LPS: Clases Anidadas

Federico Peinado
www.federicopeinado.es

Depto. de Ingeniería del Software e Inteligencia Artificial
disia.fdi.ucm.es

Facultad de Informática
www.fdi.ucm.es

Universidad Complutense de Madrid
www.ucm.es

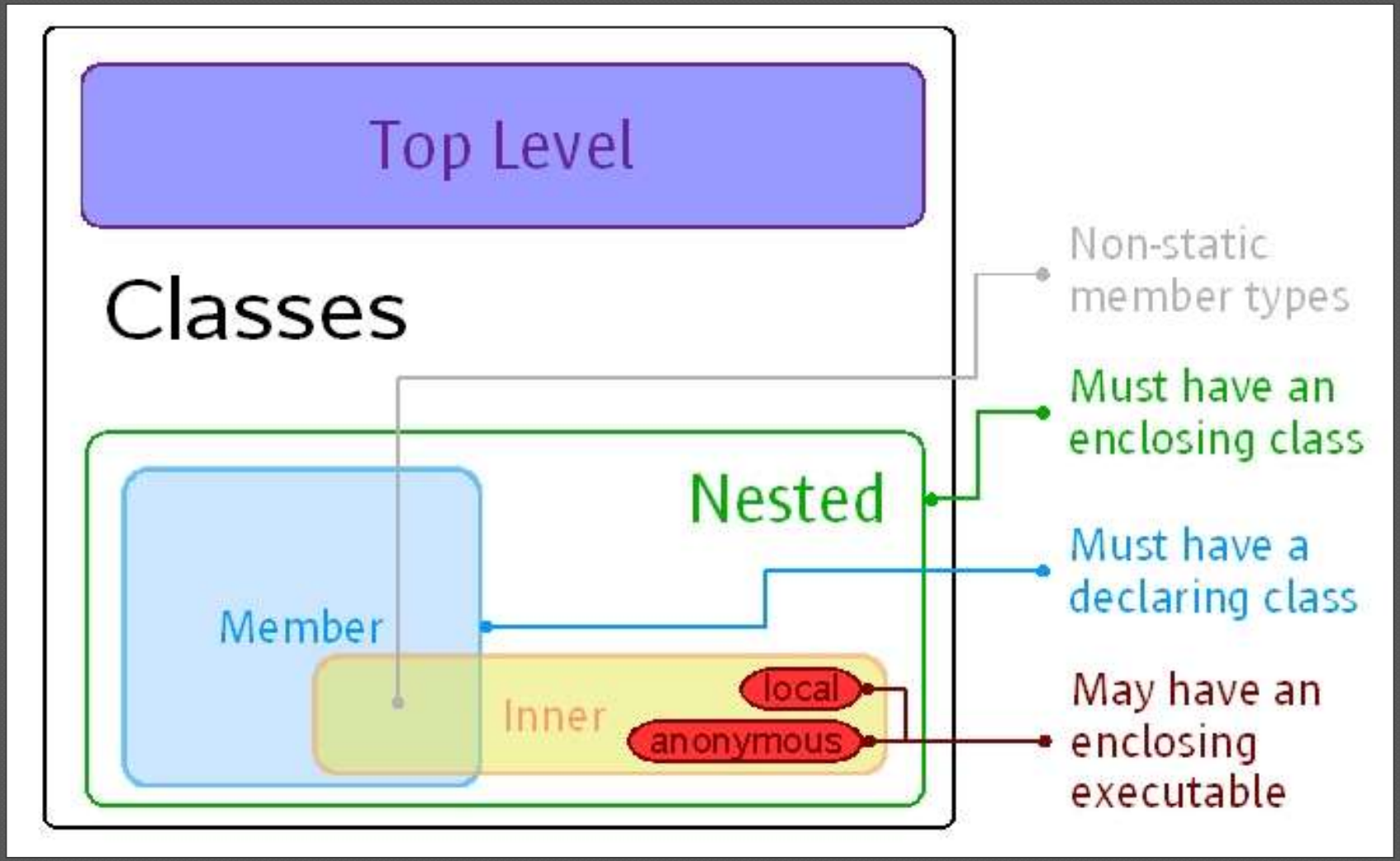
Clases anidadas

- A diferencia de las *clases de nivel superior* (las que conocemos), las clases anidadas se encuentran confinadas dentro de otra clase
  - *¡Ojo! Todo lo que estudiaremos aquí es también válido en general para interfaces anidados*
- Referencias
  - Java Tutorial http://download.oracle.com/javase/tutorial/java/javaOO/nested.html
  - Programmer's Guide to Java Certification: A Comprehensive Primer. Addison-Wesley (Capítulo 7) http://java.sun.com/developer/Books/certification/certbook.pdf

Utilidad de las clases anidadas

- Permiten hacer los paquetes de clases más compactos
  - Si la clase A sólo será utilizada por la clase B (o es auxiliar de B, o incluso carece totalmente de sentido sin B), *anidamos A dentro de B*
- Aumentan la encapsulación
  - La clase anidada puede ocultarse de otras distintas a su clase exterior,y a la vez puede acceder a atributos de su clase exterior sin que estos dejen de ser privados
- Mejoran la legibilidad del código
  - Mientras una clase anidada sean pequeña, ayuda tenerla cerca de donde se usa (o incluso declararla sólo para un único uso, como veremos)

Tipos de clases anidadas



Tipos de clases anidadas

- Clase miembro, cuando *figura en la zona de declaración de la clase exterior* (y por lo tanto tiene nombre propio)
- Clase interior, cuando *no es estática* (¡ojo! los interfaces son "implícitamente estáticos")
  - Clase local, cuando *es interior y se declara con nombre propio dentro de un bloque* (como puede ser el cuerpo de un método o de un constructor)
  - Clase anónima, cuando *es interior y se declara sin nombre justo cuando creamos un ejemplar de ella*
    - Se usa una sintaxis especial de **`new`**

Clases miembro *estáticas*

- Clases que podrían estar perfectamente en el exterior (se usan igual), pero las hemos anidado por motivos pragmáticos
  - Su nombre incluye el de la clase exterior

    `ClaseExterior.ClaseMiembroEstatica`
  - Como el compilador crea un fichero binario por cada clase, a estas las llama así:

    `ClaseContenedora$ClaseMiembroEstatica.class`
- Admiten todos los modificadores de acceso como cualquier miembro de una clase
- Pueden acceder de manera directa *sólo a los miembros estáticos* de su clase exterior (incluso aunque sean privados, como ya se dijo)

Ejemplo

```
public class ClaseExterior {

   public void metodoNoEstatico() {
     System.out.print("metodoNoEstatico, ClaseExterior");
   }

   private static class ClaseMiembroEstatica {
      private static int i;
      private int j;
      private static void metodoEstatico() {
        //metodoNoEstatico(); ¡MAL!
         System.out.print("metodoEstatico, ClaseMiembroEstatica");
      }

      protected static class OtraClaseMiembroEstatica {
         public void metodoNoEstatico() {
            metodoEstatico(); // BIEN
         }
      }
   }
}
```

Clases miembro *interiores*

- Clases miembro *no estáticas* que suelen servir para realizar funciones auxiliares
  - Sólo se usan desde un ejemplar de su clase exterior
  - Se nombran igual que las otras clases miembro: `ClaseExterior.ClaseMiembroInterior`
  - Se crean ficheros binarios con la misma nomenclatura: `ClaseContenedora$ClaseMiembroInterior.class`
- Admiten todos los modificadores de acceso
- Pueden acceder de manera directa a todos los miembros de la clase exterior
- Obviamente, no tienen *miembros estáticos*

Ejemplo

```
public class ClaseExterior {

   private String msg = "Mensaje";
   public class ClaseMiembroInterior {
      // private static int i; ¡MAL!
      private String str;
      public InteriorNoEstatica() {
        str = msg; // BIEN
      }
      public print() {
         System.out.println(str);
         System.out.println(msg);
      }
   }

   public static void main(String args[]) {
      ClaseExterior ref = new ClaseExterior();
      ClaseExterior.ClaseMiembroInterior interior;
      // interior = new ClaseExterior.InteriorNoEstatica(); ¡MAL!
      interior = ref.new InteriorNoEstatica(); // BIEN
      interior.print();
   }
}
```

Clases locales

- Son visibles sólo dentro del bloque donde se declaran y sólo pueden instanciarse allí
  - El compilador las da nombre unívocos gracias a un *contador numérico*
    `ClaseExterior$`*`18`*`$ClaseLocal.class`
- No admiten modificadores de acceso
- No pueden declararse como estáticas
- No tienen miembros estáticos (excepto si son estáticos *y finales = constantes*)
- Como un ejemplar de una clase local puede "vivir" más que el bloque de código donde está contenido, sólo puede acceder a los parámetros y variables locales que sean además *finales*

Ejemplo

```
public class Forma {
   void dibuja() {
      System.out.println("Forma exterior");
   }
}

class PintorExterior {
   public Shape crearCirculo(final float radio) {
      class CirculoLocal extends Forma {
         void dibuja() {
            System.out.println("Círculo de radio"
                                 + radio);
         }
      }
      return new CirculoLocal();
   }
}
```

Clases anónimas

- Como se declaran justo al crear un ejemplar de ellas, no se puede crear más de un ejemplar de una clase anónima
  - Se usa **`new`** más el nombre de otra clase que extendemos (o de un interfaz que implementamos) a continuación y que nos sirve como "base"
  - El compilador también las nombra con un contador numérico **`ClaseExterior$24.class`**
- Sirven para crear objetos "al vuelo" con algo de código nuevo y/o específico
- En muchos aspectos se comportan como clases locales:
  - No admiten modificadores de acceso
  - No pueden declararse como estáticas
  - No tienen miembros estáticos (salvo estáticos *y finales*)
  - Sólo acceden a parámetros y variables locales que sean *finales*

Ejemplo (real)

```
public interface ActionListener {
   void actionPerformed(ActionEvent e);
}

public JButton … {
   void addActionListener(ActionListener l) { … }
}

class Ventana extends JPanel {
   Ventana() {
     JButton b = new JButton(“Antes”);
     b.addActionListener(
        new ActionListener (){
          public void actionPerformed(ActionEvent e) {
            b.setText(”Después");
          }
        }
     );
   }
}
```

Críticas, dudas, sugerencias…

Federico Peinado

www.federicopeinado.es