Université Tunis El-Manar	Faculté des Sciences de Tunis
Module : Programmation.O.O	**Année universitaire :** 2024-2025
Section : MI 2, PI 2	Hajer Dammak

Série N° 5 bis : Héritage

Points essentiels :

- Héritage et polymorphisme
- L'utilisation de `super`

Exercice 1 :

Soit le programme suivant. Indiquez ce que l'on obtient à l'exécution.

```
public class ObjetGraphique {
    int x, y;
    ObjetGraphique(int x1, int y1){
        x = x1;
        y = y1;
    }
    void affichage (){
    System.out.print ("Affichage d'un objet graphique"); }
}

class Rectangle extends ObjetGraphique{
    int x = 2, y = 2;
    int longueur, largeur;
    Rectangle (int L, int l) {
        super(1, 1);
        longueur = L;
        largeur = l;
    }
    @Override
    void affichage (){
        super.affichage ();
        System.out.println(" ("+super.x+" "+super.y+")");
        System.out.println("Cet objet graphique est un rectangle: "
        + "(" +x+" "+y+" "+longueur+" "+largeur+")");
    }
}
class EssaiRectangle{
    public static void main(String [] args) {
        Rectangle r = new Rectangle (10, 5);
        r. affichage ();
    }
}
```

Exercice 2 :

Soit la classe `Test` suivante :

```
class Test {
    public static void main(String[] args){
    C1 o1 = new C1();
    C1 o2 = new C11();
    C111 o3 = new C111();
    C11 o4 = new C111();
    C1 o5 = new C111();
    }
}
class C1 {}
class C11 extends C1 {}
class C111 extends C11{}
```

Déterminer si les instructions suivantes sont valides ou non. En cas d'erreur justifier rapidement pourquoi et préciser s'il y a erreur à la compilation ou bien à l'exécution. Les instructions suivantes sont indépendantes les unes des autres.

```
1) o1 = o2;
2) o1 = o3;
3) o3 = o1;
4) o4 = o5;
5) o3 = (C111)o1;
6) o4 =(C11)o5 ;
7) o4 = (C111)o2;
8) o3 = (C11)o5;
```

Exercice 3 :

Soit la classe `Vehicule` caractérisée par les attributs et méthodes suivants :

Attributs :

- `moteur` de type `boolean`, il contient `true` si le véhicule possède un moteur, sinon il contient `false`,
- `nbPneus` représentant le nombre de pneus du véhicule (2, 3, 4, ...).

Méthodes :

- Un constructeur avec paramètres.
- `getMoteur()` qui retourne la valeur de l'attribut moteur.
- `getNbPneus()` qui retourne la valeur de l'attribut `nbPneus`.
- `afficher()` qui affiche les attributs en utilisant les méthodes `getMoteur` et `getNbPneus`.

Soit la classe `Automobile` qui hérite de la classe `Vehicule` et qui est caractérisée par les attributs et méthodes suivants :

Attributs :

- `nbCylindres`, un entier qui indique le nombre de cylindres du moteur de l'automobile.
- `puissanceCylindre`, un entier qui est la valeur de la puissance nominale fournie par un cylindre.

Méthodes :

- Un constructeur avec paramètres.
- `getPuissance()` qui retourne la puissance selon la formule `nbCylindre` × `puissanceCylindre`.
- `setPuissance()` qui modifie la puissance du cylindre.
- `setNbCylindre()` qui modifie le nombre de cylindres.
- `afficher()` qui affiche les attributs de la classe.

1) Implémenter les classes `Vehicule` et `Automobile`; modifier la méthode afficher de la classe `Automobile` pour qu'elle accepte comme paramètre `nbPneus`. Qu'appelle-t-on ce mécanisme ? Implémenter cette nouvelle méthode.

2) Implémenter la classe `TestVehicule` qui permet de :
   - Créer un objet `v` de type `Vehicule`.
   - Instancier cet objet `v` par le type `Automobile`, qu'appelle-t-on ce mécanisme ?
   - Vérifier si `v` est un objet de type `Automobile`. (`v instanceof Automobile`)
   - Est-il possible d'appeler la méthode `getPuissance()` par l'objet `v` ? Quelle solution proposez-vous ? Qu'appelle-t-on ce mécanisme ?

Exercice 4 :

Soient les classes suivantes :

- `Employe` définie par les attributs `nom`, `prénom`, `adresse`, `CIN`, `matricule`, `date d'embauche` et `salaire`.
- `EmployeAssure`: un employé assuré se caractérise par les attributs `numéro CNSS` et `date d'inscription` à la CNSS.
- `EmployeNonAssure`: c'est un employé non assuré.

Les attributs `Date d'inscription` à la CNSS et `Date d'embauche` sont de type `Date`.

Chaque classe possède le/les constructeurs (avec paramètres et/ou de copie) et la méthode `afficher()`.

Soit la classe `Societe` qui embauche des employés assurés et des employés non assurés. Une société se caractérise par les attributs suivants : `code`, `raison sociale` et un tableau d'employés recrutés (qui peuvent être assurés et/ou non assurés).

On peut, pour une société afficher ses attributs, ajouter, rechercher et supprimer des employés assurés et non assurés.

1) Implémenter les classes appropriées.
2) Implémenter une classe `TestSociete`.