

## Exercice 1 :

- Ecrire une classe **Personne** permettant de décrire complètement une personne, sachant que l'on souhaite avoir autant d'informations que dans la phrase suivante :  
*"M. Holly Pierre est né en 1965, il est célibataire."*
- Ajouter un constructeur à la classe **Personne**.
- Ajouter, à la classe **Personne**, une méthode **retourneInfos**. Cette méthode doit retourner une chaîne de caractères similaire à la phrase donnée dans l'énoncé de l'exercice.
- Ecrire un programme qui déclare 3 variables de type **Personne**, les instancie et affiche les informations les concernant.
- Ajouter une méthode **age** qui renvoie l'âge de l'individu en fonction d'une année donnée en paramètre.
- Ajouter à la classe **Personne** un attribut **conjoint** et examiner les conséquences que cela peut avoir sur l'ensemble du code.  
*Attention: la polygamie et la polyandrie sont interdites :-)*
- Ajouter une méthode **marier(Personne p)** qui permet de marier une personne à une autre. Modifier la méthode **retourneInfos** de façon que le nom (traditionnel de la femme devienne). Pour simplifier : quand une femme se marie son nom devient : *"[nom de l'époux] née [nom de jeune fille]"*, par exemple : si Mlle Durant se marie avec M. Dupond, son nom deviendra Mme "Dupond née Durant"

## Exercice 2 :

- Donner la sortie du programme Java suivant et commenter très brièvement.

```
class Truc {
    public Truc() {
        System.out.println (« ++ normal »);
    }
    public void finalize() {
        System.out.println ( « -- normal »);
    }
    public static void main(String args[]) {
        Truc x;
    }
}
```

- Même question avec :

```
public static void main(String args[]) {
    Truc x = new Truc();
}
```

- Même question avec :

```
public static void main(String args[]) {
    Truc x = new Truc();
    x.finalize();
}
```

## Exercice 3 :

- Donner la sortie du programme Java et expliquer :

```
class Less {
    public int n ;
    public Less(int x) { n = x ; }
    public String toString() { return "Less ! " ; }
    public void test1() {
        System.out.println("Less.test1 : " + this.toString());
    }
    public void test2() {
        System.out.println("Less.test2 : " + this.n);
    }
}

class More extends Less{
    public int n ;
    public More(int x, int y) { super(x) ; this.n = y ; }
    public String toString() { return "More ! " ; }
    public void test1() {
        super.test1() ;
        System.out.println("More.test1 : " + this.toString() + super.toString());
    }
    public void test2() {
        super.test2() ;
        System.out.println("More.test2 : " + this.n + super.n);
    }
    public static void main(String argv[]) {
        More m = new More(1, 2) ;
        m.test1() ;
        m.test2() ;
        Less n = new Less(0) ;
        n.test1() ;
        n.test2() ;
    }
}
```

## Exercice 4 :

Donner la sortie du programme suivant :

```
import java.io.*;
class A {
    public A monA ;
    public void maMethode() { imprimer(" A ") ; }
    public void saMethode() { monA.maMethode(); }
    public void imprimer(String s) { System.out.println(s) ; }
}

class B extends A{
    public C monC ;
    public void saMethode() { monC.maMethode() ; }
}

class C extends A {
    public B monB ;
    public void maMethode() { imprimer(" C ") ; }
    public void saMethode() { monB.maMethode() ; }
}
```

```

class D extends B{
    public E monE;
    public void maMethode() { imprimer(" D ") ; }
    public void saMethode() { monE.maMethode() ; }
}

class E extends B {
    public D monD ;
    public void maMethode() { imprimer(" E ") ; }
}

class M {
    public static void main(String argv[]) {
        A a = new A();
        B b = new B();
        C c = new C();
        D d = new D();
        E e = new E();
        a.monA = b;
        b.monA = c;
        c.monA = d;
        d.monA = e;
        e.monA = a;
        b.monC = c;
        c.monB = b;
        d.monC = c;
        e.monC = c;
        d.monE = e;
        e.monD = d;
        a.maMethode();
        b.maMethode();
        c.maMethode();
        d.maMethode();
        e.maMethode();
        a.saMethode();
        b.saMethode();
        c.saMethode();
        d.saMethode();
        e.saMethode();
    }
}

```

## Exercice 5 :

Soit l'interface et les deux classes suivantes :

```

public interface AppareilRéfrigérant {
    public boolean EstActif();
    public float TempératureActuelle();
    public float TempératureIdéale();
}

class Frigo implements AppareilRéfrigérant {
    private float températureActuelle;
    private final float températureIdéale = 5.0f;
    private boolean estActif = false;

    Frigo(float températureDépart) {
        températureActuelle = températureDépart;
        MiseAJour(0);
    }
}

```

```

public void MiseAJour(int nombreMinutesEcoulees) {
    if (EstActif()) {
        temperatureActuelle -= nombreMinutesEcoulees * 0.5f;
        if (temperatureActuelle < temperatureIdéale) {
            estActif = false;
            if (temperatureActuelle < 0.0f)
                temperatureActuelle = 0.0f;
        }
    }
    else {
        temperatureActuelle += nombreMinutesEcoulees * 0.01f;
        if (temperatureActuelle > temperatureIdéale + 0.5f)
            estActif = true;
    }
}

```

```

// à compléter...
}

```

```

class Glacière implements AppareilRéfrigérant {
    private float temperatureActuelle;
    private final float temperatureIdéale = 8.0f;

    Glacière(float temperatureDépart) {
        temperatureActuelle = temperatureDépart;
    }
}

```

```

// à compléter...
}

```

```

// à compléter...

```

- Compléter les deux classes **Frigo** et **Glacière** en implémentant les méthodes manquantes.
- Ajouter une méthode **MiseAJour()** dans la classe **Glacière** reprenant le même paramètre **nombreMinutesEcoulees** que la méthode similaire de la classe **Frigo**. Cette fonction calculant simplement la dissipation de la chaleur à raison de 0.02 degrés par minute.
- Ecrire un programme **MarchandDeGlace** créant un frigo et une glacière avec comme paramètres initiaux respectifs 19. et 3 degrés. Ce programme
  - Affichera l'état d'activation du Frigo juste après sa création.
  - Opèrera une boucle de mise à jour de 60 minutes dans laquelle les deux méthodes **MiseAJour()** seront appelées.
  - Affichera l'état d'activation du Frigo après la fin de la boucle.
  - Affichera les températures actuelles finales du frigo et de la glacière.