TP N01
(Création des classes drivées)

Enoncé : On dispose de la classe suivante :

```
public class Point {
	private int x, y ;
	public void initialise (int x, int y) { this.x = x ; this.y = y ; }
	public void deplace (int dx, int dy) { x += dx ; y += dy ; }
	public int getX() { return x ; }
	public int getY() { return y ; }}
```

1. créer un projet sous le nom TP01_Heritage
2. créer une classe PointA, dérivée de la classe Point disposant d'une méthode affiche affichant (en fenêtre console) les coordonnées d'un point.
2. Ecrire un petit programme de test utilisant les deux classes Point et PointA.
3. Que se passerait-il si la classe Point ne disposait pas des méthodes getX et getY ?

Réponse

2.Il suffit de définir une classe dérivée en utilisant le mot clé **extends**.

```
public class PointA extends Point{
void affiche(){ System.out.println ("Coordonnees : " + getX() + " " + getY()) ; }}
```

3. On peut alors créer des objets de type PointA et leur appliquer aussi bien les méthodes publiques de PointA que celles de Point comme dans ce programme accompagné d'un exemple d'exécution :

```
public class TsPointA {
	public static void main (String args[])
	{ Point p = new Point () ;
	p.initialise (2, 5) ;
	System.out.println ("Coordonnees : " + p.getX() + " " + p.getY() ) ;
	PointA pa = new PointA () ;
	pa.initialise (1, 8) ; // on utilise la methode initialise de Point
	pa.affiche() ; // et la methode affiche de PointA
	}}
```

Résultat affiché sur la console :
Coordonnees : 2 5
Coordonnees : 1 8

4. Notez bien qu'un appel tel que p.affiche() conduirait à une erreur de compilation puisque la classe de p (Point) ne possède pas de méthode affiche. Si la classe Point n'avait pas disposé des méthodes d'accès getX et getY, il n'aurait pas été possible d'accéder à ses champs privés x et y depuis la classe PointA. Il n'aurait donc pas été possible de la doter de la méthode affiche. L'héritage ne permet pas de contourner le principe d'encapsulation.

TP N02
(Encapsulation dans l'héritage)

Enoncé : On dispose de la classe suivante :
On dispose de la classe suivante :

```
public class Point {
	private int x, y ;
	public void setPoint (int x, int y) { this.x = x ; this.y = y ; }
	public void deplace (int dx, int dy) { x += dx ; y += dy ; }
	public void affCoord ()
	{ System.out.println ("Coordonnees : " + x + " " + y) ;}}
```

1.Créer un projet sous le nom TP02_Heritage
2. une classe PointNom, dérivée de Point permettant de manipuler des points définis par deux coordonnées (int) et un nom (caractère). On y prévoira les méthodes suivantes :
• setPointNom pour définir les coordonnées et le nom d'un objet de type PointNom,
• setNom pour définir seulement le nom d'un tel objet,
• affCoordNom pour afficher les coordonnées et le nom d'un objet de type PointNom.
3.Écrire un petit programme utilisant la classe PointNom.

Réponse

2.

```
// la classe dérivés PointNom
public class PointNom extends Point{
	private char nom ;
	 public void setPointNom (int x, int y, char nom)
	{ setPoint (x, y) ;	this.nom = nom ;	}
	public void setNom(char nom)	{ this.nom = nom ; }
	public void affCoordNom(){ System.out.print ("Point de nom " + nom + " ")
;affCoord() ; }}
```

3.

```
// la classe de test
public class TsPointN {
	public static void main (String args[])
	{ Point p = new Point () ;
	p.setPoint (2, 5) ;
	p.affCoord() ;
	PointNom pn1 = new PointNom() ;
	pn1.setPointNom (1, 7, 'A') ; // méthode de PointNom
	pn1.affCoordNom() ; // méthode de PointNom
	pn1.deplace (9, 3) ; // méthode de Point
	pn1.affCoordNom() ; // méthode de PointNom
	PointNom pn2 = new PointNom() ;
	pn2.setPoint (4, 3) ; // méthode de Point
	pn2.setNom ('B') ; // méthode de PointNom
	pn2.affCoordNom() ; // méthode de PointNom
	pn2.affCoord() ; // méthode de Point
	}}
```

TP N03
(Héritage et appel au constructeur)

Enoncé : On dispose de la classe suivante (disposant cette fois d'un constructeur) :

```
public class Point{
      private int x, y ;
      public Point (int x, int y) { this.x = x ; this.y = y ; } // constructeur
  public void affCoord() { System.out.println ("Coordonnees : " + x + " " + y);}}
```

1.créer un projet sous le nom TP03_Heritage
2.Réaliser une classe PointNom, dérivée de Point permettant de manipuler des points définis par leurs coordonnées (entières) et un nom (caractère). On y prévoira les méthodes suivantes :
a. constructeur pour définir les coordonnées et le nom d'un objet de type PointNom,
b. affCoordNom pour afficher les coordonnées et le nom d'un objet de type PointNom.
3.Écrire un petit programme utilisant la classe PointNom.

Réponse

2.La classe dérivée PointNom doit prendre en charge la construction de l'intégralité de l'objet correspondant, une redéfinition du constructeur de la classe de base est indispensable. Rappelons que l'appel du constructeur de la classe de base (fait à l'aide du mot clé **super**) doit constituer la première instruction du constructeur de la classe dérivée.

```
public class PointNom extends Point{
      private char nom ;
      public PointNom (int x, int y, char nom)
      { super (x, y) ; // appel au constructeur de la classe de base
      this.nom = nom ;
      }
      public void affCoordNom()
      { System.out.print ("Point de nom " + nom + " ") ;
      affCoord() ; }     }
```

3.un programme de test

```
public class TsPointC {
      public static void main (String args[])
      { PointNom pn1 = new PointNom(1, 7, 'A') ;
      pn1.affCoordNom() ; // methode de PointNom
      PointNom pn2 = new PointNom(4, 3, 'B') ;
      pn2.affCoordNom() ; // methode de PointNom
      pn2.affCoord() ; // methode de Point
      }
      }
```

TP N04
(Héritage et appel au constructeur)

Enoncé : nous allons traiter les cas de définition d'un constructeur dans une classe de base et comment faire l'appeler dans la classe dérivée. Vous allez créer 4 TP en suivant le parcours en-dessous.

a.cas1

1. créer un projet sous le nom TP04_heritage_cas1
2.créer les deux classes suivante : A et B

```
public class A {  }
public class B extends A  {
	public B() {super();}}
```

3.expliquer les erreurs
4.expliquer la différence entre le constructeur de classe A et B
5.quel effet sur la création des objets pour la classe B

b.cas2

1. créer un projet sous le nom TP04_heritage_cas2
2.créer les trois classes suivantes : A et B et tp04_heritage_cas2

```
public class A {
	public A() {} //constructeur1
	public A ( int n) {} //constructeur2   }

public class B extends A{ // …..pas de constructeur }

public class tp04_heritage_cas2 {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		B b = new B();}}
```

3.expliquer les erreurs
4.expliquer la différence entre les constructeur de classe A et B
5.quel effet sur la création des objets pour la classe B

c.cas3

1. créer un projet sous le nom TP04_heritage_cas3
2.créer les trois classes suivantes : A et B et tp04_heritage_cas3

```
public class A {
public A ( int n) {} //constructeur2 }
public class B extends A { // …..pas de constructeur      }
public class tp_heritage_cas3 {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		B b = new B();} }
```

3.expliquer les erreurs
4.quel effet sur la création des objets pour la classe B

c.cas4

1. créer un projet sous le nom TP04_heritage_cas4
2.créer les trois classes suivantes : A et B et tp04_heritage_cas4

```
public class A {
// pas de constructeur2 }
public class B extends A { // …..pas de constructeur      }
public class tp_heritage_cas4 {
      public static void main(String[] args) {
            // TODO Auto-generated method stub
            B b = new B();} }
```

3.expliquer les erreurs
4.expliquer la différence entre les constructeur de classe A et B
5.quel effet sur la création des objets pour la classe B

Réponse

Cas 1

3.pas d'erreurs
4.La classe A comporte un constructeur implicite par défaut
La classe B définit un constructeur (explicite), on fait appel au constructeur de la classe mère a travers de l'instruction super ()
5.pour créer un objet de la classe B en créant un objet de la classe A en faisant appel au constructeur implicite.

Cas 2

3.pas d'erreurs
4.la classe A définit deux constructeurs l'un est par défaut (explicite) et l'autre est paramétrée tandis que la classe B ne définit aucun constructeur (ce qu'ils en résulte la présence d'un constructeur (implicite) par défaut de la classe mère A)
5.pour créer un objet de la classe B en créant un objet de la classe A en faisant appel au premier constructeur explicite de la classe A.

Cas 3

3.il existe une erreurs au niveau de la classe B, car la classe mère A ne définit pas le constructeur par défaut implicite, donc elle doit définir un constructeur par défaut explicite.
4.on ne peut pas créer des objets de la classe B

Cas 4

3.pas d'erreurs
4.les deux constructeur sont définies par défaut implicitement, ce qui implique l'appel l'Appel de constructeur par défaut de B qui Appelle le constructeur par défaut de A.
5.créer un objet de la classe B dépend de la création d'un objet de la classe A

TP N05
(La classe Object)

Enoncé : Toute classe dérive implicitement de la classe **Object.** Dans ce TP nous allons créer une classe qui modélise un objet étudiant et faire appel aux méthodes de la classe Object (toStirng,equlas,getClass, getName). Dans notre cas on considère qu'un étudiant est cratérisé par un matricule et un nom.

1. créer un projet sous le nom TP05_heritage_classObjetct
2.créer la classe Etudiant en prévoira les méthodes suivantes :
a.Deux constructeurs l'un est avec initialisation des variables et l'autre est paramétré
3.Créer une classe de test sous le nom : tp05_heritage_classObjet
a.Montrer comment faire appel aux méthodes citées au-dessus de la classe mère Object

Réponse

2.classe Etudiant

```
public class Etudiant {
	private int matricule;
	private String nom;
	public Etudiant (){matricule=1;nom="salim";} // constructeur avec
initialisation des variables
	public Etudiant (int mat,String nom){this.matricule=mat;this.nom=nom;} //
constructeur paramétré
	//vous pouvez redéfinir la méthode toString de la classe de base Object
	// un exemple de redéfinition
	//public String toString (){return (matricule+"	"+nom);}
}
```

3.la classe de test

```
public class tp05_heritage_classObject {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Etudiant e1= new Etudiant();
		Etudiant e2= new Etudiant(2,"fateh");
		// appel a la méthode toString de la classe Object
		System.out.println(e1.toString());
		// appel a la metode getClass
		System.out.println(e1.getClass());
		// appel a la méthode getClass et getName
		System.out.println(e1.getClass().getName());
		// pour tester la méthode equlas
		e2=e1; // e2 et e1 pointent sur le meme objet
		if (e2.equals(e1)) {	System.out.println("deux objet
identiques");}
		else {System.out.println("deux objet differents");}
		}}
```

TP N06
(Transtypage (cast) implicite et explicite)

Enoncé : le TP doit contenir deux demonstration, l'une pour appliquer le transtypage entre les type primitifs (**byte, int short,long, float et double**) et l'autre entre des classe définies par nous même. Dans le deuxième cas, nous allons reprendre la classe Etudiant du TP05.

Rappel : les règles pqssqge de transtypage entre les types primitifs

byte---→ short---→ int---→long---→float---→ double

I-transtypage entre les primitifs

1. créer un projet sous le nom TP06_heritage_transtypage
2.Créer une classe de test sous le nom : tp06_heritage_classObjet (contient la méthode main)
3.insérer les instructions suivantes :

```
byte b1 = 15;
int i1 = 100;
long l1 = 2000;
float f1 = 2.5f; // f pour float; par défaut de type double
double d1 = 2.5;
```

3.1.insérer l'instruction suivante : `b1=b1+1 ;`
dégager votre remarque ? proposer une correction ?
3.2.insérer l'instruction suivante : `byte b2 = i1;`
dégager votre remarque ? proposer une correction ?
3.3.insérer l'instruction suivante : `long l2 = i1;`
dégager votre remarque ? justifier ?
3.4.insérer l'instruction suivante : `int i2 = l1;`
dégager votre remarque ? proposer une correction ?
3.5.insérer l'instruction suivante : `float f2 = l1;`
dégager votre remarque ? justifier ?
3.6.insérer l'instruction suivante : `long l3 = f1;`
dégager votre remarque ? proposer une correction ?
3.7.insérer l'instruction suivante : `double d2 = f1;`
dégager votre remarque ? justifier ?
3.8.insérer l'instruction suivante : `float f3 = d1;`
dégager votre remarque ? proposer une correction ?
3.9.insérer les instructions suivantes :

```
int i3 = 15;
long l4 = i3 * (long) 2.15f;
long l5 = (long) (i3 * 2.5f);
System.out.println ("\nl4 : " + l4 + "\nl5 : " + l5);
```

qu'affiche l'instruction `System.out.println` ? justifier ?
3.10.insérer les instructions suivantes :

```
byte b3 = 10;
short s4 = 2*b3;
```

dégager votre remarque ? proposer une correction ?

3.11.insérer les instructions suivantes :

```
int i4 = 10;
long l6 = 1000;
int i5 = i4 + l6;
```

dégager votre remarque ? proposer une correction ?

II-transtypage pour les objets (classe Etudiant)

3.12.reprendre la classe Etudiant du TP05 est la copier dans le package du TP05, puis insérer les instructions suivantes :

```
Object o2=new Object();
Etudiant e2=new Etudiant();
e2=o2;

Object o1=new Object();
Etudiant e1=new Etudiant();
o1=e1;
```

dégager votre remarque ? proposer une correction ?

Réponse

3.1. Une erreur de compilation, on ne peut pas affecter un entier à un byte, un casting est nécessaire. Ajouter l'instruction suivante : `b1 = (byte) (b1 + 1);`

3.2. Une erreur de compilation, on ne peut pas affecter un entier à un byte, un casting est nécessaire. Ajouter l'instruction suivante : `byte b2 = (byte) i1;`

3.3. Pas d'erreur, on peut le faire sans cast puisque de `int` → `long`

3.4. Une erreur de compilation, on ne peut pas affecter un long à un int, un casting est nécessaire. Ajouter l'instruction suivante : `int i2 = (int) l1;`

3.5. Pas d'erreur, on peut le faire sans cast puisque de `long` → `float`

3.6. Une erreur de compilation, on ne peut pas affecter un float à un long, un casting est nécessaire. Ajouter l'instruction suivante : `long l3 = (long) f1;`

3.7. Pas d'erreur, on peut le faire sans cast puisque de `float` → `double`

3.8. Une erreur de compilation, on ne peut pas affecter un double à un float, un casting est nécessaire. Ajouter l'instruction suivante : `float f3 = (float) d1;`

3.9.elle affiche :

```
l4 : 30
l5 : 37
```

Pour l'instruction `long l4 = i3 * (long) 2.15f;` il y a un transtypage de `2.15f,` il en résulte `2` puis `2*15` vaut `30` donc `l4` vaut `30.`
Pour l'instruction `long l5 = (long) (i3 * 2.5f);` il y a un calcul de `2.5*15 ;` il en résulte `37.5` puis un transtypage de cet dernier résultat qui donne `37`

3.10. Une erreur de compilation, on ne peut pas affecter un int à un short, un casting est nécessaire. Ajouter l'instruction suivante : `short s4 = (short) (2*b3);`

3.11. Une erreur de compilation, on ne peut pas affecter un long à un int, un casting est nécessaire. Ajouter l'instruction suivante : `int i5 = (int) (i4 + l6);`

II-transtypage pour les objets (classe Etudiant)

3.12. Une erreur de compilation au niveau de l'instruction `e2=o2;`, on ne peut pas affecter un une référence d'un objet de type Object (classe mère) à un objet de type Etudiant (sous classe de l'Object), un casting est nécessaire. Ajouter l'instruction suivante : `e2=(Etudiant) o2;`

pour l'instruction `o1=e1;` il n y a pas d'erreur car la relation héritage nous permet de le faire.

TP N07
(Polymorphisme, surcharge, redéfinition des méthodes et Classe abstraite)

Enoncé : Dans un établissement d'enseignement, on trouve trois sortes de personnes : des secrétaires, des enseignants et des étudiants. Chaque personne est caractérisée par son nom et prénom, son adresse (rue et ville). Les variables d'instances (attributs) sont le nom, le prénom, la rue et la ville. nbPersonnes est une variable de classe qui comptabilise le nombre de Personne dans l'établissement. Tout les les attributs sont déclarés ***protected.***

➔ Créer un projet sous le nom TP07_pol_sur_red_clas_abstraite

Définir les méthodes public suivantes de la classe **abstraite Personne** :

1. le constructeur **Personne** (String nom, String prenom, String rue, String ville) : crée et initialise un objet de type Personne.
2. String **toString** () : fournit une chaîne de caractères correspondant aux caractéristiques (attributs) d'une personne.
3. **ecrirePersonne** () : affiche les caractéristiques d'une personne. Elle ne fait rien. Elle est déclarée ***abstraite***.
4. static **nbPersonnes** () : affiche le nombre total de personnes et le nombre de personnes par catégorie. C'est une méthode de classe.
5. **modifierPersonne** (String rue, String ville) : modifie l'adresse d'une personne et appelle **ecrirePersonne** () pour vérifier que la modification a bien été faite.

Une **Secrétaire** est une Personne ayant une caractéristique supplémentaire : un numéro de bureau est de type entier.
La variable static nbSecretaires est incrémentée dans le constructeur de l'objet Secretaire.

Définir les méthodes public suivantes de la classe **Secrétaire**:

1. le constructeur **Secrétaire** : crée et initialise un objet de type Secrétaire.
2. redéfinir la méthode String **toString** () : fournit une chaîne de caractères correspondant aux caractéristiques (attributs) d'une Secrétaire.
3. redéfinir la méthode **ecrirePersonne** () : affiche les caractéristiques d'une Secrétaire.
4. static **nbSecrétaire** () : renvoie le nombre total de Secrétaire.

5. Surcharger la méthode **modifierPersonne** (String nom,String prenom,String rue, String ville, numb) de la classe de base Personne, pour quelle puisse modifier le nom,prenom et numéro de bureau

Un **Etudiant** est une Personne préparant un diplôme. L'attribut diplôme est de type chaine de caractères. La variable static nbEtudiants compte le nombre d'étudiants ; elle est incrémentée dans le constructeur Etudiant.
Définir les méthodes public suivantes de la classe **Etudiant**:

1. le constructeur **Etudiant**: crée et initialise un objet de type Etudiant.
2. redéfinir la méthode String **toString** () : fournit une chaîne de caractères correspondant aux caractéristiques (attributs) d'un Etudiant.
3. redéfinir la méthode **ecrirePersonne** () : affiche les caractéristiques d'un Etudiant.
4. static nbEtudiant () : renvoie le nombre total des Etudiants.
5. Surcharger la méthode **modifierPersonne** (String nom,String prenom,String rue, String ville) de la classe de base Personne, pour quelle puisse modifier le nom et le prenom.

Un **Enseignant** est une Personne enseignant une spécialité. L'attribut spécialité est de type chaine de caractères.
La variable static nbEnseignants compte le nombre d'enseignants ; elle est incrémentée dans le constructeur
Enseignant.
Définir les méthodes public suivantes de la classe **Enseignant**:

1. le constructeur **Enseignant**: crée et initialise un objet de type Enseignant.
2. redéfinir la méthode String **toString** () : fournit une chaîne de caractères correspondant aux caractéristiques (attributs) d'un Enseignant.
3. redéfinir la méthode **ecrirePersonne** () : affiche les caractéristiques d'un Enseignant.
4. static nbEnseignant () : renvoie le nombre total des Enseignants.

Ecrivez une classe de test **Etablissement** qui comporte la méthode main en respectant l'ordre des opérations suivantes :
1-Déclarer un tableau de 5 personnes, puis créer 5 objets (02 Secrétaire, 02 Etudiant,01 Enseignant).
2-Remplir le tableau par les 5 objets, puis afficher les caractéristiques des personnes ayant d'adresse ville= Alger.
3-Tester les méthodes static **nbPersonnes** () et **modifierPersonne.**

Réponse :

```
abstract class Personne { // classe abstraite, non instanciable
            protected String nom;
            protected String prenom;
            protected String rue;
            protected String ville;
            protected static int nbPersonnes = 0; // nombre de Personne
            // constructeur de Personne
            Personne (String nom, String prenom, String rue, String ville) {
            this.nom = nom; // voir this page 38
            this.prenom = prenom;
            this.rue = rue;
```

```
        this.ville = ville;
        nbPersonnes++;
        }
        // fournir les caractéristiques d'une Personne sous forme
        // d'un objet de la classe String
        public String toString() {
        return nom + " " + prenom + " " + rue + " " + ville;
        }
        // méthode abstraite (à redéfinir dans les classes dérivées)
        abstract void ecrirePersonne ();
        static void nbPersonnes () {
        System.out.println (
        "\nNombre d'employés : " + nbPersonnes +

        "\nNombre de secrétaires : " + Secretaire.nbSecretaires() +

        "\nNombre d'étudiants : " + Etudiant.nbEtudiants()
        );
        }
        void modifierPersonne (String rue, String ville) {
        this.rue = rue;
        this.ville = ville;
        ecrirePersonne(); // de la classe dérivée de l'objet
        }
        } // Personne
```

```
class Etudiant extends Personne { // héritage de classe
private String diplomeEnCours;
private static int nbEtudiants = 0; // nombre d'Etudiant
Etudiant (String nom, String prenom, String rue, String ville,
String diplomeEnCours) {
super (nom, prenom, rue, ville); // appel du constructeur Personne
this.diplomeEnCours = diplomeEnCours;
nbEtudiants++;
}
public String toString () {
return super.toString() + "\n Diplome en cours : "
+ diplomeEnCours;
}
void ecrirePersonne () {
System.out.println ("Etudiant : " + toString());
}
public String diplomeEnCours () {
return diplomeEnCours;
}
static int nbEtudiants () { return nbEtudiants; }

void modifierPersonne (String nom,String prenom,String rue, String ville) {
      this.nom=nom;
      this.prenom=prenom;
      this.rue = rue;
      this.ville = ville;
      ecrirePersonne(); // de la classe dérivée de l'objet
}
}
```

```
public class Secretaire extends Personne {



	// héritage de classe
	private String numeroBureau;
	private static int nbSecretaires = 0; // nombre de Secretaire
	Secretaire (String nom, String prenom, String rue, String ville,
			String numeroBureau) {
			// le constructeur de la super-classe
			super (nom, prenom, rue, ville); // appel du constructeur
Personne
			this.numeroBureau = numeroBureau;
			nbSecretaires++;
			}
	public String toString () {
	// super.toString() : toString() de la super-classe
	return super.toString() + "\n N bureau : " + numeroBureau;
	}
	void ecrirePersonne () {
	System.out.println ("Secrétaire : " + toString());
	}
	static int nbSecretaires () { return nbSecretaires; }

	void modifierPersonne (String nom,String prenom,String rue, String ville,
String numb) {
		this.nom=nom;
		this.prenom=prenom;
		this.rue = rue;
		this.ville = ville;
		this.numeroBureau=numb;
		ecrirePersonne(); // de la classe dérivée de l'objet
	}
	} // Secretaire

public class Etablissement {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		Personne []t=new Personne[4];
		Secretaire s1=new Secretaire ("Salim","Saliha","emir
aek","alger","01");
		Secretaire s2=new Secretaire
("fatih","zoubida","amirouche","anbaba","02");
		Etudiant e1=new Etudiant ("sadik","hakim","zighoud","oran","liscence
informatique");
		Etudiant e2=new Etudiant ("abass","farih","arbi","alger","liscence
genie civil");
		t[0]=s1;t[1]=s2;t[2]=e1;t[3]=e2;
	 for (int i=0;i<t.length;i++)
		{if (t[i].ville =="alger") t[i].ecrirePersonne();}

		for (int i=0;i<t.length;i++)
		{t[i].ecrirePersonne();}

		Personne.nbPersonnes();
```

```
            s1.modifierPersonne("salim1", "saliha1", "emir slah","zabana","06");
            for (int i=0;i<t.length;i++)
            {t[i].ecrirePersonne();}
      }}
```

Remarque : vous pouvez continuez à implémenter la classe Enseignant …..

TP N08
(Interface)

Enoncé : Certain animaux peuvent crier, d'autres sont muets. On représentera le fait de crier au moyen d'une méthode affichant à l'écran le cri de l'animal.

1. créer un projet sous le nom TP08_interface
2.écrire une interface contenant la méthode permettant de crier.
3.écrire les classes des chats, des chiens et des lapins (qui sont muets)
4.écrire un programme avec un tableau pour les animaux qui savent crier, le remplir avec des chiens et des chats, puis faire crier tous ces animaux.
5.Décrire ce qui s'affiche à l'écran à l'exécution de ce programme.

Réponse :

```
2.Public interface Criant {     void crier();}
3.        public class Chat implements Criant {
                        public void crier(){System.out.println("Miou");} }
          public class Chien implements Criant {
                    public void crier(){System.out.println("ou");}}
          public class Lapin  {
               public void froncer_du_nez(){System.out.println("pas de son");}}
4. public class Animaux {

             public static void main(String[] a){
             Criant[] tab = new Criant[4];
             tab[0] = new Chat();
             tab[1] = new Chien();
             tab[2] = new Chat();
             tab[3] = new Chien();
             for (int i=0; i<4; i++){
             tab[i].crier();    }           }           }
```

5.ce qu'il affiche à l'écran

```
Miou
ou
Miou
Ou
```

Il y a deux types d'objets différents stockés dans le tableau tab (chat et chien) mais normalement un tableau stocke des éléments de même type. Ceci est traduit par la puissance du concept de l'héritage dans le paradigme orientée objet. Puis chaque objet excuse son propre méthode tout en créant le son correspondant ce qui traduit la notion de polymorphisme.