**Sommaire** :

TP0 : Bases du Java procédural 1

TP1 : Héritage 4

TP2 : Héritage, Polymorphisme et Collections 6

TP2.1 : Héritage et Polymorphisme 6

TP2.2 : Collection (Tri) 6

TP3 : Gestion des exceptions 7

TP4 : La notion d’interface 8

TP5 : Traitement du fichier de commandes Client 9

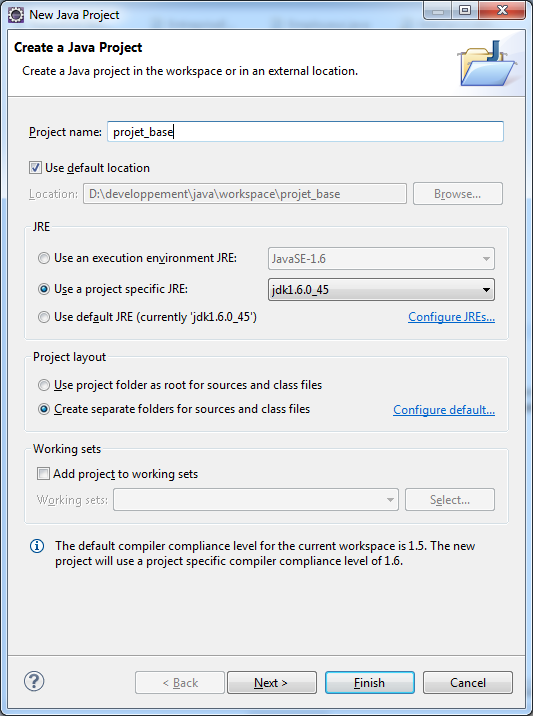
TP5.1 : Traitement en lecture 9

TP 5.2 : Traitement en écriture 9

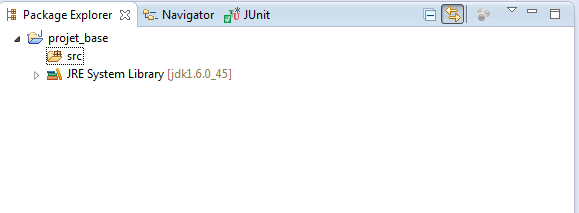
# TP0 : Bases du Java procédural

L’objectif de ce TP est de manipuler les instructions de base de la partie procédurale de Java.

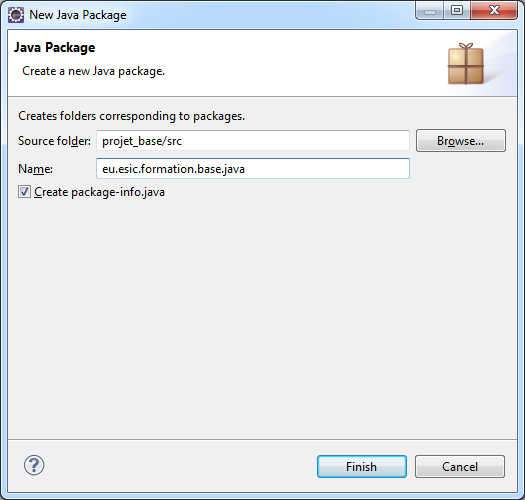
* Sous eclipse Créer un projet Java nommé «projet\_base» (File->New Project->Java Project), l’ecran suivant devrait apparaitre:



* Nommer le projet et cliquer sur «Finish», l’ecran ci-dessous devrait apparaitre:



* Faites un clic droit sur le répertoire «src» pour créer un nouveau package:

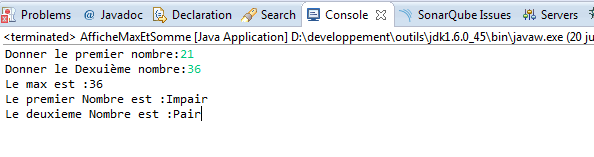


* Cliquer sur «Finish» après avoir mis le nom du package
* Créer la classe « **Test** » dont le contenu est celui du cours, et tester cette Classe

Dans le même projet nous allons créer une classe « **AfficheMaxEtSomme** » dont la méthode « main » prendra en entrée standard 2 nombres et :

* Affiche la somme et le maximum des 2 nombres.
* Affiche l’info de la parité de chacun des nombres

L’affichage devrait donner :



# TP1: Héritage

L’objectif de cette partie est de manipuler quelques notions principales liées à l’héritage.

Vous modéliserez un cas concret de la vie professionnelle d’une entreprise.

**Cas d’étude**: Une SSII souhaite concevoir un logiciel permettant de gérer son personnel, ce logiciel aura pour fonction principale de calculer le salaire annuel d’un membre du personnel, les différentes entités métier à gérer sont:

* Un membre du **personnel** est caractérisé par:
* Un nom
* Une année d’embauche
* Un salaire mensuel
* Un **développeur** (qui est évidemment un membre du personnel) caractérisé par :
* Un nombre d’Application auxquels il a participé
* Une note moyenne issue de la bonne qualité de codage
* Un **commercial** (qui est également membre du personnel) caractérisé par :
* Un nombre de clients gérés
* Un chiffre d’affaires moyenne sur une année

**Travail à faire**:

* Dessiner sur papier un diagramme de classe du système d’information
* Dans le projet précédent créer un package « eu.esic.formation.heritage.metier »
* Coder les classes du diagramme dans le package ci dessus.
* Penser aux constructeurs par défaut et avec paramètres

Nous devons désormais doter notre projet du calcul salaire annuel. Le salaire annuel est tout simplement le salaire mensuel **x 12** pour tout membre du personnel. Toutefois l’entreprise verse une prime aux commerciaux en fonction du chiffre d’affaires annuel réalisé. Le salaire d’un commercial est en fait son salaire annuel auquel on rajoute 10% du chiffre d’affaire réalisé.

**Travail à faire**:

* Implémenter ce calcul de salaire annuel de manière générale pour tout membre du personnel et de manière spécifique pour les commerciaux.

Nous allons utiliser notre projet ci dessus pour informatiser le cas d’études ci dessous. Pour ce faire supposons que dans notre SSII nous ayons les 2 membres du personnel suivant :

-Un **Developpeur** ayant les informations suivantes :

Nom : « **Liliam Thuriam** »

Année d’embauche : **1970**

 Salaire mensuel : **1800** euros

Nb D’application : **4**

 Note moyenne : **7** (sur 10)

-Un **commercial** avec les infos suivantes :

Nom: «**Yves Le Drian**»

Année d’embauche: **1985**

 Salaire mensuel: **1500** euros

Chiffre d’Affaires: **80000** euros (en 2015)

 Nb clients: **5**

Pour terminer, nous allons créer une classe avec une méthode «main» permettant de mettre en jeu le cas d’études ci-dessus, travail à faire:

* Créer une classe «**HeritageSSIIPersonnelTest**» avec une méthode «main»
* Afficher le **salaire annuel** du développeur ainsi que son **nom** et son **ancienneté**
* Afficher le **salaire annuel** du commercial ainsi que son **nom** et son **chiffre d’affaires**.

Le résultat d’affichage doit donner:

M. Liliam Thuriam (Developpeur) a gagné xxxxx.0 euros et a xxx ans d'ancienneté

M. Yves Le Drian (Commercial) a gagné xxxxx euros et a xxxxx de CA

# TP2 : Héritage, Polymorphisme et Collections

Le but de ce TP est de mettre en pratique quelques notions supplémentaires d’héritage et de polymorphisme.

L’objectif métier est de pouvoir calculer le prix total des fruits (toute nature confondue), ainsi que leur poids total.

## TP2.1 : Héritage et Polymorphisme

* Commencer par importer le projet « **tp-polymorphisme**» sous Eclipse
* Faire les héritages nécessaires
* Calculer « **getPrix**» dans toutes les classes concrètes, il faut tenir compte du fait qu’il y’a une réduction de **10%** sur le prix de la « **Pomme**»
* Compléter les Tests JUnit et faites les tourner

## TP2.2 : Collection (Tri)

L’objectif de cet exercice est de pouvoir trier les fruits suivant leur « **poids**»

* Faites le nécessaire sur la classe « Fruit » pour la rendre « **Comparable** »
* Compléter la méthode de Test Unitaire « **testTriFruitClasseParPoids**» afin qu’elle trie la collection construite grâce à la méthode fournie dans la classe de Test
* Lancer le Test Unitaire

# TP3 : Gestion des exceptions

L’objectif de cette partie c’est de pouvoir catcher et sortir en erreur les cas où le poids d’un Fruit est négatif, pour cela :

* Créer une classe « **FruitPoidsPositifException**» qui traitera l’exception en cas de poids négatif
* Vérifier le poids dans le constructeur de la classe « **Fruit**» et lancer l’exception créée ci dessus en cas de violation, si possible afficher dans le message d’erreur le poids erroné

# TP4 : La notion d’interface

L’objectif de cette partie est de pouvoir utiliser une interface pour simuler du polymorphisme.

Pour cela :

* Créer une interface « **IFruit**» avec les services ci dessous :

**public** **void** affiche();

**public** **void** affichePoids();

**public** **void** afficheCouleur();

**public** **abstract** Double getPrix();

**public** Integer getPoids();

* Créer une 2eme classe de Test contenant les mêmes méthodes que la classe de Test Précédente, excepté la méthode de Tri à supprimer.
* Remplacer la classe « Fruit » par « IFruit » dans cette classe de Test et lancer les Tests Unitaires, tout devrait marcher comme avant

# TP5 : Traitement du fichier de commandes Client

L’objectif de ce TP est de manipuler les fichiers plats en lecture en Java.

## TP5.1 : Traitement en lecture

**Travail à faire**:

* Importer le projet « **tp-filereader**», vous y trouverez un fichier dont la description est la suivante :
* 1ere Colonne : Identifiant du client
* 2eme Colonne : Nom du client
* 3 eme colonne : Adresse
* 4 eme colonne : Quantité de produits achetés
* 5 eme colonne : Le prix unitaire
* Q1 : Renseignez les **TODO 1** & **2** afin de :
  + Charger le fichier « fich\_achat\_client.txt »
  + Remonter le client ayant effectué le moins d’achat en chiffre d’Affaires
* Q2 : Renseignez les **TODO 3** & **4** afin de :
  + Charger le fichier « fich\_achat\_client.txt »
  + Remonter le client ayant effectué le plus d’achat en chiffres d’Affaires

## TP 5.2 : Traitement en écriture

Nous souhaitons repartir du même fichier précédent et en extraire les clients résidant dans la ville de « Rennes » (Code postal commençant par « 35 »)

* **Q3 : Remplissez** le TODO 5
* Q4 : Faites tourner le TU correspondant