

Interfaces et classes abstraites

○ Classe «normale»

- une **classe** «normale»
- ✓ déclare un comportement
 - ▲ ses méthodes publiques
- ✓ définit une implémentation
 - ▲ ses méthodes ont du code



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

2

Interfaces

○ Définition

- une **interface** déclare un comportement
 - ✓ il n'y a pas d'implémentation (pas de code)
 - ✓ elle présente ce dont un objet est capable, sans dire comment
- une **interface** est défini par le mot clé **interface**

```
public interface TestProduit {  
    public String getMarque();  
    public void setMarque(String marque);  
    public double getPrix();  
    public void setPrix(double prix);  
    public double getScore();  
    public void setScore(double score);  
}
```



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

3

Interfaces

○ Utilisation

- une **classe** implémente l'**interface**
- pour implémenter une interface il faut utiliser le mot clé **implements**, de la manière suivante :

```
public class TestPC implements TestProduit {  
    ...  
}
```

- lorsqu'on définit une **classe** qui implémente une **interface**, cela signifie que l'on s'engage formellement de fournir l'implémentation des méthodes déclarées dans l'**interface**



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

4

Interfaces

○ Exemple

- soit la classe Shape suivante qui définit deux méthodes : area() et draw()

```
public interface Shape {  
    public double area();  
    public void draw();  
}
```

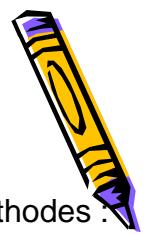
- soit une classe Circle qui implémente l'interface Shape



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

5



Interfaces

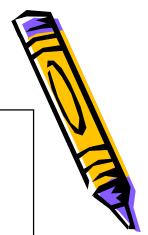
```
public class Circle implements Shape {  
    private double radius;  
    ...  
    public double area() {  
        return Math.PI * radius * radius;  
    }  
    public void draw() {  
        ... // code pour afficher un cercle  
    }  
}
```



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

6



Interfaces

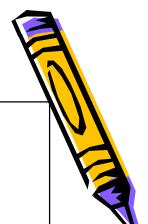
```
public class Rectangle implements Shape {  
    private double width;  
    private double height;  
    ...  
    public double area() {  
        return width * height;  
    }  
    public void draw() {  
        ... // code pour afficher un rectangle  
    }  
}
```



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

7



Interfaces

○ Représentation UML

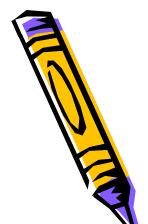
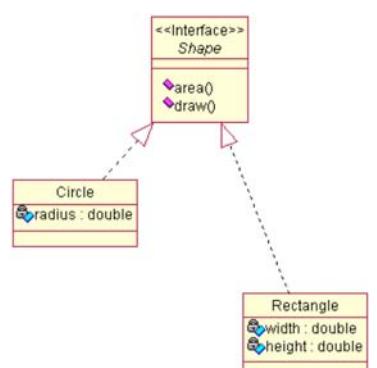
- le mot clé **implements** est indiqué par
- les classes **Circle**, **Rectangle** implémentent l'interface **Shape**



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

8



Interfaces

```
public class Geom {  
    ...  
    public static void areaPrintln (Shape shape) {  
        System.out.println("") + shape.area());  
    }  
  
    public static void main(String[] args) {  
        // intéressé uniquement par l'interface  
        Shape cercle = new Circle();  
        Shape rect = new Rectangle();  
        areaPrintln(cercle);  
        areaPrintln(rect);  
    }  
}
```



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

9



Interfaces

○ Exemple

- soit deux interfaces Shape et Drawable

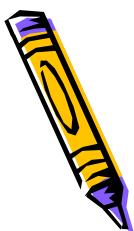
```
public interface Shape {  
    public double area();  
}  
public interface Drawable {  
    public void draw();  
}
```



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

10



Interfaces

- une classe peut implémenter plusieurs interfaces
- exemple :

```
public class Rectangle implements Drawable, Shape {  
    ...  
    public double area() //méthode de Shape  
        return width * height;  
    }  
    public void draw() //méthode de Drawable  
        ... // code pour afficher un rectangle  
    }  
}
```



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

11



Classes abstraites

○ Définition

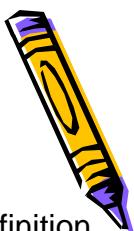
- la définition d'une classe abstraite est entre la définition d'une classe «normale» et d'une interface
 - elle déclare un comportement
 - elle ne définit pas d'implémentation
- pour définir une classe abstraite, on utilise le mot clé **abstract**
- exemple : **public abstract class Shape**



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

12



Classes abstraites

○ Méthode abstraite

- une méthode sans implémentation est obligatoirement abstraite et est défini par le mot clé **abstract**

```
public abstract void draw();
```

○ Classe abstraite

- une classe dont une méthode est abstraite est obligatoirement abstraite et est donc défini par le mot clé **abstract**

```
public abstract class Shape {  
    public abstract void draw();  
}
```



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

13



Classes abstraites

○ Méthodes «normales» et abstraites

- une classe abstraite peut mélanger des méthodes abstraites et des méthodes «normales»
- exemple :

```
public abstract class Shape {  
    ...  
    // recyclage de l'implémentation  
    public Point getPosition() {  
        return posn;  
    }  
    // recyclage de l'interface  
    public abstract void draw();  
}
```



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

14



Classes abstraites

○ Utilisation

- pour utiliser les méthodes d'une classe abstraite, on doit passer par l'héritage
- dans ce cas, on doit fournir le code de toutes les méthodes abstraites de la super-classe abstraite

```
public class Rectangle extends Shape {  
    ...  
    public void draw() {  
        ... // code pour afficher un rectangle  
    }  
}
```

Obligation de fournir le code
Pour les méthodes **abstract**



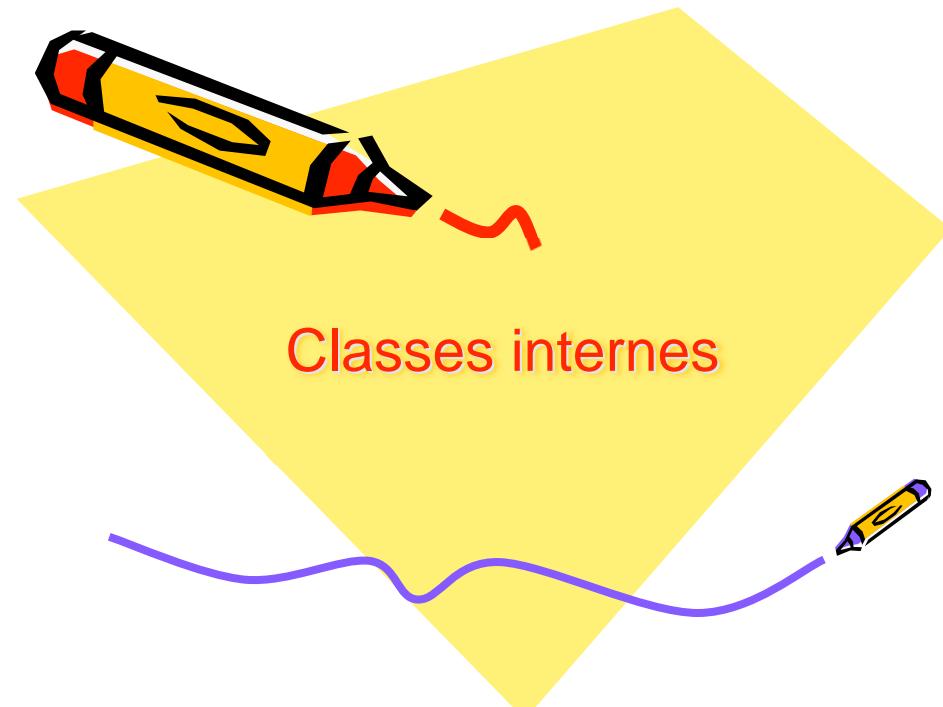
Peter Sander

ESSI-Université de Nice Sophia
Antipolis

15



Classes internes



Classes Internes

- une classe Java peut contenir
 - ✓ des **variables**
 - ▲ d'instance
 - ▲ static
 - ✓ des **méthodes**
 - ▲ d'instance
 - ▲ static
 - ✓ ...mais aussi des **classes**
 - ▲ d'instance
 - ▲ static



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

17



Taxonomie

○ Types de classes internes

- il existe 2 types de classes internes
 - ✓ des classes définies à l'extérieur de toute méthode (au même niveau que les méthodes et les variables d'instance ou de classe)
 - ✓ des classes définies à l'intérieur d'une méthode



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

18



Classes internes non incluses dans une méthode (1)

- Le code de ces classes internes est défini à l'intérieur d'une autre classe, appelée **classe englobante**, au même niveau que les variables d'instance et les méthodes

```
public class Toto //classe englobante
    private int x;
    class Titi {
        ... // code de la classe interne
    }
    public String m() { ... }
    ...
}
```



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

19

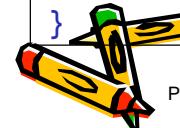


Classes internes non incluses dans une méthode (2)

- Classes «*top-level*»
 - ✓ Contenues dans un package
- Classes internes
 - ✓ Contenues dans une classe

```
package foo;
public class Toto {
    ...
}
```

```
package foo;
public class Titi {
    ...
}
```



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

20



Classes internes non incluses dans une méthode (3)

○ Modificateurs

- une telle classe peut avoir les mêmes degrés d'accessibilité que les membres d'une classe : **private**, **public**, **protected**, **package**
- elle peut aussi être abstract ou final

○ Visibilité

- une classe interne peut accéder aux membres **private** de la **classe englobante**



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

21

Classes internes non incluses dans une méthode (4)

○ Nommage d'une classe interne

- une classe interne ne peut avoir le même nom qu'une **classe englobante** (quelque soit le niveau d'imbrication)
- soit une **classe englobante ClasseE** et une **classe interne Classel** définie dans **ClasseE**. Le nom de la classe interne sera de la forme **ClasseE.Classel**

○ Importer des classes internes

- on peut importer une classe interne
import ClasseE.Classel;
- on peut aussi importer toutes les classes internes d'une classe **import ClasseE.*;**



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

22

Classes internes non incluses dans une méthode (5)

○ Types de classe internes non incluses dans une méthode

- Il existe deux types de classes internes définies à l'extérieur d'une méthode
- ✓ les classes **static** (**nested class** en anglais)
▲ leurs instances ne sont pas liées à une instance de la classe englobante
- ✓ les classes non **static** (**inner class** en anglais)
▲ une instance d'une telle classe est liée à une instance de la classe englobante



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

23

Les classes internes **static** et non **static**

• Classe non static (**Inner class**)

```
public class Toto {  
    ...  
    class Titi {  
        ...  
    }  
    ...  
}
```

- liée à un objet de type **Toto**



Peter Sander

• Classe static (**static nested class**)

```
public class Toto {  
    ...  
    static class Titi {  
        ...  
    }  
    ...  
}
```

- liée à la classe **Toto**

ESSI-Université de Nice Sophia
Antipolis

24

Classes internes static

- Les classes internes static (nested classes) n'ont pas accès aux membres non-static de la classe englobante
- une classe interne static est référencée par rapport à la classe englobante

```
public class Foo extends Toto.Titi {  
    ...  
}
```

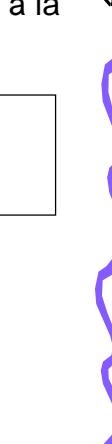
- c'est une façon de lier des classes...liées



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

25



Classe internes non static

- Une classe interne non static (inner class) est liée à un objet instance de la classe englobante
- une classe interne non static (inner class) peut être
 - ✓ membre
 - ▲ correspond aux méthodes et variables d'instances
 - ✓ locale
 - ▲ correspond aux variables locales
 - ✓ anonyme



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

26



Classes membres

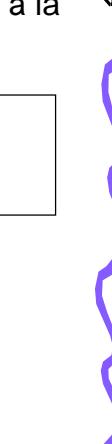
- Les **classes membres** sont similaires aux variables et méthodes d'instance
- elles ont accès à tous les membres de la classe englobante
 - ✓ même aux membres **private**
- la classe englobante a accès à tous les membres d'une **classe membre**
 - ✓ même aux membres **private**
- chaque instance est associée avec une instance de la classe englobante
 - ✓ comme pour les méthodes et les variables



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

27



Exemple (1)

```
public class TotoEtHelper {  
    private int count = 0;  
  
    public TotoEtHelper() { // constructeur  
        this.new Helper();  
        System.out.print(count + " fois...");  
        this.new Helper();  
        System.out.println(count + " fois");  
    }  
  
    private class Helper {  
        private Helper() { // constructeur  
            count++;  
        }  
    }  
}
```

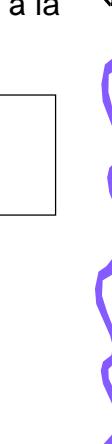


Quel sera le résultat ?

Peter Sander

ESSI-Université de Nice Sophia
Antipolis

28



Exemple (2)

```
> java TotoEtHelper  
1 fois...2 fois  
>  
> ls Toto*class  
TotoEtHelper$Helper.class  
TotoEtHelper.class
```



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

29



Classes locales (1)

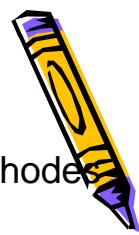
- Les **classes locales** sont similaires aux méthodes et variables locales
- elles sont déclarées localement dans un bloc de code
 - ✓ dans une méthode
 - ✓ dans un bloc initialisateur
- elles sont utilisées de la même façon que les classes membres



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

30

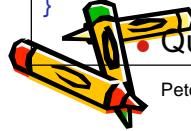


Classes locales (2)

○Visibilité

- les **classes locales** sont visibles uniquement à l'intérieur du bloc
 - ✓ elles accèdent aux variables visibles depuis le bloc
 - ✓ elles accèdent aux variables **final** locales au bloc

```
public class Toto {  
    int deux = 2;  
    ...  
    public void mesThode(int i, final String s) {  
        final float pi = 3.14;  
        ...  
        class Lowcal {...}  
    }  
    ...  
}
```

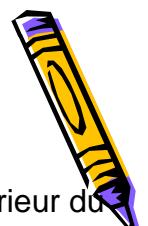


Peter Sander

ESSI-Université de Nice Sophia
Antipolis

31

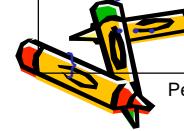
• Qui est visible depuis **Lowcal** ?



Classes anonymes (1)

- Les **classes anonymes** sont des **classes locales** sans nom
- leur définition et instanciation sont combinées

```
public class Toto {  
    ...  
    public Titi uneMethode() {  
        ...  
        return new Titi() { // création de classe  
            anonyme  
            // définition de Titi  
        };  
    }  
}
```



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

32



Classes anonymes (2)

○ Syntaxe

- Classe

```
new nom-de-classe([liste d'args]) {  
    définition de la classe  
}  
  
...  
toto = faisToto(new Titi(arg1, arg2) {  
    // définition de la classe anonyme  
});  
...
```

- la classe anonyme hérite de **Titi**
- **arg1, arg2** passées au constructeur de **Titi**
- classe anonyme. n'a pas de nom --> pas de constructeur



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

33



Classes anonymes (3)

○ Syntaxe

- Interface

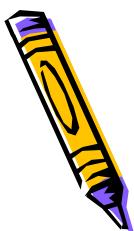
```
new nom-d'interface() {  
    définition de l'interface  
}  
  
• implémente l'interface nommée
```



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

34



Utilisation des classes anonymes (1)

- Une classe anonyme peut convenir quand
 - ✓ sa définition est brève
 - ✓ une seule instanciation
 - ✓ un constructeur n'est pas nécessaire
 - ✓ utilisée tout de suite après sa définition
 - ✓ un nom ne contribue rien à la compréhension du code
- il ne faut pas en abuser !



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

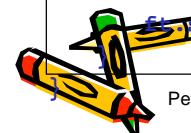
35



Utilisation des classes anonymes (2)

- Elles sont utilisées, le plus souvent pour réagir aux événements

```
public FrameTest extends JFrame {  
    ...  
    public static void main(java.lang.String[] args) {  
        final FrameTest ft = new FrameTest("Frame Test");  
        ft.addWindowListener(new WindowAdapter() {  
            public void windowClosing(WindowEvent we) {  
                ft.dispose();  
                System.exit(0);  
            }  
        });  
        ft.setSize(250, 150);  
        ft.setVisible(true);  
    }  
}
```



Peter Sander

ESSI-Université de Nice Sophia
Antipolis

36

