finally {
   sl.unlockRead(stamp) ;
}
```

Exclusivité entre read / write, mais…

StampedLock

Un lock avec lecture optimiste

```
StampedLock sl= new StampedLock() ;
```

```
long stamp = sl.tryOptimisticRead() ;
// ici on lit une variable qui peut être modifiée par un autre thread
if (lock.validate(stamp)) {
   // la lecture est validée
} else {
   // un autre thread a acquis un write lock
}
```

Concurrent HashMap

ConcurrentHashMap

Réécriture complète de ConcurrentHashMap V7

Complètement *thread-safe*

N’utilise de lock ≠ ConcurrentHashMap V7

Nouvelles méthodes

ConcurrentHashMap

6000 lignes de code

ConcurrentHashMap

6000 lignes de code

54 classes membre

ConcurrentHashMap

6000 lignes de code

54 classes membre

Pour info : 58 classes dans java.util.concurrent

ConcurrentHashMap

6000 lignes de code

54 classes membre

Pour info : 58 classes dans java.util.concurrent

Nouveaux patterns !

ConcurrentHashMap

Ne plus utiliser size()

```
int   count = map.size() ;          // ne pas utiliser

      count = map.mappingCount() ; // nouvelle méthode
```

ConcurrentHashMap

Ne plus utiliser

```
int   count = map.size() ;          // ne pas utiliser

long count = map.mappingCount() ; // nouvelle méthode
```

ConcurrentHashMap

Recherche d'éléments

```
ConcurrentHashMap<Integer, String> map = ... ;

map.search(10, (key, value) -> value.length() < key) ;
```

search(), searchKey(), searchValue(), searchEntry()

ConcurrentHashMap

Recherche d'éléments

```
ConcurrentHashMap<Integer, String> map = ... ;

map.search(10, (key, value) -> value.length() < key) ;
```

search(), searchKey(), searchValue(), searchEntry()

ConcurrentHashMap

Recherche d'éléments

```
ConcurrentHashMap<Integer, String> map = ... ;

map.search(10, (key, value) -> value.length() < key) ;
```

10 : taux de parallélisme

Si la table compte plus de 10 éléments, alors la recherche se fait en parallèle !

ConcurrentHashMap

Recherche d'éléments

```
ConcurrentHashMap<Integer, String> map = ... ;

map.search(10, (key, value) -> value.length() < key) ;
```

10 : taux de parallélisme

Si la table compte plus de 10 éléments, alors la recherche se fait en parallèle !

On peut passer 0, ou Integer.*MAX_VALUE*

ConcurrentHashMap

ForEach

```
ConcurrentHashMap<Integer, String> map = ... ;

map.forEach(10,
            (key, value) ->
            System.out.println(String.join(key, "->", value)
) ;
```

forEach(), forEachKey(), forEachEntries()

ConcurrentHashMap

Réduction

```
ConcurrentHashMap<Integer, String> map = ... ;

map.reduce(10,
           (key, value) -> value.getName(), // transformation
           (name1, name2) -> name1.length() > name2.length() ?
                             name1 : name2) // reduction
) ;
```

reduce(), reduceKey(), reduceEntries()

Pas de ConcurrentHashSet

Mais…

```
Set<String> set = ConcurrentHashMap.<String>.newKeySet() ;
```

Pas de ConcurrentHashSet

Mais…

```
Set<String> set = ConcurrentHashMap.<String>.newKeySet() ;
```

Crée une *concurrent hashmap* dont les valeurs sont Boolean.*TRUE*

Sert de set concurrent

Merci !


Q/R
#50new8
@JosePaumard