TD2

Exercice 1

1. Le Main affiche :

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

Pour « mereFille » on effectue un transtypage implicite (mettre un object de type Fille dans un object de type Mere), donc ce sont les méthodes de la classe Fille qui sont utilisées.

2. Lorsqu'on est dans Fille on accède à 2 méthodes (meth() de Mere avec le mot clé « super » et meth() de Fille).

Dans Main, on a seulement accès à la méthode meth() de la classe Fille.

3. le Main affiche :

```
42
42
24
```

42 → printMeth va appeler la méthode meth() de la classe Mere

42 → ça appel la méthode meth de la classe Mere (pas de polymorphisme pour les méthodes static)
42 → On appelle printMeth de la classe Mere

Une méthode « static » ne peut pas être sujette à l'héritage, les méthode static sont évaluées que lors de la compilation

4. le Main affiche :

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

On accède aux valeurs auxquelles le champ est rattaché en termes de classe.

Exercice 2

1. Voici les erreurs :

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
	The method miage() is undefined for the type Mere
	The method miage() is undefined for the type Mere
	The method k() from the type Mere refers to the missing type IOException
```

La méthode miage() est non définie dans Mere, et pour la méthode k() il faut importer la IOException

2. On retire (j'ai mis en commentaire) les méthodes miage() et k() qui sont appelé dans le main.

3. Une surcharge consiste à réécrire une méthode avec un nombre d'argument ou un type d'argument différent

Une redéfinition consiste à avoir deux méthodes avec le même nom et les mêmes paramètres, mais l'une des méthodes est dans la classe parente et l'autre dans la classe fille. Cependant, l'on peut changer la visibilité de la méthode tant qu'elle n'est pas « restreinte » par rapport à celle de classe mère et changer le type de retour tant que le type de retour modifié hérite du type de retour de base.

4.
```
Mere_a = L'objet mere appel sa méthode a()
Fille_a = mereFille contient une instance de fille donc appel a() de Fille
Fille_a = appel de la méthode a() de Fille car instance de Fille
Fille_a = mereFille contient une instance de fille donc appel a() de Fille malgré
le cast explicit
Fille_b(Fille) = mereFille contient une instance de fille, appel de la méthode b()
de Fille

Mere_c = c() est défini seulement dans Mère donc on appel c() de la classe Mère
Fille_c(Mere) = mereFille contient une instance de fille donc appel c(Mère) de
Fille
Fille_c(Mere) = mereFille est de classe Mere donc le compilateur appel c(Mere) de
la classe Fille
Fille_c(Mere) =
Fille_c(Fille) = appel de c(fille) de la classe Fille par l'instance fille

static Mere_d = appel de d() de la classe Mere par l'instance mere
static Mere_d = appel de d() de la classe Mere par l'instance mereFille car une
méthode static ne peux pas être appliqué à l'héritage

Mere_f = appel de printF() de la classe Mere par l'instance mere par l'instance
mere et appel f() de la classe mere
Mere_f = appel de printF() de la classe Mere par l'instance mereFille par
l'instance mere et appel f() de la classe mere

Fille_j = appel j() de la classe Fille par l'instance mereFille car c'est une
instance de Fille à l'intérieur
Fille_l = appel l() de la classe Fille par l'instance mereFille car c'est une
instance de Fille à l'intérieur
Fille_m = appel m() de la classe Fille par l'instance mereFille car c'est une
instance de Fille à l'intérieur
```