**Commandes en Java**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbole | Signification | Code | Utilisation | |
| Commandes |  |  |  | |
|  | code de base pour commencer un programme Java | public class MaClassePrincipale {  public static void main(String[] args) {    }  } |  |  |
| //  /\* \*/ | commentaire monoligne  commentaire multiligne | //Ceci est un commentaire  /\*Ceci est un commentaire\*/ |  |  |
|  |  |  |  |  |
| Déclaration et portée |  |  |  |  |
|  | déclaration d'un nouvel élément | **(portée)** **type** monElement | **int** maVariable = 3;  **String** maMethode (param) {  **MaClasse** monObjet = new MaClasse(param);  **class** MaClasse { | |
|  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **(portée)** | variable | méthode( ) | Classe { |
| **public** | accès dans et hors du package | | |
|  | dans le même package | | |
| **protected** | à l'intérieur de la classe et des classes dérivées | |  |
| **static** | … de classe = commune à tous les objets (même valeur)  (héritée mais pas copiée) | |  |
| **abstract** |  | servant pour héritage uniquement  (classe non instanciable / méthode à redéfinir) | |
| **private** | uniquement à l'intérieur de la classe  (mais pas des classes dérivées !) | |  |
| **final** | constante | … ne pouvant être redéfinie | pas d'héritage possible |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Variables |  |  | | | | |  | | |  | | |
| numérique |  |  | | | | | Valeur par défaut des numériques = 0 | | |  | | |
| byte | nb entier sur 1 octet | byte maVariable; | | | | |  | | | de -128 à +127 | | |
| short | nb entier sur 2 octets | short maVariable = 31145; | | | | |  | | | de -32 768 à +32 767 | | |
| int | nb entier sur 4 octets | int maVariable; | | | | | *Renvoie la valeur min et max que peut contenir int :*  Integer.MAX\_VALUE  Integer.MIN\_VALUE | | | de -2.109 à 2.109 | | |
| long | nb entier sur 8 octets | long maVariable = 94564758220058L; | | | | |  | | | de -9.1018 à 9.1018 | | |
| float | nb réel sur 4 octets | float maVariable = 3.14159f;  float maVariable = 2.0f; | | | | |  | | |  | | |
| double | nb réel avec un grand nombre de chiffre après la virgule | double maVariable = 0.33252583336588565545566d; | | | | |  | | |  | | |
| \_ | rend les nb lisible dans le code source sans perturber le programme | double nb = 1000000000d;  double nb = 1\_\_000\_000\_000d; | | | | |  | | | \_ pas placé au début, à la fin ni autour de la virgule ! | | |
| 0x | déclarer un nombre en hexadécimal | int nb = 255;  int nb = 0xFF; | | | | |  | | |  | | |
| 0b | déclarer un nombre en binaire | int nb = 0b1111\_1111; | | | | |  | | |  | | |
| booléen |  |  | | | | | valeur par défaut des booléens = false | | |  | | |
| boolean | contient true ou false | boolean maVariable = true; | | | | |  | | |  | | |
| chaine |  |  | | | | |  | | |  | | |
| char | contient un seul caractère | char maVariable = 'A'; | | | | |  | | |  | | |
| String | contient une chaine de caractère | String maVariable = "Mon txt"; | | | | |  | | |  | | |
|  |  |  | | | | |  | | |  | | |
| Cast | Conversion de type |  | | | | |  | | |  | | |
| =( ) | converti un type numérique en un autre | double nb1 = 10, nb2 = 3;  int nb3 = (int)nb1;  int nb3 = nb1.intValue();  int result = (int) (nb1 / nb2); | | | | | int nb1 = 10, nb2 = 3;  double result = (double)nb1 / (double)nb2; | | | ATTENTION aux règles de priorité calcul et conversion pour ne pas perdre d'information ! | | |
| .valueOf( ); | converti un type numérique en chaine | maChaine = String.valueOf(nb); | | | | |  | | |  | | |
| Integer.valueOf( ).intValue(); | converti un type chaine en type numérique | int nb = Integer.valueOf(maChaine).intValue(); | | | | | .longValue();  .doubleValue(); | | | méthode à adapter selon le cas | | |
| .parse'Type'( ) | converti un type chaine en type numérique | int nb = Integer.parseInteger(maChaine);  double nb = Double.parseDouble(maChaine); | | | | |  | | |  | | |
| Boolean.valueOf( ).booleanValue() | converti un type chaine en booléen | boolean b = Boolean.valueOf(maChaine).booleanValue(); | | | | |  | | |  | | |
|  |  |  | | | | |  | | |  | | |
| Structures |  |  | | | | |  | | |  | | |
|  | opérateurs logiques | == !=  < <= > >=  && ||  ? (opérateur ternaire) | | | | | *Comparaison entre 2 objets :*  monObjetA == monObjetB  monObjetA.equals(monObjetB) | | | Remarque : peuvent comparer les valeurs des variables de type primitif (numérique et char) mais pas les objets String !  => compare seulement les adresses mémoire  => compare le contenu des 2 objets et renvoie true s'ils contiennent les mêmes valeurs, même si objets d'instances diférentes ! | | |
| if else | structure conditionnelle | if (condition) {  insruction(s);  } else if (condition) {  instruction(s);  } else {  instruction(s);  } | | | | |  | | |  | | |
| switch case | structure conditionnelle avec un nombre restrain et connu de possibilités | switch (maVariable) {  case valeur1:  instruction(s);  break;  case valeur2:  instruction(s);  break;  default:  instruction(s);  } | | | | |  | | |  | | |
| ternaire | condition ternaire  (si la condition est vrai, la variable prend la valeur1, sinon, elle prend la valeur2) | type maVariable = (condition) ? valeur1 : valeur2; | | | | | *Ex :*  *int x = 10, y = 20;*  *int max = (x < y) ? y : x;* | | |  | | |
| while | structure de boucle | while (condition) {  instruction(s);  } | | | | | while (a++ < b)  while (++a < b) | | | Remarque : priorité des opérations  l'évaluation est effectuée puis a est incrémenté  a est incrémenté puis l'évaluation est faite | | |
| do while | structure de boucle dont le premier tour est obligatoirement réalisé (la condition n'est vérifiée qu'à partir du 2e tour) | do {  instruction(s);  } while (condition); | | | | |  | | |  | | |
| for | structure de boucle dont on connait le nombre d'itération à réaliser | for (initialisation; condition; incrémentation) {  instruction(s);  } | | | | |  | | |  | | |
| break | permet de sortir prématurément d'une boucle |  | | | | | for (int i = 0; i < 9; i++) {  if (condition) {  instructions;  } else  break;  } | | | ATTENTION : à utiliser avec parcimonie dans des cas simple. Si code devient plus complexe, préférer utiliser un booléen et boucle While (plus lisible concernant les conditions de sortie de la boucle). | | |
|  |  |  | | | | |  | | |  | | |
| Chaines |  |  | | | | |  | | |  | | |
| " \n " | retour à la ligne dans une chaine | "Txt sur 1re ligne \n Txt sur 2e ligne" | | | | | => Txt sur la 1re ligne  Txt sur la 2e ligne | | |  | | |
| " \r " | retour chariot (= avec alinéa) | "Txt sur 1re ligne \r Txt sur 2e ligne" | | | | | => Txt sur la 1re ligne  Txt sur la 2e ligne | | |  | | |
| " \t " | tabulation | "Txt sur 1re ligne \t Txt sur 1re ligne" | | | | | => Txt sur la 1re ligne Txt sur la 1re ligne | | |  | | |
| .charAt( ) | renvoi le ne caractère de la chaine  (renvoi type char) | maChaine.charAt(nb); | | | | |  | | |  | | |
| .length() | renvoi la longueur de la chaine  (renvoi type int) | maChaine.lenght(); | | | | |  | | |  | | |
| .toUpperCase() | renvoi la chaine en majuscule | maChaine.toUpperCase(); | | | | |  | | |  | | |
| .toLowerCase() | renvoi la chaine en minuscule |  | | | | |  | | |  | | |
| .isEmpty() | renvoi true si la chaine est vide ("")  ou false sinon | maChaine.isEmpty(); | | | | |  | | |  | | |
| .trim().isEmpty() | renvoi true si la chaine ne contient pas de "vrai" caractères | maChaine.trim().isEmpty(); | | | | |  | | | Rq: .trim() permet de considérer comme vide des chaines contenant que des espaces, tabulation, … | | |
| .equal( ) | compare la chaine avec celle placée en param et renvoi un booléen | machaineA.equal(maChaineB); | | | | |  | | |  | | |
| .substring( , ) | extrait la partie de la chaine entre le 1er indice (inclus) et le 2nd indice (exclu) | maChaine.substring(nb, nb); | | | | |  | | |  | | |
| .indexOf( ) | recherche le caractère ou la chaine en param dans la chaine et renvoi l'indice de sa position (renvoi type int)  renvoie -1 si non trouvé | maChaine.indexOf('c');  maChaine.indexOf("chaine"); | | | | |  | | |  | | |
| .lastIndexOf( ) | idem mais recherche en partant de la fin de la chaine |  | | | | |  | | |  | | |
| .matches(" ") | teste la ou les RegEx placée(s) en paramètre sur la chaine et retourne un booléen | maChaine.matches("maRegEx"); | | | | | private void validationEmail(String email) throws Exception {  if (!email.matches("([^.@]+)(\\.[^.@]+)\*@([^.@]+\\.)+([^.@]+)"))  throw new Exception("Merci de saisir une adresse mail valide.");  } | | | méthode permettant de vérifier le format d'une adresse mail | | |
| .split(" ") | découpe une chaine selon la RegEx placée en paramètre et retourne un tableau de chaine  selon plusieurs séparateurs : | String[] mesSousChaines = maChaine.split(";");  maString.split("separateur1|separateur2"); | | | | | *ATTENTION : chaines délimiter par .*  maString.split("\\.");  *selon les retour chariot et renvoi à la ligne :*  maString.split("\n|\t|\r"); | | |  | | |
| .split(" ", ) | découpe la chaine en un nombre limité de sous-chaine => renvoie un tableau de chaine de taille égale au nombre indiqué en paramètre | String[] mesSousChaines = maChaine.split(";", 3); | | | | | *Exemple :*  maChaine = "Bonjour; Salut; Hello; Hi; Goodby";  maChaine.split(";", 3) => {Bonjour**,** Salut**,** Hello; Hi; Goodby} | | |  | | |
| .concat( ) | concatène deux chaines (mieux que concaténation avec "+") | String maChaine = "Début ".concat("milieu, ");  String maChaine2 = maChaine.concat("et fin."); | | | | |  | | |  | | |
| .format( ) | crèè une chaine en y insérant des valeurs formatées (mieux que concaténation avec "+")  ***détails : voir fin du document*** | double = 0.45d;  String chaine1 = "valeur 1";  String maChaine2 = String.format("Valeur numérique = %d, et chaine de caractère = %s", decimal, chaine1); | | | | | *Pour afficher le symbole '%' dans la chaine :*  String.format("Taux = %d %%", 10.5); | | | => Taux = 10,5 % | | |
| StringBuilder | "String dynamique", classe facilitant la manipulation de chaines de caractères  (à privilégier car plus performant que concaténation de String classiques) | StringBuilder msg = new StringBuilder("Valeur initiale"); | | | | |  | | |  | | |
|  |  |  | | | | |  | | |  | | |
|  |  |  | | | | |  | | |  | | |
|  |  |  | | | | |  | | |  | | |
| Tableaux |  |  | | | | |  | | |  | | |
| [ ] = new [n] | déclaration d'un tableau vide à n cases | type monTableau[] = new type[nb];  ou  type[] monTableau = new type[nb]; | | | | | *Ex :*  *int tableauEntiers[ ] = new int[3];*  *int[ ] tableauEntiers = new int[3];* | | |  | | |
| [ ] = { , , } | déclaration d'un tableau avec des valeurs | type monTableau[] = {valeur1, valeur2, valeur3}; | | | | | *Ex :*  *int tableauEntiers[ ] = {0, 1, 2, 3};*  *char tableauCaracteres [ ] = {'a', 'b', 'c'};*  *String tableauChaines [ ] = {"chaineA", "chaineB", "chaineC"};* | | |  | | |
| [ ][ ] = {{ }, { }} | déclatation d'un tableau à 2 dimensions | type monTableau[ ] [ ] = { {valeurA1, valeurA2, valeurA3}, {valeurB1, valeurB2, valeurB3} }; | | | | |  | | | le 1er [ ] donne accès aux valeurs A, B, C  le 2nd [ ] aux valeurs 1, 2, 3 | | |
| .length | donne la taille du tableau | monTableau.length; | | | | |  | | |  | | |
| for ( : ) {  } | boucle permettant de parcourir tous les éléments d'un tableau un à un  *(comme* .forEach( ) *en JS)* | for (type maVariable : monTableau) {  instruction(s);  } | | | | |  | | |  | | |
| .sort( ) | permet de trier le contenu d'un tableau  - selon l'ordre numérique ou alphabétique pour les objets génériques  - selon la méthode Comparable implémentée par les objets créés | Arrays.sort(monTableau); | | | | |  | | |  | | |
| Enumération |  |  | | | | |  | | |  | | |
| enum | déclaration d'une énumération | public enum MonEnumeration {  objetA,  objetB,  objetC;  }  *// récupérer un élément à partir de son* ***name***  MonEnum elt = MonEnum.valueOf("VALEUR\_B");  *// récupérer un élément à partir d'un attribut*  MonEnum element = MonEnum**.getByCode(**"B"**)**; | | | | | public enum MonEnum {  VALEUR\_A("Valeur A", "A"),  VALEUR\_B("Valeur B", "B");  private String libelle;  private String code;  private MonEnum(String libelle, String code) {  this.libelle = libelle;  this.code = code;  }  public static MonEnum **getByCode(**String code**)** {  for(MonEnum elt : MonEnum.values()) {  if(elt.getCode().equals(code) {  return elt;  }  }  return null;  }  public static boolean isValidCode(String code) {  for(MonEnum elt : MonEnum.values()) {  if(elt.getCode().equals(code) {  return true;  }  }  return false;  }  } | | | Remarque : les objets déclarés dans l'énumérations sont de type MonEnumeration (ainsi que Object) et sont public static | | |
| for ( : .values()) {  } | boucle permettant de parcourir tous les objets d'une énumération  (semblable à celle d'un tableau, mais utilisation de .values() indispensable !) | for(MonEnumeration elt : MonEnumeration.values()) {  instruction(s);  }  *// récupérer une* ***liste de sous éléments***  List<String> codes = Arrays.asList(MonEnum.values()).stream().map(MonEnum::getCode).collect(Collectors.toList()); | | | | |  | | |
|  |  |  | | | | |  | | |  | | |
| Collection | (= *liste* en JS) |  | | | | |  | | |  | | |
| List | les collections de type List sont des liste d'objets acceptant les doublons |  | | | | |  | | |  | | |
| ArrayList | à privilégier pour liste longue et en simple lecture | List<T> maListe = new ArrayList<T> (); | | | *initialiser une liste en une seule ligne :*  List<MonObjet> maListe = new ArrayList<MonObjet>(Arrays.asList(monObjetA, monObjetB, monObjetC));  List<MonObjet> maListe = Arrays.asList(monObjetA, monObjetB, monObjetC); | | | | | ATTENTION : ArrayList n'est pas synchronisé, cad à ne pas utiliser en cas de modification par plusieurs thread. | | |
| LinkedList | à privilégier pour liste courte avec insertion, suppr. éléments au milieu | List<T> maListe = new LinkedList<T> (); | | | | |  | | |  | | |
| CopyOnWriteArrayList | ArrayList thread-safe à utiliser en cas de multi-thread uniquement car couteux en temps et en mémoire | List<T> maListe = new CopyOnWriteArrayList<T>(); | | | | |  | | | Rq : permet également de lever des exceptions de type ConcurrentModificationException en cas de modification d'une liste en cours de lecture | | |
| Set | les collections de type Set sont des liste d'objets n'acceptant pas les doublons (méthode equals ou hachage) |  | | | | |  | | |  | | |
| TreeSet  ConcurrentSkipListSet | collection d'objets ordonnés  - selon l'ordre numérique ou alphabétique pour les objets génériques  - selon la méthode Comparable implémentée par les objets créés  TreeSet thread-safe | Set<T> monSet = new TreeSet<T>(); | | | | | *Instanciation en redéfinissant le comparateur :*  TreeSet<T> monSet = new TreeSet<T>(monComparator); | | | ATTENTION : l'exception ClassCastException est levée si on essaie d'inserer un objet n'implémentant pas l'interface Comparable.  Rq : exemple de classe Comparator et comparateur anonyme => voir plus bas Comparator<T> et méthode .sort( , ) | | |
| HashSet | stocke les éléments dans une table de hachage | Set<T> maListe = new HashSet<T> (); | | | | |  | | | ATTENTION : il est préférable de rédéfinir la méthode hashCode() des objets insérés dans un HashSet | | |
| LinkedHashSet | mélange entre HashSet et LinkedList, ce qui améliore les performances en lecture et écriture du HashSet | Set<T> maListe = new LinkedHashSet<T> (); | | | | |  | | |  | | |
| .add( ) | ajout de l'objet à la liste | maListe.add(monObjet);  maListe.add("Bonjour");  maListe.add(20.32f); | | | | |  | | | Remarque : pas besoin de déclarer l'objet avant, il sera créé si besoin | | |
| .addAll( ) | ajoute tout le contenu d'une autre collection en copiant les éléments (et non pas leur référence) et retourne true si l'action a abouti | maListe.addAll(maListe2); | | | | |  | | |  | | |
| .get( ) | obtenir l'élément d'indice nb | maListe.get(nb); | | | | |  | | |  | | |
| .set( , ) | remplace l'élément d'index indiqué par l'élément donné en second paramètre | maListe.set(2, monObjet); | | | | | IndexOutOfBoundsException | | |  | | |
| .remove( ) | supprime l'élément placé en paramètre s'il est trouvé dans la liste | maListe.remove(monObjet);  maListe.remove("Bonjour"); | | | | |  | | |  | | |
| .removeAll( ) | supprime les éléments communs avec ceux de la Collection placée en paramètre (fait appel à equals) | maListe.removeAll(maListe2); | | | | |  | | | Remarque : fonctionne même si liste pas de même type ! | | |
| .clear() | supprime tous les éléments | maListe.clear(); | | | | |  | | |  | | |
| .isEmpty() | renvoie true si la liste est vide | maListe.isEmpty(); | | | | |  | | |  | | |
| .contains( ) | retourne true si l'objet placé en paramètre est trouvé dans la liste (equals) | maListe.contains(monObjet);  maListe.contains("Bonjour"); | | | | |  | | |  | | |
| .containsAll( ) | renvoie true si la collection contient TOUS les éléments de la collection en paramètre | maListe.containsAll(maListe2); | | | | |  | | |  | | |
| .retainAll( ) | ne conserve dans la collection que les éléments présents dans la collection en paramètre | maListe.retainAll(maListe2); | | | | |  | | |  | | |
| .toArray() | renvoi un tableau contenant tous les objets de la liste | Object[] monTableau = maListe.toArray(); | | | | |  | | | ATTENTION au type du tableau si la liste contient des objets de différents types ! | | |
| for( ; ; ) | boucle for permet de parcourir tous les objets d'une collection | for (int i = 0, size = maListe.size(); i < size; i++) {  System.out.println(maListe.get(i));  } | | | | | for(Type elt : maList) {  System.out.println(elt);  } | | |  | | |
| Iterator | objet permettant de parcourir une List ou un Set | Iterator it = maList.iterator();  while(it.hasNext()) {  Type valeur = (Type)it.next();  }  ----------------------------  Iterator<Type> it = maList.iterator();  while(it.hasNext()) {  Type valeur = it.next(); //cast non nécéssaire  } | | | | | *Méthode incluses dans Iterator :*  hasNext() => retourne un boléen  next() => retourne l'objet courant  remove() => suprime l'objet courant de la collection | | |  | | |
| ListIterator() | objet permettant de parcourir une List | ListIterator<Type> li = maListe.listIterator();  while(li.hasNext()) {  Type valeur = li.next();  } | | | | | *Méthode incluses dans Listterator :*  hasNext() hasPrevious()  next() previous()  remove() set( ) => remplace l'elt courant  nextIndex() previousIndex() | | | Rq : remove() ne fonctionne pas avec un CopyOnWriteArrayList car cela lève une UnsupportedOperationException | | |
| Collections | Objet contenant des méthodes permettant la gestion des collections de type Set et List |  | | | | |  | | |  | | |
| .reverse( ) | inverse l'ordre des éléments | Collections.reverse(maListe); | | | | |  | | |  | | |
| .rotate( , ) | place le nb d'élément indiqué en second paramètre en début de liste | Collections.rotate(maListe, 2); | | | | | *Exemple :*  *//avant rotation //après rotation*  *{a, b, c, d, e, f} {e, f, a, b, c, d}* | | |  | | |
| .shuffle( ) | permute les éléments de façon aléatoire | Collections.shuffle(maListe); | | | | |  | | |  | | |
| .swap( , , ) | permute les deux éléments de la liste | Collections.swap(maListe, 2, 5); | | | | |  | | |  | | |
| .sort( ) | trie une List  - selon l'ordre numérique ou alphabétique pour les objets génériques  - selon la méthode Comparable implémentée par les objets créés | Collections.sort(maList); | | | | public class MonObjet implements Comparable {  private int nb;  private String name;  public int compareTo(Object o) {  if(o.getClass().equals(MonObjet.class) {  MonObjet monObj = (MonObjet)o;  if(this.name.compareTo(monObj.getName() == 0) {  return this.nb.compareTo(monObj.getNb());  return this.name.compareTo(monObj.getName());  }  return -1;  }  } | | | | Rq :  compareTo retourne une valeur   * négative si l'élément est plus petit * 0 s'ils sont égaux * positive s'il est plus grand.   Rq : la méthode .sort( ) ne marche pas pour les collections de type Set car celle-ci sont triées automatiquements. Pour modifier l'ordre de trie, il faut définir un comparateur au moment de l'instanciation de la collection. | | |
| .sort( , )  Comparator<T> | trie une List avec un comparateur spécifique prévalant sur compareTo( )  défini un comparateur pouvant servir de référence de tri lors d'une instanciation d'un Set | public class MonComparateur implements Comparator<MonObjet> {  public int compare(MonObjet obj1, MonObjet obj2) {  int nb1 = (Integer)obj1.getNb();  int nb2 = (Integer)obj2.getNb();  int result = nb1.compareTo(nb2);  if(result == 0)  return obj1.compareTo(obj2);  return result;  }  }  Collections.sort(maList, new monComparateur()); | | | | | | *Avec classe anonyme :*  Collections.sort(maList, new Comparator<MonObjet>() {  public int compare(MonObjet obj1, MonObjet obj2) {  int nb1 = (Integer)obj1.getNb();  int nb2 = (Integer)obj2.getNb();  int result = nb1.compareTo(nb2);  if(result == 0)  return obj1.compareTo(obj2);  return result;  }  }); | | |  | |
|  |  |  | | | | |  | | |  | | |
| Map | les collections de type Map sont des listes d'objets identifiés par un syteme clé-valeur (les clés ne pouvant être identique) |  | | | | |  | | |  | | |
| TreeMap  ConcurrentSkipListMap | stocke les éléments dont les clés sont triès  - selon l'ordre numérique ou alphabétique pour les objets génériques  - selon la méthode Comparable implémentée par les objets créés  TreeMap thread-safe | Map<K, V> maMap = new TreeMap<K, V>(); | | | | | *Instanciation en redéfinissant le comparateur :*  Map<K, V> maMap2 = new TreeMap<K, V>(new MapComparator(maMap1));  maMap2.putAll(map1); | | |  | | |
| HashMap | stocke les éléments en utilisant une table de hachage (non thread-safe) |  | | | | | *Déclaration d'une Map en une ligne :*   * *en java 8*   Map<String, String> myMap = new HashMap<String, String>() {{  put("a", "b");  put("c", "d");  }};   * *en Java 9*   Map<String, String> test2 = Map.ofEntries(  entry("a", "b"),  entry("c", "d")  ); | | | Rq: ici comme pour LinkedHashMap, la valeur null est autorisée comme clé et comme valeur | | |
| LinkedHashMap | combine table de hachage et liens chainés facilitant l'insertion et la supression |  | | | | | Rq: en utilisant un iterator sur le Set (issu de la map par .entrySet()) on peut parcourir les éléments dans lordre dans lequel ils ont été ajoutés | | |
| WeakHashMap | les éléments sont automatiquement suprimés de la Map lorsque la clé utilisée devient inacessible, cad éliminée de la mémoire via le Garbage Collector |  | | | | |  | | |  | | |
| .get( ) | retourne la valeur associée à la clé passée en paramètre ou null si elle n'existe pas | TypeV maValeur = maMap.get(maCle); | | | | |  | | |  | | |
| .clear() | supprime tous les éléments | maMap.clear(); | | | | |  | | |  | | |
| .containsKey( ) | retourne true si l'objet placé en paramètre est trouvé dans la liste de clé (equals) | maMap.containsKey(maCle); | | | | |  | | |  | | |
| .containsValue( ) | retourne true si l'objet placé en paramètre est trouvé dans la liste de valeur (equals) | maMap.containsValue(maValeur); | | | | |  | | |  | | |
| .isEmpty() | retourne true si la map ne contient aucun couple clé-valeur | maMap.isEmplty(); | | | | |  | | |  | | |
| .put( , ) | ajoute le couple clé-valeur à la map ou met à jour la valeur si la clé existe déjà | maMap.put(maCle, maValeur); | | | | |  | | | Rq: cette méthode retourne la valeur du couple ajouté | | |
| .putAll( ) | ajoute tout le contenu d'une autre Map en copiant les éléments (et non pas leur référence) et retourne true si l'action a abouti | maMap1.putAll(maMap2); | | | | |  | | |  | | |
| .remove( ) | supprime le couple clé-valeur associé à la clé passée en paramètre, et retourne la valeur suprimée | maMap.remove(maCle); | | | | |  | | |  | | |
| Map.Entry<K, V>  .entrySet()  .keySet()  .values() | boucle for sur une collection de type Map permettant de récupérer les clés et les valeurs | for (Map.Entry<Integer, String> entry : maMap.entrySet()) {  Integer cle = entry.getKey();  String valeur = entry.getValue();  } | | | | | *Boucle for uniquement sur les clés ou les valeurs :*  for(TypeK cle : maMap.keySet()) {  System.out.println(cle);  }  for(TypeV value : maMap.values() {  System.out.println(value);  } | | | Rq :  Set<K> mesCles = maMap.keySet();  Collection<V> mesValeurs = maMap.values(); | | |
| Comparator<T> | défini un comparateur pouvant servir de référence de tri lors d'une instanciation d'une Map | public class MapComparator implements Comparator<MonObjet> {  Map<Integer, MonObjet> map;  public MapComparator(Map<Integer, MonObjet>) {  this.map = map;  }  public int compare(Integer key1, Integer key2) {  //comparaison selon la clé  return key1.compareTo(key2);  //comparaison selon la valeur  MonObjet obj1 = map.get(key1);  MonObjet obj2 = map.get(key2);  return obj1.compareTo(obj2);  }  } | | | | |  | | |  | | |
|  |  |  | | | | |  | | |  | | |
|  |  |  | | | | |  | | |  | | |
|  |  |  | | | | |  | | |  | | |
| Méthode |  |  | | | | |  | | |  | | |
| public static | déclaration d'une nouvelle méthode  avec valeur retournée  sans valeur retournée | public static type maMethode(type param1, type param2) {  instruction(s);  return (type) valeur;  }  public static void maMethode(type param1, type param2) {  instruction(s);  } | | | | | public class MonProjet {  public static void main(String[] args) {  instruction(s);  maMethode(valeur1, valeur2);  instruction(s);  }  public static void maMethode (…) {  instruction(s);  }  } | | | le code source de la méthode doit être placé à l'intérieur de la classe "monProjet", mais à l'extérieur de la méthode principale "main" (chargée automatiquement) !  ATTENTION : une méthode ne marche qu'avec le même type de valeur que celui défini  Pour utiliser la même méthode avec un autre type de variable, il faut la Surcharger (= copier le code de la méthode en gardant le nom mais en modifiant les types des paramètres) | | |
| .toString() | méthodes de Object => commune à tous les objets de Java  renvoie une chaine de caractère décrivant l'instance en question | maClasse.toString(); | | | | | public class MaClasse {  private int maValeur = 0;  private String maString = "Bonjour";  …  public String toString() {  return "Mes agruments sont : maValeur = " + maValeur + " et maString = " + maString;  } | | |  | | |
| .equals( ) | méthodes de Object => commune à tous les objets de Java  compare le contenu de deux objets et renvoie true si les valeurs sont identiques (même si non issu de la même instance) | if(monObjet1TypeA.equals(monObjet2TypeA)) {  …  } | | | | | public class MaClasse {  private int valeur = 0;  private String str = "";  …  public boolean **equals(Object o) {**  if(o != null && (o.getClass().equals(this.getClass()))) {  MaClasse mC = (MaClasse) o;  return ((this.valeur == mC.getValeur()) && (this.str.equals(mC.getStr()));  } else  return false;  **}**  } | | | | |  |
|  |  |  | | | | |  | | |  | | |
| Classe |  |  | | | | |  | | |  | | |
| public class | créé une nouvelle classe (= prototype d'objet) | public class MaClasse {  instruction(s);  } | | | | |  | | |  | | |
| constructeur | une classe doit contenir un constructeur qui servira lors de l'instanciation | public class MaClasse {  type prop1;  type prop2;  public MaClasse(param1, param2) {  prop1 = param1;  prop2 = param2;  }  } | | | | |  | | | Remarque : le constructeur   * n'a aucun type (ni double, ni String, ni void ! * les noms des propriétés (variables d'instances) n'ont pas à recevoir de type (puisqu'elles ont déjà été créées dans le corps de la classe elle-même) | | |
| new | l'instanciation d'un nouvel objet à partir de la classe créé se fait grâce à la méthode constructeur | MaClasse monObjet = new MaClasse(valeur1, valeur2); | | | | |  | | |  | | |
| variables d'instance | = propriété des objets qui reçoivent une valeur particulière lors de l'instanciation |  | | | | | public class MaClasse {  private type prop1;  protected type prop2;  public type getProp1() {  return prop1;  }  public void setProp1(type param1) {  prop1 = param1;  }  } | | | il peut être préférable d'empêcher la modification des valeurs de la classe et des objets qui en sont issus depuis main du programme, pour cela on utilise private ou protected  pour avoir accès à la valeur des prop. on utilise alors une méthode accesseur/getter dans le corps de la classe  et pour les modifier : méthode mutateur/setter  que l'on appelle avec :  monObjet.getProp1();  monObjet.setProp1(valeur); | | |
| static | variables de classe  = propriété présentant la même valeur pour tous les objets issus de l'instanciation à partir de cette classe | public static type maVariable = valeur;  private static type maVariable = valeur; | | | | | public static type getMaVariable() {  return maVariable;  } | | | Remarque : les méthodes utilisant uniquement des variables de classes doivent être déclarées avec static  ce sont des méthodes de classes | | |
| extends | Classe dérivée  classe héritant des propriétés d'une classe de base et en ajoutant de nouvelles | public class MaClasseDerivee extends MaClasse { | | | | | public class MaClasseDerivee extends MaClasse {  this.prop1 = valeur; Erreur !!!  this.prop2 = valeur; Bon  } | | | seule les variables déclarées public ou protected sont directement accessible dans le corps de la classe dérivée | | |
| super | fait référence à une des méthodes de la classe de base dans le corps de la classe dérivée  ou à son constructeur | super.maMethodeDeBase();  super(); | | | | | MaClasseDerivee(type val1, type val2, type val3) {  super(valeur1, valeur2);  this.prop3 = val3;  }  *Ex :*  String str = super.decrisToi() + " nouvelle info : " + prop5; | | | le constructeur de la classe dérivée doit obligatoirement faire appel au constructeur de la classe de base dès la 1er instruction !  fait référence et enrichie une méthode de la classe de base avec nouvelles infos | | |
| abstract | classe abstraite servant de classe de base et permettant l'héritage mais pas l'instanciation  méthode abstraite à redéfinir | public abstract class MaClasseAbstraite {  protected String maProp1;    protected MaClasseAbstraite(String maProp1) {  this.maProp1 = maProp1;  }    protected Sring meDecrire() {  String dscpt = "Je suis un objet de la " + this.getClass() + " et maProp1 à la valeur : " + this.maProp1;  }  public abstract String maMethodeAbstraite();  }  public class MaClasseDerivee extends maClasseAbstraite {  public maClasseDerivee() {  }  public maClasseDerivee(String valeur1) {  super(valeur1);  }  public String maMethodeAbstraite() {  String str = "Méthode redéfinie";  return str;  }  } | | | | | | | | Remarque : une classe abstraite peut ne pas contenir de méthode abstraite  Mais une méthode abstraite doit obligatoirement être dans une classe abstraite  Remarque : une classe abstraite ne peut être instanciée (seulement héritée). Mais elle peut contenir un constructeur qui servira dans le constructeur de la classe dérivée en appelant la méthode super().  ATTENTION : il est déconseillé de créer une nouvelle méthode dans la classe dérivée sans la créer abstraite dans la classe de base  Sinon : Tableau d'objet de type MaClasseAbstraite => bug lors de l'appel de cette méthode  = perte du Polymorphisme | | |
|  |  |  | | | | |  | | |  | | |
| Classe générique |  |  | | | | |  | | |  | | |
| <T> | définit une classe générique, cad une classe neutre sans type particulier dont le type de donné géré sera attribué à l'objet instancié à partir d'elle | public class maGenerique <T> {  private T prop1;  //Constructeurs  public maGenerique() {  this.prop1 = null;  }  public maGenerique(T valeur1) {  this.prop1 = valeur1;  }  //Accesseurs  public void setProp1(T valeurN) {  this.prop1 = valeurN;  }  public T getProp1() {  return this.prop1;  }  } | | | | | public class maGenerique <T, U, V> {  private T prop1;  private U prop2;  private V prop3;  …  maGenerique<type> monObjet = new maGenerique<type>(valeur);  *Ex :*  *maGenerique<String> monObjet = new maGenerique<String>("Bonjour");* | | | classe générique prenant plusieurs types génériques si plusieurs paramètres de différents types  l'instanciation d'un objet doit se faire avec la déclaration du type de donnée choisi !  ATTENTION : le type doit être déclaré sous forme de classe et non pas sous forme primitif  Il suffit de mettre une majuscule au nom primitif sauf pour :  int => Integer  char => Character | | |
| static <T> | méthode générique statique | public class TestsUtils {  public static <T> boolean contenuListesIdentiques(List<T> listA, List<T> listB) {  return listA.containsAll(listB) && listB.containsAll(listA);  }  } | | | | | | | |  | | |
| Interface |  |  | | | | |  | | |  | | |
| interface | déclare une nouvelle interface (= classe 100% abstraite) | public interface MonInterface {  public void maMethodeA();  public String maMethodeB();  } | | | | | *Interface héritant d'une/plusieurs autre(s) interface(s)*  public interface MonInterface extends MonAutreInterfaceA, MonAutreInterfaceB {  public void maMethodeA();  public String maMethodeB();  } | | | Remarque : l'interface étant totalement abstraite par définition, inutile de préciser abstract devant les méthodes  ATTENTION : **contrairement aux classes**, une interface peut hériter de plusieurs autres interfaces !  En revanche, la notion d'implémentation n'existe pas pour une interface. | | |
| implements | implémente une interface dans une classe | public class MaClasse implements MonInterface {  public void maMethodeA() {  instruction(s);  }  …  } | | | | | public class MaClasse extends MaClasseDeBase implements MonInterface {  public class MaClasse implements InterfaceA, InterfaceB { | | | héritage en premier, implémentation en second  possibilité de plusieurs implémentations  ATTENTION : toutes les méthodes de l'interface doivent être redéfinies dans le corps de la classe implémentée !!! | | |
| default | déclare une méthode ayant un comportement par défaut => pas systématiquement nécéssaire de la redéfinir dans la classe implémentant l'interface | public interface MonInterface {  public void maMethodeA();  default String maMethodeB() {  return "Réponse par défaut";  }  } | | | | | *Interface héritant d'une/plusieurs autre(s) interface(s)*  public interface MonInterface1 extends MonI2 {  public void maMethodeA();  default String maMethodeB() {  return MonI2.msgDefaultIssuDeMonI2() + "complete dans MonInterface1";  }  } | | | Rq: seulement depuis Java 8  Rq: limite les problèmes de compatibilité en cas d'oubli de redéfinition de la méthode dans chcune des classes implémentant cette interface. | | |
|  |  |  | | | | |  | | |  | | |
| Packages |  |  | | | | |  | | |  | | |
| import | permet l'importation d'un package complet ou d'une classe d'un autre package | import monpackage;  import monpackage.maClasse; | | | | | import monpackage;  public class maClasse {  public static void main {  …  }  } | | | doit être placé au tout début du code source, avant le public class maClasse {  Raccourcit clavier : Ctrl+Maj+O | | |
| import java. | permet l'importation de package ou de classes particulière dans le package de java | import java.util.\*;  import.java.util.Scanner; | | | | |  | | | doit être placé au tout début du code source, avant le public class maClasse { | | |
|  |  |  | | | | |  | | |  | | |
| Lambda | permet de redéfinir une interface fonctionelle anonyme en une seule ligne | uniquement depuis Java 8 | | | | |  | | |  | | |
| () -> ;  () -> { ; ; ;}; | sans paramètre | *avec interface fonctionelle perso*  public interface RetournerConstante {  public double getConstante();  }  RetournerConstante rcPi = () -> 3.14159265359d;  System.out.println(rcPi.getConstante();  RetournerConstante rcPi = () -> {  System.out.print("Pi = ");  return 3.14159265359d;  };  rcPi.getConstante(); | | | | | *ou avec une interface fonctionnelle de Java*  import java.util.function.\*;  Supplier<Double> suplPi = () -> 3.14159265359d;  System.out.println(suplPi.get()); | | |  | | |
| ( , ) -> ;  ( , ) -> { ; ;}; | avec paramètres | (param1, param2) -> *valeur retournée*;  (param1, param2) -> {  *instruction;*  *instruction;*  return valeur;  }; | | | | | import java.util.function.\*;  Function<Personne, String> fNom = (Personne p) -> p.getNom();  fNom.apply(personageA); | | |  | | |
| :: | utilise la référence d'une méthode pour redéfinir celle d'une interface fonctionelle |  | | Function<Personne, String> fNom = (Personne p) -> p.getNom();  *équivalent à*  Function<Personne, String> fNom = Personne::getNom(); | | | | | |  | | |
| Fonctions |  |  | | | | |  | | |  | | |
| *Function<T, R>* | Fonction  (T) -> R | Function<T, R> maFunction = (elt) -> {  traitement(s);  return monR;  };  R monR = maFunction.apply(monT); | | | | | Function<User, String> concatPseudoAndAge = (user) -> {  String result = user.getPseudo();  result += user.getAge();  return result;  };  *ou bien plus simplement :*  Function<User, String> accolerNameAndAge = user -> user.getPseudo() + user.getAge(); | | |  | | |
| *Predicate<T>* | Prédicat : effectu un test et retourne true ou false  (T) -> boolean | Predicate<T> monPredicate = (elt) -> *test*;  boolean result = monPredicate.test(monT); | | | | | Predicate<User> isTeenage = (user) -> 12 < user.getAge() && user.getAge() < 18; | | |  | | |
| *Consumer<T>* | Consomateur : applique un traitement en modifiant l'objet reçu en paramètre  (T) -> void | Consumer<T> monConsumer = (elt) -> *traitement*;  monConsumer.accept(monT); | | | | | *Exemples :*  Consumer<User> add10years = (user) -> user.setAge(user.getAge() + 10);  Consumer<List<User>> add10yearsToAll = (users) -> users.stream().forEach(add10years);  *Utilisation du 1er consumer dans un stream :*  users.stream().forEach(add10years);  *Utilisation directe du 2nd consumer :*  add10yearsToAll.accept(users); | | |  | | |
| *Supplier<R>* | Fournisseur : retourne un élément de type R sans prendre de paramètre  () -> R | R monR = monSupplier.get(); | | | | |  | | |  | | |
| *BinaryOperator<T>* | Opérateur : effectu une opération entre deux éléments et renvoi le résultat  (T, T) -> T | BinaryOperator<T> maBiOp = (elt1, elt2) -> *regroupement*;  T monT3 = maBinaryOperator.apply(monT1, monT2); | | | | | BinaryOperator<Integer> sumAges = (age1, age2) -> age1 + age2;  *utilisation dans un stream :*  int result = users.stream().map(user -> user.getAge()).reduce(sumAges).orElse(0); | | | Rq: pour utiliser directement .reduce( ) sans faire de mapping au préalable => voir ci-dessous *BiFunction* | | |
| *BiFunction<T, U, R>* | Fonction prenant 2 éléments en entrée (potentiellement de type diférents) et retournant un élément en sortie de type différent de celui/ceux en entrée  (T, U) -> R | BiFunction<T, U, R> maBiFunction = (elt1, elt2) -> *traitement(s)*;  R monR = maBiFunction.apply(monT, monU); | | | | | BiFunction<Integer, User, Integer> sumUsersAges = (sum, user) -> sum + user.getAge();  *utilisation dans un stream avec .reduce :*  int result = users.stream().reduce(0, sumUsersAges, sumAges); | | | Rq: pour plus de détails => voir Stream.reduce( , , ) | | |
| .andThen( ) | permet de chainer plusieurs fonctions  Rq: le résultat de la précédente sert de paramètre à la suivante, il faut donc que le nombre et le type de paramètre/retour soient compatibles ! | maFonctionA.andThen(maFonctionB) | | | | | BiFunction<Integer, Integer, Integer> addition = (a, b) -> a + b;  Function<Integer, Integer> carre = (b) -> b\*b;  addition.andThen(carre) | | |  | | |
|  |  |  | | | | |  | | |  | | |
| Stream | patern d'itération permettant le traitement/la manipulation des données | uniquement depuis Java 8 | | | | | java.util.stream | | | Rq: ne stocke pas les données, ne modifie pas les données sources. Une fois le stream consomé (traitement fini) il est détruit. Pour un nouveau traitement, il faut le regénéré. | | |
| .iterate( , ).limit( ) | création d'un traitement d'itération à partir de Stream | *Stream infini :*  Stream.iterate(valeurInitiale, UnaryOperator);  *Stream limité à 100 itérations :*  Stream.iterate(valeurInitiale, UnaryOperator).limit(100); | | | | | | | Stream.iterate(1, n -> n + 1).limit(100).forEach(System.out::println); | ATTENTION : ne JAMAIS faire de stream infini ! => PB de mémoire et de performance !  Toujours imposer une limite ! | | |
| Arrays.stream( ) | création d'un stream à partir d'un array | String[] monArray = {"valeur1", "valeur2", "valeur3"};  Stream<String> monStream = Arrays.stream(monArray); | | | | | | |  |  | | |
| .stream() | création d'un stream à partir d'une collection ou d'une map | Stream<MonBean> stream = maList.stream(); | | | | | | |  |  | | |
| .parallel()  .parallelStream() | permet d'obtenir un stream qui sera affecté à un autre thread | Stream.iterate(…).parallel()…  maList.parallelStream() | | | | | | |  |  | | |
| .concat( , ) | réunir 2 streams du même type | Stream<MonObjet> st1;  Stream<MonObjet> st2;  Stream<MonObjet> stGlobal = Stream.concat(st1, st2); | | | | | | |  |  | | |
| Pattern | création d'un stream à partir du résultat de l'analyse d'une string par une une RegEx | String monTxt = "Abcd";  Pattern monPattern = Pattern.compile("maRegEx");  Stream<String> stm = monPattern.splitAsStream(monTxt); | | | | | | |  |  | | |
|  | méthodes intermédiaires gardant le stream ouvert |  | | | | |  | | |  | | |
| .limit( ) | élimine les éléments au dela du rang pacé en paramètre | monStream.limit(100); => nb d'elt max = 100 | | | | |  | | |  | | |
| .skip( ) | élimine les éléments précédents et celui égal au rang placé en paramètre | monStream.skip(20); => renvoi le 21e et les suivants | | | | |  | | |  | | |
| .sorted()  .sorted( ) | tri les éléments selon la methode compareTo() des objets qu'il contient, ou selon le comparateur en paramètre | monStream.sorted();  monStream.sorted(Comparator); | | | | | pizzas.stream().sorted((p1, p2) -> p1.getPrice().compareTo(p2.getPrice())); | | |  | | |
| .filter( ) | parcours les éléments en ne renvoyant que ceux qui vérifient la condition | monStream.filter(Predicate); | | | | | stream.filter(x -> x.getAge() > 18).forEach(System.out::println); | | |  | | |
| .map( ) | ne récupère de l'objet source uniquement la/les info(s) qui nous intéressent | monStream.map(Function); | | | | | stream.map(x -> x.getAge()).forEach(System.out::println); | | |  | | |
| .flatMap( ) | fusionner les éléments de sous-listes dans une seule liste | List<ArrayList<String>> listeDeListe = new ArrayList<ArrayList<String>>();  ArrayList<String> sousListe1 = new ArrayList<String>(Arrays.asList("A", "B", "C"));  ArrayList<String> sousListe2 = new ArrayList<String>(Arrays.asList("A", "E", "F"));  listeDeListe.add(sousListe1);  listeDeListe.add(sousListe2);  List<String> listeMerge = listeDeListe.stream().flatMap(**sListe -> sListe.stream()**).distinct().collect(Collectors.toList());   |  |  | | --- | --- | | System.out.println(listeMerge.toString()); | sans .distinct() => ["A", "B", "C", "A", "E", "F"]  avec .distinct() => ["A", "B", "C", "E", "F"] | | | | | | | | |  | | |
| .peek( ) | permet de débuguer entre chaque oppération |  | | | | |  | | |  | | |
| .distinct() | option à ajouter sur une liste permetant d'éliminer les doublons | maListe.stream().map(order -> order.getCustomer()).distinct().collect(Collectors.toList()); | | | | | | | | ATTTENTION : se base sur la méthode equals( ) des objets  Rq: si on souhaite éliminer les doublons d'une liste en personnalisant la regle de discrimination sans modifier la méthode equals de l'objet, plusieurs possibilités : voir Java-ExempleCodes > *Exemples pratiques* | | |
|  | méthodes terminales consommant le stream |  | | | | |  | | |  | | |
| .count() | retourne le nombre d'éléments | monStream.count(); | | | | |  | | |  | | |
| .allMatch( ) | retourne true si tous les éléments vérifient le test. Sinon retourne false | boolean result = monStream.allMatch(Predicate); | | | | |  | | |  | | |
| .anyMatch( ) | retourne true si au moins un élément vérifie le test. Sinon retourne false | boolean result = monStream.anyMatch(Predicate); | | | | |  | | |  | | |
| .noneMatch( ) | retourne true si aucun élément ne vérifie le test. Sinon retourne false | boolean result = monStream.noneMatch(Predicate); | | | | |  | | |  | | |
| .min( )  .max( ) | retourne l'élément le plus grand/le plus petit selon le comparateur indiqué encapsulé dans un Optionnal | Optionnel<Type> opt = monStream.max(Comparator); | | | | | pizzas.stream().max((p1, p2) -> p1.getPrice().compareTo(p2.getPrice())); | | |  | | |
| .findFirst() | retourne le premier élément du stream encapsulé dans un objet Optionnal | Optionnal<Type> opt = monStream.findFirst(); | | | | |  | | |  | | |
| .reduce( ) | effectu un calcul à partir de tous les éléments de la liste pour ne renvoyer qu'une seule valeur | monStream.reduce(valeurInit, BinaryOperator); | | | | | double sommePoids = stream.map(x -> x.getPoids()).reduce(0.0d, (x, y) -> x + y); | | |  | | |
| .reduce( , , ) | effectu un calcul à partir des valeurs d'un champ des éléments de la liste | monStream.reduce(valeurInit, BiFunction, BinaryOperator); | | | | | int sumAges = users.stream().reduce(0, (sum, user) -> sum + user.getAge(), (age1, age2) -> age1 + age2); | | |  | | |
| .forEach( ) | parcours les éléments | monStream.forEach(Consumer); | | | | | stream.forEach(System.out::println); | | |  | | |
| .collect( ) | rassemble les éléments dans un seul objet (liste, set, map, …) | List<Type> maListe = monStream.collect(Collectors.toList()); | | | | |  | | |  | | |
| Optionnal<T> | objet pouvant être retourné par les méthodes terminales et permettant de stocker la valeur finale et de gérer les retours null si les filtres succéssifs n'ont retournés aucune valeur |  | Optional<Double> sommePoids = stream  .filter(x -> x.getPoids() > 150)  .map(x -> x.getPoids())  .reduce((x, y) -> x + y); *//Rq: Optional non nécéssaire avec* .reduce(0.0d, …);  if(sommePoids.isPresent())  System.out.println(sommePoids);  *ou plus simplement :*  System.out.println(sommePoids.orElse(0.0d)); | | | | | | |  | | |
| Collectors | objet permettant aux stream de retourner une collection (liste, map, array) | monStream.collect(**Collectors.**xxxxxx()); | | | | |  | | |  | | |
| .toList() | pour récupèrer une liste | Collectors.toList() | | | | |  | | |  | | |
| .toSet() | pour récupérer un set |  | | | | |  | | |  | | |
| .toMap( , )  .toMap( , , ) | pour récupérer une map simple de type clé-valeur  param1: clé  param2: valeur  param3 facultatif : stratégie de remplacement ou de conservation en cas de conflit (plusieurs objets ayant la même clé)  Rq: pour regrouper les objets en une map clé-liste utiliser .groupingBy( ) | Collectors.toMap(Function, Function);  Collectors.toMap(Function, Function, Comparator); | | | | | pizzas.stream().toMap(Pizza::getPrice, Pizza::getName, (existing, replacement) -> existing)); | | |  | | |
| .partitioningBy( ) | pour récupérer une map regroupant les éléments en deux groupes : true et false | Map<Boolean, List<Object> maMap = monStream.collect(Collectors.partitioningBy(Predicate)); | | | | |  | | |  | | |
| .groupingBy( ) | pour récupérer une map clé-liste en regroupant les objets automatiquement  param : **clé** | Map<Object, List<Object>> maMap = monStream.collect(Collectors.groupingBy(Function)); | | | | | Map<Client, List<Commande>> result = commandes.stream().collect(Collectors.groupingBy(Commande::getClient)); | | |  | | |
|  | en faisant un calcul sur l'une des valeurs de l'objet (somme, moyenne, concaténation, ...) | Map<Client, Integer> result = commandes.stream().collect(Collectors.groupingBy(Commande::getClient, Collectors.summingInt(Commande::getTotalCommande))); | | | | | | | |  | | |
|  | en faisant une liste sur l'un des champs de l'objet | Map<Client, List<Integer>> result = commandes.stream().collect(Collectors.groupingBy(Commande::getClient, Collectors**.mapping(**Commande::getTotalCommande, Collectors.toList()**)**)); | | | | | | | |  | | |
| .summingInt( )  .summingDouble( )  .summingLong( ) | calcule la somme d'un champ de type Integer, Double ou Long des éléments du stream | Collectors.summingInt(Function); | | | | | Map<Department, Integer> totalSalariesByDept = employees.stream().collect(Collectors**.groupingBy(**Employee::getDepartment, Collectors.summingInt(Employee::getSalary)**)**); | | | Rq: permet de pouvoir effectuer ce calcul dans un .collect( ) associé à un autre Collectors comme le groupingBy() par exemple. | | |
| .joining()  .joining( )  .joining( , , ) | regroupe les éléments d'un stream de type String en une seule chaine de caractère  séparés par un séparateur  avec préfixe et sufixe | Collectors.joining();  Collectors.joining("separteur");  Collectors.joining("separateur", "prefixe", "sufixe"); | | | | |  | | | Rq: ne s'applique que sur des streams de type String ! | | |
|  |  |  | | | | |  | | |  | | |
|  |  |  | | | | |  | | |  | | |
|  |  |  | | | | |  | | |  | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Exceptions |  |  |  |  |
| try catch | structure permettent de capturer les exceptions | try {  instruction(s) pouvant générer une exception;  } catch (TypeException e) {  instruction(s) à réaliser en cas d'exception;  } finally {  instruction(s) à effectuer systématiquement;  } | *(si plusieurs exceptions à capturer en même temps)*  try {  instruction(s);  } catch (MonException1 e) {  instruction(s);  } catch (MonException2 e2) {  instruction(s);  } finally {  instruction(s);  } | try teste s'il y a exception  catch sera réalisé uniquement s'il y a eu exception  finaly (facultatif) sera réalisé dans tous les cas  finaly sert not. pour les instructions qui pourraient être bloquées par l'exception mais que l'on souhaite forcer malgré tout (fermeture d'un fichier, déconnexion d'une base de donnée ou d'un réseau, …) |
|  |  |  |  |  |
| e.getMessage() | donne le message d'erreur de l'exception | } catch (TypeException e) {  System.out.println(e.getMessage());  } |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  | ArithmeticException  ClassCastExeption |  |  |
|  |  |  |  |  |
| class extends Exception { | création d'une nouvelle exception personnalisée | class MonTypeException extends Exception {  public MonTypeException(param) {  *instruction(s) à réaliser en cas d'exception de ce type*;  }  } | *Ex : exception pour un nb d'habitant négatif*  *class NbHabNegatifException extends Exception {*  *public NbHabNegatifException(int nb) {*  *System.out.println("Le nb d'habitant proposé : " + nb + " est négatif !");*  *}* | Remarque : paramètre facultatif permet éventuellement de pouvoir donner une info sur la valeur qui pose problème |
| throws {  throw | détection et capture d'une exception personnalisée  dans une méthode ou dans le constructeur d'une classe | pblic MaClasse(param) throws MonTypeException {  if (condition) {  throw new MonTypeException(param);  } else {  *instruction(s) si pas d'exception = comportement normal*;  }  } | *Ex : constructeur d'une "ville"*  *public Ville(String nom, int nb) throws NbHabNegatifException {*  *if (nb < 0)*  *new NbHabNegatifException(nb);*  *else {*  *nomVille = nom;*  *nbHab = nb;*  *}*  *}* | ATTENTION : throws doit être placé directement après la méthode et non inclus dans des { } ! |
| try catch | gère la nouvelle exception personnalisée comme une exception normale | voir plus haut | *Ex : création d'une nouvelle "ville"*  *Ville maVille = null;*  *try {*  *maVille = new Ville("Abc", 500);*  *} catch (NbHabNegatifException e) {*  *} finally {*  *if (maVille == null) {*  *maVille = new Ville();*  *}*  *}*  *System.out.println(maVille.nom + " à " + maVille.nbHab + " habitants.");* | déclaration à l'extérieur du bloc throws pour ne pas être une variable locale !  rien à préciser, le msg affiché est donné par l'objet lui-même  fait appel au constructeur par défaut s'il y a eu une exception  pour définir la ville et ne pas provoquer de blocage sur la dernière instruction (impossible si null !) |
|  | si plusieurs exceptions personnalisées à gérer en même temps | public MaClasse(param) throws MonException1, MonException2 {  if (condition1) {  throw new MonException1();  }  if(condition2) {  throw new MonException2();  }  else {  … |  |  |
|  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | |  | |  |
| Class |  |  | |  | |  |
| .class  .getClass() | permet d'obtenir l'objet Class associé à chaque classe du programme et stockant les informations relatives à l'élément auquel il appartient | Class c = MaClasse.class;  Class c = monObjet.getClass(); | |  | |  |
| .getName() | donne le nom de la classe (elle-même ou de la classe dont l'objet est issu) | c.getName(); | |  | |  |
| .getSuperclass() | donne le nom de la classe supérieure (Object ou autre classe si héritage) | c.getSuperclass(); | |  | |  |
| .getInterfaces() | renvoi un tableau de type Class avec la liste des interfaces implémentées par la classe | Class[] itfcs = c.getInterfaces(); | |  | |  |
| .getPackage() | permet de récupérer un Package depuis une classe de ce même package | Package monPack = MaClasse.class.getPackage(); | |  | |  |
| .getMethods() | renvoi un tableau de type Method avec la liste des méthodes de la classe (directes et celles héritées) | Method[] mtds = c.getMethods(); | | import java.lang.reflect.Method | |  |
| .getDeclaredFields() | renvoi un tableau de type Field avec la liste des champs (= variables d'instance et de classe) de la classe directe (mais pas celles héritées d'une classe mère) | Field flds = c.getDeclaredFields(); | | import java.lang.reflect.Field | |  |
| .getConstructors() | renvoi un tableau de type Constructor avec la liste des constructeurs de la classe directe | Constructor cstrtrs = c.getConstructors(); | | import java.lang.reflect.Constructor | |  |
| .getAnnotations() | renvoi un tableau de type Annotation avec la liste des annotations utilisable par l'objet en question  *(voir doc : Annotation Java)* | Annotation[] annotations = c.getAnnotations() | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
| Classes utilitaires natives | pré-chargée dans Java et utilisable directement | package Java.lang | |  | |  |
|  |  |  | |  | |  |
| Math |  |  | |  | |  |
|  | renvoi type double |  | |  | |  |
| .random() | renvoi un nb aléatoire entre 0 et 1 | Math.random(); | |  | |  |
| .abs( ) | valeur absolue | Math.abs(nb); | |  | |  |
| .pow( , ) | calcule la puissance d'un nb | Math.pow(nb, nbPuissance); | |  | |  |
| .sin( ) | renvoi le sin d'un angle donné en degré | Math.sin(angle); | |  | |  |
| .floor( ) | tronque un nb de type double | Math.floor(nb); | |  | |  |
|  | renvoi type int |  | |  | |  |
| .round( ) | arrondi le nb | Math.round(nb); | |  | |  |
|  |  |  | |  | |  |
| Random | classe permettant de générer un pseudo nombre aléatoire | *obtenir un nombre entier aléatoire [0; 100[*  Random random = new Random();  int nbAleat = random.nextInt(100); | |  | |  |
| SecureRandom | classe permettant de générer de vrais nombres aléatoires |  | | *génére un id aléatoire sur 60 bits encodé en base 16*  private String getRandomName() {  SecureRandom random = new SecureRandom();  return new BigInteger(60, random).toString(16);  } | |  |
|  |  |  | |  | |  |
| System | classe utilitaire |  | |  | |  |
| .out.print( ) | affiche le paramètre | System.out.print("Txt à afficher"); | |  | |  |
| .out.println( ) | affiche le paramètre avec un retour à la ligne juste après | System.out.println("Txt à afficher"); | |  | |  |
| .gc() | force le passage du ramasse-miète (Garbage Collector), cad MàJ la mémoire vive du prog et élimine les éléments innacessibles | System.gc(); | |  | |  |
|  |  |  | |  | |  |
| Classes utilitaires non-natives | non pré-chargée par Java, nécessitant l'appel d'un package |  | | Package : | |  |
|  |  |  | |  | |  |
| Scanner | enregistre des données claviers |  | | import java.util.Scanner; | |  |
| Scanner(System.in) | instanciation d'un objet de type Scanner pour enregistrer des données claviers | Scanner sc = new Scanner(System.in); | | Scanner sc = new Scanner(System.in);  System.out.println("Entrez un nombre :");  int nb1 = sc.nextInt();  System.out.println("Entrez un nombre :");  int nb2 = sc.nextInt();  System.out.println("Entrez un mot :");  sc.nextLine();  String str = sc.nextLine();  System.out.println("Vous avez saisi : " + nb1 + ", " + nb2 + " et " + str); | |  |
| .nextLine() | pour une chaine | String str = sc.nextLine(); | | ATTENTION : pour pouvoir utiliser .nextLine() après un .next*Nombre*(), on doit d'abord appeler la méthode seule avant d'enregistrer les données saisie dans la 2e |
| .nextInt()  .nextFloat() | pour un nb entier codé sur 4 octets  pour un nb réel codé sur 4 octets | int nb = sc.nextInt();  float nb = sc.nextFloat(); | |  |
|  |  |  | |  | |  |
| Date / Time |  |  | |  | |  |
| LocalDate | classe permettant de définir une date  (format yyyy-MM-dd) | LocalDate.now();  LocalDate.of(2019, 12, 1); => 1er decembre 2019  LocalDate.parse(“2019-12-01”); | | DateTimeFormatter formatter = DateTimeFormatter.ofPattern(“dd/MM/yy”);  LocalDate.parse(“01/12/19”, formatter); | |  |
|  |  | *obtenir la date du jour dans un format spécifique*  **DateTimeFormatter DATE\_STD\_FORMATTER = DateTimeFormatter.ofPattern(“dd/MM/yy”);**  **String today = LocalDate.now().format(DATE\_STD\_FORMATTER);** | | | |  |
| LocalTime | classe permettant de définir une heure  (format hh:mm:ss) | LocalTime.now();  LocalTime.of(14, 10, 55); => 14h 10mn 55s  LocalTime.parse(“14:10:55”); | | DateTimeFormatter formatter = DateTimeFormatter.ofPattern(“HH’h’ mm”);  LocalTime.parse(“14h 10”, formatter); | |  |
| LocalDateTime | classe permettant de définir une date et un heure  (format yyyy-MM-ddThh:mm:ss) | LocalDateTime.now();  LocalDateTime.of(2019, 12, 1, 14, 10, 55);  LocalDateTime.parse("2019-12-01T14:10:55"); | | DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yy HH'h'mm'min'ss's'");  LocalDateTime.parse("01/12/19 14h10min55s", formatter); | |  |
|  | DateTimeException |  | |  | |  |
|  |  |  | |  | |  |
| Period  Duration | classes permettant de travailler avec des durées temporelle | Period.of(1, 2, 4); => 1 an, 2 mois, 4 jours  Period.ofDays(4);  Period.ofWeeks(3);  Period.ofMonths(2);  Period.ofYears(1);  Period.between(finalDate, initialDate); | | Duration.of(435, ChronoUnit.SECONDS);  Duration.ofDays(4);  Duration.ofHours(3);  Duration.ofMinutes(2);  Duration.ofSeconds(1);  Duration.ofMillis(50);  Duration.ofNanos(700);  Duration.between(finalTime, initialTime); | |  |
| DateTimeFormatter | permet le formatage d'une date  DateTimeParseException | DateTimeFormatter FORMATTER = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss,SSS");  String dateFormatee = monDateTime.format(FORMATTER);  DateTimeFormatter.ofPattern("dd/MM/yy HH'h'mm'min'ss's'");  => "01/12/19 14h09min55s" | | |  |  |  |  | | --- | --- | --- | --- | | **y** | année | y *ou* yyyy  yy | 2019  19 | | **M** | mois | M MM  MMM  MMMM | 12 12  janv.  janvier | | **d** | jour du mois | d dd | 1 01 | | **E** | jour de la semaine | E  EEEE | dim.  dimanche | | **a** | AM/PM | a | AM PM | | **h** | heure (0-12) | h hh | 2 02 | | **H** | heure (0-24) | H HH | 14 14 | | **m** | minute | m mm | 9 09 | | **s** | seconde | s ss | 55 55 | | **S** | milliseconde | SSS | 000 | | |  |
|  |  |  | |  | |  |
| .get( ) | retourne tout ou partie de l’élément | maDate.toString(); => 2019-12-01  maDate.getDayOfWeek(); => SUNDAY  maDate.getDayOfMonth(); => 1  maDate.getDayOfYear(); => 335  maDate.getMonth(); => DECEMBER  maDate.getMonthValue(); => 12  maDate.getYear(); => 2019  maDate.getEra(); => CE | | monTime.toString(); => 14:10:55  monTime.getHour(); => 14  monTime.getMinute(); => 10  monTime.getSecond(); => 55  monTime.getNano(); => 0 | |  |
| .plus( ) | retourne une copie de l’élément augmentée de X unite  (l’objet maDate ou monTime n’est pas modifié !) | maDate.plus(5, ChronoUnit.DAYS);  maDate.plusDays(3);  maDate.plusMonths(2);  maDate.plusYeaurs(1);  maDate.plus(Period.ofDays(5)); | | monTime.plus(5, ChronoUnit.HOURS);  monTime.plusHours(3);  monTime.plusMinutes(2);  monTime.plusSeconds(1);  monTime.plusNanos(4);  monTime.plus(Duration.ofHours(5)); | |  |
| .minus( ) | retourne une copie de l’élément diminué de X unite  (l’objet maDate ou monTime n’est pas modifié !) | maDate.minus(5, ChronoUnit.MONTHS);  maDate.minusDays(3);  maDate.minusMonths(2);  maDate.minusYears(1);  maDate.minus(Period.ofMonths(5)); | | monTime.minus(5, ChronoUnit.HOURS);  monTime.minusHours(3);  monTime.minusMinutes(2);  monTime.minusSeconds(1);  monTime.minusNanos(4);  monTime.minus(Duration.ofHours(5)); | |  |
| .isAfter( )  .isBefore( ) | comparaison entre 2 éléments | maDateA.isAfter(maDateB); | | monTimeA.isAfter(monTimeB); | |  |
| .isLeapYear() | determine s’il s’agit d’une année bissextile | maDate.isLeapYear();  maDateTime.isLeapYear(); | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
| File | lien avec des fichiers externes |  | | import java.io | | Ce package traite les flux de données octet par octet |
| File( ); | créer un objet File pointant vers un élément de l'ordinateur : fichier, dossier, … | File monFile = new File("nomDuFichier.ext");  File monFile = new File("src/nomDuFichier.ext"); | |  | | Remarque : nom du fichier si dans le répertoire racine du projet Java |
| .getAbsolutePath() | donne l'adresse complète de l'élément | monFile.getAbsolutePath(); | |  | | Remarque : donne adresse ou nom tel que défini lors de l'instanciation, même si fichier non trouvé ! |
| .getName() | donne le nom de l'élément (et son extension si fichier) | monFile.getName(); | |  | |
| .lenght() | donne la taille du fichier (en ko) | monFile.lenght(); | |  | |  |
| .exists() | renvoi true si l'élément est présent | monFile.exists(); | |  | |  |
| .isDirectory() | renvoi true si l'élément est un dossier | monFile.isDirectory(); | |  | |  |
| .isFile() | renvoi true si l'élément est un fichier | monFile.isFile(); | |  | |  |
| .delete() | supprime l'élément | monFile.delete(); | |  | | ATTENTION : supprime directement sans l'envoyer à la corbeille ! |
| .mkdir( ) | crée le répertoire (le dossier) avec le nom placé en paramètre | monFile.mkdir("monDossier"); | |  | |  |
|  |  |  | |  | |  |
| .listRoots() | appelée sur n'importe quel objet de type File, elle fournit sous forme de tableau les lecteurs racines de l'ordi (C:\, D:\, lecteurs amovibles) | monFile.listRoots(); | | *Ex : affichage du contenu direct de ces éléments racines en distinguant dossiers et fichiers*  *voir cours p. 33* | |  |
| .listFiles() | retourne sous forme de tableau de File les éléments (dossiers et fichier) contenu dans le dossier sur lequel elle est apliquée | File[] filesTable = monFile.listFile(); | |  | |  |
|  |  |  | |  | |  |
| FileInputStream | objet de flux d'entrée permettant la lecture du contenu d'un fichier | FileInputStream fis = new FileInputStream(monFile);  FileInputStream fis = new FileInputStream(new File("monFichier"); | | *Code permettant la copie d'un fichier*  FileInputStream fis = null;  FileOutputStream fos = null;  try {  fis = new FileInputStream(new File("test.txt"));  fos = new FileOutputStream(new File("testCopie.txt"));  byte[] buffer = new byte[8];  int no = 0;  while ((no = fis.read(buffer)) >= 0) {  fos.write(buffer);  buffer = new byte[8];  }  System.out.println("Copie terminée !");  } catch (FileNotFoundException e1) {  //captée si fis ne trouve pas le fichier indiqué  e1.printStackTrace();  } catch (IOException e2) {  //captée lors d'une erreur de lecture ou d'écriture  e2.printStackTrace();  } finally {  //On referme les flux dans finally pour s'assurer que ces instructions soient bien accomplies même en cas d'exception !  try {  if (fis != null)  fis.close();  } catch (IOException e) {  e.printStackTrace();  }  try {  if (fos != null)  fos.close();  } catch (IOException e) {  e.printStackTrace();  }  } | |  |
| .read( ) | permet la lecture des données du fichier octet après octet et de les copier dans le tableau placé en paramètre | fis.read(monBuffer); | | Remarque : renvoi le nombre d'octet ayant été lu et stockée dans le buffer (= tampon) de type tableau  et -1 si plus rien à lire (fin du fichier) |
| FileOutputStream | objet de flux permettant l'écriture du contenu dans un fichier | FileOutputStream fis = new FileOutputStream(monFile);  FileOutputStream fis = new FileOutputStream(new File("monFichier"); | | Remarque : si le fichier indiqué n'existe pas, il le créé automatiquement |
| .write( ) | permet l'écriture des données placées en paramètre dans le fichier | fos.write(monBuffer);  (ici, le contenu de monBuffer est écrit dans le fichier défini lors de la création de l'objet fos) | |  |
| BufferedXxStream | objet de flux permettant une optimisation du temps de traitement (les données sont entièrement chargées en mémoire tampon avant leur traitement) | BufferedInputStream bis = new BufferedInputStream(new FileInputStream(new File("monFichier.txt"); | | DataOutputStream dos = new DataOutputStream(  new BufferedOutputStream(  new FileOutputStream(  new File("monFichier.txt")))); | | Remarque : il est possible (et même conseillé) de faire une instance à partir de plusieurs de ces objets afin d'accumuler leurs avantages |
| DataXxStream | objet de flux permettant de traiter directement les types primitifs | DataInputStream dis = new DataInputStream(new FileInputStream(new File("monFichier.txt");  dis.readDouble()  dis.readInt() | |
| LineNumberXxStream | objet de flux permettant de récupérer le numéro de la ligne en cours de traitement à l'instant T |  | |
| PushbackXxStream | objet de flux permettant de remettre un octet déjà traité dans le flux |  | |
| BufferedReader | objet de flux permettant not. la lecture d'un fichier ligne par ligne |  | | BufferedReader br = new BufferedReader(new FileReader(file));  String line;  while ((line = br.readLine()) != null) {  *// process the line.*  }  br.close(); | | ATTENTION : pour lecture d'un fichier texte ligne par ligne => voir Files.lines( ) depuis Java 8 |
| FileWriter  FileReader | objet de flux permettant des échanges de données directement sous forme de texte | FileWriter fw = new FileWriter(new File("monFichier.txt"); | | File file = new File("monFichier.txt");  FileWriter fw;  FileReader fr;  try {  fw = new FileWriter(file);  String str = "Ma chaine à sauvegarder \n dans un fichier.";  fw.write(str);    fr = new FileReader(file);  str = "";  int i = 0;  while((i = fr.read()) != -1)  str += (char)i;  System.out.println(str);  } catch (FileNotFoundException e) {  } catch (IOException e) {  } finaly {  if(fw != null) {  try {  fw.close();  } catch (IOException e) {  }  if(fr != null) {  try {  fr.close();  } catch(IOException e) {  }  } | |  |
|  |  |  | |  | |  |
| Serializable | interface marqueur  seul les objets issus de l'instanciation d'une classe implémentant cette interface peuvent être sérialisé  *(sérialisé = enregistré dans un fichier pour sauvegarde)* | public class MaClasse implements Serializable { | |  | | Attention : si des variables d'instances ou des méthodes de l'objet utilisent d'autres objets, il faut aussi que ces derniers soient sérialisable !  sinon, ils doivent être définit transient pour être ignorés lors de la sérialisation (voir ci-dessous) |
| transient | permet de définir une variable/méthode d'un objet à ignorer lors de la sérialisation | public class MaClasse implements Serializable {  private String maStrig;  private transient MaClasse2 monObjet; | |  | |  |
| ObjectInputStream | objet de flux permettant de lire un objet dans un fichier = désérialisation | ois = new ObjectInputStream(  new BufferedInputStream(  new FileInputStream(  new File("monFichier.txt"))));  try {  MaClasseObjet monObjet = (MaClasseObjet)ois.readObject();  } catch (ClassNotFoundException e) {  e.printStackTrace();  }    ois.close(); | | Code permettant l'écriture puis la lecture d'un objet  ObjectOutputStream oos = null;  try {  //localization du fichier cible ou création  oos = new ObjectOutputStream(  new BufferedOutputStream(  new FileOutputStream(  new File("monFichier.txt"))));  //Ecriture de l'objet dans le fichier  oos.writeObject(monObjet);  } catch (IOException e) {  //captée lors d'une erreur d'écriture  e.printStackTrace();  } finally {  //On s'assurer de fermeture du flux  try {  if (oos != null)  oos.close();  } catch (IOException e) {  e.printStackTrace();  }  }  --------------------------------------  ObjectInputStream ois = null;  try {  //localization du fichier cible  ois = new ObjectInputStream(  new BufferedInputStream(  new FileInputStream(  new File("monFichier.txt"))));  //Lecture de l'objet dans le fichier  MaClasseObjet monObjet = (MaClasseObjet)ois.readObject();  } catch (FileNotFoundException e1) {  //captée si ois ne trouve pas le fichier indiqué  e1.printStackTrace();  } catch (IOException e2) {  //captée lors d'une erreur de lecture  e2.printStackTrace();  } catch (ClassNotFoundException e3) {  //captée si classe de l'objet inadaptée  e3.printStackTrace();  } finally {  //On s'assurer de fermeture du flux  try {  if (ois != null)  ois.close();  } catch (IOException e) {  e.printStackTrace();  }  } | |  |
| .readObject() | méthode permettant de lire l'objet dans le fichier | Remarque : pour être utilisable, l'objet doit subir un cast pour lui définir son type ! |
| ObjectOutputStream | objet de flux permettant d'écrire un objet dans un fichier = sérialisation | oos = new ObjectOutputStream(  new BufferedOutputStream(  new FileOutputStream(  new File("monFichier.txt"))));  oos.writeObject(monObjet);  oos.close(); | |  |
| .writeObject( ) | méthode permettant l'écriture de l'objet placé en paramètre dans le fichier |  |
|  |  |  | |  | |  |
|  |  |  | | java.nio | | Ce package traite les flux de données par blocs |
| .getChannel( ) | méthode à appliquer sur un FileXxStream pour rendre le flux plus rapide et performant (mieux qu'avec BufferedXxStrem) |  | | fis = new FileInputStream(new File("monFichier.txt"));  fc = fis.getChannel(); //On récupère le canal  int size = (int) fc.size(); //On en déduit la taille  //buffer de taille et type adapté  ByteBuffer bBuff = ByteBuffer.allocate(size);  //Transfert des données du fichier vers le buffer  fc.read(bBuff);  bBuff.flip(); //prépare buffer la lecture  byte[] tabByte = bBuff.array(); //buffer => tableau  fis.close(); | |  |
| .size() | permet de connaitre la taille totale du fichier à transférer via le channel |  | |  |
| XxxBuffer | buffer à associer au flux pour le rendre plus performant | ByteBuffer ShortBuffer IntBuffer LongBuffer  FloatBuffer DoubleBuffer  CharBuffer | |  |
| .read() | permet le transfert des données vers le buffer |  | |  |
| .flip() | permet de préparer le buffer à la lecture |  | |  |
| .array() | permet de créer un tableau de données à partir du buffer |  | |  |
|  |  |  | |  | |  |
| Path et Files | destinées à remplacer la classe File |  | | java.nio | |  |
| Path | interface permettant la manipulation des chemins de fichiers | Path path = Paths.get("monFichier.txt"); | | path.toAbsolutePath(); => chemin absolu du fichier  path.getFileName(); | |  |
| Files | classe contenant des méthodes simplifiant les actions sur les fichiers | Files.exists(path);  Files.isDirectory(path);  Files.createFile(path);  Files.delete(path); | |  | |  |
| .copy( , , ) | copie de fichier | Path source = Paths.get("monFichier.txt");  Path cible = Paths.get("maCopie.txt");  try {  Files.copy(source, cible, StandardCopyOption.REPLACE\_EXISTING);  } catch (IOException e) {  } | | .REPLACE\_EXISTING  .COPY\_ATTRIBUTES  .ATOMIC\_MOVE => copie atomique  .NOFOLLOW\_LINKS => ne prend pas les liens en compte | |  |
| .move( , , ) | copie un fichier en le déplaçant | Path source = Paths.get("monFichier.txt");  Path cible = Paths.get(autreChemin/"maCopie.txt");  try {  Files.move(source, cible, StandardCopyOption.REPLACE\_EXISTING);  } catch (IOException e) {  } | |  | |  |
| DirectoryStream< > | lister le contenu d'un répertoire |  | //Récupération des éléments racines de la machine  Iterable<Path> roots = FileSystems.getDefault().getRootDirectories();  for(Path eltRacine : roots) {  System.out.println(eltRacine);  try(DirectoryStream<Path> contenu = Files.newDirectoryStream(eltRacine) {    for(Path nom : contenu)  System.out.println((Files.isDirectory(nom) ? nom + "/" : nom));  } catch (IOException e) {  }  ------------------------------------------------------  DirectoryStream<Path> contenu = Files.newDirectoryStream(eltRacine, "\*.txt") | | | Rq: listing du contenu de répertoire avec un filtre |
| XxxStream | ouverture de flux  *pouvant ici tous se faire dans un* try *afin de ne pas à avoir à les femrmer manuellement dans un bloc* finaly *(contrairement à* File*)* | try (InputStream intup = Files.newInputStream(path)) {  try(BufferedReader reader = Files.newBufferedReader(path, StandardCharset.UTF\_8) { | | |  |  |
| .lines( ) | lecture d'un fichier txt ligne par ligne | *lecture simple :*  try (Stream<String> lines = Files.lines(Paths.get("c://monFichier.txt"))) {  lines.forEach(System.out::println);  } catch (IOException e) {  e.printStackTrace();  }  *dans une méthode dédiée, lecture et évaluation :*  private static void searchForEdi() throws IOException {  Stream lines = Files.lines(Paths.get("c://jours.txt"));  Optional hasEdi = lines.filter(s -> s.contains("edi")).findFirst();  if(hasEdi.isPresent()){  System.out.println(hasEdi.get());  }  *//Fermer le flux*  lines.close();  } | | | |  |
|  |  |  | |  | |  |
| Thread | Multi-threading |  | |  | |  |
|  | sans l’API Java Concurency |  | | java.lang | |  |
| Thread( ) | constructeur permettant de créer un nouveau Thread en prenant en paramètre une classe implémentant Runnable | public static void main(…) {  Thread monThread = new Thread(new RunImpl());  monThread.start();  }  public class RunImpl implements Runnable {  *variables…*  *méthodes() {…}*  public void run() {  *instruction(s);*  }  } | | ***Avec classe Runnable anonyme :***  Thread monThread = new Thread(new Runnable() {  public void run() {  monBouton.setText("Cliqué " + (++count) + " fois");  }  });  monThread.start();  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  ***Thread anonyme :***  new Thread(new Runnable() {  public void run() {  monBouton.setText("Cliqué " + (++count) + " fois");  }  }).start(); | | la méthode run() doit être redéfinie dans la classe implémentant Runnable. C'est cette méthode qui sera appelée lors de l'invocation de monThread.start(); dans Main(…) |
| .sleep( ) | interruption du thread pendant un temps défini  InterruptedException | try {  Thread.sleep(1000);  } catch (InterruptedException e) {  e.printStackTrace();  } | |  | | Rq : l’interruption du thread n’est pas toujours exactement la valeur indiquée et peut varier de ± qqs milisecondes : ce phénomène est la dérive d’hrologe  Pour éviter ce phénomène => utiliser API Java Concurency |
| synchronized | les méthodes utilisées par plusieurs Threads différents sur un même objet doivent être définies comme synchronized | public synchronized void maMethode {  *instruction(s);*  } | |  | | l'utilisation de synchronized sur la méthode en question oblige le thread à terminer son action sur l'objet avant d'autoriser le thread suivant à faire appel à cette méthode. Indispensable si cette action modifie l'objet en question. |
|  |  |  | |  | |  |
|  | avec l’API Java Concurency | API Java Concurency *(depuis Java 5)* | | java.util.concurrent | |  |
|  |  |  | |  | |  |
| Executors | classe utilitaire implémentant ExecutorService et présentant des méthodes simples permettant d’obtenir une instance pré-paramétrée et dirrectement utilisable pour le multi-threading |  | |  | |  |
| .newSingleThreadExecutor() | créé un thread unique | ExecutorService monExecutor = Executors.newSingleThreadExecutor() ;  try {  monExecutor.execute(() -> {  *//mon traitement*  });  }catch(Exception e) {  log.error("Erreur lors de l’exécution du thread »);  }finally {  if(monExecutor != null) {  monExecutor.shutdown();  }  } | | ExecutorService monExecutor = Executors.newSingleThreadExecutor() ;  try {  Future<MonObjet> monObjFut = monExecutor.submit(() -> {  *//mon traitement*  return monObjet;  });  *//exploitation du résultat avec temps d’attente max*  System.out.println("Résultat = " + monObjFut.get(3, TimeUnit.SECONDS));  }catch(Exception e) {  log.error("Erreur lors de l’exécution du thread »);  }finally {  if(monExecutor != null) {  monExecutor.shutdown();  }  } | |  |
| .newFixedThreadPool( ) | créé un pool de N threads qui s’exécuterons en parallèle | ExecutorService monExecutor = Executors.newFixedThreadPool(3);  try {  Callable<MonObjet> monThreadA = () -> {  *// traitement*  return monObjet;  }  …  List<Callable<MonObjet>> mesThreads = Arrays.asList(monThreadA, monThreadB, monThreadC);  List<Future<MonObjet>> mesResultats = monExecutor.invokeAll(mesThreads);  }catch(Exception e) {  }finally {  if(monExecutor != null) {  monExecutor.shutdown();  }  } | |  | |  |
| .newCachedThreadPool() | prépare un pool réunitisable de threads  la création concrète du thread sera faite à la demande | ExecutorService monExecutor = Executors.newCachedThreadPool();  try {  monExecutor.execute(() -> {  *//mon traitement*  });  Future<MonObjet> monObjFut = monExecutor.submit(() -> {  *//mon traitement*  return monObjet;  });  }catch(Exception e) {  log.error("Erreur lors de l’exécution du thread »);  }finally {  if(monExecutor != null) {  monExecutor.shutdown();  }  } | |  | |  |
| .newSingleThreadScheduledExecutor() | crée un thread unique paramétrable pour pouvoir s'exécuter périodiquement ou après un délai spécifié | SheduledExecutorService monExecutor = Executors.newSingleThreadScheduledExecutor() ;  try {  ScheduledFuture<MonObjet> monObjSFut = monExecutor.schedule(() -> {  *//mon traitement*  return monObjet;  }, 10, TimeUnit.SECONDS);  *//exploitation du résultat avec temps d’attente max*  System.out.println("Résultat = " + monObjSFut.get(3, TimeUnit.SECONDS));  }catch(Exception e) {  log.error("Erreur lors de l’exécution du thread »);  }finally {  if(monExecutor != null) {  monExecutor.shutdown();  }  } | |  | |  |
| .newScheduledThreadPool( ) | crée un pool de N de threads paramétrables pour s'exécuter périodiquement ou après un délai spécifié | SheduledExecutorService monExecutor = Executors. .newScheduledThreadPool(3);  try {  Callable<MonObjet> monThreadA = () -> {  *// traitement*  return monObjet;  }  …  List<Callable<MonObjet>> mesThreads = Arrays.asList(monThreadA, monThreadB, monThreadC);  List<ScheduledFuture<MonObjet>> mesResultats = monExecutor.invokeAll(mesThreads);  }catch(Exception e) {  }finally {  if(monExecutor != null) {  monExecutor.shutdown();  }  } | |  | |  |
|  |  |  | |  | |  |
| ExecutorService | interface permettant de créer et gérer les Thread |  | |  | |  |
| .execute( ) | exécute un thread de type Runnable (cad implémentant Runnable et ne retournant rien) |  | |  | |  |
| .submit( ) | exécute un thread de type Runnable et retournant un objet qui peut être de tout type encapsulé dans un Future |  | |  | |  |
| .submit( ) | exécute un thread de type Callable (cad implémentant Callable et retournant un objet de type T) |  | |  | |  |
| .invokeAny( ) | exécute une collection de threads de type Callable en retournant uniquement l’objet résultant du premier thread à se terminer (les autres threads sont alors interrompus) |  | |  | | *Exemple d'utilisation : optimisation de la recherche d'un élément dans un grand ensemble => on effectue une dichotomie de ce grand ensemble et affecter chaque sous-ensemble à un thread. Dès qu'un thread à trouvé l'élément cherché il interromp les autres* |
| .invokeAll( ) | exécute une collection de threads Callable et retourne une liste de résultats Future<T> dont l'ordre respecte le même positionnement des threads dans la collection d'entrées |  | |  | | ATTENTION : la liste des résultats obtenue n’est utilisable que lorsque TOUS les threads en entrée ont terminé leur exécution |
| .shutdown() | demande l'arrêt explicite d'un thread s'il n'a plus de tâche à exécuter. | …  }finally {  if(monExecutor != null) {  monExecutor.shutdown();  }  } | |  | | ATTENTION : un thread créé via l'API ExecutorService doit être explicitement arrêté s'il a terminé son travail (pour libérer la mémoire de la JVM) ! |
|  |  |  | |  | |  |
| .awaitTermination( , ) | interomp le processus principal (celui qui à lancé les threads secondaires)  utile dans le cas où le résultat des threads est nécéssaire au traitement dans la suite du processus principal | monExecutor.awaitTermination(50, TimeUnit.SECONDS); | |  | |  |
|  |  |  | |  | |  |
| ScheduledExecutorService | interface permettant de créer des threads et de gerer leur temps de démarrage |  | |  | | Rq: classe héritant des méthodes d'ExecutorService |
| .schedule( , , ) | identique à la méthode .submit( ) en ajoutant un délai initial avant le lancement du thread Runnable ou Callable |  | |  | |  |
| .scheduleWithFixedDelay( , , , ) | exécute un thread une 1re fois après le délai initial (parm 2), puis de manière periodique après le délai fixe (param 3) déclanché à la fin du traitement précédent |  | | monExecutor.scheduleWithFixedDelay(monThread, 3, 5, TimeUnit.SECONDS); | | Rq: dans l'exemple, si tps traitement A = 2s, tps trait B = 7s et tps trait C = 4s alors on aura la séquence :  3s-2s-5s-7s-5s-4s-5s-… |
| .scheduleAtFixedRate( , , , ); | exécute un thread une 1re fois après le délai initial (parm 2), puis de manière periodique après le délai minimal imposé (param 3) incluant le temps du traitement (si le traitement est plus long que le délai, le traitement suivant démarre imédiatement) |  | | monExecutor.scheduleAtFixedRate(monThread, 3, 5, TimeUnit.SECONDS); | | Rq: dans l'exemple, si tps traitement A = 2s, tps trait B = 7s et tps trait C = 4s alors on aura la séquence :  3s-2s+3s-7s-4s+1s-… |
|  |  |  | |  | |  |
| Future | objets encapsulant l’objet résultant de l’exécution d’un Thread  (même principe que Optionnal) |  | |  | |  |
| .isDone() | teste la terminaison du thread pour savoir si la donnée résultante de son exécution est disponible | monObjFut.isDone(); | |  | |  |
| .get() | retourne le résultat de la tâche exécutée par le thread | monObjFut.get(); | |  | | Remarque : 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 |
| .get( , ) | retourne le résultat de la tâche exécutée par le thread avec un temps d’attente max  TimeoutException | monObjFut.get(3, TimeUnit.SECONDS); | |  | |  |
|  |  |  | |  | |  |
|  |  |  | |  | |  |

String.format(" ", )

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Code** | **Type** | **Resultat** | | **%a** | nb réel (except BigDecimal) | valeur hexadecimale d'un nb réel | | **%b** | Any type | “true” si non-null, “false” sinon | | **%c** | character | caractère Unicode | | **%d** | nb entier (inclu bigint) | valeur du nb entier | | **%e** | nb réel | nb en notation scientifique | | **%f** | nb réel | valeur du nb réel | | **%g** | nb réel | nb réel ou notation scientifique selonla valeur | | **%h** | tous types | Hex String of value from hashCode() method. | | **%n** | / | retour à la ligne | | **%o** | nb entier | valeur octale | | **%s** | tous types | String | | **%S** | tous types | String en Uppercase | | **%t** | Date/Time (long, Calendar, Date and TemporalAccessor) | Date/Time (voir ci-contre) | | **%x** | nb entier (inclu bigint) | valeur hexadecimale | | |  |  | | --- | --- | | **Date/Time format** | | | %tA | Full name of the day of the week, e.g. “Sunday“, “Monday“ | | %ta | Abbreviated name of the week day e.g. “Sun“, “Mon“, etc. | | %tB | Full name of the month e.g. “January“, “February“, etc. | | %tb | Abbreviated month name e.g. “Jan“, “Feb“, etc. | | %tC | Century part of year formatted with two digits e.g. “00” through “99”. | | %tc | Date and time formatted with “%ta %tb %td %tT %tZ %tY” e.g. “Fri Feb 17 07:45:42 PST 2017“ | | %tD | Date formatted as “%tm/%td/%ty“ | | %td | Day of the month formatted with two digits. e.g. “01” to “31“. | | %te | Day of the month formatted without a leading 0 e.g. “1” to “31”. | | %tF | ISO 8601 formatted date with “%tY-%tm-%td“. | | %tH | Hour of the day for the 24-hour clock e.g. “00” to “23“. | | %th | Same as %tb. | | %tI | Hour of the day for the 12-hour clock e.g. “01” – “12“. | | %tj | Day of the year formatted with leading 0s e.g. “001” to “366“. | | %tk | Hour of the day for the 24 hour clock without a leading 0 e.g. “0” to “23“. | | %tl | Hour of the day for the 12-hour click without a leading 0 e.g. “1” to “12“. | | %tM | Minute within the hour formatted a leading 0 e.g. “00” to “59“. | | %tm | Month formatted with a leading 0 e.g. “01” to “12“. | | %tN | Nanosecond formatted with 9 digits and leading 0s e.g. “000000000” to “999999999”. | | %tp | Locale specific “am” or “pm” marker. | | %tQ | Milliseconds since epoch Jan 1 , 1970 00:00:00 UTC. | | %tR | Time formatted as 24-hours e.g. “%tH:%tM“. | | %tr | Time formatted as 12-hours e.g. “%tI:%tM:%tS %Tp“. | | %tS | Seconds within the minute formatted with 2 digits e.g. “00” to “60”. “60” is required to support leap seconds. | | %ts | Seconds since the epoch Jan 1, 1970 00:00:00 UTC. | | %tT | Time formatted as 24-hours e.g. “%tH:%tM:%tS“. | | %tY | Year formatted with 4 digits e.g. “0000” to “9999“. | | %ty | Year formatted with 2 digits e.g. “00” to “99“. | | %tZ | Time zone abbreviation. e.g. “UTC“, “PST“, etc. | | %tz | Time Zone Offset from GMT e.g. “-0800“. | |
| *Index de l'argument*  String.format("%2$s", 32, "Hello"); // prints: "Hello"  *Taille min*  String.format("|%20d|", 93); // prints: | 93|  String.format("|%-20s|", "Hello"); // prints: |Hello |  *nb de caractère max*  String.format("|%.5s|", "Hello World"); // prints: |Hello|  *nb chiffre ap virgule max (=> arrondie)*  String.format("|%.5f|", 10,26548568); // prints: |10,27|  *Nombre*  String.format("|%020d|", 93); // prints: |00000000000000000093|  String.format("|%+d|', 93); // prints: |+93| => affiche + si nb positif  String.format("|%+d|', -36); // prints: |-36|  String.format("|% d|", 93); // prints: | 93| => garde espace pour le signe  String.format("|% d|", -36); // prints: |-36|  String.format("|%,d|", 10000000); // prints: |10,000,000|  String.format("|%#o|", 93); // prints: 0135  String.format("|%#x|", 93); // prints: 0x5d  String.format("|%#X|", 93); // prints: 0X5D |