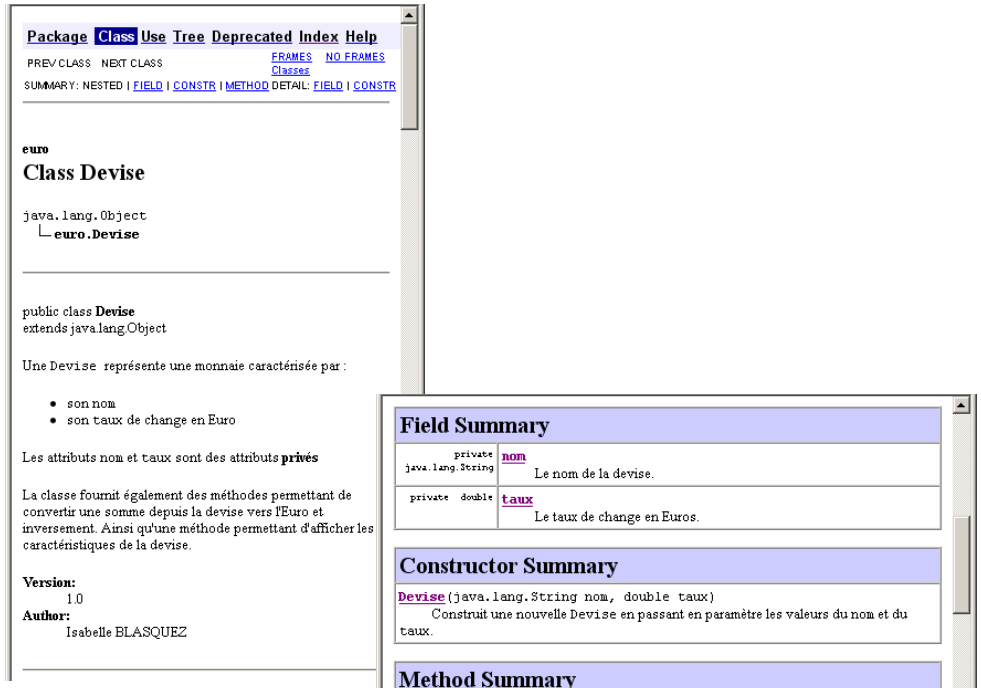
TD JAVA n°1: Introduction à Java, Classe et Objet

Exercice 1 : Premières classes JAVA

On souhaite écrire une application **`TestConversion`** permettant de convertir une **`Devise`** en Euros.
Définition d'une devise (Petit Larousse) : monnaie considérée par rapport aux monnaies d'autres pays par rapport à son taux de change.

1. Ecrire la classe **`Devise`** qui a permis de générer la **Javadoc** suivante
*(Pour le moment, on ne vous demande pas d'écrire les commentaires Javadoc, **seulement le code Java** ! ! !*
La description détaillée de la Javadoc est donnée en partie dans l'annexe I…)



Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR

euro

Class Devise

java.lang.Object
└ **euro.Devise**

public class **Devise**
extends java.lang.Object

Une Devise représente une monnaie caractérisée par :

- son nom
- son taux de change en Euro

Les attributs nom et taux sont des attributs **privés**

La classe fournit également des méthodes permettant de convertir une somme depuis la devise vers l'Euro et inversement. Ainsi qu'une méthode permettant d'afficher les caractéristiques de la devise.

Version: 1.0
Author: Isabelle BLASQUEZ

Field Summary

private java.lang.String	**nom** Le nom de la devise.
private double	**taux** Le taux de change en Euros.

Constructor Summary

Devise(java.lang.String nom, double taux)
Construit une nouvelle Devise en passant en paramètre les valeurs du nom et du taux.

Method Summary

double	**conversionDepuisEuro**(double somme) Convertit une somme en Euros en une somme de cette Devise.
double	**conversionEnEuro**(double somme) Convertit une somme de cette Devise en Euros.
java.lang.String	**getNom**() Retourne le nom de cette Devise sous forme de String.
double	**getTaux**() Retourne le taux de change de cette Devise sous forme de double.
void	**setNom**(java.lang.String nom) Mise à jour du nom de cette Devise.
void	**setTaux**(double taux) Mise à jour du taux de change de cette Devise.
java.lang.String	**toString**() Retourne un String qui contient les caractéristiques de cette Devise.

La description détaillée
se trouve
en annexe I

2. Rajouter à la classe **`Devise`** un constructeur à un seul paramètre, ce paramètre étant le `nom` de la devise. En appelant ce constructeur, le `taux` sera automatiquement fixé à 1.0.

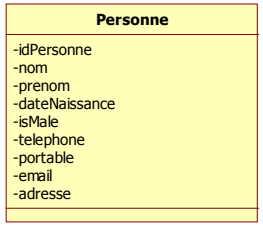
3. Afin de tester cette classe, nous allons écrire une classe **`TestConversion`** qui nous permettra d'obtenir le jeu d'essai ci-contre…

```
 --> Création d'une nouvelle devise
        Saisir le nom de la devise et son taux :
Franc 6.55957

 La devise en cours est désormais: Franc(taux de change en Euro =6.55957)

 --> Saisir la Somme en Euros à convertir
2.0
2.0 Euro(s) = 13.11914 Franc(s)
```

Exercice 2 : Fil Rouge Cabinet Médical
Définition de la classe `Personne`

Cette année, l'étude de cas UML vous propose de travailler sur un cabinet médical.
Le cabinet médical, est un cabinet de professionnels du monde médical, destiné à soigner les patients.
Le secrétariat souhaite avoir un outil de gestion des rendez-vous entre patients et professionnels.

Le secrétariat du cabinet médical gère avant tout des personnes (professionnel, patient).
Nous devons donc commencer par créer une classe `Personne` que nous définirons de la manière suivante :



Remarques :
`idPersonne` *sera mémorisée dans une variable de type* `Integer`

`dateNaissance` *sera mémorisée dans une variable de type* `Date` *du paquetage* `java.sql.Date`

`isMale` *sera un booléen représentant le sexe de la personne et devra être initialisé par défaut à vrai*

`adresse` *sera une variable de type* `Adresse`. *Considérer que la classe* `Adresse` *existe, elle vous sera donnée en TP. La javadoc est disponible en Annexe II.*

Pour l'instant, tous les autres champs seront mémorisés dans des `String`.

1. Implémenter la classe `Personne` en respectant la modélisation précédente. Cette classe devra contenir:
   - les **attributs** privés adéquats
   - les **constructeurs** suivants :
     - → Un constructeur ***sans*** argument pour respecter la norme Java Bean. Dans ce constructeur, on décide de n'écrire aucun code.
     - → Un constructeur qui permet de spécifier une valeur pour ***tous*** les attributs de la classe
     - → Un constructeur avec ***5 arguments*** (`nom, prenom, date de Naissance, isMale, adresse`) correspondant aux attributs significatifs qui doivent obligatoirement être renseignés.

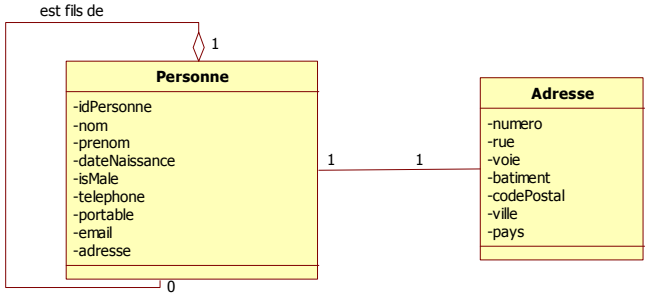
↳ les **méthodes suivantes :**

- → **Toutes** les méthodes « **getteur»** renvoyant les caractéristiques de cette `Personne`.
- → **Toutes** les méthodes « **setteur»** modifiant les caractéristiques de cette `Personne`.
- → Une méthode que vous appellerez **`toString`** et qui retourne un `String` contenant les caractéristiques d'une `Personne` comme le montre la copie d'écran ci-dessous.

```
Numéro: 1
Nom : DUPONT
Prenom : Julie
DateNaissance : 1960-05-21
isMale : false
Telephone :  0555434355
Portable : 0606060606
Email : julie.dupont@tralala.fr
Adresse :
        numéro: 15
        rue: avenue Jean Jaurès
        voie: null
        batiment: null
        codePostal: 87000
        ville: Limoges
        pays: France
```

En écrivant une telle classe, vous respecterez les règles d'écriture d'une classe donnée en annexe III.
Dorénavant, toutes vos classes devront respecter ces règles…

2. Les patients d'un cabinet médical viennent habituellement « en famille » : il peut donc être intéressant de savoir si une `Personne` *est fils d'* une autre `Personne`.



Pour modéliser cette ***ascendance potentielle***, il est nécessaire de modifier la représentation UML de la classe `Personne` précédente en mettant en évidence une **agrégation** « *est fils de* » de la classe `Personne` vers la classe `Personne.`

Il est à noter que pour l'instant, nous considérerons qu'une `Personne` ne peut avoir au plus qu'1 seul ascendant de type `Personne`.

→ Modifier la classe `Personne` écrite précédemment pour prendre en compte **`unAscendant.`**

Remarque : En ce qui concerne les constructeurs, on souhaite au final pouvoir disposer des 4 constructeurs suivants :

- → Un constructeur par défaut (***sans*** argument) pour respecter la norme Java Bean. Dans ce constructeur, on décide de n'écrire aucun code.
- → Un constructeur qui permet de spécifier une valeur pour ***tous*** les attributs de la classe
- → Un constructeur avec ***5 arguments significatifs*** sans ascendant (`nom, prenom, date de Naissance, isMale, adresse`)
- → Un constructeur avec ***6 arguments significatifs*** avec ascendant (`nom, prenom, date de Naissance, isMale, adresse, ascendant`)

Annexe I: Description détaillée de la javadoc de la classe Devise

Field Detail

nom

```
private java.lang.String nom
```

Le nom de la devise.

See Also:
getNom(), setNom(java.lang.String)

---

taux

```
private double taux
```

Le taux de change en Euros.

See Also:
getTaux(), setTaux(double)

Constructor Detail

Devise

```
public Devise(java.lang.String nom,
              double taux)
```

Construit une nouvelle Devise en passant en paramètre les valeurs du nom et du taux. Si le taux a une valeur inférieure ou égale à 0.0, la valeur de 1.0 sera automatiquement asssignée au taux.

Parameters:
nom - le nom de la devise.
taux - le taux de change par rapport à l'euro

Method Detail

getNom

```
public java.lang.String getNom()
```

Retourne le nom de cette Devise sous forme de String.

Returns:
le nom de cette devise

---

setNom

```
public void setNom(java.lang.String nom)
```

Mise à jour du nom de cette Devise.

Parameters:
nom - le nouveau nom de cette Devise

---

getTaux

```
public double getTaux()
```

Retourne le taux de change de cette Devise sous forme de double.

Returns:
le taux de change par rapport aux Euros

---

setTaux

```
public void setTaux(double taux)
```

Mise à jour du taux de change de cette Devise. Si le taux a une valeur inférieure ou égale à 0.0, la valeur de 1.0 sera automatiquement asssignée au taux.

Parameters:
taux - le nouveau taux de cette Devise

---

conversionEnEuro

```
public double conversionEnEuro(double somme)
```

Convertit une somme de cette Devise en Euros.

Parameters:
somme - la somme à convertir
Returns:
la somme en Euros obtenue après conversion

---

conversionDepuisEuro

```
public double conversionDepuisEuro(double somme)
```

Convertit une somme en Euros en une somme de cette Devise.

Parameters:
somme - la somme à convertir
Returns:
la somme obtenue après conversion

---

toString

```
public java.lang.String toString()
```

Retourne un String qui contient les caractéristiques de cette Devise.

Returns:
une chaine de caractères contenant les caractéristiques de cette Devise sous la forme :
nom (taux de change en Euro = taux)

Annexe II: Javadoc de la classe Adresse

com.iut.cabinet.metier

Class Adresse

```
java.lang.Object
  └ com.iut.cabinet.metier.Adresse
```

---

public class **Adresse**
extends java.lang.Object

Une `Adresse` représente une adresse postale. Elle est caractérisée par :

- un `numéro`
- un libellé de `rue`
- un libellé de `voie`
- un complément d'identification du point géographique comme un `b`
- un `code postal`
- une localité de destination (`ville`)
- un `pays`

Version:
1.0
Author:
Isabelle BLASQUEZ

Field Summary

private java.lang.String	**batiment** Le complément d'identification du point géographique.
private java.lang.String	**codePostal** Le code Postal.
private java.lang.String	**numero** Le numero
private java.lang.String	**pays** Le pays.
private java.lang.String	**rue** Le libellé de la rue.
private java.lang.String	**ville** La localité de destination
private java.lang.String	**voie** Le libellé de la voie.

Constructor Summary

Adresse() Construit une nouvelle `Adresse` en passant des valeurs par défaut pour tous les attributs
Adresse(java.lang.String numero, java.lang.String rue, java.lang.String voie, java.lang.String batiment, java.lang.String codePostal, java.lang.String ville, java.lang.String pays) Construit une nouvelle `Adresse` en passant une valeur spécifique pour tous les attributs

Method Summary

boolean	**equals**(java.lang.Object o) Teste si l'`Objet` spécifié est bien une `Adresse` et si cette dernière est égale à cette `Adresse` en comparant les valeurs des attributs suivants : `numero`, `rue`, `codePostal`, `ville` et `pays`.
java.lang.String	**getBatiment**() Retourne le complément d'identification du point géographique de cette `Adresse` sous forme de `String`.
java.lang.String	**getCodePostal**() Retourne le code postal cette `Adresse` sous forme de `String`.
java.lang.String	**getNumero**() Retourne le numero de cette `Adresse` sous forme de `String`.
java.lang.String	**getPays**() Retourne le pays de cette `Adresse` sous forme de `String`.
java.lang.String	**getRue**() Retourne le libellé de la rue de cette `Adresse` sous forme de `String`.
java.lang.String	**getVille**() Retourne la localité de destination de cette `Adresse` sous forme de `String`.
java.lang.String	**getVoie**() Retourne le libellé de la voie de cette `Adresse` sous forme de `String`.
void	**setBatiment**(java.lang.String batiment) Mise à jour du complément d'identification du point géographique cette `Adresse`.
void	**setCodePostal**(java.lang.String codePostal) Mise à jour du code postal cette `Adresse`.
void	**setNumero**(java.lang.String numero) Mise à jour du numéro de cette `Adresse`.
void	**setPays**(java.lang.String pays) Mise à jour du pays cette `Adresse`.
void	**setRue**(java.lang.String rue) Mise à jour du libellé de la rue de cette `Adresse`.
void	**setVille**(java.lang.String ville) Mise à jour de la localité de destination cette `Adresse`.
void	**setVoie**(java.lang.String voie) Mise à jour du libellé de la voie de cette `Adresse`.
java.lang.String	**toString**() Retourne un `String` qui contient les caractéristiques de cette `Adresse`.

Annexe III: Règles d'écriture d'une classe métier pour le projet Cabinet Médical

Toutes les classes du cabinet médical devront respecter les règles de conception suivantes :

➢ Règles de nommage :

Choisissez toujours des noms significatifs et non ambigus quant au concept qu'il désigne.
Pour plus de lisibilité, une majuscule sera utilisée pour chaque nouveau mot d'un nom composé : `dateNaissance`.

Type	Règle de nommage	Exemple
Classe	La première lettre doit être une majuscule. Éviter les acronymes, choisir des noms simples et descriptifs	`class Personne` `class RendezVous`
Interface	même règle que pour les classes	`interface Comparaison`

Méthodes	Les noms des méthodes doivent représenter une action.
	Choisir de préférence des verbes.
	Première lettre en minuscule.
		`String toString()`
		`String getNom()`
		`boolean verifierNumSecu()`

Variables Attributs	Première lettre en minuscule. Nom relativement court mais descriptif. Éviter les noms à une lettre sauf pour les compteurs internes et les variable temporaires	`String nom`
Constante	En majuscule, Le séparateur devient _	`int NBPERS_MAX = 100`

➢ Règles de conception :

1. Les classes devront être publiques

2. Les attributs seront privés pour respecter le principe d'encapsulation

3. Les classes devront possédaient ***au minimum*** 3 constructeurs :
   - → 2.1 Un constructeur par défaut (***sans*** argument) pour respecter la norme Java Bean. Dans ce constructeur, on décide de n'écrire aucun code. Les attributs de la classe auront alors leur valeur par défaut.
   - → 2.2 Un constructeur qui permet de spécifier une valeur pour ***tous*** les attributs de la classe
   - → 2.3 Un constructeur avec ***x arguments*** correspondant aux attributs ***significatifs*** (nécessaires) qui doivent ***obligatoirement être renseignés*** pour que l'objet ait un sens. Ces arguments constituent un choix de conception, il n'y aura jamais de `id` dans ces arguments car un `id` est un identifiant technique qui sera nécessaire au moment de l'enregistrement dans la base de données et non lors de la création de l'objet…

4. Les propriétés de la classe (variables d'instances/attributs) doivent respecter la norme suivante :
   - → être accessibles via des méthodes **`getXxx()`** où **`Xxx`** est le nom de la propriété : on parlera alors de **getteur**
   - → être éventuellement modifiables via des méthodes **`setXxx()`** où **`Xxx`** est le nom de la propriété : on parlera alors de **setteur**

5. Les classes devront ***redéfinir*** les méthodes suivantes :
   - → `String toString()` qui renverra dans un String le nom de chaque variable d'instance et sa valeur associée.
   - → `public boolean equals(Object o)` permettra de comparer 2 objets de la classe en testant la valeur de certaines variables d'instance : la « sélection » de ces variables correspond à des choix de conception ayant été réalisé durant la phase d'analyse.
   - → `public int hashCode()` méthode qui donne un ***code de hachage*** pour l 'objet.

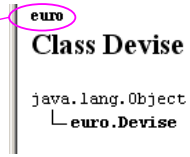
Remarque : les méthodes `hashCode` et `equals` pourront être générés automatiquement sous Eclipse..

6. La classe doit être "Serializable" pour pouvoir sauver et restaurer l'état d'instances de cette classe, c'est à dire implémenter l'interface `java.io.Serializable` (la notion de sérialisation fera l'objet d'un cours)

➢ Complément des Règles de conception au cours de l'avancée du projet (à compléter par vos soins) :

7. Correction TD JAVA n°1: Introduction à Java, Classe et Objet

Correction Exercice 1 : Premières classes JAVA



1. La classe Devise à partir de la javadoc.
Attention, la Javadoc nous indique que la classe se trouve dans le paquetage **euro**

```
package euro;

    public class Devise {

        ////////////////////
        // Attributs
        ////////////////////
        private String nom;
        private double taux;

        ////////////////////
        // Constructeur
        ////////////////////
        public Devise(String nom, double taux) {
           this.nom=nom;
           if (taux>0.0) this.taux = taux;
           else this.taux = 1.0;}

        ////////////////////////
        // Getters et Setters
        ////////////////////////
         public String getNom() {
                  return nom;}

        public void setNom(String nom) {
                  this.nom = nom;}

       public double getTaux() {
              return taux;}

       public void setTaux(double taux) {
          if (taux>0.0) this.taux = taux;
                else this.taux = 1.0;}

        ////////////////////////
        // Autres méthodes
        ////////////////////////
         public double conversionEnEuro(double somme){
          return somme/taux;  }

         public double conversionDepuisEuro(double somme){
          return somme*taux;  }

         public String toString(){
          return nom + "(taux de change en Euro ="+taux+")";
         }
    }
```


2. Rajouter à la classe `Devise` un constructeur à un paramètre.

On pourrait bien sûr écrire :

```
public Devise(String nom) {
            this.nom=nom;
            this.taux = 1.0;}
```

… mais je souhaite illustrer ici l'appel au constructeur préalablement créé avec `this`…

```
public Devise(String nom) {
          this(nom,1.0);
           }
```

4. **`TestConversion`** :

```
package euro; // Toujours dans la package euro ! ! ! sinon import…

import java.util.Scanner; // pour pouvoir utiliser Scanner pour la saisie

public class TestConversion {

      public static void main(String[] args) {
            String nom;
            double taux;
            double somme; // somme à convertir
            double resultat; // somme convertie

            ////////////////////////////////////////////////
            /// Instanciation d'une nouvelle Devise
            ////////////////////////////////////////////////
            // Saisie des caractéristiques de la nouvelle devise
            System.out.println(" --> Création d'une nouvelle devise ");
            System.out.println(" \t Saisir le nom de la devise et son taux : ");
            Scanner sc = new Scanner(System.in);//Nouvel objet Scanner
            nom= sc.next(); // Récupération de la chaîne de caractère

            String taux_str = sc.next();
            taux = Double.parseDouble(taux_str);

            //Instanciation de la devise
            Devise maDevise = new Devise(nom, taux);

            ////////////////////////////////////////////////////////
            /// Affichage des caractéristiques de la devise en cours
            ////////////////////////////////////////////////////////
            System.out.print("\n La devise en cours est désormais: ");
            System.out.println(maDevise); // utilisation de la méthode toString...

            ////////////////////////////////////////////////
            /// Conversion Euros vers Devise
            ////////////////////////////////////////////////
            System.out.println("\n --> Saisir la Somme en Euros à convertir");
            sc = new Scanner(System.in); // facultatif, mais…
            // pour une nouvelle saisie, mieux vaut instancier un nouveau scanner
            // cela permet de vider le flux …
            String somme_str = sc.next();
            somme = Double.parseDouble(somme_str);

            // appel méthode de conversion et affichage du résultat
            resultat=maDevise.conversionDepuisEuro(somme);

            // affichage résultat
            System.out.println(somme + " Euro(s) = "+resultat+" "+
maDevise.getNom()+"(s)");
      }
}
```

↳ Une classe exécutable est une classe qui contient une méthode spécifique (`main`) utilisée comme point de départ de l'exécution. (voir cours n°1)

↳ Les différentes étapes du programme :

→ 1. Saisie des données ⇒ on utilisera la classe `Scanner` : à noter 2 données sur une même ligne (voir cours n°1 : transparent *Interactivité avec l'utilisateur en mode console*)
→ 2. Instanciation d'une nouvelle `Devise`
→ 3. Affichage des caractéristiques de la devise instanciée ⇒ il serait bien d'utiliser `toString`…
→ 4. Nouvelle Saisie ⇒ attention, s'assurer que le flux est bien vide ! ! !
→ 5. Utilisation d'une méthode de conversion
→ 6. Affichage à l'aide d'un getteur ⇒ pas d'accès direct au champ (`private`).

↳ Remarques et variantes :

→ **1. Saisie d'une donnée de type `double`**

- Utilisation de la méthode `next`() et de la méthode statique `Double.parseDouble`

```
Scanner sc = new Scanner(System.in);
nom= sc.next();
String taux_str = sc.next();
taux = Double.parseDouble(taux_str);
```

Taper `12.5`

- Utilisation de la méthode `nextDouble`() directement :

```
taux= sc.nextDouble();
```

Taper `12,5`

Attention pour la saisie des nombres réels. En France, nous devons taper par exemple "`12,5`" au lieu de "`12.5`". Si vous désirez effectuer la saisie avec le point comme séparateur de la partie entière avec la partie décimale, vous devez changer de localité afin que cela soit considéré comme nombre réel écrit sous la forme US (USA). Utilisez pour cela la méthode `useLocale()` en spécifiant l'argument Locale.US (par défaut : Locale.FRENCH).

```
import java.util.Locale; //pour utiliser la Cte Locale.Us
sc.useLocale(Locale.US);
taux= sc.nextDouble();
```

`void useLocale(Locale localité)` permet de changer de localité. Lorsque nous utilisons la classe `Scanner` dans un système d'exploitation réglé en zone française, le paramètre localité est positionné par défaut à Locale.FRENCH. Du coup en France, les nombres réels s'expriment au moyen de la virgule. Si vous faites une saisie depuis le clavier, cela ne pose pas de problème, bien au contraire. Malgré tout, si vous désirez effectuer la saisie en considérant qu'il s'agit d'un double c'est-à-dire en respectant l'écriture américaine, vous devez changer de localité. Placez alors la constante Locale.US en argument de cette méthode.

→ **3. Affichage des caractéristiques de la devise instanciée**

- Utilisation de la méthode `toString`()

```
System.out.println(maDevise);
```

- Utilisation des getteurs ⇒ pas d'accès direct aux champs (`private`).

```
System.out.println(maDevise.getNom() + "(taux de change en Euro =" +
                                                maDevise.getTaux()+")";);
```

→ **4. Nouvelle Saisie ⇒ attention, s'assurer que le flux est bien vide ! ! !**

```
//Si on est sûr que le flux est vide, on peut directement  utiliser l'instruction
      String somme_str = sc.next();
//…mais si on utilise cette instruction sur la saisie suivante : Franc 6.55957 d
(c.a.d 3 mots saisis), et que l'on a lu pour le moment que deux mots, c'est le « d »
qui va être pris en compte et on ne pourra rien saisir…
mieux vaut donc réinstancier le scanner, si on n'est pas sûr de nous (si 2 mots OK, si
3 rique de PB… surtout si le 3ème n'est pas une double…)
//on peut dire que cela revient à vider le flux (scanf du C et fflush en quelque sorte…
sc = new Scanner(System.in);
```


Correction Exercice 2 : Fil Rouge Cabinet Médical PIC'OUZ
Définition de la classe `Personne`

1. **Première implémentation** de la classe Personne sans ascendant, adresse étant représentée par une variable de la classe `Adresse`.

A propos de `idPersonne` :

Cet `id` n'est donc pour le moment, pas l'`id` de la BD… D'ailleurs, on va commencer par enregistrer dans un fichier avant la Base de Données. Pour l'instant idPersonne sert à attribué un numéro à une personne, on aurait pu l'appeler numPersonne… On rajoutera les `id` dans la conception des classes au moment où l'on en aura besoin, c'est à dire lorsqu'on travaillera avec la Base de Données (TD/TP JDBC)…

Pourquoi un constructeur sans argument et « sans code »=> Les variables ont des valeurs d'initialisation par défaut…après le constucteur on pourra toujours utiliser des setteurs…

↳ Dans un fichier **`Personne.java`**

```
import java.sql.Date; // pour pourvoir utiliser le format Date

public class Personne {
        private Integer idPersonne; //identifiant
        private String nom;
        private String prenom;
        private Date dateNaissance;
        private boolean isMale = true; //Initialisation obligatoire (par défaut false)
        private String telephone;
        private String portable;
        private String email;
        private Adresse adresse;

        ////////////////////
        // Constructeurs
        ////////////////////
        public Personne ()
        {     }

        public Personne(Integer idPersonne, String nom, String prenom,Date,
                        dateNaissance, boolean isMale, String telephone,
                        String portable, String email, Adresse adresse)
        {     this.idPersonne=idPersonne;
              this.nom=nom;
              this.prenom=prenom;
              this.dateNaissance=dateNaissance;
              this.isMale= isMale;
              this.telephone=telephone;
              this.portable=portable;
              this.email=email;
              this.adresse=adresse;
        }

        public Personne (String nom, String prenom,Date dateNaissance,boolean isMale,
                        Adresse adresse)
        {
      this (null,nom,prenom,dateNaissance,isMale,null,null,null,adresse);
        }
```

Initialisation des attributs :
Normalement l'initialisation est faite par le système…
- `null` pour les objets
- `false` pour les booléens

on peut cela tout de même initialisé les attributs (par principe de précaution…)
… En tous cas, `isMale` doit obligatoirement être initialisé à `true` (car `false` par défaut…)

```
//////////////////////////
// Getteurs et Setteurs
//////////////////////////
/////////////
// idPersonne
public void setIdPersonne(Integer idPersonne)
{       this.idPersonne = idPersonne;    }

public Integer getIdPersonne()
{       return idPersonne;   }
/////////////
// Nom
public void setNom(String nom)
{       this.nom = nom;      }

public String getNom()
{       return nom;    }

/////////////
// Prenom
public void setPrenom(String prenom)
{       this.prenom = prenom;   }

public String getPrenom()
{       return prenom;       }

//////////////////////
// Date de Naissance
public void setDateNaissance(Date dateNaissance)
{       this.dateNaissance = dateNaissance;    }

public Date getDateNaissance()
{       return dateNaissance;      }

//////////////////////
// Sexe
public void setIsMale(boolean isMale)
{       this.isMale = isMale;      }

public boolean isIsMale()
{       return isMale;       }

//////////////////////
//Telephone
public void setTelephone(String telephone)
{       this.telephone = telephone;      }

public String getTelephone()
{       return telephone;    }

//////////////////////
//Portable
public void setPortable(String portable)
{       this.portable = portable;   }

public String getPortable()
{       return portable;     }

//////////////////////
//email
public void setEmail(String email)
{       this.email = email;   }

public String getEmail()
{       return email;   }
```



```
        //////////////////////
        //Adresse
        public void setAdresse(Adresse adresse)
        {       this.adresse = adresse;    }

        public Adresse getAdresse()
        {       return adresse;      }



        //////////////////////////
        // Autre(s) méthode(s)
        //////////////////////////
        public String toString()  {
                String s= "";
                s="Numéro: "+idPersonne+"\n"+
                "Nom : "+nom +"\n"+
                "Prenom : "+prenom +"\n"+
                "DateNaissance : "+dateNaissance +"\n"+
                "isMale : "+isMale +"\n"+
                "Telephone :  "+telephone +"\n"+
                "Portable : "+portable +"\n"+
                "Email : "+email +"\n"+
                "Adresse : \n "+adresse ;
                return s;
             }
}
```

Remarque affichage du format de la date :

Si on écrit :

- `"DateNaissance : "+ dateNaissance + "\n"` : on affichera la date dans la format anglais

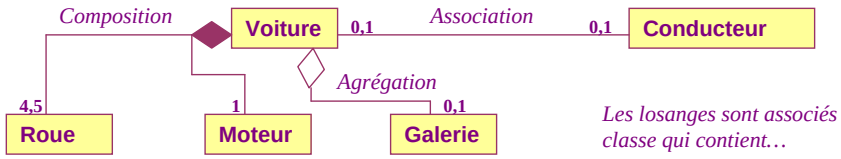
  `DateNaissance : 1991-01-25`

- `"DateNaissance : "+DateUtil.toString(dateNaissance)+"\n"` permettra d'afficher la date dans le format français en en utilisant la **méthode statique de la classe `DateUtil`** qui sera donnée en TP… :

  `DateNaissance : 25/01/1991`

2. Prise en compte de l'ascendance :

Retour sur la modélisation UML…



Les losanges sont associés à la classe qui contient…

- *Association :* les objets sont sémantiquement liée :
  Exemple : une voiture est conduite par un Conducteur…

- *Agrégation :* cycle de vie indépendant
  Exemple : une voiture et une galerie
  L'objet galerie n'envoie pas de message à l'objet voiture
  Un objet Galerie est transmis au moment de la construction de la voiture.

- *Composition:* cycle de vie identiques
  Exemple : une voiture possède une moteur qui dure la vie de la voiture

Association, Agrégation et Composition sont 3 relations d'association qui se différencient suivant leur ***degré de dureté d'association*** du plus « léger » pour l'association au plus « fort » pour la composition.
En ce qui concerne la composition , le degré le plus fort, on peut dire que l'objet composé ne peut pas exister sans celui qui le compose… Exemple : ligne de commandes ne peuvent pas exister sans une commande…

Prise en compte d'un ascendant… Pourquoi une telle modélisation ?

Les patients d'un cabinet médical viennent habituellement « en famille » : il peut donc être intéressant de savoir si une `Personne` *est fils d'* une autre `Personne`.

Pour modéliser cette ***ascendance potentielle***, il est nécessaire de modifier la représentation UML de la classe `Personne` précédente en mettant en évidence une **agrégation** « *est fils de* » de la classe `Personne` vers la classe `Personne`.

Il est à noter que pour l'instant, nous considérerons qu'une `Personne` peut avoir au plus qu'1 ascendant de type `Personne`.

Ici, une `Personne` et son `ascendant` ont une durée de vie indépendante
Un `ascendant` est associé par agrégation à une `Personne` car une `Personne` et son `ascendant` ont une durée de vie indépendante.

Nous allons modifier la classe `Personne` précédent pour prendre en compte un ascendant.

Pourquoi une telle modélisation ?

→ Plus tard pour le numéro de sécurité sociale…. S'il y a un ascendant, il faudra aller chercher son numéro de sécu…

2.1 Modifier la classe `Personne` écrite précédemment pour prendre en compte **`unAscendant`.**
Les constructeurs resteront inchangés.

On rajoute seulement un champ `unAscendant` avec ses setteur/getteur….
Puisqu'un ascendant est associé (par agrégation) à la classe Personne, il faut donc le rajouter comme attribut …

```
package com.iut.cabinet.metier;

import java.util.Date; // pour pourvoir utiliser le format Date

public class Personne {
       private String nom;
        private String prenom;
        private Date dateNaissance;
        private boolean isMale = true;
        private String telephone;
        private String portable;
        private String email;
        private String adresse;
        private Personne unAscendant;

        ////////////////////
        // Constructeurs
        ////////////////////
public Personne(Integer idPersonne,String nom,String prenom,Date dateNaissance,
                boolean isMale,String telephone, String portable,String email,
                Adresse adresse, Personne unAscendant)

{       this.idPersonne=idPersonne;
              this.nom=nom;
              this.prenom=prenom;
              this.dateNaissance=dateNaissance;
              this.isMale=isMale;
              this.telephone=telephone;
              this.portable=portable;
              this.email=email;
              this.adresse=adresse;
              this.unAscendant=unAscendant ;
        }

        public Personne(String nom,String prenom,Date dateNaissance,boolean isMale,
Adresse adresse,Personne unAscendant)
        {
      this (null,nom,prenom,dateNaissance,isMale,null,null,null,adresse, unAscendant);
        }

        ////////////////////////////
        // Getteurs et Setteurs
        ////////////////////////////

        ..........................................................................
        .....on rajoute juste les getteurs/setteurs pour le nouvel attribut .....
        ..........................................................................

        ///////////////////////
        //UnAscendant
        public void setUnAscendant(Personne unAscendant)
        {    this.unAscendant = unAscendant; }

        public Personne getUnAscendant()
        {  return unAscendant;}

        ////////////////////////////
        // Autre(s) méthode(s)
```

Dans le premier constructeur RAS
Dans le secons constructeur ascandant à NULL.

```
    ////////////////////////
    public String toString()  {
            String s= "";
            s="Nom : "+nom +"\n"+
            "Prenom : "+prenom +"\n"+
            "DateNaissance : "+dateNaissance +"\n"+
            "isMale : "+isMale +"\n"+
            "Telephone :  "+telephone +"\n"+
            "Portable : "+portable +"\n"+
            "Email : "+email +"\n"+
            "Adresse : "+adresse +"\n"+
            "Ascendant : "+unAscendant ;
            return s;
          }

}
```