Les classes internes

Table des matières

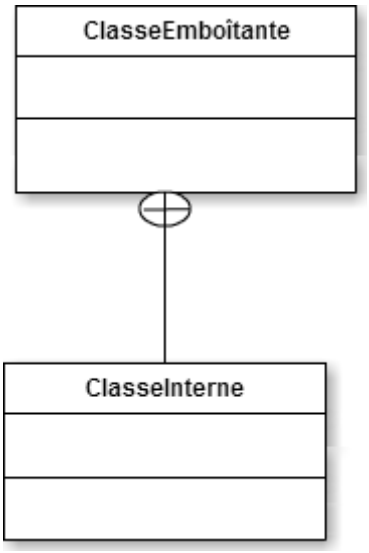
Les classes et interfaces emboîtées

Une classe peut être définie à l'intérieur d'une autre classe ou interface. De même une interface peut être définie à l'intérieur d'une autre classe ou interface. Elles sont considérées comme membre de la classe englobante au même titre que les champs et les méthodes et peuvent donc avoir les mêmes modificateurs : `public`, `private`, `protected`, `static`, …

Si une classe ou une interface est emboîtée dans une interface, elle sera bien entendu d'office `public` et `static`. Une interface emboîtée (même dans une classe) est toujours `static`.

La classe emboîtante et la classe emboîtée ont accès à tous les champs l'une de l'autre, comme si le code de la classe emboîtée faisait partie de la classe emboîtante.

En UML, on représente les classes internes de façon suivante :



Les classes et interfaces emboîtées statiques

Si une classe `StaticInterne` est déclarée `static` à l'intérieur d'une classe `Externe`, cette classe aura pour nom extérieur `Externe.StaticInterne`.

Les classes internes

Une classe emboîtée qui n'est pas `static` est appelée classe interne (inner class).

Une instance de la classe interne ne pourra exister qu'à l'intérieur d'une instance de la classe externe.

A l'intérieur d'une classe interne le mot clé `this` se rapporte à l'instance en cours de la classe interne. Pour avoir accès à l'objet en cours de la classe externe, il faut indiquer *`Externe`*`.this`.

Les classes internes (donc non `static`) ne peuvent pas avoir de membres `static` à l'exception de constantes (champs `final static`).

Une classe interne peut étendre n'importe quelle classe et implémenter n'importe quelle interface.

Une classe interne peut être déclarée `abstract` ou `final` (pas les deux, bien sûr).

Une classe interne peut avoir des sous-classes.

Exemple

Dans la classe Compte, on désire garder trace de la dernière opération effectuée. On pourra y arriver grâce à une classe interne `Operation`.

```
public abstract class Compte {

    private final String titulaire;
    private final String numeroDeCompte;
    private double solde = 0;
    private Operation derniere = new Operation("", 0);

    public Compte( String titulaire, String numeroDeCompte ) {
```

```
        this.titulaire = titulaire;
        this.numeroDeCompte = numeroDeCompte;
    }

    public Compte( String titulaire, String numeroDeCompte,
                                            double montantInitial ) {
        this(titulaire, numeroDeCompte);
        this.solde = montantInitial;
    }

    private void transaction( double montant ) {
        this.solde += montant;
    }

    public final void  depot( double montant ) throws
MontantNegatifException {
        if ( montant > 0 ) {
            transaction(montant);
            derniere = new Operation("dépot", montant);
            return;
         }
         throw new MontantNegatifException("depot : " + montant + " EUROS");
    }

    public final void retrait( double montant) throws
                    SoldeInsuffisantException, MontantNegatifException {
        if ( montant > 0) {
            if (getSolde() - montant >= getSeuil()){
                transaction(-montant);
                 derniere = new Operation("retrait", montant);
                return;
            }
             throw new SoldeInsuffisantException("retrait : solde = "
                + getSolde() + " EUROS, seuil = " + getSeuil() + " EUROS");
         }
        throw new MontantNegatifException("retrait " + montant + " EUROS");
    }

    public void virement(double montant, Compte c) throws
                SoldeInsuffisantException, MontantNegatifException,
                      VirementInterditException {
        throw new VirementInterditException(
                              "virement sur un compte" + getType());
    }

    public final String getTitulaire() {
        return this.titulaire;
    }

    public final String getNumeroDeCompte() {
        return this.numeroDeCompte;
    }

    public final double getSolde() {
        return this.solde;
    }

    public double getSeuil() {
        return 0;
    }

    public void setSeuil(double seuil) throws
                                    SeuilNonModifiableException {
        throw new SeuilNonModifiableException();
    }

    public abstract String getType();
```

```
    public String dernièreOperation() {
        return dernière.toString();
    }

    private class Operation {
        private String action;
        private double montant;
        Operation(String action, double montant) {
            this.action = action;
            this.montant = montant;
        }

        public String toString() {
            return numeroDeCompte + ": " + action + " " + montant;
        }
    } // fin de la classe Operation
} // fin de la classe Compte
```

Accès aux membres des classes externes et internes

Une classe interne a accès aux membres (même privés) de la classe englobante et ce sans aucune qualification.

La classe englobante a accès aux membres (même privés) de la classe interne mais uniquement via un objet de cette classe (sauf pour les membres `static` des classes emboîtées `static` – rappel : les classes internes (i.e. non `static`) ne peuvent pas avoir de membres `static`)

Hiérarchie de classes emboîtées

Une classe ou une interface emboîtée peut elle-même avoir des classes ou interfaces emboîtés et ainsi de suite. Cette possibilité est néanmoins à proscrire. Pour des raisons de lisibilité, il vaut mieux se limiter à un niveau d'emboîtement.

Classe interne locale

On peut définir des classes internes locales à une méthode, un constructeur ou un bloc d'initialisation. Elles jouent alors le même rôle qu'une variable locale. Une telle classe interne est donc temporaire et inaccessible en dehors de la méthode. Le seul modificateur permis dans ce cas est `final`.

Dans une telle classe interne, on pourra accéder à tout ce qui est accessible dans le bloc où elle est définie. Toutefois **pour pouvoir accéder à une variable locale ou à un paramètre, ceux-ci doivent être déclarés `final`**.

Classe interne anonyme

On peut même définir des classes internes anonymes pour autant qu'elles soient sous classe d'une autre classe ou qu'elles implémentent une interface. Et comme toute classe dérive d'`Object` !

Ces classes sont définies au moment où on les instancie avec `new` suivi d'un appel à un constructeur de la classe Parent ou à un constructeur sans paramètre dans le cas de l'implémentation d'une interface.

Une telle classe interne anonyme ne peut pas définir de nouveaux constructeurs : elle n'a en effet pas de nom.

Elle ne peut pas non plus avoir de clauses `extends` ou `implements` explicite.

En général, on n'utilisera de classe anonyme que si leur code ne dépasse pas cinq ou six lignes.

Leur usage est fréquent pour ajouter un `Listener` à un objet graphique.