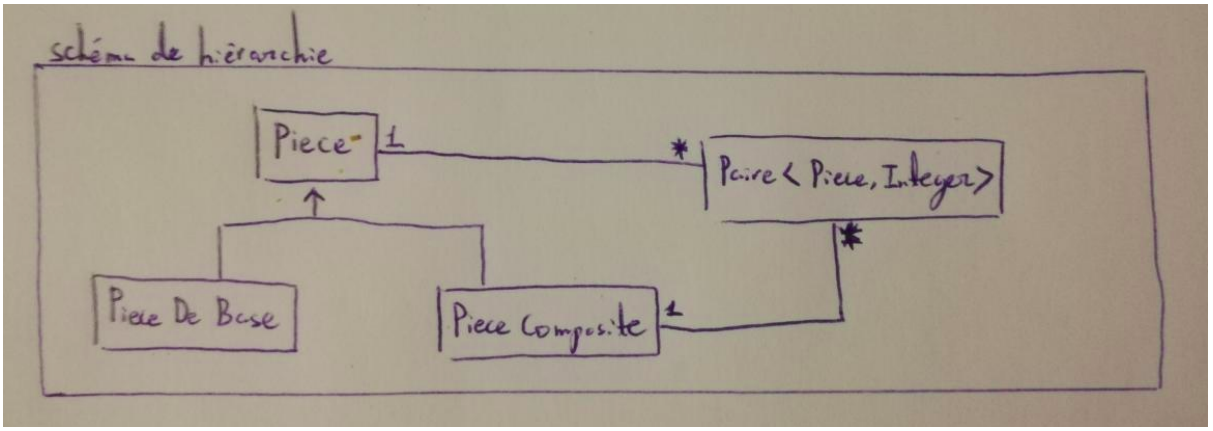
Nomenclature de pièces

Tous les exemples sont faite avec test1.nom donné dans moodle

Exercice 1 -Hiérarchie de classes

Question 1:

Schéma de hiérarchie(Vérifié par le professeur)



Question2:

1)Pièce est une classe type “abstract”, tous ses attributs sont privés.
Donc on initialise des getteurs et des setteurs .
2)On doit rédefinir toString,equals,compareTo si on veut utiliser .

Pièce_Base et Pièce_Composite héritent de Pièce .
Et Pièce_composite contient un LinkedList <Pair<Pièce,Integer>>
De syntaxe :
“private LinkedList<Pair<Piece, Integer>> TabPiece =
new LinkedList<Pair<Piece, Integer>>();”

Question3:

1)Comme les trois paramètres sont hérité, on fait un constructeur, et on utilise la méthode “super()”

```
public Piece_composite(int reference, String String, float poids) {
    super(reference,String,poids);
    this.TotalPoids = poids;
}
```

2)On peut redéfinir toString dans la classe “Pièce”, puis j’utiliser la méthode “toString” de Pièce par super.toString();

```
public String toString() {
    return super.toString()+"\n";
}
```

Question 4:

1)La constructeur de Pièce composite:

```
public Piece_composite(int reference, String String, float poids) {
    super(reference,String,poids);
    this.TotalPoids = poids;
}
```

Dans laquelle il contient une table vide de pièce:

private LinkedList<Pair<Piece, Integer>> TabPiece

2)POur la méthode ajout:

```
// Methodes
public void add_Pair(Piece a, int i) throws ExceptionContaineSelf {
    if(this.equals(a) ) { throw new ExceptionContaineSelf();}
    else if (a instanceof Piece_composite && Piece.est_composante((Piece)this,(Piece_composite)a)) {
        throw new ExceptionContaineSelf();
    }
    TabPiece.add(new Pair<Piece, Integer>(a, i));
    TotalPoids += a.get_poids() * i;
    TotalPiece ++;
}
```

ExceptionContaineSelf:Une pièce composite ne peut pas contenir elle- même .

Si a est la pièce “this” lui même , ou si a est une pièce composite , et “this” est déjà une pièce de a , alors on lance une exception.

Sinon on l’ajoute ce couple(Pièce+Occurrence) dans notre tableau Pair de LindkeyList.

Aussi on incrémente “TotalPiece”, et initialise le “TotalPoids”。

Question 5:

D’abord, dans la classe Pièce, je redéfinie la méthode toString:

```
public String toString()
{
    return "Réference:"+this.get_reference()+",Denomination "+this.get_Denomination()+", poids:"+this.get_poids();
}
public boolean equals(Piece o)
```

Puis je redéfinie toString dans classe composite:

```
public String toString() {
    return super.toString()+"\n";
}
```

Enfin, je définie la méthode “affiche_tous_les_sous_composants(int decal)”

```
public String affiche_tous_les_sous_composants(Piece a, int decal) {
    StringBuffer LC = new StringBuffer();
    String decaleEspace = "";

    for (int i = 0; i < decal; i++) {
        decaleEspace += " ";
    }
    for (Pair<Piece, Integer> e : TabPiece) {
        LC.append(decaleEspace + decaleEspace + e.getPiece().get_Denomination() + ",Quantité:" + e.getInteger()
                + ",Poids:" + e.getPiece().get_poids() + "\n");
        if (e.getPiece() instanceof Piece_composite) {
            // LC.append(affiche_tous_les_sous_composants(e.getPiece(),2*decal));
            Piece_composite c = (Piece_composite) e.getPiece();
            LC.append(c.affiche_tous_les_sous_composants(c, 2 * decal));
        }
    }
    return LC.toString();
}
```

1)StringBuffer :Une String modifiable.

2)decaleEspace:initialement="", puis à chaque appel récursive , on change le nombre d'espaces .
3)Ici , on a utilisé une fonction récursive , car chaque pièce composite peut avoir une forme différente .
4)Tous les retours de toString sont sauvegarde dans StringBuffer.

Un exemple de retour pour l'option 6(Détaille d'une pièce) qui utilise affiche tous:

```
Réference:20,Denomination meuble, poids:11.0
  vif,Quantité:5,Poids:10.0
```

Question 6:

La question correspond à l'option 2 dans la nomenclature:

```
public void case2_add_pièce() throws ExceptionContaineSame
{
    System.out.println("On va ajouter l'existance d'une pièce dans votre liste des pièces ,1.Piece basique ou 2.composé");
    if(sc.nextInt()==1) {
        int i,j;
        String s;
        System.out.println("Entrez la réference  de la pièce");
        i=sc.nextInt();
        System.out.println("Entrez le nom  de la pièce");
        s=sc.next();
        System.out.println("Entrez le poid  de la pièce");
        j=sc.nextInt();
         this.add_piece( new Piece_Base(i,s,j));
    }
    else {
        int i;
        float j;
        String s;
        System.out.println("Entrez la réference  de la pièce");
        i=sc.nextInt();
        System.out.println("Entrez le nom  de la pièce");
        s=sc.next();
        System.out.println("Entrez le poid  de la pièce");
        j=sc.nextFloat();
         this.add_piece(new Piece_composite(i,s,j));
    }
}
```

On demande à l'utilisateur s'il veut ajouter une pièce composite ou basique . Puis selon le choix d'utilisateur, on va instancier base/Composite. Puis on utiliser add_pièce pour ajouter cette pièce dans la tableau des pièces .

Question 7:

J'utilise un getteur pour afficher le pods d'une pièce :

```
@Override
public float get_poids() {
    // TODO Auto-generated method stub
    return this.poids;
}
```

Puis, je vais la mettre dans toString:

```java
public String toString()
{
    return "Réference:"+this.get_reference()+",Denomination "+this.get_Denomination()+", poids:"+this.get_poids();
}
public boolean equals(Piece a)
```

Exercice 2-Détection d'erreurs.

Question 1:

cas1: P1 possède la pièce p1 dans sa liste de pièce

On aura une boucle infinie, car on a une fonction récursive

cas2:On aura la même problème:Une boucle infinie des appels

Question 2:

```java
static public boolean est_composante(Piece a, Piece_composite b)
{
    for(Pair<Piece, Integer> i:b.getTabPiece()) {
        if(i.getPiece().equals(a)) {
            return true;
        }
        else if(i.getPiece() instanceof Piece_composite) {
            i.getPiece();
            if(Piece.est_composante(a, (Piece_composite)i.getPiece())==true) {
                return true;
            }
        }
    }
    return false;
}
```

1)C'est une fonction récursive

2)D'abord on parcourt la TabPiece de b , pour vérifier si a se trouve dedans ou pas .

Puis on parcourt tous les piece composite dans TabPiece de b

Et on vérifier si a se trouve dedans ou pas.

Ainsi on fait des boucles .

Question 3)Exception traitée à l'extérieur .

1)D'abord on définit la classe Exception

```java
public class ExceptionContaineSelf extends Exception {

    public ExceptionContaineSelf()
    {
        super("\"Error:l'objet ne peut pas contenir lui meme!\"");
    }
```

2)On utilise la fonction "est_composante" définie ci-dessus.

```java
// Methodes
public void add_Pair(Piece a, int i) throws ExceptionContaineSelf {
    if(this.equals(a) ) { throw new ExceptionContaineSelf();}
    else if (a instanceof Piece_composite && Piece.est_composante((Piece)this,(Piece_composite)a)) {
        throw new ExceptionContaineSelf();
    }
```

Et chaque fois qu'on veut ajouter une pièce , on vérifie d'abord avec add_Pair.

Exemple:

```
Voulez ajouter 1.Piece basique ou 2.compose?
2
Entrez la reference de piece composite(Principale):
20
Entrez la reference  de la piece a composite(a ajouter)
20
Entrez le nombre  de la piece
5
Exception in thread "main" mini_projet.ExceptionContaineSelf: "Error:l'objet ne peut pas contenir lui meme!"
        at mini_projet.Piece_composite.add_Pair(Piece_composite.java:18)
        at mini_projet.Nomenclature.case5_ajouter_composante_a_une_piece(Nomenclature.java:154)
        at mini_projet.Main.Menu(Main.java:50)
        at mini_projet.Main.main(Main.java:10)
```

Ex3-Nomenclature

Question 1)

On crée une classe Nomenclature .

```
public void add_piece(Piece a) throws ExceptionContaineSame
{
    for(Piece e:TabPieces) {
        if(e.get_reference()==a.get_reference()) {
            throw new ExceptionContaineSame();
        }
    }
    TabPieces.add(a);
    this.total_pièce++;
}
```

D'abord on vérifie si la pièce nouvelle existe déjà dans la nomenclature ou pas .
Si oui , on lance une Exception
Sinon , on l'ajoute .

Question 2)

```
public String toString(){
    StringBuffer LC =new StringBuffer("La Nomenclature :"+this.name +", contient au total: "+this.total_pièce+" pièces,dont :\n");
    for(Piece e:TabPieces) {
        LC.append("Piece:"+e.get_Denomination()+"\n");
    }
    return LC.toString();
}
```

1-J'utilise un StringBuffer;
2-Et je parcours la TabPieces , et j'ajoute consécutivement les informations dans StringBuffer.

Question 3)

On crée une nouvelle classe d'Exception:

```
package mini_projet;

public class ExceptionContaineSame extends Exception {

    public ExceptionContaineSame() {
        super("Cette pièce existe déja!");
    }
}
```

Question 4)

```
    public Piece Chercher_pièce(int reference) {
        for(Piece e:TabPieces) {
            if(e.get_reference()==reference) return e;
        }
        return null;
    }
```

1-J'utilise un boucle for

2-J'utilise un getteur sur reférence pour récupérer la pièce.

Question 5)

```
public String pièce_composite_correspondantes(Piece a) {
    StringBuffer LP=new StringBuffer("Les pièces composites contiennent cette pièce sont:");
    for(Piece e:TabPieces) {
        if(e instanceof Piece_composite) {
            if(Piece.est_composante(a, (Piece_composite)e)) {
                LP.append(e.get_Denomination());
            }
        }
    }

    return LP.toString();
}
```

1-J'utilise un StringBuffer
2-Je parcours TabPieces, si je rencontre une pièce composite, je vérifie si la pièce a se trouve dans cette pièce ou pas .
Si oui , j'ajoute le nom de cette pièce dans LP.

Question 6)

```
public boolean sup_piece(Piece a) {
    for(Piece e:TabPieces) {
        if(e instanceof Piece_composite) {
            if(Piece.est_composante(a, (Piece_composite)e)) {
                return false;
            }
        }
    }
    TabPieces.remove(a);
    return true ;
}
```

1-Je parcours la TabPieces
2-si e une pièce composite de TabPieces , et si a est une pièce qui compose e , alors on ne peut pas supprimer a.
Sinon on peut supprimer a.

Question 7)

```java
public static int Menu() throws NumberFormatException, ExceptionContaineSelf, ExceptionContaineSame, FileNotFoundException {
    //On donne un nom, puis on créer une nomenclature
    Scanner sc=new Scanner(System.in);
    System.out.println("Saisir le nom de votre Nomenclature.");
    Nomenclature thisN=new Nomenclature(sc.next());
    System.out.println("Votre nom est :"+thisN.name +"\n");
    //J'initialise qlq pièces dans nomenclature
    thisN.add_piece(new Piece_Base(10,"vif",10));
    thisN.add_piece(new Piece_Base(13,"vif",10));
    thisN.add_piece(new Piece_composite(20,"meuble",11));
    //J'affiche la phrase de commencement
    System.out.println("0-Exit\n1-Nouvelle nomenclature\n2-Ajouter une piéce sans ces composants\n3-Afficher la nomenclature\n4-List
            + "7-Supprimer une piéce\n8-trier(Par réference)\n9-Afficher les pièces de bases\n10-Sauvegarder\n11-Lire une nomenclatu
    System.out.println("Entrer la valeur correspondant aux opération desiré:");
    int a=9;
    while(!(a==0||a==1)) {
        switch(a=sc.nextInt()) {
            case 0:System.out.println("Merci, a bientot!");return 0;
            case 1:{
                System.out.println("On va créer une nouvelle nomenclature");
                return 1;
            }
            case 2:{
                thisN.case2_add_pièce();
                break;
            }
            case 3:{
                System.out.println(thisN.toString()+"\n");
                break;
            }
```

1-Scanner sc=new Scanner(System.in);
sc.next();
Ceci permet de saisir des données par Clavier

2-si a==0 : c'est la case 0: On sort de la nomenclature
si a==1: c'est la case 1: on créer une nouvelle nomenclature
RQ: ici, créer une nouvelle nomenclature=réinitialiser tous les attributs

3-On utilise un switch pour chaque option possible.

4-On n'oublie pas un default dans switch :

```java
            default:{
                System.out.println("L'option que vous demandez n'existe pas ");
            }
```

Exercice4-Pièces particulières

Question 1)

D'abord, je crée la classe Lampe :

```java
public class Lampe extends Piece implements Piece_fragile{
    float puissance;
    public Lampe(int reference,String denomination,float poids,float puissance)
    {
        super(reference,denomination,poids);
        this.puissance=puissance;
    }
```

Hérité de la classe Piece, et implements de la classe Piece_Fragile

De même , on crée la classe Cadres:

```java
4
5 public class Cadres extends Piece_composite {
6     LinkedList<Piece_fragile> TabPieces= new LinkedList<Piece_fragile>();
7     public Cadres(int reference,String denomination,float poids) {
8         super(reference,denomination,poids);
9     }
10 }
11
```

J'ajoute une LinkedList<Piece_fragile> TabPieces pour sauvegarder des cadres et des plaques de verres.

Question2)

Donc la solution que je propose est d'ajouter une interface "Piece_fragile"

```
package mini_projet;

public interface Piece_fragile {

}
```

Car en Java , on peut mettre tous les classes implémentées d'une classe A dans un tableau de type A

Puis dans la classe Nomenclature, on ajoute une tableau de type Piece_fragile.

```
LinkedList<Piece> TabPieces= new LinkedList<Piece>();
LinkedList<Piece_fragile> TabPieces_fragile= new LinkedList<Piece_fragile>();
```

Exercice 5- Gestion de la nomenclature

Question 1)

D'abord je dois rédefinir "toCompare" dans la classe Pièce, car "Collection.sort " utilise cette fonction pour trier.

-1 si plus petit, 0 si égau, et 1 si plus grand

```
public int compareTo(Piece e) {
    if(this.get_reference()<e.get_reference()) return -1;
    else if(this.get_reference()==e.get_reference()) return 0;
    else return 1;
}
```

Puis je définie ma fonction case8_trier_piece

```
public String case8_trier_piece()
{
    Collections.sort(this.TabPieces);
    StringBuffer PB= new StringBuffer("Les pièces(trié par réference) sont:\n");
    for(Piece e:this.TabPieces) {
        PB.append("nom :"+e.get_Denomination()+" reference:"+e.get_reference()+"\n");
    }
    return(PB.toString());
}
```

Collections.sort(this.TabPieces) m'aide à classer les objets selon leurs références.

Une fois triée, je vais afficher les pièces un après l'autre grâce à une StringBuffer.

Voici un exemple avec test1.nom:

```
Entrer la valeur correspondant aux opération desiré:
11
Entrer le nom de fichier

test1.nom
L'opération est faite,qu'est ce que vous voulez faire maintenant?
8
Les pieces(trie par reference) sont:
nom :planche reference:19
nom :pied reference:21
nom :planche2 reference:29
nom :boulon reference:34
nom :barre reference:37
nom :vase reference:43
nom :table reference:56
nom :planche reference:77
nom :tabouret reference:88

L'opération est faite,qu'est ce que vous voulez faire maintenant?
```

Question2)

L'idée est "instanceof",on parcourt la TabPieces et on vérifier si la pièce est "instanceof Piece_base"

```
public String case9_piece_base()
{
    StringBuffer PB= new StringBuffer("Les pièces de base sont:\n");
    for(Piece e:this.TabPieces) {
        if(e instanceof Piece_Base) {
        PB.append("nom :"+e.get_Denomination()+" reference:"+e.get_reference()+"\n");
        }
    }
    return(PB.toString());
}
```

Un exemple avec test1.nom:

```
L'opération est faite,qu'est ce que vous voulez faire maintenant?
9
Les pieces de base sont:
nom :planche reference:19
nom :planche2 reference:29
nom :boulon reference:34
nom :barre reference:37
nom :vase reference:43
nom :planche reference:77

L'opération est faite,qu'est ce que vous voulez faire maintenant?
```

Exercice 6-Chargement et sauvegarde d'une nomenclature

Question1)

Pour sauvegarder la nomenclature

```
public void case10_sauvegarder() throws FileNotFoundException
{
    PrintWriter writer2 = new PrintWriter(this.name+".txt");
    writer2.println(this.name +" "+  this.total_pièce);
    for(Piece e:this.TabPieces) {
        writer2.println(e.get_reference()+" "+e.get_Denomination()+" "+e.get_poids() );
        if(e instanceof Piece_composite) {
            Piece_composite t=(Piece_composite)e;
            for(Pair<Piece, Integer> r:t.getTabPiece()) {
                writer2.println("    "+r.getPiece().get_reference()+" "+r.getInteger());

            }
        }
        writer2.println("    -1 -1");
    }
    writer2.close();
    System.out.println("Votre Nomenclature est bien sauvergardé\n");
}
```

1)PrintWriter writer2=new PrintWriter(this.name+".txt");

Ceci me permet de créer un nouveau document, avec un nom saisit par clavier.

2)La deuxième ligne de code enregistre d'abord le nom de nomenclature, et nombre total des pièces .

3)writer2.println(e.get_reference()+" "+e.get_Denomination()+" "+e.get_poids());

Enrégistre tous les pièces existend dans cette nomenclature.

Et chaque fois on vérifie si c'est une pièce composite , si oui , avec:

```
for(Pair<Piece, Integer> r:t.getTabPiece()) {
    writer2.println("    "+r.getPiece().get_reference()+" "+r.getInteger());
```

On va afficher tous ses composantes.

Et enfin on ajoute writer2.println(" -1 -1"); pour la fin de chaque pièce dans nomenclature

Question2)

Pour lire une nomenclature, j'ai pensé à une telle méthode:

D'abord , j'initialise les deux tableaux pour sauvegarder les données.

```
this.TabPieces=new  LinkedList<Piece>();
this.TabPieces_fragile=new LinkedList<Piece_fragile>();
```

Puis je saisis le nom du fichier à lire

```
System.out.println("Entrer le nom de fichier\n");
String str=sc.next();
```

J'initialise un lecteur

```
java.io.File fic = new java.io.File(str);
Scanner lecteur = new Scanner(fic);
```

Et je vais lire ligne par ligne

Les mots sont séparés par un espace .

```
String[] temp;
temp= lecteur.nextLine().split(" ");
```

J'utilise un boucle While avec la condition :Tant qu'il y a encore des lignes .

Je lis une ligne , les mots sont séparés par un espace .

Si dans ce ligne contient 3 mots, on passe à l'étape suivant

Si je lis la ligne suivant , si la ligne suivant est" -1 -1" , alors c'est une pièce base, et on l'ajoute dans TabPiece

Sinon on l'ajoute dans TabPiece en tant qu'une piece_composite

```
while(lecteur.hasNextLine()) {
    temp= lecteur.nextLine().split(" ");
    if(temp.length==3) {
        String[] temp2=lecteur.nextLine().split(" ");
        if(Integer.parseInt(temp2[4])==-1 && Integer.parseInt(temp2[5])==-1) {
            this.add_piece(new Piece_Base(Integer.parseInt(temp[0]),temp[1],Float.parseFloat(temp[2])));
        }
        else {
            Piece_composite pc=new Piece_composite(Integer.parseInt(temp[0]),temp[1],Float.parseFloat(temp[2]));
            this.add_piece(pc);
        }
    }
 }
lecteur.close();
```

Je réinitialise le lecteur , car je veux mettre le pointeur dans le début du fichier texte. cette fois ci je vais ajouter les composantes de chaque Piece_composite.

La première lecteur = new Scanner(fic); est pour sauter la première ligne , qui a le nom de nomenclature et le nombre totale des pièces .

La deuxième lecteur = new Scanner(fic); est pour sauter tous les pièces qui sont déjà ajouté dans TabPieces lors du premier boucle while .

Et à partir de troisieme lecteur = new Scanner(fic); tant que la ligne n'est pas " -1 -1" , il veut dire que c'est des composantes d'une pièce composite , donc on utilise add_Pair pour ajouter cette couple(piece,nombre d'occurence) dans la pièce composite

```
lecteur = new Scanner(fic);
temp= lecteur.nextLine().split(" ");
while(lecteur.hasNextLine() ) {
    temp= lecteur.nextLine().split(" ");
    Piece p1=this.Chercher_piece(Integer.parseInt(temp[0]));
    temp= lecteur.nextLine().split(" ");
    while(Integer.parseInt(temp[4])!=-1 && Integer.parseInt(temp[5])!=-1) {
        Piece_composite p2=(Piece_composite) p1;
        p2.add_Pair(this.Chercher_piece(Integer.parseInt(temp[4])), Integer.parseInt(temp[5]));
        temp= lecteur.nextLine().split(" ");
    }
 }
lecteur.close();
```

Un exemple d'option 3 sur test1.nom une fois chargé:

```
L'opération est faite,qu'est ce que vous voulez faire maintenant?
3
La Nomenclature :yyy, contient au total: 9 pieces,dont :
Piece:planche
Piece:pied
Piece:planche2
Piece:boulon
Piece:barre
Piece:vase
Piece:table
Piece:planche
Piece:tabouret
```

Exercice 7-Tests expérimentaux

Question 1)

Initialement j'ai rencontré pleins des problèmes ,
Mais après des améliorations, il marche bien sur test1.nom et test2.nom

Question 2)

Pour charger un fichier de grande instance, ça m'a pris quelques minutes(3 à 4 minutes) . Or pour sauvegarder une nomenclature ça m'a pris quelques secondes voire immédiat
Car quand on sauvegarder une nomenclature , il va juste mettre des informations ligne par lignes .Or pour lire une nomenclature, il y a trop des vérifications (Voire s'il y a des erreurs , comparées , et parcours deux fois pour ajouter des composantes d'une piece_composite)