|  |  |
| --- | --- |
| |  | | --- | |  | |
| C34-VM Programmation Orientée Objet 1  **MODULE 04**  **Tableaux – Comparaisons - Dates**  Automne 2023 |

Table des matières

[1. Notes de cours : comparaisons d'objets 2](#_Toc1852423329)

[1.1 Égalité entre des valeurs de types prédéfinis : 3](#_Toc987217806)

[1.2 Notion de reference 3](#_Toc361676304)

[1.3 Égalité entre 2 objets ( autres que String ) 5](#_Toc296476816)

[1.4 Égalité entre 2 objets ( Strings ) 7](#_Toc429773620)

[L04A\_Comparaison d’objets 8](#_Toc288773720)

[2. Les tableaux (Arrays) 8](#_Toc34287503)

[2.1 Création / initialisation d'un tableau ( 3 façons ) : 9](#_Toc885728148)

[2.2 Tableaux d’Objets 9](#_Toc903659772)

[2.3 Tableaux à 2 dimensions 10](#_Toc679601286)

[2.4 La valeur null 11](#_Toc191455191)

[2.5 Déclaration, définition et initialisations d’objets 12](#_Toc1958341992)

[2.6 Notes additionnelles sur les tableaux – lettre ouverte des développeurs 15](#_Toc780248286)

[L04B\_Les tableaux 17](#_Toc1792780852)

[3. Traitement de dates et heure 18](#_Toc1530615391)

[3.1 Créer une variable qui contient une date/heure 19](#_Toc1380739678)

[3.2 Le cas spécial pour “maintenant” 19](#_Toc515588243)

[3.3 Afficher/imprimer une date 19](#_Toc1023739032)

[3.4 Ajouter ou retirer du temps 20](#_Toc1447033566)

[3.5 Extraire un des composants d’un Date/Heure 20](#_Toc1692462406)

[3.6 Méthodes pour LocalDate et LocalTime 20](#_Toc1263174511)

[3.7 ZoneDateTime 21](#_Toc1350573508)

[3.8 Les intervalles de temps 21](#_Toc1218986783)

[3.9 Classe Duration 22](#_Toc180479928)

[3.10 Formatage de dates/heures 23](#_Toc1351062987)

[L04C\_Dates et heures 26](#_Toc784511558)

# Notes de cours : comparaisons d'objets

## Égalité entre des valeurs de types prédéfinis :

C'est simple, on emploie comme à l'habitude l'opérateur ==.

Ex. : int a = 9;

int b = 9;

if ( a == b ) à return true

## Notion de reference

La syntaxe **new Produit()** créé un objet de la classe Produit.

Cet objet existe en mémoire vive à une adresse mémoire donnée.

Pour accéder à l’objet il faut connaitre cette adresse mémoire.

La syntaxe **Message objMessage = new Message()** créé :

* Un objet de la classe **Message**, stocké en RAM à une adresse mémoire donnée
* Une variable **objMessage** qui contient l’adresse mémoire pour accéder à l’objet.

La variable est une **référence** à l’objet.

|  |
| --- |
|  |

Considérez l’exemple suivant :

|  |
| --- |
| public class Message {  public String texteMsg = "Hello";   public void afficherMsg(){  System.*out*.println(texteMsg);  } } |

|  |
| --- |
| public class ProgPrincipal {  public static void main(String[] args) {  new Message().afficherMsg();  } } |

Dans la méthode main, on ne créé pas de variable, seulement un objet dont on utilise la méthode directement.

Mais on ne peut plus ré-utiliser cet objet, on n’a pas mémorisé son adresse en mémoire.

Si on créé plutôt une référence à l’objet dès sa création, on peut manipuler et réutiliser l’objet, ce qui n’est pas possible sans référence :

|  |
| --- |
| public class ProgPrincipal {  public static void main(String[] args) {  Message objMsg = new Message();  objMsg.afficherMsg();  objMsg.texteMsg = "Bye Bye";  objMsg.afficherMsg();  } } |

Note: naturellement, on préfèrerait que tous les champs soient **private** et soient accédés par des méthodes publiques 😊

## Égalité entre 2 objets ( autres que String )

* Il faut d'abord distinguer si on parle de deux objets ou de deux références à un objet; des références étant des adresses indiquant où se trouvent les variables et les méthodes d'un objet donné.

Ex.: Point p1, p2;

p1 = new Point ( 100, 99 );

p2 = p1;

* La classe point est une classe prédéfinie de Java qui contient une coordonnée en **x** une coordonnée en **y** et quelques méthodes pour les manipuler.
* Dans cet exemple, un seul objet est créé mais il y a deux références à cet objet ( p1 et p2 ) Ainsi, toute modification faite sur cet objet à partir d'une ou l'autre des références aura un impact sur l'objet.

|  |
| --- |
|  |

Ex.: p1.setX( 200 );

p1.getX() à 200

p2.getX() à 200 également puisqu'on parle d'un seul objet

* l'opérateur == retournera true uniquement si on compare des références à un même objet

Ex.: if(p1==p2) à return true car ce sont deux références à un même objet

Ex.: Point p3 = new Point ( 15,15 );

Point p4 = new Point ( 15,15 );

if ( p3 == p4 ) à return false ( ce ne sont pas deux références à un même objet. )

Pour comparer deux objets entre eux, il faut vérifier si toutes les variables d'instance sont égales. La méthode equals est définie dans certaines classes pour réaliser cette tâche ( classes String, Color, Font, Point, Calendar, Hashtable, Vector, etc. )

Ex.: Point p5 = new Point ( 20,20 );

Point p6 = new Point ( 20,20 );

if ( p5.equals ( p6 ) ) à return true . ( On a employé equals car on avait affaire à deux objets différents )

Si le type des objets à comparer ne définit pas la méthode equals, on doit la redéfinir ( la coder ) nous-mêmes. Attention, l'appel de la méthode equals sur des objets où elle n'est pas redéfinie fonctionnera mais il s'agira de la méthode equals de la classe Object ( héritage ). L'emploi de cette méthode est équivalent à == dans ces cas.

Ex.: Roi r1 = new Roi ("k", "noir" );

Roi r2 = new Roi ("k", "noir" );

if ( r1.equals(r2) ) à return false car la méthode equals n'a pas été redéfinie dans la classe Roi ou Piece, on est donc en train d'utiliser la méthode equals de la classe Object.

## Égalité entre 2 objets ( Strings )

* Le cas des Strings est particulier. Comme on l'a vu, ce type est mi-prédéfini, mi-objet. Le choix d'utiliser == ou equals pour les comparer réside dans la définition des Strings.

Ex. : String s = "bonjour"; // chaîne littérale

String t = new String ("bonjour" ); // forme objet

* Dans le cas des chaînes littérales, on peut utiliser l'opérateur == en autant qu'on veuille comparer deux références à des chaînes littérales. On peut utiliser == car les chaînes littérales sont automatiquement associées à un même objet à l'interne lorsque leurs valeurs sont égales ( procédé de l'interning ).

Ex. : String a = "allo";

String b = "allo";

if ( a == b ) à return true // 2 chaînes littérales donc associées au même objet

String a = "allo";

if ( a == "allo" ) à return true // idem

String a = "allo";

String b = champTexte.getText(); // contenant "allo"

if ( a==b ) à return false car b n'est pas une chaîne littérale

String a = "allo";

String b = new String ("allo");

if (a==b) à return false car b n'est pas une chaîne littérale

* pour comparer deux formes objets ou une forme objet avec une forme littérale, on doit utiliser la méthode equals.

Ex.: String a = "allo";

String b = new String ("allo");

if ( a.equals(b) ) à return true

* finalement, on peut faire de l'interning sur des formes objets String en appelant la méthode intern()

Ex.: String a = "allo";

String b = new String ("allo");

b = b.intern();

if ( a.equals (b) ) à return true

if ( a == b ) à return true ( elle est devenue littérale )

### L04A\_Comparaison d’objets

Le laboratoire L04A permet de pratiquer la comparaison d’objets.

# Les tableaux (Arrays)

## Création / initialisation d'un tableau ( 3 façons ) :

Un tableau doit être déclaré ET initialisé pour qu’on puisse y assigner des valeurs.

A) int[] tableau = new int[12];

Déclaration et initialisation en 1 ligne.

Réserve de l’espace-mémoire pour 12 valeurs entières

Lorsqu’on initialise un tableau sans donner de valeurs aux éléments, des valeurs par défaut sont automatiquement assignées. Ces valeurs dépendent du type des éléments :

|  |  |
| --- | --- |
| **Type** | **Valeur par défaut** |
| boolean | false |
| int | 0 |
| double | 0.0 |
| String | null |
| Objet ou type défini par l’utilisateur | null |
|  |  |

Dans l’exemple :

Les éléments sont initialisés à : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Le tableau ressemblerait à : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

B) int[] tableau; // présentation, déclaration d’une variable d’instance

tableau = new int[12]; // création du tableau, dans un constructeur par exemple

Déclaration et initialisation en 2 lignes.

L’initialisation d’un tableau à une variable déjà déclarée doit utiliser le mot **new**.

C) Si on connaît déjà ce qu’on veut placer dans le tableau, on peut le créer en extension :

int [] tableau = { 3,4,5,6,2,0,0,9 };

Déclaration, initialisation (tableau de 8 éléments dans l’exemple) et assignation de valeurs.

\*\*\* un tableau a une grandeur fixe, on ne peut pas dépasser la capacité indiquée au départ

## Tableaux d’Objets

IMPORTANT : On doit créer le tableau ET les objets faisant partie du tableau

C’est la référence aux objets qui est stockée dans le tableau.

Ex. : LocationFilm[] tab = new LocationFilm[3]; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

tab[0] = \_\_\_\_\_\_\_\_\_\_\_\_\_\_

tab[0] = new LocationFilm ( "Forrest Gump" );

tab[1] = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

tab[2] = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ex.2 :

public class Province

{

private String[] ensemble;

public Province ()

{

ensemble[0] = "Montréal";

ensemble[1] = "Québec";

ensemble[2] = "Laval";

}

}

Quel est le problème dans cette classe relatif à l'initialisation du tableau ? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Tableaux à 2 dimensions

int[][] tableau = new int[2][4]; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Exercices :**

int tableau1[] = new int[13];

boolean tableau2[] = new boolean[4];

ex.: tableau1[2] = \_\_\_\_\_\_\_\_\_\_\_\_\_

tableau2[0] = \_\_\_\_\_\_\_\_\_\_\_\_\_

**création et initialisation d'un tableau :**

int tableau3 [ ] = { 5, 4, 4, 78 }

int tableau4 [ ][ ] = { {2,3}, {3,4}, {3,9} }

ex.: tableau3 [1] = \_\_\_\_\_\_\_\_\_\_\_\_\_

tableau3.length = \_\_\_\_\_\_\_\_\_\_

tableau4[2][1] = \_\_\_\_\_\_\_\_\_\_\_

tableau4.length = \_\_\_\_\_\_\_\_\_\_

tableau4[1].length = \_\_\_\_\_\_\_\_

**tableau d'objets :**

Paiement tableau5 [ ] = new Paiement [3]

tableau5 [0] = new Paiement (1,4,0); // nbre de 2$, de 1$, de 0,25$

tableau5 [1] = new Paiement (0,3,0);

tableau5 [2] = new Paiement (1,0,0);

ex.: tableau5[2].getNbreDeux () = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Paiement[0].getNbreUn () = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

tableau5.length = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## La valeur null

En java le mot clé **null** indique l’absence de valeur.

On ne peut pas assigner **null** à une variable de **type primitif**. (~~int nombre = null~~).

On peut assigner **null** à une **référence ou un pointeur**, sa signification est que cette variable n’indique aucune adresse mémoire.

Parce qu’une variable d’objet ou de tableau est une référence, java autorise l’assignation de null à ces variables.

int tableau[] = null; // autorisé

Produit objProduit = null; // autorisé

Il faut faire attention en manipulant des variables nulles, essayer d’en traiter la valeur génère une exception.

Produit objProduit = null; // autorisé

String nom = objProduit.getNom(); // Exception!!!

Mais on peut gérer ceci en vérifiant d’abord si une référence est nulle:

if (objProduit == null){

System.out.println(“Pas de produit à traiter”);

else

System.out.println(objProduit.getNom());

On ne peut passer une référence null en paramètre à une méthode.

Parce que la valeur null peut causer des exceptions dans plusieurs situations, on l’utilise seulement lorsque d’autres moyens ne sont pas disponibles.

## Déclaration, définition et initialisations d’objets

La **déclaration** consiste à indiquer qu’une variable existe (je déclare qu’il y aura une variable X de type Y).

Ça signifie souvent, le type et le nom sans valeur (pas de =)

* Déclarer une variable de type primitif:

int nombre;

* Déclarer un objet:

NomDeClasse objet;

* Déclarer un tableau:

NomDeClasse tableau[]; // Un tableau est un objet

On peut, par la suite, définir ces variables.

* Définir une variable de type primitif:

nombre = 10;

* Définir un objet:

objet = new NomDeClasse();

* Définir un tableau:

tableau = new NomDeClasse[TAILLE\_TABLEAU]; // Dans la définition, la taille du tableau est obligatoire

**Initialiser** une variable consiste à la définir pour la première fois, lui donner une valeur de départ.

Lors de la modélisation d’une classe, la convention est de déclarer les champs sans les définir, puis de les définir dans le constructeur ou dans une méthode.

C’est le rôle du constructeur de créer l’objet et toutes les initialisations de valeurs se retrouvent alors à un seul endroit.

**Exemple:**

|  |
| --- |
| public UneClasse{  Public static final UNE\_CONSTANTE = 4; // Une constante est déclarée et initialisée en même temps  int unNombre; // Primitive déclarée et non définie  UneAutreClasse unObjet; // Objet déclaré et non défini  EncoreUneClasse[] tableauObjets // Tableau déclaré et non défini - ici un tableau d’objets    // Le constructeur  Public UneClasse(int unNombre, UneAutreClasse unObjet){  this.unNombre = unNombre; // Défini par paramètre  this.unObjet = unObjet; // Défini par paramètre, pas besoin de new, le new a été fait ailleurs  tableauObjets = new EncoreUneClasse[UNE\_CONSTANTE]; // Le tableau est défini, pas ses éléments  for(int i = 0; i < UNE\_CONSTANTE; i++){  tableauObjets[i] = new EncoreUneClasse();  }  // Maintenant les éléments du tableau sont définis  }  } |

## Notes additionnelles sur les tableaux – lettre ouverte des développeurs

*Le courrier Java*

**SUJET DE LA SEMAINE : LES FAMEUX TABLEAUX**

Cher Java,

J’ai entendu de belles choses à propos des tableaux d’objets mais je suis timide, je n’ose pas les approcher de peur de faire une erreur. Que faire ??

Une admiratrice

Chère admiratrice,

Un tableau d’objets se manipule de la même manière qu’un tableau de int ou de double ( types prédéfinis)  à la différence qu’il faut créer les objets faisant partie du tableau.

* Un tableau de 10 valeurs double :

double[] tab = new double[10]; // tous les éléments valent 0

* Un tableau de 5 objets GregorianCalendar :

GregorianCalendar tab2 = new GregorianCalendar[5];

//tous les éléments valent null, aucun objet n’a été créé à l’intérieur du tableau

Pour initialiser les éléments du tableau, on a qu’à leur assigner une valeur dans le cas d’un tableau de type prédéfini ou un objet dans le cas d’un tableau d’objets

tab[3] = 87.905;

tab[6] = 9;

tab2[0] = new GregorianCalendar(); // représentant aujourd’hui

tab2[1] = new GregorianCalendar(1995,2,20); //représentant le 20 mars 1995

Cher Java,

Je ne sais pas comment m'y prendre pour passer des tableaux comme paramètre de méthodes ou de constructeurs. Rien que je ne fasse semble plaire au compilateur.

Un programmeur frustré

Cher programmeur frustré,

Les tableaux sont des objets jusqu'à un certain point. Entre autres, on peut les initialiser directement sans avoir à passer élément par élément. Par exemple,

public class Vendeur{

private int [] numContrats;

public Vendeur ( int [] numContrats )

{

this.numContrats = numContrats;

}

...

}

/////////////////////////////////////////////////////////////////////////////

Cher Java,

Mon tableau ne veut rien savoir de mes initialisations. À chaque fois que je veux initialiser un élément de mon cher tableau, rien à faire, il n'en fait qu'à sa tête. Que puis-je faire pour lui donner des valeurs ???

Pablo Tablo

Cher Pablo,

Il faut toujours s'assurer que le tableau est créé avant d'avoir accès à ses éléments sans générer d'erreurs ou d'exception lors de l'exécution. Il faut distinguer 2 cas, l'un ou le tableau est passé en paramètre et l'autre où il ne l'est pas.

-Cas vécu 1-

public class Vendeur{

private int [] numContrats;

public Vendeur ( int [] nbContrats )

{

this.numContrats = nbContrats;

if ( numContrats[0] == 567665 )

...

}

...

}

Dans ce premier cas vécu, il n'y a aucun problème à accéder aux éléments du tableau, car celui-ci vient d'un paramètre, on suppose donc que le tableau a déjà été créé ailleurs. Lorsqu'on appellera ce constructeur ( dans une interface graphique par exemple ) , on prendra soin de construire le tableau passé en paramètre avec new . Cet état est vrai pour tous les types d'objets ( Vector, classes, etc. )

-Cas vécu 2-

public class Vendeur

{

private int [] numContrats;

public Vendeur ()

{

if ( numContrats[0] == 567665 )

...

}

...

}

Cette fois-ci, ce cas ne fonctionne pas. Le tableau, attribut de la classe Vendeur, n'ayant pas été créé avec new ou ne provenant pas d'un paramètre extérieur à la classe, on ne peut donc pas accéder à un élément du tableau pour l'initialiser ou pour le comparer. Pour ce faire, il faudrait d'abord créer le tableau avec new :

public class Vendeur

{

private int [] numContrats;

public Vendeur (){

numContrats = new int [10]; // grandeur du tableau

numContrats[0] = 2141;

numContrats[1] = 91847;

...

if ( numContrats[0] == 567665 )

...

}

...

}

**Ou**

Si le tableau à utiliser n'est pas une variable d'instance mais une variable locale à une méthode, on peut l'initialiser en extension sans l'usage du mot-clé new

public class Vendeur

{

…

public void contratsPresents ()

{

int[] contratsActuels = {2141, 91847, 28437,...}; // remplace le new;

if ( numContrats[0] == 567665 )

...

}

...}

### L04B\_Les tableaux

Le laboratoire L04B permet de pratiquer l’utilisation de tableaux d’objets.

# Traitement de dates et heure

Le traitement de dates et d’heures est fréquent en développement d’applications.

Java offre (depuis la version 8) la suite d’APIs **Date** and **Time** pour gérer les dates et heures. Les classes sont disponibles dans le package **java.time**.

Plusieurs classes sont disponibles avec leurs méthodes, on en présente quelques unes dans ce cours.

On travaille avec 6 des classes principales:

|  |  |
| --- | --- |
| **Classe** | **Description** |
| **LocalDate** | Date sans l’heure - fuseau horaire local (local zone) |
| **LocalTime** | Heure sans la date - fuseau horaire local (local zone) |
| **LocalDateTime** | Date et heure - fuseau horaire local (local zone) |
| **ZoneDateTime** | Date et heure – manipulation d’autres fuseaux horaires (zones) |
| **Period** | Intervalle de temps basé sur de composants de dates (année, mois, jour) |
| **Duration** | Intervalle de temps basé sur des unités d’heure (heures, minutes, secondes, nanosecondes...) |

## Créer une variable qui contient une date/heure

On commence avec les méthodes du fuseau horaire local (Local...)

|  |  |
| --- | --- |
| LocalDateTime uneDate =  LocalDateTime.**of**(2006, 03, 22, 21, 15, 36); | Le 22 mars 2006, à 9h 15 minutes et 36 secondes PM. |
| LocalDateTime uneDate =  LocalDateTime.**parse**("2006-03-22T21:15:36"); | **parse** peut être utilisé au lieu de **of** si la date est disponible en format texte (chaine de caractère), et si cette chaine respecte le format date/heure demandé. |

La syntaxe est semblable pour **LocalDate** et **LocalTime**, en utilisant seulement les paramètres pertinents.

## Le cas spécial pour “maintenant”

|  |  |
| --- | --- |
| LocalDateTime uneDate = LocalDateTime.now() | La méhode **.now** créé une variable avec la date et l’heure actuelle. |

## Afficher/imprimer une date

On voit un peu plus loin des méthodes avancées pour formatter l’affichage d’une date/heure. En attendant, pour permettre de voir les résultats des méthodes, on peut afficher directement la variable **DateTime** avec **println**, elle sera convertie en chaine de caractère, formatée en **DateTime** standard.

|  |
| --- |
| System.out.println(uneDate); |

## Ajouter ou retirer du temps

|  |  |
| --- | --- |
| LocalDateTime uneDate = LocalDateTime.now();  ...println(uneDate.plusDays(3)); | Afficher maintenant + 3 jours.  **uneDate** n’a pas changé, un LocalDateTime est immuable |
| LocalDateTime uneDate = LocalDateTime.now();  ...println(uneDate.minusDays(3)); | Afficher maintenant + 3 jours. |
| Aussi disponibles:  .plusNanos()  .plusSeconds()  .plusMinutes()  .plusHours()  .plusDays()  .plusWeeks()  .plusMonths()  .plusYears()  … et les toutes les méthodes ci-dessus en version minus … | Ces méthodes gèrent automatiquement le parcours sur plusieurs mois, années, gère les années bisextiles etc. |

## Extraire un des composants d’un Date/Heure

Des méthodes d’accès (getters) sont disponibles pour extraire un des composants d’un Date/Heure.

En partant de LocalDateTime unDate = quelquechose...

|  |  |
| --- | --- |
|  |  |
| int jourDuMois = uneDate.getDayOfMonth() | Retourne un entier qui est le jour du mois de la variable LocalDateTime. |
| D’autres méthodes sont disponibles pour tous les composants de dates incluant même le jour de la semaine. | Essayez uneDate.getMonth(), de quel format est la sortie?  Utilisez un des outils d’aide à votre disposition pour les explorer (l’autocomplétion d’IntelliJ est un bon point de départ...) |

## Méthodes pour LocalDate et LocalTime

La majorité des méthodes vues son disponibles avec **LocalDate** et **LocalTime**.

Par exemple, les énoncés suivants sont valides:

LocalDate uneDate = LocalDate.now().plusDays(100);

LocalDate uneDate = LocalDate.now().year();

LocalTime uneHeure = LocalTime.now().minusMinutes(30);

Naturellement, il faut tenir compte du contexte, une méthode comme .plusYear() ne sera pas disponible avec LocalTime.

## ZoneDateTime

Ajoute la notion de fuseau horaire à LocalDateTime.

On utilise ZoneDateTime lorsque la notion de fuseau horaire est importante dans une application.

Par exemple si vous vouliez afficher plusieurs horloges montrant l’heure de différents pays, on pour répondre à la question “il est quelle heure à Tokyo présentement?”, calculer l’heure d’arrivée d’un vol d’avion international, etc.

Exemple de déclaration:

ZonedDateTime now = ZonedDateTime.now( ZoneId.of("GMT+05:30") );

Il y a plusieurs façons de traiter les ZoneId.

La classe ZoneDateTime a plusieurs méthodes particulières.

On n’explore pas cette classe en détails dans ce cours.

## Les intervalles de temps

Un intervalle de temps n’indique pas une heure particulière mais une quantité de temps. Les classes disponibles pour gérer des intervalles de temps sont:

**Period** : permet de gérer un intervalle de temps en années/mois/semaines/jours

**Duration**: permet de gérer un intervalle de temps en heures/minutes/secondes/nanosecondes

Plusieurs des méthodes déjà vues sont disponibles pour ces méthodes, il faut garder en tête que c’est un intervalle et non une date.

|  |  |
| --- | --- |
| Period unePeriode =  LocalDateTime.**of**(4, 1, 2); | Une période de 4 ans, 1 mois et 2 jours.  **PAS** le 2 janvier de l’an 0004! |
| unePeriode.getDays(); | Le composant **jour** de unePeriode. Dans l’exemple, c’est **2**.  Il y a des méthodes .getYears(), .getMonths etc. |
| Period unePeriode = Period.ofDays(128); | Créé une période basé sur un nombre de jours  Il y a des méthodes .ofYears(), .ofMonths etc. |
| .plusDays(x);  .plusMonths(x);  .plusYears(x); | Ces méthodes ajoutent à l’intervalle une quantité x correspondante. Si x est négatif, c’est l'équivalent de faire une soustraction. |

Un moyen d’ajouter/soustraire du temps à une date en utilisant les périodes (il y a plusieurs autres moyens d’atteindre cet objectif).

1. Méthode .plus de LocalDateTime

|  |
| --- |
| LocalDateTime maintenant = LocalDateTime.now();  Period ajouterTemps = Period.of(1, 1, 1); // Periode de 1 and, 1 mois et 1 jour  System.out.println("Dans 1 an 1 mois et un jour nous serons le: " + maintenant.plus(ajouterTemps)); |

## Classe Duration

La classe duration est aussi une classe d’intervalle de temps. Elle poursuit des objectifs similaires à la classe Period, mais avec une plus petite granularité de temps: heures, minutes, secondes, nanosecondes.

Si vous désirez coder une application de type chronomètre, par exemple, cette classe serait plus appropriée.

Une des méthodes utiles de cette classe est **.between**, qui créé un intervalle de temps entre 2 date/heures.

Voici un exemple:

LocalDateTime maintenant = LocalDateTime.now();

LocalDateTime dans3Jours = maintenant.plusDays(3);

Duration intervalle = Duration.between(maintenant, dans3Jours);

System.out.println("Temps écoulé en jours: " + intervalle.toDays());

System.out.println("Temps écoulé en heures: " + intervalle.toHours());

System.out.println("Temps écoulé en minutes: " + intervalle.toMinutes());

On vous laisse explorer plus en détails les méthodes de cette classe par vous-même.

## Formatage de dates/heures

On représente (affiche) le temps sous de multiples formes pour toutes sortes de raisons:

Par exemple, le 25 décembre 2001 pourrait être affiché de toutes les façons suivantes, et ce n’est qu’un petit échantillon:

12/25/2001

25/12/2001

2001/25/12

2001-25-12

2001 déc 25

25 décembre 2001

décembre 2001, le 25

* Année: 2001
* Mois: 12
* Jour: 25

Etc.

Java offre la classe **DateTimeFormatter** pour gérer ceci.

**DateTimeFormatter** est disponible à partir de la librairie, **java.text**

Le champ principal d’un objet DateTimeFormatter est une chaine de caractère qui représente le format d’affichage lui-même (un modèle).

Par exemple:

DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd");

Cette instruction indique que l’affichage d’une date avec cette variable sera affiché de la façon suivante:

1. L'année sur 4 chiffres (yyyy)
2. Suivi d’un tiret (-)
3. Suivi du mois sur 2 chiffres (MM)
4. Suivi d’un tiret (-)
5. Suivi du jour sur 2 chiffres (dd)

On utilise ensuite la méthode .**format** de l’objet DateTimeFormatter pour créer une chaine de caractère dans le format spécifié, en lui passant l’objet de date/heure désiré.

Exemple:

LocalDate noel2001 = LocalDate.of(2001, 12, 25);  
DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd");  
System.out.println("Noel 2001 était le: " + dtf.format(noel2001));

Notez qu’on passe une date à l’objet dtf et non l’inverse.

Pour déterminer le format des composants de date/heure, il faut utiliser les symboles approuvés pour ce modèle.

En voici quelques-uns, vous pouvez consulter la doc officielle ici:

<https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html>

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbole** | **Explication** | **Présentation** | **Examples** |
| y | Année de calendrier | Année | 2004, 04 |
| Y | Année de la semaine | Année | 2004, 04 |
| M | Mois de l’année | Nombre/texte | 7, 07, Jul, July, J |
| w | Semaine de l’année | Nombre | 27 |
| W | Semaine du mois | Nombre | 3 |
| D | Jour de l’année | Nombre | 189 |
| d | Jour du mois | Nombre | 10 |
| H | Heure de la journée (0-23) | Nombre | 0 |
| m | Minute de l’heure | Nombre | 30 |
| s | Secondes de la minute | Nombre | 5 |

**Quelques exemples:**

Pour le code suivant:

String modele = "";

LocalDateTime uneDate = LocalDateTime.of(2016, 2, 14, 16, 1, 1);

DateTimeFormatter dtf = DateTimeFormatter.ofPattern(modele);

System.out.println(dtf.format(uneDate));

|  |  |
| --- | --- |
| **Pour une valeur de modele de:** | **L’affichage sera:** |
| "yyyy-M-dd" | 2016-2-14 |
| "yy-MM-dd" | 16-02-14 |
| "yyyy/MMM/dd" | 2016/févr./14 |
| "Date: yyyy-MMMM-dd" | Exception... Unknown pattern letter: t |
| "'Date: 'yyyy-MMMM-dd" | Certains caractères comme – et / sont acceptés, mais si on veut écrire du texte, il vaut encadrer avec des apostrophes. |
| "'Date: 'dd MMMM yyyy 'Heure: 'HH:mm" | Date: 14 février 2016 Heure: 16:01 |

### L04C\_Dates et heures

Le laboratoire L04C, permet de pratiquer la manipulation et l’affichage de dates et heures.