Surcharge, Redéfinition, Polymorphisme, Liaison tardive

Exercice 1 - Redéfinition, liaison tardive (late binding)

```
class A {
  int x = 1;

  public int getX() {
    return x;
  }

  static void printX(A a) {
    System.out.println("in A.printX");
    System.out.println("x " + a.x);
    System.out.println("getX() " + a.getX());
  }

  int m(A a) { return 1; }
}

class B extends A {
  int x = 2;

  public int getX() {
    return x;
  }

  static void printX(B b) {
    System.out.println("in B.printX");
    System.out.println("x " + b.x);
    System.out.println("getX() " + b.getX());
  }

  int m(B b) { return 2; }
}

public class Overridings {
  public static void main(String[] args) {
    A.printX(new A());     // 1
    //B.printX(new B());   // 2
    //A.printX(new B());   // 3
    A a = new B();
    //a.m(a);              // 4
  }
}
```

Expliquer, pour chacun des quatre appels numérotés //1, //2, //3 et //4, ce qui devrait être affiché, simplement à la lecture du code. Puis, pour chaque appel, exécuter le code pour confirmer (ou infirmer) votre intuition.
Note : on peut remarquer que en Java, on peut mettre plusieurs classes dans un fichier Java pourvu qu'une seule classe soit déclarée public ; on ne le refera plus car cela trouble tout le monde, IDEs compris. Dans ce cas, le nom du fichier .java doit être le même que le nom de la classe public.

Exercice 2 - Le fruit de votre labeur

On souhaite écrire un programme simple permettant de gérer la facturation d'une coopérative fruitière vendant des pommes bio. Une personne voulant acheter des pommes prend un panier, choisit plusieurs pommes et paye. Chaque pomme possède un poids, en grammes. Le prix d'une pomme, quelque soit sont type, est égal à son poids (mesuré en grammes) divisé par 2.

L'affichage d'un panier doit indiquer une pomme par ligne avec, pour chaque pomme, le type de pomme, le poids de la pomme, sa quantité (1 pour l'instant), puis une ligne indiquant le prix total du panier.

```
Golden 20 g x 1
Pink Lady 40 g x 1
price: 30
```

Il n'y a pas de taille maximale pour le panier. Vous utiliserez donc la classe `java.util.ArrayList` pour stocker les pommes.

1. On veut avoir une méthode `main` permettant les appels suivants :

```
public static void main(String[] args) {
  var apple1 = new Apple(20, "Golden");
  var apple2 = new Apple(40, "Pink Lady");

  var basket = new Basket();
  basket.add(apple1);
  basket.add(apple2);
  System.out.println(basket);
}
```

   Écrivez les classes `Apple` et `Basket` ainsi que leur méthodes nécéssaires.

2. Vérifiez que le code suivant marche aussi (le résultat doit être vrai); sinon, corrigez votre code :

```
public static void main(String[] args) {
  var set = new HashSet<Apple>();
  set.add(new Apple(20, "Golden"));
  System.out.println(set.contains(new Apple(20, "Golden")));
}
```

3. Finalement, la coopérative commence à être rentable. Il est donc temps de se diversifier en vendant aussi des poires. Une poire possède juste une valeur de 0 à 9 indiquant un facteur de jus. Le prix d'une poire est égal à 3 fois son facteur de jus.
   On veut donc que le `main` suivant fonctionne :

```
public static void main(String[] args) {
  var apple1 = new Apple(20, "Golden");
  var apple2 = new Apple(40, "Pink Lady");
  var pear = new Pear(5);

  var basket = new Basket();
  basket.add(apple1);
  basket.add(apple2);  // une pomme
  basket.add(pear);    // une poire
  System.out.println(basket);
}
```

   Écrivez et modifiez les classes et les méthodes nécéssaires.

4. On veut permettre d'ajouter plusieurs pommes ou plusieurs poires à notre panier d'un coup en indiquant un paramètre supplémentaire à la méthode `add`, indiquant la quantité de pommes ou de poires que l'on veut mettre dans le panier.
   L'affichage du panier devra indiquer la quantité de chaque pomme et poire :

```
public static void main(String[] args) {
  var apple1 = new Apple(20, "Golden");
  var apple2 = new Apple(40, "Pink Lady");
  var pear = new Pear(5);

  var basket = new Basket();
  basket.add(apple1, 5);      // 5 pommes
  basket.add(apple2);
  basket.add(pear, 7);        // 7 poires
```

```
    System.out.println(basket);
  }
```

Modifiez le code en conséquence, sachant que pour représenter une pomme (ou une poire) et une quantité, le plus simple est d'utiliser une table de hachage (aussi appelé dictionnaire) comme `java.util.HashMap` pour associer une pomme (ou une poire) à une quantité (une valeur entière).
Rappel: `HashMap.put()` permet de stocker des éléments, `HashMap.get()` ou `getOrDefault()` permet de sortir les élements, pour parcourir, on utilisera `for(var entry: map.entrySet()) { ... entry.getKey() ... entry.getValue() ... }`.

5. Enfin, au lieu de représenter les types de pomme par des `String`, on voudrait éviter les erreurs de saisie en representant les valeurs d'une énumération.

```
  public enum AppleKind {
    Golden, PinkLady, GrannySmith;
  }
```

Modifiez votre code pour que le code du `main` suivant fonctionne :

```
  public static void main(String[] args) {
    var apple1 = new Apple(20, AppleKind.Golden);
    var apple2 = new Apple(40, AppleKind.PinkLady);
    var pear = new Pear(5);

    var basket = new Basket();
    basket.add(apple1, 5);
    basket.add(apple2);
    basket.add(pear, 7);
    System.out.println(basket);
  }
```

Attention, l'affichage devra rester identique (avec l'espace entre Pink et Lady).
Note : en Java, on a le droit de redéfinir la méthode `toString()` d'un enum. Si, si, vous n'avez qu'à essayer !
