TP JAVA n°3: Interface, Exceptions, Vector

Exercice 1 : Interface

Sous votre workspace **`TP_Java`** , créer un nouveau projet que vous appellerez **`TestInterface`** (**`File → New → Project`**…).

1. Implémenter **`l'interface Comparaison`** vue en TD
2. Implémenter la **`classe Ville`** qui implémente cette interface.
3. Tester votre code en instanciant deux villes.

… Retour au Cabinet Médical **…**

Exercice 2 : Cabinet Médical-Mise en place d'une exception métier : `CabinetMedicalException`

1. Implémenter dans le package **`com.iut.cabinet.metier`** la classe **`CabinetMedicalException`** vue en TD.
2. Modifier la méthode **`verifierNir`** de la classe **`PatientRegle`** pour qu'elle ***lève une exception*** de type **`CabinetMedicalException`** dans son code et la laisse remonter (cf question 3.1 du TD)
3. Ecrire dans le paquetage **`com.iut.cabinet.essai`** une application **`EssaiException`** qui permet de ***traiter l'exception*** **`CabinetMedicalException`** levée par la méthode `verifierNir` (cf question 3.2 du TD)
4. Rajouter dans ce programme test un appel à la méthode `printStackTrace` sur l'exception capturée et exécuter. Utiliser également la méthode `getMessage`.

Remarque : Lors de la création d'une exception, la Machine Virtuelle calcule le `StackTrace` (pile d'appels). Le `StackTrace` correspond aux fonctions empilées (dans la pile) lors de la création. Le calcul du `StackTrace` est couteux en temps d'exécution, c'est pourquoi on vous conseille d'utiliser la méthode `printStackTrace` dans les phases de debuggage, et de l'enlever dans la version définitive.

Exercice 3 : Cabinet Médical-Amélioration de la méthode : `verifierNIR`

1. Revenir sur la classe **`EssaiException`** et exécuter l'application avec le NIR suivant :
   `1Z8072B12345651`
   Que constatez-vous?
2. Pour rendre la méthode **`verifierNIR`** robuste, il est nécessaire d'attraper dans cette méthode *l'exception non contrôlée* qui pourrait éventuellement survenir (c-a-d celle que pourrait lancer la méthode `parseLong`).
   Nous souhaitons capturer cette exception, et relancer à la place une **`CabinetMedicalException`** avec un message approprié du genre :
   `"Le NIR proposé est incorrect :"` suivi bien sûr de la valeur de ce NIR incorrect.
   Ainsi la méthode **`verifierNIR`** **ne lancera que des** **`CabinetMedicalException`**.

   Modifier le code de la méthode `verifierNIR` dans ce sens.
3. Pour contrôler votre code, relancer l'application **`EssaiException`** , le message suivant devrait alors apparaître :

```
Le NIR proposé est incorrect : 1Z8072B12345651
```

Exercice 4 : Mise en place de l'exception métier `CabinetMedicalException` **dans la classe métier** `Patient` ***pour un NIR non VALIDE***

1. **Modification préalable des classes métier du projet :**

   ↳ Modifier **toutes les classes métier** du projet cabinet médical en tenant compte de la nouvelle règle d'écriture des classes métier vue en TD à savoir :

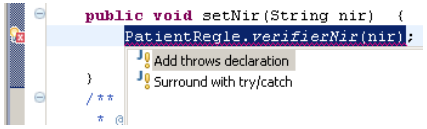
   Nouvelle règle d'écriture des classes métier

   Les constructeurs doivent utiliser des appels aux *setteurs* plutôt que des affectations directes du type `this.unAttribut=uneNouvelleValeur`

   ↳ Mettre à jour le document reprenant les règles d'écriture des classes métier du projet Cabinet Médical (si cela n'a pas été déjà fait…)

2. ↳2.1 **Modifier le setteur du `nir` : `setNir`** pour qu'il vérifie la validité de la clé de contrôle et ***propage éventuellement*** une `CabinetMedicalException`.

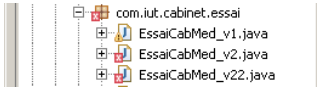
   Pour aller plus vite, vous pouvez utiliser l'aide d'Eclipse. En cliquant sur le symbole : 
   Vous obtenez un menu qui propose deux possibilités :
   - soit ***déclarer (lancer)*** l'exception : **`Add throws declaration`**
   - soit ***traiter (attraper)*** l'exception : **`Surround with try/catch`**

   Cliquez sur la première ligne qui permet de lancer l'exception. Le code correspondant ainsi que le tag javadoc **`@throws`** se rajoutent alors à votre programme.
   Il ne vous reste plus qu'à compléter les commentaires javadoc …

   ↳2.2 ***Modifier les constructeurs*** afin qu'ils déclarent (*propagent*) l'exception `CabinetMedicalException`
   Ne pas oublier de compléter la javadoc …

   ↳2.3 Sauver vos modifications…
   Que constatez-vous sur les fichiers **`EssaiCabMed_vx.java`** créés lors des TP précédents…

   ↳2.4 On modifiera uniquement que le fichier **`EssaiCabMed_v22.java`.**
   Revenir sur le fichier **`EssaiCabMed_v22.java`** et modifier le pour pouvoir le compiler et l'exécuter Correctement. Tester également avec un NIR erroné.

Exercice 5 : Mise en place d'une exception métier pour s'assurer de l'existence d'une `Adresse` ***pour un*** `Patient`

Précédemment, nous avons souhaité lever une `CabinetMedicalException` lors de la saisie d'un NIR invalide.
Dans cette partie, nous souhaitons également lever une exception pour interdire de créer un `Patient` qui n'a pas d'`Adresse` (c-a-d champ `Adresse` à `null`).

Attention, vous devez respecter ***la contrainte n°1*** suivante :
Interdire de créer un `Patient` sans adresse ne revient pas à interdire de créer une `Personne` sans adresse…. En effet, nous souhaitons que l'exception ne soit levée que dans le cas d'un `Patient`, c'est-à-dire :
- qu'un `Patient` doit ABSOLUMENT avoir une Adresse (champ `Adresse` à `null` non permis)

… mais …

- qu'un `Professionnel` peut ne pas avoir d'Adresse (champ `Adresse` à `null` permis)…

Avant de vous lancer dans le code, réfléchissez bien à la mise en place de cette contrainte (où et comment) et aux modifications que cela va entraîner dans votre code. Plusieurs solutions sont envisageables.

Quelle que soit la solution implémentée, vérifier avec des tests (jeux d'essais) si vous respectez bien la contrainte n°1 de départ…

Attention : Pour la prochaine séance …
→ L'exception métier `CabinetMedicalException` pour la détection d'un *NIR INVALIDE* doit absolument être mise en place dans le projet… et fonctionner ! ! !

→ Le traitement pour s'assurer de l'existence d'une `Adresse` avant de créer un `Patient` sera bien sûr pris en compte dans la notation finale, mais n'est pas indispensable à la suite du bon fonctionnement du projet …

Continuez sur votre lancée et contrôlez à l'aide d'expressions régulières les formats des autres attributs...

En effet, afin de rendre votre application encore plus robuste, vous devez également penser à contrôler les autres attributs. Par exemple :
- le `nom` et le `prenom` ne devront contenir que des lettres (pas de chiffres)
- le format du numéro de `telephone` devra posséder 10 chiffres et pourra être saisi suivant l'une des trois syntaxes suivantes : **`xx.xx.xx.xx.xx ou xx-xx-xx-xx-xx`** ou **`xxxxxxxxxx`**
- Un `email` devra comporter un @ et un `.` afin de respecter le format suivant: **xxx@xxx.xx**
- etc …

Ainsi de la même manière que vous avez contrôlé le `nir`, vous pouvez contrôler les autres attributs en complétant par exemple la classe `PatientRegle`, voire en créant une classe `PersonneRegle` et en lançant des `CabinetMedicalException` . Vous pouvez écrire vos tests avec de simple **`if`**, mais vous pouvez également utiliser des ***expressions régulières*** *(API* ***`regex`*** *qui propose entre autres les classes* ***`Pattern, Matcher`*** *: pour en savoir plus...effectuez une recherche sous* ***Google...****)*
Nous tenons à vous informer dès à présent que la notation du TP prendra bien évidemment en compte la mise en place du contrôle des formats... et l'utilisation d'expressions régulières !...

Correction TP JAVA n°3: Interfaces, Exceptions, Vector

Exercice 3 : Cabinet Médical-Amélioration de la méthode : `verifierNIR`

1. Revenir sur la classe **`EssaiException`** et exécuter l'application avec le NIR suivant :

`1Z8072B12345651`

Que constatez-vous?

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "1Z80720123456"
        at java.lang.NumberFormatException.forInputString(Unknown Source)
        at java.lang.Long.parseLong(Unknown Source)
        at java.lang.Long.parseLong(Unknown Source)
        at com.iut.cabinet.metier.PatientRegle.verifierNir(PatientRegle.java:58)
        at com.iut.cabinet.essai.EssaiException.main(EssaiException.java:25)
```

Une `java.lang.NumberFormatException` levée par la méthode `parseLong` apparaît et plante le programme.
Cette exception est une exception non contrôlée par le compilateur:
" ***non contrôlée***"⇒ le compilateur ne nous oblige pas à attraper (`try…catch`) ou relancer (`throws`) l'exception.

2. Pour rendre la méthode **`verifierNIR`** robuste, il est nécessaire d'attraper dans cette méthode toutes les exceptions non contrôlées qui pourraient apparaître

`code` ⇒ **`CabinetMedicalException`** levée à l'exercice précédent pour détecter l'invalidité du NIR (mauvaise saisie de chiffres par exemple)
`code` ⇒ **`CabinetMedicalException`** levée à cette exercice pour détecter l'invalidité du NIR (saisie d'une lettre à la place d'un chiffre soulève une exception non contrôlée lors de l'utilisation des méthodes **`parseXXX`**)

```
public static void verifierNir(String nirATester) throws CabinetMedicalException
        {
       ////////////////////////////////////////////////////
       // Premiere Etape :
       // Extraire les 13 premiers caractères et le cle
       ////////////////////////////////////////////////////
      if (nirATester == null) throw new CabinetMedicalException("Aucun NIR passé en
      paramètre (objet null)");

        if (nirATester.length() !=15) throw new CabinetMedicalException("Le NIR proposé
est incorrect - Un nir est composé de 15 chiffres "+ nirATester);

       String nir_13 = nirATester.substring(0,13); //13 car endIndex- 1  voir doc !!!

int cle=0; // déclaration avant le try
long nir=0;
      try // pour attraper entre autres les NumberFormatException
            // susceptibles d'être levées par les méthodes parseInt ou parseLong
      {
```



```
       cle = Integer.parseInt(nirATester.substring(13,15));

       ////////////////////////////////////////////////////
       // Deuxième Etape :
       // Conversion du String en long...
       ////////////////////////////////////////////////////
       // au préalable test pour savoir si on a une lettre ou pas
       // Corse 2A ou 2B => A ou B position 7 donc indice=6
       // ATTENTION, il faut bien des long
       // car les int sont codés sur 4 octets [4 -2 147 483 648 à 2 147 483 647]
       // => 10 chiffres : c'est trop court !!!… on a besoin de 13 chiffres…
       // long sont des entiers codés sur 8 octets [-9223372036854775808 à
9223372036854775807] // => ça passe ! ! !
switch  (nir_13.charAt(6))  // Attention il faut à tout prix les () autour de la variable
à tester...
            {
             case 'a' : nir_13=nir_13.replace('a','0');
                        nir = Long.parseLong(nir_13);
                        nir = nir - 1000000;
                        //System.out.println("\n \t \t ---- a détectée ... : ");
                        break;
             case 'A' : nir_13=nir_13.replace('A','0');
                        nir = Long.parseLong(nir_13);
                        nir = nir - 1000000;
                        //System.out.println("\n \t \t ---- A détectée ... : ");
                        break;
             case 'b' : nir_13=nir_13.replace('b','0');
                        nir = Long.parseLong(nir_13);
                        nir = nir - 2000000;
                        //System.out.println("\n \t \t ---- b détectée ... : ");
                        break;
             case 'B' : nir_13=nir_13.replace('B','0');
                        nir = Long.parseLong(nir_13);
                        nir = nir - 2000000;
                        //System.out.println("\n \t \t ---- B détectée ... : "+
nir);
                        break;
             default  : nir = Long.parseLong(nir_13);// on avait déjà un chiffre
                        break;                                          // juste la
conversion String  int...
            }
      }
      catch (NumberFormatException e)
                  //on peut être plus généraliste avec un catch (Exception e)
      {
            throw new CabinetMedicalException("Le NIR proposé est incorrect : "+
nirATester);
      }
       ////////////////////////////////////////////////////
       // Troisième Etape : Calcul de la clé ...
       ////////////////////////////////////////////////////
      long reste = nir%97; // reste de la division par 97
      long cleCalculee = 97-reste;                          // puis le complément à 97)
       ////////////////////////////////////////////////////////////////////
       //  Quatrième Etape : Validité de la clé
      //                     Vérification des clés : extraite et calculée...
      ////////////////////////////////////////////////////////////////////
       if (cleCalculee!=cle)
         { throw new CabinetMedicalException("Le NIR proposé est incorrect : "+
nirATester);
          }
        }
```

Exercice 4 : Mise en place de l'exception métier ***CabinetMedicalException*** *dans la classe métier* ***Patient*** pour un NIR non VALIDE

↳2.4 On modifiera uniquement que le fichier **EssaiCabMed_v22.java.**
Revenir sur le fichier **EssaiCabMed_v22.java** et modifier le pour pouvoir le compiler et l'exécuter Correctement. Tester également avec un NIR erroné.

→ Soit on propage avec des `throws`…

→ Soit on écrit du `try`…`catch`… mais attention, il faudra faire :

```
Patient1= null ;    et
if (patient1!= null)     maListe.ajouterPersonne(patient1);
```

… sinon on aura comme erreur de compilation :

```
the local variable Patient1 may not have been initialized…
```

```
Patient patient1=null;
            try {
                  patient1 = new Patient(1,"DUPONT","Julie",
                              DateUtil.toDate("21/05/1960",DateUtil.FRENCH_DEFAUT),
                              false,"0555434355","0606060606","julie.dupont@tralala.fr",
             new Adresse("15","avenue Jean Jaurès",null,null,"87000","Limoges","France"),
                              null,
                        "260058700112367","MARTIN Paul");
            } catch (CabinetMedicalException e) {
                  // Auto-generated catch block
                  e.printStackTrace();
            } // le

if (patient1!= null) maListe.ajouterPersonne(patient1);
```


Exercice 5 : Mise en place de l'exception métier `CabinetMedicalException` ***pour s'assurer de l'existence d'une*** `Adresse` ***pour un*** `Patient`

Précédemment, nous avons souhaité lever une `CabinetMedicalException` lors de la saisie d'un NIR invalide.Dans cette partie, nous souhaitons également lever une `CabinetMedicalException` pour interdire de saisir un patient qui n'a pas d'`Adresse` (c-a-d champ `Adresse` à `null`).

Nous souhaitons que l'exception ne soit levée que dans le cas d'un Patient… c'est-à-dire
- qu'un Patient doit ABSOLUMENT avoir une Adresse (champ `Adresse` à `null` non permis

… mais …
- un professionnel peut ne pas avoir d'Adresse (champ `Adresse` à `null` permis)…

`ATTENTION : Interdire de saisir un Patient sans adresse ne revient pas à interdire de saisir une Personne sans adresse`….

⇨ **REDEDINIR un setteur `setAdresse` dans la classe `Patient`** pour déclencher une exception uniquemment dans cette classe…

```
public void setAdresse(Adresse adresse) throws CabinetMedicalException {
          if (adresse==null)
           { throw new CabinetMedicalException("Aucune adresse n'est précisée");}
          else this.adresse = adresse;
          }
```

c'est bien des Modification(s) dans la classe `Patient` et non dans la classe `Personne`

Pourquoi cette redéfinition est-elle nécessaire ?
On a posé comme hypothèse qu'un `Professionnel` pouvait avoir une adresse à `null` or lorsqu'on créée un `Professionnel`, quelque soit le constructeur appelé, il apppelle le `setAdresse` de `Personne`…
Et si ce `setAdresse` est modifié, il sera aussi bien valable pour `Patient` et `Professionnel`, ce qui veut dire que `Patient` et `Professionnel` auraient le même comportement, et ce n'est pas le cas…..

⇨ **Conséquences de la Redéfinition de `setAdresse` dans la classe `Patient` :**
Il faut redéfinir des constructeurs pour `Patient` qui appelle `setAdresse` de la classe `Patient`

➢ MODIFICATIONS dans la classe `Patient` …

```
public class Patient extends Personne{

      ////////////////////////////
      /// Nouveaux attributs
      private String nir;
      private String medecinTraitant;

        ////////////////////
        // Constructeurs
        ////////////////////
        public Patient()
```

```
    {    }


    // Constructeurs avec tous les arguments
    // arguments de la Classe Personne
    // et les 2 attributs de Patient (nir et medecinTraitant)
    public Patient (Integer idPersonne,String nom,String prenom,
        Date dateNaissance, boolean isMale,String telephone, String portable,
        String email, Adresse adresse,Personne unAscendant, String nir, String
medecinTraitant) throws CabinetMedicalException
    {
         super(idPersonne,nom,prenom,dateNaissance,isMale,telephone,
                 portable,email,adresse,unAscendant);
         setNir(nir);
         setMedecinTraitant(medecinTraitant);
         setAdresse(this.adresse);
// il faut refaire un set Adresse pour éventuellement lever l'exception
// j'appelle en fait  ici le setAdresse redéfini dans cette classe, la classe
Patient ...
    }

   // constructeur avec 6 arguments significatifs sans ascendant
   // 5 arguments venant de la classe Personne :nom, prenom, date de
Naissance, isMale, adresse,
   // 1 argument de la classe Patient : nir
    public Patient (String nom,String prenom,Date dateNaissance,boolean
isMale, Adresse adresse, String nir)
                        throws CabinetMedicalException
    {
         super (nom,prenom,dateNaissance,isMale,adresse);
         setNir(nir);
         setAdresse(this.adresse);
    }

    // constructeur avec 7 arguments significatifs AVEC ascendant
    // 6 arguments venant de la classe Personne :nom, prenom, date de
Naissance, isMale, adresse, ascendant
    // 1 argument de la classe Patient : nir
    public Patient (String nom,String prenom,Date dateNaissance,boolean
isMale, Adresse adresse,Personne unAscendant,String nir)
                    throws CabinetMedicalException
    {
         super (nom,prenom,dateNaissance,isMale,adresse,unAscendant);
         setNir(nir);
        setAdresse(this.adresse);
    }
```



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

    //…… en plus des getteurs/et setteurs du nir et de medecinTraitant

    ////////////////////////////////////////////
    /// REDEFINITION du setAdresse
    /// pour pouvoir déclencher l'exception ...
    public void setAdresse (Adresse adresse) throws CabinetMedicalException
    {
   if (adresse==null)
                throw new CabinetMedicalException("Un patient doit forcément
            avoir une adresse !!!");

   this.adresse=adresse;
    }


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

  //…… toString, equals et hascode ……..

   }
```

➢ **MODIFICATIONS dans la classe Personne …**

```
public class Personne implements Serializable{

   ////////////////////////
   /// Attributs
   ////////////////////////

   /// … vos declarations en protected …

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

    public Personne(Integer idPersonne,String nom,String prenom,Date
dateNaissance, boolean isMale,String telephone, String portable,  String email,
Adresse adresse,Personne unAscendant) throws CabinetMedicalException
    {     /// … votre code     }

    public Personne(String nom,String prenom,Date dateNaissance,boolean
isMale, Adresse adresse,Personne unAscendant) throws CabinetMedicalException
    {     /// … votre code     }

    public Personne(String nom,String prenom,Date dateNaissance,boolean
isMale, Adresse adresse) throws CabinetMedicalException
      {   /// … votre code     }
```

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

        // idPersonne, Nom, Prenom, Date de Naissance, Sexe, Telephone, Portable
        // email unAscendant… inchangés

        ///////////////////
        //Adresse :
        // on change la déclaration pour être compatible avec la classe Patient
        public void setAdresse(Adresse adresse) throws CabinetMedicalException
        {
              this.adresse = adresse;
        }

        public Adresse getAdresse()
        {      return adresse;    }

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

      //…… toString, equals et hascode ……..
     //…… getAge

}
```

→ Remarque : normalement il faut déclarer (propager) l'exception **CabinetMedicalException** dans les ***constructeurs*** de la classe `Personne, mais ces` **`throws CabinetMedicalException`**) étaient déjà présents à cause du Nir ….
La seule modification dans la classe Personne est donc la déclaration d'une `CabinetMedicalException` sur le `setAdresse`

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


public Personne(String nom,String prenom,Date dateNaissance,boolean isMale,
Adresse adresse,Personne unAscendant) throws CabinetMedicalException


public Personne(String nom,String prenom,Date dateNaissance,boolean isMale,
Adresse adresse) throws CabinetMedicalException
```


⇨ **Comment reconnaître une `CabinetMedicalException` due à un NIR invalide ou à une Adresse nulle pour personaliser message de l'utilisateur…**
- ⇨ solution n°1 : chercher dans le message, le mot NIR ou le mot Adresse
- ⇨ solution n°2 : rajouter un champs code dans la classe **`CabinetMedicalException`** et affecter une valeur différente à ce champs suivant que l'on déclenche pour un NIR ou pour une Adresse
- ⇨ solution n°3 : sous-classer la **`CabinetMedicalException`** en **`CabinetMedicalNIRException`** et **`CabinetMedicalAdresseException`**

…. A vous de voir et de choisir la solution qui vous convient ….
Quelle que soit la solution choisie, il faut toujours vérifier avec des tests (jeux d'essais) …

↳ Autre solution pour s'assurer que seul un `Patient` n'a pas une adresse nulle…
On peut effectuer un test **directement dans le `setAdresse` de `Personne`** en testant la classe effective de la `Personne` grâce à un **`instance of`** …

```
public abstract class Personne {

// …

      /**
       * fonction qui permet d'attribuer une nouvelle adresse a une personne
       * @param adresse adresse de la personne
       * @throws CabinetMedicalException
       */
      public void setAdresse(Adresse adresse) throws CabinetMedicalException
      {
        if(this instanceof Patient && adresse==null)
            {
            throw new CabinetMedicalException ("Vous devez saisir une adresse pour
      un patient!!");
            }
            else
            {
            this.adresse=adresse;
            }
      }

// …

}// fin classe Personne
```