

## 1) Héritage - Visibilité

la table de visibilité selon les droits d'accès (niveaux de visibilité):

Visibilité				
Modificateur	Classe	Paquetage	Sous-classe	Partout
private	Oui	Non	Non	Non
Rien	Oui	Oui	Non	Non
<i>protected</i>	Oui	Oui	Oui	Non
Public	Oui	Oui	Oui	Oui

Indiquez par Oui ou Non après les signes de commentaires (//) si les attributs de la classe `Personne` sont accessibles ou visibles dans les classes `FaculteSciences`, `Employe`, `DepartementInformatique`, `Enseignant`, et `Universite`.

<pre>package p1;  public class Personne {     protected String nom ;     public String prenom ;     private int age ;     double taille; }</pre>	<pre>package p1;  public class FaculteSciences {      public void estAccessible(){          Personne p = new Personne();         p.nom      = "Daniel";    //____         p.prenom   = "Diderot";   //____         p.age      = 99;          //____         p.taille   = 1.83;        //____      }  }</pre>
<pre>package p2;  import p1.Personne;  public class Employe extends Personne {      public void estAccessible(){          nom      = "Daniel";    //____         prenom    = "Diderot";   //____         age       = 99;          //____         taille    = 1.83;        //____      }  }</pre>	<pre>package p2;  import p1.Personne;  public class DepartementInformatique {      public void estAccessible(){          Personne p = new Personne();         p.nom      = "Daniel";    //____         p.prenom   = "Diderot";   //____         p.age      = 99;          //____         p.taille   = 1.83;        //____      }  }</pre>
<pre>package p3;  import p2.Employe;  public class Enseignant extends Employe {      public void estAccessible(){          nom      = "Daniel";    //____         prenom    = "Diderot";   //____         age       = 99;          //____         taille    = 1.83;        //____      }  }</pre>	<pre>package p3;  import p2.Employe;  public class Universite {      public void estAccessible(){          Personne p = new Personne();         p.nom      = "Daniel";    //____         p.prenom   = "Diderot";   //____         p.age      = 99;          //____         p.taille   = 1.83;        //____      }  }</pre>

## 2) Héritage - Constructeurs des superclasses & sous-classes

Selon la définition des constructeurs de la superclasse `Personne`, indiquez par Oui ou Non si la classe `Employe` compile sans aucune erreur de compilation.

a)

<pre>package constrcuteur;  public class Personne {     private String prenom;     private String nom;      public Personne(){     }      public Personne(String prenom,                      String nom){         this.prenom = prenom;         this.nom = nom;     } }</pre>	<pre>package constructeur;  public class Employe extends Personne {     private String matricule;     private float salaire;      public Employe(){         super();     }      public Employe(String prenom, String nom,                   String matricule, float salaire ){         super(prenom, nom);         this.matricule = matricule;         this.salaire = salaire;     } }</pre>
OUI <input type="checkbox"/> NON <input type="checkbox"/>	

b)

<pre>package constrcuteur;  public class Personne {     private String prenom;     private String nom;      public Personne(String prenom,                      String nom){         this.prenom = prenom;         this.nom = nom;     } }</pre>	<pre>package constructeur;  public class Employe extends Personne {     private String matricule;     private float salaire;      public Employe(){     }      public Employe(String prenom, String nom,                   String matricule, float salaire ){         super(prenom, nom);         this.matricule = matricule;         this.salaire = salaire;     } }</pre>
OUI <input type="checkbox"/> NON <input type="checkbox"/>	

c)

<pre>package constrcuteur;  public class Personne {     private String prenom;     private String nom; }</pre>	<pre>package constructeur;  public class Employe extends Personne {     private String matricule;     private float salaire;      public Employe(){      }      public Employe(String prenom, String nom,         String matricule, float salaire ){         this.matricule = matricule;         this.salaire = salaire;     } }</pre>
<p>OUI <input type="checkbox"/>                      NON <input type="checkbox"/></p>	

d)

<pre>package constrcuteur;  public class Personne {     private String prenom;     private String nom;      public Personne(String prenom,         String nom){         this.prenom = prenom;         this.nom = nom;     } }</pre>	<pre>package constructeur;  public class Employe extends Personne {     private String matricule;     private float salaire;      public Employe(String prenom, String nom,         String matricule, float salaire ){         this.matricule = matricule;         this.salaire = salaire;     } }</pre>
<p>OUI <input type="checkbox"/>                      NON <input type="checkbox"/></p>	

e)

<pre>package constrcuteur;  public class Personne {     private String prenom;     private String nom; }</pre>	<pre>package constructeur;  public class Employe extends Personne {     private String matricule;     private float salaire;      public Employe(){         super();     }      public Employe(String prenom, String nom,         String matricule, float salaire ){         super();         this.matricule = matricule;         this.salaire = salaire;     } }</pre>
<p>OUI <input type="checkbox"/>                      NON <input type="checkbox"/></p>	

### 3) Héritage - Redéfinition (masquage) / Surcharge (surdéfinition)

<pre>package redefinition;  public class Personne {      public boolean equals(Object obj) {         System.out.println("Personne-equals(Object)");         return true;     }      public String toString() {         System.out.println("Personne-toString()");         return "";     } }</pre>	<pre>package redefinition;  public class Employe extends Personne {      public boolean equals(Object obj) {         System.out.println("Employe-equals(Object)");         return true;     }      public String toString() {         System.out.println("Employe-toString()");         return "";     } }</pre>
<pre>package redefinition;  public class Enseignant extends Employe {      public boolean equals(Object obj) {         System.out.println("Enseignant-equals(Object)");         return true;     }      public String toString() {         System.out.println("Enseignant - toString()");         return "";     } }</pre>	<pre>package redefinition;  public class TestRedefinition {      public static void main(String[] args) {         Object o = null;          Personne pers = new Personne();         pers.equals(o);         pers.toString();          Employe emp = new Employe();         emp.equals(o);         emp.toString();          Enseignant ens = new Enseignant();         ens.equals(o);         ens.toString();          pers = emp;         pers.equals(o);         pers.toString();          pers = ens;         pers.equals(o);         pers.toString();     } }</pre>

a) Les méthodes `equals` et `toString` sont redéfinies dans les classes `Personne`, `Employe`, et `Enseignant`. Donnez ce qui sera affiché après l'exécution des instructions de la méthode `main` de la classe `TestRedefinition`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

<pre>package surcharge;  public class Personne {      public void valider(Personne p) {         System.out.println("Personne-valider(Personne)");     }  }</pre>	<pre>package surcharge;  public class Employe extends Personne {      public void valider(Employe emp) {         System.out.println("Employe-valider(Employe)");     }  }</pre>
<pre>package surcharge;  public class Enseignant extends Employe {      public void valider(Enseignant ens) {         System.out.println("Enseignant-valider(Enseignant)");     }  }</pre>	<pre>package surcharge;  public class TestSurcharge {      public static void main(String[] args) {          Personne pers = new Personne();         Employe emp = new Employe();         Enseignant ens = new Enseignant();          ens.valider(ens);         ens.valider(emp);         ens.valider(pers);      }  }</pre>
<p>b) La méthode <code>valider</code> est surchargée dans les classes <code>Employe</code>, et <code>Enseignant</code>. Donnez ce qui sera affiché après l'exécution des instructions de la méthode <code>main</code> de la classe <code>TestSurcharge</code>.</p> <p>.....</p> <p>.....</p> <p>.....</p>	

<pre>package surcharge;  public class Personne {      public void valider(Personne pers) {         System.out.println("Personne-valider(Personne)");     }      public void comparer(Personne pers) {         System.out.println("Personne-comparer(Personne)");     }      public void demenager(Personne pers) {         System.out.println("Personne-demenager(Personne)");     } }</pre>	<pre>package surcharge;  public class Employe extends Personne {      public void valider(Object o) {         System.out.println("Employe-valider(Object)");     }      public void comparer(Employe emp) {         System.out.println("Employe-comparer(Employe)");     }      public void demenager(Object o) {         System.out.println("Employe - demenager(Object)");     } }</pre>
<pre>package surcharge;  public class Enseignant extends Employe {      public void valider(Employe emp) {         System.out.println("Enseignant - valider(Employe)");     }      public void comparer(Object o) {         System.out.println("Enseignant - comparer(Object)");     }      public void demenager(Enseignant ens) {         System.out.println("Enseignant-demenager(Enseignant)");     } }</pre>	<pre>package surcharge;  public class TestSurcharge {      public static void main(String[] args) {          Employe emp = new Employe();         Enseignant ens = new Enseignant();          ens.valider(ens);         ens.comparer(ens);         ens.demenager(emp);      } }</pre>
<p>c) Les méthodes valider, comparer, et demenager sont surchargées dans les classes Employe, et Enseignant. Donnez ce qui sera affiché après l'exécution des instructions de la méthode main de la classe TestSurcharge.</p> <p>.....</p> <p>.....</p> <p>.....</p>	

<pre>package surcharge;  public class Personne {      public void valider() {         System.out.println("Personne-valider()");     }      public void comparer(Object o) {         System.out.println("Personne-comparer(Object)");     }      public void demenager(Personne pers) {         System.out.println("Personne-demenager(Personne)");     } }</pre>	<pre>package surcharge;  public class Employe extends Personne {      public void valider(Object o) {         System.out.println("Employe-valider(Object)");     }      public void comparer(Personne pers) {         System.out.println("Employe-comparer(Personne)");     }      public void demenager() {         System.out.println("Employe - demenager()");     } }</pre>
<pre>package surcharge;  public class Enseignant extends Employe {      public void valider(Personne pers) {         System.out.println("Enseignant - valider(Personne)");     }      public void comparer(Object o) {         System.out.println("Enseignant - comparer(Object)");     }      public void demenager(Object o) {         System.out.println("Enseignant-demenager(Object)");     } }</pre>	<pre>package surcharge;  public class TestSurcharge {      public static void main(String[] args) {          Enseignant ens = new Enseignant();         ens.valider(ens);         ens.comparer(ens);         ens.demenager(ens);     } }</pre>
<p>d) Les méthodes <code>valider</code>, <code>comparer</code>, et <code>demenager</code> sont surchargées dans les classes <code>Employe</code>, et <code>Enseignant</code>. Donnez ce qui sera affiché après l'exécution des instructions de la méthode <code>main</code> de la classe <code>TestSurcharge</code>.</p> <p>.....</p> <p>.....</p> <p>.....</p>	







## 6) Récursivité - Méthodes et traces d'exécution

a) Ci-dessous la méthode itérative `afficher` qui prend en paramètre une valeur entière `n` supérieure ou égale à 0 et affiche les valeurs de 1 à `n`.

```
public static void afficher (int n) {  
    for (int i = 1; i <= n; i++) {  
        System.out.println(i);  
    }  
}
```

Écrire la **version récursive** de la méthode `afficher`.

```
public static void afficherRecursive(int n) {
```

b) Écrire une méthode récursive `public static String inverserChaine (String chaine)` qui reçoit en paramètre une chaîne de caractères et retourne une nouvelle chaîne qui est l'inverse de la chaîne reçue en paramètre. Le paramètre `chaine` ne peut pas être nul ou vide.

```
public static String inverserChaine(String chaine) {
```

- c) Écrire la méthode `public static boolean estUnPalindrome (String chaine)` qui retourne vrai si la chaîne reçue en paramètre est un palindrome, sinon faux. Une chaîne est un palindrome si elle est égale à son inverse Ex: laval = laval (inversé). Le paramètre `chaine` ne peut pas être nul ou vide.

```
public static boolean estUnPalindrome(String chaine) {
```

}