Licence Informatique **UE J1IN4W01**
Université de Bordeaux **Programmation 2**
2015-2016 **Travaux Dirigés - feuille 6**

Surcharge - Redéfinition - Algorithme de sélection

Pour chacune des applications qui suivent, donner, si possible, des exemples de surcharge, de redéfinition. Donner le résultat de leur exécution. Pour chaque appel de méthode, préciser le résultat de chacune des étapes de l'algorithme de sélection.

Exercice 1.

```
class A {
    public void f() {
        System.out.println("methode␣f,␣classe␣A");
    }
}

class B extends A {
    public void f() {
        System.out.println("methode␣f,␣classe␣B");
    }
}

public class Test1AlgoSelection {
    public static void main(String[] args) {
        A a = new A();
        a.f();

        B b = new B();
        b.f();

        A a1 = new B();
        a1.f();
    }
}
```

Exercice 2.

```
class A {
    public void f() {
        System.out.println("methode␣f,␣classe␣A");
    }
    public void g() {
        f();
    }
}

class B extends A {
    public void f() {
        System.out.println("methode␣f,␣classe␣B");
    }
}

public class Test2AlgoSelection {
    public static void main(String[] args) {
        A a = new A();
        a.g();
        B b = new B();
        b.g();
```

```
        A a1 = new B();
        a1.g();
    }
}
```

Exercice 3.

```
class A {
    public void f(A a) {
        System.out.println("methode f, classe A");
    }

    public void g(A a) {
        System.out.println("methode g, classe A");
    }
}

class B extends A {
    public void f(B b) {
        System.out.println("methode f, classe B");
    }

    public void g(A a) {
        System.out.println("methode g, classe B");
    }
}

public class Test3AlgoSelection {
    public static void main(String[] args) {
        A a = new A();
        A a1 = new A();
        A a2 = new B();

        B b = new B();
        B b1 = new B();

        a1.f(a);
        a1.f(a2);
        a1.f(b1);

        a2.f(a);
        a2.f(a2);
        a2.f(b1);

        b.f(a);
        b.f(a2);
        b.f(b1);

        a1.g(a);
        a1.g(a2);
        a1.g(b1);

        a2.g(a);
        a2.g(a2);
        a2.g(b1);

        b.g(a);
        b.g(a2);
        b.g(b1);
    }
}
```

Exercice 4. Qu'affiche l'application suivante ?

```
class Nourriture {

}

class Herbe extends Nourriture {

}

class Animal {
    public void mange (Nourriture n) {
        System.out.println("Je␣mange␣toute␣sorte␣de␣nourriture");
    }
}

class Lapin extends Animal{
    public void mange (Herbe h) {
        System.out.println("Je␣mange␣de␣l'herbe");
    }
}

public class AppliLapinMange{
    public static void main(String[] args) {
        Herbe h = new Herbe();
        Nourriture n = h;

        Lapin l = new Lapin();
        l.mange(h);
        l.mange(n);

        Animal a = l;
        a.mange(h);
        a.mange(n);
    }
}
```

Modifier le code de cette application afin d'obtenir le résultat correct suivant :

```
$ java AppliLapinMange
Je mange de l'herbe
Je mange de l'herbe
Je mange de l'herbe
Je mange de l'herbe
```

Exercice 5. Qu'affiche le programme suivant ? Expliquez.

```
class A{
    int i=1;
    int f(){
        return i;
    }
}

class B extends A{
    int i=5;
    int f(){
        return super.f()+i;
    }
}

class Test{
    public static void main(String[] args){
        A b=new B();
        System.out.println(b.i);
        System.out.println(b.f());
    }
}
```

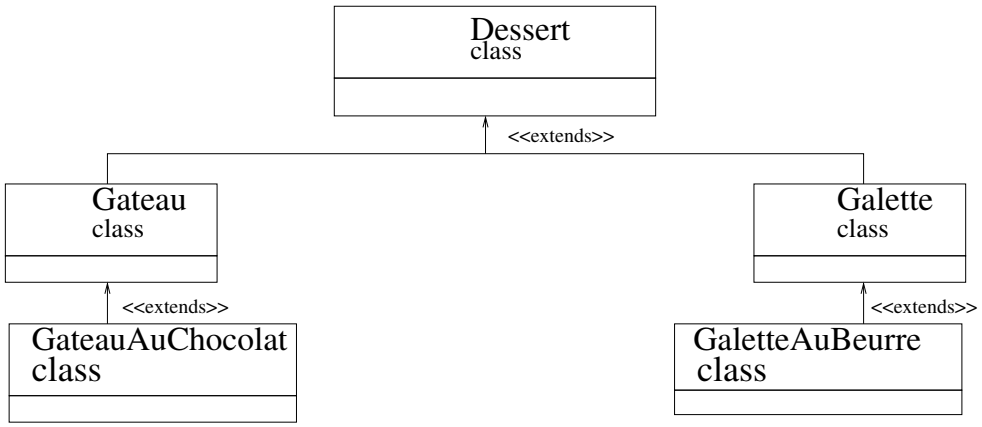

FIGURE 1 – Algorithme de sélection

Exercice 6. Soit la hiérarchie de types illustrée sur la Fig.1.
On suppose de plus qu'on a une classe `TestDessert` qui contient plusieurs méthodes de classe surchargées, qui prennent des combinaisons particulières de paramètres `Dessert` :

```
void gouter(Dessert d, Galette s){/* premiere forme */}
void gouter(Gateau c, Dessert d){/* seconde forme */}
void gouter(GateauAuChocolat cc, Galette g){/* troisieme forme */}
```

On considère les appels au `gouter` suivants dans la méthode `main` déclarée dans la classe `TestDessert` :

```
gouter(dessertRef, galetteRef);
gouter(gateauAuChocolatRef, dessertRef);
gouter(gateauAuChocolatRef, galetteAuBeurreRef);
gouter(gateauRef, galetteRef);
```

Indiquer pour chaque appel s'il est valide ?
Pour chaque appel valide indiquer quelle forme de `gouter` est invoquée ?