

TP JAVA n°1: Prise en main d'Eclipse



Dorénavant, vous utiliserez l'environnement **Eclipse**.

Eclipse est un IDE Open Source (IDE : Environnement de Développement Intégré (EDI))

Exercice 1 : Tester l'environnement - Premier programme

Pour installer sur votre portable votre environnement de développement Java, vous devez :

→ d'abord installer le **JDK** (<http://www.oracle.com/technetwork/java/javase/downloads/index.html>)

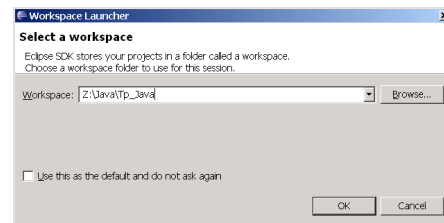
→ puis télécharger une version d'**Eclipse** (<http://www.eclipse.org/downloads/>) comme **Eclipse Classic**.

Pour installer Eclipse, il suffit de le dézipper.

Pour lancer Eclipse, cliquez sur l'icône Eclipse disponible sur votre bureau.

Si jamais Eclipse n'apparaissait pas sur votre bureau à l'IUT (voir page 16)

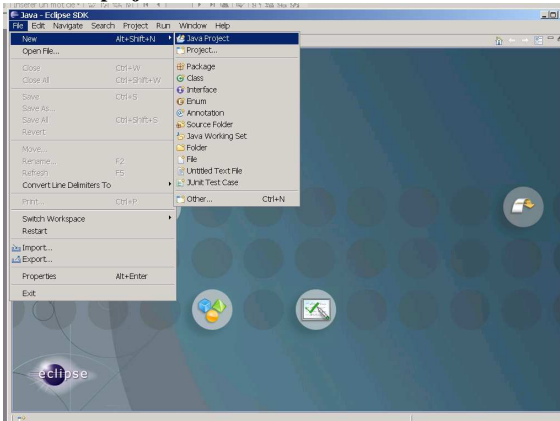
Au premier lancement, Eclipse vous demande de renseigner le chemin de votre dossier de travail (*Workspace*) où seront rangés par défaut les projets. Taper le chemin **Z:\Java\TP_Java**



Une page d'accueil présentant les fonctionnalités d'Eclipse est alors affichée.

► Création d'un projet :

Pour créer un **projet**, choisissez **File → New → Java Project**



L'assistant *New Java Project* permet de choisir le nom du projet, le dossier dans lequel il sera enregistré, la version du JDK avec lequel il est compatible ainsi que les sous-dossiers où seront rangés les fichiers source `.java` et les fichiers `.class`.

Pour cette première approche, on remplit juste le **nom** du projet : **TP1** et on vérifie que l'option **Create new project in workspace** est bien cochée.

Laissez les autres options cochées.

Cliquez sur **Next**.

L'assistant permet alors de sélectionner les sous-projets et les bibliothèques nécessaires au projet. Ne rien changer et cliquez sur **Finish**.

Isabelle BLASQUEZ - Dpt Informatique S3 – TP 1 : Prise en main d'Eclipse

Si nécessaire, **fermer l'onglet Welcome** pour continuer ...

► Création d'une classe :

Pour créer une **classe**, choisissez **File → New → Class**

(Remarque : Si l'élément **Class** n'apparaît pas, sélectionnez **Other...** puis **Class** dans la liste qui s'affiche)

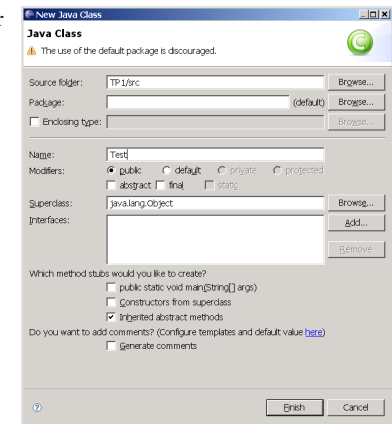
L'assistant *New Java Class* permet de renseigner l'identificateur de la nouvelle classe, son paquetage, sa super-classe et diverses options comme l'ajout d'une méthode `main`, l'implémentation automatique des méthodes abstraites, ...

Pour cette première approche, on remplit juste le nom de la classe : **Test** (pas besoin de l'extension `.java`).

On peut éventuellement cocher l'option **public static void main (String[] args)**

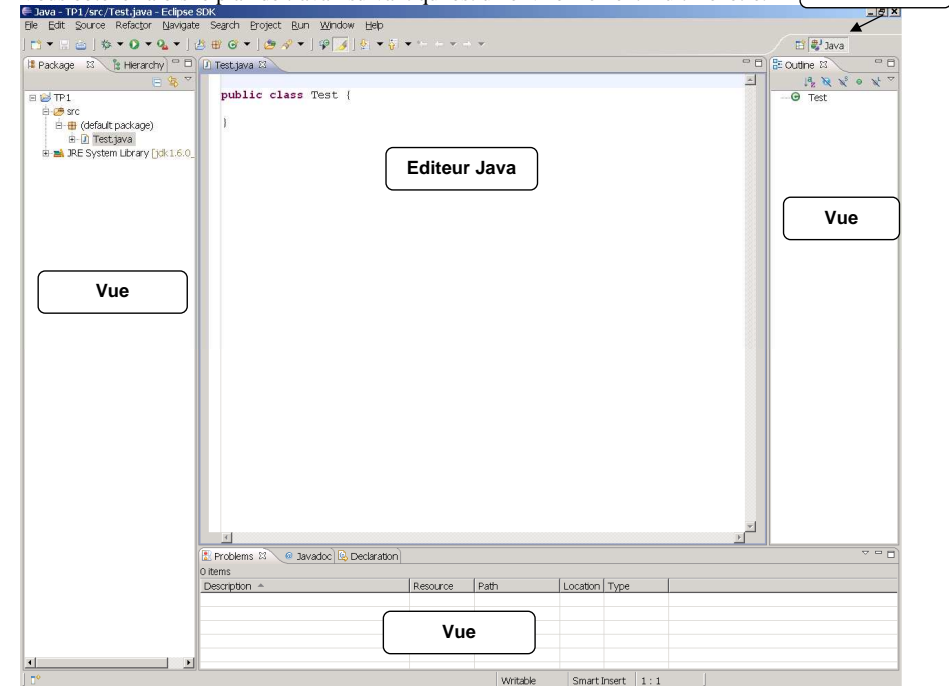
Cliquez sur **Finish**.

Si Eclipse vous propose de passer en *perspective Java*, répondez par l'affirmative pour voir apparaître votre nouvelle classe dans la liste des classes à l'écran.



► Plan de travail Eclipse : Vues, Editeur et Perspectives :

Vous obtenez alors le plan de travail suivant qui est un environnement multi-fenêtre.



Isabelle BLASQUEZ - Dpt Informatique S3 – TP 1 : Prise en main d'Eclipse

Au centre du plan de travail, on trouve l'**éditeur Java** qui contient le fichier `Test.java`.

Les autres fenêtres représentent le(s) projet(s) selon un certain point de vue (Package Explorer, Outline, ...). Une telle fenêtre est appelée **une «Vue»**.

La partie gauche du plan de travail présente la vue **Package Explorer** et la vue **Hierarchy**. La vue **Package Explorer** permet d'avoir une vision d'ensemble du paquetage développé et de naviguer dans les différents projets Java en cours : ainsi il est possible d'accéder rapidement au fichier de code. La vue **Hierarchy** permet d'examiner la hiérarchie des types (on en reparlera plus tard).

La partie droite du plan de travail présente la vue **Outline** qui propose une vision hiérarchique et structurée du contenu du fichier ouvert dans la fenêtre d'édition : cette vue permettra d'accéder directement aux différents éléments codés dans le fichier.

La partie inférieure du plan de travail présente les vues **Problems**, **Javadoc**, **Declaration**. C'est également dans cette partie qu'apparaîtra la vue **Console** qui montrera le résultat de l'exécution du programme sur la console de sortie.

Afin de gérer au mieux l'espace visuel à l'écran, Eclipse propose un système d'onglets permettant de basculer d'une vue à l'autre. Il est possible de fermer une vue en cliquant sur la croix associée à la vue.

Le **choix d'une vue** s'effectue de la manière suivante :

Window → Show View → Others...

En plus du concept de vue, Eclipse introduit également un concept de **«Perspective»**. En général, le programmeur ne décide pas lui-même des vues et des éditeurs associés à un projet ; le choix de ceux-ci est conditionné dans le cadre de *perspectives* qui sont des groupes de choix liés à un type de développement.

La copie d'écran de la page précédente (qui doit correspondre à votre plan de travail à l'écran) correspond à la **perspective Java**. Elle est choisie *automatiquement* par Eclipse dès que le programmeur crée un projet Java et permet de visualiser les packages et les classes du projet et leur contenu.

Si le programmeur débogue un programme Java, c'est la perspective **Debug** qui devra être choisie par Eclipse.

Eclipse propose différentes perspectives sur un même projet. Le choix d'une perspective s'effectue de la manière suivante : **Window → Open Perspectives**

En choisissant **Others...** on obtient la liste des perspectives possibles.

Choisir par exemple, la perspective **Resource** et cliquez sur **OK**.

La **perspective Resource** permet de visualiser les fichiers du projet avec une vue **Navigator**.

Il est possible de passer d'une perspective à l'autre en cliquant sur les onglets **Resource** et **Java** en haut à droite du plan de travail.



► Travail à réaliser :

Ecrire dans le fichier **Test.java** une application réalisant simplement l'affichage du message : "Ca marche !!!"

► Compilation et exécution :

☞ Enregistrer votre fichier **Test.java**.

Un fichier Java est **compilé automatiquement** au moment où vous l'enregistrez si l'option **Build Automatically** du menu **Project** est bien cochée. Vérifiez si c'est bien le cas sous votre Eclipse...

N'oubliez pas d'enregistrer fréquemment vos fichiers !!!

Remarque : Vous pouvez compiler *manuellement* si **Build Automatically** n'est pas coché : utilisez alors les éléments **Build** du menu **Project**.

☞ Pour **exécuter une application** : A partir de la barre des menus :

Run → Run as → Java Application

Dans le cas d'une application, le programme s'exécute dans la console qui apparaît dans la partie inférieure du plan de travail en tant que vue et cela quelque soit votre perspective de travail ...



Vous pouvez aussi vous placer dans le fichier **.java**, cliquer sur le bouton droit de la souris et sélectionner directement : **Run as → Java Application**

☞ Remarque : Sous Eclipse, vous pouvez utiliser le **débogger** à partir de

Run → Debug as → Java Application

Exercice 2 : Argument passé en ligne de commande.

Restez dans le même projet et créez une nouvelle classe **TestArg** afin d'implémenter et de tester le programme `TestArg` du transparent n°38 du cours n°1 « Introduction au langage JAVA ».


Ce code nécessite la **lecture d'arguments en ligne de commande**.

Pour passer un argument en ligne de commande, il faut modifier la configuration d'exécution du projet.

Pour exécuter l'application, au lieu de **Run → Run as**, choisir :

Run → Run Configurations ...

Ceci ouvre un boîte de dialogue qui permet entre autres de saisir les arguments du programme en

sélectionnant l'onglet suivant :  Arguments

Entrer la(les) valeur(s) à convertir dans la zone **Program Arguments**

Une fois, la(les) valeur(s) souhaitée(s) entrée(s), cliquez sur **Run**

Exercice 3 : Premières classes JAVA

1. Toujours dans le workspace TP_Java, créez un **nouveau Projet**, que vous appellerez : **TP1_Devis**

File → New → Project... → Java Project

2. Création du paquetage : euro

Dans la vue **Package Explorer** (partie gauche du plan de travail), dépliez si nécessaire le projet **TP1_Devis** et repliez le projet **TP1**.

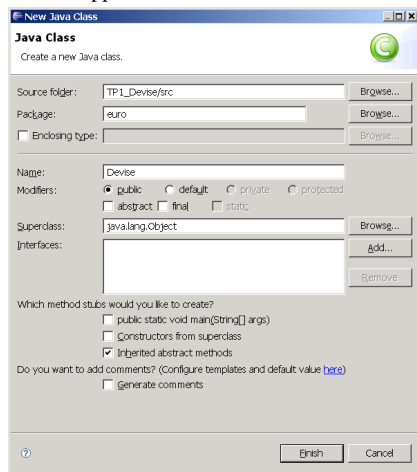
➤ Pour créer, le **paquetage euro** placez-vous sur **src**, cliquez sur le **bouton droit** de la souris, choisissez **New** puis **Package**.

Dans l'assistant « **New Java Package** », vérifiez que le Source folder soit bien : **TP1_Devis/src**

et compléter le Name du paquetage par : **euro**

Cliquez sur **Finish**.

euro apparaît dans l'arborescence du Package Explorer



A partir de ce paquetage, créez une nouvelle classe **Devis**.

Dans l'assistant « **New Java Class** », vérifiez que le champs Package soit bien à **euro**.

De même créez dans le même paquetage une classe **TestConversion**.

Implémentez les classes **Devis** et **TestConversion** en vous aidant du TD n°1 « Introduction au langage JAVA ».

Et testez avec le jeu d'essai suivant :

```
--> 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)
```

3. Création d'une documentation Javadoc

L'énoncé du TD vous donnait la **Javadoc** de la classe **Devis**.

En vous aidant de l'annexe **Javadoc** distribuée en cours et des conventions d'écriture de Sun (<http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html>), rajouter dans le fichier

Devis.java des commentaires javadoc pour :

- la présentation de la classe
- l'attribut **nom**
- le constructeur
- la méthode **ConversionEnEuro**

Pour le contenu des commentaires, voir l'énoncé du TD n°1 : et aidez-vous de l'annexe du cours n°1...

Le fichier **Devis.java** contient désormais des commentaires JavaDoc, il est donc temps d'apprendre à générer les documents Javadoc, qui constituent la documentation standard des programmes Java...

Pour générer des commentaires Javadoc à partir d'un fichier code, il faut choisir :

Project → Generate Javadoc...

qui ouvre la boîte de dialogue ci-contre « **Javadoc Generation** ». Celle-ci permet de sélectionner le projet à partir duquel on veut générer un document Javadoc.

Si vous utilisez votre portable, vous aurez sûrement besoin de configurer la ligne Javadoc Command.

Dans ce cas, cliquez sur le bouton

Configure... et allez chercher **javadoc.exe** dans votre installation Java.

Vérifiez que **TP1_Devis** est bien coché.

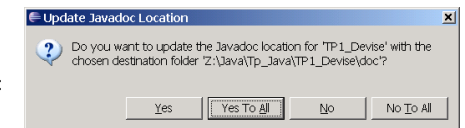
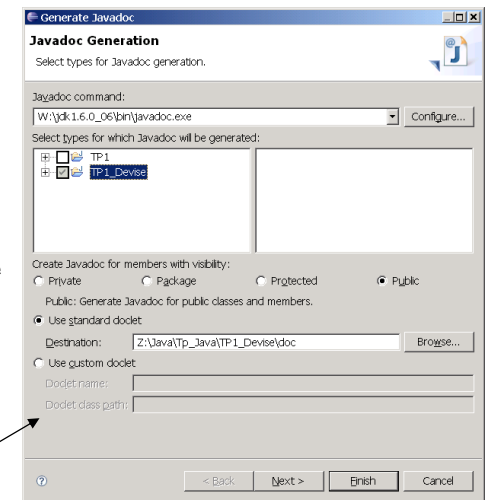
Vérifiez que la **Destination** est bien remplie avec : **Z:\Java\Tp_Java\TP1_Devis\doc**

Attention sous Eclipse, **par défaut seuls les champs public** apparaissent dans la javadoc...

... lors de la génération ne pas oublier de sélectionner **Private** si vous voulez générer la javadoc pour des champs privés. En cochant **Private**, vous générerez alors la javadoc pour tous les champs (niveau **Private** et au-dessus...)

Cliquez sur **Next**, puis sur **Finish**.

Puis validez par **Yes To All** le message suivant :



Ainsi un répertoire **doc** est créé dans votre projet et ce répertoire contient désormais la documentation Javadoc.

Pour visualiser les documents Javadoc associés à un projet, choisir sous Eclipse:

Navigate → **Open External Javadoc**

Remarque : On aurait également pu lancer manuellement la documentation Javadoc en sélection le fichier **index.html** du répertoire **doc** de votre projet.

Afin de tester rapidement, le fonctionnement de la Javadoc, nous avons limité nos commentaires à :

- la présentation de la classe
- l'attribut `nom`
- le constructeur
- la méthode `ConversionEnEuro`

Il est bien évident que tous les attributs, tous les constructeurs, toutes les méthodes (getteurs/setteurs) d'une classe doivent posséder des commentaires Javadoc afin de fournir la documentation correspondante.

Si vous avez du temps à la fin du TP, vous reviendrez compléter la javadoc de la classe `Devise`.

... et dorénavant, pour chaque nouvelle classe écrite, vous devrez écrire les commentaires Javadoc et générer la documentation correspondante...

*La notation du projet **Cabinet Médical** prendra bien sûr en compte la présence et la pertinence de la Javadoc ...*

Exercice 3 : Fil Rouge Cabinet Médical : Préparation du projet

*Implémentation et test la classe **Personne** vue en TD*

Préliminaire : Cette page consiste juste en une *présentation de l'arborescence* qui sera obtenue à la fin de la partie « 3.Création des paquetages : **application,essai,metier,presentation,user et util** »

CabinetMedical étant un gros projet sur plusieurs séances, il est nécessaire d'organiser notre travail. Pour ce projet, nous souhaitons travailler avec la **hiérarchie suivante**.

L'arborescence proposée est similaire à aux arborescences pouvant être réellement utilisées dans les entreprises. Elle permet de respecter le découpage en couches logicielles, découpage préconisé dans les applications informatiques d'aujourd'hui.

Ainsi, on retrouvera les classes de stéréotype **UML entity** dans le package métier `com.iut.cabinet.metier`, celles de stéréotype **UML boundary** dans le paquetage `com.iut.cabinet.presentation` et elles de stéréotype **UML control** dans le paquetage `com.iut.cabinet.application`.

Le nom des paquetages est purement conventionnel, cependant dans les projets Java EE certaines notations reviennent souvent.

Explications de notre arborescence:

cabinetMedical : nom du projet sur lequel nous travaillons

bin : les fichiers `.class` se trouvent dans ce répertoire.

doc : la documentation HTML générée est située dans ce répertoire

src : les sources du projet (fichier `.java`)

com :

iut : nom de la formation qui développe le projet

cabinet : nom du projet

application : pour les contrôleurs de Use Case

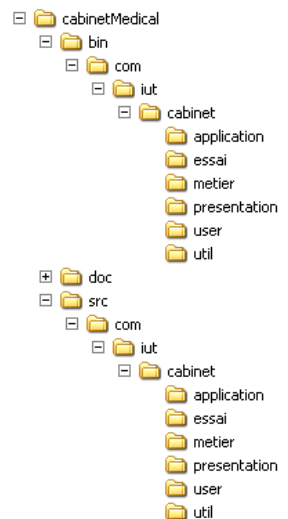
essai : pour les programmes qui vont nous permettre de réaliser des jeux d'essais personnels sur les classes...

metier : pour l'«interface» métier (classes métier + exceptions métier)

présentation : pour l'«interface» présentation (1 sous répertoire par use case)

user : pour les exceptions génériques et le DTO

util : pour les utilitaires tels qu'une classe permettant de manipuler les dates...



Au travers de cette arborescence, on souhaite que :

- tous les fichiers de code (`.java`) associés au projet soient stockés dans le répertoire **src**
- tous les fichiers de code binaire engendré (`.class`) soient stockés dans le répertoire **bin** (appelé parfois **build**)
- tous les documents constituant la documentation soient stockés dans le répertoire **doc**

1. Toujours dans le workspace TP_Java, créez un **nouveau Projet**, que vous appellerez : **cabinetMedical**

File → New → Project... → Java Project

Avant de cliquer sur **Next**, vérifiez que l'option **Create separate source and output folders** soit bien sélectionnée.

2. Vérifiez que sous l'onglet **Source** du panneau « **Java Settings** » de la boîte de dialogue « **New Java Project** » le répertoire **src** apparaît sous **cabinet**.

Vérifiez également que **cabinetMedical/bin** apparaît bien dans l'option **Default output folder**

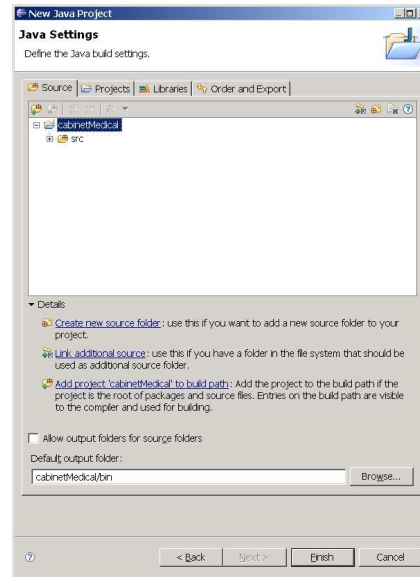
Cliquez sur **Finish**.

En configurant ainsi un projet, tous les fichiers source seront *automatiquement* stockés dans le répertoire **src** et tous les fichiers binaires engendrés seront stockés dans le répertoire **bin**.

Remarque : le répertoire bin n'apparaîtra pas dans la vue Package Explorer car il ne contient pas de fichiers sources.



Vous pouvez vérifier la création des répertoires **bin** et **src** en ouvrant votre explorateur de fichiers.



3. Création des paquetages : application,essai,metier,presentation,user et util

Dans la vue **Package Explorer** (partie gauche du plan de travail), déployez si nécessaire le **cabinetMedical** et repliez les autres projets.

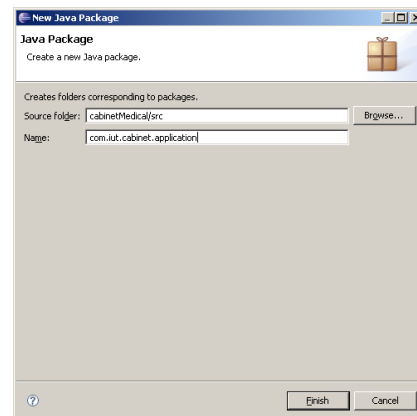
➤ Pour créer, le **paquetage application**, placez-vous sur **src**, cliquez sur le **bouton droit** de la souris, choisissez **New** puis **Package**.

Dans l'assistant « **New Java Package** », vérifiez que le Source folder soit bien : **cabinetMedical /src**

et compléter le Name du paquetage par : **com.iut.cabinet.application**

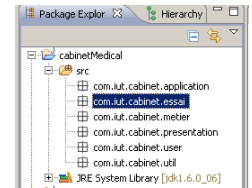
Cliquez sur **Finish**.

com.iut.cabinet.application apparaît dans l'arborescence du Package Explorer



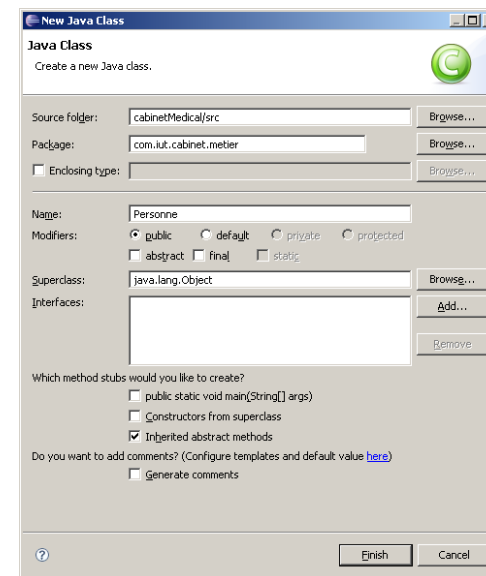
En ouvrant votre explorateur de fichiers, vous constaterez que la création de ce paquetage a bien engendré la création des répertoires **com**, **iut**, **cabinet** présentés dans l'arborescence de départ.

➤ Créez de même le paquetage **com.iut.cabinet.essai** et le paquetage **com.iut.cabinet.metier** et le paquetage **com.iut.cabinet.presentation** et le paquetage **com.iut.cabinet.user** et le paquetage **com.iut.cabinet.util**



4. Une première classe associée au package com.iut.cabinet.metier :

Pour créer une classe dans le paquetage **com.iut.cabinet.metier**, il suffit de se placer sur ce répertoire dans la vue **Package Explorer**, puis de cliquer sur le bouton droit de la souris, choisir **New → Class**



Vérifiez que le champ **Source folder** soit : **cabinetMedical/src**

et que le champ **Package** soit : **com.iut.cabinet.metier** (c'est ce champ qui permet d'associer la classe au paquetage souhaité... si ce n'est pas le cas cliquez sur **Browse** et choisissez le bon paquetage...)

Complétez le champ **Name** par : **Personne** (pas besoin de l'extension .java)

Cliquez sur **Finish**

La classe **Personne.java** apparaît dans l'éditeur et comme elle a été créée en la reliant au paquetage **metier**, Eclipse a automatiquement rajouté la ligne **package com.iut.cabinet.metier;**

A partir de l'explorateur de fichiers, vous pouvez vérifier qu'un fichier **Personne.java** a bien été créé dans le répertoire : **cabinetMedical\src\com\iut\cabinet\metier**

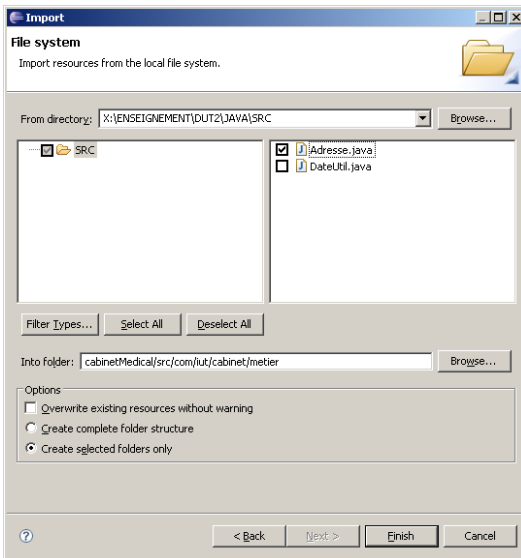
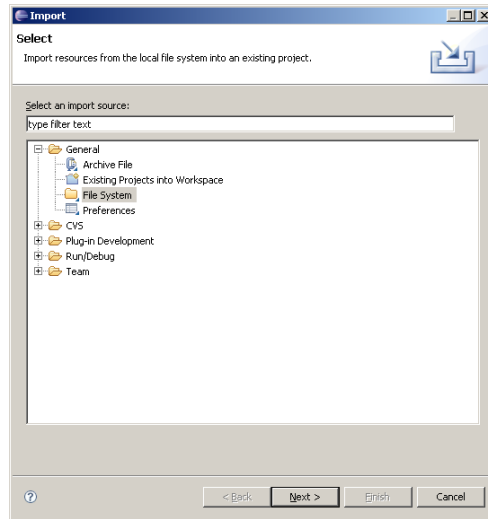
5. Importation d'une classe dans le package `com.iut.cabinet.metier` :

Pour importer une classe dans le paquetage `com.iut.cabinet.metier`, il suffit de se placer sur ce répertoire dans la vue *Package Explorer*, puis de cliquer sur le bouton droit de la souris, et de sélectionner **Import**.

Ouvrir l'onglet **General**,

Sélectionner **File System**

Cliquez sur **Next**



Remplir le champ **From directory**, en cliquant sur **Browse** et en sélectionnant le chemin jusqu'au répertoire SRC de la zone libre : `X:\ENSEIGNEMENT\DUT2\JAVA\SRC`

Cocher en suite, le fichier qui nous intéresse, pour le paquetage métier, ce sera le **Adresse.java**

Vérifier que le champ **Into folder** contient bien le paquetage destination, dans notre cas ce doit être : `cabinetMedical/src/com/iut/cabinet/metier`

Cliquez sur **finish** et vérifiez que la classe `Adresse` fait bien partie du paquetage `metier`

Ce fichier contient des commentaires Javadoc, générer la documentation et consulter la.

6. Importation d'une classe dans le package `com.iut.cabinet.util` :

Dans le paquetage `com.iut.cabinet.util`, importer la classe `DateUtil` (disponible sur la zone libre `X:\ENSEIGNEMENT\DUT2\JAVA\SRC`)

Générer la javadoc et prenez-en connaissance afin de vous familiariser avec cette classe.

Des fonctionnalités d'Eclipses très pratiques...

✧ A ce propos, l'éditeur Java d'Eclipse propose une fonctionnalité intéressante appelée « **completion** » (ou assistant de code) qui se déclenche soit volontairement grâce au raccourci clavier **Ctrl+Espace**, soit automatiquement dans certaines situations.

- Pour taper les commentaires Javadoc, la complétion s'avère très utile. En effet, si on tape `@param` puis **Ctrl+Espace**, la liste des paramètres de la méthode concernée sera listée... Si on tape `@exception`, la liste des exceptions sera listée, etc... Si on tape `@` dans un commentaire et si on suspend la frappe un instant, les mots-clés javadoc comme `@author`, `@deprecated`, etc... seront listés.
- Pour faire un affichage dans la console, vous devez taper la ligne de commande `System.out.println()` ; La complétion peut vous simplifier la tâche en tapant : `sysout` puis **Ctrl+Espace** ...et toute la ligne de commande précédente apparaît...

✧ **Pour gagner du temps**, Eclipse permet aussi de **générer automatiquement du code** grâce à certaines options du menu **Source**. Par exemple après avoir écrit la déclaration des deux attributs, on peut générer automatiquement les getteurs setteurs en sélectionnant : **Generate Getter and Setter**

➤ Revenir sur le fichier `Personne.java`. Grâce aux explications suivantes, vous allez coder très **rapidement** la classe `Personne` vue en TD (avec un `Ascendant`) en utilisant la génération automatique du code :

✧ **Attributs** : Pour commencer, vous devez écrire vous-même la déclaration de tous les attributs de la classe `Personne` :

```
idPersonne de type Integer ,
nom, prenom, telephone, portable, email de type String,
dateNaissance de type Date (du paquetage java.sql)
isMale de type boolean (initialisé par défaut à true)
adresse de type Adresse;
unAscendant de type Personne
```

ce n'est qu'ensuite que vous pourrez **générer automatiquement les getteurs et setteurs** :

Source → Generate Getter and Setter

La fenêtre **Generate Getter and Setter** s'ouvre.

Cliquez sur **Select All**,

Vous pouvez également générer automatiquement une partie de la javadoc en cochant l'option :

Generate method comments

Validez avec **OK**.

✧ **Getteurs & Setteurs** : De même, générer automatiquement les constructeurs avec à partir de :

Source → Generate Constructeur using Fields

Pour commencer, nous allons générer le constructeur qui permet de spécifier une valeur pour *tous* les attributs de la classe.

Une fois l'option **Generate Constructeur using Fields** sélectionnée, nous obtenons une fenêtre qui récapitule tous les attributs dans l'ordre alphabétique. Si vous cliquez tout de suite sur le bouton **OK**, un constructeur serait généré et l'ordre de ces paramètres respecterait l'ordre alphabétique, ce qui par la suite ne serait pas très pratique ...

Tout d'abord, nous allons donc ordonner les attributs suivant nos besoins.

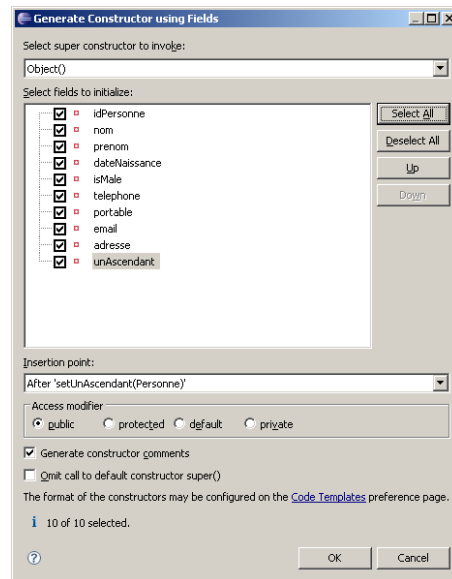
Cliquez par exemple sur **idPersonne**, puis cliquez sur le Bouton **Up** autant de fois que nécessaire pour qu'**idPersonne** remonte en première position.

Si vous voulez faire descendre **idPersonne** d'une position vous n'auriez qu'à cliquer sur le bouton **Down**.

Ordonnez les champs comme l'indique la copie d'écran ci-contre c'est-à-dire :

```
IdPersonne, nom, prenom,  
dateNaissance, isMale,  
telephone, Portable,  
email, adresse, unAscendant
```

Si tous les champs sont bien cochés, vous pouvez alors cliquer sur **OK** pour générer le constructeur mentionnant *tous* les attributs.



→ Pour générer le constructeur par défaut (*sans* argument), réouvrez la même fenêtre :

Source → Generate Constructeur using Fields

Cliquez sur le bouton **Deselect All**, puis sur **OK**.

Remarque : Conformément à la fin de l'énoncé du TD n°1, 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, dateNaissance, isMale, adresse)
- Un constructeur avec **6 arguments significatifs** avec ascendant (nom, prenom, dateNaissance, isMale, adresse, unAscendant)

→ Il ne vous reste donc plus qu'à générer **les constructeurs à 5 et 6 arguments** de la même manière en ordonnant et sélectionnant les attributs voulus.

Pour le constructeur à 5 arguments significatifs, vous devrez respecter l'ordre :

```
nom, prenom, dateNaissance, isMale, adresse
```

Pour le constructeur à 6 arguments significatifs, vous devrez respecter l'ordre :

nom, prenom, dateNaissance, isMale, adresse, unAscendant

hashCode et equals: De même il est possible de générer automatiquement les méthodes hashCode et equals.

Source → Generate hashCode() and equals()

Cliquez sur OK.

toString: Pour terminer, il ne vous reste plus qu'à coder la méthode toString :

```
public String toString() { ... }
```

... Vous venez de créer en quelques clics et quelques secondes la **classe métier** **Personne**...

Remarque :

- *N'oubliez pas de commenter vos programmes !!!*
- *N'oubliez pas de créer ou de compléter la documentation Javadoc !!!*

A propos des raccourcis clavier :

Petit à petit, vous allez vous familiariser avec votre environnement de développement.

Pour accélérer vos développements, vous apprécierez alors de connaître et d'utiliser les raccourcis claviers de votre IDE. Une liste des raccourcis les plus utilisés est par exemple disponible sur le blog d'Ippon Technologies à l'adresse suivante : <http://blog.ippon.fr/2011/10/03/eclipse-ameliorer-sa-productivite-grace-aux-raccourcis-clavier/>. Consultez cette liste, gardez-la à portée de main pour toutes les séances de TP à venir et n'hésitez pas à vous créer votre propre liste de raccourcis clavier !

Exercice 4 : Réaliser une application EssaiCabMed_v1

Pour tester la classe **Personne**, nous devons maintenant écrire une application.

1. Créez une nouvelle classe **EssaiCabMed_v1** dans le paquetage **com.iut.cabinet.essai**
Après avoir sélectionné **src** dans le package explorer, puis clic droit, puis **New** → **Class**
Choisir le **Package** correspondant grâce au bouton **Browse** et créer la classe **EssaiCabMed_v1**
2. Ce fichier comme dorénavant toutes les applications, aura la forme suivante :
(la méthode **main** appellera uniquement le constructeur et tout le code se trouvera dans le constructeur)

```
public class EssaiCabMed_v1
{

    public static void main(String args[])
    {
        // ici on se contente d'appeler le constructeur.
        new EssaiCabMed_v1 ();
    }

    EssaiCabMed_v1 ()
    {
        // placer ici le code principal de l'application
    }
}
```

Coder cette application pour qu'elle réalise :

- **l'instanciation** de deux personnes :
- la première aura le numéro 1, sera de sexe féminin, s'appellera DUPONT Julie. Elle sera née le 21/05/1960. Son numéro de téléphone sera le 0555434355, son numéro de portable le 0606060606, son email : julie.dupont@tralala.fr et son adresse : 15, avenue Jean Jaurès - 87000 Limoges –France. Elle n'aura aucun ascendant.
 - la seconde aura le numéro 2, sera de sexe masculin, s'appellera DUPONT Toto. Il sera né le 25/12/1991. Son numéro de téléphone sera le 0555434355, son numéro de portable le 0605040302, son email : toto.dupont@etu.unilim.fr et son adresse : Résidence La Borie - 185, avenue Albert Thomas - 87065 Limoges -France

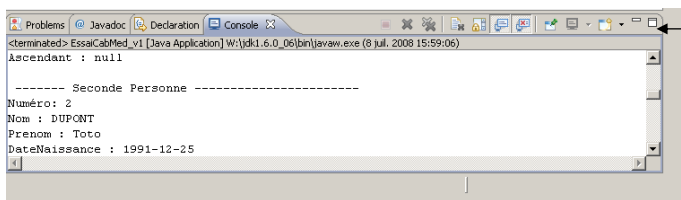
Remarque : pour instancier une Date, utiliser la méthode `toDate` de la classe `DateUtil`. En effet, le constructeur `Date(int year, int month, int day)` de la classe `java.sql.Date` est « deprecated » comme l'indique la javadoc (sur <http://docs.oracle.com/javase/7/docs/api/>)

➤ **l'affichage** des caractéristiques de ces deux personnes.

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

----- Seconde Personne -----
Numéro: 2
Nom : DUPONT
Prenom : Toto
DateNaissance : 1991-12-25
isMale : true
Telephone : 0555434355
Portable : 0605040302
Email : toto.dupont@etu.unilim.fr
Adresse :
    numéro: 185
    rue: avenue Albert Thomas
    voie: null
    batiment: Résidence La Borie
    codePostal: 87065
    ville: Limoges
    pays: France
    Ascendant : Numéro: 1
    Nom : DUPONT
    Prenom : Julie
    DateNaissance : 1960-05-21
    isMale : false
    Telephone : 0555434355
    Portable : 0606060606
    Email : julie.dupont@tralaia.fr
    Adresse :
        numéro: 15
        rue: avenue Jean Jaurès
        voie: null
        batiment: null
        codePostal: 87000
        ville: Limoges
        pays: France
        Ascendant : null
```

➤ La **modification**, directement dans le programme (aucune saisie pour le moment) d'un ou plusieurs numéros de téléphone....et **l'affichage** des caractéristiques de ces deux personnes après modification.



Pour agrandir la console, cliquez ICI

Exercice 5 : DateUtil et complément de la classe Personne

➤ Remarquez dans le jeu d'essai précédent l'affichage de la date de naissance :

DateNaissance : 1991-12-25

En utilisant une méthode de la classe DateUtil, afficher la date de naissance au format français, soit:

DateNaissance : 25/12/1991

➤ Nous souhaitons compléter la classe Personne déjà écrite en ajoutant une méthode « calculée » getAge de la forme : public int getAge() qui à partir de la dateNaissance et de la date du jour nous permettra de renvoyer l'âge de la personne (en année uniquement) sous forme d'entier.

Exemple : 18 ans

➤ Tester votre code dans l'application **EssaiCabMed_v1**.

Pour obtenir la date du jour, aller voir dans la documentation officielle de la classe Date et la classe GregorianCalendar !!!

Attention : Pour la prochaine séance la classe Personne doit absolument être implémentée (avec getAge), tester et marcher

Version Eclipse utilisée à l'IUT : <http://www.eclipse.org/downloads/>



Si jamais Eclipse n'apparaissait pas sur votre bureau à l'IUT, ce programme est accessible **depuis le réseau**. Vous pourrez ainsi le lancer **en vous rendant sur le lecteur W : et en cliquant sur le raccourci Eclipse**.

Version Java utilisée à l'IUT : Version 6 Update 6

<http://www.oracle.com/technetwork/java/index.html>

Au cours des TD/TP de Java, nous allons œuvrer à l'implémentation d'une partie du mini-projet **cabinetMedical**. Les notions étudiées durant les séances de cours seront introduites au fur et à mesure dans notre application.

Nous nous intéresserons plus particulièrement à la **gestion des Patients**.

- Les exercices « guidés » des TD/TP permettront notamment de mettre en place la **création d'un patient** dans un premier temps dans un mode console, puis une interface graphique sera réalisée.
- La persistance des données se fera tout d'abord à l'aide d'un fichier, puis à l'aide d'une base de données.

Ce projet peut être vu comme un projet transversal car il fera intervenir :

- **Le cours d'analyse** et plus particulièrement la **méthode UML**, puisqu'un travail d'analyse est forcément nécessaire avant de se lancer dans une implémentation quelconque.
- **Le cours d'expression/communication** puisqu'un rapport vous sera demandé à la fin du module. Ce rapport sera évalué sur la forme et sur le fond par les enseignants informatiques.
- et bien sûr le **cours programmation Objet appliqué au langage JAVA** puisque l'implémentation se fera en JAVA. Les programmes sources (avec javadoc) seront relevés et notés en milieu et fin de module.

Déroulement des séances de TP :

Au cours des TP, il vous sera proposé de compléter l'application **cabinetMedical** en implémentant un exercice vu en TD, et bien souvent, vous devrez implémenter un exercice complémentaire.

Il ne sera pas demandé de compte-rendu à chaque séance de TP JAVA.

En revanche, **vous devez IMPERATIVEMENT terminer vos programmes du mini-projet cabinetMedical pour la séance suivante** puisque l'application va se compléter au fur et à mesure des séances...

Les intervenants se réservent le droit de noter autant de fois qu'ils le souhaitent le respect de l'avancée du projet en début de séance.

Quant au rapport final, il devra contenir une introduction et une conclusion (voir cours C/E). Il présentera le projet, son évolution, il justifiera l'utilisation des notions objets, l'organisation des classes sans oublier des extraits de programmes commentés (surtout pas tout le code), ainsi que des jeux d'essais pertinents et commentés.

Les rapports seront évalués sur le fond et la forme.

Votre travail d'une séance à l'autre :

Vous devez IMPERATIVEMENT terminer vos programmes du « fil rouge »

cabinetMedical ...

Isabelle BLASQUEZ - Dpt Informatique S3 - TP 1 : Prise en main d'Eclipse