TD2 – Java- JAMEL ALLAL

Exercice 1

1)

```
42
42
24
24
24
24
```

```
20          Mere mere = new Mere();
21          System.out.println(mere.meth());
22          mere.printMeth();
```

Les deux premier 42 sont affichés car on fait deux fois appels à la méthode meth() d'un objet mère qui retourne toujours 42 : Ligne 21 on fais un syout de mère.meth() et Ligne 22 mere.printMeth() appel ce code :

```
public void printMeth() {
    System.out.println(meth());
}
```

Qui est également un systout de meth() .

Ensuite on affiche 2 fois 24 car on fais deux fois appelle à la méthode meth() d'un objet fille qui retourne toujours 24. Même si la méthode printMeth() est définis dans la classé mère, étant donné que la classe fille hérite de la classe mère elle peut faire appelle à cette méthode également. Et lorsque que le sysout (meth()) ; sera executé ce sera la methode meth() de la classe fille.

```
23          Fille fille = new Fille();
24          System.out.println(fille.meth());
25          fille.printMeth();
```

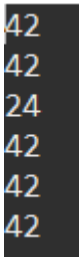
Les 2 dernier print affiche 24 car même si le type déclaré est Mère , le type instancié est Fille, cela signifie que l'objet MereFille pourra utiliser uniquement les méthodes existantes dans la classe mère (dont la fille hérite) . Comme il s'agit d'une Fille , c'est la méthode meth() de fille qui sera appelée.

```
26          Mere mereFille = new Fille();
27          System.out.println(mereFille.meth());
28          mereFille.printMeth();
```

2) si il est dans Fille, un objet de type Fille à accès à 2 méthodes meth(). Il peut accéder à la sienne directement en faisant this.meth() ou meth () tout court . Et il peut accéder à celle de sa classe Mère en faisant super.meth() car la methode meth() de la classe Mère est en protected (ce qui donne accès à ses classe filles). Si il est dans Main , un objet de type fille ne peut accéder directement qu'a une seule méthode meth() qui est la sienne.

3)

Voici l'affichage si les méthodes meth() sont statiques



Dans le cas d'une méthode static, elle est appelée directement sur une classe. Donc en comparaison avec la Q1, ici, seule fille.meth() renvoie 24.
En effet, mereFille est un objet dont le type déclaré est "Mere" et étant donné que la méthode est devenue static on appelle directement la méthode meth() de la classe mère et non de la fille.

4) On obtient le même affichage qu'a la question précédente.

En effet, comme on passe par les attributs, les objets déclarés en tant que type Mère vont récupérer l'attribut qui est lié à Mère et donc afficher 42. (c'est le cas pour l'objet mereFille qui est de type Mere)

De ce fait, quand on appelle la méthode printMeth() depuis la Fille, on va appeler la méthode printMeth() qui se situe dans la mère qui va récupérer la valeur de l'attribut meth de la Mère. C'est pour cette raison que fille.printmeth() va renvoyer 42 et non 24 comme à la question 1.

Exercice 2

1)

Les erreurs de compilation sont les suivantes :

> The method miage() is undefined for the type Mere
>
> The method miage() is undefined for the type Mere

Elles apparaissent car la méthode miage() n'est pas implémentée dans la classe Mère.

2) Quand on enlève les appels à la méthode Miage on à plus d'erreur.

3)

- Une redéfinition, c'est lorsqu'une classe Fille a une classe Mere et que la classe Fille utilise la même signature qu'une fonction de la classe Mere avec le même nom mais avec un comportement différent. (on ajoute l'annotation @Override)

- Une surcharge est le fait de définir le même nom de fonction plusieurs fois dans une classe mais avec des signatures différentes (type d'arguments différents, nombre d'arguments différents…

Les redéfinitions :

- a(), b(), c(), d(), f(), g(), j(), l(), m()

Les surcharges dans Mere :

- c()

Les surcharges dans Fille :

- c(), d()

4)

Miage -> `fille.miage();` -> La méthode miage() de la classe Fille sert à afficher « miage »

Miage -> `((Fille) mereFille).miage();` -> on peut appeler la méthode miage de Fille

Mere_a -> `mere.a();` -> La méthode a() de la classe Mere permet d'afficher « Mere_a »

Fille_a -> `mereFille.a();` -> L'objet mereFille est de type Mere mais il est instancié en Fille. Ainsi lorsqu'on fait cette exécution, on appelle la méthode a() de la classe Fille, c'est pourquoi on affiche « Fille_a. »

Fille_a-> `fille.a();` ->L'objet mereFille est de type Mere mais il est instancié en Fille. Ainsi lorsqu'on fait cette exécution, on appelle la méthode a() de la classe Fille, c'est pourquoi on affiche « Fille_a. »

Fille_a-> `((Mere) mereFille).a();` -> mereFille.a() et ((Mere) mereFille).a() renvoie le même résultat car mereFille est déjà de type mère donc le caster en Mere ne change rien. On apelle donc une nouvelle fois la méthode a() de la classe Fille qui affiche « Fille_a »

Fille_b(Fille) -> `mereFille.b(null);` -> L'objet mereFille est de type Mere mais il est instancié en Fille. Ainsi, lorsqu'on exceute ce code on appelle la méthode b(Fille fille) de la classe Fille qui affiche « Fille_b(Fille). »

Mere_c -> `mereFille.c();` -> On appelle la méthode c() de la classe Mere lors de cette exécution car cette méthode n'a pas été redéfinit dans la classe Fille.

Fille_c(Mere) -> `mereFille.c(mere);` -> L'objet mereFille est de type Mere mais il est instancié en Fille. Ainsi, lorsqu'on exécute ce code on a la méthode c(Mere mere) de la classe Fille qui affiche « Fille_c(Mere) »

Fille_c(Mere) -> `mereFille.c(mereFille);` -> L'objet mereFille étant de type Mere, on suitle même raisonnement qu'a l'affichage précédent et on affiche « Fille_c(Mere) »

Fille_c(Mere) -> `mereFille.c(fille);` -> l'objet mereFille est de type Mere mais la classe Mere n' a pas de méthode c(fille), comme la classe Fille étend la classe Mere, alors on prend la méthode c(Mere) dans la classe Fille pour exécuter cette ligne

Fille_c(Fille) -> `mereFille.c(mere);` -> La méthode c(fille) de la classe Fille affiche « Fille_c(Fille) »

static Mere_d -> `mere.d();` -> La méthode d() de la classe Mere affiche « static Mere_d »

static Mere_d -> `mereFille.d();` -> L'objet mereFille est de type Mere et il est instancié en Fille. La méthode d() de la classe Mere étant static elle ne dépend pas de l'instance mais du type, ainsi c'est pour cela que c'est la méthode d() de la classe Mere qui est appelé pour cette ligne et affiche « static Mere_d »

Mere_f -> `mere.printF();` -> La méthode printF() de la classe Mere appelle la méthode f() de la classe mère qui affiche « Mere_f »

Mere_f -> `mereFille.printF();` -> L'objet mereFille est de type Mere et instancié en Fille. La méthode printF() n'existe que dans la classe Mere donc c'est elle qui est exécuté pour cette ligne. On affiche donc« Mere_f ».

Fille_j -> `mereFille.j();` -> L'objet mereFille est de type Mere mais instancié en Fille. Ainsi, lors de l'appel de la méthode j() on exécute la redéfinition de j() dans la classe Fille et on affiche « Fille_j »

Fille_k -> `mereFille.k();` -> L'objet mereFille est de type Mere et instancié en Fille. La méthode k() n'existe que dans la classe Mere donc c'est elle qui est exécuté pour cette ligne d'où l'affichage de « Mere_k».

Fille_ -> l `mereFille.l();` -> L'objet mereFille est de type Mere et instancié en Fille. Ainsi, lors de l'appel de la méthode l() on exécute la redéfinition de l() dans la classe Fille, et on affiche donc « Fille_l »

Fille_m -> `mereFille.m();` -> L'objet mereFille est de type Mere et instancié en Fille. Ainsi, lors de l'appel de la méthode m() on exécute la redéfinition de m() dans la classe Fille, et on affiche « Fille_m »

Exercice 3

Code github