

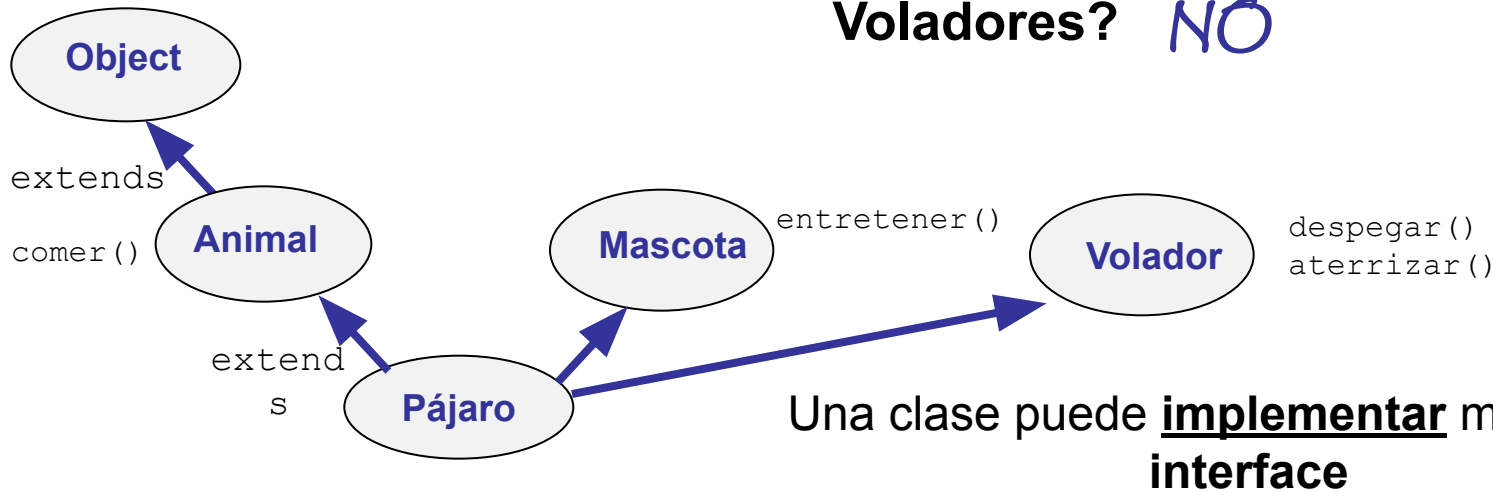
Interfaces en JAVA

- ¿Qué son las interfaces?, ¿Para qué sirven?
- Sintaxis de la declaración de una interfaces
- Ejemplo: la interface Volador
 - Definición de la interface
 - Implementación de la interface: clases que la implementan
 - TODO JUNTO: clases e interfaces
- ¿Cómo ordenamos arreglos de Strings?
 - El método sort() de Arrays
- ¿Cómo ordenamos arreglos de **Personas**?
- La interface **Comparable**
 - Ejemplo: ordenamos arreglos de **Personas**

Interfaces

¿Qué son?, ¿Para qué sirven?

¿La clase Pájaro podría extender la clase Animal, Mascota y Voladores? **NO**



- Una interface en Java es una colección de definiciones de métodos **abstractos** y de **declaraciones de constantes de clase**, agrupadas bajo un nombre.
- Las interfaces no tienen implementación, sólo **definen un protocolo** y no se pueden crear instancias a partir de ellas.
- Las **interfaces complementan las clases**: las clases implementan interfaces.
- Una interface establece **qué** debe hacer la clase que la implementa, sin especificar el **cómo**.
- Las interfaces **no imponen una relación de herencia** entre las clases que las implementan,

Sintaxis de la declaración de Interfaces

lista de nombres de interfaces



```
package paquete1;  
public interface Inter extends Inter1, Inter2, Inter3 {  
  
    Declaración de métodos: implícitamente public y abstract  
    Declaración de constantes de clase: implícitamente public, static y  
    final  
}
```

- El especificador de acceso **public** establece que la interface puede ser usada por cualquier clase o interface de cualquier paquete, **sería parte de la API pública**. Si **se omite el especificador de acceso**, la interface solamente podría ser usada por las clases e interfaces contenidas en el mismo paquete que la interface, **formaría parte de la implementación**.
- Una interface puede extender múltiples interfaces. Hay herencia múltiple de interfaces.
- Una interface hereda todas las constantes y métodos de sus **superInterfaces**: **NO SE HEREDA CÓDIGO, SOLAMENTE FIRMAS DE MÉTODOS**.

Ejemplo: Definición de la interface Volador

```
package clase5;

public interface Volador {
    long UN_SEGUNDO=1000;
    long UN_MINUTO=60000;
    String despegar();
    String aterrizar();
}
```

- La interface **Volador** establece **qué** hacen los objetos de Voladores, no indica **cómo (NO HAY CÓDIGO)**.
- Las clases que implementen **Volador** establecen el **cómo**: deben implementar los métodos **despegar()** y **aterrizar()**, todos públicos y podrán usar las constantes **UN_SEGUNDO** y **UN_MINUTO**.
- Las variables son implícitamente **public static** y **final** (constantes de clase). Los métodos de una interface son implícitamente **public** y **abstract**.
- El código fuente de las interfaces de igual manera que el de las clases se guarda en archivos con el mismo nombre de la interface y con extensión **.java** y se compilan a **.class**

Ejemplo: Implementación de la interface Volador

Para indicar que una clase implementa un interface se usa la palabra clave **implements**.

```
package clase4;
public class Pajaro implements Volador {
    //código de la clase
    // DEBEN ESTAR IMPLEMENTADOS TODOS
    //LOS MÉTODOS DE VOLADOR
}
```

```
package clase4;
public interface Volador {
    long UN_SEGUNDO=1000;
    long UN_MINUTO=60000;
    String despegar();
    String aterrizar();
}
```

- Una clase que implementa una interface, hereda las constantes y **debe implementar cada uno de los métodos de la interface**.
- Una clase puede implementar más de una interface y de esta manera los objetos de dicha clase son de muchos tipos, eso podría “relacionarse” como herencia múltiple, sin embrago no lo es.

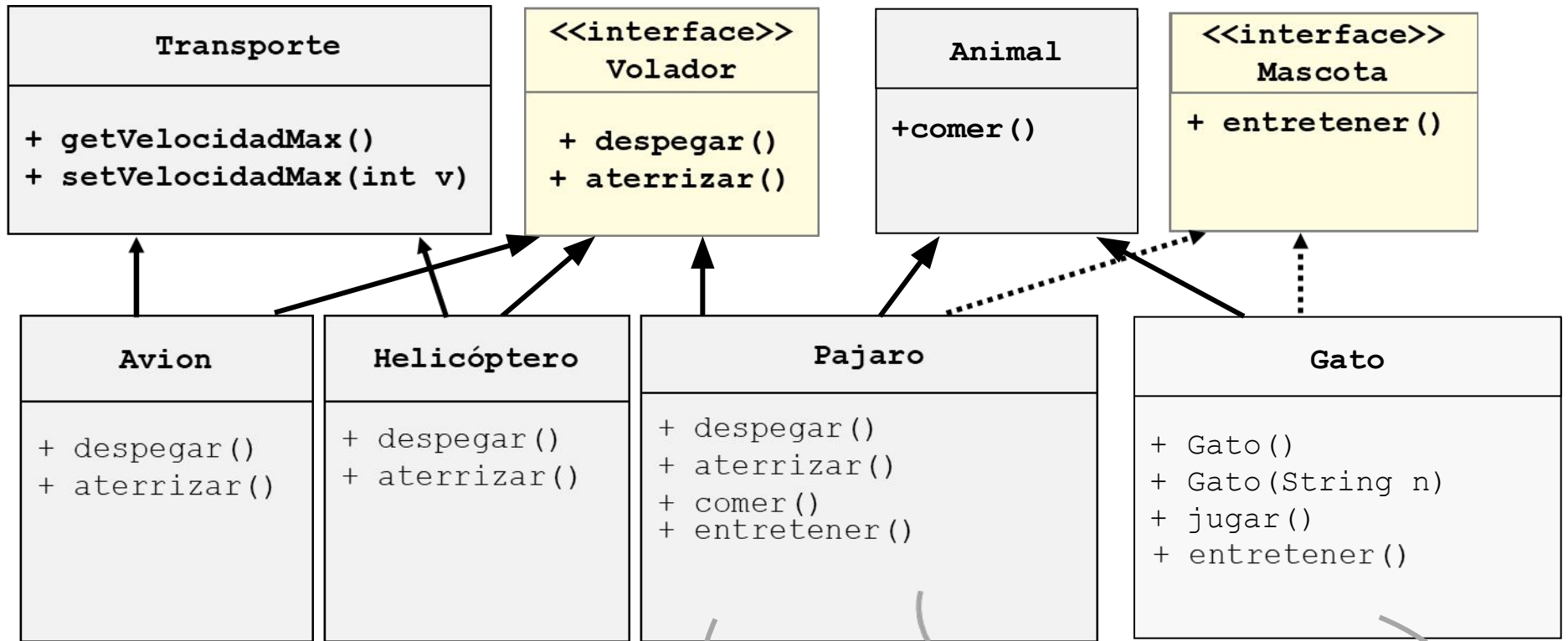
```
package clase4;
public class Pajaro extends Animal implements
    public String despegar(){//código del método}
    public String aterrizar(){//código del método}
    public String entretener(){//código del método}
}
```

```
package clase4;
public interface Mascota{
    String entretener();
}
```

Clases e Interfaces: TODO JUNTO

La interface **Volador** describe las **cosas que vuelan** y la interface **Mascota** las **cosas que entretienen y acompañan**.

Múltiples clases las implementan y pertenecen a jerarquías de herencia diferente.



Un objeto **Avión** y un objeto **Helicóptero** son medios de **Transporte** y también son objetos **Voladores**.

Un objeto **Pájaro** es un **Animal**, es un objeto **Volador** y una **Mascota**.

Una clase puede implementar más de una interface

Un objeto **Gato** es un **Animal** y es una **Mascota**

Implementación de Interfaces

Cuando una clase implementa una interface se *establece un contrato* entre la interface y la clase que la implementa: **la clase DEBE implementar TODOS los métodos de la interface.**

El compilador chequea que la clase implemente todos los métodos de la interface.

```
public class Pajaro extends Animal
    implements Mascota, Volador {
    private String nombre;

    // Métodos de Pajaro
    public void setNombre(String nom){
        nombre=nom;
    }
    public String getNombre(){
        return nombre;
    }
    // Método de la interface Mascota
    public String entretener(){
        return "Método entretener() de:";
    }
    // Métodos de la interface Volador
    public String despegar(){
        return("Elevar alas");
    }
    public String aterrizar(){
        return("Bajar alas");
    }
}
```

```
public interface Volador {
    long UN_SEGUNDO=1000;
    long UN_MINUTO=60000;
    String despegar();
    String aterrizar();
}
```

```
public class Avion extends Transporte
    implements Volador{
    int velocidadMax;

    // Métodos de la interface Volador
    public String despegar(){
        return("Elevar y guardar ruedas");
    }
    public String aterrizar(){
        return("Desplegar ruedas");
    }
}
```

¿Cómo ordenamos un arreglo de Personas?

```
public class Test {  
    public static void main(String[] args) {  
        Persona personas[] = new Persona[4];  
        personas[0]= new Persona("Paula","Gomez",16);  
        personas[1]= new Persona("Ana","Rios",6);  
        personas[2]= new  
        Persona("Maria","Ferrer",55);  
        personas[3]= new Persona("Juana","Araoz",54);  
  
        for (int i=0; i<4;i++){  
            System.out.println(i+":personas[i]);  
        }  
        Arrays.sort(personas);  
        for (int i = 0; i<4; i++) {  
            System.out.println(i+": "+personas[i]);  
        }  
    }  
}
```

Error en ejecución!!

```
public class Persona {  
    private String nombre;  
    private String apellido;  
    private int edad;  
    public Persona  
        (String a,String n,int e){  
        nombre=n;  
        apellido=a;  
        edad=e;  
    }  
    public String toString(){  
        return apellido+", "+nombre;  
    }  
}
```

Al invocar al método **sort()** de **Arrays** y pasarle un arreglo de **Personas**, se genere un error en ejecución porque los objetos **Persona** no son comparables. No está definido ningún algoritmo de comparación entre objetos de tipo **Persona**.

La interface `java.lang.Comparable`

La interface genérica **Comparable** de JAVA tiene un único método que compara 2 objetos de tipo T y devuelve un valor entero :

```
public interface Comparable<T> {  
    public int compareTo(T o);  
}
```

Este método retorna:

- =0: si el objeto receptor es igual al pasado en el argumento.
- >0: si el objeto receptor es mayor que el pasado como parámetro.
- <0: si el objeto receptor es menor que el pasado como parámetro.

Si una clase implementa la interface **java.lang.Comparable** sus instancias son **comparables**.

Es posible ordenar objetos **comparables** porque se cuenta con un algoritmo de comparación, el método **compareTo()**.

El método **sort()** de **Arrays** ordena arreglos de objetos que implementen **Comparable**. El método **sort()** internamente **usa** el método **compareTo()** para ordenar los elementos del arreglo.

El método **compareTo()** permite comparar el objeto receptor con el pasado como argumento.

¿Cómo ordenamos arreglos de Personas?

La clase Persona implementa la interface Comparable

```
public class Test {  
    public static void main(String[] args){  
        Persona personas[] = new Persona[3];  
        personas[0]= new Persona("Paula","Gomez",16);  
        personas[1]= new Persona("Ana","Rios",6);  
        personas[2]= new  
Persona("Maria","Ferrer",55);  
        personas[3]= new Persona("Juana","Araoz",54);  
        for (int i=0; i<4;i++){  
            System.out.println(i+": "+personas[i]);  
        }  
        Arrays.sort(personas);  
        for (int i = 0; i<4; i++) {  
            System.out.println(i+": "+personas[i]);  
        }  
    }  
}
```

Al invocar al método **sort()**, ahora sí los puede ordenar!!, con el criterio establecido en el **compareTo()**

0:Gomez, Paula:16
1:Rios, Ana:6
2:Ferrer, Maria:55
3:Araoz, Juana:54

0:Rios, Ana:6
1:Gomez, Paula:16
2:Araoz, Juana:54
3:Ferrer, Maria:55

```
import java.util.*;  
  
public class Persona  
    implements Comparable<Persona> {  
    private String nombre;  
    private String apellido;  
    private int edad;  
    public Persona(String n,String a,  
int e){  
        nombre=n;  
        apellido=a;  
        edad=e;  
    }  
    public String toString(){  
        return apellido+", "+nombre;  
    }  
    public int compareTo(Persona o){  
        return this.getEdad() - o.getEdad();  
    }  
}
```

El orden que se define con la interface Comparable se llama "orden natural"

¿Dónde usamos la interface Comparable?

Siempre que usemos estructuras de datos **ordenadas**, los objetos que se guardan en ellas deben implementar la interface **Comparable**.

Los objetos que se guardan en **árboles binarios de búsqueda** y en las **heaps** **DEBEN** implementar la interface **Comparable** porque en los algoritmos de estas estructuras de datos se requiere de una manera de comparación predeterminada que garantice el orden.

```
public class MaxHeap <T extends Comparable<T>>
```

```
public class ArbolBinarioDeBusqueda <T extends Comparable<T>>
```

Esta cláusula dice: todos los elementos que se guardan en esta estructura de datos **IMPLEMENTAN** la interface **COMPARABLE**. Aquí el “extends” es sinónimo de “implements”.