Travaux Dirigés n°2

Java Avancé

—M1—

Redéfinitions, late binding, surcharge etc.

https ://classroom.github.com/a/6fhyeqaB

▸ Exercice 1. Redéfinition

```
class Mere {
  protected int meth() {
    return 42;
  }
  public void printMeth() {
    System.out.println(meth());
  }
}
class Fille extends Mere {
  public int meth() {
    return 24;
  }
}
class Main {
  public static void main(String[] args) {
    Mere mere = new Mere();
    System.out.println(mere.meth());
    mere.printMeth();

    Fille fille = new Fille();
    System.out.println(fille.meth());
    fille.printMeth();

    Mere mereFille = new Fille();
    System.out.println(mereFille.meth());
    mereFille.printMeth();
  }
}
```

1. Qu'affiche le main et pourquoi ?
2. S'il est dans `Fille`, à combien de méthodes `meth()` un objet de type `Fille` à accès (et comment il y accède) ? Et s'il est dans `Main` ?
3. Quel est le comportement si les méthodes `meth()` sont statiques ?
4. Et si meth sont maintenant des champs ? Pourquoi ?

```
class Mere {
  protected int meth=42;
  public void printMeth() {
    System.out.println(meth);
  }
}
class Fille extends Mere {
  public int meth = 24;
}
class Main {
  public static void main(String[] args) {
    Mere mere = new Mere();
    System.out.println(mere.meth);
    mere.printMeth();

    Fille fille = new Fille();
    System.out.println(fille.meth);
    fille.printMeth();

    Mere mereFille = new Fille();
    System.out.println(mereFille.meth);
    mereFille.printMeth();
  }
}
```

▸ Exercice 2. Redéfinition - surcharges

```
1  class Mere {
2    public void a() {System.out.println("Mere_a"); }
3    void b(Fille fille) {System.out.println("Mere_b(Fille)");}
4
5    void c() {System.out.println("Mere_c");}
6    void c(Mere mere) {System.out.println("Mere_c(Mere)"); }
7
8    static void d() {System.out.println("static Mere_d");}
9
10   protected void e() {System.out.println("Mere_e");}
11   private void f() {System.out.println("Mere_f");}
12   public void printF() { f(); }
13
14   Object g() {System.out.println("Mere_g"); return 2;}
15   int h() {System.out.println("Mere_h"); return 2;}
16   void i() {System.out.println("Mere_i");}
17
18   void j() throws Exception {System.out.println("Mere_j"); }
19   void k() throws IOException {System.out.println("Mere_k"); }
20   void l() throws Exception {System.out.println("Mere_l"); }
21   void m() throws Exception, ArrayIndexOutOfBoundsException {System.out.println("Mere_m"); }
22 }
23 class Fille extends Mere{
24   void miage() {System.out.println("Miage");}
25
26   public void a() {System.out.println("Fille_a"); }
27
28   protected void b(Fille fille) {System.out.println("Fille_b(Fille)");}
29
30   public void c(Mere mere) {System.out.println("Fille_c(Mere)");}
31   void c(Fille b) {System.out.println("Fille_c(Fille)"); }
32
33   static void d() {System.out.println("static Fille_d");}
34   static void d(Mere mere) {System.out.println("Fille_d(Mere)");}
35
36   private void e() {System.out.println("Fille_e");}
```

```
37   protected void f() {System.out.println("Fille_f");}
38
39   String g() {System.out.println("Fille_g"); return "c";}
40   char h() {System.out.println("Fille_h"); return 'c';}
41   int i() {System.out.println("Fille_i"); return 3; }
42
43   void j() throws IOException {System.out.println("Fille_j"); }
44   void k() throws Exception {System.out.println("Fille_k"); }
45   void l() {System.out.println("Fille_l");}
46   void m() throws IOException, IllegalArgumentException {System.out.println("Fille_m"); }
47 }
48 public class Main {
49   public static void main(String[] args) throws Exception {
50     Mere mere=new Mere();
51     Mere mereFille=new Fille();
52     Fille fille=new Fille();
53
54     mere.miage();
55     fille.miage();
56     mereFille.miage();
57     ((Fille)mereFille).miage();
58
59     mere.a();
60     mereFille.a();
61     fille.a();
62     ((Mere)mereFille).a();
63     mereFille.b(null);
64
65     mereFille.c();
66     mereFille.c(mere);
67     mereFille.c(mereFille);
68     mereFille.c(fille);
69     fille.c(fille);
70
71     mere.d();
72     mereFille.d();
73
74     mere.printF();
75     mereFille.printF();
76
77     mereFille.j();
78     mereFille.k();
79     mereFille.l();
80     mereFille.m();
81   }
82 }
```

1. Quelles sont les erreurs de compilation et pourquoi ?
2. Retirer les méthodes provoquant les erreurs.
3. Rappeler ce qu'est une redéfinition et une surcharge, et indiquer où sont les surcharges et où sont les redéfinitions ici.
4. Expliquer chaque affichage.

▸ Exercice 3. Expressions arithmétiques

On cherche à évaluer une expression arithmétique simple, représenté par un arbre. On veut un type commun `Expr` représentant des expressions arithmétiques, pouvant être soit de valeur réelle (`Value`) soit une opération d'addition (`Add`), permettant l'addition

entre deux expressions. On veut pouvoir évaluer la valeur d'une expression au moyen d'une méthode `eval()`.

Par exemple :

```
Expr val = new Value(1337.0);
System.out.println(val.eval()); //affiche 1337.0
Expr add = new Add(new Value(327.0), val);
System.out.println(add.eval()); //affiche 1664.0
Expr e = new Add(new Value(350.0), add);
System.out.println(e.eval()); //affiche 2014.0
```

1. Écrire les types `Expr, Value, Add`, les méthodes `eval` et une classe `Main` avec un Main de test dans un même package.
2. Implémenter l'affichage d'une expression arithmétique non évaluée.
3. Ajouter l'opération de multiplication.
4. Ajouter l'opération de racine carrée. Pour l'affichage, vous utiliserez le symbole unicode `\u221a`. Pour l'évaluation, vous utiliserez la méthode `Math.sqrt()`.