SEQUENCE 3 – HERITAGE
SEANCE 5 - REVISIONS

OBJECTIFS

Revoir et synthétiser les principes d'héritage, d'abstraction et de polymorphisme.

ICI, ON UTILISE DES CONSOLE.WRITELINE DANS LES CLASSES POUR METTRE EN EVIDENCE DES NOTIONS !

EXERCICE 1 : SURDEFINITION DE METHODE

Dessinez le graphe d'héritage puis indiquez ce qu'affiche le programme

```
class A
    {
        public virtual void  Affiche ()
        { Console.WriteLine("Je suis un A"); }
    }
class B : A  { }
class C : A
    {
        public override void Affiche()
        { Console.WriteLine("Je suis un C"); }
    }
class D : A
    {
        public override void Affiche()
        {   base.Affiche();
            Console.WriteLine("Je suis un D");
        }
    }
class E : A
    {
        public override void Affiche()
        {   Console.WriteLine("Je suis un E");
            base.Affiche();
        }
    }
class F : B    { }
class G : D
    {
        public override void Affiche()
        {   base.Affiche();
            Console.WriteLine("Je suis un G");
        }
    }
class H : B
    {
        public override void Affiche()
        {   base.Affiche();
            Console.WriteLine("Je suis un H");
        }
    }
class I : C
    {
        public override void Affiche()
        { Console.WriteLine("Je suis un I"); }
    }
class J : C
    {
        public override void Affiche()
        {   base.Affiche();
            Console.WriteLine("Je suis un J");
```

```
class Program
    {
      static void Main(string[] args)
        {
            A a = new A();
            a.Affiche();
            B b = new B();
            b.Affiche();
            C c = new C();
            c.Affiche();
            D d = new D();
            d.Affiche();
            E e = new E();
            e.Affiche();
            F f = new F();
            f.Affiche();
            G g = new G();
            g.Affiche();
            H h = new H();
            h.Affiche();
            I i = new I();
            i.Affiche();
            J j = new J();
            j.Affiche();
        }
    }
```

```
        }
    }
```

EXERCICE 2 : SURDEFINITION DE CONSTRUCTEUR

1. **Le programme ci-dessous affiche :**
   Je construis le A
   Je construis le B
   Pourquoi ?

```
class A {
  public A()
   { Console.WriteLine("Je construis le A "); }
 }
class B : A  {
  public B()
   { Console.WriteLine("Je construis le B "); }
 }
```

```
static void Main(string[] args)
    {
      B b = new B();
    }
```

2. **Quelles valeurs vont contenir les objets p1, p2 et p3 ?**

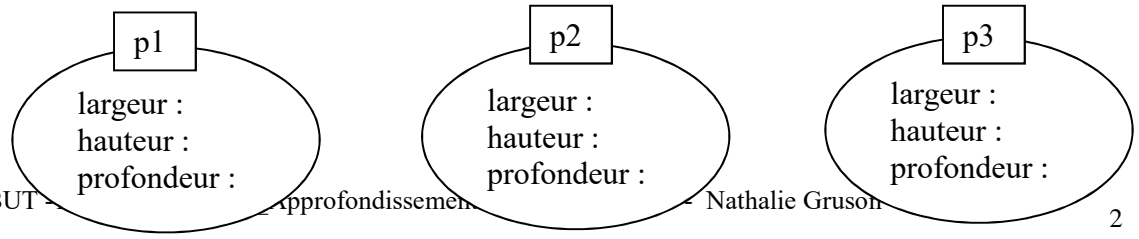
```
class Rectangle
   {
    private int largeur;
    private int hauteur;
    // il y a aussi les propriétés Largeur, Hauteur

    public Rectangle(): this (1,1) { }
    public Rectangle (int largeur, int hauteur)
      { this.Largeur = largeur;
        this.Hauteur = hauteur; }
    }

class Parallelepipede : Rectangle
    {
     private int profondeur ;
     // il y a aussi la propriété Profondeur

      public Parallelepipede()
         { this.Profondeur = 1; }
      public Parallelepipede (int largeur, int hauteur, int profondeur)
         { this.Profondeur = profondeur; }
      public Parallelepipede (int val) : base(val,val)
         { this.Profondeur = val; }
    }
```

```
static void Main(string[] args)
    {
     Parallelepipede  p1 = new Parallelepipede();
     Parallelepipede  p2 = new Parallelepipede(60,20,30);
     Parallelepipede  p3 = new Parallelepipede(30);
    }
```



EXERCICE 3 : RETROUVEZ DES ERREURS

1. **Il y a une erreur dans un constructeur de la classe Rectangle : corrigez-le sans toucher à la classe Carre.**

```
class Carre
 {
  private int largeur;

  public int Largeur
    {
     get { return this.largeur; }
     set { this.largeur = value; }
    }

  public Carre(int largeur)
      { this.Largeur = largeur; }
  }
```

```
class Rectangle : Carre
 {
   private int longueur;

   public int Longueur
     {
       get { return this.longueur; }
       set { this.longueur = value; }
     }

   public Rectangle(int longueur, int largeur)
                    : base(largeur)
       { this.Longueur = longueur; }

  public Rectangle()
       { this.Longueur = 0; }

   }
```

2. **Il y a un warning concernant `CalculeSurface`. Quelle(s) rectification(s) faut-il apporter ?**

```
class Carre
  {
      private int largeur;
      public int Largeur
      {
        get { return this.largeur; }
        set { this.largeur = value; }
      }

    public double CalculeSurface()
  { return this.Largeur * this.Largeur; }
    }
```

```
class Rectangle : Carre
 {
   private int longueur;
    public int Longueur
    {
         get { return this.longueur; }
         set { this.longueur = value; }
    }

    public double CalculeSurface()
    { return this.Largeur * this.Longueur; }
 }
```

3. **Il y a une erreur concernant `ToString` de la classe Rectangle. Quelle(s) rectification(s) faut-il apporter ?**

```
class Carre
{
    private int largeur;
    public int Largeur
     {
      get { return this.largeur; }
      set { this.largeur = value; }
     }
  public override string ToString()
 { return "Largeur " + this.Largeur; }
}
```

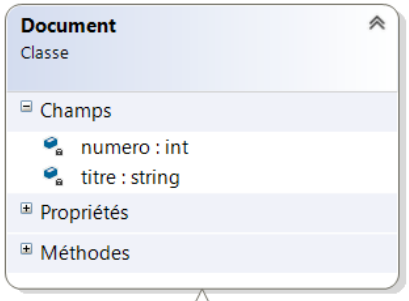
```
class Rectangle : Carre
{
    private int longueur;
    public int Longueur
    {
         get { return this.longueur; }
         set { this.longueur = value; }
    }

   public override string ToString()
    { return this.ToString()
        + "Longueur "+ this.Longueur; }
 }
```

4. Les constructeurs de la classe Revue sont incorrects. Pourquoi ?

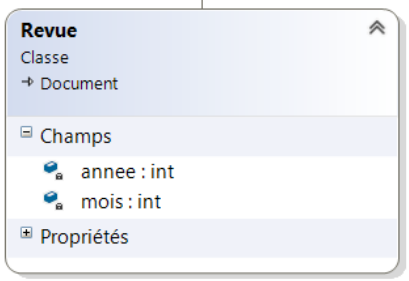
```
class Document
{

  private int numero;
  private String titre;
  public Document(int numero, String titre)
   {
     this.Numero = numero;
     this.Titre = titre;
   }
```




```
class Revue : Document
{
  private int mois;
  private int annee;

  public Revue( String titre, int mois, int annee)
              : base (titre)
   {
    this.Mois = mois;
    this.Annee = annee;
   }
 public Revue(int numero, String titre, int mois, int annee) :base (titre, numero)
        {
            this.Mois = mois;
            this.Annee = annee;
        }
```



EXERCICE 4 :

1. **On peut écrire :**

`Document d1 = new Livre(123, "Les Fourmis", "Werber");` `Document d2 = new Revue(30058, "Capital", 3, 2020);`	Vrai / Faux
`Livre l1 = new Document(145, "L'empire des anges");` `Revue r1 = new Document(30059, "Capital");`	Vrai / Faux

2. Complétez le code ci-dessous :

```
List<Document> liste = new List<Document>();
liste.Add(new Document(1, "La banquise"));
liste.Add(new Livre(123, "Les Fourmis", "Werber"));
liste.Add(new Revue(30058, "Capital", 3, 2020));

foreach ( _________________ element in liste)
    Console.WriteLine(element);
```

3. Expliquez le résultat obtenu :

Program.cs	Résultat

`Document d= new Document(1, "La banquise");`
`Livre l = new Livre(1, "La banquise", "Werber");`
`Console.WriteLine("d==l =>" + (d.Equals( l)));`
`Console.WriteLine("l==d =>" + (l.Equals( d)));`
	`d==l =>True`
	`l==d =>False`

Document	Livre

`public override bool Equals(object obj)`
`{`
`return obj is Document document &&`
`this.Numero == document.Numero &&`
`this.Titre == document.Titre;`
`}`
	`public override bool Equals(object obj)`
	`{`
	`return obj is Livre livre &&`
	`base.Equals(obj) &&`
	`this.Auteur == livre.Auteur;`
	`}`

4. Expliquez pour quelle raison, on aurait

Document	Livre

`public override bool Equals(object obj)`
`{`
`return (obj.GetType() == this.GetType())&&`
`this.Numero == ((Document)obj).Numero &&`
`this.Titre == ((Document)obj).Titre;`
`}`
	`public override bool Equals(object obj)`
	`{`
	`return (obj.GetType() == this.GetType())&&`
	`base.Equals(obj) &&`
	`this.Auteur == ((Livre)obj).Auteur;`
	`}`

5. Quel est le résultat de l'exécution du code ci-dessous à l'affichage ?

```
class A
{   public virtual void SayHello()
        { Console.WriteLine("Hello A");}
}
class B : A
{   public override void SayHello()
            { Console.WriteLine ("Hello B");}
 }
class Program
{
  public static void Main(String[] args)
  {
  List<A> l = new List<A>( ) ;
  l.Add(new A());
  l.Add(new A());
  l.Add(new B());
  foreach ( A unElement in l )
       unElement.SayHello();
  }
}
```

6. **Ce code provoque une erreur de compilation. Expliquez et veuillez trouver une solution.**

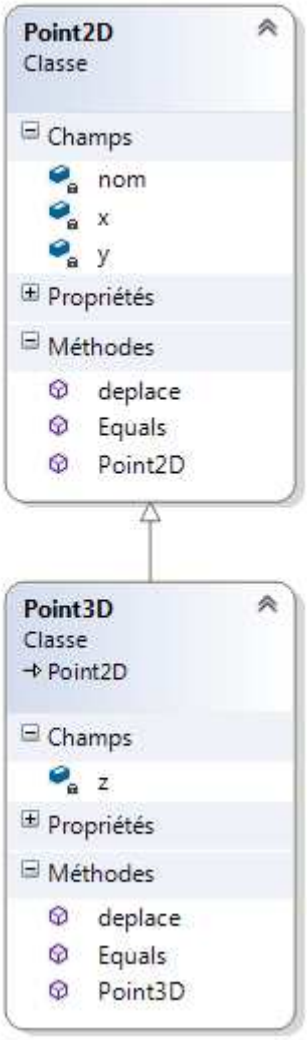
```
class A
{
   public void TraitementA(){Console.WriteLine
("traitement A");}
}
class B : A
{
   public void TraitementB(){Console.WriteLine
("traitement B");}
 }
class Program
{
  public static void main(String[] args)
  {
  List<A> l = new List<A>( ) ;
  l.Add(new A());
  l.Add(new B());
  foreach ( A unElement in l )
    {
      if (unElement is A)
         unElement.TraitementA();
      if (unElement is B)
         unElement.TraitementB();
  }
}
```

EXERCICE 5 :

```
class Point2D
{
    private char nom;
    private int x;
    private int y;

    // il y a aussi les propriétés ..

   public Point2D(char nom, int x, int y)
    {
        this.Nom = nom;
        this.X = x;
        this.Y = y;
    }
    public void Deplace(int pasX, int pasY)
    {
        this.X += pasX;
        this.Y += pasY;
    }
    public override bool Equals(object obj)
    {
        if (obj == null)
            return false;
        if (obj.GetType() != this.GetType())
            return false;
        Point2D p = (Point2D)obj;
        return p.X == this.X
              && p.Y == this.Y && p.Nom == this.Nom;
    }
```



1. **On decide de surcharger le constructeur de Point2D complétez le code ci-dessous :**

```
public Point2D(char nom) : ……
 { }
```

2. **Complétez le code des constructeurs Point3D:**

```
public Point3D(char nom, int x, int y, int z ) : ……

 {  ……                                                    }

public Point3D(char nom) : ……

 { }
```

3. **Complétez le code de la méthode Deplace. Expliquez pourquoi il n'y a pas besoin de mettre le mot clef virtual dans la classe Point2D, ni override dans Point3D**

```
  public void Deplace(int pasX, int pasY, int pasZ)
  {
            ……

            ……
  }
```

EXERCICE 6 :

1. Pourquoi fait-on des classes abstraites ?
2. Une classe abstraite a-t-elle forcément des méthodes abstraites ?

EXERCICE 7 :

Les Pokémons sont des gentils animaux qui sont passionnés par la programmation objet en général et par le polymorphisme en particulier. On veut pouvoir gérer une liste de Pokemons, pouvoir afficher la vitesse de chaque pokemon de la liste, on veut pouvoir trier les pokemons par rapidité, et encore plein d'autres fonctionnalités.
Pour chacune des quatre catégories de pokémons, on désire disposer d'une méthode toString qui retourne (dans une chaîne de caractères) les caractéristiques du pokémon.

En fonction des détails donnés ci-dessous : dans un 1er temps définissez le diagramme de classe. Indiquez les classes et les méthodes qui seront abstraites. Puis codez.

Il existe quatre grandes catégories de pokémons :
- **Les pokémons sportifs** : Ces pokémons sont caractérisés par un nom, un poids (en kg), un nombre de pattes, une taille (en mètres) et une fréquence cardiaque mesurée en nombre de pulsations à la minute. Ces pokémons se déplacent sur la terre à une certaine vitesse que l'on peut calculer grâce à la formule suivante : vitesse = nombre de pattes * taille * 3
- **Les pokémons casaniers** : Ces pokémons sont caractérisés par un nom, un poids (en kg), un nombre de pattes, une taille (en mètres) et le nombre d'heures par jour où ils regardent la télévision. Ces pokémons se déplacent également sur la terre à une certaine vitesse que l'on peut calculer grâce à la formule suivante : vitesse = nombre de pattes * taille * 1.5
- **Les pokémons des mers** : Ces pokémons sont caractérisés par un nom, un poids (en kg) et un nombre de nageoires. Ces pokémons ne se déplacent que dans la mer à une vitesse que l'on peut calculer grâce à la formule suivante : vitesse = poids / 25 * nombre de nageoires
- **Les pokémons de croisière** : Ces pokémons sont des pokémons des mers qui vont 2 fois plus vite.