

Formation Orienté Objet et Java de base

Atelier 5 : Redéfinition et utilisation de Final

Encadré par M.BOULCHAHOU

Objectif de l'atelier.....	1
Avant de commencer !	1
Exécution des méthodes par héritage	3
Redéfinition des méthodes	5
<i>Réduction de la visibilité des méthodes.....</i>	6
<i>Modification des paramètres des méthodes</i>	7
<i>Modification du type de retour par un type différent</i>	9
<i>Modification du type de retour par un type fils.....</i>	10
Le mot réservé « final ».....	11
« final » avant une méthode	11
« final » avant une classe	13
« final » avant un attribut.....	14
Conclusion.....	16

Objectif de l'atelier

L'objectif de cet atelier est de comprendre le mot réservé « final »

.

Vous allez apprendre à:

- *Les règles à appliquer lors de la redéfinition d'une méthode.*
- *Le rôle de final lorsqu'il est appliqué à une classe*
- *Le rôle de final lorsqu'il est appliqué à une méthode*
- *Le rôle de final lorsqu'il est appliqué à un attribut*

Avant de commencer !

Supposons que dans votre projet, vous faites la gestion des étudiants.

Créer un projet Java avec le nom TPFinal

Créer un package ma.gov.fst.tpfinal

Créer deux classes Etudiant, Professeur et Personne.

La classe Personne.java devrait être la classe mère des classes Etudiant et Professeur.

Formation Orienté Objet et Java de base

Atelier 5 : Redéfinition et utilisation de Final

Encadré par M.BOULCHAHOU

Créer une classe Test.java contenant la méthode main.

Classe	Contenu
Etudiant.java	Attribut age de type Integer Le getter et le setter de l'attribut age
Professeur.java	Attribut salaire de type Integer Le getter et le setter de l'attribut salaire
Professeur.java	Attribut nom de type String Le getter et le setter de l'attribut nom
Test.java	La méthode main

Dans la classe Personne nous souhaitons créer une méthode qui retourne la fonction de l'objet passé en paramètre.

```
public String fonctionPersonne(Personne p){  
    if(p instanceof Etudiant){  
        return "La fonction est etudiant...";  
    }  
  
    if(p instanceof Professeur){  
        return "La fonction est professeur...";  
    }  
  
    return "La fonction est inconnue";  
}
```

Le mot réservé *instanceof* vérifier le type d'un objet passé en paramètre.

(p *instanceof* Etudiant) vérifier si l'objet p a le type Etudiant

Formation Orienté Objet et Java de base

Atelier 5 : Redéfinition et utilisation de Final

Encadré par M.BOULCHAOUB

Exécution des méthodes par héritage

Dans la méthode main de la classe Test.java, créer un objet de type Personne (personne), un objet de type Etudiant (etudiant) et un objet de type Professeur (professeur).

Puis à travers l'objet personne, appeler la méthode fonctionPersonne() avec les trois types d'objets comme arguments.

Afficher ensuite les résultats retournées.

```
package ma.gov.fst.tpfinal;

public class Test {

    public static void main(String[] args) {

        Etudiant etudiant = new Etudiant();
        Professeur professeur = new Professeur();
        Personne personne = new Personne();

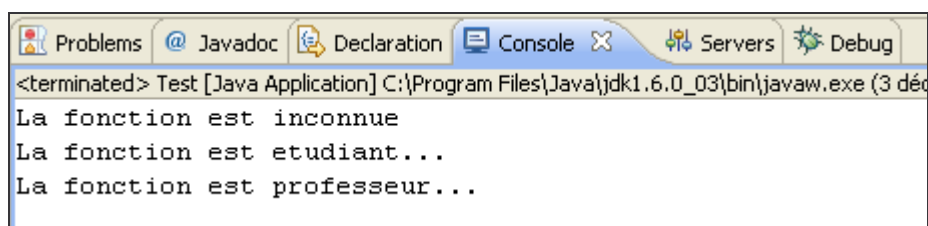
        String fctPeronne= personne.fonctionPersonne(personne);
        String fctEtudiant= personne.fonctionPersonne(etudiant);
        String fctProf= personne.fonctionPersonne(professeur);

        System.out.println(fctPeronne);
        System.out.println(fctEtudiant);
        System.out.println(fctProf);

    }

}
```

Faites l'exécution de cette classe. Vous devriez avoir le résultat suivant :



```
<terminated> Test [Java Application] C:\Program Files\Java\jdk1.6.0_03\bin\javaw.exe (3 déc
La fonction est inconnue
La fonction est etudiant...
La fonction est professeur...
```

Formation Orienté Objet et Java de base

Atelier 5 : Redéfinition et utilisation de Final

Encadré par M.BOULCHAOUB

Maintenant appeler la méthode fonctionPersonne() à travers l'objet etudiant.

```
package ma.gov.fst.tpfinal;

public class Test {

    public static void main(String[] args) {

        Etudiant etudiant =new Etudiant();
        Professeur professeur = new Professeur();
        Personne personne = new Personne();

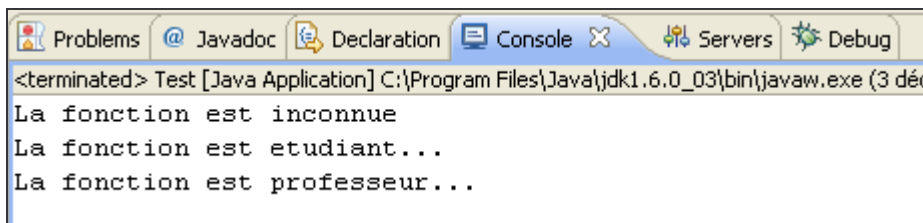
        String fctPeronne= etudiant.fonctionPersonne(personne);
        String fctEtudiant= etudiant.fonctionPersonne(etudiant);
        String fctProf= etudiant.fonctionPersonne(professeur);

        System.out.println(fctPeronne);
        System.out.println(fctEtudiant);
        System.out.println(fctProf);

    }

}
```

Faites l'exécution de cette classe. Vous devriez avoir le résultat suivant :



Maintenant appeler la méthode fonctionPersonne() à travers l'objet professeur.

```
package ma.gov.fst.tpfinal;

public class Test {

    public static void main(String[] args) {

        Etudiant etudiant =new Etudiant();
        Professeur professeur = new Professeur();
        Personne personne = new Personne();
```

Formation Orienté Objet et Java de base

Atelier 5 : Redéfinition et utilisation de Final

Encadré par M.BOULCHAOUB

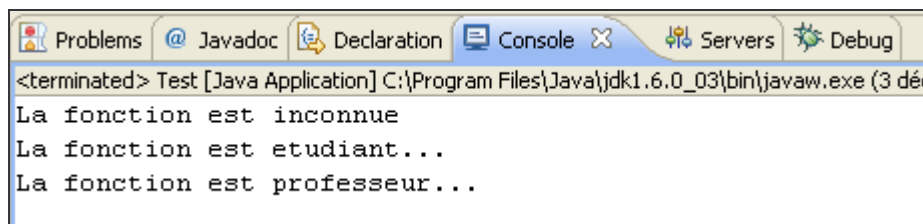
```
String fctPeronne= professeur.fonctionPersonne(personne);
String fctEtudiant= professeur.fonctionPersonne(etudiant);
String fctProf= professeur.fonctionPersonne(professeur);

System.out.println(fctPeronne);
System.out.println(fctEtudiant);
System.out.println(fctProf);

}

}
```

Faites l'exécution de cette classe. Vous devriez avoir le résultat suivant :



The screenshot shows the 'Console' tab of a Java IDE. The output text is as follows:

```
<terminated> Test [Java Application] C:\Program Files\Java\jdk1.6.0_03\bin\javaw.exe (3 déc
La fonction est inconnue
La fonction est etudiant...
La fonction est professeur...
```

Même si on change l'objet appelant, l'exécution ne change pas.

L'objet personne fait l'exécution de la fonction définie au de sa classe Personne.java, c'est une exécution normale d'une méthode.

Les deux objets etudiant et professeur font l'exécution de la méthode définie au niveau de leur classe mère Personne : C'est une exécution par héritage.

Redéfinition des méthodes

Nous souhaitons redéfinir la méthode fonctionPersonne() au niveau de la classe Etudiant.java, comme suivant

```
package ma.gov.fst.tpfinal;

public class Etudiant extends Personne {

    private String age;

    public String getAge() {
        return age;
    }
}
```

Formation Orienté Objet et Java de base

Atelier 5 : Redéfinition et utilisation de Final

Encadré par M.BOULCHAHOU

```
public void setAge(String age) {  
    this.age = age;  
}  
  
public String fonctionPersonne(Personne p) {  
  
    return "Il s'agit de la redéfinition de cette méthode";  
}  
  
}
```

Lors de la redéfinition d'une méthode, on doit respecter les règles de la redéfinition :

Réduction de la visibilité des méthodes

Essayer de reduire la visibilité de la méthode fonctionPersonne() au niveau de la classe Etudiant.java.

*L'ordre de la visibilité est le suivant :
Public> Protected> default>private*

Dans la classe Etudiant.java et dans la méthode fonctionPersonne(), mettre protected à la place de public :

```
package ma.gov.fst.tpfinal;  
  
public class Etudiant extends Personne {  
  
    private String age;  
  
    public String getAge() {  
        return age;  
    }  
  
    public void setAge(String age) {  
        this.age = age;  
    }  
  
    protected String fonctionPersonne(Personne p) {  
  
        return "Il s'agit de la redéfinition de cette méthode";  
    }  
  
}
```

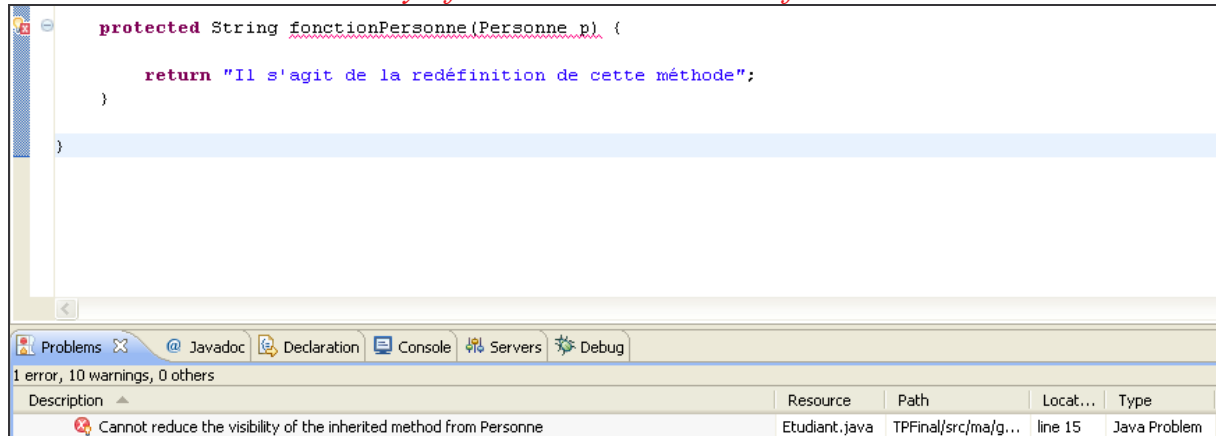
Formation Orienté Objet et Java de base

Atelier 5 : Redéfinition et utilisation de Final

Encadré par M.BOULCHAHOU

La réduction de la visibilité de public à protected donne une erreur de compilation lors de la redéfinition. Vous aurez l'erreur suivante:

Cannot reduce the visibility of the inherited method from Personne



Règle 1 :

Lors de la redéfinition des méthodes, la visibilité ne pourra jamais être réduite. Au contraire elle pourra être augmentée.

Modification des paramètres des méthodes

Essayons de modifier les paramètres de la méthode fonctionPersonne() par un paramètre String à titre d'exemple. Vous au niveau de la classe Etudiant.

```
package ma.gov.fst.tpfinal;  
  
public class Etudiant extends Personne {  
    private String age;  
  
    public String getAge() {  
        return age;  
    }  
  
    public void setAge(String age) {  
        this.age = age;  
    }  
  
    public String fonctionPersonne(String s) {  
        return "Il s'agit de la redéfinition de cette méthode";  
    }  
}
```

Formation Orienté Objet et Java de base

Atelier 5 : Redéfinition et utilisation de Final

Encadré par M.BOULCHAOUB

```
}
```

Avec cette implémentation, nous parlons de la surcharge de la méthode `fonctionPersonne()`. **Il ne s'agit pas d'une redéfinition**

La classe `Etudiant` contiennent deux méthodes ayant le même nom mais avec le type de paramètres différent.

Essayer d'appeler la méthode `fonctionPersonne()` avec `Personne` puis avec `String`.

```
package ma.gov.fst.tpfinal;

public class Test {

    public static void main(String[] args) {

        Etudiant etudiant = new Etudiant();

        Personne personne = new Personne();

        String fctPeronne = etudiant.fonctionPersonne(personne);

        String fctString = etudiant.fonctionPersonne("test");

        System.out.println(fctPeronne);
        System.out.println(fctString);

    }

}
```

Constat : Si on change les types de paramètres d'une méthode au niveau d'une classe fille, on ne parle plus de la redéfinition mais plutôt d'une surcharge.

Règle 2 :

Lors de la redéfinition des méthodes, le type des paramètres ne peut pas être modifié. Si le type d'un paramètre est modifié, on parle d'une surcharge.

Formation Orienté Objet et Java de base

Atelier 5 : Redéfinition et utilisation de Final

Encadré par M.BOULCHAHOU

Modification du type de retour par un type différent

Un type différent est un type n'ayant aucune relation d'héritage avec le type de départ.

Exemple :

String et *int* n'ont pas de relation d'héritage.

Personne et *Etudiant* ont une relation d'héritage.

Maintenant, garder le même nom de la méthode et les mêmes types de paramètres.

Modifier le type de retour de la méthode *fonctionPersonne ()* en *int* par exemple.

```
package ma.gov.fst.tpfinal;

public class Etudiant extends Personne {

    private String age;

    public String getAge() {
        return age;
    }

    public void setAge(String age) {
        this.age = age;
    }

    public int fonctionPersonne(Personne p) {
        System.out.println( "Il s'agit de la redéfinition de cette
méthode");
        return 0;
    }
}
```

Vous devriez avoir une erreur de compilation qui dit :

The return type is incompatible with Personne.fonctionPersonne(Personne)

Règle 3 :

Formation Orienté Objet et Java de base

Atelier 5 : Redéfinition et utilisation de Final

Encadré par M.BOULCHAOUB

On comprend alors que le type de retour d'une méthode ne peut pas être modifié par un type différent.

Modification du type de retour par un type fils

Imaginons maintenant que le type de retour de la méthode fonctionPersonne () est Personne à titre d'exemple. Vous aurez alors la classe Personne.java comme suit :

```
package ma.gov.fst.tpfinal;

public class Personne {

    private String nom;

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public Personne fonctionPersonne(Personne p) {
        if (p instanceof Etudiant) {
            return new Etudiant();
        }

        if (p instanceof Professeur) {
            return new Professeur();
        }

        return new Personne();
    }
}
```

Redéfinir la méthode au niveau de la classe Etudiant en changeant le type de retour Personne par Etudiant.

```
package ma.gov.fst.tpfinal;
```

Formation Orienté Objet et Java de base

Atelier 5 : Redéfinition et utilisation de Final

Encadré par M.BOULCHAOUB

```
public class Etudiant extends Personne {  
  
    private String age;  
  
    public String getAge() {  
        return age;  
    }  
  
    public void setAge(String age) {  
        this.age = age;  
    }  
  
    public Etudiant fonctionPersonne(Personne p) {  
        return new Etudiant();  
    }  
  
}
```

Aucune erreur de compilation n'est signalée. Alors on peut donc aller d'un type parent vers type fils lors de la redéfinition.

Règle 4 :

On comprend alors que le type de retour d'une méthode peut être modifié par un type fils.

Le mot réservé « final »

« final » avant une méthode

Au niveau de la classe Personne, faire précéder la méthode fonctionPersonne () par le mot réservé final.

La classe Personne.java

```
package ma.gov.fst.tpfinal;  
  
public class Personne {  
  
    private String nom;  
  
    public String getNom() {  
        return nom;  
    }  
  
}
```

Formation Orienté Objet et Java de base

Atelier 5 : Redéfinition et utilisation de Final

Encadré par M.BOULCHAOUB

```
public void setNom(String nom) {
    this.nom = nom;
}

final public Personne fonctionPersonne(Personne p) {
    if (p instanceof Etudiant) {
        return new Etudiant();
    }

    if (p instanceof Professeur) {
        return new Professeur();
    }

    return new Personne();
}
}
```

Essayer de redéfinir cette méthode au niveau d'une classe fille (Etudiant par exemple)

La classe Etudiant.java

```
package ma.gov.fst.tpfinal;

public class Etudiant extends Personne {

    private String age;

    public String getAge() {
        return age;
    }

    public void setAge(String age) {
        this.age = age;
    }

    public Etudiant fonctionPersonne(Personne p) {
        return new Etudiant();
    }
}
```

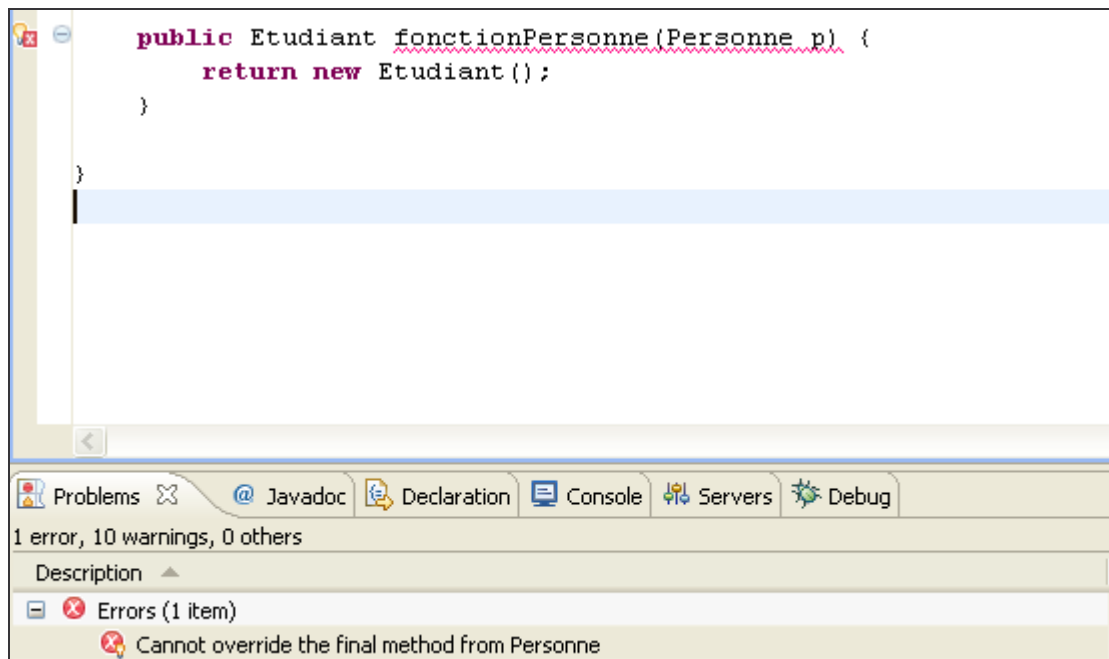
Vous auriez l'erreur de compilation suivante :

Cannot override the final method from Personne

Formation Orienté Objet et Java de base

Atelier 5 : Redéfinition et utilisation de Final

Encadré par M.BOULCHAHOU



Une méthode déclarée finale ne peut jamais être redéfinie au niveau des classes filles.

« final » avant une classe

Faites précéder la classe Personne par le mot réservé final.

Remarquer des erreurs de compilation au niveau des classes filles de la classe Personne.

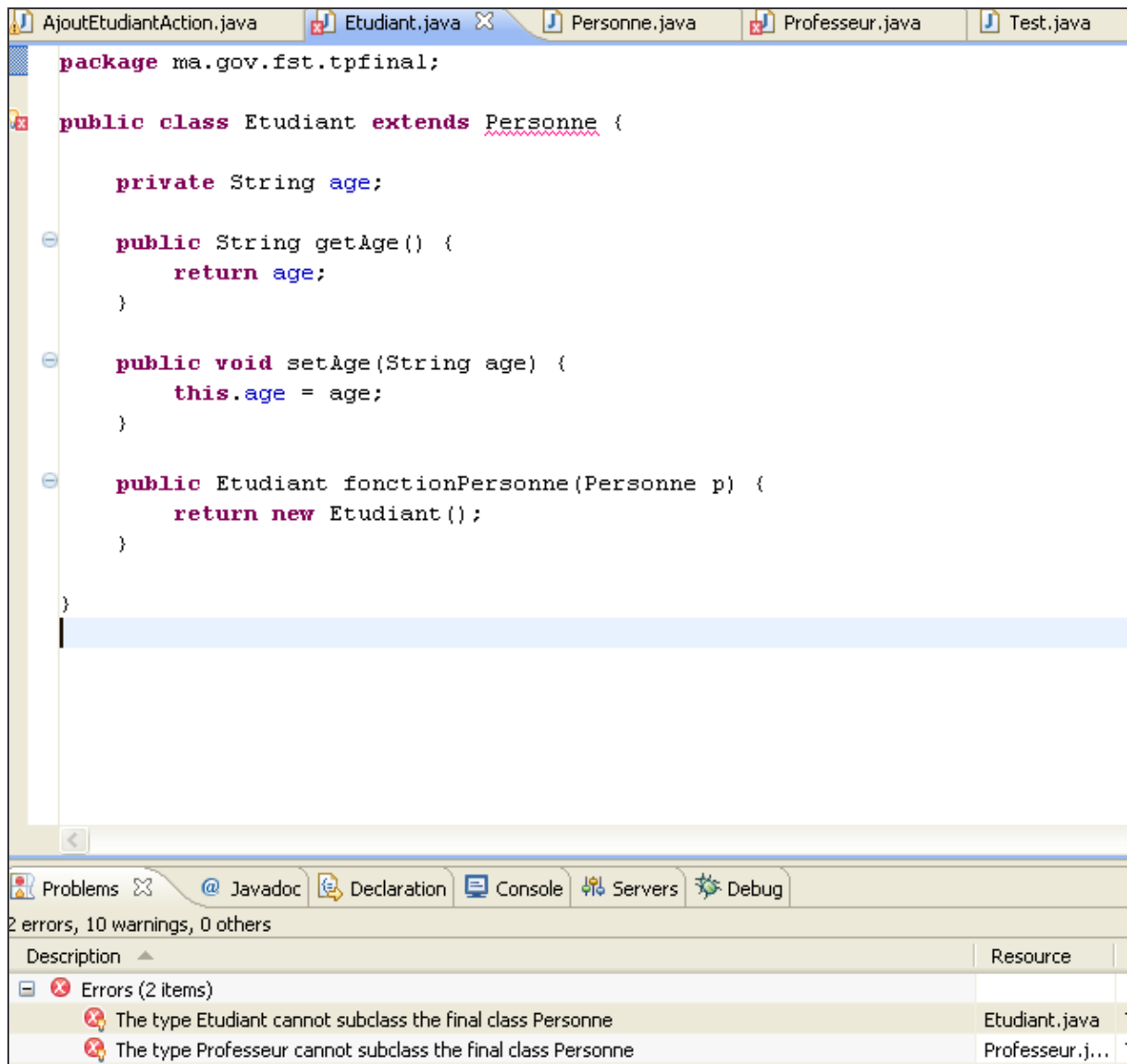
L'erreur de compilation en question est la suivante :

The type Etudiant cannot subclass the final class Personne

Formation Orienté Objet et Java de base

Atelier 5 : Redéfinition et utilisation de Final

Encadré par M.BOULCHAHOUB



Une classe déclarée finale ne peut jamais être étendue c.à.d. ne peut pas avoir des classes filles

« final » avant un attribut

Créer un attribut de type Double au niveau de la classe Personne. Nommez le PI. Déclarer cette attribut final.

Formation Orienté Objet et Java de base

Atelier 5 : Redéfinition et utilisation de Final

Encadré par M.BOULCHAHOU

```
package ma.gov.fst.tpfinal;

public class Personne {

    private final Double PI;

    private String nom;

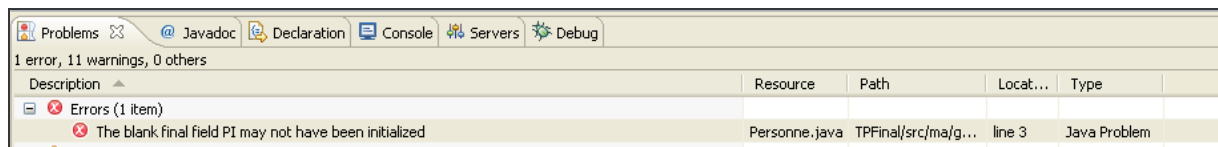
    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

}
```

Vous devriez avoir une erreur de compilation.

The blank final field PI may not have been initialized



Suite à cette erreur, on doit initialiser l'attribut PI. Il s'agit alors d'une constante.

Un attribut déclaré final est une constante. Son nom doit être en majuscule suite à la convention de nommage relative aux constantes.

L'ordre du final et public n'est pas important. On peut déclarer public final ou bien final public. La règle valable pour tous les modificateurs d'accès.

Formation Orienté Objet et Java de base

Atelier 5 : Redéfinition et utilisation de Final

Encadré par M.BOULCHAHOU

Conclusion

	Classe	Méthode	Attribut
Final	Une classe déclarée finale ne peut jamais être étendue c.à.d. ne peut pas avoir des classes filles	Une méthode déclarée finale ne peut jamais être redéfinie au niveau des classes filles.	Un attribut déclaré final est une constante.

FIN DE L'ATELIER