# **Exo 1**

1. Le main affiche

42

42

24

24

24

24

En effet, pour les deux premiers outputs, on appelle les méthodes meth et printmeth (qui renvoient la même chose) définies par la classe Mere. Comme c’est un objet de Mere qui appelle ces méthodes, ce sont les méthodes de cette classe qui seront exécutées et la sortie attendue est bien 42.

Pour les deux outputs suivants, on appelle toujours les deux méthodes meth et printMeth mais cette fois-ci par un objet de la classe Fille, héritant de la classe Mere. Comme la méthode Fille a redéfini les deux méthodes (avec une valeur de retour différente de celle de la même méthode dans Mere), ce sont celles-ci qui seront exécutées lors de leur appel par l’objet Fille. C’est pourquoi, la sortie est différente, c’est 24.

Finalement, pour les deux derniers outputs, par principe de polymorphisme, ce sont les méthodes redéfinies dans Fille qui sont exécutées.

1. Qu’il soit dans la méthode Main ou Fille, un objet de type fille a accès à une seul méthode meth (sous réserve que toutes les classes soient déclarées dans le même package. La seule différence réside dans la possiblité d’user du polymorphisme et de de déclarer l’objet fille en passant par le super-type Mere. Dans ce dernier cas, ce sera la méthode meth() de Mere qui sera appelée bien qu’au final ce soit la méthode meth() de Fille qui soit exécutée.

?

1. Si les méthodes meth() sont statiques, elles ne dépendent plus d’un objet mais d’une classe. Cela implique que :

* meth() est appelé sans passé par un objet
* On ne peut pas appeler meth() dans Main car cette classe ne redéfinit pas la méthode

1. On obtient en sortie :

42

42

24

42

42

42

Pour les deux premiers outputs, on obtient 42 et 42 car une première fois on appelle l’attribut meth de Mere qui vaut 42 et une seconde fois la méthode printMeth qui renvoie la valeur de cet attribut.

Pour les deux suivants, on obtient 24 puis 42 car dans un premier temps on appelle l’attribut meth de Mere qui vaut 24 puis, dans un second temps, on appelle la méthode printMeth héritée de Mere et qui n’a pas été redéfinie, c’est donc cette méthode (renvoyant 42) qui est exécutée.

Finalement, pour les deux derniers outputs, on obtient 42 et 42 car l’objet mereFille est définie par le super-type Mere, l’attribut meth qui lui est accordé est donc celui de Mere (valant 42). Le second 42 s’explique de la même manière que précédemment.

# **Exo 2**

1. Plusieurs erreurs de compilation.

Dans Mere : Aucune erreur

Dans Fille :

* private void e() : comme Fille étend Mere et e() est une méthode de Mere, alors Fille la réécrit. Mais Fille n’a pas le droit de restreindre la visibilité de la méthode qu’elle réécrit. Il faut soit laisser e() en protected soit la passer en private.
* char h() et int i() : Comme Fille réécrit ces deux méthodes de Mere, elle n’a pas le droit de changer le type de retour. Ici, Fille transgresse cela puisque Mere déclare le type de h() en int et le type de i() en void. Si on souhaite déclarer des toutes autres méthodes, il suffit de leur choisir un autre nom.
* void j() et void k() : Comme Fille redéfinit ces deux méthodes de la classe Mere, elle doit utiliser les mêmes levées d’exception ou un sous-type de l’exception figurant dans la méthode de la classe Mere. Autrement dit, le passage de throws Exception à thrwos IOException ne pose pas problème, mais dans l’autre sens, ce n’est pas possible. Il faut par ailleurs ajouter un import java.io.IOException dans Fille.

Dans Main

* mere.miage() et mereFille.miage() : mere et mereFille sont deux objet créés avec le super-type Mere, donc toutes les méthodes appelées via cet objet sont celles de Mere. Or, Mere n’a pas de méthode miage(). Donc ces lignes génèrent des erreurs. Il faudrait donc définir une méthode miage() dans Mere.

1. En suivant les explications données à la question précédente, les méthodes à retirer sont les suivantes.

* Dans Mere : void e() + int h() + void i() + void k()
* Dans Fille : Aucune
* Dans Main : Supprimer les lignes *mere.miage()* et *mereFille.miage()* à moins de définir une méthode miage() dans la classe Mere, et la ligne *mereFille.k()* suite à la suppression de void k() dans Mere

1. Surcharge : deux méthodes de **même** nom dans la **même** classe mais qui change l’implémentation de la méthode ou ses paramètres. Ex du cours : m(List<Character> a) et m(Object a)

* Dans ce TD : Dans Fille, *void c(Fille b)* surcharge *void c(Mere mere)* et *void d(Mere mere)* surcharge *void d()*

Redéfinition : deux méthodes de **même** nom dans deux classes **différentes**. Ex du cours : m(CharSequence a) dans A et dans B

* Dans ce TD, la classe Fille redéfinit plusieurs méthodes de Mere, ce sont les suivantes (ans compter celles qui ont été supprimées à la question précédente) : void a() + void b(Fille fille) + void c(Mere mere) + void d() + void f() + void j() + void l() + void m()

1. En prenant en compte les suppression faites à la question 2, les affichages du Main sont les suivants :

Miage 🡪 Appel par un objet Fille de la méthode void miage()

Miage 🡪 Appel de la méthode void miage() de la classe Fille car conversion de l’objet Mere appelant en objet Fille

Mere\_a 🡪 Appel par un objet Mere de la méthode void a()

Fille\_a 🡪 Appel par un objet Mere de la méthode void a() qui exécute cependant la méthode a() redéfinie dans Fille (polymorphisme)

Fille\_a 🡪 Appel par un objet Fille de la méthode void a()

Fille\_a 🡪 Appel par un objet Mere de la méthode void a() qui exécute cependant la méthode a() redéfinie dans Fille (polymorphisme)

Fille\_b(Fille) 🡪 Appel par un objet Mere de la méthode void b() qui exécute cependant la méthode b() redéfinie dans Fille (polymorphisme)

Mere\_c 🡪 Appel par un objet Mere de la méthode void c()

Fille\_c(Mere) 🡪 Appel par un objet Mere de la méthode void c() qui exécute cependant la méthode c(Mere mere) redéfinie dans Fille (polymorphisme)

Fille\_c(Mere) 🡪 Appel par un objet Mere de la méthode void a() qui exécute cependant la méthode void c(Mere mere) redéfinie dans Fille (polymorphisme)

Fille\_c(Mere) 🡪 Appel par un objet Mere de la méthode void a() qui exécute cependant la méthode void c(Mere mere) redéfinie dans Fille (polymorphisme)

Fille\_c(Fille) 🡪 Appel par un objet Fille de la méthode void a() qui exécute cependant la méthode a() redéfinie dans Fille (polymorphisme)

static Mere\_d 🡪 Appel par un objet Mere de la méthode void d()

static Mere\_d 🡪 Appel par un objet Mere de la méthode void d()

Mere\_f 🡪 Appel par un objet Mere de la méthode printF()

Mere\_f 🡪 Appel par un objet Mere de la méthode printF()

Fille\_j 🡪 Appel par un objet Mere de la méthode void j() qui exécute cependant la méthode void j() redéfinie dans Fille (polymorphisme)

Fille\_l 🡪 Appel par un objet Mere de la méthode void l() qui exécute cependant la méthode void l() redéfinie dans Fille (polymorphisme)

Fille\_m 🡪 Appel par un objet Mere de la méthode void m() qui exécute cependant la méthode void m() redéfinie dans Fille (polymorphisme)

# **Exo 3**

Voir les classes sur le git : Expr, Value, Add, OneArgument, TwoArguments, Sqrt, Mult