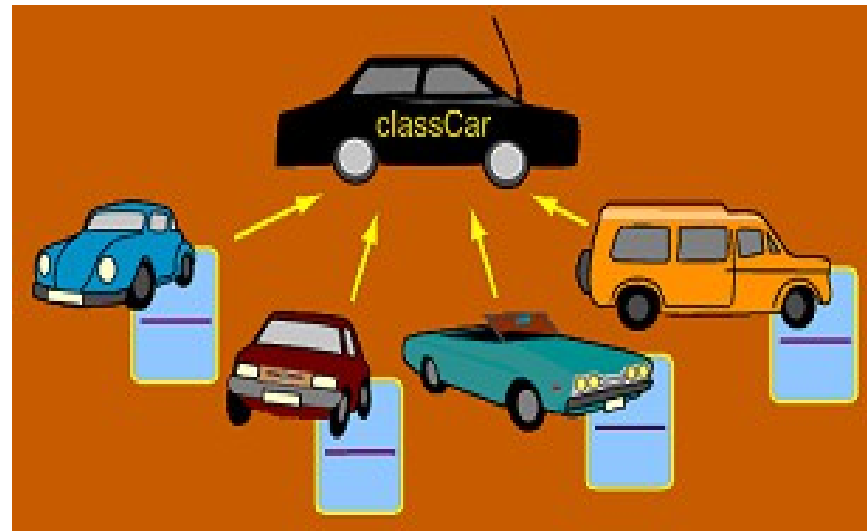


# Tema 3



**Reutilización y polimorfismo**

# Contenidos



Lección	Título	Nº sesiones (horas)
3.1	Mecanismos de reutilización de código	3
3.2	Representación en UML de los mecanismos de reutilización	1
3.3	El polimorfismo	2

[http://groups.diigo.com/group/pdoo\\_ugr](http://groups.diigo.com/group/pdoo_ugr)



# **Lección 3.3**

## **El polimorfismo**

# Objetivos de aprendizaje



- Aprender el concepto de polimorfismo.
- Conocer la regla de compatibilidad de tipos para el polimorfismo.
- Comprender como a través del polimorfismo, la ejecución de un mismo mensaje será muy diferente dependiendo del objeto receptor.
- Relacionar el polimorfismo con la ligadura dinámica de métodos y la redefinición de métodos heredados.
- Analizar el polimorfismo en los lenguajes con definición estática de tipo (Java).
- Analizar el polimorfismo en los lenguajes sin definición estática de tipo (Ruby).

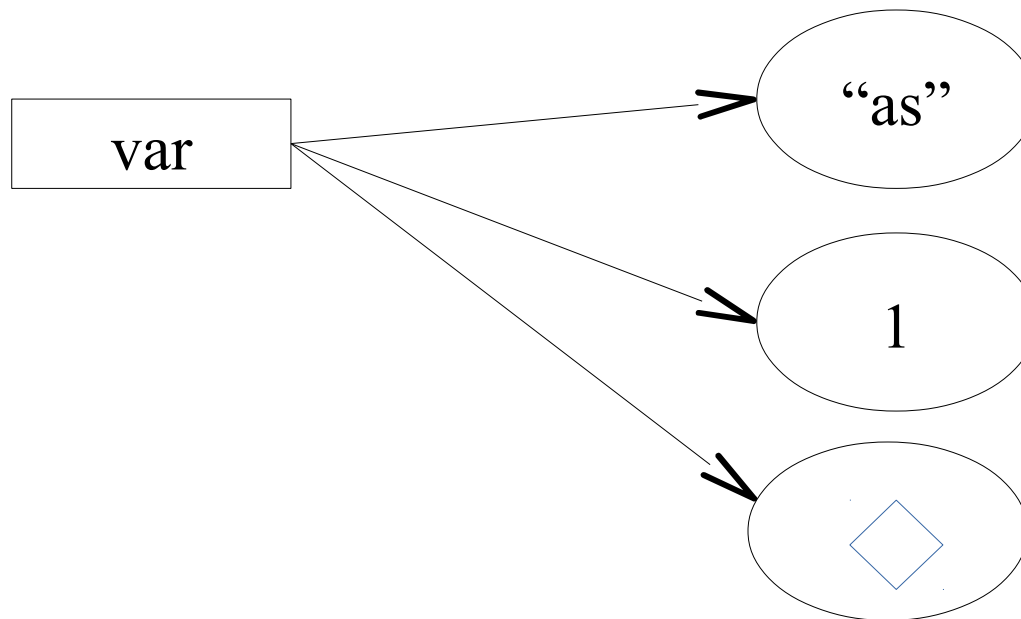
# Contenidos



1. Definición de polimorfismo.
2. Tipo estático y dinámico.
3. Polimorfismo y ligadura dinámica.
4. Polimorfismo y lenguajes con tipo estático.
5. Polimorfismo y lenguajes sin tipo estático.
6. No es polimorfismo.

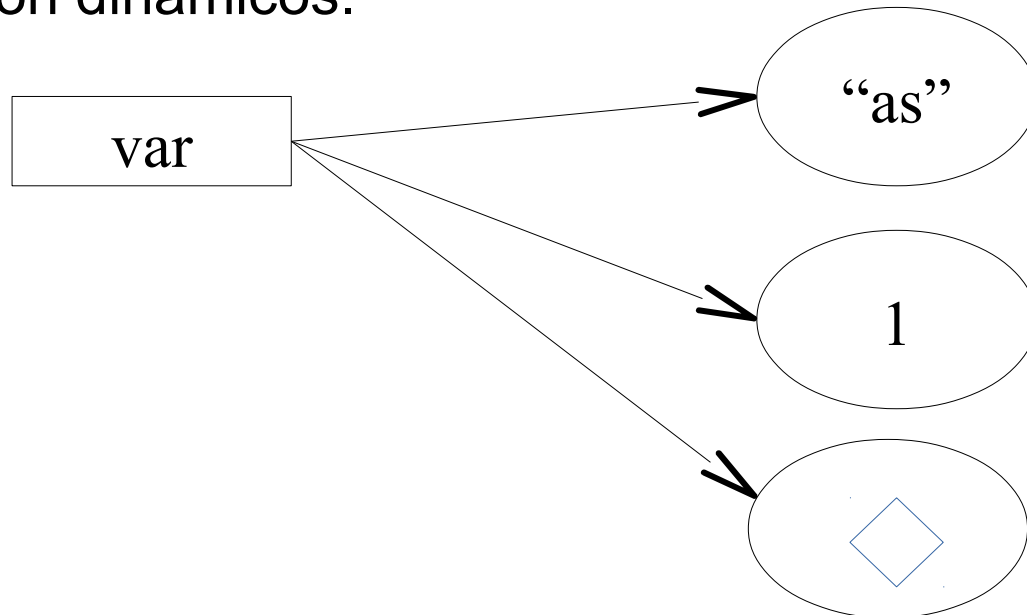
# 1. Definición de polimorfismo

- La palabra polimorfismo (“multi-formas”) se refiere a la capacidad para adoptar varias formas.
- **Polimorfismo:** Capacidad de una entidad (identificador) de referenciar a objetos de diferentes clases (tipo) durante la ejecución de un programa.



## 2. Tipo estático y dinámico

- **Tipo estático:** Tipo (clase) del que se declara la variable, no cambia durante la ejecución.
- **Tipo dinámico:** Clase a la que pertenece el objeto al va referenciando una variable a lo largo de la ejecución, va cambiando durante la ejecución.
- Java es un lenguaje con tipo estático y dinámico y en Ruby los tipos son dinámicos.



## 2. Tipo estático y dinámico: Ejemplo



```
class Persona {... void hablar() {return "bla bla";}}  
class Alumno extends Persona{...  
    @Override  
    void hablar() {return "soy un alumno";}  
    void estudiar() {return "estudiando";}  
}
```

---

```
Persona p; //Tipo estático de p: Persona, clase en la declaración
```

```
p= new Persona(); /**Tipo dinámico de p: Persona (clase del  
objeto), tipo estático sigue siendo Persona*/
```

```
p= new Alumno(); /**Tipo dinámico de p : Alumno (clase del  
objeto), tipo estático sigue siendo Persona*/
```



# 3. Polimorfismo y ligadura dinámica

- **Ligadura Estática:** El enlace del método al mensaje se basa en el tipo estático de la variable y se realiza en tiempo de compilación. En C++ y Objective C si se especifica. Ej. C++ ligadura estática.

```
Ave * var = new Ave();  
var->canta(); // “♪♪♪soy una avecilla que canta al albor ♪♪♪”  
var = new Pato();  
var->canta(); // “♪♪♪soy una avecilla que canta al albor ♪♪♪”
```

- **Ligadura Dinámica:** El enlace del método al mensaje se basa en el tipo dinámico de la variable y se realiza en tiempo de ejecución. En Java, Smalltalk, Ruby, C++ con métodos “virtual” (forma de indicar ligadura dinámica en C++). Ej. Java

```
Ave var = new Ave();  
var.canta(); // “♪♪♪soy una avecilla que canta al albor ♪♪♪”  
var = new Pato();  
var.canta(); // “♪♪♪cuack, cuack, cuack ♪♪♪”
```

### 3. Polimorfismo y ligadura dinámica

---

- **No existe polimorfismo sin ligadura dinámica**, ya que ésta permite que:
  - Una variable referencie en tiempo de ejecución a objetos de diferente clase.
  - Un mensaje se ligue a un método u otro dependiendo de la clase del objeto receptor en ese momento.

# 3. Polimorfismo y ligadura dinámica



```
class Persona {... void hablar() {return "bla bla";}}  
class Alumno extends Persona{...  
    @Override  
    void hablar() {return "soy un alumno";}  
    void estudiar() {return "estudiando";}  
}
```


---

```
Persona p; //Tipo estático de p: Persona  
p= new Persona(); // Tipo dinámico de p: Persona  
p.hablar(); //ligadura dinámica del mensaje al método  
  
p= new Alumno(); //Tipo dinámico de p: Alumno  
p.hablar(); //ligadura dinámica del mensaje al método  
p.estudiar(); //ligadura dinámica del mensaje al método
```

## 4. Polimorfismo y lenguajes con tipo estático

- En los lenguajes con tipo estático, la “forma” que puede adoptar una variable, es decir, las clases de objetos que puede referenciar está limitada por el tipo estático de la variable. Estas limitaciones se conocen como

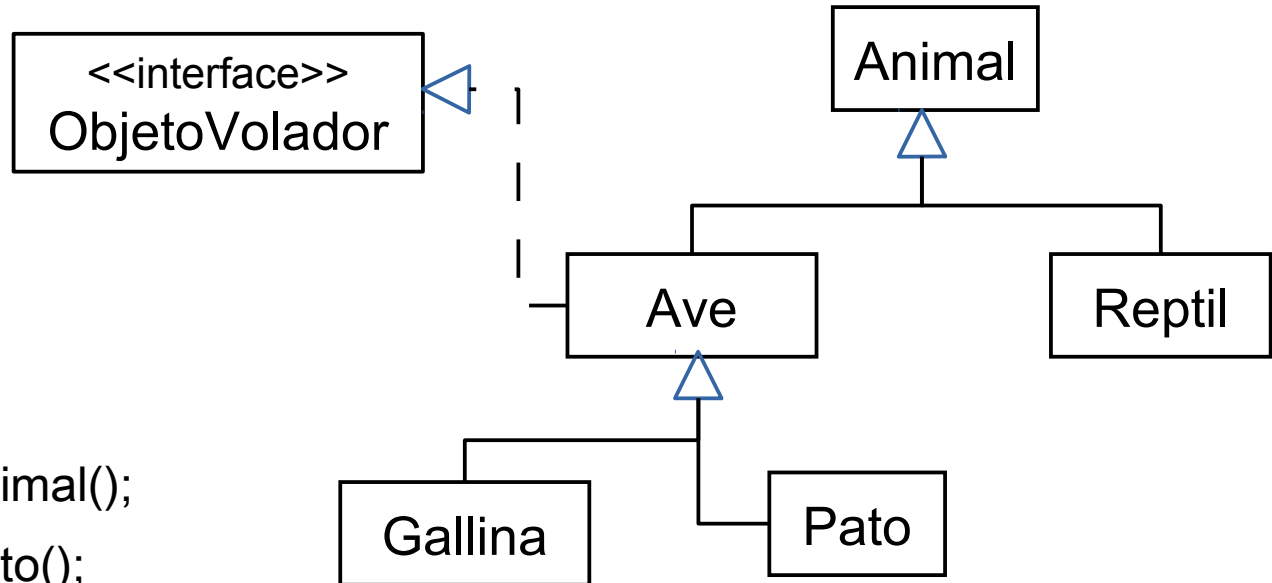
### Reglas de compatibilidad de tipos en OO:

- **Entre clases (a través de la herencia):** El tipo dinámico de una variable puede ser la clase declarada, que es su tipo estático, o el de alguna de sus subclases.
- **Entre Interfaz y clases (a través de la realización):** El tipo dinámico de una variable puede ser el tipo estático de la interfaz que implementa. 
- En lenguajes con herencia simple y monojerárquica, si queremos eliminar esta limitación, podemos declarar la variable del tipo raíz (generalmente Object).


## 4. Polimorfismo y lenguajes con tipo estático



Ejemplo de uso de la regla de compatibilidad de tipos



```
Ave var1;  
var1 = new Animal();  
var1 = new Pato();  
var1 = new Reptil();  
-----  
ObjetoVolador var2;  
var2 = new Reptil();  
var2 = new Ave();  
var2 = new Gallina();
```



¿En qué líneas de código hay error de compilación por no cumplir alguna regla de compatibilidad de tipos?

## 4. Polimorfismo y lenguajes con tipo estático



**Problema en compilación:** En compilación el tipo que se conoce de una variable es el estático ¿qué ocurre cuando enviamos un mensaje a un objeto cuyo método no está definido en la clase de su tipo estático?

```
Object var = new Diamond();
```

```
var.power();
```

/\* Error en compilación, el método power() no está definido en Object, que es dónde busca el compilador, ya que éste sólo conoce el tipo estático de la variable \*/

Solución ---> hacer lo que se conoce como casting (ojo: no hay conversión de tipos, solamente indicación explícita de cuál es el tipo dinámico esperado)

```
((Diamond)var).power();
```

## 4. Polimorfismo y lenguajes con tipo estático



**Problema en ejecución:** cuando no se corresponde el tipo dinámico de la variable con el tipo que se le ha indicado para evitar un error en compilación.

```
Persona var1 = new Estudiante("Juan");
```

```
Persona var2 = (Profesor)var1;
```

O

```
Persona var3 = new Profesor("Pedro");
```

```
((Estudiante)var3).estudiar();
```

No cambia  
el tipo  
dinámico  
de las  
variables  
*var1* ni  
*var3*

En ambos casos se produce el siguiente **error en ejecución**:

```
java.lang.ClassCastException
```

¿qué significa  
esta excepción?



## 4. Polimorfismo y lenguajes con tipo estático



```
Animal animal= new Animal();
```

```
Reptil reptil = new Reptil();
```

```
animal = reptil; // OK
```

```
reptil = animal; // error en compilación ¿por qué?
```

---

```
Animal animal= new Animal();
```

```
Reptil reptil = new Reptil();
```

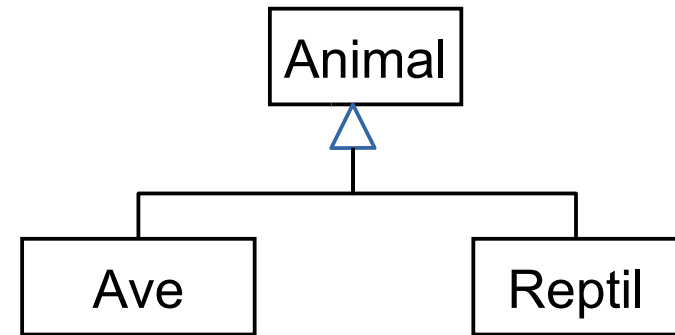
```
reptil = (Reptil) animal; //error en ejecución ¿por qué?
```

---

```
Animal animal= new Reptil();
```

```
Reptil reptil = new Reptil();
```

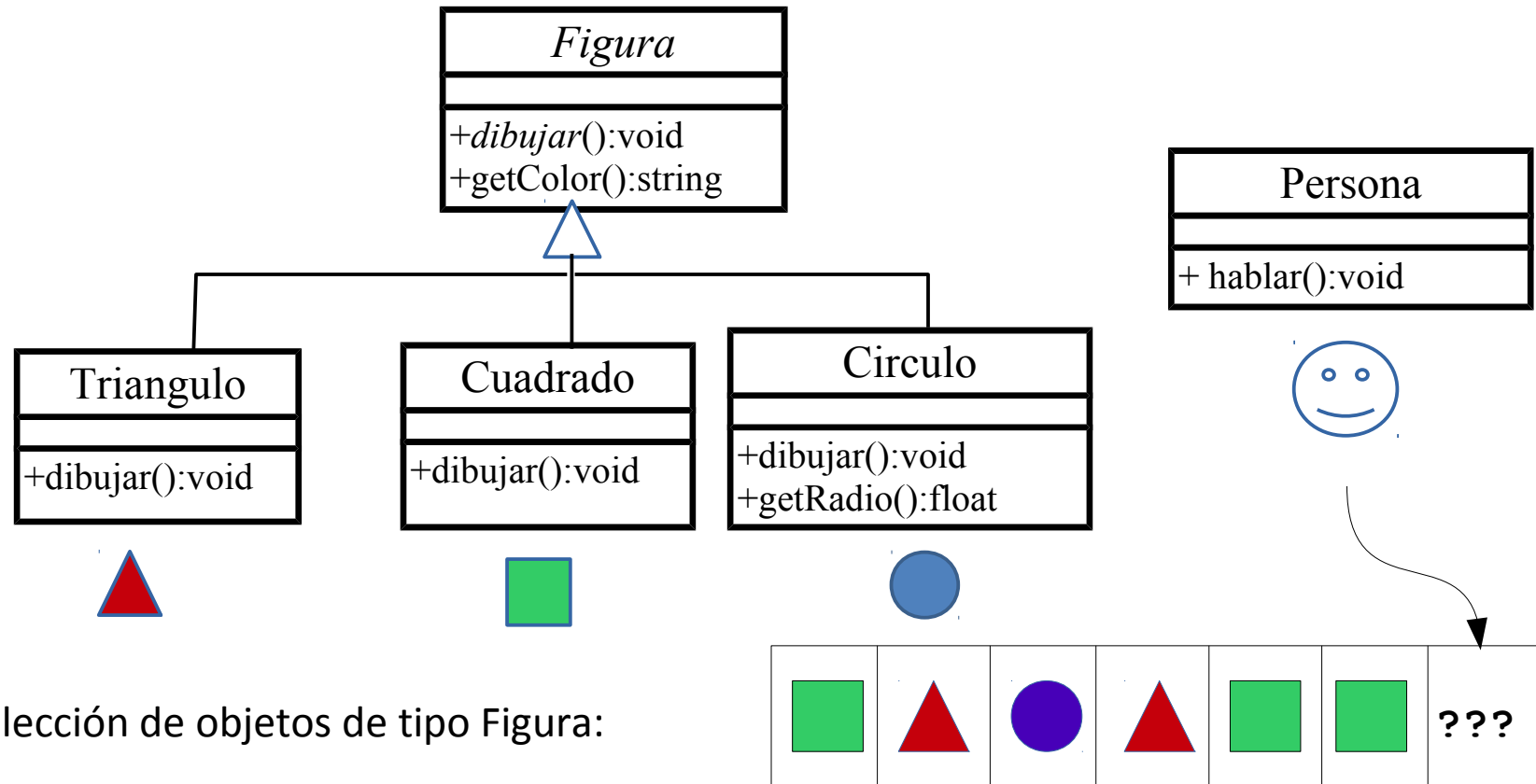
```
reptil = animal; //¿Hay error algún error? ¿por qué?
```





## 4. Polimorfismo y lenguajes con tipo estático

### El “problema del contenedor”



¿Cómo se comportan los distintos elementos en compilación y ejecución antes los envíos de mensajes `dibujar()` y `getRadio()`? y ¿cuándo se conoce el objeto receptor de estos envíos de mensaje?



## 4. Polimorfismo y lenguajes con tipo estático



### El “problema del contenedor”

//Circulo y Cuadrado heredan de Figura

```
Figura[] figuras = new Figura[3];
```

```
Figura figuras[0]= new Circulo(“rojo”);
```

```
Circulo figuras[1]=new Circulo(“azul”);
```

```
Cuadrado figuras[2]=new Cuadrado(“verde”);
```

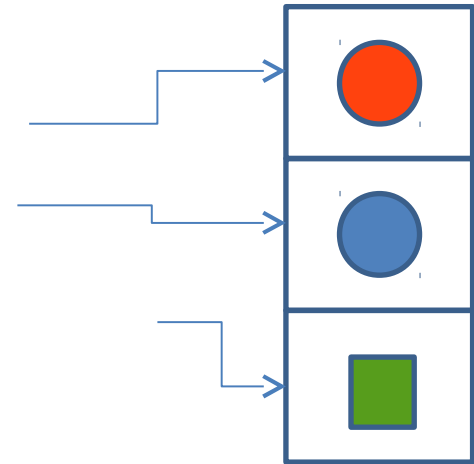
```
for (Figura figura:figuras){
```

```
    figura.getColor();
```

```
    figura.dibujar();
```

```
    figura.getRadio(); // error en compilación
```

```
}
```



## 4. Polimorfismo y lenguajes con tipo estático



### El “problema del contenedor”

```
//Circulo y Cuadrado heredan de Figura
```

```
Figura[] figuras = new Figura[3];
```

```
Figura figuras[0]= new Circulo(“rojo”);
```

```
Circulo figuras[1]=new Circulo(“azul”);
```

```
Cuadrado figuras[2]=new Cuadrado(“verde”);
```

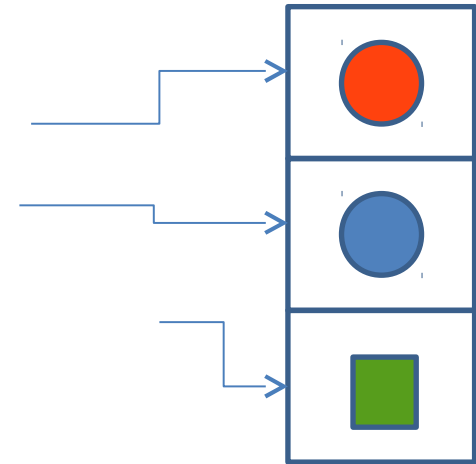
```
for (Figura figura:figuras){
```

```
    figura.getColor();
```

```
    figura.dibujar();
```

```
    ((Circulo)figura).getRadio();
```

```
// solucionado el error de compilación
```



¿Habría error en ejecución?



## 4. Polimorfismo y lenguajes con tipo estático



### El “problema del contenedor”

//Circulo y Cuadrado heredan de Figura

```
Figura[] figuras = new Figura[3];
```

```
Figura figuras[0] = new Circulo(“rojo”);
```

```
Circulo figuras[1] = new Circulo(“azul”);
```

```
Cuadrado figuras[2] = new Cuadrado(“verde”)
```

```
for (Figura figura:figuras){
```


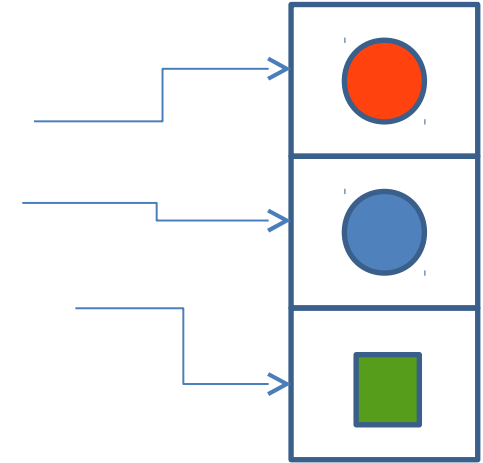
```
    figura.getColor();
```

```
    figura.dibujar();
```

```
//solucionado el error en ejecución
```

```
if (figura instanceof Circulo)
```

```
    ((Circulo)figura).getRadio();
```

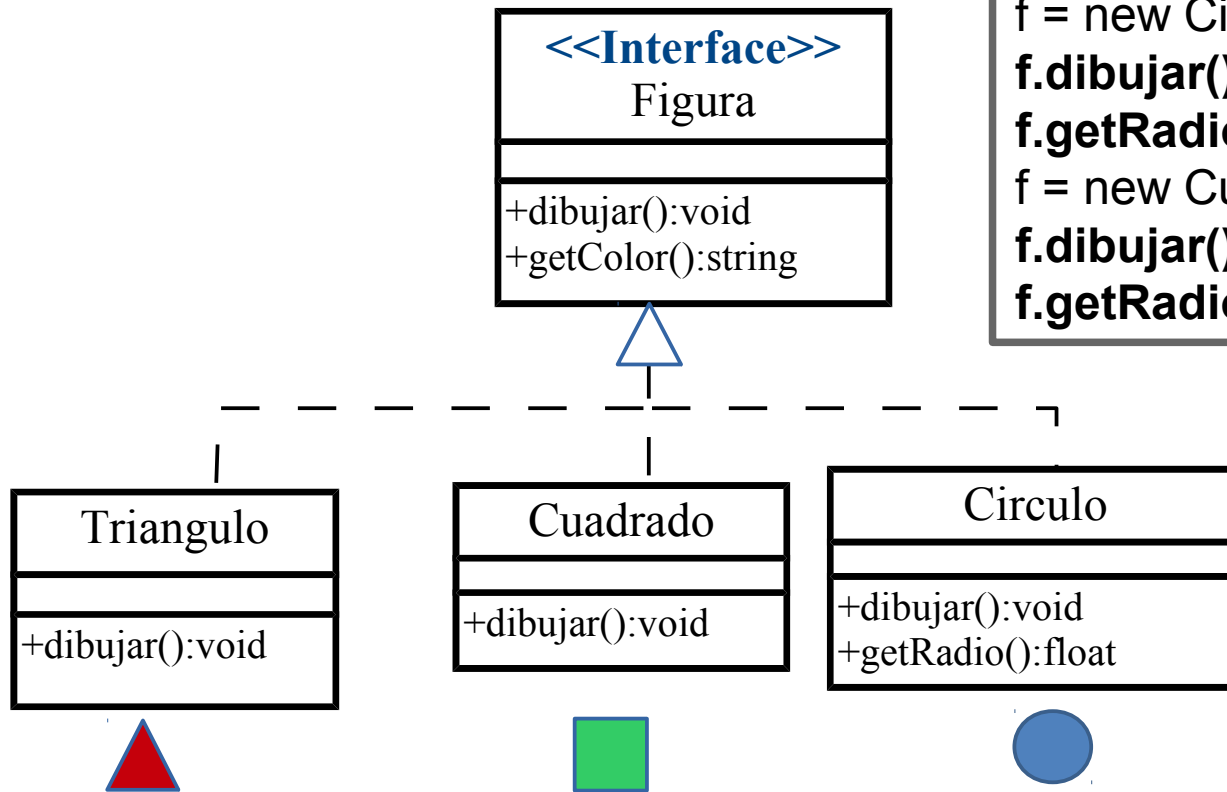


Esta solución  
no es la más  
adecuada.  
¿Alternativa?

## 4. Polimorfismo y lenguajes con tipo estático



La interfaz funciona a efectos de polimorfismo y tipo estático como una clase abstracta, ejemplo:



```

Figura f;
f = new Circulo();
f.dibujar();
f.getRadio();
f = new Cuadrado();
f.dibujar();
f.getRadio();
  
```

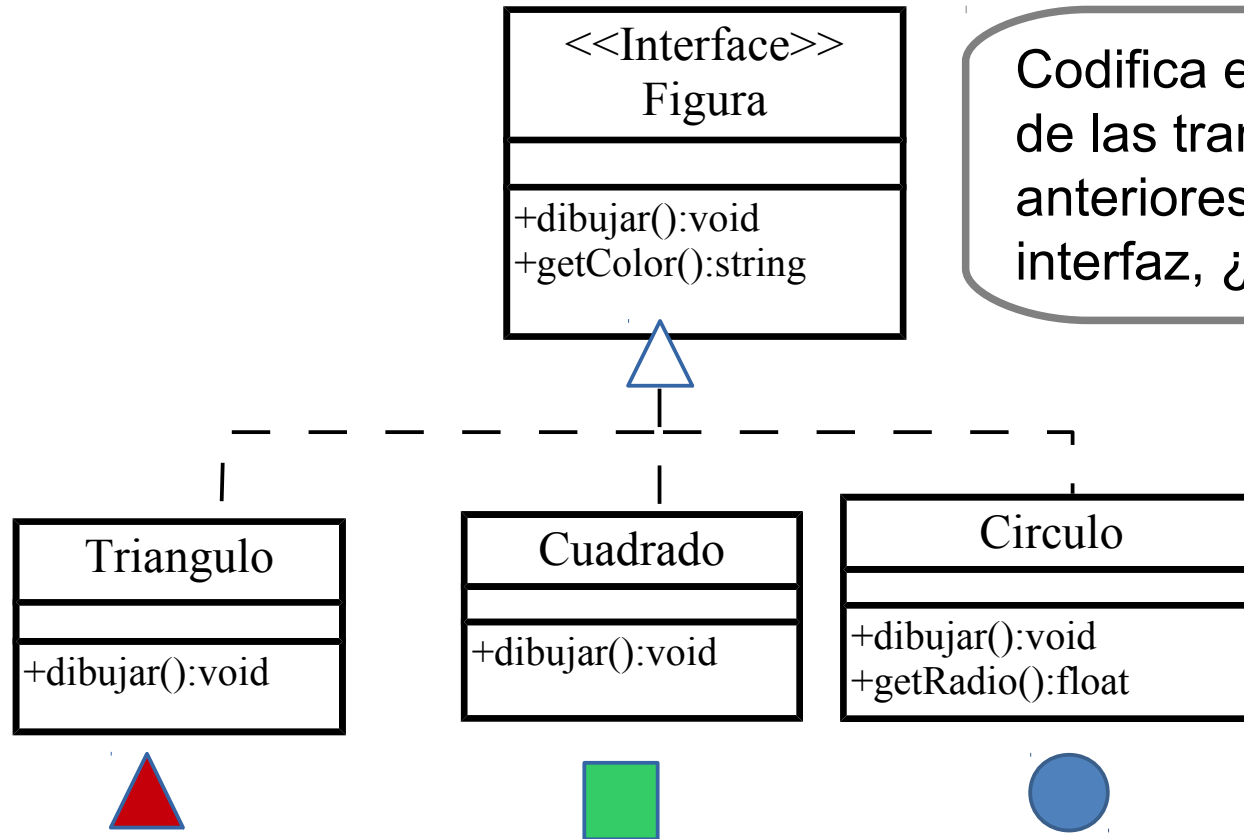
Corrige los posibles errores de compilación y de ejecución

¿Es correcta la asignación: `f=new Figura()`?

## 4. Polimorfismo y lenguajes con tipo estático



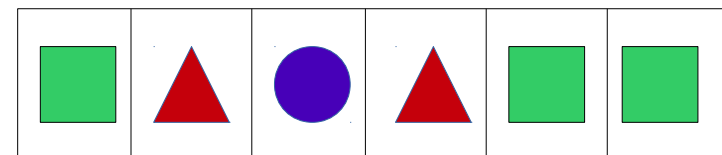
### El “problema del contenedor” usando interfaces



Codifica el mismo ejemplo de las transparencias anteriores usando la interfaz, ¿cambia algo?

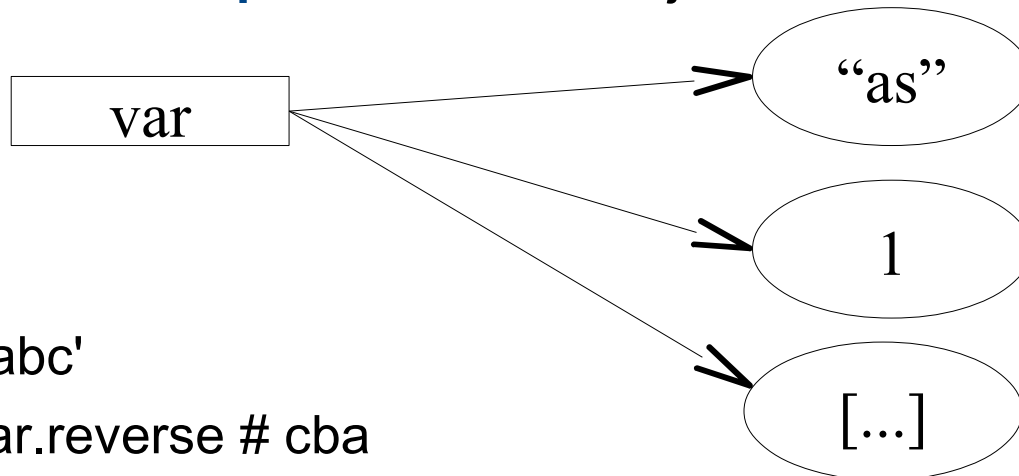


Colección de objetos de tipo **Figura**



## 5. Polimorfismo en lenguajes sin tipo estático

En los lenguajes sin definición de tipo estático no hay limitación en cuanto a la “forma” que puede adoptar una variable, es decir, puede referenciar a cualquier objeto durante la ejecución. Pero sí hay **limitación con el tipo dinámico** en ejecución.



```
var = 'abc'
puts var.reverse # cba
var = 8
puts var.even # true
var = ['abc',1,8.5]
print var.reverse # [8.5, 1, "abc"]
puts var.even
```

Resultado de ejecución de la última línea de código

**#NoMethodError**:undefined method `even' for ["abc",1,8.5]:Array

## 6. No es polimorfismo

- Los mecanismos de abstracción y reusabilidad que no son *polimorfismo de mensajes mediante herencia e interfaces*, no se consideran polimorfismo. Por ejemplo:

- Sobrecarga de operadores, funciones o métodos

Ejemplo de sobrecarga de operador:

**3+5 = 9      'ho' + 'la' = 'hola'**

Ejemplo de sobrecarga de método:

**figura.rotar(punto)      figura.rotar(x,y)**

- Tipos y clases parametrizables

Ejemplo de clase parametrizable:

**ArrayList<Integer>      ArrayList<String>**