**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 | String[] mesSousChaines = maChaine.split(";"); | | | |  | | |  | | |
| .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} | | |  | | |
|  |  |  | | | |  | | |  | | |
|  |  |  | | | |  | | |  | | |
| 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;  } | | | | public enum MonEnum {  VALEURA("Valeur A"),  VALEURB("Valeur B");  private String valeur = "";  private MonEnum(String str) {  this.name = str;  }  public String toString() {  return valeur;  }  } | | | 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);  } | | | |  | | |  | | |
|  |  |  | | | |  | | |  | | |
| 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> (); | | | |  | | | 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( , , ) | échage 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) |  | | | |  | | | 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 | | |
|  |  |  | | | |  | | |  | | |
| 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 | RetournerConstante rcPi = () -> 3.14159265359d;  System.out.println(rcPi.getConstante();  RetournerConstante rcPi = () -> {System.out.print("Pi = "); return 3.14159265359d;};  System.out.println(rcPi.getConstante(); | | | | *avec interface fonctionelle perso*  public interface RetournerConstante {  public double 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) -> {instruction; instruction; return valeur;}; | | | | import java.util.function.\*;  Function<Personne, String> fNom = (Personne p) -> p.getNom();  fNom.apply(personageA);  Consumer<Personne> vieillir10 = (Personne p) -> p.setAge(p.getAge() + 10);  vieillir10.accept(personnageA); | | |  | | |
| :: | 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(); | | | | |  | | |
|  |  |  | | | |  | | |  | | |
|  |  |  | | | |  | | |  | | |
|  |  |  | | | |  | | |  | | |
|  |  |  | | | |  | | |  | | |
|  |  |  | | | |  | | |  | | |
|  |  |  | | | |  | | |  | | |
| 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( , ) | 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, (x) -> x + 1).forEach(System.out::println); |  | | |
| .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()* | | | | | |  |  | | |
|  | 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 égal au rang placé en paramètre | monStream.skip(20); => renvoi le 21e et les suivants | | | |  | | |  | | |
| .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); | | |  | | |
| .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()); | | | | | | |  | | |
|  | méthodes terminales consommant le stream |  | | | |  | | |  | | |
| .forEach( ) | parcours les éléments | monStream.forEach(Consumer); | | | | stream.forEach(System.out::println); | | |  | | |
| .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); | | |  | | |
| .count() | retourne le nombre d'éléments | monStream.count(); | | | |  | | |  | | |
| .collect( ) | rassemble les éléments dans un seul objet (liste, set, map, …) | List<Type> maListe = monStream.collect(Collectors.toList()); | | | |  | | |  | | |
| .findFirst() | retourne le premier élément du stream encapsulé dans un objet Optionnal | Optionnal<Type> opt = monStream.findFirst(); | | | |  | | |  | | |
| 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)); | | | | | |  | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 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); | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
| 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(); | |  | |
|  |  |  | |  | | | |  | |
| 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 |  | |
| 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.newBufferedWriter(path, StandardCharset.UTF\_8) { | | |  | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
| Fenêtre |  |  | |  | | | |  | |
| JFrame | Classe permettant d'instancier une nouvelle fenêtre | JFrame maFenetre = new JFrame(); | | javax.swing | | | |  | |
| .setVisible( ) | permet d'afficher la fenêtre à l'écran | maFenetre.setVisible(true); | |  | | | | ATTENTION : cette méthode ne doit être appelée que lorsque tous les paramètres d'affichages de la fenêtre ont étés définis | |
| .setTitle( ) | donne un titre à la fenêtre | maFenetre.setTitle("Titre de ma fenêtre"); | |  | | | |  | |
| .setDefaultCloseOperation( ) | défini le comportement lorsque la fenêtre est fermée  arret complet du programme  fermeture de la fenetre uniquement | maFenetre.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  JFrame.EXIT\_ON\_CLOSE  JFrame.DISPOSE\_ON\_CLOSE | |  | | | |  | |
| .setSize( , ) | défini les dimensions (largeur, hauteur) en pixel de la fenêtre | maFenetre.setSize(nb, nb); | |  | | | |  | |
| .setLocationRelativeTo( ) | défini la position relative de la fenêtre  centrée | maFenetre.setLocationRelativeTo(null); | |  | | | |  | |
| .setLocation( , ) | défini la postition en pixel du coin sup gauche de la fenêtre par rapport au coin sup gauche de l'écran | maFenetre.setLocation(nb, nb); | |  | | | |  | |
| .setResizable( ) | empêche le redimensionnement de la fenêtre si false (true par défaut) | maFenetre.setResizable(false); | |  | | | |  | |
| .setAlwaysOnTop( ) | garde la fenêtre toujours au premier plan si true (false par défaut) | maFenetre.setAlwaysOnTop(true); | |  | | | |  | |
| .setUndecorated( ) | retire le contour et les boutons de contrôles si true (false par défaut) | maFenetre.setUndecorated(true); | |  | | | |  | |
|  |  |  | |  | | | |  | |
| Panel |  |  | | javax.swing | | | |  | |
| JPanel | objet permettant l'instanciation d'un nouveau panel (= panneau) qui sert à construire notre page avant de l'insérer dans la fenêtre | JPanel monPanel = new JPanel(); | | *Portion de code intéressant :*  //Icone  icon = new JLabel(new ImageIcon("imgs/icone.jpg"));  JPanel panIcon = new JPanel();  panIcon.setBackground(Color.WHITE);  panIcon.setLayout(new BorderLayout());  panIcon.add(icon);  //Le nom  JPanel panNom = new JPanel();  panNom.setBackground(Color.white);  panNom.setPreferredSize(new Dimension(220, 60));  nom = new JTextField();  nom.setPreferredSize(new Dimension(100, 25));  panNom.setBorder(BorderFactory.createTitledBorder("Nom du personnage"));  nomLabel = new JLabel("Saisir un nom : ");  panNom.add(nomLabel);  panNom.add(nom); | | | |  | |
| .setContentPane( ) | méthode de l'objet JFrame (= fenêtre) permettant d'insérer le contenu du panel dans la fenêtre | maFenetre.setContentPane(monPanel); | |  | | | | ATTENTION : ici aussi cette méthode ne doit être appliquée qu'une fois toutes les modifications du Panel réalisées | |
|  |  |  | |  | | | |  | |
| Mise en page |  |  | | Java.awt | | | |  | |
| Layout managers | mise en page des éléments dans la fenêtre |  | |  | | | |  | |
| .setLayout( ) | méthode à appliquer aux objets JFrame ? et JPanel pour controler la mise en page  elle prend en paramètre un objet de type Layout | FlowLayout monLayout = new FlowLayout();  maFenetre.setLayout(monLayout); ???  monPanel.setLayout(monLayout); | |  | | | |  | |
| FlowLayout | centre tous les éléments sur une ligne et passe à la ligne suivante quand plus de place | monPanel.setLayout(new FlowLayout()); | |  | | | | Remarque : par défaut pour les éléments JFrame | |
| BorderLayout | place les objets selon 4 emplacements cardinaux + centre | monPanel.setLayout(new BorderLayout());  monPanel.add(monBouton, BorderLayout.SOUTH); | |  | | | | Remarque : par défaut pour les éléments JPanel | |
| GridLayout | découpe la page en un tableau dont on choisit le nb de ligne et colonne | monPanel.setLayout(new GridLayout(nbLi, nbCol));  monPanel.add(monBouton); | |  | | | |  | |
| .setHgap( )  .setVgap( ) | définit l'espace (en pxl) entre   * les colonnes (horizontal) * les lignes (vertical) | monGridLayout.setHgap(nb); | |  | | | |  | |
| BoxLayout | range les éléments en une seule :  ligne  colonne | monPanel.setLayout(new BoxLayout(monPanel, BoxLayout.LINE\_AXIS));  monPanel.setLayout(new BoxLayout(monPanel, BoxLayout.PAGE\_AXIS)); | |  | | | |  | |
| Box | permet aussi de créer des "Box" à utiliser comme des JPanel pour les remplir et les insérer dans le JPanel ou directement dans la fenêtre (ContentPane) | Box b1 = Box.createHorizontalBox();  b1.add(monBouton);  Box b2 = Box.createVerticalBox();  monPanel.add(b1); | |  | | | |  | |
| CardLayout | créer plusieurs panneaux superposés dont un seul est visible et pouvoir passer de l'un à l'autre facilement grâce à leur nom | CardLayout maCL = new CardLayout();  monConteneur.setLayout(maCL);  monConteneur.add(monPanel1, "Nom panel1"); | |  | | | |  | |
| .next( ) | affiche dans la page le panel suivant dans la liste | maCL.next(monConteneur); | |  | | | |  | |
| .show( ) | affiche dans la page le panel correspondant au nom donné | maCL.show(monConteneur, "Nom panelX"); | |  | | | |  | |
| GridBagLayout | permet de définir avec précision l'emplacement et la taille relative des éléments dans notre fenêtre découpée selon une grille infinie et basée sur le premier élément inséré | monPanel.setPreferredSize(new Dimension(nb, nb);  monPanel.setLayout(new GridBagLayout()); | |  | | | | Remarque : on doit définir une dimension initiale de du panel pour servir de référence pour les dimensions relatives des éléments | |
| GridBag  Constraints | Classe permettant de définir les propriétés d'affichage d'un élément avant de l'insérer dans le GrigBagLayout | GridBagConstraints gbc = new GridBagConstraints();  gbc.gridx = 0; //1er élément de la ligne  gbc.gridy = 1; //2e colonne  gbc.gridwidth = 1; //largeur d'une case  gbc.gridheight = 2; //hauteur de 2 case  gbc.fill = GridBagConstraints.VERTICAL; //la cellule à fusionnée 2 cases verticales  monPanel.add(panel5, gbc); | |  | | | | Pour plus de détail sur ce Layout : PDF cours p. 289 et 291 | |
| Onglets |  |  | | javax.swing | | | |  | |
| JTabbedPane | objet accueillant plusieurs JPanel afin de créer des onglets dans la fenêtre JFrame | JPanel panA = new JPanel(), panB = new JPanel();  mesOnglets = new JTabbedPane();  mesOnglets.add("Onglet A", panA);  mesOnglets.add("Onglet B", panB);  this.getContentPane().add(mesOnglets, BorderLayout.CENTER); | | *En précisant la position de la barre d'onglets dans le constructeur :*  mesOnglets = new JTabbedPane(JTabbedPane.LEFT);  RIGHT  BOTTOM | | | | Rq: par défaut, la position de la barre d'onglets est en haut de la fenêtre. Elle peut être modifié en le précisant dans le constructeur ou avec la méthode .setTabbedPlacement( ) qui prend les même paramètres. | |
| .setIconAt( , ) | ajout d'une image devant le titre de l'onglet  param : (index de l'onglet, ImageIcon) | mesOnglets.setIconAt(0, new ImageIcon("adresse")); | |  | | | | Rq: l'index des onglets commence à 0, comme pour les tableaux | |
| .setTabPlacement( ) | modifie la position de la barre d'onglets | mesOnglets.setTabPlacement(JTabbedPane.LEFT);  TOP, RIGHT, BOTTOM | |  | | | |  | |
| .remove( ) | retire l'onglet de l'index indiqué | mesOnglets.remove(1); | |  | | | |  | |
| .getSelectedIndex() | renvoi l'index de l'onglet actuellement actif dans la fenêtre | int indexOngletActif = mesOnglets.getSelectedIndex(); | |  | | | |  | |
| Fenêtre interne |  |  | |  | | | |  | |
| JDesktopPane  JInternalFrame | conteneur permettant d'insérer des fenêtre mobiles dans la page | class MaFenetrePrincipale extends JFrame {  desktop = new JDesktopPane();  miniFenetre = new JIntenalFrame();  desktop.add(miniFenetre);  this.getContentPane().add(desktop, Layout…);  } | | classe interne automatisant la création de mini-fenetres :  class MaFenetrePrincipale extends JFrame {  private static nbFenetre = 0;  private static int xy = 10;  class MiniFenetre extends JInternalFrame {  public MiniFenetre() {  xy += 4;  this.setTitle("MiniFenetre" + (++nbFenetre));  this.setSize(150,80);  this.setLocation(xy, xy);  this.setResizable(true);  this.setClosable(true);  this.setVisible(true);  }  }  } | | | |  | |
| JWindow | fenêtre sans contour et les controles classique (fermer, réduire, …) souvent utilisé pour les splash screens (écran de chargement, …) |  | | class Window extends JWindow {  public Window() {  this.setSize(220, 170);  this.setLocationRelativeTo(null);  Jpanel pan = new JPanel();  JLabel img = new JLabel(new ImageIcon("adresse"));  img.setVerticalAlignment(JLabel.CENTER);  img.setHorizontalAlignment(JLabel.CENTER);  pan.setBorder(BorderFactory.createLineBorder(Color.BLUE);  pan.add(img);  this.getContentPane().add(pan);  this.setVisible(true);  }  } | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
| Boutons |  |  | | javax.swing | | | |  | |
| JButton | classe permettant l'instanciation d'un bouton | JButton monBouton = new JButton("Etiquette"); | |  | | | |  | |
| .setText( ) | modifie l'étiquette du bouton | monBouton.setText("Nouvelle étiquette"); | |  | | | |  | |
| .setEnabled( ) | rend le bouton cliquable si true (par défaut) ou non si false | monBouton.setEnabled(false); | |  | | | |  | |
|  |  |  | |  | | | |  | |
| Texte |  |  | | javax.swing | | | |  | |
| JLabel | classe permettant l'instanciation d'un élément de type texte | JLabel monLabel= new JLabel("Mon Texte"); | |  | | | |  | |
| .setText( ) | modifie le texte du label | monLabel.setText("Mon nouveau texte"); | |  | | | |  | |
| .setFont( ) | définie la police du texte en prenant en paramètre un objet de type Font | Font maPolice = new Font("nomPolice", Font.BOLD, 30);  monLabel.setFont(maPolice); | |  | | | |  | |
| .setHorizontalAlignment( )  .setVerticalAlignment( ) | définit la position du label dans son conteneur | monLabel.setHorizontalAlignment(JLabel.CENTER); | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
| Décoration des éléments | méthodes communes à quasiment tous les composant et permettant leur mise en page, mise en forme, décoration, … | si non précisé = valable pour monConteneur et monElement | |  | | | |  | |
| .setPreferredSize( ) | défini la taille par défaut à l'affichage | .setPreferedSize(new Dimension(Xpx, Ypx); | |  | | | |  | |
| .setForeground( ) | modifier la couleur de la police | .setForeground(Color.BLUE); | |  | | | |  | |
| .setVerticalAlignment( )  .setHorizontalAlignment( ) | contrôle l'alignement d'un objet dans son conteneur | monElement.setVerticalAlignment(JLabel.CENTER);  monElement.setHorizontalAlignment(JLabel.LEFT); | |  | | | |  | |
| .setBorder( ) | bordure des éléments | .setBorder(BorderFactory.createTitledBorder("Titre de la bordure"));  .setBorder(BorderFactory.createLineBorder(Color.BLUE)); | | *Autres types de bordures :*  .createBevelBorder(BevelBorder.LOWERED, Color.black, Color.red),  .createEtchedBorder(Color.BLUE, Color.GRAY),  .createLineBorder(Color.green),  .createMatteBorder(5, 2, 5, 2, Color.MAGENTA),  .createRaisedBevelBorder(),  .createTitledBorder("Titre"),  .createCompoundBorder(borderA, borderB) | | | | Rq: ici Compound = Bevel + Matte | |
|  |  |  | |  | | | |  | |
| .setToolTipText( ) | affiche une bulle d'information au survol prolongé de la souris sur l'élément | .setToolTipText("Info complémentaires sur l'élément"); | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
| JScrollPane | ajoute des ascenseurs au panel lorsque le contenu dépasse la taille de la page | monPanel = new JPanel();  JScrollPane scroll = new JScrollPane(monPaneau);  this.getContentPane().add(scroll, BorderLayout.CENTER); | |  | | | | ATTENTION : l'objet JScrollPane encapsule le Panel sur lequel il est appliqué. Pour afficher cet élément, il faut afficher le Scroll. | |
| .setHorizontalScrollBarPolicy( )  .setVerticalScrollBarPolicy( ) | défini le comportement du scroll horizontal du composant  3 paramètres possibles :  scroll apparait quand c'est nécessaire  n'apparait jamais  reste tjs affiché | monScroll.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);  JScrollPane.HORIZONTAL\_SCROLLBAR\_AS\_NEEDED  JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER  JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS | | (par défaut) | | | | Rq: idem pour Vertical. | |
|  |  |  | |  | | | |  | |
| Barre de séparation mobile |  |  | |  | | | |  | |
| JSplitPane | défini une barre de séparation mobile entre deux éléments de type JComponent (JPanel, JButton, JComboBox, JRadioButton, …) | JPanel pan1 = new JPanel(), pan2 = new JPanel();  split = new JSplitPane(JSplitPane.HORIZONTAL\_SPLIT, pan1, pan2);  this.getContentPane().add(split, BorderLayout.CENTER); | |  | | | | ATTENTION : une fois le split appliqué sur les Panel, ces derniers appartiennent au composant split (le Split encapsule les autres). Pour les afficher tous les trois, il suffit donc d'insérer le split dans la page. | |
| .setOneTouchExpandable( ) | affiche les boutons sur la barre pour pouvoir faire complètement disparaitre l'un ou l'autre des conteneurs | split.setOneTouchExpandable(true); | |  | | | |  | |
| .setDividerSize( ) | défini la taille de la barre (en pixel) | split.setDividerSize(5); | |  | | | |  | |
| .setDividerLocation( ) | défini la position de la barre lors de l'affichage de la fenêtre (en pixel par rapport à la gauche ou au haut) | split.setDividerLocation(10); | |  | | | |  | |
|  |  |  | |  | | | |  | |
| JSlider | barre permettant de suivre en direct la valeur et de sélectionner un nb dans un intervalle donné | slider = new JSlider();  slider.setMaximum(100);  slider.setMinimum(0);  slider.setValue(30);  slider.setPaintTicks(true);  slider.setPaintLabels(true);  slider.setMinorTickSpacing(10);  slider.setMajorTickSpacing(20);  slider.addChangeListener(new ChangeListener() {  public void stateChanged(ChangeEvent e) {  label.setText("Valeur actuelle : " + ((JSlider)e.getSource()).getValue());  }  });  monPanel.add(slider, BorderLayout.CENTER); | |  | | | |  | |
|  |  |  | |  | | | |  | |
| JProgressBar | barre de progression / de chargement  progression en % visible  valeur entre 0 et 500 liée à la progression du traitement | class ProgressFenetre extends JFrame {  public ProgressFenetre() {  bar = new JProgressBar();  bar.setMaximum(500);  bar.setMinimum(0);  bar.setStringPainted(true);    this.getContentPane().add(bar, BorderLayout.CENTER);  t = new Thread(new Traitement());  t.start();  this.setVisible(true);  class Traitement implements Runnable {  public void run() {  instruction(s);  bar.setValue(valeurDeProgression);  }  }  }  } | |  | | | | Rq: la barre de progression ne peut suivre que les calculs d'un autre Thread, sinon on n'observera pas la progression (elle passera directement de 0 à 100%) | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
| Graphics | les objets de type Graphics sont générés automatiquement par le programme lors de l'affichage d'un composant dans la fenêtre mais aussi à chaque redimensionnement, …  Ces objets vont alors être utilisés pour modifier et personnaliser nos éléments (panneau, bouton, …) | | | java.awt | | | |  | |
| paintComponent(Graphics g) | méthode qui sera lancé à chaque fois qu'un objet Graphics sera généré pour l'objet auquel elle appartient | public class monElement extends JXXX {  public void paintComponent(Graphics g) {  instruction(s) pour modifier l'apparence;  }  } | |  | | | |  | |
| .repaint() | méthode à appeler sur un objet contenant un élément Graphics pour lancer la méthode paintComponent( ) | monElement.repaint(); | |  | | | |  | |
|  | ATTENTION : toutes les méthodes suivantes sont à utiliser dans la méthode paintComponent( ) { |  | |  | | | |  | |
| .getWidth() | donne la largeur de l'élément | this.getWidth(); | |  | | | |  | |
| .getHeight() | donne la hauteur de l'élément |  | |  | | | |  | |
| .setColor( ) | modifie la couleur de l'élément tracé en précisant une couleur reconnue par le programme | g.setColor(Color.maCouleur);  *g.setColor(Color.RED);* | |  | | | |  | |
| .drawOval( , , , ) | dessine un ovale vide avec (abscisse, ordonnée, largeur, hauteur) en pixel | g.drawOval(x, y, w, h);  *g.drawOval(10, 10, 50, 75);* | |  | | | |  | |
| .fillOval( , , , ) | dessine un ovale plein | g.fillOval(x, y, w, h | |  | | | |  | |
| .drawRect( , , , ) | dessine un rectangle vide | g.drawRect(x, y, w, h) | |  | | | |  | |
| .fillRect( , , , ) | dessine un rectangle plein | g.fillRect(x, y, w, h) | |  | | | |  | |
| .drawRoundRect( , , , , , ) | dessine un rectangle vides avec bords arrondis, les 2 derniers paramètres sont (arcWidth, arcHeight) | g.drawRoundRect(x, y, w, h, aw, ah) | |  | | | |  | |
| .fillRoundRect(, , , , , ) | dessine un rectangle plein avec bords arrondis | g.fillRoundRect(x, y, w, h, aw, ah) | |  | | | |  | |
| .drawLine( , , , ) | trace une ligne droite depuis le point initial jusqu'au point final | g.drawLine(xi, yi, xf, yf)  *g.drawLine(0, 10, this.getWidth(), 10);* | |  | | | |  | |
| .drawPolyline( , , ) | trace une ligne brisée passant par tous les points de coordonnées définies par les tableaux x et y, nbS indique le nb de point à utiliser | g.drawPolyline(x[ ], y[ ], nbS)  *int[] abs = {50, 100, 75, 25, 0};*  *int[] ord = {0, 33, 100, 100, 33};*  *g.drawPolyline(abs, ord, 5);* | |  | | | |  | |
| .drawPolygon( , , ) | trace un polygone dont on définit les coordonnées de tous les points dans 2 tableaux (un pour x, l'autre pour y), nbS indique le nb de sommet du polygone (la figure se referme automatiquement) | g.drawPolygon(x[ ], y[ ], nbS) | |  | | | |  | |
| .fillPolygon( , , ) | idem en plein | g.fillPolygon(x[ ], y[ ], nbS) | |  | | | |  | |
| .drawString( , , ) | affiche du texte aux coordonnées indiquées (coin bas. gauche du texte) | g.drawString("Ma phrase à afficher", x, y); | |  | | | |  | |
| .setFont( ) | modifie la police du texte en prenant en paramètre un objet de type Font | Font maPolice = new Font("nomPolice", Font.BOLD, 20);  g.setFont(maPolice); | |  | | | |  | |
| .drawImage( , , , , , ) | affiche une image dans notre élément  elle comprend 4 param. obligatoires et 2 facultatifs :   * l'image * coord. du coin sup gauche * *(les dimensions)* * son conteneur (souvent : this) | try {  Image monImg = ImageIO.read(new File("adresse/image.jpg");  g.drawImage(monImg, x, y, *w, h,* cont);  } catch (IOException e) {  e.printStackTrace();  } | |  | | | |  | |
| Graphics2D | permet d'utiliser des effets plus avancés que Graphics simple | public void paintComponent(Graphics g) {  Graphics2D g2d = (Graphics2D)g;  g2d. …  } | |  | | | | Remarque : pour pouvoir utiliser les effets de Graphics2D il faut caster notre objet Graphics en objets Graphics2D | |
| GradientPaint( , , , , , , ) | Classe permettant de définir un gradiant de couleur avant de l'appliquer à un élément | GradientPaint gp = new GradientPaint(x1, y1, Color.couleur1, x2, y2, Color.couleur2, bool);  g2d.setPaint(gp);  g2d.fillRect(0, 0, this.getWidth(), this.getHeight()); | |  | | | | * x1, y1 coord. de départ de 1re couleur * x2, y2 coord. de départ de 2e couleur * booléen true : répétition gradient, false sinon | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
| Prog. Evènementielle |  |  | | java.awt | | | |  | |
|  | la mise en place d'un gestionnaire d'événement sur un élément se fait en 3 étapes :   * implémentation de sa classe avec l'interface de l'évènement * ajout du gestionnaire sur l'élément lui-même * redéfinition de la/des méthodes qui nous intéresse | public class Bouton implements MouseListener {  this.addMouseListener(this);  public void mouseEntered(MouseEvent e) {  instruction(s);  } | |  | | | |  | |
| MouseListener | gestionnaire d'événement souris  clic sur l'élément  survol de l'élément  curseur sort de la zone de l'élément  bouton gauche enfoncé sur l'élément  bouton gauche de la souris relâché | implements MouseListener {  this.addMouseListener(this);  public void mouseClicked(MouseEvent e) {  instruction(s);  }  mouseClicked( ) {  mouseEntered( ) {  mouseExited( ){  mousePressed( ){  mouseReleased( ){ | | *Avec classe anonyme :*  JPanel pan = new JPanel();  pan.addMouseListener(new MouseAdapter() {  public void mouseReleased(MouseEvent e) {  instruction(s);  }  }); | | | | Dans une classe anonyme, la déclaration d'un MouseListener peut se faire avec MouseListenr(), mais MouseAdapter() présente l'avantage de ne pas avoir à redéfinir toutes les méthodes de MouseListener, seulement celle(s) qui nous intéresse(nt) (plus pertinent pour une classe anonyme réservée aux cas les plus simples de gestionnaires) | |
| MouseMotionListener | gestionnaire d'événement suivant le mouvement de la sourie  suit les mvts de la souris dans le composant sans clic  suit les mvmts de la souris dans le composant qd clic gauche maintenu enfoncé | implements MouseMotionListener {  public void mouseMoved(MouseEvent e) {  instruction(s);  }  public void mouseDragged(MouseEvent e) {  instruction(s);  } | |  | | | |  | |
| KeyListener | gestionnaire d'évènement clavier  pression sur une touche  entre deux : touche pressée non relâchée  relâchement de la touche | implements KeyListener {  this.addKeyListener(this);  public void keyPressed(KeyEvent e) {  instuction(s);  }  keyTyped( ) {  keyReleased( ) { | | *Avec classe interne :*  class Fenêtre extends JFrame {  …  jtf = new JTextField();  jtf.addKeyListener(new ClavierListener());  }  class ClavierListener implements KeyListener {  public void keyPressed(KeyEvent e) {  …  …  } | | | |  | |
| e.getKeyCode() | retourne le code de la touche |  | |  | | | |  | |
| e.getKeyChar() | retourne le caractère de la touche |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
| ActionListener | gestionnaire d'événement qui permet de réagir dans une autre classe à un événement intervenu sur un élément présentant son propre gestionnaire | implements ActionListener {  monObjet.addActionListener(this);  public void actionPerformed(ActionEvent e) {  instruction(s);  } | |  | | | | Remarque : monObjet est l'élément qui présente déjà un gestionnaire classique (souris, …)  une seule méthode à redéfinir pour l'action | |
| e.getSource() | donne le nom de l'élément sur lequel à eu lieu l'action détectée,  utile si un même ActionListener "écoute" plusieurs événements et y réagit différement | e.getSource(); | |  | | | |  | |
| classe interne | si une seule classe doit suivre plusieurs événements, il est préférable de déclarer une classe interne pour chaque gestionnaire ActionListener afin de ne pas surcharger la méthode actionPerformed( ) | monBouton.addActionListener(MaClasseInterne); | | *Exemple de déclaration avec classe interne :*  class Fenetre extends JFrame {  …  JButton monBouton = new JButton("Go");  monBouton.addActionListener(new BoutonGoListener());  …  }  class BoutonGoListener implements ActionListener {  public void actionPerformed(ActionEvent e) {  instruction(s);  }  } | | | |  | |
| classe anonyme | dans un cas simple de gestion d'événement (1 ou 2 lignes de code) il est plus rapide de faire appel à une classe anonyme | monBouton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent event) {  instruction(s);  }  }); | |  | | | |  | |
| ItemListener | gestionnaire d'évènement spécifique aux liste déroulantes (JComboBox)  *voir* Cours - Formulaires) |  | |  | | | |  | |
|  |  |  | |  | | | |  | |
| Drag 'n Drop | Glisser-Déposer |  | |  | | | |  | |
| .setDragEnabled( ) | permet d'activer le Drag par défaut pour le contenu de certains éléments  (peut être personalisé en attribuant ensuite un TransferHandler personalisé au composant) | JTextArea monTxtArea = new JTextArea("Contenu déplassable");  monTxtArea.setDragEnable(true);  monTxtAream.setTransferHandler(new MonTransferHandler); | | *Composant permettant*   |  |  |  | | --- | --- | --- | | *Drag* | *Drop* | *Les 2* | | JFileChooser  JTable  JTree  JList | JPasswordTextField | JEditorPane  JColorChooser  JTextPane  JTextField  JTextArea  JFormattedTextField | | | | | Rq: JLabel ne permet ni l'un, ni l'autre ! | |
| TransferHandler  .exportAsDrag( , , ) | Pour tous les JCompnent ne présentant pas cette option par défaut, on doit :  lui affecter un TransferHandler (qui permet de gérer les données transférées entre les éléments lors du déplacement)  amorcer la procédure du Drag n Drop lors de l'événement souris  crée un racourcis vers les données cibles  n'autorise rien | monComposant.setTransferHandler(new TransferHandler("Text"));  monComposant.addMouseListener(new MouseAdapter() {  public void mousePressed(MouseEvent e) {  JComponent monComp = (JComponent)e.getSource();  TransferHandler tHandler = monComp.getTransferHandler();  tHandler.exportAsDrag(monComp, e, TransferHandler.COPY);  }  });  TransferHandler.COPY  TransferHandler.MOVE  TransferHandler.COPY\_OR\_MOVE  TransferHandler.LINK  TransferHandler.NONE | | | | *Avec TransferHandler personalisé :*  monComposant.setTransferHandler(new MonTransferHandlerPerso()); | | Rq: possibilité de personnaliser entièrement le TransferHandler *(voir Java-ExempleCodes)*  ATTENTION : si TransferHandler est instancié avec "Text", il n'acceptera que .COPY | |
| DropLocation | objet contenu dans TransferSupport et permettant des actions spécifiques avec les JList, JTree et JTable | *Dans le TransferHandler :*  public boolean importData(TransferHandler.TransferSupport support){  if(!canImport(support))  return false;    JTree.DropLocation dl = (JTree.DropLocation)support.getDropLocation();  *Dans la méthode de construction du JTree :*  tree.setDragEnabled(true);  tree.setTransferHandler(new MonTreeTransferHandler(tree));  tree.setDropMode(DropMode.ON\_OR\_INSERT); | | | | | |  |  | | --- | --- | | *JList* | *JTable* | | dl.isInsert()  dl.getIndex() | dl.isInsertRow()  dl.isInsertColumn()  dl.getRow()  dl.getColumn() | | *JTree* | | dl.getChildIndex()  dl.getPath() | | | *(voir Java-ExempleCodes)* |
|  |  |  | |  | | | |  | |
|  |  |  | |  | | | |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | | |  | | | | |  | | |
| Thread |  |  | | | 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(…) | | |
| 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. | | |
|  | InterruptedException | try {  Thread.sleep(1000);  } catch (InterruptedException e) {  e.printStackTrace();  } | | |  | | | | |  | | |
| EDT | Event Dispatch Thread = thread reservé aux composants graphiques |  | | |  | | | | |  | | |
| SwingUtilities | classe offrant plusieurs méthodes statiques permettant d'insérer des instructions liès aux composants graphisques dans l'EDT |  | | | Méthodes thread-safe pouvant être appellées depuis n'importe quel thread :  .paint() .repaint()  .validate() .revalidate() .invalidate() | | | | |  | | |
| .invokeLater( ) | exécute le thread passé en paramètre dans l'EDT et rend immédiatement la main au thread principal | Thread t = new Thread(new Runnable() {  public void run() {  instruction(s);  }  });  SwingUtilities.invokeLater(t); | | | Thread t = new Thread(new Runnable() {  public void run() {  monBouton.setText("Cliqué " + (++count) + " fois");  }  });  if(SwingUtilities.isEventDispathThread()) {  t.start();  } else {  SwingUtilities.invokeLater(t);  } | | | | |  | | |
| .inkokeAndWait( ) | idem, mais attends la fin du traitement avant la reprise du thread principal | InterruptedException  InvocationTargetException | | |  | | |
| .isEventDispatchThread() | retourne true si le thread dans lequel cette méthode est apellé s'éxécute dans l'EDT | SwingUtilities.isEventDispathThread() | | |  | | |
| SwingWorker<T, V> | classe abstraite à utiliser pour réaliser des traitments lourds devant intéragir avec EDT  interaction avec EDT :   * dirrectement dans done() * par setProgress(int nb) dans doInBackground() via un PropertyChangeListener | class MaSwingWorker extends SwingWorker<ObjectA, ObjectB> {  *//méthode appelée par .execute()*  protected ObjectA doInBackground() throws Exception {  *instruction(s) en tâche de fond;*  setProgress(nb); *//changement de la propriété d'état*  publish(monObjectB); *//publication intermédiaire*  return monObjectA;  }  *//méthode appelée à la fin du traitement de doInBackground()*  public void done() {  *instruction(s) à faire dans l'EDT;*  try {  ObjectA monObjectA = get();  } catch (InterruptedException e) {  } catch (ExecutionException e) {  }  }  *//méthode gérant les résultats intemédiaires*  public void process(List<ObjectB> list) {  instruction(s);  }  }  *Dans le thread principal :*  MaSwingWorker maSW = new maSW();  maSW.addPropertyChangeListener(new PropertyChangeListener() {  public void propertyChange(PropertyChangeEvent e) {  if("progress".equals(e.getPropertyName()))  *instruction(s) à faire d'ans l'EDT*  (Integer)e.getNewValue());  }  });  maSW.execute(); | | | | | | *Classe anonyme :*  SwingWorker sw = new SwingWorker() {  protected Object doInBackground() throws Exception {  *instruction(s) en tâche de fond;*  }  public void done() {  *instruction(s) à faire dans l'EDT;*  }  };  sw.execute(); | | ATTENTION : seuls .done() et le PropertyChangeListener s'assurent que ses instructions se fassent dans l'EDT.  Rq: .done() est appelée automatiquement une fois à la fin du traitement de .doInBackground()  Rq: seul .doInBackground() doit être obligatoirement redéfinie, les autres méthades et listeners étants facultatifs. | | |
| get() | récupère le résultat (de type ObjetA) retourné à la fin du traitement de doInBackground() | pubic void done() {  ObjetA objA = get(); | | |  | | | | |  | | |
| cancel()  isConcelled() | essaie d'interrompre la tache de doInBackground() en cours  renvoie true si tache intérompue | pubic void done() {  cancel();  if(isCancelled()) { | | |  | | | | |  | | |
| isDone() | retourne true si l'action de doInBackground est finie | pubic void done() {  if(isDone()) { | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
| Formulaires |  |  | | | java.swing | | | | |  | | |
| JComboBox( ) | liste déroulante  le constructeur peut prendre un tableau d'objet en paramètre  sinon, utilisation de la méthode .addItem( ) | combo = new JComboBox();  combo.addItem(monObjetA);  combo.addItem(monObjetB);  combo.setPreferredSize(new Dimension(100, 20));  String[ ] tableauDeChoix = {"Option A", "Option B", … };  combo = JComboBox(tableauDeChoix);  combo.setPreferredSize(new Dimension(100, 20)); | | | *Avec un type générique :*  JComboBox<String> combo = new JComboBox<String>(); | | | | | Rq: pour afficher l'option dans la liste, JComboBox appellera la méthode .toString() des objets qui lui sont placés en paramètre | | |
| .setSelectedIndex( ) | assigne un choix par défaut à la liste (facultatif) | combo.setSelectedIndex(2); | | |  | | | | |  | | |
| ActionListener | pour obtenir la valeur sélectionnée en direct par l'utilisateur, on utilise une classe implémentant ActionListener | combo.addActionListener(new ItemAction());  class ItemAction implements ActionListener {  public void actionPerformed(ActionEvent e) {  System.out.println("Action sur : " + combo.getSelectedItem());  }  } | | |  | | | | | Rq: la classe peut être déclarée en interne dans le même fichier si instructions pas trop longues.  ATTENTION : la méthode .getSelectedItem() renvoie un objet et non l'intitulé du choix (ici : *Option B*). Pour obtenir ce dernier, on utilise .toString() | | |
|  |  |  | | |  | | | | |  | | |
| JCheckBox | case à cocher | maCheckBox = new JCheckBox("Choix 1"); | | |  | | | | |  | | |
| .setSelected( ) | case cochée par défaut (facult.) | maCheckBox.setSelected(true); | | |  | | | | |  | | |
| ActListener |  | (JCheckBox)e.getSource()).getText();  (JCheckBox)e.getSource()).isSelected();  maCheckBox.isSelected(); | | | => Choix 1  => true / false  => true / false | | | | |  | | |
| JRadioButton | bouton radio | JRadioButton jrb1 = newJRadioButton("Choix A");  JRadioButton jrb2 = newJRadioButton("Choix B");  ButtonGroup bg = new ButtonGroup();  bg.add(jrb1);  bg.add(jrb2); | | | *Insertions des boutons dans le panel :*  monPanel.add(jrb1);  monPanel.add(jrb2); | | | | | ATTENTION : ButtonGroup ne peut être ajouté au panel en une seule étape. On doit ajouter les boutons un à un après avoir lié leur traitement avec ButtonGroup | | |
| .setSelected( ) | case pré-selectionnée (facult.) | jrb2.setSelected(true); | | |  | | | | |  | | |
| ActListener |  | jrb1.getText();  jrb2.isSelected; | | | => Choix A  => true / false | | | | |  | | |
| JTextField | champ de texte | JTextField jtf = new JTextField("Valeur par défaut"); | | |  | | | | |  | | |
| .setText( ) | remplis le contenu du champs | jtf.setText("Nouvelle valeur par défaut"); | | |  | | | | |  | | |
| .getText() | récupère le contenu du champ | String contenu = jtf.getText(); | | |  | | | | |  | | |
| .setPreferedSize( ) | définit les dimensions par défaut | jtf.setPreferedSize(new Dimension(300, 30)); | | |  | | | | |  | | |
| .setForegroung( ) | définit la couleur de la police | jtf.setForeground(Color.BLUE); | | |  | | | | |  | | |
| .setFont( ) | définit la police à utiliser | Font police = new Font("Arial", Font.BOLD, 14);  jtf.setFont(police); | | |  | | | | |  | | |
| JFormattedTextField | champ de texte au contenu formaté (= n'acceptant qu'un nombre entier, une date, un horaire, un pourcentage, …) | jftf = new JFormattedTextField(NumberFormat.getIntegerInstance()); | | |  | | | | | Rq: la méthode .getText() renverra alors   * le contenu du champ s'il est valide * rien si contenu non valide et pas de valeur valide précédente * valeur valide précédente si le champ à déjà été utilisé et que la nouvelle valeur n'est pas valide | | |
|  | formatassions de bases :  nombre entier  pourcentage  nb jusqu'à 3 chiffres après la virgule  avec format à définir | NumberFormat.getIntegerInstance()  NumberFormat.getPercentInstance()  NumberFormat.getNumberInstance()  DateFormat.getTimeInstance()  DateFormat.getDateInstance()  MessageFormat | | |  | | | | |  | | |
| MaskFormatter | objet permettant de définir un format personnalisé grâce aux métacaractères | try {  MaskFormatter monFormat = new MaskFormatter(maChaineMetachar);  JFormattedTextField jftf = new JFormattedTextField(monFormat);  } catch (ParseException e) {  e.printStackTrace();  } | | | *Métacaractères :*   * ' : caractère d'échappement * \* : n'importe quel caractère * # : chiffre * A : chiffre ou lettre * ? : lettre * U : lettre (min -> MAJ) * L : lettre (MAJ -> min) * H : caractère hexadécimal | | | | | *Exemple*  *n° de tel : "## ## ## ## ##"* | | |
| JTextArea | zone de texte étendue  comme un JTextField en plus grand  (méthodes identiques à JTextField) | JTextArea textPane = new JTextArea(); | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
| Boite de dialogue |  |  | | | javax.swing | | | | |  | | |
| JOptionPane | boite de dialogue | JOptionPane jop = new JOptionPane; | | |  | | | | |  | | |
| types de messages | 5 types de messages permettent de définir les boites de dialogues qui suivent |  | | | *5 types de messages possibles :*  JOptionPane.PLAIN\_MESSAGE  JOptionPane.INFORMATION\_MESSAGE  JOptionPane.QUESTION\_MESSAGE  JOptionPane.WARNING\_MESSAGE  JOptionPane.ERROR\_MESSAGE | | | | |  | | |
| .showMessageDialog( ) | défini une boite de dialogue informative  peut prendre 2, 4 ou 5 paramètres :   * **composant parent (si pas :** null**)** * **corps du message** * titre de la fenêtre * type de message * *image de l'icone* | jop.showMessageDialog(Component parentComponent, String message);  jop.showMessageDialog(Component parentComponent, String message, String titre, int typeDeMessage);  jop.showMessageDialog(Component parentComponent, String message, String titre, int typeDeMessage, ImageIcon image); | | | *Création d'un objet de type* ImageIcon *:*  ImageIcon img = new ImageIcon("Adresse de l'image depuis la racine du projet Eclipse"); | | | | |  | | |
| .showConfirmDialog( ) | défini une boite de confirmation permettant de valider, invalider ou annuler l'action en cours  elle retourne un entier en fonction du choix de l'utilisateur  Paramètres :   * composant parent (si pas : null) * corps du message * titre de la fenêtre * **type d'options dans la fenêtre** * type de message * *image de l'icone* | int option = jop.showConfirmDialog(null, "Etes-vous sûr ?", "Demande de confirmation", JOptionPane.YES\_NO\_OPTION, JOptionPane.QUESTION\_MESSAGE);  if(option == JOptionPane.OK\_OPTION) {  instruction(s);  } | | | *Types d'options proposés par la fenêtre :*  JOptionPane.YES\_NO\_OPTION  JOptionPane.YES\_NO\_CANCEL\_OPTION  *4 valeurs de renvoie sont possibles :*  JOptionPane.OK\_OPTION = JOptionPane.YES\_OPTION = 0  JOptionPane.NO\_OPTION = 1  JOptionPane.CANCEL\_OPTION = 2  JOptionPane.CLOSED\_OPTION = -1 | | | | |  | | |
| .showInputDialog( ) | défini une boite de saisie  retourne une String  Paramètres :   * composant parent (si pas : null) * corps du message * titre de la fenêtre * type de message * *image de l'icone* | String maChaine = jop.showInputDialog(null, "Entrez une valeur", "Demande info", JOptionPane.INFORMATION\_MESSAGE); | | |  | | | | |  | | |
|  | pour définir une liste déroulante dans une boite de saisie, il suffit de rajouter des paramètres :   * 6e : tableau de valeur * 7e : valeur par défaut de la liste | String[ ] monTableau = {"Choix 1", "Choix 2", … };  String maChaine = jop.showInputDialog(null, "Entrez une valeur", "Demande info", JOptionPane.INFORMATION\_MESSAGE, null, monTableau, monTableau[2]); | | |  | | | | | Rq: ici, le 2nd null dans les paramètres correspond à l'image de l'icône si on ne souhaite pas lui en attribuer une.  ATTENTION : comme avec une ComboBox, la valeur retournée est l'objet du tableau lui-même et non pas une String | | |
| JDialog | classe permettant de créer une boite de dialogue personnalisée  initComponent() contiens ensuite les instructions pour remplir la boite de dialogue avec les mêmes éléments qu'une fenêtre classique (JPanel, Layout, JButton, JTextField, JComboBox, JRadioButton, ...)  et les gestionnaires associés pour la récupération des données saisies dans cette boite de dialogue  Le dialogue débute quand la fenêtre devient visible : maDialogBox.setVisible(true);  et se termine quand elle disparait :  maDialogBox.setVisible(false);  instructions gérées par des Listeners | class maDialogBox extends JDialog {  public maDialogBox(JFrame parent, String title, boolean modal) {  //utilise le contructeur de JDialog  super(parent, title, modal);  this.setSize(550, 270);  this.setLocationRelativeTo(null);  this.setResizable(false);  this.setDefaultCloseOperation(  JDialog.DO\_NOTHING\_ON\_CLOSE);  this.initComponent();  }  private void initComponent() {  instruction(s);  } | | | *Exemple complet de boite de dialogue personnalisée :*  *voir* Eclipse-Java projet : cours 10-boiteDial  *Portion de code intéressant :*  //Icone  icon = new JLabel(new ImageIcon("imgs/icone.jpg"));  JPanel panIcon = new JPanel();  panIcon.setBackground(Color.WHITE);  panIcon.setLayout(new BorderLayout());  panIcon.add(icon);  //Le nom  JPanel panNom = new JPanel();  panNom.setBackground(Color.white);  panNom.setPreferredSize(new Dimension(220, 60));  nom = new JTextField();  nom.setPreferredSize(new Dimension(100, 25));  panNom.setBorder(BorderFactory.createTitledBorder("Nom du personnage"));  nomLabel = new JLabel("Saisir un nom : ");  panNom.add(nomLabel);  panNom.add(nom); | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
| Menus |  |  | | | javax.swing | | | | |  | | |
| JMenuBar | barre de menus |  | | | public class maFenetre extends JFrame {  private JMenuBar menuBar = new JMenuBar();  private JMenu menuA = new JMenu("Fichier");  private JMenu menuAa = new JMenu("Sous-fichier");  private JMenu menuB = new JMenu("Edition");  private JMenuItem itemA1 = new JMenuItem("Ouvrir");  private JMenuItem itemAa1 = new JMenuItem("Sous-option 1");  private JMenuItem itemAa2 = new JMenuItem("Sous-option 2");  private JMenuItem itemA2 = new JMenuItem("Fermer");  …  public maFenetre() {  …  this.menuA.add(itemA1);  this.manuAa.add(itemAa1);  this.menuAa.add(itemAa2);  this.menuA.add(menuAa);  this.menuA.addSeparator();  this.itemA2.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  System.excit(0);  }  });  this.menuA.add(itemA2);  this.menuB.add(…);  this.menuBar.add(menuA);  this.menuBar.add(menuB);  this.setJMenuBar(menuBar);  this.setVisible(true);  } | | | | |  | | |
| JMenu | menu ou sous-menu |  | | |  | | |
| JMenuItem | élément du menu induisant une action lorsqu'il est sélectionné |  | | | *Dans un Listener :*  ((JMenuItem)e.getSource()).getText() | | |
| .add( ) | permet d'insérer un élément dans un menu ou dans la barre de menu |  | | |  | | |
| .addSeparator() | ajoute un séparateur entre deux éléments ou deux menus |  | | |  | | |
| .setJMenuBar( ) | A appliquer sur l'objet maFenetre pour ajouter la barre de menu préalablement définie à la fenêtre |  | | |  | | |
| JCheckBoxMenuItem | case à cocher spécifique aux menus |  | | |  | | | | |  | | |
| JRadioButtonMenuItem | boutons radio spécifiques aux menus |  | | |  | | | | | *Dans un Listener :*  ((JRadioButtonMenuItem)e.getSource()).getText() | | |
|  |  |  | | |  | | | | |  | | |
| Raccourcis claviers |  |  | | |  | | | | |  | | |
| .setMnemonic( ) | définit un raccourci clavier mnémonique (accessible par alt + touche) donnant accès aux menus  pour JMenu et JMenuItem  paramètre : char | monJMenu = new JMenu("Fichier", 'F');  monJMenu.setMnemonic('F');  monJMenuItem.setMnemonic('I'); | | |  | | | | |  | | |
| .setAccelerator( ) | définit un raccourci clavier accélérateur (combinaison de touches) effectuant une action directe  pour JMenuItem | monJMenuItem.setAccelerator(KeyStroke.getKeyStroke('a'));  monJMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK\_A, KeyEvent.CTRL\_DOWN\_MASK));  monJMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK\_A, KeyEvent.CTRL\_DOWN\_MASK + KeyEvent.SHIFT\_DOWN\_MASK)); | | |  | | | | | ATTENTION : dans le premier cas, 'a' ≠ 'A' (= Maj + a)  Rq: VK\_ = Value of Key  ATTENTION : .getKeyStroke ne prend   * qu'un param : char * 2 param : VK\_ lettre/chiffre , CTRL   si on veut Ctrl + Maj + lettre : VK\_ , CTRL + SHIFT | | |
|  |  |  | | |  | | | | |  | | |
| Menu contextuel | = menu clic droit |  | | |  | | | | |  | | |
| JPopupMenu | menu contextuel se remplissant avec des JMenu et JMenuItem | jpm = new JPopupMenu();  jpm.add(jmenuA);  jpm.add(jmenuitem1); | | | *Déclaration d'un menu contextuel s'affichant au clic droit sur un JPanel :*  monPanel = new JPanel();  monPanel.addMouseListener(new MouseAdapter() {  public void mouseReleased(MouseEvent e) {  if(e.isPopupTrigger()) {  jpm.show(monPanel, e.getX(), e.getY());  }  }  });  container.add(monPanel, BorderLayout.CENTER);  maFenetre.setContentPane(container); | | | | | ATTENTION : pour une même instruction dans la barre de menus et dans le menu contextuel, DEUX JMenuItem doivent être créés et insérés un dans chaque menu (car les JMenuItem n'apparaissent qu'une seule fois dans la fenêtre : dans le menu auquel il est affecté en dernier dans le code)  En revanche, ils peuvent être suivit par la même instance de Listener ! | | |
| .show( , , ) | affiche le menu contextuel  param :   * composant parent qui invoque le menu * position en X * position en Y |  | | |  | | |
| .getInvoker() | permet d'appeler un 2nd JPopupMenu à partir du premier | jpm.getInvoker();  jpm.getInvoker().getX();  jpm.getInvoker().getY(); | private void initPopupMenu() {  JMenuItem ajouterEmplMI = new JMenuItem("Ajouter Employé");  ajouterEmplMI.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  initEmplPotentielsMenu();  emplPotentielsMenu.show(jpm.getInvoker(), me.getX(), me.getY());  }  });  jpm.add(ajouterEmplMI);  } | | | | | | |  | | |
| Barre d'outils |  |  | | |  | | | | |  | | |
| JToolBar | barre d'outil se remplissant avec des JButton | JToolBar maBarreOutils = new JToolBar();  JButton bouton1 = new JButton(new ImageIcon("imgs/monimage.jpg"));  Color fondBouton = Color.WHITE;  bouton1.setBackground(fondBouton);  bouton1.addActionListener(monListener);  maBarreOutils.add(bouton1);  maFenetre.add(maBarreOutil, BorderLayout.NORTH); | | *Cas où les boutons de la barre d'outils font référence à des boutons radio dans le menu : appel à ces bouton radio par la méthode* .doClick()  class FormeListener implements ActionListener {  public void actionPerformed(ActionEvent e) {  //si l'action vient d'un bouton radio, ok  if(e.getSource().getClass().getName().equals("javax.swing.JRadioButtonMenuItem"))  pan.setForme((JRadioButtonMenuItem)e.getSource()).getText());  else { //si vient de barre d'outils : appel artificiel au bouton radio correspondant  if(e.getSource() == square)  carre.doClick();  else if(e.getSource() == tri)  triangle.doClick();  else  rond.doClick();    }  }  } | | | | | | | | Rq: ici, carre, triangle et rond sont des JRadioButtonMeniItem, alors que square et tri sont des JButton insérés dans la barre d'outils |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
| Arbres |  |  | | |  | | | | |  | | |
| JTree  DefaultMutableTreeNode | défini une arborescence (= succession de nœuds qui peuvent être indifféremment des dossier ou des fichiers) | racine = new DefaultMutableTreeNode("Racine");  doss = new DefaultMutableTreeNode("DossierA");  fichierA1 = new DefaultMutableTreeNode("Fichier1");  fichier2 = new DefaultMutableTreeNode("Fichier 2");  doss.add(fichierA1);  racine.add(doss);  racine.add(fichier2);  arbre = new JTree(racine);  maFenetre.getContentPane().add(new JScrollPane(arbre)); | | | | *Création arbre à partir lecteurs racines de l'ordinateur :* ***voir Java-ExempleCodes*** | | | |  | | |
| .setRootVisible( ) | appliqué à l'objet JTree avec un booléen pour ne pas afficher l'élément "racine" s'il ne correspond à rien | arbre.setRootVisible(false); | | |  | | | | |  | | |
| TreeSelectionListener | listener de l'objet JTree permettant d'interagir avec lui  ici en classe anonyme  mais possible avec implémentation | arbre.addTreeSelectionListener(new TreeSelectionListener() {  public void valueChanged(TreeSelectionEvent e) {  if(arbre.getLasteSelectedPathComponent() != null) {  instruction(s);  }  }  }); | | | | |  | | |  | | |
| .getLastSelectedPathComponent().toString() | appliqué sur JTree : donne le nom de l'élément selectionné | arbre.getLastSelectedPathComponent().toString() | | |  | | | | |  | | |
| e.getPath() | retourne un objet TreePath contenant une liste d'objet correspondant aux nœuds du chemin d'accès à un élément de l'arbre | arbre.getSelectedPathComponent | | | *Donne le chemin d'accès complet à l'élément sélectionné :*  *(dans* valueChanged(…) *du Listener :)*  getAbsolutePath(e.getPath());  *(2nd méthode du Listener :)*  private String getAbsolutePath(TreePath tp) {  String str = "";  //on balaie le contenu de l'objet TreePath  for(Object name : tp.getPath()) {  //si l'objet a un nom, on l'ajoute au chemin d'accés  str += name.toString();  }  return str;  } | | | | | Rq : avec cette méthode dans le Listener, pour récupérer un objet, il suffit de faire :  File file = new File(getAbsolutePath(e.getPath)); | | |
| Modifier le contenu d'un arbre | ajouter/supprimer/renommer les éléments  => voir ***cours - Modifier le contenu de nos arbres*** |  | | |  | | | | |  | | |
| DefaultTreeCellRenderer | modifier la mise en forme de l'arbre  icone de dossier fermé  icone de dossier ouvert  icone de fichier | dtcr = new DefaultTreeCellRenderer();  dtcr.setClosedIcon(new ImageIcon("adresse"));  dtcr.setOpenIcon(…);  dtcr.setLeafIcon(…);  arbre.setCellRenderer(dtcr); | | | *Utiliser le "look and feel" de notre système d'exploitation :*  *(a insérer dans constructeur de la fenêtre)*  try {  //On force à utiliser le « look and feel » du système  UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());  //Ici on force tous les composants de notre fenêtre (this) à se redessiner avec le « look and feel » du système  SwingUtilities.updateComponentTreeUI(this);  } catch (InstantiationException e) {  } catch (ClassNotFoundException e) {  } catch (UnsupportedLookAndFeelException e) {  } catch (IllegalAccessException e) {  }  this.setVisible(true); | | | | | |  | |
|  |  |  | | |  | | | | |  | | |
| Interfaces Tableaux |  |  | | |  | | | | |  | | |
| JTable | défini un tableau pouvant être affiché à partir des données d'un tableau à double entrée d'objets  ATTENTION : tjs insérer le tableau dans un JScrollPane avant de le placer dans la fenêtre, sinon il n'affichera pas le titre des colonnes ! | Object[ ][ } data = {  {objetA, "String A", 1},  {objetB, "String B", 2}  };  String[ ] titles = {"Colonne1", "Colonne2"};  tableau = new JTable(data, titles);  maFenetre.getContentPane.add(new JScrollPane(tableau)); | | | *Constructeur à partir d'un JTableModel :*  modelT = new ModelTPerso(data, titles);  tableau = new JTable(modelT);  *(voir classe abstraite de JTableModel plus bas)* | | | | | ATTENTION : l'objet JTable n'accepte que des données sous forme de tableau d'Object !  Rq : les titres des colonnes peuvent être sous forme de tableau d'Object ou de tableau de String. | | |
| .getValueAt( , ) | retourne le contenu de la cellule de la ligne et colonne indiquée  Param : ligne, colonne | tableau.getValueAt(0, 1); => "String A" | | |  | | | | |  | | |
| .getRowCount()  .getColumnCount() | retourne le nombre de ligne/colonne contenu dans le tableau | tableau.getRowCount();  tableau.getColumnCount(); | | |  | | | | |  | | |
| .setRowHeight( , ) | défini la hauteur (en pixel) de la cellule d'index donné | tableau.setRowHeight(2, 40); | | |  | | | | |  | | |
| TableColumn | objet regroupant les informations liées à chaque colonne du tableau | TableColumn tCol = tableau.getColumnModel().getColumn(2); | | |  | | | | |  | | |
| .setPreferredWidth( ) | défini la largeur (en px) de la colonne | tCol.setPreferredWidth(70); | | |  | | | | |  | | |
| Mise en forme |  |  | | |  | | | | |  | | |
| JTableModel | objet regroupant toutes les informations liées au tableau. Il fait le lien entre les données et le tableau. |  | | |  | | | | |  | | |
| AbstractTableModel | pour définir/redéfinir/contrôler le contenu des cellules du tableau en fonction de leur type, on crée une classe (interne ou non) héritant de cette classe abstraite   * affiche booléen en case cochée * permet l'édition des données des cellules | ModelTPerso modelT = new ModelTPerso(data, titles);  tableau = new JTable(modelT);  - - - - - - - - - - - -  class ModelTPerso extends AbstractTableModel {  private Object[][] data;  private String[] title;    //Constructeur  public ModelTPerso(Object[][] data, String[] title) {  this.data = data;  this.title = title;  }    public int getColumnCount() {  return this.title.length;  }  public int getRowCount() {  return this.data.length;  }  public Object getValueAt(int row, int col) {  return this.data[row][col];  }  public String getColumnName(int col) {  return this.title[col];  }  instruction(s);  } | | | *méthodes facultatives pouvant être utiles de rajouter dans la classe précédente :*  //transforme les booléens en case à cocher  public Class getColumnClass(int col) {  //le type de contenu ne dépends que de la colonne (il est id pour toutes les lignes)  return this.data[0][col].getClass();  }    //retourne vrai si la cellule est éditable (cad tjs sauf pour un bouton)  public boolean isCellEditable(int row, int col) {  if(getValueAt(0, col) instanceof JButton)  return false;  return true;  }  //interdit la modification du contenu même si action dessus (important pour les boutons !)  public void setValueAt(Object value, int row, int col) {  //on interdit la modification sur certaines colonnes  if(!this.getColumnName(col).equals("Bouton") && !this.getColumnName(col).equals("Supression"))  this.data[row][col] = value;  } | | | | |  | | |
| DefaultTableCellRender | objet gérant l'affichage (MeP) du tableau, du contenu des cellules  (qd on rajoute getColumnClass() à la classe abstraite ci-dessus, c'est lui qui utilise cette info pour transformer les booléens en case à cocher)   * affiche les JButton comme bouton * affiche les JComboBox comme liste déroulantes | *Classe héritant de TableCellRenderer et permettant d'afficher automatiquement les JButton et ComboBox dans les cellules concernées :*  tableau.setDefaultRenderer(JComponent.class, new TableComponent());  - - - - - - - - - - -  class TableComponent extends DefaultTableCellRenderer {  public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column) {  //si la valeur de la cellule est un JButton ou JComboBox, on transtype cette valeur  if (value instanceof JButton)  return (JButton) value;  else if(value instanceof JComboBox)  return (JComboBox) value;  else  return this;  }  } | | | | | | | *Idem avec classes internes :*  //MaJ du tableau en remplaçant le contenu de la colonne choisie avec les boutons  tableau.getColumn("TitreColB").setCellRenderer(new ButtonRenderer());  //idem pour les ComboBoxde  tableau.getColumn("TitreColC").setCellRenderer(new ComboRenderer());  - - - - - - - - - - - - -  class ButtonRenderer extends JButton implements TableCellRenderer {  public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean isFocus, int row, int col) {  //on écrit dans le bouton ce que contient la cellule  this.setText((value != null) ? value.toString() : "");  //on retourne le bouton  return this;  }  }    class ComboRenderer extends JComboBox implements TableCellRenderer {  public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean isFocus, int row, int col) {  //on remplie la liste  this.addItem("Choix1");  this.addItem("Choix2");  //on retourne la ComboBox  return this;  }  } | | | |
| Action sur les éléments du tableau |  |  | | |  | | | | |  | | |
| DefaultCellEditor | objet permettant de suivre les actions sur le tableau   * action sur les boutons * action sur les liste déroulantes   ATTENTION : DefaultCellEditor ne gère que la partie action de l'utilisateur. Pour afficher les éléments sous le bon format il faut au préalable les avoir mis en forma grâce aux méthodes vues ci-dessus.  Rq: le constructeur de DefaultCellEditor peut prendre en paramètre   * JComboBox * ou JTextField   mais pas un JButton | *Pour des listes déroulantes :*  private String[ ] comboData = {"Choix 1", "Choix 2"};  JcomboBox combo = new JComboBox'comboData);  tableau.getColumn("TitreColC").setCellEditor(new DefaultCellEditor(combo)); | | | *Le constructeur ne prenant pas en charge les* JButton*, on doit nécessairement pour eux passer par une classe externe héritant de cette classe :*  tableau.getColumn("TitreColB").setCellEditor(new ButtonEditor());  - - - - - - - - - - - - -  public class ButtonEditor extends DefaultCellEditor {  protected JButton button;  private ButtonListener bListener = new ButtonListener();  public ButtonEditor() {  super(new JCheckBox()); //Contructeur avec une CheckBox par défaut  button = new JButton(); //on crée à nouveau un bouton  button.setOpaque(true);  button.addActionListener(bListener); //on lui attribue un listener  }    public Component getTableCellEditorComponent(JTable table, Object value, boolean isSelected, int row, int column) {  bListener.setRow(row); //donne les infos au Listener  bListener.setColumn(column);  bListener.setTable(table);    //on affecte le libélé au bouton  button.setText((value == null) ? "" : value.toString());  return button;  }    class ButtonListener implements ActionListener {  private int column, row;  private JTable table;  private JButton button;  public void setColumn(int col) {this.column = col;}  public void setRow(int row) {this.row = row;}  public void setTable(JTable table) {this.table = table;}  public JButton getButton() {return this.button;}    public void actionPerformed(ActionEvent e) {  System.out.println("Action sur le bouton : " + ((JButton)e.getSource()).getText());  //on affecte un nouveau libélé à une autre cellule de la ligne  table.setValueAt("New value " + (++nb), this.row, (this.column - 1));  //si affichage des boutons controlé par un Modele personnalisé (classe héritée de AbstractTableModel), contrôle l'affichage de ces derniers en faisant le lien avec ce modele  ((AbstractTableModel)table.getModel()).fireTableCellUpdated(this.row, this.column - 1);  this.button = ((JButton)e.getSource());  }  }  } | | | | | | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |