



Exercices Java

L'encapsulation

- Série 1 -





Sommaire

- 1) **Objectifs** : Découvrir et expérimenter la création de **classes**. Mettre en œuvre les mécanismes de base de la P.O.O. : l'encapsulation, le constructeur, les accesseurs. Assimiler la relation **d'agrégation**. Maîtriser le mécanisme des **exceptions**.
- 2) **Estimation du temps de réalisation** : 6 heures.
- 3) **Vocabulaire utilisé** : encapsulation, méthode filtre, accesseurs, casting, classe, énumération, exception , polymorphisme, agrégation, ...
- 4) **Environnement technique** : J2SE, JDK 7. Un EDI (type *Eclipse/NetBeans*).
La documentation **Java Oracle** .
- 5) **Pré-requis et recommandations** :
 - a) Réalisez **préalablement** le **Q.C.M.** du document **QCM-Exercices-Java-Série-1** avec rigueur et réflexion avant de réaliser les T.P.s. Donnez un sens à chaque réponse que vous jugerez bonne.
 - b) N'abordez pas les TP dédiés aux **classes** tant que ces notions ne vous sont pas complètement familières.



TP 1.1 : *Calculatrice*

Objectifs :

Assimiler les concepts de **classe** et de **méthodes de classe** en **Java**. Maîtriser le mécanisme des **exceptions**.

Déroulement :

Vous allez, dans ce TP nommé *Calculatrice*, reprendre la notion de *méthode static* abordée dans le support « *Objets et Classes - Principes de l'encapsulation* ». Pour cela, vous construirez une classe *Calculatrice* permettant d'effectuer les 4 opérations arithmétiques de base, par le biais d'une méthode de classe *calculer*.

Il est demandé de créer un projet **Java SE** avec l'IDE de votre choix, - *NetBeans*, *Eclipse*, ...

Remarques :

Revoyez éventuellement comment créer un projet avec *NetBeans* – ou *Eclipse* – dans la séance d'apprentissage « *S'approprier l'environnement de développement* ».

Au cours de ce TP, vous mettrez en évidence :

- ✓ La notion de **classe**.
- ✓ La notion de **méthode de classe**, vue comme un utilitaire.
- ✓ La notion **d'exception** par la suite. (lorsque vous aurez acquis les compétences relatives aux exceptions).

Etapas de réalisation

1. Créez une classe **Calculatrice** permettant le calcul et l'affichage des 4 opérations arithmétiques de base. Prévoyez pour cela une **méthode de classe** *calculer* qui admettra la liste de paramètres suivante : (*char* operateur , *double* var1 , *double* var2). L'opérateur, correspondant au premier argument, sera choisi dans la liste suivante : '+', '-', '*', ou '/'.

2. La méthode *calculer* renverra le **résultat** de l'opération effectuée.

3. Appelez la calculatrice avec plusieurs jeux d'essais comme par exemple :

```
Calculatrice.calculer( '+', 1, 2 );  
Calculatrice.calculer( '-', 2078, 64 );  
Calculatrice.calculer( '*', 15, 3 );  
Calculatrice.calculer( '/', 42, 9 );  
Calculatrice.calculer('/', 15, 2) + Calculatrice.calculer('*', 2, 3)
```

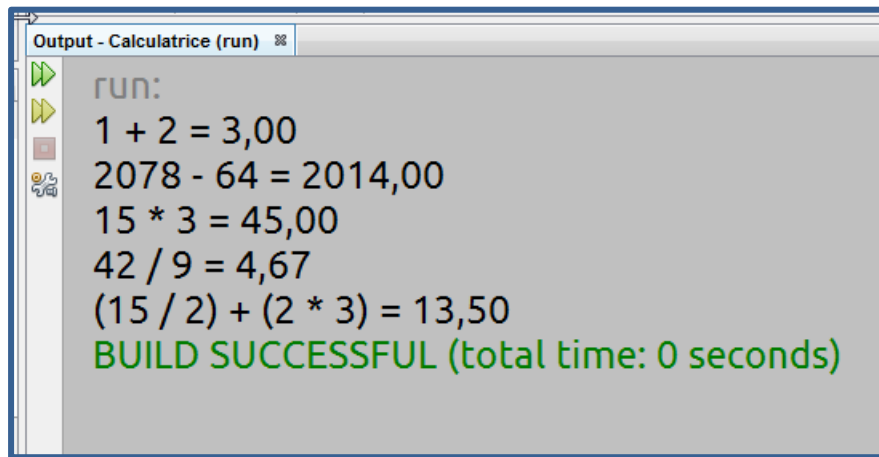


Figure 1 : Les résultats des calculs précédents

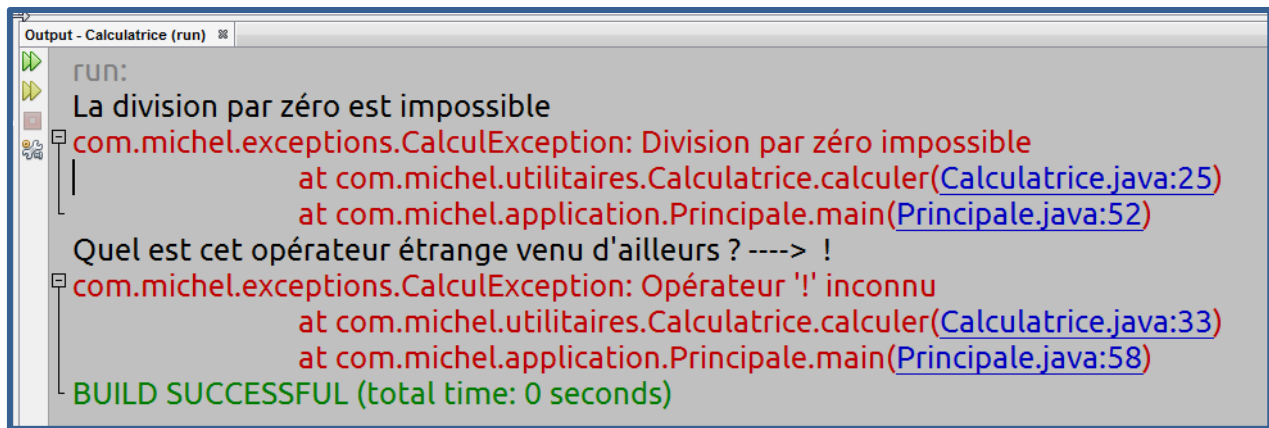
4. Lorsque vous aurez parcouru le support dédié aux **exceptions**, codez la fonctionnalité suivante : si la tentative d'une division par zéro est constatée, veuillez prévoir une **levée d'exception**, ainsi que l'affichage d'un message explicite .
5. Si un opérateur fantaisiste est envoyé à la méthode *calculer* (autre que '+', '-', '*', ou '/'), une **exception sera également levée** mais avec un message adapté à cette situation précise qui produiront les résultats de la figure 2.

PS : N'oublier pas d'entourer chaque appel à *Calculatrice.calculer(...)* par **try / catch** .

6. Ajoutez par exemple ces deux appels pour vérifier le bon fonctionnement de votre gestionnaire d'exceptions.

```
try {  
    System.out.printf("15 / 0 = %2.2f\n", Calculatrice.calculer('/', 15, 0));  
} catch (CalculException e) {  
    e.printStackTrace();  
}  
  
try {  
    System.out.printf("1 ! 2 = %2.2f\n", Calculatrice.calculer('!', 1, 2));  
} catch (CalculException e) {  
    e.printStackTrace();  
}
```

qui devraient produire :



```
Output - Calculatrice (run)
run:
La division par zéro est impossible
com.michel.exceptions.CalculException: Division par zéro impossible
    at com.michel.utilitaires.Calculatrice.calculer(Calculatrice.java:25)
    at com.michel.application.Principale.main(Principale.java:52)
Quel est cet opérateur étrange venu d'ailleurs ? ----> !
com.michel.exceptions.CalculException: Opérateur '!' inconnu
    at com.michel.utilitaires.Calculatrice.calculer(Calculatrice.java:33)
    at com.michel.application.Principale.main(Principale.java:58)
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 2 : Les affichages liés aux exceptions

Aides pour le TP 1.1

1. Utilisez la méthode *System.out.printf* pour afficher une chaîne de caractères formatée dotée de spécificateurs de format pour chaque variable (par exemple *System.out.printf("%2.2f + %2.2f\n", 7.5, 3.2)* affichera 7.50 + 3.20) ou employez l'instruction *System.out.println*.
2. Utilisez les mots-clés ***switch/case*** pour distinguer et évaluer les différents caractères de l'opérateur arithmétique.
3. Le retour de l'opération sera exploité et affiché dans la fonction principale *main*.

TP 1.2 : *PrincipesEncapsulation*

Objectifs : Assimiler les principes liés à une propriété fondatrice de la P.O.O. : **l'encapsulation**.

Déroulement : Vous allez dans ce TP reprendre l'ensemble des notions abordées dans le support « *Objets et Classes - Principes de l'encapsulation* ». Pour cela, vous construirez une classe **Personne** que vous enrichirez au fil des étapes .
Il est demandé de créer un projet **Java SE** avec l'IDE de votre choix, - **NetBeans**, **Eclipse**, ...

Remarques : Revoyez éventuellement comment créer un projet avec **NetBeans** – ou **Eclipse** – dans la séance d'apprentissage « *S'approprier l'environnement de développement* ».

Au cours de ce TP, vous mettrez en évidence :

- ✓ La notion de classe.
- ✓ La notion de constructeurs surchargés.
- ✓ La notion d'accesseurs en lecture/écriture sur les variables d'instances privées.
- ✓ L'utilisation de méthodes filtres qui permettent la vérification de valeur (ici dans un **constructeur** et un **setter**).

Réalisez les étapes chronologiques suivantes :

1. Créez un projet **Java SE** standard nommé *PrincipesEncapsulation*.
2. Créez une classe **Personne** dans un package dédié. (*com.votre_prenom.entites*).



NB : Dans tous les TP à venir, vous créerez systématiquement vos classes type entités dans ce package.

3. Ajoutez les variables d'instances de type *String* **nom** et **societe**. N'oubliez pas de les rendre **private**.
4. Mettez en place les **accessors/mutators** grâce aux assistants de **NetBeans**.
5. Mettez en place les **constructeurs** dans cette classe **Personne**. Un avec **un** paramètre de type *String* pour valoriser le nom (la société sera inconnue). Un autre avec **deux** paramètres de type *String* pour valoriser le nom et la société. Utilisez **this** pour qualifier la ou les variables d'instance. Passez le nom de la personne en majuscule (*toUpperCase*). Là aussi, utilisez les assistants.
6. La convention « ? » pour une société inconnue sera utilisée et consignée dans une méthode qui va être ajoutée un peu plus tard dans cette progression.

7. Ajoutez la constante suivante, comme suggéré dans le support :

```
// Variable de classe matérialisant le non rattachement à une société selon notre convention .  
private static final String PAS_DE_SOCIETE = "?";
```

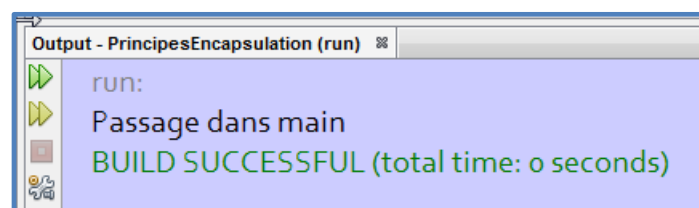
8. Ajoutez la méthode *afficher()* qui affichera à l'écran le nom de la personne et son employeur s'il est connu.
9. Créez, maintenant , la classe *Principale* dans le package (*com.votre_prénom.application*) et insérez-y la méthode *static main* . C'est dans cette méthode que vous ferez les instanciations.
10. Votre classe devrait ressembler à ceci :

```
// Nom du package où est définie la méthode main  
package com.michel.application;  
  
// Importation de la classe Personne avec son package d'appartenance  
import com.michel.entites.Personne;  
  
/**  
 *  
 * @author Michel-HP  
 */  
public class Principale {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        .....  
    }  
}
```

11. Procédez par étapes : ajoutez l'instruction suivante dans *main* pour vous rassurer sur la validité de votre projet :

```
System.out.println("Passage dans main");
```

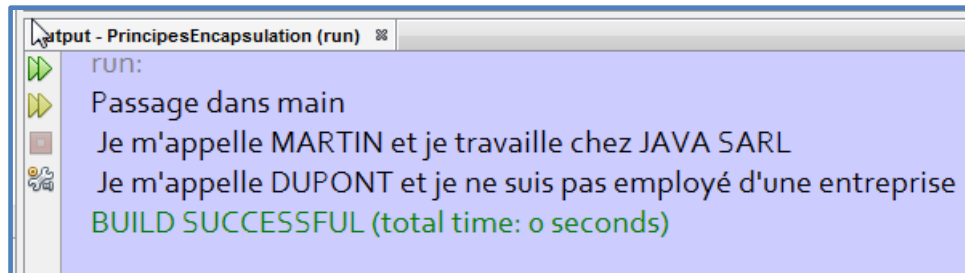
... Et lancez votre programme . Vous devriez afficher :



12. Instanciez plusieurs objets *Personne* dans la classe principale *main*. Produisez les affichages suivants. Par exemple :

```
Personne martin = new Personne ( "martin","Java SARL");
Personne dupont = new Personne ("dupont") ;
martin.afficher();
dupont.afficher();
```

13. Affichage des propriétés de chaque instance.



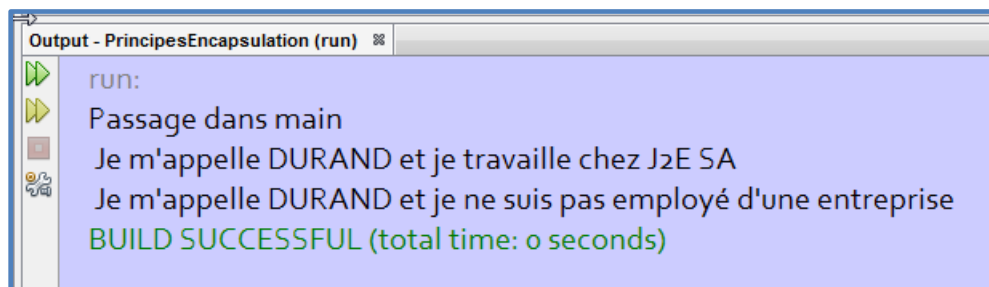
```
run:
Passage dans main
Je m'appelle MARTIN et je travaille chez JAVA SARL
Je m'appelle DUPONT et je ne suis pas employé d'une entreprise
BUILD SUCCESSFUL (total time: 0 seconds)
```

14. Ajoutez la méthode *quitterSociete()* qui met «?» (avec la constante *PAS_DE_SOCIETE*) dans la v.i. *societe* . Si la personne était déjà sans société, on quittera, arbitrairement, le programme en affichant les caractéristiques de la personne.

```
public void quitterSociete() {
    if ( ..... ) {
        // La personne n'est pas rattachée à une société
        afficher (); // On décide d'arrêter l'application
        System.out.println ( "Je ne suis pas salarié : impossible de quitter la société" );
        System.exit(1); // Arrêt de l'exécution, code erreur 1
    }
    // Ici, il y a bien une société à quitter → on applique la convention
    societe = PAS_DE_SOCIETE ;
}
```

```
Personne durand = new Personne ("durand", "J2E SA");
durand.afficher();
durand.quitterSociete();
durand.afficher();
```

Le code précédent affiche :

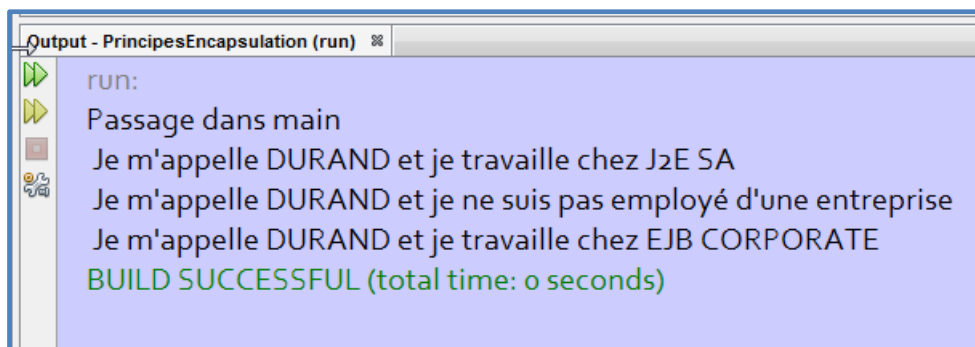


```
run:
Passage dans main
Je m'appelle DURAND et je travaille chez J2E SA
Je m'appelle DURAND et je ne suis pas employé d'une entreprise
BUILD SUCCESSFUL (total time: 0 seconds)
```

15. Changez l'entreprise de *durand* par le **mutator** (aussi appelé *setter*).

```
public static void main(String[] args) {  
  
    System.out.println("Passage dans main");  
    Personne durand = new Personne ("durand", "J2E SA");  
    durand.afficher();  
    durand.quitterSociete();  
    durand.afficher();  
    durand.setSociete("EJB Corporate");  
    durand.afficher();  
}
```

16. De nouveau, affichez les propriétés de *durand*.



```
Output - PrincipesEncapsulation (run) x  
run:  
Passage dans main  
Je m'appelle DURAND et je travaille chez J2E SA  
Je m'appelle DURAND et je ne suis pas employé d'une entreprise  
Je m'appelle DURAND et je travaille chez EJB CORPORATE  
BUILD SUCCESSFUL (total time: 0 seconds)
```

17. Implémentez la méthode filtre *validerSociete (String)* qui sera **private** et appelée par :

- d'une part le constructeur à deux arguments et ...
- d'autre part par le **setter** de *societe* afin de s'assurer que la valeur que l'on tente d'utiliser pour affecter la v.i. *societe* est bien conforme au cahier des charges (longueur <= 30 caractères et nom de la société autre que notre convention (« ? »)).
- Si un de ces critères n'est pas respecté, on quittera l'application après avoir affiché le nom de l'entreprise qui ne convient pas .

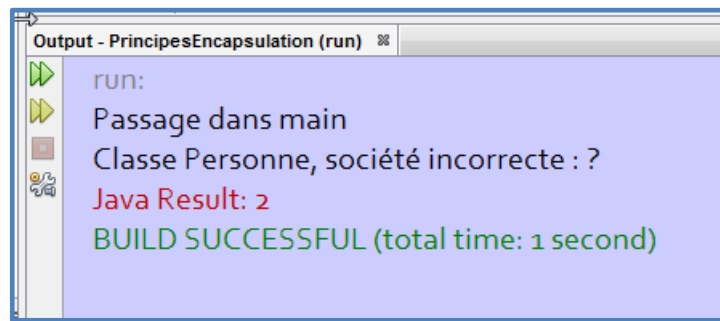
L'appel à cette méthode-filtre (dans le constructeur et le setter) se fera ainsi :

```
societe = validerSociete(entreprise).toUpperCase() ;
```

18. Tentez d'affecter la société de nom «?» pour vérifier que votre méthode filtre est opérationnelle.

```
public static void main(String[] args) {  
    System.out.println("Passage dans main");  
    Personne martin = new Personne ("martin", "?");  
    martin.afficher();  
}
```

```
}
```



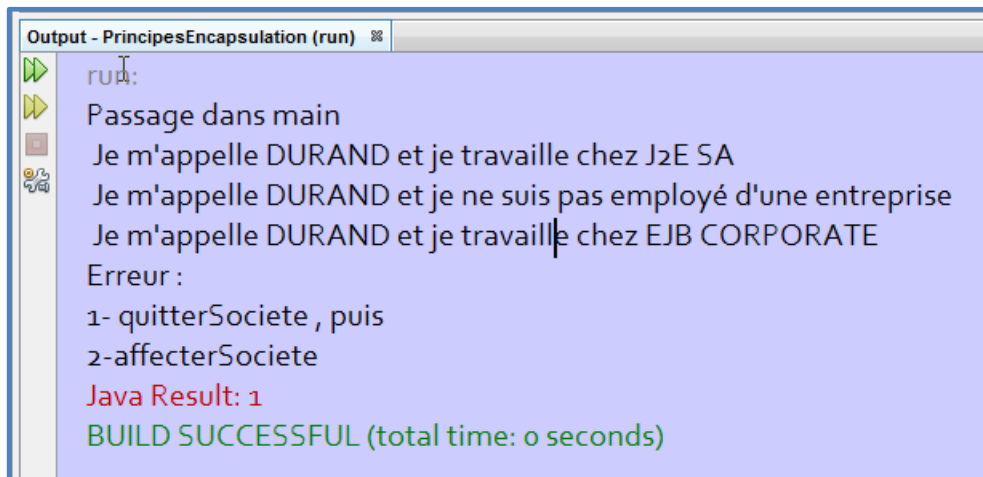
```
Output - PrincipesEncapsulation (run) %  
run:  
Passage dans main  
Classe Personne, société incorrecte : ?  
Java Result: 2  
BUILD SUCCESSFUL (total time: 1 second)
```

19. Vérifiez que deux appels successifs à `setSociete(...)` provoque une erreur : il faut d'abord quitter une société (ici « J2E SA ») **AVANT** d'en intégrer une autre (ici « EJB Corporate »).

```
Personne durand = new Personne ("durand", "J2E SA");  
durand.afficher() ;  
durand.quitterSociete();  
durand.afficher() ;  
durand.setSociete("EJB Corporate");  
durand.afficher() ;  
durand.setSociete("EJB Corporate");  
durand.afficher() ;
```

20. Affichez le mode opératoire à respecter afin d'orienter l'utilisateur lors une telle situation :

Erreur : 1 quitterSociete, puis 2-affecterSociete



```
Output - PrincipesEncapsulation (run) %  
run:  
Passage dans main  
Je m'appelle DURAND et je travaille chez J2E SA  
Je m'appelle DURAND et je ne suis pas employé d'une entreprise  
Je m'appelle DURAND et je travaille chez EJB CORPORATE  
Erreur :  
1- quitterSociete , puis  
2-affecterSociete  
Java Result: 1  
BUILD SUCCESSFUL (total time: 0 seconds)
```

La solution *PrincipesEncapsulation* vous sera fournie en fin de séance au format *zip* afin de comparer votre proposition avec cette proposition.

TP 1.3 : *PersonneAdresse*

Objectifs :

1. Poursuivre l'étude des principes liés à l'**encapsulation**.
2. Mettre en œuvre la relation de **composition** « a-un » entre deux classes.
3. Utiliser les boîtes de dialogue disponibles dans Java 7 avec le package *javax.swing*.
4. Assimiler les notions de **classe**, de **référence**, et de méthode **static**.

Déroulement :

Vous allez dans ce TP reprendre et compléter l'ensemble des notions abordées dans le support « *Objets et Classes - Principes de l'encapsulation* ».

Pour cela, vous construirez une classe **Adresse** avec ses variables d'instances caractérisant cette notion d'adresse, puis une autre classe : **Personne**. Ceci vous fera découvrir la notion de **composition** appelée également **agrégation**.

Parmi les variables d'instance de **Personne**, vous allez en créer une qui représentera le lieu de résidence de la personne, soit une instance de type **Adresse**.

Afin de bien comprendre les notions liées aux **références**, vous ferez en sorte que des personnes déménagent ou bien encore que deux personnes habitent sous le même toit. (elles changeront donc ... d'adresse).

Il est demandé de créer un projet **Java SE** avec l'IDE de votre choix, - *NetBeans*, *Eclipse*, ...

Remarques : Revoyez éventuellement comment créer un projet avec *NetBeans* – ou *Eclipse* – dans la séance d'apprentissage « *S'appropriier l'environnement de développement* ».

Etapas à réaliser :

Créez une classe **Adresse** dans le package dédié (voir page 8). Vous y associerez les variables d'instance suivantes : *numero_rue (int)*, *nom_rue (String)*, *codePostal (int)* et *ville (String)*.

1. Mettez en œuvre les accesseurs (lecture et écriture) associés.
2. Instanciez trois adresses selon le jeu d'essais proposé suivant :

```
2 , « rue Victor Hugo », 75008, « Paris »  
17, « rue de la République », 44000, « Nantes »  
55, « Bld de la Libération », 59000, « Lille »
```

3. Mettez en œuvre le **constructeur** afin de transmettre et renseigner l'ensemble des 4 informations caractérisant une adresse.

4. Mettez en œuvre la redéfinition de *toString()* afin de produire les affichages suivants, dans les boîtes de dialogue (vues précédemment).



5. Créez une classe *Personne* dotée des quatre variables d'instance suivantes, naturellement toutes *private* :

- String prenom.
- String nom.
- int age.
- Adresse adresse.

6. Ajoutez les outils suivants : accesseurs, constructeur, redéfinition de *toString()* qui sera codé de la manière suivante :

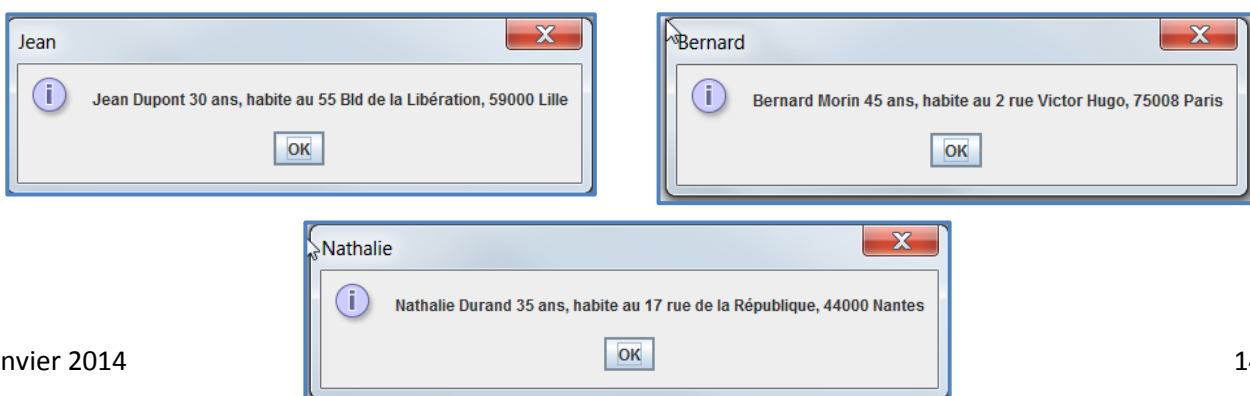
```
@Override
public String toString() {
    return prenom + " " + nom + " " + age + " ans, habite au " + adresse ;
}
```

7. A noter la propriété suivante, extrêmement pratique : *Java*, à la lecture de la variable d'instance *adresse* dans le *return* précédent va automatiquement déclencher la méthode *toString()* de la classe *Adresse*, en l'occurrence celle que vous avez redéfinie dans le point 4.

8. Instancier 3 personnes selon le jeu d'essais proposé suivant :

Jean Dupont 30 ans, habitant à l'adresse de Lille précédente.
Bernard Morin 45 ans, habitant à l'adresse de Paris précédente.
Nathalie Durand 35 ans, habitant à l'adresse de Nantes précédente .

9. Produisez les affichages suivants, grâce aux boîtes de dialogue. Vous interviendrez dans la redéfinition de *toString()* de la classe *Adresse* afin de supprimer la chaîne de caractères « Adresse : » apparaissant au début de la chaîne construite et retournée.



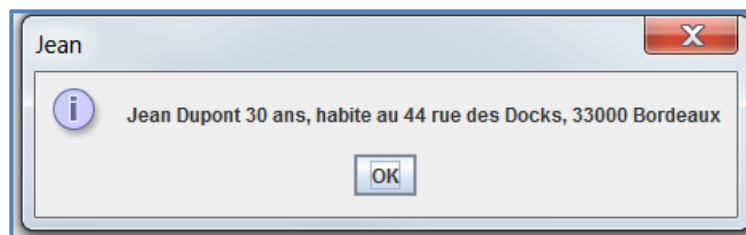
Jean, pour des raisons professionnelles, à l'obligation de déménager. Il habite maintenant à l'adresse suivante : 44 Rue des Docks 33000 Bordeaux .

10. Instancier cette nouvelle adresse et faites déménager Jean. Utilisez pour cela évidemment le **setter** adapté. A noter que vous pouvez instancier cette nouvelle adresse directement en tant que paramètre dans le **setter** de la classe *Personne* . Comme ceci :

```
jean.setXxxx( new Adresse (...) );
```

Une **instance anonyme** sera ainsi créée et transmise au paramètre du **setter** qui, lui-même, le transmettra à la variable d'instance *adresse* de l'instance *jean*.

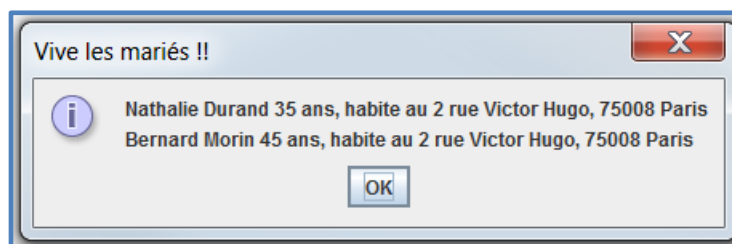
Vérifiez que Jean a bien déménagé.



11. Au hasard d'une rencontre, Nathalie et Bernard se découvrent une passion commune, réciproque et dévorante. Ils se marient et Nathalie rejoint Bernard à Paris pour habiter ensemble . Sans instancier de nouvelle adresse, déménagez Nathalie et contrôlez que les nouveaux mariés résident bien sous le même toit.



Insérez dans la boîte de dialogue la séquence de caractères suivante entre Nathalie et Bernard : « \n ». Elle génèrera un saut de ligne .



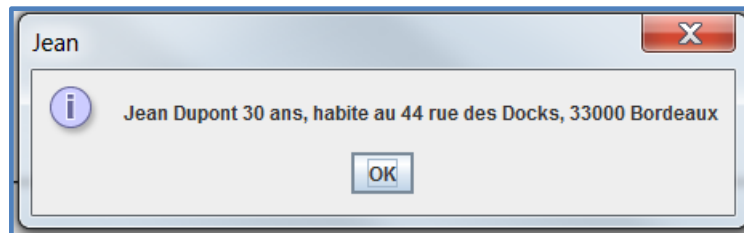
12. Dernier point : créer une classe utilitaire *Outils* dans le package associé *com.votre_prenom.utilitaires* et développez une méthode de classe *afficher* permettant d'éviter d'appeler la syntaxe lourde liée à l'appel de l'affichage de la boîte de dialogue et donc de simplifier l'écriture tout en produisant le même effet.

13. Appelez cette méthode *afficher* de la façon suivante pour afficher classiquement la boîte de dialogue ci-dessous (seuls le titre et le message à afficher sont transmis, le reste est pris en charge par *afficher*).

```
Outils.afficher(« jean », jean ) ; // Concis
```

14. Importez les propriétés statiques de la classe *Outils* afin d'éviter de préfixer l'appel d'*afficher* par *Outils*. Comme ceci :

```
afficher (« jean »,jean) ; // Encore plus concis
```



Félicitations : vous venez de mettre en évidence, grâce à ce TP, l'immense intérêt de manipuler, en **Java**, des objets par le biais de pointeurs qui les désignent : les **références**. Ici une nouvelle référence à permis de déménager Jean puis Nathalie d'une manière très naturelle.

On voit que par une simple affectation, les relations entre deux classes sont très facilement changeables.

TP 1.4 : *Utilisateur* et *Message*

Objectifs :



1. Poursuivre l'étude des principes liés à l'**encapsulation**.
2. Mettre en œuvre la notion d'**énumération**.
3. Utiliser les boîtes de dialogue disponibles dans Java 7 avec le package *javax.swing*.
4. Assimiler les notions de **classe**, de **référence**.

Déroulement :

Vous allez dans ce TP reprendre et compléter l'ensemble des notions abordées dans le support « *Objets et Classes - Principes de l'encapsulation* ».

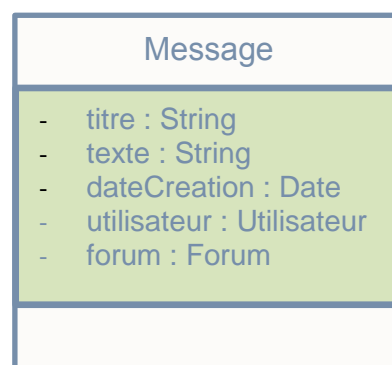
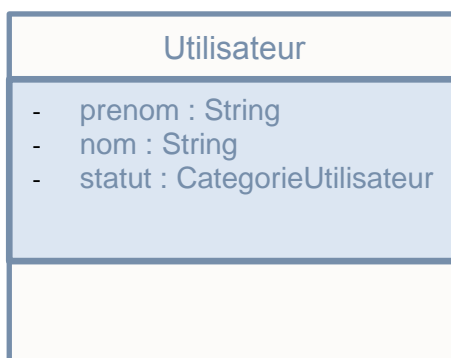
- Il s'agit de simuler, de façon très simpliste, un forum avec trois notions associées : le forum, l'utilisateur et le message.
- Un utilisateur écrit un message et un message est écrit par un utilisateur. En d'autres termes, il faut que l'on puisse savoir quel(s) message(s) a écrit un utilisateur et réciproquement : à partir d'un message, on doit pouvoir connaître son auteur.

Pour cela, vous construirez une classe *Utilisateur* avec ses variables d'instances caractérisant un utilisateur de forum : son nom , son prénom , son statut (modérateur ou simple utilisateur) .

Concernant la classe *Message*, elle sera caractérisée par un *titre* (de type *String*), un *texte* (de type *String*), une *date de création* (de type *Date*), et enfin son auteur (de type *Utilisateur*). Les messages ayant besoin du forum pour y être déposés, vous allez ajouter en tant que variable de classe, une instance de la classe *Forum*. Ainsi :

```
public static final Forum forum = new Forum ();
```

En utilisant **UML**, pour exprimer la représentation de ces deux classes, on aurait (la lecture de ce diagramme de classes étant assez intuitive) :



- ✓ **NB₁** : le - devant chaque variable d'instance traduit le mode de protection, soit *private*.
le + devant chaque variable d'instance traduit le mode de protection, soit *public*.

Ce qui est important dans ce TP est la mise en œuvre des références croisées entre une instance d'utilisateur et une instance de message.

Remarques :

Ce TP permettra également de vous initier à la notion de **type énuméré** qui permet de déclarer un type de données avec une liste finie de valeurs prises par des variables de ce type. Ici, vous allez mettre en œuvre le type *CategorieUtilisateur* dont les variables associées pourront prendre les valeurs MODERATEUR ou STANDARD.

Etapes à réaliser :

1. Créez le type énuméré *CategorieUtilisateur* dans le package *com.votre_prenom.types*.

```
public enum CategorieUtilisateur {  
    MODERATEUR,  
    STANDARD  
}
```

Le code ci-dessus crée un type de données en **énumérant** l'ensemble des valeurs possibles (deux) pour des variables déclarées selon ce type. Très pratique : les valeurs sont implicitement *static* et *public*.

2. Créez une classe *Utilisateur* dans le package dédié . Vous y associerez les variables d'instance suivantes : *prenom* (*String*), *nom* (*String*) et *statut* (*CategorieUtilisateur*).

On voit que, une fois défini, le type énuméré s'utilise comme un type usuel .

3. Mettez en œuvre les accesseurs (lecture et écriture) associés, constructeur, redéfinition de *toString()* de la façon suivante :

```
@Override  
public String toString() {  
    return "Utilisateur : " + prenom + " " + nom + ", " + statut ;  
}
```

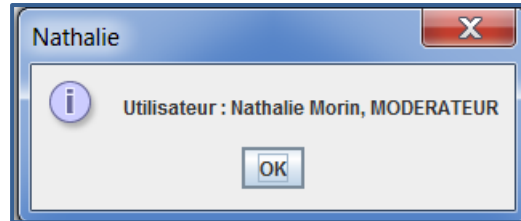
Cette redéfinition permet d'afficher pour un utilisateur ses nom, prénom et statut.

4. Dans *main*, instanciez un utilisateur modérateur et deux utilisateurs standard.

NB : Pour donner à une variable, une valeur issue d'un type énuméré, procédez de la façon suivante : `TypeEnumere.VALEUR` . Par exemple .

```
Utilisateur jean = new Utilisateur("Jean", "Dupont", CategorieUtilisateur.STANDARD);  
Utilisateur bernard = new Utilisateur("Bernard", "Morin", CategorieUtilisateur.STANDARD);  
Utilisateur nathalie = new Utilisateur("Nathalie", "Morin", CategorieUtilisateur.MODERATEUR);
```

Affichez, dans une boîte de dialogue les trois utilisateurs précédents. Par exemple, pour Nathalie :



5. Créez la classe *Message* avec les quatre variables d'instance décrites au-dessus : *String titre*, *String texte*, *Date dateCreation*, *Utilisateur utilisateur*.
6. Le type *Date* sera celui correspondant à la classe *java.util.Date*.
7. Mettez en œuvre le constructeur, qui n'admettra que trois arguments : *titre*, *texte* et *utilisateur* : la date de création étant directementinstanciée au sein de ce constructeur :

```
...  
dateCreation = new Date() ;  
.....
```

8. Redéfinissez *toString()* :

```
@Override  
public String toString() {  
    return titre + " -- " + texte + "\nDate de création : " + getDateCreation() + "\n" +  
        utilisateur ;  
}
```

NB : La séquence de caractères « *\n* » permet de générer un saut de ligne dans la chaîne.

9. Mettez en œuvre les accesseurs suivants : *getter* et *setter* pour *titre* et *texte* . Uniquement *getter* pour *utilisateur* et *dateCreation* (on n'a pas à les modifier).

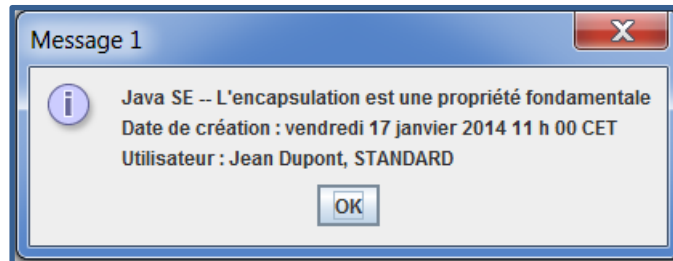
Concernant la variable d'instance *dateCreation*, codez le *getter* de la façon suivante afin que le retour ne soit pas une *Date* mais une chaîne (*String*), formatée selon le format de date français : Lundi 5 Mars 2014 , par exemple .

```
public String getDateCreation() {  
    DateFormat df = DateFormat.getDateInstance(DateFormat.FULL,DateFormat.FULL);  
    return df.format(dateCreation);  
}
```

10. Instanciez un message dont l'auteur est Jean. Par exemple :

```
Message message1 = new Message("Java SE", "L'encapsulation est une propriété fondamentale",  
jean);
```

11. Affichez-le en vous appuyant sur *toString()* redéfinie précédemment :



Nous souhaitons maintenant disposer d'une fonctionnalité consistant à **collectionner** les messages créés et leur auteur respectif. Nous allons pour cela ranger chaque message et son auteur sous forme de couple : (**message** , **auteur**) .

Cette collection sera initialement implémentée sous forme d'un tableau, où seront stockés ces couples, et dont la taille va être déterminée dans le code. Plus tard, vous remplacerez le tableau par une collection – plus appropriée - dont la taille est dynamique.

Dès qu'un message *m1* est créé (rappel : un message est créé avec son auteur, ici ce sera *utilisateur1*), le couple (*m1* , *utilisateur1*) doit être automatiquement rangé dans le tableau.

Quel est le type du tableau à déclarer pour collectionner ces couples ?

Il s'agit d'un tableau à une dimension d'éléments de type (*Message* , *Utilisateur*). Ce type n'existant pas, vous allez créer une classe nommée *MessageUtilisateur* à partir de laquelle vousinstancierez des objets à chaque création de message.

12. Créez cette classe *MessageUtilisateur* avec deux variables d'instance : *utilisateur(Utilisateur)* et *message(Message)*.

13. Créez le constructeur fixant ces variables d'instance.

14. Ajoutez les accesseurs en lecture pour ces deux variables d'instance.

15. A chaque message créé (dans le constructeur de *Message* donc), vous appellerez la méthode d'instance *ajouterUtilisateurMessages* sur le forum. Le forum sera donc instancié en tant que **variable de classe** dans la classe *Message*.

16. Tant que vous êtes dans la classe *Message* , rajoutez la méthode suivante :

```
public String getAuteur() {  
    return utilisateur.getPrenom() + " " + utilisateur.getNom() + ", " + utilisateur.getStatut();  
}
```

17. Développez maintenant la classe *Forum*. Prévoyez une constante entière *NBR_MESSAGES* pour dimensionner le tableau qui va gérer les couples (*message*, *utilisateur*) , une variable d'instance pour comptabiliser le nombre de messages actuellement déposés et enfin le tableau *messages* de type *MessageUtilisateur[]* et de taille *NBR_MESSAGES*.

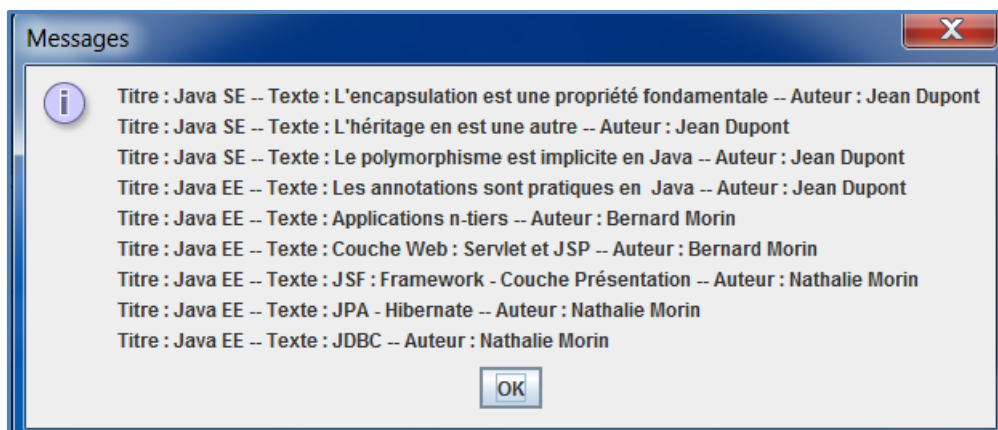
18. Instanciez le tableau *messages* dans le constructeur de *Forum*.

19. Codez la méthode *ajouterUtilisateurMessages* dont voici l'entête. Vousinstancierez un *MessageUtilisateur* et le déposerez dans le tableau.

```
public boolean ajouterUtilisateurMessages ( Utilisateur utilisateur , Message message) {...}
```

le booléen retourné indique si l'instance créée a pu être déposée dans le tableau. Dans le cas contraire, un message informera l'utilisateur .

20. Implémentez la méthode *getListeMessagesAuteur()* qui retournera une chaîne de caractères constituée des messages et des auteurs qui les ont rédigés . Cette *String* retournée sera exploitée dans une boîte dialogue pour produire l'affichage suivant :



Félicitations : vous avez réussi ce TP et l'ensemble des notions qu'il mettait en œuvre !!

Copyright

➤ **Chef de projet (responsable du produit de formation)**

PERRACHON Chantal, DIIP Neuilly-sur-Marne

➤ **Ont participé à la conception**

COULARD Michel, CFPA Evry Ris Orangis

➤ **Réalisation technique**

COULARD Michel, CFPA Evry Ris Orangis

➤ **Crédit photographique/illustration**

Sans objet

➤ **Reproduction interdite / Edition 2013**

AFPA Février 2014

Association nationale pour la Formation Professionnelle des Adultes

13 place du Général de Gaulle – 93108 Montreuil Cedex

www.afpa.fr