

- 1) Quand c'est un objet instancié en tant que Fille, meth return 24 et quand c'est initialisé en tant que Mère, meth() return 42.

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

- 2) Si il est dans Fille, il a accès à deux méthodes meth(), il peut y accéder avec le mot clé « super » car la méthode meth() de Mère est déclaré en Protected. Si il est dans Main, il a accès à une seule méthode meth(), celle de sa classe.
- 3) Le comportement de meth dépendra uniquement de la classe de l'objet, ainsi pour un objet référencé Mere, meth renverra 42 et pour un objet référencé Fille meth renverra 24.
- 4) Si meth est un champ alors la methode printMeth() va toujours afficher 42 que ce soit un objet Fille ou un objet Mère. Il y uniquement pour le cas ou appel meth depuis un objet Fille que on affiche 24.

Le comportement sera le même si les champs sont static.

```
<terminated> Main (8) [Java Application] C:\Prog
42
42
24
42
42
42
```

## Exercice 2

- 1) - Ligne 42 : `char h() {System.out.println("Fille_h"); return 'c';}`  
 - Ligne 44 : `void k() throws Exception {System.out.println("Fille_k"); }`  
 - Ligne 36 : `private void e() {System.out.println("Fille_e");}`  
 - Ligne 41 : `int i() {System.out.println("Fille_i"); return 3; }`

On ne peut pas faire une redéfinition d'une méthode en modifiant la signature (type de renvoi, ou type d'exception jeter (si celle-ci est une exception Mère), visibilité). Cependant, ces erreurs n'empêchent pas la compilation dans l'état actuel du main () car ces méthodes ne sont tout simplement pas appelées.

```
mere.miage();
mereFille.miage();
```

La méthode miage() n'existe pas dans Mère et empêchent la compilation.

- 2) **Redéfinition** : deux méthodes de mêmes noms et de même profil dans 2 classes dont l'une hérite de l'autre.  
**Surcharge** : des méthodes de mêmes noms mais de profils différents dans une même classe. Choisi à la compilation selon les arguments.

Redéfinition : a, b, c, d, f, g, j, l, m.

Surcharge :

Dans la classe Fille : c(Mere mere) et c(Fille b). d() et d(Mere mere)

Dans la classe Mère : c(), c(Mere mere)

4)

```

Miage
Miage
Mere_a
Fille_a
Fille_a
Fille_a
Fille_b(Fille)
Mere_c
Fille_c(Mere)
Fille_c(Mere)
Fille_c(Mere)
Fille_c(Fille)
static Mere_d
static Mere_d
Mere_f
Mere_f
Fille_j
Mere_k
Fille_l
Fille_m

```

Ligne	Justification
55 fille.miage();	La méthode miage() affiche « miage() »
57 ((Fille)mereFille).miage();	mereFille est un objet initialisé Fille donc il peut utiliser miage()
59 mere.a();	La méthode a() de la classe Mère affiche « Mère_a »
60 mereFille.a();	La méthode a() de la classe Fille affiche « Fille_a » car mereFille est initialisé en tant que Fille.
61 fille.a();	fille est un objet référencié et initialisé fille et la méthode a() de la classe Fille affiche « Fille_a »
62 ((Mere)mereFille).a();	mereFille est « casté » en Mere mais reste instancié en Fille donc elle utilise a() la classe Fille et affiche « Fille_a »
63 mereFille.b(null);	mereFille est un objet instancié Fille donc il appelle b() de la classe Fille et affiche « Fille_b(Fille) »
65 mereFille.c();	mereFille est un objet instancié Fille mais c() (sans paramètre) est définie uniquement dans la classe Mère, donc il appelle c() de Mère et affiche « Mere_c »
66 mereFille.c(mere);	mereFille appelle c(Mère mère) dans Fille et affiche « Fille_c(Mère) »
67 mereFille.c(mereFille);	mereFille est instancié en tant que Fille donc l'appel c(mereFille) appelle la methode c(Mère mère) de fille et affiche « Fille_c(Mère ) »
68 mereFille.c(fille);	mereFille est référencé en tant que Mère donc l'appelé à c(Mère mère) de la classe Mère sera utilisé car l'objet fille extends la classe Mère. Donc l'affichage sera « fille_c(mère) ».
69 fille.c(fille);	L'objet fille est instancié et référencié en tant que Fille, donc c(fille) appel c(Fille fille) de la classe Fille et affiche "Fille_c(Fille)".
71 mere.d();	L'objet mère est instancié en tant que mère donc l'appel à d() affichera "static Mere_d".

72 mereFille.d();	L'objet mereFille est référencé en tant que Mère donc d() affichera "static Mere_d".
74 mere.printF();	printF() appelle f() et donc affiche "Mere_f" car mère est un objet référencé Mère
75 mereFille.printF();	printF() appelle f() et donc affiche "Mere_f" car mereFille est un objet référencé Mère
77 mereFille.j();	mereFille est instancié en tant que Fille donc elle va appeler en priorité la méthode j() de la classe Fille et affiché « Fille_j »
78 mereFille.k() ;	k() n'existe que dans Mère (supprimé à cause de la question 2) donc par héritage mereFille va appeler k() de la classe Mère.
79 mereFille.l();	mereFille est instancié en tant que Fille donc elle va appeler en priorité la méthode l() de la classe Fille et affiché « Fille_l »
80 mereFille.m();	mereFille est instancié en tant que Fille donc elle va appeler en priorité la méthode l() de la classe Fille et affiché « Fille_m »