Classes internes membres statiques et d'instance

Description du contenu :
Cette séquence traite des classes internes membres en Java. Après avoir situé ces classes dans le cadre plus générales des classes internes en Java, les connaissances à acquérir sont étudiées par l'étude d'une classe interne membre statique et d'une classe interne membre d'instance.

Auteur : Yannick.Parchemal@parisdescartes.fr
Objectifs : savoir définir et utiliser des classes internes membres

Connaissances acquises : classes membres d'instance et statiques
Compétences acquises : comprendre un programme utilisant des classes membres. Savoir définir et utiliser de telles classes
Pré requis : classes et instances en Java

Classe, classe interne, classe interne membre

Classe

Les membres d'une classe sont

- **des attributs (d'instance ou statique),**
- **des constructeurs,**
- **des méthodes (d'instance ou statique)**
- **<u>et peuvent être aussi</u> des classes appelées classes internes**

Classe interne

C'est une classe définie dans une autre classe

classe interne membre

- statique
- d'instance

- **définie au même niveau que les autres membres**
- **a un niveau de visibilité (private, package, protected, public)**

classe locale

définie dans une méthode ou dans un constructeur

Les classes membres sont souvent appelées classes internes…

Classes membres internes

```
public class UneClasseEnglobante {

    private static int att1 ;

    private int att2 ;

    public int meth(String s){…}

    public static class UneClasseMembreStatique {
        private int b ;
     …
        }
    public class UneClasseMembreInstance {
        private int c ;
    …
    }
}
```

Nom pleinement qualifié (ou nom complet) des classes membres internes

```
package yp.test;
public class UneClasseEnglobante {
        .....
        public static class UneClasseMembreStatique {
                        ...
        }
        public class UneClasseMembreInstance {
                        ...
        }
}
```

nom pleinement qualifié des classes :
yp.test.UneClasseEnglobante;
yp.test.UneClasseEnglobante.UneClasseMembreStatique;
yp.test.UneClasseEnglobante.UneClasseMembreInstance;

import possible :

```
import yp.test.UneClasseEnglobante;
import yp.test.UneClasseEnglobante.UneClasseMembreStatique;
import yp.test.UneClasseEnglobante.UneClasseMembreInstance;
```

Classes membres statiques

Les classes membres statiques ont accès à tous les membres statiques de la classe englobante

```
package test;
public class UneClasseEnglobante {

        private static int base=10 ;

        public static class UneClasseMembreStatique {
            private int total ;
            public UneClasseMembreStatique(int val){
                this.total=val+base;
            }
            public int getTotal (){
               return this.total;}
    }
}
```

Exemple d'utilisation (dans une autre classe) :

```
import test.UneClasseEnglobante.UneClasseMembreStatique ;
…
UneClasseMembreStatique ucms = new  UneClasseMembreStatique(5):
System.out.println(ucms.getTotal( )); // 15
```

Les classes membres statiques permettent
de bien packager ses classes : c'est leur principale utilité

Classes membres d'instance

```
public class Banque {

    public Banque(String nom){ …}
    public String toString( ){…}

    public  class Compte {
        public Compte(String nomTitulaire){…}
        public void ajouterOperation(int montant){…}
        public String toString( ){..}
}
```

une instance d'une classe membre d'instance
est toujours liée à une instance de la classe englobante

```
Banque banque= new Banque("YPB");
Compte compte1 = banque.new Compte("Dupond");
compte1.ajouterOperation(100);
Compte compte2 = banque.new Compte("Durand");
compte2.ajouterOperation(300);
System.out.println(compte1.toString()); //Compte :Dupond 100 YPB
System.out.println(banque.toString());  //Banque YPB capital 400
```

Accès aux membres de la classe englobante

la classe membre d'instance a accès à tous les membres de la classe englobante

```
public class Banque {
    private String nom;  private long capital=0;

    public Banque(String nom){this.nom = nom;}
    public String toString( ){
        return "Banque "+this.nom+" capital "+this.capital;
        }
    public  class Compte {
        private String nomTitulaire;  private int solde=0;

        public Compte(String nomTitulaire){
            this.nomTitulaire=nomTitulaire;
        }
        public void ajouterOperation(int montant){
            this.solde+=montant;
            Banque.this.capital+=montant;
            }
        public String toString( ){
            return "Compte :"+nomTitulaire+" "+solde+" "+Banque.this.nom;
        }}
```

this implicite

```
public class Banque {
    private String nom;  private long capital=0;

    public Banque(String nom){ this.nom = nom;}// this. obligatoire
    public String toString( ){
        return "Banque "+this.nom+" capital "+this.capital;
        }
    public  class Compte {
        private String nomTitulaire;  private int solde=0;

        public Compte(String nomTitulaire){
            this.nomTitulaire=nomTitulaire; // this. obligatoire
        }
        public void ajouterOperation(int montant){
            this.solde+=montant;
            Banque.this.capital+=montant;
            }
        public String toString( ){
            return "Compte :"+this.nomTitulaire+" "+solde+" "+Banque.this.nom;
        }}
```

Classes internes membres d'instances : syntaxe spécifique

Création d'une instance d'une classe interne membre d'instance

<instance de la classe englobante>.new

exemple :
Banque banque= **new Banque("YPB");**
Compte compte1 = banque.**new Compte("Dupond");**

Accès à partir de la classe interne à l'instance de la classe englobante

<nom de la classe englobante>.this

```
public class Banque {
        private long capital=0;
        …
        public  class Compte {
                ...  Banque.this.capital+=montant;// ou capital+=montant;
```

Classes internes membres : lexique

Terme	
Classe interne	Une classe interne est une classe définie dans une autre classe
Classe interne membre	Les classes internes membres sont définies à l'intérieur d'une classe au même niveau que les autres membres de la classe (constructeurs, méthodes, attributs). Ce peut être des membres d'instance ou statiques
Classe englobante	La classe dans laquelle est définie une classe interne est appelée la classe englobante

Classes internes membres

Les activités associées sont sur le site du cours.
Il est nécessaire de faire ces activités pour pouvoir considérer la séquence comme terminée.

Classes locales et lambda expressions

Description de la séquence

Description du contenu :

Durée de travail estimée :
Auteur : Yannick.Parchemal@parisdescartes.fr
Objectifs : étude des classes locales

Connaissances acquises : classes locales et lambda expressions
Compétences acquises : comprendre un programme utilisant des classes locales. Savoir définir et utiliser de telles classes
Pré requis : classes internes membres

Classe, classe interne, classe interne membre

Classe

Les membres d'une classe sont

- **des attributs (d'instance ou statique),**
- **des constructeurs,**
- **des méthodes (d'instance ou statique)**
- **<u>et peuvent être aussi</u> des classes appelées classes internes**

Classe interne

C'est une classe définie dans une autre classe

classe interne membre

- **définie au même niveau que les autres membres**

classe locale

classe locale (nommée)

classe locale anonyme

- **définie dans une méthode ou dans un constructeur**

lambda expression

- **une syntaxe simplifiée pour la definition de classes locales anonymes**

La méthode sort de java.util.List

En vue de l'exemple suivant …

```
public interface List <E> { …
/** Sorts this list according to the order induced by the specified Comparator.
 *   All elements in this list must be mutually comparable using the specified comparator
 * @params   c the Comparator used to compare list elements. A null value indicates that the
elements' natural ordering should be used
* @ since 1.8
**/
default void sort(Comparator<? super E> c)
```

```
public interface Comparator<T> { …
/**
* @param o1 - the first object to be compared.
* @param o2 - the second object to be compared.
* @ return a negative integer, zero, or a positive integer as the first argument is less than
*                          , equal to, or greater than the second.
**/
int compare(T o1, T o2)
```

La méthode sort de java.util.List

Exemple d'utilisation de la méthode sort de java.util.List :

```
public class Critere implements Comparator<Integer>{

        public Critere (boolean ordreCroissant){
            this.ordreCroissant = ordreCroissant;
            }
        @Override
        public int compare(Integer int1, Integer int2) {
            return (this.ordreCroissant?1:-1)* int1.compareTo(int2);
        }
}

// dans une fonction …
List<Integer> list= Arrays.asList(3,17,81,9,12);
list.sort(new Critere(false)); // list vaut alors [81, 17, 12, 9, 3]
```

Classes locales

```
public static void trier(List<Integer> list,boolean ordreCroissant){

    class Critere implements Comparator<Integer>{

        @Override
        public int compare(Integer int1, Integer int2) {
            return (ordreCroissant?1:-1)* int1.compareTo(int2);
        }
    }
    list.sort(new Critere());
}

public static void main(String [] args){

    List<Integer> list= Arrays.asList(3,17,81,9,12);
    trier(list,false);
    System.out.println(list);// [81, 17, 12, 9, 3]
    trier(list,true);
    System.out.println(list);// [3, 9, 12, 17, 81]
}
```

Classes locales anonymes

Classe locale nommée

```
public static void trier(List<Integer> list,boolean ordreCroissant){
    class Critere implements Comparator<Integer>{
        @Override
        public int compare(Integer int1, Integer int2) {
            return (ordreCroissant?1:-1)* int1.compareTo(int2);
        }
    }
    list.sort(new Critere());
}
```

Classe locale anonyme

C'est une classe locale sans nom définie et instanciée sur la même instruction.

```
public static void trier(List<Integer> list,boolean ordreCroissant){
    list.sort(new Comparator<Integer>( ){
        @Override
        public int compare(Integer int1, Integer int2) {
            return (ordreCroissant?1:-1)* int1.compareTo(int2);
        }
        );
}
```

Classes locales

Elles sont définies dans des fonctions à l'emplacement d'une instrcution

Comme pour les classes internes membres d'instance

- une instance d'une classe locale est toujours liée à une instance de la classe englobante
- la classe locale a accès à tous les membres de la classe englobante

Accès aux variables de la fonction

la classe locale a accès à
- **toutes les variables "final" de sa zone de portée**

Accessibilité de la classe locale

- **les règles d'accessibilité sont les mêmes que pour une variable**
  - **accessibles dans le bloc où elle est définie juste après sa définition**

Lambda expression

Classe locale anonyme

```
public static void trier(List<Integer> list,boolean ordreCroissant){

    list.sort(new Comparator<Integer>( ){
        @Override
        public int compare(Integer int1, Integer int2) {
            return (ordreCroissant?1:-1)* int1.compareTo(int2);
        }
        );
}
```

Les lambda expression sont une autre syntaxe utilisable pour les classes locales anonymes lorsque la classe anonyme n'a qu'une seule méthode héritée (en dehors des méthodes de la classe Object)

Lambda expression

```
public static void trier(List<Integer> list,boolean ordreCroissant){
    list.sort( (int1,int2) -> return (ordreCroissant?1:-1)* int1.compareTo(int2));
}
```