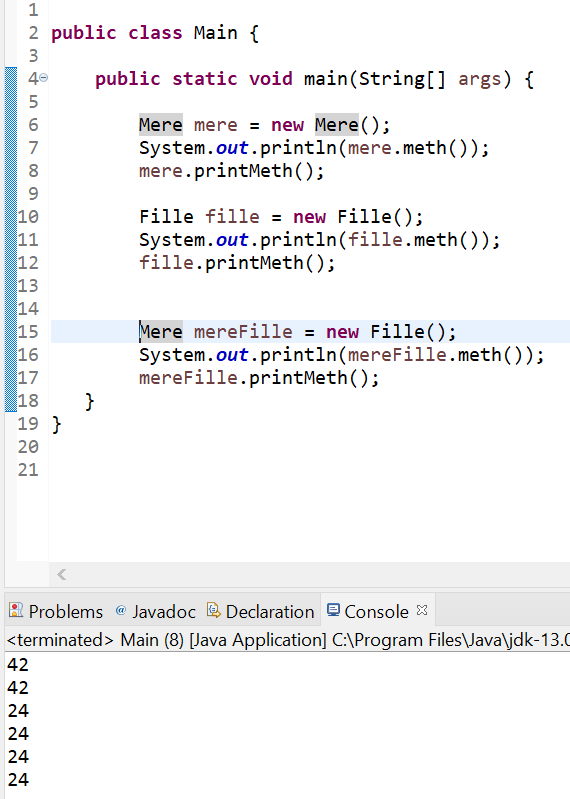
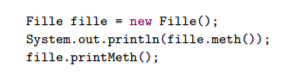
Exercice 1 **YANIS LAGHMOURI**

1)

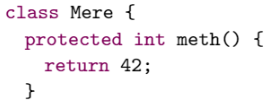
Le main affiche 42 pour les deux premières méthodes car elles sont appliquées à un objet instanciée avec la classe Mere().   
La méthode int meth() créée dans la classe Mere retourne 42, la méthode void printMeth() affiche l’éxécution de meth() retournant 42.



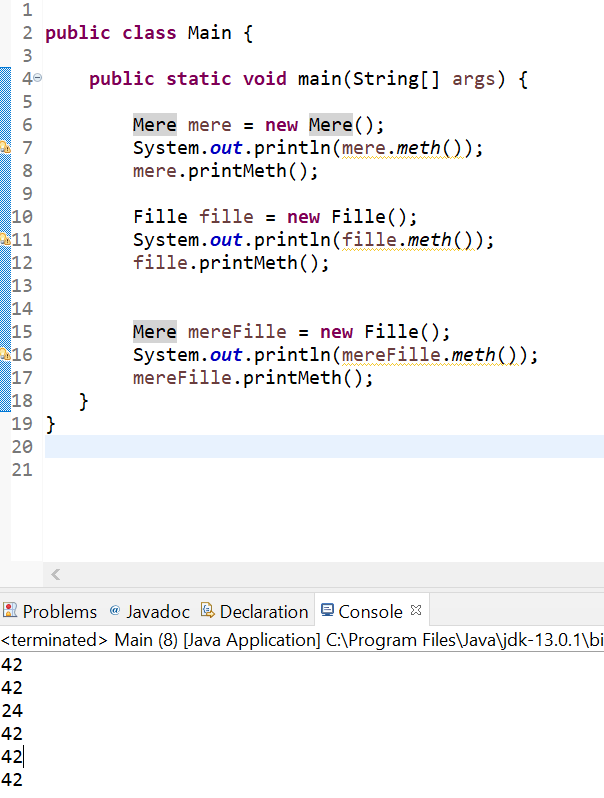
Le main affiche 24 car les méthodes définies appelées en classe Fille que l’on appelle, retourne 24.



Ici, cas particulier, à un objet déclaré Mere, on l’initialise avec la classe Fille(). L’exécution des méthodes ne va pas retourner 42 mais plutôt 24 car ce sont les méthodes de la classe Fille qui vont être appelées, c’est le principe de **Polymorphisme**.

2) S’il est dans Fille, un objet de type Fille a accès à 2 méthodes meth().  
 Celle de la classe Mère et celle de la classe Fille.  
    
Il a accès à celle de la classe Mère car meth() est déclaré en protected (visibles par les héritiers et classes du même package).  
Il y accède grâce au mot clé Super.

Dans un Main, on ne peut accéder seulement à la fonction de la Fille car on ne peut pas y utiliser le mot clé Super.

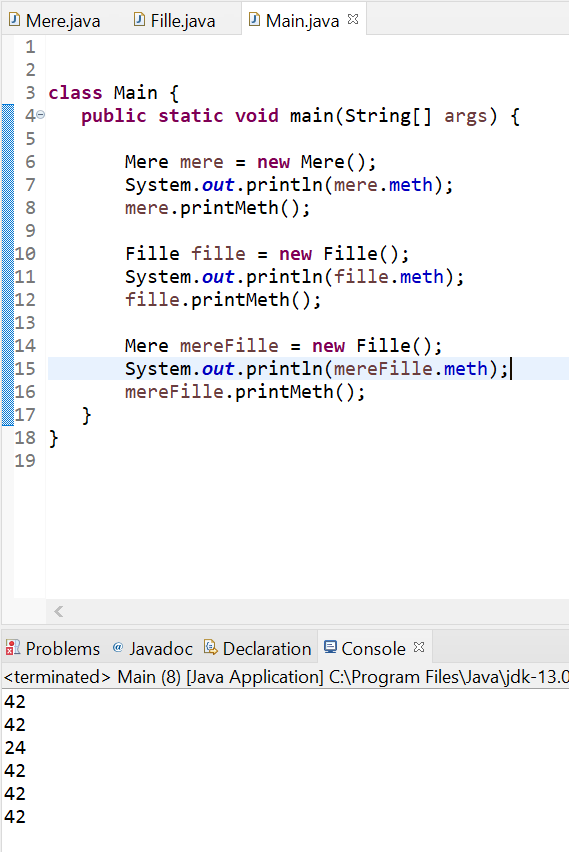
3)

Les méthodes meth(), une fois déclarées statiques appartiennent à la classe plutôt qu’à une instance de classe.

C’est pourquoi avec un objet de type Mere, même instancié avec une autre classe (exemple Mere merefille = new Fille () ), l’execution de meth() renverra 42.

Si l’objet est de type Fille, meth() renverra 24 (execution de la méthode de classe meth() de Fille). En revanche printMeth() est appelée dans la classe Mere où est déclarée aussi la méthode statique meth() (appelée aussi méthode de classe), c’est pourquoi ce sera 24 (le retour de la méthode meth() de la classe Mere) qui s’affichera.

4)

Le comportement est affiché dans la console.

Dans la classe Mere, meth est protected.   
L’appel à mere.meth et mere.printMeth() ne posent pas de probleme de conflit ou de choix, on se situe toujours dans la classe Mere.

fille.meth affiche 24, c’est l’attribut meth déclaré à 24 en public dans la même classe, l’ordinateur n’a pas à faire de choix.  
En revanche le cas de fille.printMeth est différent : ici c’est la méthode printMeth() de la classe **Mere qui** est utilisée et c’est l’attribut meth de cette même classe qui est choisi.

mereFille.meth et printMeth() retourne 42 car l’objet mereFille est à la base déclaré avec Mere.

Pour que fille.printMeth() affiche 24, il faudrait redéfinir printMeth() dans la classe Fille.

**Exercice 2. Redéfinition – surcharges**

1) Tout d’abord il y a des erreurs concernant IOException.   
  
 On les résout en important **import** java.io.IOException;

Erreur à la compilation car la méthode miage() n’est pas implémentée dans la classe Mère.

Ces méthodes, si elles sont appelées, empêcheraient la compilation :

**private** **void** e() {System.***out***.println("Fille\_e");}

**char** h() {System.out.println("Fille\_h"); **return** ’c’;}

**int** i() {System.***out***.println("Fille\_i"); **return** 3; }

Ces méthodes sont redéfinies en modifiant la signature (type de retour, nombre de parametres, types d’erreur rejetées, la lisibilité), ce qui n’est pas possible lors d’une redéfinition.



2)

Après avoir mis en commentaire les appels à miage(), il n’y a plus d’erreur à la compilation.

3) La surcharge et la redéfinition sont deux choses différentes.

**Surcharge** : Lorsque deux méthodes ou plus dans une classe ont le **même nom de méthode** mais des **paramètres différents**.

**Redéfinition** : Lorsque deux méthodes ont le même nom et les mêmes paramètres, en sachant que l’une des méthodes est dans la classe parente et l’autre dans la classe fille.

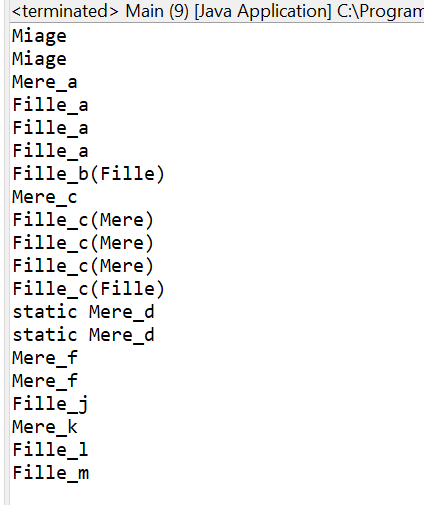
Les surcharges sont les méthodes suivantes :

Dans la classe mère : les deux méthodes c()

Dans la classe fille : les deux méthodes c() et les deux méthodes d()

Les redéfinitions sont les méthodes suivantes :   
 a()   
 b()  
 c()  
 d()  
 f()  
 g()  
 j()  
 l()  
 m()

4)



fille.miage(); affiche miage car la classe fille utilise sa méthode miage() qui print Miage

((Fille)mereFille).miage(); affiche miage car on effectue un typage vers Fille, ainsi la méthode miage() est visible

mere.a(); affiche Mere\_a car on se situe dans la classe Mère et on appelle a() qui print Mere\_a

mereFille.a(); affiche Fille\_a car mereFille instanciée avec Fille, execution car méthode redéfinie

fille.a(); affiche Fille\_a . Il s’agit de la méthode a() de la classe Fille qui affiche Fille\_a.

((Mere)mereFille).a(); affiche Fille\_a . mereFille reste une instance de Fille et utilise a() de la classe Fille malgré qu’elle soit castée en Mère

mereFille.b(**null**); affiche Fille\_b(Fille) mereFille a été instancié Fille, il affiche Fill\_b(Fille)

mereFille.c(); affiche Mere\_c . mereFille est un objet instancié fille.

mereFille.c(mere); affiche Fille\_c(Mere)

mereFille.c(mereFille); affiche Fille\_c(Mere)

mereFille.c(fille); affiche Fille\_c(Mere) . MerFille est typé en tant que Mere

fille.c(fille); affiche Fille\_c(Fille). L’objet Fille appelle la méthode c(fille) de la classe Fille.

mere.*d*(); affiche static Mere\_d. L’objet de la classe Mere appelle la methode d() de la classe Mere

mereFille.*d*(); affiche static Mere\_d. mereFille a été typé en tant que Mere, la methode parcourue en premier est la methode d() de la classe Mere

mere.printF(); affiche Mere\_f printF() appelle f() de la classe Mère car mere en est une instance.

mereFille.printF(); affiche Mere\_f. mereFille a été typé en tant que Mere, la fonction f() parcourue en premier est celle de la classe Mere.

mereFille.j(); affiche Fille\_j

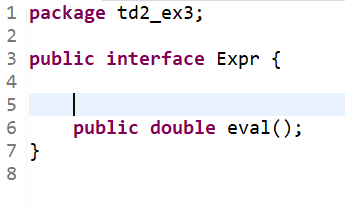
mereFille.k(); affiche Fille\_k

mereFille.l(); affiche Fille\_l

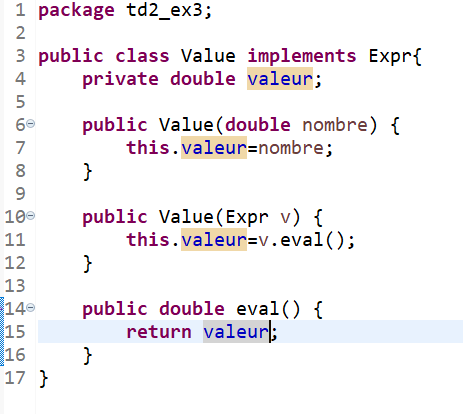
mereFille.m(); affiche Fille\_m

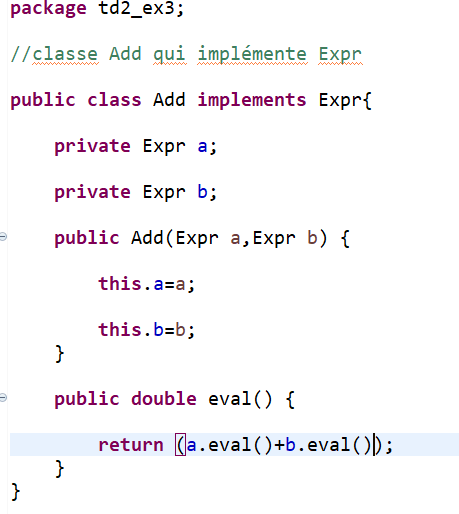
Exercice 3. Expressions arithmétiques

1) et 2)



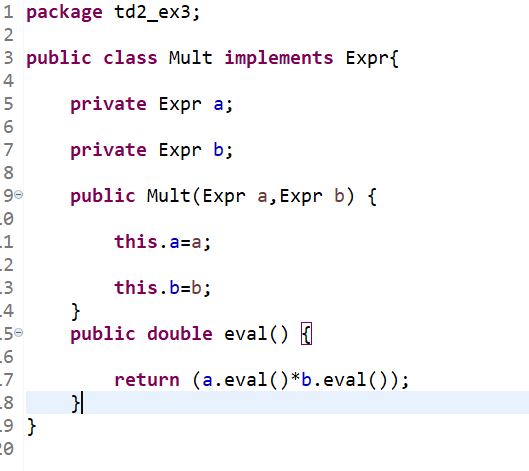
La méthode eval() à implémenter, on la met dans une interface Expr qui sera implémentée par les autres classes.

 Classe Value



Classe Add

3)



Classe Mult pour implémenter la multiplication

4)

Classe qui implémente la racine carrée

