Annexe 8 - Notes de cours : comparaisons d'objets

1. 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 ) → return true
```

2. 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, 100 );
p2 = p1;
```

- 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() → 200
p2.getX() → 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)` → 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 ) → 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 ) )` → 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) )` → 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`.

3. Egalité 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 )` → return true // 2 chaînes littérales donc associées au même objet

```
String a = "allo";
```
`if ( a == "allo" )` → return true // idem

```
String a = "allo";
String b = champTexte.getText(); // contenant "allo"
if ( a==b ) → return false car b n'est pas une chaîne littérale

String a = "allo";
String b = new String ("allo");
if (a==b) → 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) ) → 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) ) → return true
if ( a == b ) → return true ( elle est devenue littérale )
```

Annexe 8B - Exercices sur les références à un objet

Répondez aux questions suivantes en supposant que tous les imports nécessaires ont été faits afin que les méthodes compilent bien.

```
1.   public static void main(String[] args) {
            String nom = "Lachance";
            String prenom = "Louise";
     }
```

A) Combien d'objets ont été créés dans cette méthode ? 2 car string est un objet, defini de façon literal

```
2.   public static void main(String[] args) {
       String dossier = "AE454";
       int nombre = 90;
       char lettre = 'a';
       char choix =  dossier.charAt(3);
     }
```

A) Combien d'objets ont été créés dans cette méthode? 1
B) Quelle est le contenu de la variable choix à la fin de la méthode ? '5'

```
3.   public static void main(String[] args) {
      CompteBancaire cb1 = new CompteBancaire("Marie", 50);
      CompteBancaire cb2 = cb1;
      System.out.println ( cb1 == cb2 );
     }
```

A) Qu'affichera l'appel de la méthode `println` ? true car on compare deux reference au meme objet

```
4.   public static void main(String[] args) {
       Point p1 = new Point ( 5, 6);
       Point p3 = new Point ( 6, 8);
       Point p4, p5;
       p4 = p1;

;
       System.out.println ( p3.getX());
```

```
        System.out.println ( p1.getY());
        System.out.println ( p4.getX());
    }
```

A) Combien d'objets ont été créés dans cette méthode ? 2

B) Qu'afficheront les 3 appels à `println` ?

6,6,impossible *null pointer exception*

```
5.   public static void main(String[] args) {
       CompteBancaire cb1 = new CompteBancaire("Marie", 50);
       CompteBancaire cb2 = new CompteBancaire("Marie", 50);
       System.out.println ( cb1 == cb2 );
       System.out.println ( cb1.equals(cb2));
     }
```

A) Lequel des 2 appels permettra de réellement comparer si les deux objets sont identiques ? Pourquoi ?

Le == faire compare des reference a un même objet a condition que la methode equals aie été codee dans la class compte bonquare , alors c'est equal

```
6.  public static void main(String[] args) {
       String a = "Raymond";
       String b = new String ("Raymond");
       System.out.println ( a == b );
       System.out.println ( a.equals(b));
     }
```

A) Qu'afficheront les deux appels de la méthode `println` ?

_____false ne sont pas deux form literal et true deux string identique_____

Annexe 9 - Notes de cours sur les tableaux

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

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

Réserve de l'espace-mémoire pour 12 valeurs entières

Les éléments sont initialisés à : 0

Le tableau ressemblerait à :{0,0,0,0,0,0,0,0,0,0,0,0} 12x0

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

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

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 };
```

*** 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

Ex. : `LocationFilm[] tab = new LocationFilm[3];on vient de créer un tableau avec un espace réserver pour trois objet locationfilm`

```
tab[0] =null

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

tab[1] = new LocationFilm (« »);

tab[2] = new LocationFilm (« »);
```

Ex.2 :

```
public class Province
{
private String[]ensemble;

public Province ()
        {
                Ensemble =  new String [3];
                ensemble[0] = "Montréal";
                ensemble[1] = "Québec";
                ensemble[2] = "Laval";
         }
}
```

Quel est le problème dans cette classe relatif à l'initialisation du tableau ? le tableau ensemble n'est pas creer

Tableaux à 2 dimensions

```
int[][] tableau = new int[2][4];   ligne / colonne tableau avec 2
lign et 4 colonne
```

Exercices :

Par defaut

```
int tableau1[] = new int[13];
boolean tableau2[] = new boolean[4];
```

ex.: `tableau1[2]` = 0
`tableau2[0]` = false

création et initialisation d'un tableau :

`int tableau3[ ]` = { 5, 4, 4, 78 }
`int tableau4[ ][ ]` = { {2,3}, {3,4}, {3,9} }

ex.: `tableau3[1]` = 4
`tableau3.length` = 4
`tableau4[2][1]` =9
`tableau4.length` = 3
`tableau4[1].length` = 2 car le un fait un mini tableau

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 ()` =1

`Paiement[0].getNbreUn ()` = ne fonctionne pas il fallait ecrirt tableau 5

`tableau5.length` = 3

Annexe 10 – Exercice sur les tableaux

1. Soit la modélisation d'un appareil servant à indiquer au placier d'un restaurant les tables qui sont occupées et celles qui sont libres. Le placier pourra prendre les informations d'un groupe de clients et les diriger ainsi à une table convenant à leurs exigences à l'aide d'une interface lui indiquant quelle table leur assigner.

tiré de Diner Dash, PlayFirst

A) Copiez le projet `Annexe10` de LÉA dans votre Y. Ouvrez-le dans IntelliJ. Vous y retrouvez une classe `Table` représentant une table du restaurant. Examinez-la attentivement. Elle contient 4 variables d'instance représentant l'état de l'objet `Table`, la variable occupe est initialisée à false dans le constructeur car lorsqu'on ouvre le restaurant toutes les tables sont vides bien évidemment.

B) Dans le même projet, créez une nouvelle classe `Restaurant` qui représentera le restaurant dans l'optique du logiciel d'assignation des tables aux clients.

2. La seule variable d'instance du restaurant doit être un tableau d'objet `Table`.

3. Codez un constructeur permettant d'initialiser la variable d'instance. Ce constructeur représentera l'ouverture du resto donc ne prendra aucun paramètre. Vous devrez donc initialiser le tableau d'objets `Table` avec les informations suivantes :

- 30 tables seront des tables pour 4 personnes avec banquette
- 30 tables seront des tables pour 2 personnes avec banquette
- 10 tables seront des tables pour 4 personnes sans banquette
- 5 tables seront des tables pour 2 personnes sans banquette

*** donnez comme numéro de table l'indice du tableau + 1 (les tables seront donc numérotées 1,2,3, …75.)

4. Codez une méthode (`nbTableOccupees`) permettant de retourner le nombre de tables du restaurant qui sont occupées à un moment de la journée.

5. Codez une méthode (`assignerTableDispo`) qui permettra d'assigner une table disponible lorsqu'un groupe de personnes se présente au restaurant

- quelles informations avez-vous besoin de savoir à propos du groupe de personnes donc des informations venant de l'extérieur du modèle `Restaurant` ?
- on supposera que les groupes ne dépasseront pas 4 personnes
- quelle variable d'instance de l'objet `Table` choisi avez-vous besoin de modifier ?

6. Codez une méthode (`verifierSiTableOccupe`) qui permettra, avec l'aide d'un numéro de table passé en paramètre, de vérifier si la table avec ce numéro est occupée ou non

7. À l'image de l'annexe 6B, copiez les deux fichiers représentant l'interface graphique **et copiez-les au même endroit que vos autres fichiers .java. Vous devrez changer le lien dans le fichier .form comme dans l'annexe 6B.**

Annexe 11 - Notes sur les classes Calendar, GregorianCalendar et SimpleDateFormat

Classes Calendar, GregorianCalendar

- encapsule en un objet les différents champs (jour, mois, année, heure…) relatifs à une date
- font partie du package `java.util`

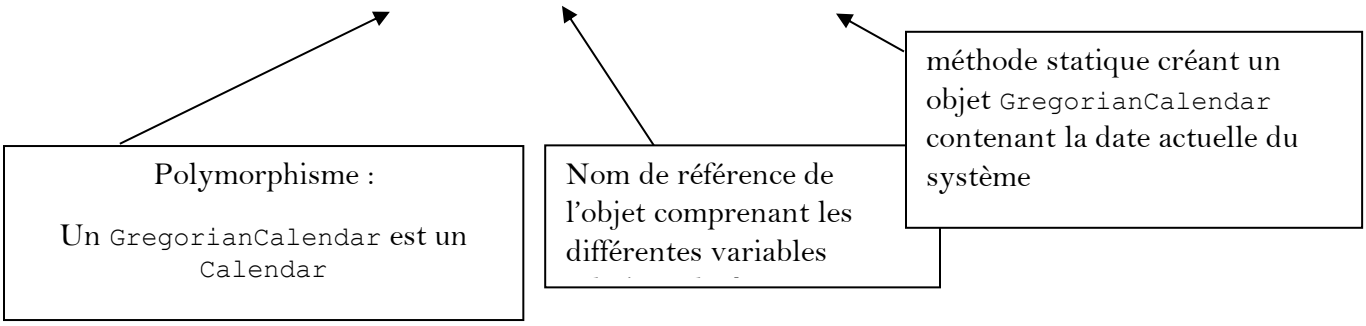
Classe Calendar

- il s'agit d'une classe abstraite, mettant en œuvre deux interfaces
- il n'est donc pas possible de créer d'objet `Calendar` à l'aide de new
- une sous-classe `GregorianCalendar` existe : on peut donc créer des objets de cette sous-classe :

```
GregorianCalendar gc = new GregorianCalendar();
```

- D'une autre façon, une méthode statique `getInstance` permet de créer un objet `GregorianCalendar` à partir de la classe abstraite `Calendar`, comme ceci :

```
Calendar c = Calendar.getInstance();
```



Classe GregorianCalendar

- Créer un objet contenant la date actuelle du système (similaire à `getInstance`) :

```
GregorianCalendar present = new GregorianCalendar();
```

- Créer un objet contenant une date quelconque :

```
GregorianCalendar pass = new GregorianCalendar (1990,1,6);
```

➔ représente le 6 février 1990 *** **valeur du mois commence à 0 !!!**

- Pour aller chercher un attribut / variable de l'objet : méthode get avec paramètre représentant la variable

```
pass.get(Calendar.DAY_OF_MONTH)  == 6
pass.get( Calendar.YEAR ) == 1990
pass.get  ( Calendar.DAY_OF_YEAR ) == 37  ( 37 ème jour de l'année  )
```

- Pour modifier un attribut de l'objet : methode set

`pass.set( Calendar.MONTH, 3 )` → change le mois de février à avril
`pass.set( 2000, 2, 13 )` → modifie année, mois, jour

- Pour augmenter ou diminuer la valeur d'un attribut, méthode add

`present.add( Calendar.DAY_OF_MONTH, 2 )` → on est rendus 2 jours plus tard

- Pour savoir si un objet `Calendar` est chronologiquement avant un autre : méthode `before`

`present.before(pass)` → retourne false

Classe SimpleDateFormat

- utile lorsqu'on veut afficher des dates contenues dans un objet GregorianCalendar
- fonctionne en créant un modèle qui sera passé dans l'appel du constructeur du SimpleDateFormat (comme avec le DecimalFormat)
- Les caractères principaux utilisés dans le modèle sont :

Caractères	Correspondant à…	Exemple
yyyy	Année	1989
MMM (3 ou plus)	Mois dans l'année (en lettres)	Juillet
MM (2 caractères)	Mois dans l'année (en chiffres)	07
dd	Jour dans le mois	22
HH	Heure dans le jour (00 – 23)	15
mm	Minutes	56
ss	Secondes	43

➔ il existe d'autres variantes : voir l'API

Exemple :

```
GregorianCalendar gc = new GregorianCalendar();
SimpleDateFormat sdf = new SimpleDateFormat("dd/MMMM/yy");
System.out.println ( sdf.format( gc.getTime()));
```

➔ 7/octobre/21 (aujourd'hui)

Annexe 11B - Transtypage

```
// byte<short<int<long<float<double
int i = 10;
long l = i;             // 10

// tronquer
double d = 10.02;
long l = (long)d;       // 10

// de type prédéfini à String
String.valueOf(10);     // "10"

// de String à int
Integer.parseInt("10");   // 10

// de String à double
Double.parseDouble("10"); // 10.0

// de classe à sous-classe
Object o = new Point(3,5); // type de référence Object, type d'objet Point
Point p = ( Object ) o; // type de référence et type d'objet est Point
```

Annexe 12 - Sous-Classes / héritage

Une sous-classe hérite des methodes et des variables d'instance de sa superclasse. Elle peut donc les utiliser, les redefinir et en créer d'autres, en autant que les modificateurs d'accès le permettent.

Signature d'une sous-classe :

`public class` *nom de la sous-classe* `extends` *nom de la classe*

Mot-clé super :

- permet d'accéder aux méthodes de la superclasse qui ont été modifiées à l'intérieur de la sous-classe.

Ex.:

Voir annex 12

On ajoute super devant une methode afin d'instister pour que la methode apppeler sois celle de la superclasse(en cas de confusion)

- super est très utilisé dans les constructeurs de sous-classes.

Ex.: Voir annex 12

NB. :super(paramètres) doit absolument être a la premiere ligne

Exercice :

A- Créer une classe Prisme servant à modéliser des prismes rectangulaires. Au-delà des méthodes habituelles, codez 2 constructeurs : un initialisant toutes les variables à 1 unité et l'autre permettant d'initialiser les variables à des valeurs quelconques.

B- De plus, coder une méthode permettant de calculer l'aire de la surface du `Prisme` modélisé et une permettant de calculer le volume du `Prisme` modélisé.

C- Créer dans le même package que la classe `Prisme` une nouvelle classe - Cube - modélisant des cubes !

1) Cette classe est une sous-classe de Prisme
2) Bâtir les deux constructeurs appropriés (faire appel aux constructeurs de Prisme avec super)
3) Déterminer si de nouvelles versions d'aire et de volume sont nécessaires

D- Créer dans le même package que la classe Prisme une nouvelle classe - `PyramideBaseCarree` – modélisant des pyramides à base carrée

1) Cette classe est une sous-classe de Prisme
2) Bâtir les deux constructeurs appropriés
3) Redéfinir les méthodes aire() et volume() (il vous faut connaître l'apothème)

E-Pour tester le tout, créer une classe TestPyramide contenant qu'une méthode, soit `main ( String [] args )` dans laquelle vous…

- créez deux prismes:
  - p1 avec le constructeur par défaut
  - p2 ayant une longueur et une hauteur de 2 unités et une largeur de 4 unités

- créez deux pyramides à base carrée : py1 avec le constructeur par défaut, py2 avec une hauteur de 6 et une arête de base de longueur = 9.

- créez un Cube c1 avec une arête de longueur 35.

- affichez les données / résultats de méthodes suivantes, lorsque possible :

- p1.setLongueur(25)
- p1.aire()
- py2.volume ()
- py1.setHauteur (35)
- py1.volume()
- py2.aire()
- c1.aire()

Rep

102

162

11.6

2167350

Annexe12B La classe DecimalFormat

Les objets `DecimalFormat` permettent de modifier l'apparence (`formater`) des nombres décimaux. Un objet `DecimalFormat` comprend un modèle qui est appliqué aux différents nombres décimaux à l'aide de la méthode format.

Les `DecimalFormat` ne servent qu'à modifier l'affichage d'un nombre. La valeur intrinsèque du nombre demeure intacte.

Exercice :

1. Créez une classe ne contenant qu'une méthode main pour tester.

2. Déclarez une variable de type double valant 1234865.579

3. À l'aide de l'API Java (classe `DecimalFormat`) , créez les modèles correspondants aux situations suivantes et appliquez-les à la valeur double à l'aide de la méthode format:

A) Affichez le nombre avec deux décimales. Est-ce arrondi ? Oui, l'affichage est arrondi ne pas confendre abec round up
B) Affichez le nombre comme il s'agissait d'un montant monétaire
C) Affichez le nombre avec des séparateurs pour les milliers
D) Affichez le nombre avec 5 décimales

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 )
         ...
      }
   ...}
```