3.5 Interfaces

Análisis y Diseño de Software
2º Ingeniería Informática
Universidad Autónoma de Madrid

Interfaces: ¿Qué son?

- Los métodos de una clase son su interfaz con el resto de clases.
- Una interfaz es un grupo de métodos relacionados (solo las cabeceras), que permiten modelar el comportamiento de un tipo de objeto.
- En Java una interfaz es una colección de definiciones de métodos (sin implementación) y de constantes.
- Interfaces vacías: Uso como etiquetas semánticas, ej: Serializable, Cloneable.

¿Por qué son necesarias?

- Una interfaz define un tipo. Es similar a una clase con sólo métodos públicos abstractos, y constantes públicas (variables públicas, finales y estáticas)
- Diferencia: Herencia múltiple
- Java no admite herencia múltiple entre clases
- Una interfaz puede heredar de una o varias interfaces, pero no puede heredar de una clase
- Una clase sólo puede heredar de una clase, pero puede además *implementar* cualquier número de interfaces

Ejemplo

```
public interface Trabajo{
  void ejecutarTrabajo();
}

public class RealizarBackup
       implements Trabajo
{
  Database d;
  public void ejecutarTrabajo(){
    // Código de realizar backup
  }
}
```

```
public class ColaTrabajo
       implements Trabajo
{
  List<Trabajo> pendientes;
  public void addTrabajo(Trabajo t){
    pendientes.add(t);
  }
  public void ejecutarTrabajo(){
    for (Trabajo t:pendientes){
      t.ejecutarTrabajo();
    }
    pendientes.clear();
  }
}
```

Ejemplo

"implements" declara que la clase implementa una o varias interfaces

```
public interface Trabajo{
  void ejecutarTrabajo();
}

public class RealizarBackup
      implements Trabajo
{
  Database d;
  public void ejecutarTrabajo(){
    // Código de realizar backup
  }
}
```

```
public class colaTrabajo
      implements Trabajo
{
  List<Trabajo> pendientes;
  public void addTrabajo(Trabajo t){
    pendientes.add(t);
  }
  public void ejecutarTrabajo(){
    for (Trabajo t:pendientes){
      t.ejecutarTrabajo();
    }
    pendientes.clear();
  }
}
```

Ejemplo

```
public interface Trabajo{
  void ejecutarTrabajo();
}

public class RealizarBackup
       implements Trabajo
{
  Database d;
  public void ejecutarTrabajo(){
    // Código de realizar backup
  }
}
```

```
public class colaTrabajo
       implements Trabajo
{
  List<Trabajo> pendientes;
  public void addTrabajo(Trabajo t){
    pendientes.add(t);
  }
  public void ejecutarTrabajo(){
    for (Trabajo t:pendientes){
      t.ejecutarTrabajo();
    }
    pendientes.clear();
  }
}
```

Ambas clases deben implementar los métodos definidos en la interfaz

Ejemplo

```
public interface Trabajo{
  void ejecutarTrabajo();
}

public class RealizarBackup
       implements Trabajo
{
  Database d;
  public void ejecutarTrabajo(){
    // Código de realizar backup
  }
}
```

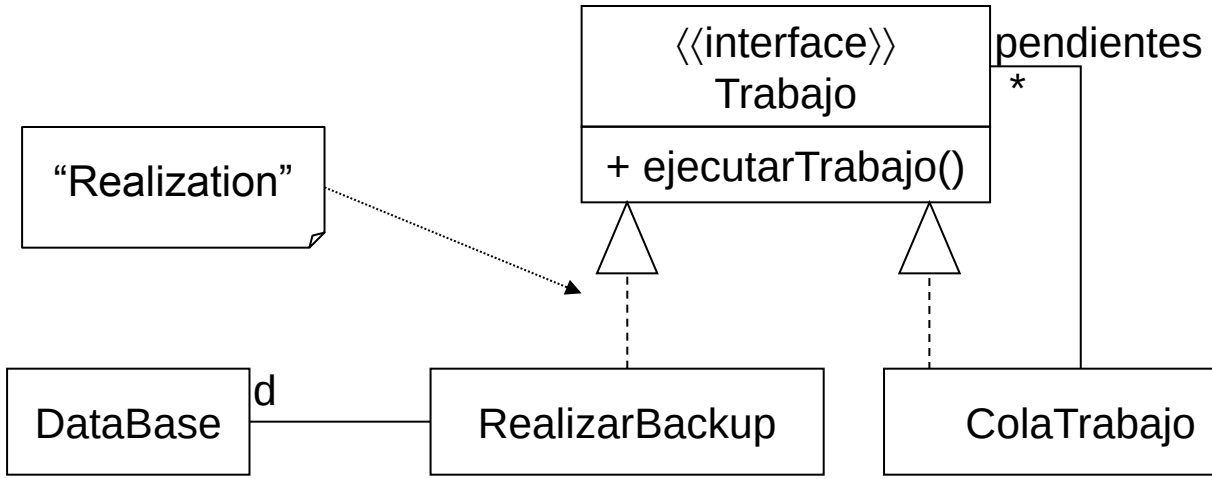
```
public class colaTrabajo
       implements Trabajo
{
  List<Trabajo> pendientes;
  public void addTrabajo(Trabajo t){
    pendientes.add(t);
  }
  public void ejecutarTrabajo(){
    for (Trabajo t:pendientes){
      t.ejecutarTrabajo();
    }
    pendientes.clear();
  }
}
```

Trabajo es un tipo. Cualquier objeto de una clase que implementa esta interfaz se considera una instancia de dicho tipo.

Definición de interfaces: Sintaxis

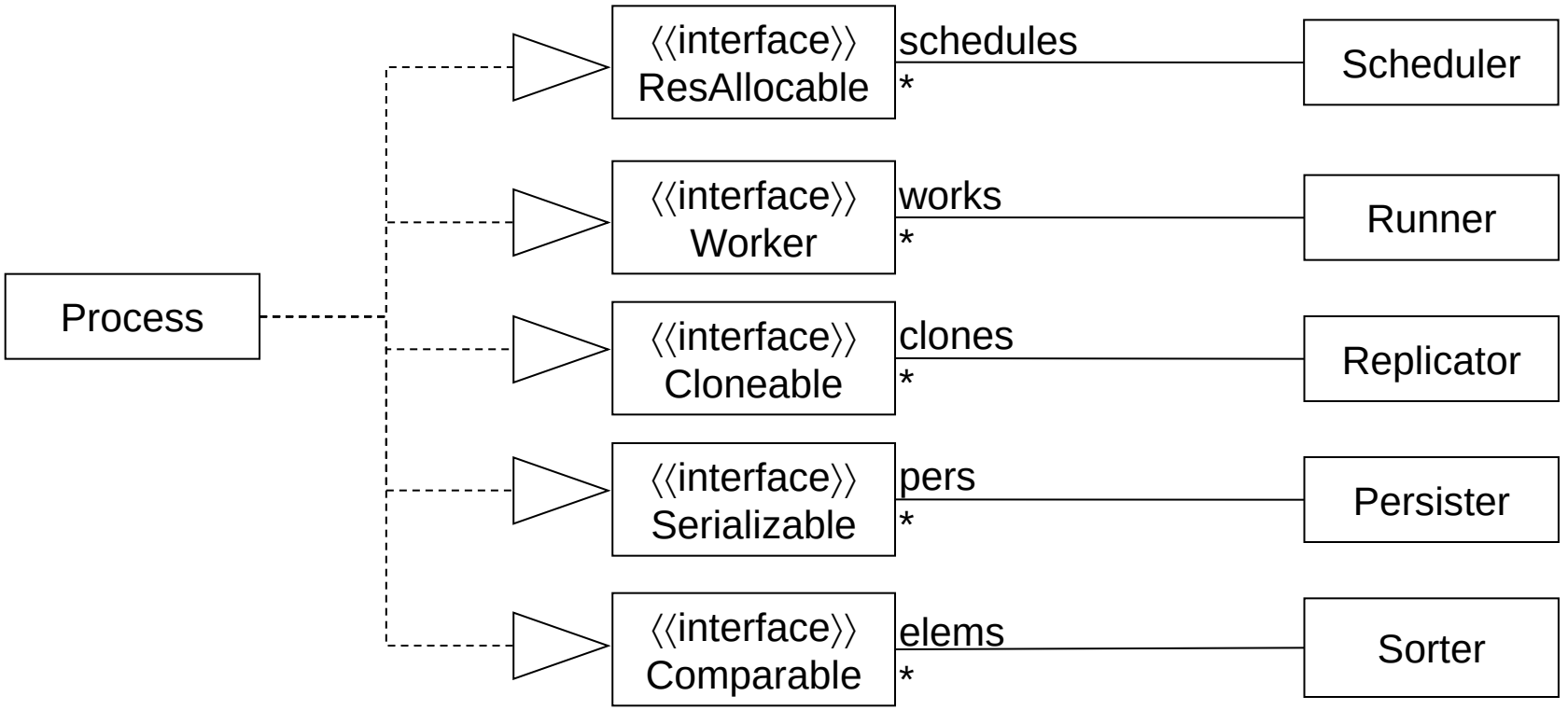
- `[public | ...] interface <nombre-de-interfaz> [extends <interface1>,<interface2> ... ]`
- Una interface puede heredar de otras interfaces.
- Las variables de la interface:
  - `[public static final] <tipo> <nombre-variable>=<valor>;`
  - Son variables “estáticas y finales” (constantes), implicitamente
  - Los métodos se definen sin código (igual que los métodos abstractos) y cualquier clase que implemente la interfaz los debe definir. Se admite public y abstract, pero son opcionales, ya que son modificadores implícitos.

Definición de Interfaces en UML



Utilidad

- Permite definir conjuntos de funcionalidad – interfaces – para poder acceder a clases heterogéneas de manera uniforme.
- Permite ver una clase desde distintas perspectivas.



Ejemplo

Interfaz Comparable

```
public interface Comparable{
  int compareTo(Object o)
}

public class Coche
implements Comparable
{
  int cilindrada;
  String marca;
  String modelo;
  //…
  int compareTo (Object o){
    Coche c= (Coche) o;
    return cilindrada-c.cilindrada;
  }
}
List listaDeCoches;
// …
Collections.sort(listaDeCoches.sort);
```

- Definida en `java.lang`.
- Da el “orden natural” de una clase.
- Las clases que la implementan se pueden ordenar, por ejemplo mediante: Collections.sort y Arrays.sort.

Problema: Dos clases que implementan esta interfaz pueden no ser comparables directamente

Ejemplo: Interfaces genéricas

Interfaz Comparable<T>

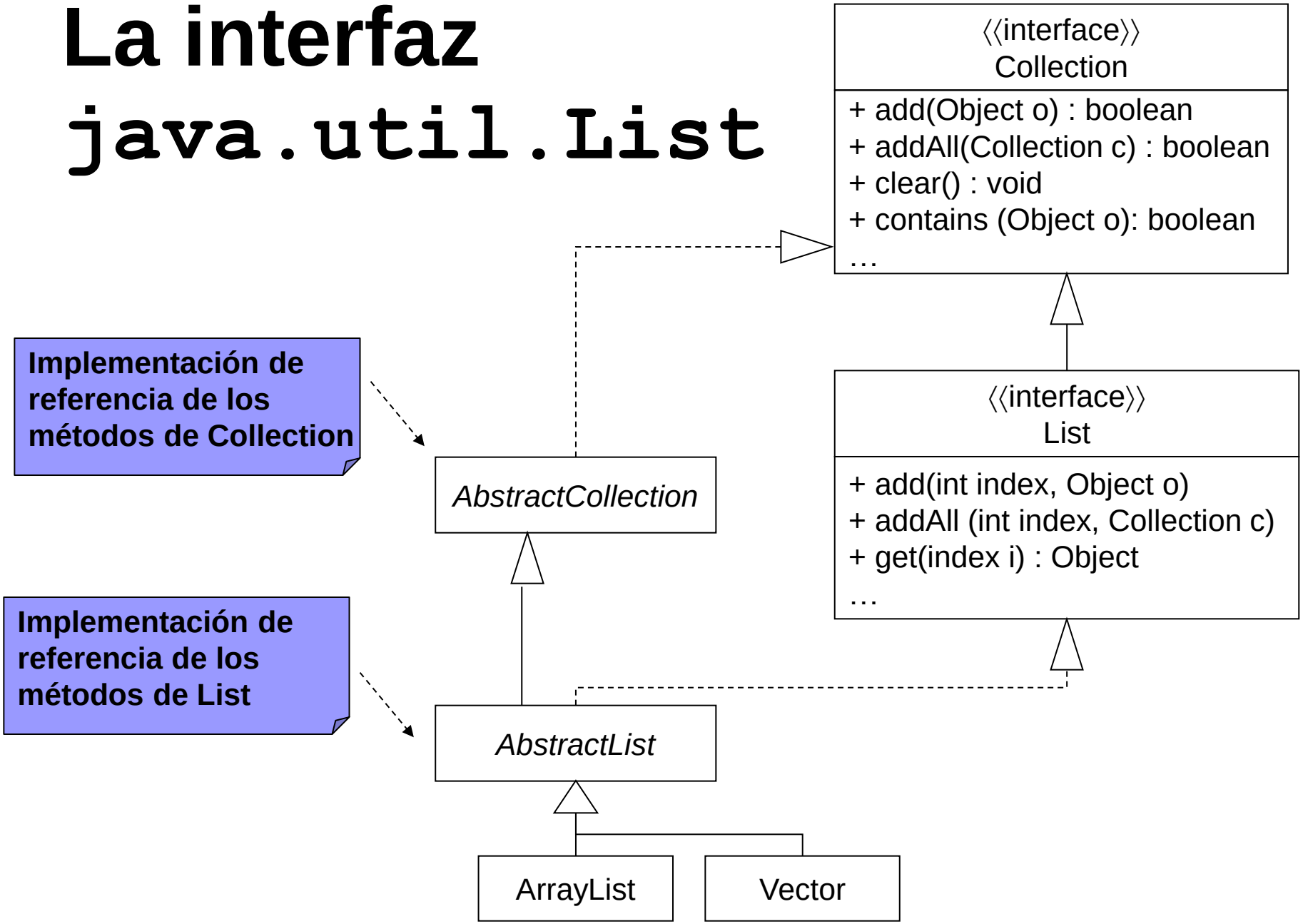
```
public interface Comparable<T>{
  int compareTo(T o)
}

public class Coche
implements Comparable<Coche>
{
  int cilindrada;
  String marca;
  String modelo;
  // …
  int compareTo (Coche c){
    return cilindrada-c.cilindrada;
  }
}
List<Coche> lista;
…
lista.sort();
```

- Interfaz Comparable genérica desde Java 5.0
- Evita los problemas de seguridad de tipos en la comparación.

Ejemplos de Interfaces de la librería estandar

- *Cloneable*: Indica al método Object.clone() que se puede copiar campo a campo.
- *Comparable<T>*: Objetos comparables, orden natural
- *Comparator<T>*: Función de comparación
  - int compare(T o1, T o2))
- *Serializable*: El objeto se puede serializar (copia binaria a disco o enviado por red)
- TADs: Collection, Iterable, Queue, Deque, List, Map, Set, TreeModel
- DOM: Document, Node, Element, Attr

La interfaz `java.util.List`



Modificación de interfaces

- Una vez creada una interfaz y las clases que la implementan, al añadir o modificar un método de una interfaz es necesario cambiar todas las clases que lo implementan.

- ¿Soluciones?
  - Crear una nueva interfaz, que hereda de la original y contiene los nuevos métodos
  - Dar una *implementación de referencia* en una clase (problema: en Java no tenemos herencia múltiple de clases)
  - Dar una implementación de referencia usando métodos estáticos (Previsto para el JDK 8)

Ejemplo de modificación

```
public interface Trabajo{
  void ejecutarTrabajo();
}

public interface TrabajoTransaccional extends Trabajo{
  void ejecutarEnTransaccion(Transaccion t);
}
```

- Las nueva interfaz ofrece una mayor funcionalidad, y no requiere modificar las clases que implementan la versión reducida

- **Problema**: Las clases existentes no cumplen la nueva interfaz, aunque los nuevos métodos se puedan implementar usando los antiguos

Una implementación de referencia

```
public interface Arbol{
  Object getElemento();
  Arbol hijoIzq();
  Arbol hijoDer();
  boolean esHoja();
  boolean esVacio();
  Object search(Object o);
}
```

Subclases de ArbolAbstracto sobreescribirán algunos métodos, algunos también por razones de eficiencia.


```
public abstract class ArbolAbstracto
implements Arbol
{
  public boolean esHoja(){
    return hijoIzq().
           esVacio()&&
           hijoDer().
           esVacio(); }
  public Object search(Object o){
  //implementa búsqueda binaria…
  }
  // Podemos implementar los otros
  // métodos simplemente lanzando
  // una excepción.
}
```

Usando una implementación de referencia

- Sirve para simplificar la implementación de interfaces con numerosos métodos, donde algunos pueden apoyarse en otros para su implementación.

- Ejemplo en el JDK: `java.util.AbstractList`
  - Para una lista de solo lectura, sólo es necesario implementar *get(int)* y *size()*
  - Proporciona: *iterator*, *equals*, *indexOf(Object)*, *lastIndexOf(Object)*, *subList()*, etc.

Herencia múltiple con interfaces

```
class C extends A, B{
}
```

- No se admite en Java, usamos interfaces en su lugar:

```
interface A{
}
interface B{
}
interface C extends A, B{
}
class AImpl implements A{ }
class BImpl implements B{ }
```

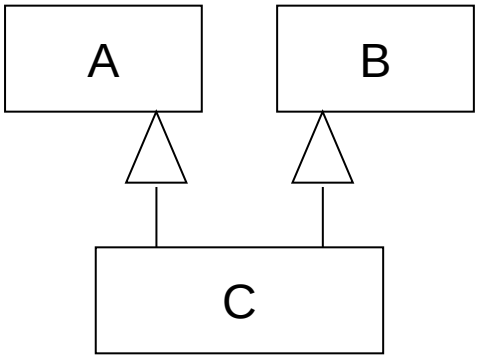
- Heredamos de la clase más compleja y usamos delegación en la otra:

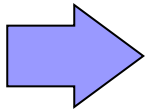
```
class CImpl extends AImpl
            implements C
{
  B b= new BImpl();
}
```

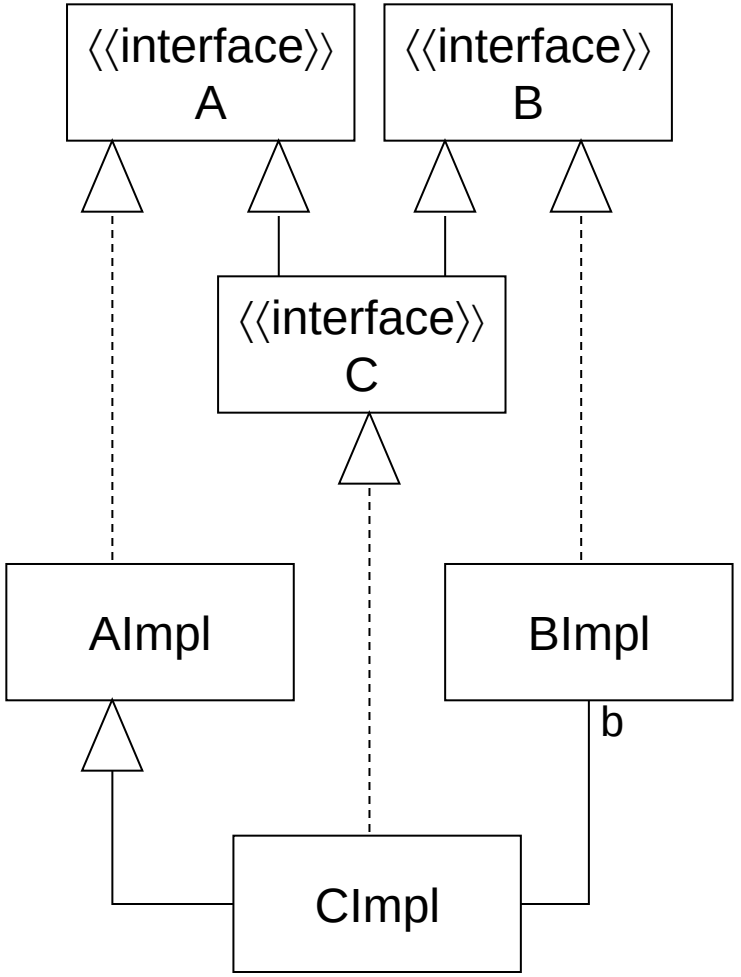
- Delegamos en «b» los métodos propios de B mientras que los de A se heredan, por ejemplo:

```
  public int bIntMethod() {
      return b.bIntMethod();
   }
```

Herencia múltiple con interfaces







Válido en UML, y en lenguajes como C++, pero no en Java

Resumen

- Una interfaz define un protocolo de comunicación entre objetos.
- La interfaz contiene declaraciones, sin implementación, de métodos y constantes.
- Una clase que implementa una interfaz ha de implementar todos los métodos de la interfaz (si no tendría que ser abstracta)
- Una interfaz define un tipo: su nombre se puede utilizar en cualquier sitio donde se pueda utilizar un tipo