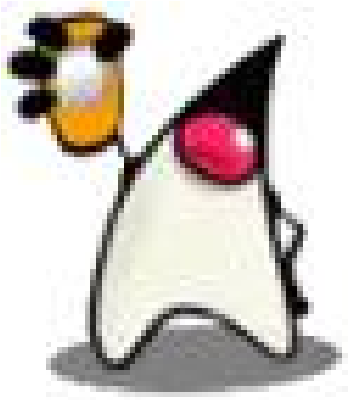
Tema 12: Interfaces y Clases internas

Modificador abstract

- Indica que debe ampliarse un método (sólo está indicado el comportamiento) o que una clase derivada (llamada también subclase) debe proporcionar la implementación.



Clase abstracta

- Con una clase de este tipo se tiene la posibilidad de declarar clases que definen como se utiliza solamente, sin tener que implementar métodos.
- Las subclases de una clase abstracta deben sobreescribir todos los métodos abstractos.
- Las subclases de una clase abstracta deben ser declaradas como clases abstractas.
- Una clase abstracta puede incluir miembros no abstractos.
- No se puede instanciar una clase abstracta pero sí crear referencias de ella.

Clase abstracta

- Si se desea que una clase no sea abstracta entonces basta con que el cuerpo del método(s) abstract sea vacío.

- Ejemplos:

```
public abstract class ClasePadre1{
        public double metodo1(){
            //acciones
       }
        public abstract metodo2();
}
```

```
public class ClasePadre2{
        public double metodo1(){
            //acciones
       }
        public abstract metodo2(){ }
}
```

Método abstracto

- Un método es abstracto si no tiene implementación.

  public **abstract** metodo();

- Todas las clases que heredan de la clase abstracta tiene la “obligación” de implementar el método abstracto (hacerlo concreto).

- No es válido constructores ni métodos static abstract.

Interfaz

- Definición

  Es un tipo de clase que sólo denota comportamiento. La keyword utilizada es *interface*.

- Características:

- Permite simular la herencia múltiple.

- La interfaz sólo es *public o default.*

- Todos sus métodos son *public abstract*.

- Todos los atributos son *public static final*.

Características

- Ejemplo:

```
interface UnEjemplo {
        int CONSTANTE = 100;
        int metodoAbstracto( int parametro );
}
```

- Cuando una clase va a utilizar los miembros de una interfaz se usa la keyword *implements*.

- La interfaz no es instanciable.

Características

- Si una clase no implementa todos los métodos de la interfaz debe ser declarada como abstracta.

```
class  ClaseUno implements UnEjemplo {
    int multiplicando=CONSTANTE;
    int metodoAbstracto( int parametro ){
        return ( parametro * multiplicando );
    }
}
```

- Las constantes definidas en una interfaz pueden usarse en otra clase , uanque no implementen la interfaz, de la siguiente manera: nomInterfaz.CTE;

```
int r= 3.25 * UnEjemplo.COSNTANTE;
```

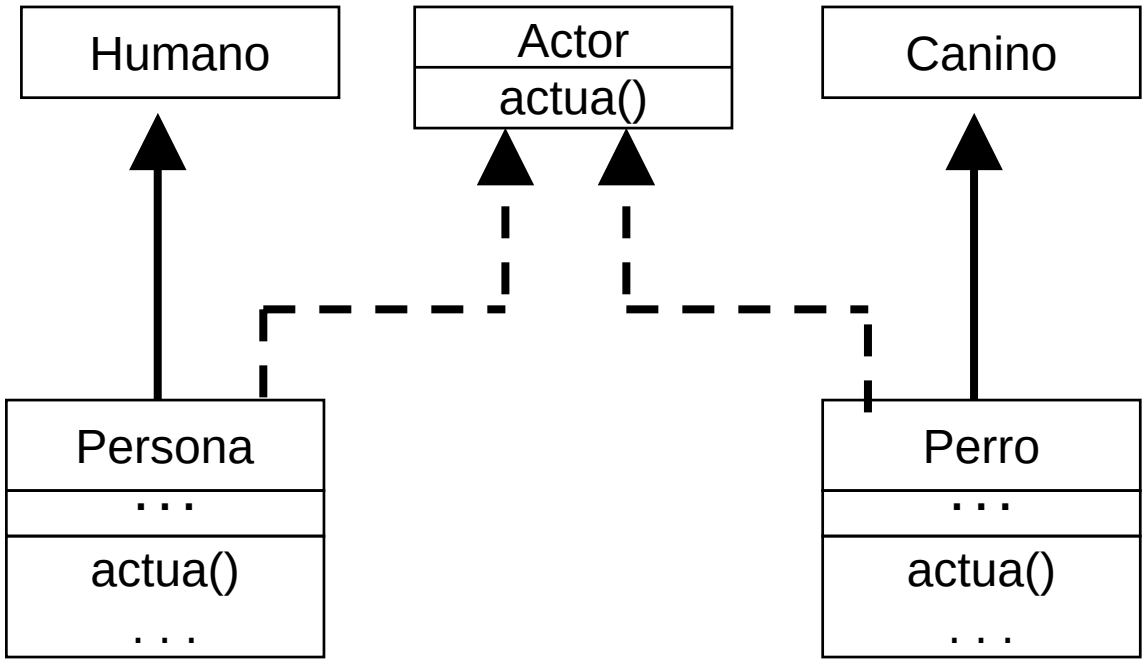
Interfaz

- ¿Cuándo usar una interfaz?

  Cuando se trate de manipular similitudes entre clases no relacionadas sin forzar una relación entre ellas.

  Cuando se sabe que se quiere hacer pero no como (aún o porque lo hará otro usuario).

- Ejemplo:



Interfaz

- Una interfaz puede heredar de muchas clases.

- Una clase puede implementar muchas interfaces

```
public claseUno implements inter1,inter2,inter3{
        //sentencias
    }
```

Ejemplo 1 Interfaz

```
interface Act {
      void act();
    }

    class Actor1 implements Act {
      public void act() {
        System.out.println("To be, or not to be");
      }
    }

    class Actor2 implements Act {
      public void act() {
        System.out.println("Wherefore art thou Romeo?"
    );
      }
    }

    public class TryOut {
      public static void main(String args[]) {
        Actor1 hamlet = new Actor1();
        Actor2 juliet = new Actor2();
        tryout(hamlet);
        tryout(juliet);
      }

      private static void tryout(Act actor) {
        actor.act();
      }
    }
```

Ejemplo 2 Interfaz

```
interface Monster {
      void menace();
    }
interface DangerousMonster extends Monster {
      void destroy();
}

interface Lethal {
      void kill();
}

class DragonZilla implements DangerousMonste{
      public void menace() {    }
      public void destroy() {    }
}

interface Vampire extends DangerousMonster,
                                    Lethal {
      void drinkBlood();
    }
```

Ejemplo 2b Interfaz

```
class VeryBadVampire implements Vampire {
      public void menace() {  }
       public void destroy() {   }
        public void kill() {   }
       public void drinkBlood() {   }
     }

     public class HorrorShow {
        static void u(Monster b) {
            b.menace();
         }

         static void v(DangerousMonster d) {
               d.menace();
               d.destroy();
         }

        static void w(Lethal l) {
               l.kill();
         }
```

Ejemplo 3c Interfaz

```
public static void main(String[] args) {

   DangerousMonster barney = new DragonZilla(
   );
   u(barney);
   v(barney);

  Vampire vlad = new VeryBadVampire();
   u(vlad);
    v(vlad);
    w(vlad);
  }
}
```

Diferencias entre clase, clase abstracta e interfaz

Tipo	**class**	**abstract**	**interface**
Herencias			
Instanciable			
implementa			
atributos			

Clases internas

- Clase interna
  - ✓ De manera general es una clase como cualquier otra. Se declara de la misma forma que las demás pero dentro de otra clase, específicamente dentro del bloque de código especificado por las llaves.
  - ✓ En el código las clases internas proporcionan beneficios de organización, pero también tienen la habilidad de acceder a miembros de las clases externas.
  - ✓ Generalmente son usadas en el modelado de eventos de Java.

Tipos de clases internas

- Al compilar una clase con clases internas se generan varios *.class*

  El nombre tendría la forma:

  ClaseExterna**$**ClaseInterna.class

- Existen 4 tipos:

  - ✓ Clases internas static
  - ✓ Clases internas miembro
  - ✓ Clases internas locales
  - ✓ Clases anónimas

Clases internas static

- Sólo pueden ser creadas dentro del bloque de definición de la clase externa o dentro de una interfaz.
- La invocación es ClasExterna.Claseinterna
- Pueden usar los miembros static de la clase externa.
- No se necesitan objetos de la clase externa para poder crear objetos de la clase interna static.
- La keyword import puede usarse para importar una clase interna static.

  Ejemplo:

```
... implements List.Linkeable{ ...}

import List.*; // o
import List.Linkeable;
```

Clases internas miembro

- Son llamadas nested class o simplemente clases internas.
- No se pueden llamar igual que la clase externa.
- No pueden contener miembros de tipo static.
- Cada objeto de la clase interna existe siempre dentro de un objeto de la clase externa implícitamente.
- Las clases internas pueden ser private o protected además de public y default.
- Los métodos de la clase interna acceden directamente a todos los miembros de la clase externa y viceversa.
- Una clase miembro puede contener otra clase.

Clases internas miembro

- En la clase interna la palabra this se refiere al objeto de la propia clase, para accesar al objeto de la clase externa se usa la siguiente sintaxis:

  ClaseExterna.this.var

- Ejemplo:

  C es una clase interna de B, que a su vez es una clase interna de A.

```
                                  // se crea un objeto
A a=new A();                      // de la clase A
A.B b= a.new B();                 // de B dentro del objeto a
A.B.C c= b.new C(); // de C dentro del objeto B
```

Clases internas locales

- Son declaradas dentro de un método o en un bloque estático.
- Su comportamiento es análogo a las variables locales. Sólo son visibles dentro del bloque de código donde fueron definidas.
- Tienen acceso a todas las variables miembro y métodos de la clase externa así como a los miembros heredados.
- Pueden usar las variables locales y argumentos de métodos visibles en ese bloque de código sólo si son final.
- La keyword **this** se usa igual que en las clases internas miembro pero no **new** ni super.
- No pueden definir variables, métodos o clases static.
- No usan ningún modificador de acceso.

Clases internas anónimas

- No llevan nombre.
- La definición de la clase y la creación se hacen en un mismo paso.
- No definen constructores pero si bloques estáticos.
- Formas de definición:
  - ✓ new seguida de la definición de la clase entre llaves {...}
  - ✓ new seguida del nombre de la clase de la que hereda (no extends) y la definición de la clase anónima entre llaves.
  - ✓ new seguida del nombre la interface que implementa (no implements) y la definición de la clase anónima entre llaves
- El compilador produce archivos de la forma: Claseexterna$1.class
- Ejemplo (de awt)

```
unObjeto.addACtionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) { ...}
} );
```

Ejemplo 1

```
public class Robot {
    private int x;
    private DescripcionRobot descripcion;
    public Robot (int x) {
        this.x=x;
        this.descripcion=new DescripcionRobot();
    }
    // clase interna
    public class DescripcionRobot {
        private String nombre, color;
        public DescripcionRobot() {
            if (x>0) {
                this.nombre="A";
                this.color="blanco";
            } else {
                this.nombre=”B";
                this.color=”rojo";
            }
        }
    }
```

...Ejemplo 1

```
    public void avanzar (int d) {
        this.x+=d;
    }
    public int valorX () {
        return this.x;
    }
    public void asignarX (int x) {
        this.x=x;
    }
    public String toString() {
        return "Robot(" + this.x + ")";
    }
}
```

- La clase anidada puede ser “static” (clase Robot.DescripcionRobot)
- Las clase anidada tiene acceso a los miembros de la clase que la contiene
- Las instancias se crean, por ejemplo:
  Robot.DescripcionRobot r =
  (new Robot(3)).new DescripcionRobot()

Ejemplo 2

```
public class Parcel1 {

  class Contents {
    private int i = 11;
   public int value() {
      return i;
    }
  }

  class Destination {
    private String label;
    Destination(String whereTo) {
      label = whereTo;
    }

    String readLabel() {
      return label;
    }
  }

  public void ship(String dest) {
    Contents c = new Contents();
    Destination d = new Destination(dest);
    System.out.println(d.readLabel());
  }

  public static void main(String[] args) {
    Parcel1 p = new Parcel1();
    p.ship("Tanzania");
  }
}
```

Ejemplo 3

```
public class Parcel2 {    // Retorno de una referencia
   class Contents {       // a una clase interna
      private int i = 11;
      public int value() { return i;  }
   }

   class Destination {
     private String label;
     Destination(String whereTo) {  label = whereTo; }
    String readLabel() {
      return label; }    }

   public Destinationto(String s) {
     return new Destination(s);    }

   public Contents cont() {
     return new Contents(); }

   public void ship(String dest) {
     Contents c = cont();
     Destination d = to(dest);
     System.out.println(d.readLabel() ) ;   }

   public static void main(String[] args) {
     Parcel2 p = new Parcel2();
     p.ship("Tanzania");
     Parcel2 q = new Parcel2();
     // Definiendo una referencia a una clase interna
     Parcel2.Contents c = q.cont();
     Parcel2.Destination d = q.to("Borneo");  }
}
```

Ejemplo 4

```
//Herencia de una clase interna
class WithInner {
      class Inner {
      }
}

public class InheritInner extends WithInner.Inner
{
  //! InheritInner() {} // No se puede compilar
  InheritInner(WithInner wi) {
    wi.super();
  }

  public static void main(String[] args) {
    WithInner wi = new WithInner();
    InheritInner ii = new InheritInner(wi);
  }
}
```

Ejemplo 5

```
//Creación de un constructor para una clase anónima

abstract class Base {
      public Base(int i) {
        System.out.println("Base constructor, i = " + i);
      }

      public abstract void f();
    }

    public class AnonymousConstructor {

      public static Base getBase(int i) {
        return new Base(i) {
          {
            System.out.println(“Dentro de la inicialización de
    instancia");
          }

          public void f() {
            System.out.println(“En metodo f() anonimo");
          }
        };
      }

      public static void main(String[] args) {
        Base base = getBase(47);
        base.f();
      }
    }
```

Ejemplo 6

```
 //Una clase anónima
//Devuelve un tipo enumerado sobre la pila mediante
    una clase anónima.

public Enumeration elementos( ){
       return new Enumeration( ){
       int cont= cima;
       public boolean  hashMoreElements(){
          return cont >0;
     }
     public Object nextElement(){
         if (cont ==0)
             throw new NoSuchElementException("Pila
    en array");
        return array[--cont] ;
      }
   };
 }
```