BOURGEOIS Charles
StJoSup, Le Havre

SIO2

Compte-rendu

Polymorphisme

2022

Sommaire :

1) Le polymorphisme ? C’est quoi ?

Il est important dans un programme de savoir référencer différents types d’objets, ces variables permettant cela s’appellent des variables polymorphes.

Elles pourront référencer des objets de leur type ou des classes qui ont héritées de cette dernières (leurs sous-classe).

Dans notre cas personnel sur les robots et les robots de combat, le polymorphisme pourrait permettre de référencer un robot de combat ou bien un robot qui n’aurait rien à faire la … comme un robot farceur !

Le polymorphisme fait intervenir ce que l’on appelle des méthodes virtuelles, permettant d’établir un lien dynamique entre le type de l’objet référencé et non son type déclaré, c’est une liaison dynamique. Nous mettons virtuelle devant la fonction de base et override devant la fonction qui est référencée de cette dernière.

2) La définition des méthodes sous forme algorithmique

Reprenons le programme Produit du compte-rendu héritage :

```
10 références
class Produit
{
    protected string reference;
    protected string designation;
    protected int prixdevente;
    2 références
    public Produit(string referenceSai, string designationSai, int prixdeventeSai)
    {
        this.reference = referenceSai;
        this.designation = designationSai;
        this.prixdevente = prixdeventeSai;
    }
```

Puis dans notre méthode Affiche(), nous rajoutons le mot *virtual :*

```
6 références
public virtual void Affiche(Produit pdt)
{
    Console.WriteLine("REFERENCE DU PRODUIT : " + this.reference + "\n DESIGNATION : " + this.designation + "\n PRIX DE VENTE : " + th
}
```

Maintenant dans notre méthode Affiche de la sous-classe PdtAch, nous mettons le mot *override* étant la méthode qui référence :

```
5 références
public override void Affiche(Produit pdt)
{
    base.Affiche(pdt);
    Console.WriteLine("REFERENCE : " + this.reference + "\n DESIGNA
}
```

Dans notre programme de test, nous n'avons plus qu'à déclarer qu'un produit Pdt correspond à un produit PdtAch, puis nous référençons soit PdtAch ou PdtFab pour afficher la méthode que l'on souhaite :

```
Console.WriteLine("Affichage d'un produit fabriqué");
Pdt.Affiche(pf);
Console.WriteLine("Affichage d'un produit acheté :");;
Pdt.Affiche(pa);
Console.WriteLine("\n");
```

Dans ce cas nous référençons PdtAch.

Nous avons donc pour illustrer cela la fonction Cout() et Marge () qui chacune référence une autre entre plusieurs objets :

Dans la classe Produit :

```
public virtual double Cout()
{
    return 1000;
}


public double marge()
{
    return prixdevente - Cout();
}
```

Dans la classe PdtAch :

```
2 références
public override double Cout()
{
    return prixAchat + 100;
}
```

Selon la méthode que l'on souhaiteras référencer, le programme s'adaptera.

Cela est identique pour notre exemple personnel sur les robots, cependant voici à quoi cela pourrait ressembler avec l'arrivée d'un robot farceur nommé Valentin..

```
3 références
class RobotFarceur : Robot
{
    private string typeBlague;
    private string marrant;

    1 référence
    public RobotFarceur(string nomRobotSai, int nbBrasRobotSai, int nbJambesRobotSai, string couleurRobotSai, string typeBlagueSai, st
    {
        this.typeBlague = typeBlagueSai;
        this.marrant = marrantSai;
    }
```

Nous avons notre classe RobotFarceur.

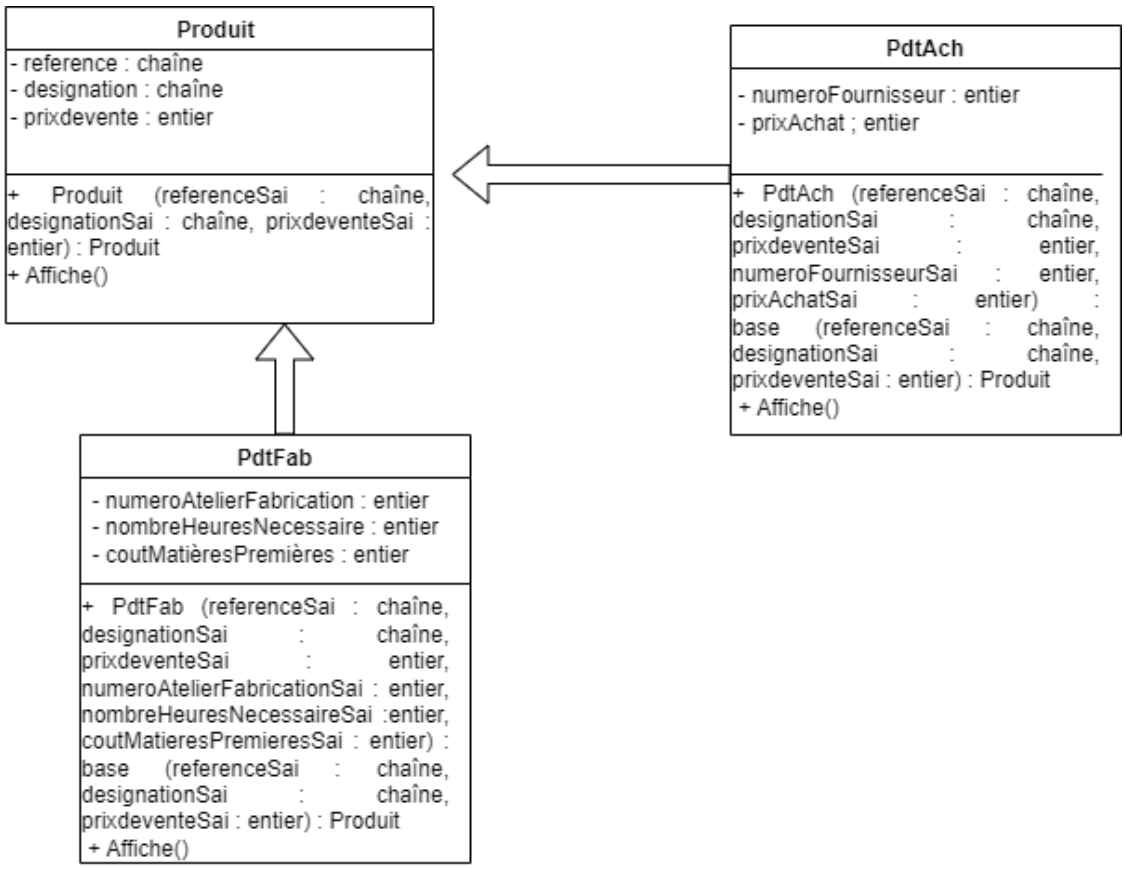
Dans notre programme de test nous référençons ce robot après avoir référencé et ajouter notre robot de combat dans l'atelier :

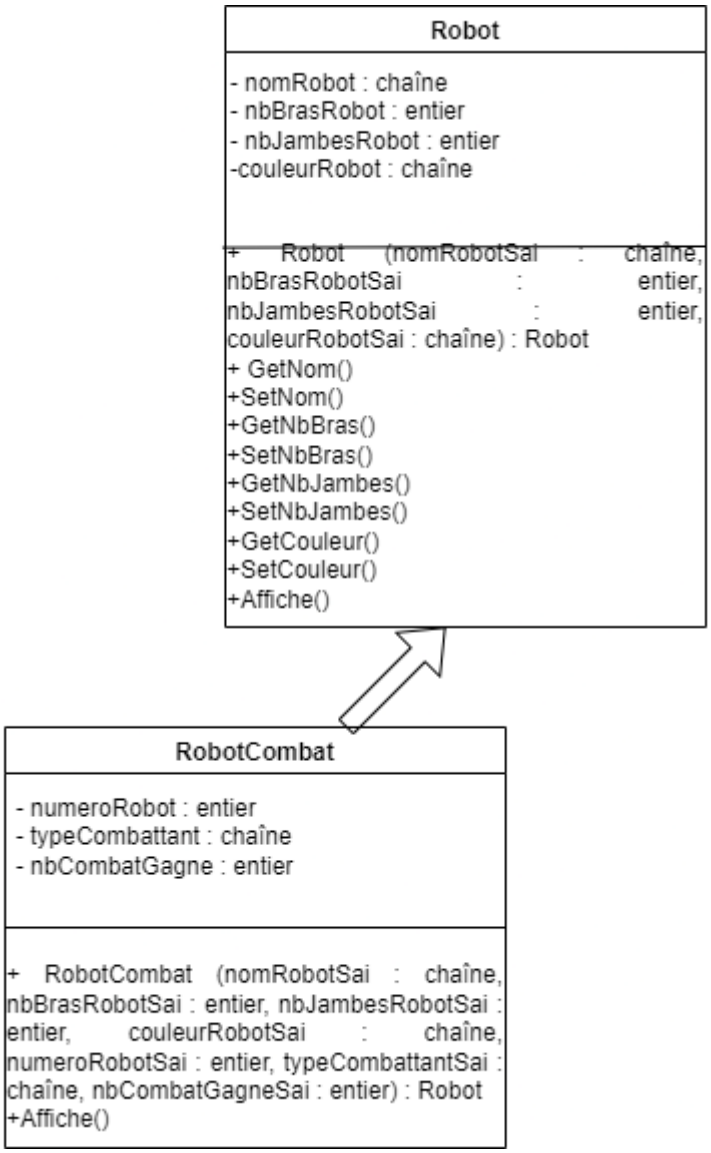
```
robot.Affiche(rbtCombat);
atelier.AjtRobot(rbtCombat);
Console.WriteLine("\n Oh un robot farceur s'est introduit dans l'atelier");
robot.Affiche(rbtFarceur);
atelier.AjtRobot(rbtFarceur);
atelier.Affiche();
```

Valentin s'ajoute donc à notre atelier !

3) Le diagramme de classe UML

Le diagramme se trouve être similaire au diagramme de l'héritage :




Robot
- nomRobot : chaîne
- nbBrasRobot : entier
- nbJambesRobot : entier
-couleurRobot : chaîne
+ Robot (nomRobotSai : chaîne, nbBrasRobotSai : entier, nbJambesRobotSai : entier, couleurRobotSai : chaîne) : Robot
+ GetNom()
+SetNom()
+GetNbBras()
+SetNbBras()
+GetNbJambes()
+SetNbJambes()
+GetCouleur()
+SetCouleur()
+Affiche()
RobotCombat
- numeroRobot : entier
- typeCombattant : chaîne
- nbCombatGagne : entier
+ RobotCombat (nomRobotSai : chaîne, nbBrasRobotSai : entier, nbJambesRobotSai : entier, couleurRobotSai : chaîne, numeroRobotSai : entier, typeCombattantSai : chaîne, nbCombatGagneSai : entier) : Robot
+Affiche()

4) Conclusion

Le polymorphisme est utile notamment avec le pas à pas, pour se retrouver dans les différents référencements qu'il permet.

Nous avons donc des méthodes qui afficheront l'objet référencé grâce à des méthodes virtuelles, ce qui est pratique lorsque l'on manipule beaucoup d'objets du même type.