TP 0(Révisions)

On désire écrire un programme de prise en main de JAVA à travers NETBEANS ou Eclipse.

1. Créer un projet java en Utilisant NETBEANS ou ECLIPSE
2. Ajoutez à votre projet une classe ***Application*** et y ajoutez la méthode représentant le point d'entrée d'un programme JAVA
3. Dans la méthode main ajoutez un code qui permet d'afficher le message **« *Bonjour Tout Le monde* »**
4. Toujours dans le ***main***, ajouter un bout de code qui demande à l'utilisateur de donner son nom, sexe et année de naissance, puis affiche un message sous la forme **« *M./Mme nom vous avez .... ans* »**

   Indication : (A partir des annexes)

   `Utiliser la classe` ***Scanner*** `pour la lecture`

   `Utiliser la classe` ***Date*** `ou` ***GregorianCalendar*** `pour gérer les Dates`
5. Ajouter la déclaration suivante ***double[] values*** dans la méthode main de la classe Principale. Que représente cette déclaration ?
6. Ajouter la déclaration suivante **`double[] tab2={7,16,8,22,15.3,8,7};`** dans la méthode main de la classe Principale. Que représente cette déclaration ?
7. Ajouter a la classe Principale la méthode suivante :

```
static void lireValeur(double[] v){
        Scanner keyb=new Scanner(System.in) ;
        System.out.println("Entrez le nombre de valeurs") ;
        int n=keyb.nextInt() ;
        v=new double[n] ;
        for(int i=0 ;i<n ;i++){
                System.out.print("Valeur de rang  "+(i+1)+ " :" ) ;
                v[i]=keyb.nextDouble() ;
        }
}
```

   a) Quelle différence faites-vous entre **System.out.println** et **System.out.print**
   b) Quel est le mode de passage du paramètre v de cette fonction ?
   c) Quelle est la visibilité de cette méthode ?
   d) A quoi renvoi le mot ***static*** de la fonction (GOOGLE is your Friend) ?
   e) Que fait cette méthode ?
8. Ajouter à la classe Principale, une méthode ***static void affiche(double[] v)*** qui affiche les valeurs du tableau v
9. Ajouter à la classe Principale, une méthode ***static double minimum(double[] v)*** qui renvoie le minimum du tableau passé en paramètre ; Faites pareil pour les méthodes maximum, moyenne, variance, ecartType.
10. Ajouter à la classe Principale, une méthode ***static double[] fusion(double[] v1, double v2)*** qui renvoie un tableau contenant toutes les valeurs de v1 et v2. Les doublons sont autorisés.

    Indication : *`Penser à Arrays.copyOf(…)`*
11. Ecrire une méthode ***static boolean contient(double[] v, double x)*** qui renvoie un booléen indiquant si le tableau v contient x
12. Ajouter à la classe Principale, une méthode ***static double[] fusion2(double[] v1, double v2)*** qui renvoie un tableau contenant la fusion de v1 et v2. Les doublons ne sont pas autorisés.
13. Dans le ***main***, écrire la suite d'instructions permettant de : lire les valeurs, afficher le minimum et maximum, moyenne, ecrat type, fusionner les values et tab2(avec les deux techniques) et afficher les résultats.
14. Faites le nécessaire pour placer vos classes dans le paquetage ***java.novice***

TP 1

Les classes de ce TP seront positionnées dans le paquetage ***java.tp.geometrie***

Exercice 1 - Point

On cherche à écrire une classe **Point** stockant un point graphique en coordonnées cartésiennes (appelons les x et y).

1. Déclarer une classe **Point** contenant les deux champs privés x et y. Puis essayer le code suivant dans la méthode ***main*** de la classe Point.

```
Point p=new Point();
System.out.println(p.x+" "+p.y);
```

Expliquer pourquoi l'accès aux champs privés est autorisé ici.

2. Créer une classe **TestPoint** (dans un fichier ***TestPoint.java***) et recopiez le ***main*** de Point dans la classe **TestPoint**.
   Quel est le problème à l'exécution de la classe TestPoint? Comment peut-on le corriger ?
3. Pourquoi doit-on toujours déclarer les champs privés ?
4. Qu'est-ce qu'un accesseur ? Quels sont les accesseurs que l'on doit mettre ici ?
5. Ajouter un constructeur initialisant les coordonnés du point avec deux paramètres (appelons les px et py).
   Quel est le problème ? Quel est la solution ?
6. Modifier le constructeur précédent pour que les deux paramètres s'appellent x et y.
   Quel est le problème ? Quel est la solution ?
7. Comment faire en sorte de connaître, à tout moment, le nombre total de points qui ont été créés par la classe depuis le lancement de l'application ? Créer une méthode ***totalCreated***() de sorte à obtenir le comportement suivant:

```
public static void main(String[] args) {
     Point p = new Point();
     System.out.println(p.x+" "+p.y);                 // 0 0
     System.out.println("Total: " + totalCreated()); // Total: 1
     Point p1 = new Point(1,1);
     System.out.println("Total: " + totalCreated()); // Total: 2
     Point p2 = new Point(2,2);
     System.out.println("Total: " + totalCreated()); // Total: 3
     Point p0 = new Point();
     System.out.println("Total: " + totalCreated()); // Total: 4
}
```

8. Écrire un autre constructeur qui prend un point en paramètre et utilise les coordonnées de celui-ci pour initialiser le point courant. Comment le compilateur fait-il pour savoir quel constructeur appeler

Exercice 2 - Test d'égalité

En utilisant la classe `Point` de l'exercice précédent.

```
Point p1=new Point(1,2);
Point p2=p1;
Point p3=new Point(1,2);
System.out.println(p1==p2);
```

```
System.out.println(p1==p3);
```

1. Qu'affiche le code ci-dessus ? Expliquez.
2. Écrire dans la classe **Point** une méthode ***isSameAs***() (à vous de trouver la signature exacte de la méthode) qui renvoie **true** si deux points ont les mêmes coordonnées.
3. La classe java.util.ArrayList fournit une implementation correspondant à un tableau qui s'agrandi dynamiquement, au besoin. À quoi selon vous peut servir la méthode ***indexOf*** de cette classe?
4. Executez le code suivant :

```
public static void main(String[] args){
  Point p1=new Point(1,2);
  Point p2=p1;
  Point p3=new Point(1,2);
  ArrayList list = new ArrayList();
  list.add(p1);
  System.out.println(list.indexOf(p2));
  System.out.println(list.indexOf(p3));
}
```

   Quel est le problème avec les résultats affichés sur la console. Note : ici, le compilateur génère des *warnings* à la création et à l'utilisation de l'ArrayList. Nous verrons dans les prochains chapitres comment les éviter.

5. Quelle méthode de Point est appelée par ***ArrayList.indexOf*** ?
   Lire la doc !!!
6. Modifier la classe Point pour que ***indexOf***() teste suivant les coordonées des Points et pas suivant les références.
7. Utiliser l'annotation **@Override** pour vérifier que vous avez bien écrit la signature de la méthode ajoutée à Point.
8. A quoi sert l'annotation **@Override** ?

Exercice 3 - Comment afficher un Point ?

On aimerait pouvoir afficher les caractéristiques d'un point, par le code Java suivant :

```
Point point=...
System.out.println(point);
```

Java sait faire cela, à condition de mettre dans la classe Point une méthode **public String toString()** (la définition de cette méthode est dans la classe ***java.lang.Object***) retournant une chaîne de caractères, qu'on construit typiquement à partir des attributs de l'objet. Rappel: en Java on peut faire un '+' entre une String et n'importe quoi, le résultat est la concatenation entre la String et le n'importe quoi vu comme une suite de caractère.

1. Ecrire cette méthode, pour obtenir par exemple l'affichage suivant : **(x,y)**
2. Peut-on utiliser l'annotation @Override, ici ?

Exercice 4 - Les archives

1. Générez l'archive de votre projet
2. Créer un nouveau projet et ajouter votre comme librairie au nouveau projet

3. Créer une Classe TestPoint dans ce nouveau Projet qui crée deux Points, les affiche et ensuite teste l'égalité entre les deux points et affiche le résultat.

TP 2 :

Le but de ce TP noté, est d'écrire quelques classes simples permettant de vérifier que vous maitrisez la mise en oeuvre des mécanismes de base pour la création de classes et d'objets, de composition, de délégation et de responsabilité, de sous-typage par héritage ou implémentation d'interfaces, ainsi que l'utilisation de collections élémentaires pour la réalisation de petits algorithmes simples en Java. Ces classes seront positionnées dans le paquetage **com.java.shop**

On souhaite écrire des classes modélisant des articles (en vente dans un supermarché par exemple), ainsi qu'un panier d'articles permettant de les stocker.

Exercice 1 - Des articles (3 points)

1. Écrire une classe **Item** représentant un article ayant deux champs, **name** pour son nom de type **String** et **price** pour son prix en CFA de type **long** (on supposera que toutes les manipulations de prix doivent se faire avec des entiers long). Cette classe devra posséder deux accesseurs permettant d'obtenir le nom de l'article et son prix, ainsi qu'un constructeur qui permet d'en créer des instances. On veut que le prix d'un article et son nom ne changent pas pour toute la durée de vie de l'objet.
2. Écrire une classe **Main** permettant de tester la création d'un article et le fonctionnement des accesseurs de la manière suivante:

```
Item item = new Item("corn flakes", 500);
System.out.println(item.getPrice());        // affiche: 500
System.out.println(item.getName());         // affiche: corn flakes
```

3. On souhaite que lorsque l'on affiche un article avec System.out.println(), le nom de l'article s'affiche suivi de deux points (":"), d'un espace et de son prix en CFA avec séparateur de millier le tout suivi d'un espace et du symbole 'CFA'. Par exemple:

```
Item item = new Item("corn flakes", 5000)
 System.out.println(item);        // affiche: corn flakes: 5 000 CFA
 Item chewingGum = new Item("chewing gum",1403);
 System.out.println(chewingGum);// affiche: chewing gum: 1 403 CFA
```

Indication: utilisez la classe DecimalFormat

```
DecimalFormat f=new DecimalFormat("#,## 0CFA");
System.out.println(f.format(2555)); // affiche 2 555 CFA
```

Exercice 2 - Le panier d'achats (8 points)

1. Écrire une classe **ShoppingCart** modélisant un panier d'articles dans lequel on peut:
   - ajouter un article avec **addItem**();
   - retirer un article avec **removeItem**(), qui devra renvoyer false si **l'item** que l'on essaye de supprimer n'existe pas;
   - connaître le nombre d'articles avec **itemCount**();
   - calculer le prix total du panier avec **totalPrice**() (vous écrirez en commentaire au début de la méthode quel est l'ordre de grandeur de la complexité de cette méthode);

Indication: utilisez une collection de java.util pour stocker les articles du panier. Modifier votre Main pour faire des tests en utilisant le code ci-dessous

```
Item item1 = new Item("corn flakes", 500);
Item item2 = new Item("caviar", 50000);
Item item3 = new Item("water", 101);
ShoppingCart cart = new ShoppingCart();
cart.addItem(item1);
cart.addItem(item2);
cart.addItem(item3);
```

```
System.out.println(cart.itemCount());       // affiche: 3
System.out.println(cart.totalPrice());      // affiche: 50601
```

2. Vérifier que le code suivant affiche bien 0.

```
ShoppingCart cart = new ShoppingCart();
Item item = new Item("corn flakes", 500);
cart.addItem(item);
cart.removeItem(new Item("corn flakes", 500));
System.out.println(cart.itemCount());       // affiche: 0
```

   Sinon, changer votre implémentation en conséquence.

3. Ajouter un poids (**weight** de type **int**) exprimé en grammes à la classe **Item** et modifiez le constructeur pour qu'il accepte ce paramètre. Modifier l'implantation de votre panier pour que l'on ne puisse pas ajouter d'article dans le panier si le poids de ce dernier doit dépasser 10 kg. Vous utiliserez pour cela l'exception **IllegalStateException**. Attention: le test du poids devra se faire en temps constant (O(1)). Modifier votre **Main** pour vérifer que cela fonctionne. Par exemple, testez:

```
Item item1 = new Item("corn flakes", 501, 1000);
Item item2 = new Item("caviar", 50000, 500);
Item item3 = new Item("water", 500, 5000);
ShoppingCart cart = new ShoppingCart();
cart.addItem(item1);
cart.addItem(item2);
cart.addItem(item3);
// cart.addItem(item3); // lève java.lang.IllegalStateException
cart.removeItem(new Item("eau", 500, 5000));
// cart.addItem(item3);  // lève java.lang.IllegalStateException
cart.removeItem(new Item("water", 500, 5000));
cart.addItem(item3);        // ajout possible!
```

   Indication: penser que **removeItem**() a une valeur de retour.

4. On souhaite que *chaque panier d'achat créé puisse disposer automatiquement à sa création d'un numéro de série unique* (qui commence à 1 et qui est incrémenté de 1 à chaque nouveau panier créé), et qui soit connu comme l'identifiant (**id**) de ce panier. Ajoutez une méthode **getId**() à la classe **ShoppingCart** qui retourne cet entier de type int, et tout ce dont vous avez besoin pour l'implémenter. Par exemple, vous pouvez tester avec le code suivant.

```
ShoppingCart c1 = new ShoppingCart();
System.out.println(c1.getId());         // affiche: 1
Item item1 = new Item("corn flakes", 501, 1000);
c1.addItem(item1);
Item item2 = new Item("caviar", 50000, 500);
c1.addItem(item2);
System.out.println(c1.getId());         // affiche: 1
ShoppingCart c2 = new ShoppingCart();
ShoppingCart c3 = new ShoppingCart();
Item item3 = new Item("water", 500, 5000);
c3.addItem(item3);
System.out.println(c2.getId());         // affiche: 2
System.out.println(c3.getId());         // affiche: 3
```

5. Ajouter à la classe **ShoppingCart** une méthode **toString**() qui retourne une représentation du contenu du panier, commençant par l'identifiant unique du panier et le nombre d'articles contenus, puis affichant tous les articles du panier, un article par ligne. Par exemple, avec le code d'exemple de la question précédente, vous devez obtenir:

```
System.out.println(c1); // affiche: panier 1 [2 article(s)]
```

```
                          //            corn flakes: 501 CFA
                          //            caviar: 500 CFA
System.out.println(c2); // affiche: panier 2 [0 article(s)]
System.out.println(c3); // affiche: panier 3 [1 article(s)]
                          //            water: 500 CFA
```

Exercice 3 - Des produits frais (3 points)

Certains articles sont particulièrement frais et peuvent nécessiter la prise en compte d'une date limite de consommation. Néanmoins, lorsqu'on les met dans un panier d'achat, ils se comportent comme des articles classiques et ont un nom, un prix et un poids.

1. Écrire une classe (**FreshItem**) qui correspond à un article frais ayant, en plus des informations stockées dans la classe Item, un champ **bestBeforeDate** de type String correspondant à la date limite de consommation au format YYYY-MM-DD.
   L'affichage d'un article frais par **System.out.println**() doit afficher les informations de l'article dans le même format que pour un Item, mais précédées de la date limite de consommation.
   Testez avec:
   ```
   Item item1 = new Item("corn flakes", 500, 1000);
   System.out.println(item1);     // affiche: corn flakes: 500 CFA
   FreshItem fresh = new FreshItem("Salmon", 1450, 800, "2012-04-11");
   System.out.println(fresh);     // affiche: BBD:2012-04-11 Salmon: 1450 CFA
   ```
2. Verifier que le code suivant fonctionne, sinon faites les changements qui s'imposent.
   ```
   Item tin = new Item("sardine", 500, 500);
   FreshItem fresh = new FreshItem("sardine", 500, 500, "2012-04-11");
   ShoppingCart cart = new ShoppingCart();
   cart.addItem(fresh);
   cart.removeItem(tin);
   System.out.println(cart); // affiche: panier 1 [1 article(s)]
                             //            BBD: 2012-04-11 sardine: 5.00 €
   ```

Exercice 4 - La petite note (6 points)

On souhaite maintenant éditer des factures, mais pas seulement d'articles ou d'articles frais, mais plus généralement pour une liste de choses qui peuvent être payées.

1. Pour commencer, on représente par le type **Ticket** des billets d'évènements ou de spectacles qui peuvent être vendus dans le même magasin que nos articles (Item) ou nos articles frais (FreshItem). Un billet est représenté par une référence (**reference** de type String) et par un prix en centimes d'euros (**price** de type long).
   Écrire une classe Ticket avec les champs nécessaires et un constructeur permettant de créer, par exemple:
   ```
   Ticket ticket = new Ticket("RGBY17032012 - Walles-France", 9000);
   ```
2. On souhaite maintenant disposer d'un type **Payable**, qui dispose des méthodes:
   - **label**() qui retourne une String représentant une descritpion textuelle de ce qui doit être payé;
   - **cost**() qui retourne le coût de ce qui doit être payé (un entier long);
   - **taxRatePerTenThousand**() qui retourne la proportion du coût qui est de la taxe, exprimée en centièmes de pourcents (en "pour-dix-mille") sous la forme d'un entier long. Par exemple, dans cette unité, 550 représente un taux de taxe de 5,5% et 1960 représente un taux de taxe de 19,6%;

   Définir le type **Payable** et modifier la classe **Ticket** de sorte que le code suivant fonctionne (on considérera que toutes les instances de la classe Ticket sont par défaut taxées à 19,25%):
   ```
   Payable payable = new Ticket("RGBY17032012 - Walles-France", 9000);
   System.out.println(payable.label())// affiche: RGBY17032012 - Walles-France
   System.out.println(payable.cost());                    // affiche: 9000
   System.out.println(payable.taxRatePerTenThousand());  // affiche: 19,25
   ```

3. On représente maintenant une facture comme une liste de choses à payer. Créer une classe ***Invoice*** qui dispose pour l'instant d'un seul constructeur sans argument et d'une unique méthode ***add(Payable p)*** qui ajoute à la liste de choses à payer l'argument p. Le code suivant doit fonctionner:

```
Invoice invoice = new Invoice();
Payable payable = new Ticket("RGBY20120317 - Walles-France", 9000);
Ticket ticket = new Ticket("MUSI20120612 - RollingStones",12000);
invoice.add(payable);
invoice.add(ticket);
```

4. Faites ce qu'il faut pour que la dernière ligne du code suivant fonctionne également (toutes les instances de la classe ***Item*** sont taxées à ***10%*** par défaut).

```
Invoice invoice = new Invoice();
Payable payable = new Ticket("RGBY20120317 - Walles-France", 9000);
Ticket ticket = new Ticket("MUSI20120612 - RollingStones",12000);
invoice.add(payable);
invoice.add(ticket);
Item item = new Item("corn flakes", 500, 1000);
invoice.add(item);
```

5. Pour les articles frais, instances de ***FreshItem***, la taxe est réduite de ***0,1%*** par tranche d'un kilo de produit. Par exemple, les sardines en boite sont taxées à 10%, mais les sardines fraiches sont taxées à 10% s'il y en a moins d'1 kg, et à 10% - 0,1% s'il y en a entre 1 et 2 kg... Modifiez ce qu'il faut et vérifiez:

```
Item tin = new Item("sardine", 500, 500);
FreshItem fresh = new FreshItem("sardine", 500, 500, "2012-04-11");
FreshItem fresh2 = new FreshItem("sardine x3", 1500, 1500, "2012-04-11");
System.out.println(tin.taxRatePerTenThousand());     // affiche: 1000
System.out.println(fresh.taxRatePerTenThousand());   // affiche: 1000
System.out.println(fresh2.taxRatePerTenThousand());  // affiche:  990
```

6. On souhaite ajouter dans la classe ***Invoice*** deux méthodes : ***totalAmount***() qui retourne un long représentant le montant total des choses à payer de cette facture, et ***totalTax***() qui retourne un long représentant le montant total des taxes de cette facture. Que proposez-vous pour que le code suivant ne soit pas obligé de parcourir 2 fois l'ensemble des articles de la facture.

```
Invoice invoice = new Invoice();
invoice.add(tin);
invoice.add(fresh);
invoice.add(fresh2);
System.out.println(invoice.totalAmount());  // affiche: 2500
System.out.println(invoice.totalTax());     // affiche:  248
```

TP 3