BOURGEOIS Charles
SIO2
StJoSup, Le Havre

Compte-rendu

Héritage

2022

Sommaire

1) Qu’est-ce que l’héritage ?

La notion d’héritage est indispensable à connaître en programmation orientée objets, en effet elle permet un l’usage de définir les termes de « sous-classes » ou « classe-fille » crées à partir d’une classe.

Par exemple, prenons un robot, ce robot aura un nombre défini de bras, une couleur etc… Créons maintenant un robot de combat, la classe « Robot Combat » va hériter de la classe « Robot » donc reprendre tous ses attributs et ses méthodes.

La classe Robot Combat sera donc une classe issue de la classe de « base » qui est Robot.

Cependant la classe fille ne pourra pas hériter des données membres de la classe mère. Cela provoque une erreur de compilation. Pour remédier à cela il est nécessaire d’ajouter le mot ***protégé***, cela permet d’accéder aux données à partir de la classe mère ou fille.

Il est aussi possible dans la classe fille de rajouter des attributs et des méthodes qui ne sont pas dans la classe mère de base.

L’héritage permet donc de créer différentes nouvelles classes à partir d’une classe d’origine.

2) La définition des méthodes sous formes algorithmiques

a) Produit

Prenons le programme Produit, avec une classe Produit, Produit Acheté et Produit Fabriqué.

Voici les attributs et le constructeur de la classe Produit :

```
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;
    }
```

Nous avons ajouté « Protected » pour chaque attribut.

Nous avons ajouté une méthode Affiche :

```
4 références
public void Affiche(Produit pdt)
{
    Console.WriteLine("REFERENCE DU PRODUIT : " + this.reference + "\n DESIGNATION :
}
```

Passons à la classe Produit Acheté :

```
3 références
class PdtAch : Produit
{
    private int numeroFournisseur;
    private int prixAchat;
```

PdtAch : Produit signifie que la classe PdtAch hérite de la classe Produit.

Dans le constructeur, nous indiquons que les attributs de la classe Produit sont bien hérités dans cette nouvelle classe :

```
référence
ublic PdtAch (string referenceSai, string designationSai, int prixdeventeSai, int numeroFournisseurSai, int prixAchatSai):base (referenceSai, designa
```

Nous avons rajouté dans la déclaration du constructeur *base :* avec les attributs de la classe Produit.

Ajoutons aussi une méthode Affiche dans cette classe :

```
0 références
public void Affiche()
{
    base.Affiche();
    Console.WriteLine("REFERENCE : " + this.reference + "\n DESIGNATION : " + this.designation + "\n PRIX DE VENTE : " + this.prix
}
```

Nous avons notre méthode Affiche ainsi que la méthode Affiche de la classe mère, elle est indiquée par *base.Affiche()*.

C’est la même chose pour la classe Produit Fabriqué.

b) Perso

Nous avons notre classe Robot avec ses attributs et son constructeur :

```
class Robot
{
    protected string nomRobot;
    protected int nbBrasRobot;
    protected int nbJambesRobot;
    protected string couleurRobot;


    public Robot(string nomRobotSai, int nbBrasRobotSai, int nbJambesRobotSai, string couleurRobotSai)
    {
        this.nomRobot = nomRobotSai;
        this.nbBrasRobot = nbBrasRobotSai;
        this.nbJambesRobot = nbJambesRobotSai;
        this.couleurRobot = couleurRobotSai;
    }
```

*Toujours avec le mot **protected**.*

Nous avons maintenant notre classe RobotCombat qui hérite de la classe Robot :

```
3 références
class RobotCombat : Robot
{
    private int numeroRobot;
    private string typeCombattant;
    private int nbCombatGagne;
```

```
int nbCombatGagneSai ):base(nomRobotSai, nl
```

Puis nous créons une méthode Affiche avec *base.Affichet()* :

```
1 référence
public void Affiche(Robot rbt)
{
    base.Affiche(rbt);
    Console.WriteLine("NUMERO ROBOT : " + this.nu
}
```

3) La classe PDO

Petite intermède, parlons de la classe PDO.

La classe PDO en PHP permet d'accéder et de communiquer avec n'importe quelle base de données. Il suffit de l'instancier de la manière suivante dans une variable :

```php
<?php

$connexionBDD = new PDO ('mysql:host=localhost;dbname=clicksmoviz;charset=utf8', 'root',
```

Voici les différentes méthodes de la classe PDO : `class PDO {`

```
/* Méthodes */
public __construct(
    string $dsn,
    ?string $username = null,
    ?string $password = null,
    ?array $options = null
)
public beginTransaction(): bool
public commit(): bool
public errorCode(): ?string
public errorInfo(): array
public exec(string $statement): int|false
public getAttribute(int $attribute): mixed
public static getAvailableDrivers(): array
public inTransaction(): bool
public lastInsertId(?string $name = null): string|false
public prepare(string $query, array $options = []): PDOStatement|false
public query(string $query, ?int $fetchMode = null): PDOStatement|false
public query(string $query, ?int $fetchMode =
PDO::FETCH_COLUMN, int $colno): PDOStatement|false
public query(
    string $query,
    ?int $fetchMode =                                    PDO::FETCH_CLASS,
    string $classname,
    array $constructorArgs
): PDOStatement|false
public query(string $query, ?int $fetchMode =
PDO::FETCH_INTO, object $object): PDOStatement|false
public quote(string $string, int $type = PDO::PARAM_STR): string|false
public rollBack(): bool
public setAttribute(int $attribute, mixed $value): bool
}
```

4) La classe Object

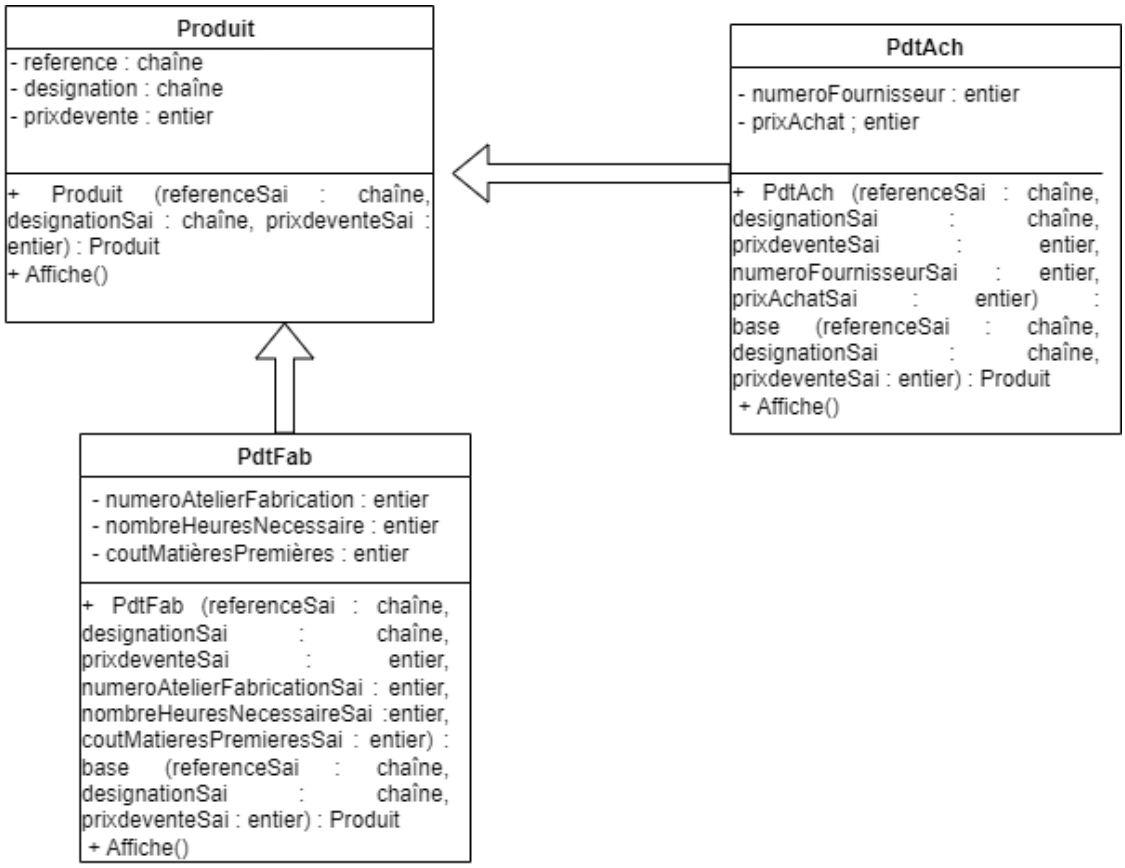
Chaque méthode, programme hérite de la classe Object. Toutes nos méthodes « de base » sont issues de la classe object.

Voici certaines méthodes de la classe objet :

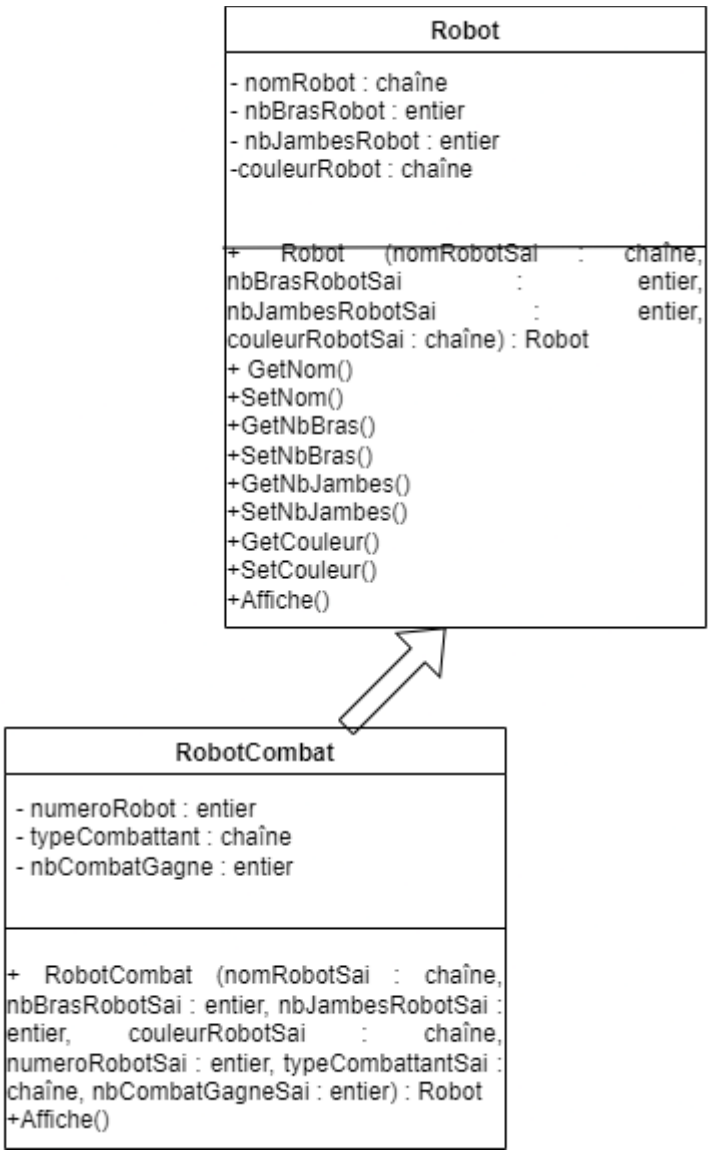
- Equals - Prend en charge les comparaisons entre les objets.
- Finalize - Effectue des opérations de nettoyage avant qu'un objet ne soit automatiquement récupéré.
- GetHashCode - Génère un nombre correspondant à la valeur de l'objet pour prendre en charge l'utilisation d'une table de hachage.
- ToString - Fabrique une chaîne de texte lisible par l'homme qui décrit une instance de la classe.

5) Le diagramme de classe UML

PRODUIT :



PERSO :


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()

6) Conclusion

Nous avons donc acquis la notion d’héritage, qui se révèle essentielle à la P.O.O notamment dans l’utilisation de classe mère et de classe fille.

Nous pouvons donc créer à partir d’une seule classe, différentes sous classes permettant d’aller encore plus loin dans la programmation.