1. Predicate

```
public interface Predicate<T> { public abstract boolean test(T t); }
```

→ **interface fonctionnelle** : une seule méthode abstraite (eg. compare de Comparator)

2. StringJoiner

- `public StringJoiner(CharSequence delimiter, CharSequence prefix, CharSequence suffix)`
- impl. de `CharSequence` : `CharBuffer, Segment, String, StringBuffer, StringBuilder`
- `public StringJoiner merge(StringJoiner other)` (ajoute b à la fin de a en utilisant le délim. du récept.)
- `public StringJoiner add(CharSequence newElement)`

3. Optional

- `public static <T> Optional<T> of(T value)`
- `public T get()`
- `public T orElse(T other)`
- `public T orElseThrow(IllegalArgumentException::new)`

4. Stream

4.1. Méthodes d'instanciation

- `static <T> Stream<T> empty()`
- `static <T> Stream<T> of(T... values)`
- `static Stream<T> iterate(T i, UnaryOperator<T> f)` (retourne le flot infini des applications successives de l'opérateur f à la valeur initiale i ; la première valeur du flot est donc i !)
- `static <T> Stream<T> concat(Stream<? extends T> a, Stream<? extends T> b)`

4.2. Méthodes intermédiaires

- `Stream<T> limit(long/int l)` (prend les l premiers)
- `Stream<T> skip(long/int n)` (saute les n premiers)
- `Stream<T> sorted()` (croissant par def)
- `Stream<T> sorted(Comparator<T> c)`
- `Stream<T> filter(Predicate<T> p)`
- `Stream<R> map(Function<T,R> f)`
- `Stream<R> flatMap(Function<T,Stream<R>> f)`

4.3. Méthodes logiques

- `boolean allMatch(Predicate<T> p)`
- `boolean anyMatch(Predicate<T> p)`
- `boolean noneMatch(Predicate<T> p)`

8. List

- `of(E... elements)`
- `copyOf`
- `addAll(Collection<? extends E> c)`
- `addAll(int index, Collection<? extends E> c)`
- `isEmpty()`

5. Arrays

- `sort` (attention, renvoie void !)
- `asList` (attention, Arrays.asList sur un tableau de primitif ne se comporte pas comme prévu : `Arrays.asList(new int[]{1, 2, 3})` ne renverra pas une `List<Integer>` mais une `List<int[]>`.

6. Collections

- `unmodifiableSet`
- `unmodifiableList`
- `stream`

7. Collection

- `void clear()`
- `boolean remove(Object e)`
- `boolean removeAll(Collection<E> c)`
- `boolean removeIf(Predicate<E> p)` (supprime tous les éléments qui satisfont le prédicat donné)
- `boolean retainAll(Collection<E> c)` (supprime tous les éléments qui ne se trouvent pas dans la collection donnée)

4.5. Méthodes de consommation

- `void forEach(Consumer<T> c)`

4.6. Méthodes terminales

à ajouter avant chaque méthode : `.collect(Collectors` (p. ex `.collect(Collectors.joining())`)

- `joining()` /`joining(String delimiter)`/`joining(String delimiter, String prefix, String suffix)`
- `toList()`
- `toSet()`
- `toMap(Function<T,K> k, Function<T,V> v)`
- `groupingBy`

pas un collecteur :

- `T reduce(T identity, BinaryOperator<T> accumulator)` (l'accumulateur prend previous, current)

9. Map

Note : on peut construire une `TreeMap` en lui donnant un comparateur.

- `public TreeMap(Comparator<? super K> comparator)`
- `Set<K> keySet()` (renvoie une vue)
- `Collection<V> values()` (renvoie une vue)
- `Set<Map.Entry<K, V>> entrySet()` (renvoie une vue)
- `V getOrDefault(K k, V d)`
- `V put(K k, V v)`
- `V putIfAbsent(K k, V v)`
- `V computeIfAbsent(K k, Function<K,V> f)`
- `V merge(K k, V v, BiFunction<V,V,V> f)`

9.1. Map.Entry

- `K getKey()`
- `V getValue()`
- `entrySet().stream().sorted(Map.Entry.comparingByKey())`
- `entrySet().stream().sorted(Map.Entry.comparingByValue())`

10. Comparator

- `int compare(T o1, T o2)` (renvoie un entier négatif, zéro ou un entier positif si le premier argument est inférieur, égal ou supérieur au second)

11. Comparable

- `int compareTo(T o)` (même chose)

12. System

- `static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)`
- `System.out.printf("Coucou, je suis %s et j'ai %2d ans\n", nom, age);` ou `String.format`:

```
return String.format("%d-%02d-%02d",
   year(pkDate),
   month(pkDate),
   day(pkDate)
);
```

13. Patrons de conception

- décorateur : système de cache + itérateur
- composite : système de fichiers
- adapter : bookinfo, on créé une classe qui forward les appels vers le book interne
- observer : gare CFF

14. Erreurs fréquentes

- attention aux magic numbers!
- attention à bien utiliser `put` et pas `set` pour les Map !
- ne pas oublier qu'on peut utiliser le mot-clef `class SomeClass <T extends EnumMap<T>> {}` pour limiter le type générique de la classe
- ne pas oublier les `Objects.requireNonNull`
- toujours bien utiliser `StringBuilder` pour ne jamais faire de concaténation de chaînes du type `"0" + month`, mais `sb.append("0").append(month)`
- attention à bien vérifier le type de chaque variable (pour ne pas écrire `String month = month(pkDate)`)
- bien vérifier les casts !
- bien utiliser `static` si on nous demande de coder une méthode utilitaire
- utiliser `Integer.SIZE` pour la taille d'un entier
- utiliser `((1 << bitCount) - 1` pour générer `bitCount` x 1.
- itérer proprement : `for (int m = 1; m != 0; m <<= 1) { v & m }`
- faire `i & 4095` ou `i % 4096`