TD-TP : Objet et Référence

Objectif : mettre en évidence les problèmes liés à la différence entre l'objet et son adresse

1. Structure d'une classe

Etant donné le code de la classe `Ingredients` ci-joint (cf. sur eLearn en TP) donner la représentation de cette classe selon un schéma de classe UML

> *Dans chacune de vos futures productions de classes* `Java`*,*
> *de façon similaire au code de la classe* `Ingredients` *vous devrez*
> ***respecter systématiquement les zones de commentaires mises en évidence***

2. Différencier un objet de la référence à un objet

Etant donné le code `MainIncomplet.java` ci-joint, substituer chacune des sections de code étiquetées: `// xxx;` par le code correspondant aux directives données en commentaires.

En TP, une fois réalisés `MesPremiersPasEnJavaSousEclipse` décrits sur eLearn :

- créer le projet `1.ObjetEtReference` avec les fichiers `Ingredients.java` et `MainIncomplet.java` présents sur eLearn,
- réaliser les modifications dans le fichier `MainIncomplet.java` pour obtenir une exécution correcte.

```java
/**
 * Classe : Ingredients
 * @author: Lopistéguy Philippe
 * @date : jj/mm/aa
 **/

public class Ingredients {

// ATTRIBUTS
    // ATTRIBUTS de classe – NON -

    // ATTRIBUTS d'instance
        private String _libelle;
        private int _quantite;

// CONSTRUCTEURS complete la/les facon/s de creer un objet

        // sans parametre => initialise les attributs par defaut
        public Ingredients () {
                _libelle="";
                _quantite=0;
        }

        // un parametre par attribut => affecte les attributs
        public Ingredients(String l, int q) {
                set_libelle (l); set_quantite (q);
        }

        // constructeur par recopie (cf. 3.a)
        // => recopie/affecte les meme valeurs que celles de l'objet modele
        public Ingredients (Ingredients ingredientModele) {
                set_libelle(ingredientModele.get_libelle());
                set_quantite(ingredientModele.get_quantite());
        }

// METHODES D'ENCAPSULATION : get&set quantite, libelle

        // encapsulation de _quantite ===========
        public void set_quantite (int q) {
                _quantite = q;
        }
        public int get_quantite () {
                return _quantite;
        }

        // encapsulation de _libelle ===========
        public void set_libelle (String l) {
                _libelle = l;
        }
        public String get_libelle () {
                return (_libelle);
        }
```

```
// METHODES USUELLES : equals, toString, clone

        // Dit si les attributs de l'objet courrant et ceux de l'objet
        // reference par unIngredient ont les meme valeurs
        public boolean equals (Ingredients unIngredient) {
                boolean resultat;  // le resultat doit etre booleen
                resultat = (get_libelle() == unIngredient.get_libelle()
                                        &&
                                        get_quantite() == unIngredient.get_quantite());
                return resultat; // une fois etabli, on retourne le resultat booleen
        }

        // Solution alternative au constructeur par recopie (cf. la methode clone())
        public Ingredients dupliquer() {
                Ingredients laCopie;         // declare un pointeur
                laCopie = new Ingredients(); // cree un objet sur lequel pointe laCopie
                laCopie.set_libelle(get_libelle());
                laCopie.set_quantite(get_quantite());
                return laCopie;
                // tout ce code équivaut a :
                // return (new Ingredients(get_libelle(), get_quantite()));
        }

        // Produit une version String de l'objet
        public String toString() {
                String message;
                message = "_libelle ("+get_libelle()+"), _quantitie ("+get_quantite()+")";
                return message;
        }

    // METHODES SPECIFIQUES : afficher
        // Affice directement les valeurs de l'objet
        public void afficher() {
                System.out.println("je suis un ingredient "
                                        + get_libelle() + " "
                                        + get_quantite());
        }
}
```

```java
/**
 * Classe : Main
 * Objectif : Mettre en évidence les problèmes liés à la différence entre l'objet et son adresse
 *
 * 1. Créer un pointeur c'est différent que de créer un objet
 * 2. Cas de l'objet référencé par deux pointeurs
 * 3. Créer un objet à l'identique d'un objet modèle : constructeur par recopie
 *
 * 4. Comparer deux pointeurs versus comparer deux objets
 *  4.a Comparer 2 pointeurs
 *  4.b Comparer 2 objets => la méthode boolean equals(objetModele)
 *
 * 5. Paramètres des fonctions et méthodes : seule la valeur de la variable est transmise
 *  5.a Paramètre "type primitif"
 *      => modifications de la variable sans effet au niveau appelant
 *  5.b Paramètre "pointeur"
 *      => modification de la référence sans effet au niveau appelant
 *      => modification de l'objet référencé avec effet au niveau appelant
 * @version 2.0
 * @author Lopistéguy Philippe
 * @date jj/mm/aa
 */
public class Main {

/** 1.ATTRIBUTS              -non- **/
/** 2.CONSTRUCTEURS          -non- **/
/** 3.METHODES ENCAPSULATION -non- **/
/** 4.METHODES USUELLES      -non- **/

/** 5.METHODES SPECIFIQUES : modifierLeParametreDeTypePrimitifInt, modifierLeParametreAdresse,
     modifierObjetPointeParLeParametre **/

static public void modifierLeParametreDeTypePrimitifInt (int entierRecu) {
   System.out.println ("... je recois l'entier "+ entierRecu +" et l'augmente de 1");
   entierRecu++;  // incrémentation
   System.out.println ("... pour moi il vaut "+ entierRecu +" et la fonction se termine");
}

static public void modifierLeParametrePointeur (Ingredients ingredient) {
   System.out.println ("... je recois l'adresse de l'ingredient "+ingredient.toString());
   Ingredients unNouveau;
   unNouveau = new Ingredients ("vin", 40); // Crée unNouveau ingredient
   System.out.println ("... je cree un nouvel ingredient "+ unNouveau.toString());
   ingredient = unNouveau;
   System.out.println ("... le parametre recu pointe sur ce nouvel ingredient "+ingredient.toString()+
                       " et la fonction se termine");
}

static public void modifierObjetPointéParLeParametre (Ingredients ingredient) {
   System.out.println ("... je recois l'adresse de l'ingredient "+ingredient.toString());
   ingredient.set_quantite(0);
   System.out.println ("... je modifie la quantite "+ ingredient.toString()+" et la fonction se termine");
}

/** 6.METHODE PRINCIPALE: main() **/
public static void main(String[] args) {

    /* 1. Créer un pointeur c'est différent que de créer un objet
    * - un pointeur est capable de stocker l'adresse d'un objet
    * - un objet est créé par un 'new'
    * - un 'new' retourne l'adresse de l'objet créé
    * => on récupère l'adresse dans un pointeur
   */
   System.out.println("\n1. Creer un pointeur c'est different que de creer un objet");
   // **** // On crée le pointeur ingredient10,
   // **** // On crée l'objet <"lait",10> et le pointeur ingredient10 récupère l'adresse de l'objet <"lait",10>
   // **** // ON affiche ingredient10
```

```
/* 2. Cas de l'objet reference par deux pointeurs
 *  a. declarer 2 pointeurs
 *  b. creer un objet et garder son @ dans le 1er pointeur
 *  c. copier cette @ dans le 2eme pointeur
 *  d. afficher l'objet que pointeur le 1er et le 2eme pointeur
 *  e. modifier l'objet via le 2eme pointeur & constater la modification
 *  f. afficher l'objet pointe par chacun de ces 2 pointeurs
 * => les modifications de l'objet via le 1er pointeur ont affecté les valeurs
 *    de l'objet pointé par le 2eme pointeur. Normal ! c'est le même objet
 */
System.out.println("\n2. Cas de l'objet reference par deux pointeurs");
// **** // a. création d'un 1er pointeur ingredient21
// **** // a. création d'un 2eme pointeur ingredient22
// **** // b. le 1er pointe sur new objet <"sel", 21>
// **** // c. copier l'@ de l'objet dans le 2ème pointeur
// **** // d. affichage via 1er pointeur
// **** // d. affichage via 2ème pointeur
// **** // e. modifier l'objet via le 1er pointeur
// **** // f. affichage via 1er pointeur
// **** //    et 2ème pointeur

/* 3. Créer un objet à l'identique d'un objet modèle : constructeur par recopie
 *   a. écrire un constructeur avec un objet modèle en paramètre (cf. classe Ingredients)
 *   b. créer un 1er objet référencé par un 1er pointeur
 *   c. créer un 2eme objet (sur la base du 1er objet) référencé par un 2ème pointeur
 *   d. constater que les 2 pointeurs se réfèrent à des objets de même valeur
 *   e. modifier le premier objet
 *   f. constater que chaque pointeur se réfère à des objets de valeurs différentes
 * => Constructeur par recopie
 */
System.out.println("\n3. Creer un objet a l'identique d'un objet modele : constructeur par recopie");
// a. est déjà fait : soyez tranquilles...
// **** // b. crée le 1er pointeur    ingredient31
// **** // b. le 1er pointeur prend l'@ d'un ouvel objet <"milk", 31>
// **** // c. crée le 2ème pointeur   ingredient32
// **** // c. le 2ème pointeur prend l'@ du 2ème objet qui est construit
        //    sur la base du 1er objet, cad construit par recopie
// **** // d. affichage de ce que pointe    ingredient31
// **** // d. affichage de ce que pointe    ingredient32
// **** // e. modification de l'un des 2 objets
// **** // f. affichage de ce que pointent ingredient31 et ingredient32
```

```
    /* 4. Comparer deux pointeurs versus comparer deux objets
     *  4.a Commparer 2 pointeurs
     *      a. créer un objet référencé par un pointeur
     *      b. copier le pointeur dans un second pointeur
     *      c. observer qu'ils sont égaux (ils référencent le même objet)
     *  4.b Comparer 2 objets
     *      a. créer et initialiser un premier objet référencé par un 1er pointeur
     *      b. créer et initialiser (aux même valeurs) un second objet référencé par un 2ème pointeur
     *      c. comparer les pointeurs et constater qu'ils sont différentss alors que les 2 objets référencés sont
     *         identiques
     *      d. (c'est déjà fait) surcharger la méthode "static public boolean equals(objetModele)"
     *         cf. classe Ingredients, de sorte à ce que l'objet considéré compare ses attributs à ceux de
     *         l'objet modèle et retourne vrai en cas d'égalités
     */
    System.out.println("\n4. Comparer deux pointeurs versus comparer deux objets");
    System.out.println("   4.a Comparer deux pointeurs : pointeurs egaux");
    // **** // a. créer un objet <"salt", 41> pointé par ingredient41
    // **** // b. copier le pointeur dans un second pointeur ingredient42
    // **** // c. observer que ingredient41 et ingredien42 sont égaux

    System.out.println("\n   4.b Comparer deux objets => la methode boolean equals(objetModele)");
    // **** // a. créer un objet pointé par ingredient43 aux valeurs <"vin", 49>
    // **** // b. créer un 2ème objet pointé par ingredient44 aux mêmes valeurs <"vin", 49>
    // **** // c. différence d'adresse des objets, mais mêmes valeurs des objets identiques, cf. methode equals

     /* 5. Paramètres des fonctions et méthodes : c'est la valeur de la variable qui est transmise
      *   5.a Paramètre "type primitif" => modifications de la variable sans effet au niveau appelant
      */
    System.out.println("\n5. Paramètres des fonctions : c'est la valeur de la variable qui est transmise");
    System.out.println("     5.a Paramètre primitif => modification de la variable sans effet au niveau appelant");
    int unEntier = 20;                                              // avant = 20
    System.out.println ("La valeur de unEntier "+unEntier+" est transmise a la fonction");
    modifierLeParametreDeTypePrimitifInt (unEntier);                // après = toujours 20
    System.out.println ("La valeur de unEntier "+unEntier+" et est INCHANGE apres la fonction");

    //  5.b Paramètre "pointeur" => modification de la référence sans effet au niveau appelant
    System.out.println("\n     5.b Parametre pointeur => modification de la reference sans effet au niveau appelant");
    Ingredients ingredient00 = new Ingredients ("huile", 10);       // avant <huile,10>
    System.out.println ("La valeur de l'ingredient "+ingredient00.toString()+" est transmise à la fonction");
    modifierLeParametrePointeur (ingredient00);                     // après <huile,10> inchangé !!!!
    System.out.println ("La valeur de l'ingredient "+ingredient00.toString()+" apres la fonction INCHANGE");

    //  5.c Paramètre "pointeur" => modification de l'objet référencé avec effet au niveau appelant
    System.out.println("\n     5.c Parametre pointeur => modification de l'objet avec effet au niveau appelant");
    Ingredients ingredient11 = new Ingredients ("the", 11);         // avant <thé,10>
    System.out.println ("La valeur de l'objet pointe "+ingredient11.toString()+" est transmise a la fonction"); //
    modifierObjetPointéParLeParametre (ingredient11);               // aprés <vin,40> inchangé !!!!
    System.out.println ("La valeur de l'objet pointe "+ingredient11.toString()+" apres la fonction C H A N G E");
  }
}
```