SCJP – Capitulo 9

Clases Internas

Agenda

- Clases Internas
- Clases Internas (en Métodos)
- Clases Anónimas
- Clases Anidadas (estáticas)

Clase Internas

- Es sabido que la Reusabilidad, Flexibilidad y Extensibilidad se consigue especializando las clases de nuestro diseño orientado a objetos.
- Cualquier otro comportamiento debe ser parte de otra clase…
- … pero a veces se necesita un comportamiento particular, pero con una estrecha relación entre clases…

Clases Internas

```
class Externa {
   class Interna{  }
}
```

] javac Externa.java

Se crean los siguiente bytecode

Externa.class

Externa$Interna.class

Aunque son clases distintas, sólo puedes acceder a la clase interna es através de una instancia de la clase externa.

Clases Internas

```
class Externa {
    private int x = 7;
    // definición de la clase Interna
    class Interna {
        public void mirarExterna() {
            System.out.println("Externa x es: " + x);
        }
    } // fin - clase Interna
} // fin - clase Externa
```

Aunque el atributo x (de Externa) es privado, puede ser accedido sin problemas por la clase Interna.

Ya que, la clase Interna es un "miembro más" de la clase Externa

Instanciando una clase interna

```
class Externa {
    private int x = 7;
    public void crearInterna(){
        Interna in = this.new Interna();
        in.mirarExterna();// ejecuta el método
    }
    // definición de la clase Interna
    class Interna {
        public void mirarExterna() {
            System.out.println("Externa x es: " + x);
        }
    } // fin - clase Interna
} // fin - clase Externa
```

Dentro de la clase Externa se puede crear una objeto de la clase Interna (normalmente las clase internas son "auxiliares" y sólo son usadas de manera privada

Para crear una clase interna, se debe tener una instancia de la clase Externa.

¡¡No hay excepción para esta regla!!

Instanciando una clase interna

```
// ahora estamos en otra clase
public static void main(String[ ] args){
    Externa ex = new Externa();
    Externa.Interna in = ex.new Interna();// el ex
```

punto new interna, instancia a la clase interna de Externa, solo con el ex.new interna. Ojo que para que funcione las clases Externa interna no deben ser private.

```
    in.mirarExterna();
}
```

Versión de una línea

Externa.Interna in = new Externa().new Interna();

La referencia **this**

```
class Externa {
    private int x = 7;
    ...
    // definición de la clase Interna
    class Interna {
        public void mirarExterna() {
            System.out.println("Externa x es: " + x); //mira internamente
            System.out.println("Mi Ref: " + this); //mira internamente
            System.out.println("Ref a Externa: " + Externa.this); //mira externamente
        }
    } // fin - clase Interna
} // fin - clase Externa
//Salida
Externa x es: 7
Mi Ref : Externa$Interna@1d3cdaa
Ref a Externa : Externa@6355dc
```

Modificadores para usar

- Modificadores que se pueden usar en las clases Internas
  - final
  - Abstract //no se puede usar final y Abstract a la vez
  - public
  - private
  - protected
  - static (pero pasa a ser una clase anidada)
  - strictfp

Clases Internas (en métodos)

```
class Externa {
    private int x = 7;
    void metodo(){
        class InternaDeMetodo {
            public void mirarExterna() {
                System.out.println("Externa x es: " + x);
            }
        } // fin - clase Interna de método
        InternaDeMetodo in = new InternaDeMetodo();
        in.mirarExterna();
    } // fin - método
} // fin - clase Externa
//Salida
Externa x es: 7
```

La clases internas de método sólo pueden ser instanciadas desde el método en el cual fueron declaradas

Clases Internas (en métodos)

```
class Externa {
      private String x = "externa";
      void metodo(){
            final String y = "metodo";
            class InternaDeMetodo {
                  private String z = "interna";
                  public void mirarExterna() {
                      System.out.println("Externa x es: " + x);
                      System.out.println("Metodo y es: " + y);
                      System.out.println("Interna z es: " + z);
                  }
            } // fin - clase Interna de método
            InternaDeMetodo in = new InternaDeMetodo();
            in.mirarExterna();
      } // fin - método
} // fin - clase Externa
//Salida
Externa x es: externa
Metodo y es: metodo
Interna z es: interna
```

La clases internas de método No pueden usar variables locales de método… (a menos que se marquen con final)

Debido a que las variables de método sólo "viven" mientras el método es ejecutado (stack), y no hay garantías de que siga existiendo después que el método finaliza…

Clases Internas Anónimas

```
class Popcorn {
    public void pop() {
        System.out.println("popcorn");
    }
}
class Food {
    Popcorn p = new Popcorn ( ) { //no termina con un ";" sino que con un {,
        public void pop() { // esto sobreescribe  el método de clase anterior
            System.out.println("popcorn anónimo");
        }
    }; //nótese el punto y coma…
}
```

Lo que en verdad esta sucediendo es lo siguiente:
Se esta extendiendo de la clase **Popcorn** y se esta sobrescribiendo el método **pop()**
Es una subclase "sin nombre" (anónima) de Popcorn

Equivalente

```
class ANONIMA extends Popcorn {
  public void pop() {
    System.out.println("popcorn anónimo");
  }
}
…
Popcorn p = new ANONIMA();
```

Clases Internas Anónimas

```
class Popcorn {
      public void pop() {
            System.out.println("popcorn");
      }
}
class Food {
      Popcorn p = new Popcorn ( ) {
            public void pop() {
                  System.out.println("popcorn anónimo");
            }
            public void corn() {
                  System.out.println("nuevo método");
            }
      }; //nótese el punto y coma…
      void popIt() {
            p.pop();
            p.corn(); // ERROR no
                      //puede acceder
      }
}
```

Cuando la clase anónima extiende la funcionalidad de la clase base (agrega más métodos), estos no son "visibles" por la variable de referencia "**p**"…

Esto sucede ya que el tipo de referencia es **Popcorn** (y sólo sabe hacer "**pop()**"), aunque la instancia anónima define más comportamientos, estos no son "conocidos" por el tipo de dato **Popcorn**

*Recordar: Lo que esta a la izquierda de la variable (el tipo de dato) me dice **"que es lo que puedo hacer"**, y lo que esta a la derecha (la instancia) me dice **"como lo debo hacer"***

Clases Internas Anónimas

```
interface Cocinable {
    public void cocinar();
}

class Food {
    Cocinable c = new Cocinable ( ) {
        public void cocinar() {
            System.out.println("cocinando…");
        }
    }; //nótese el punto y coma…
}
```

Ya qua las interfaces nos dicen que pueden hacer (pero no como hacerlo), al crear una "implementación" anónima de interfaz se deben implementar todos los métodos definidos en ésta

Equivalente

```
class ANONIMA implements Cocinable {
  public void cocinar() {
    System.out.println("cocinando…");
  }
}
…
Cocinable p = new ANONIMA();
```

Clases Internas Anónimas

- Algunas restricciones:
  - Las clases anónimas no tienen constructores propios (ya que no tienen nombre, no puede declarar un constructor)
  - Las implementaciones de interfaz anónima, sólo pueden implementar una interfaz (una clase no-anónima puede implementar varias)

Argumentos Internos Anónimos

```
interface Cocinable {
      public void cocinar();
}
class Food {
      public void hacerPorotos (Cocinable c) { … }
}
class Restorant {
      public static void main (String[ ] args){
            Food comida = new Food();
            cominda.hacerPorotos ( new Cocinable ( ) {
                  public void cocinar() {
                        System.out.println("cocinando…");
                  } // fin de implementación de método : cocinar()
            } ); // fin de clase, paréntesis de método y sentencia ; del new Cocinable";"
      } // fin de método : main( … )
} // fin de clase restorant
```

Nótese que justo al momento de enviar el argumento del método **hacerPorotos(…)** que recibe como parámetro una implementación de la interfaz **Cocinable…**

Esta implementación se hace con la estrategia de clase anónima (se crea una implementación sin nombre), e inmediatamente estregada como parámetro

Clases estáticas Anidadas

```
class Externa {
    private static String nombre = "Externa";
    private Object inaccesiblePorEstaticaExterna;
    static class EstaticaInterna {
        String nombre = "Estatica Interna";
        public String toString ( ) {
            String yo = "soy : " + nombre;
            yo += ", estoy dentro de : ";
            yo += Externa.nombre;
            return yo;
        }
    }
}
```

Pero esto no es una clase interna ????

No… aunque tiene privilegios de acceder a miembros privados, éstos sólo pueden ser estáticos

Y no puede acceder a los miembros de instancia de la clase Externa

Uso en otra clase (debe usar ambos nombre de clase):

```
Externa.EstaticaInterna ex =  new Externa.EstaticaInterna();
System.out.println ( ex ); // soy : Estatica Interna, estoy dentro de : Externa
```