



RTTI y reflexión

Modelos de objetos

Carlos Fontela
cfontela@fi.uba.ar



Temario

RTTI (información de tipos en tiempo de ejecución)

Reflexión

Modelos de objetos en distintos lenguajes



Polimorfismo de tipos

Transformación automática

```
Elipse e = new Elipse();  
Figura f = e;
```

Transformación explícita

```
Elipse e2 = (Elipse)f;  
// f.setRadioMayor(3); f es una Elipse, pero...  
(Elipse)f.setRadioMayor(3);
```

Se hace chequeo

¡Todo objeto conoce su tipo!

¡El tipo del objeto nunca cambia!

En Smalltalk es todavía más obvio



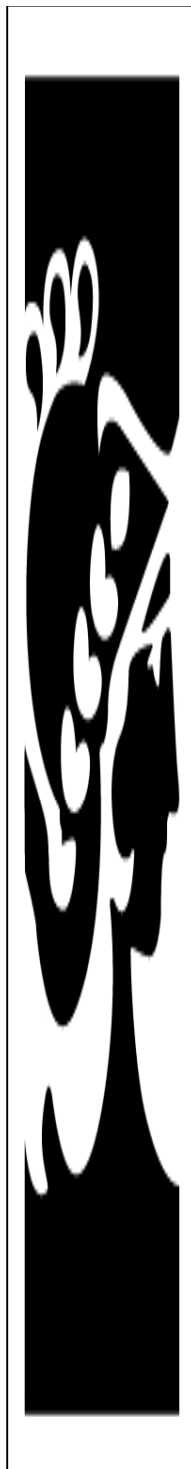
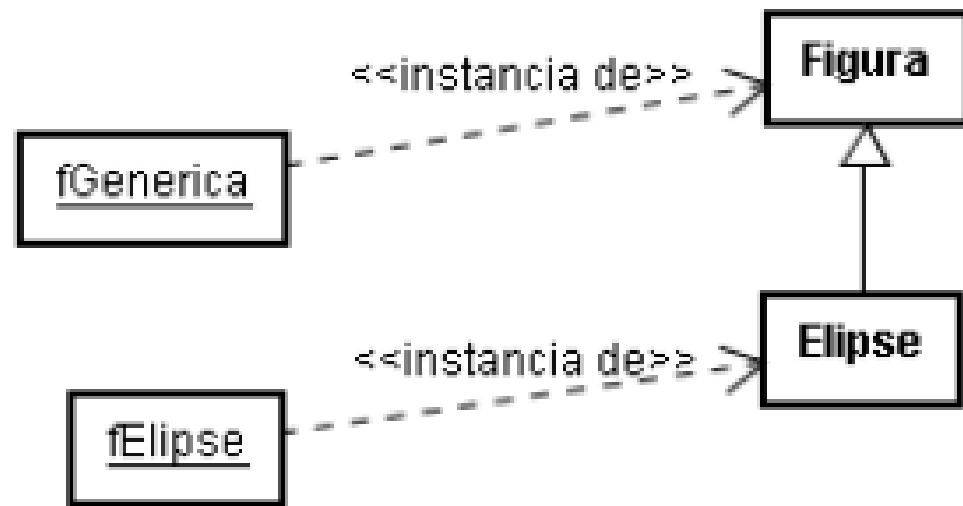
Ejemplo marco

Java

```
Figura fGenerica = new Figura ( );  
Figura fElipse = new Elipse ( );
```

Smalltalk

```
fGenerica := Figura new.  
fElipse := Elipse new.
```



RTTI

Información de tipos en tiempo de ejecución

2 situaciones

Quiero conocer la clase exacta de un objeto

Quiero conocer la familia de la clase del objeto

Veremos:

Cómo lograrlo

Inconvenientes de RTTI

Cómo está implementado



RTTI: conocer la clase

Java

```
assertTrue (fGenerica.getClass( ) == Figura.class);  
assertTrue (fElipse.getClass ( ) == Elipse.class);  
assertFalse (fElipse.getClass ( ) == Figura.class);
```

Smalltalk

```
self assert: (fGenerica class = Figura).  
self assert: (fElipse class = Elipse).  
self deny: (fElipse class = Figura).
```



RTTI: conocer la familia

Java

```
assertTrue (fGenerica instanceof Figura);  
assertTrue (fElipse instanceof Figura);  
assertTrue (fElipse instanceof Elipse);
```

Smalltalk

```
self assert: (fGenerica isKindOf: Figura).  
self assert: (fElipse isKindOf: Elipse).  
self assert: (fElipse isKindOf: Figura).
```



Problemas con RTTI

Compromete la extensibilidad

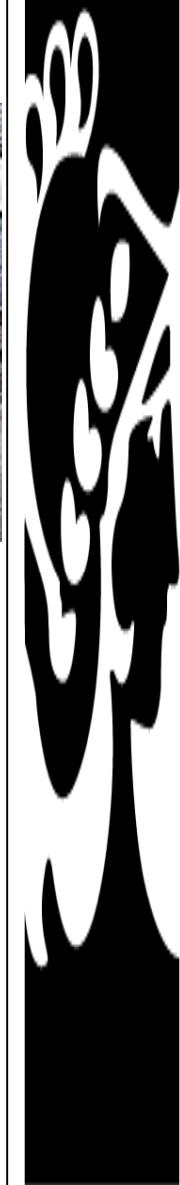
```
public Punto posicion ( ) {  
    if (this instanceof Elipse)  
        return ((Elipse) this).getCentro( );  
    if (this instanceof Poligono)  
        return  
            ((Poligono) this).getContorno( )[0];  
    throw new IllegalArgumentException( );  
}
```



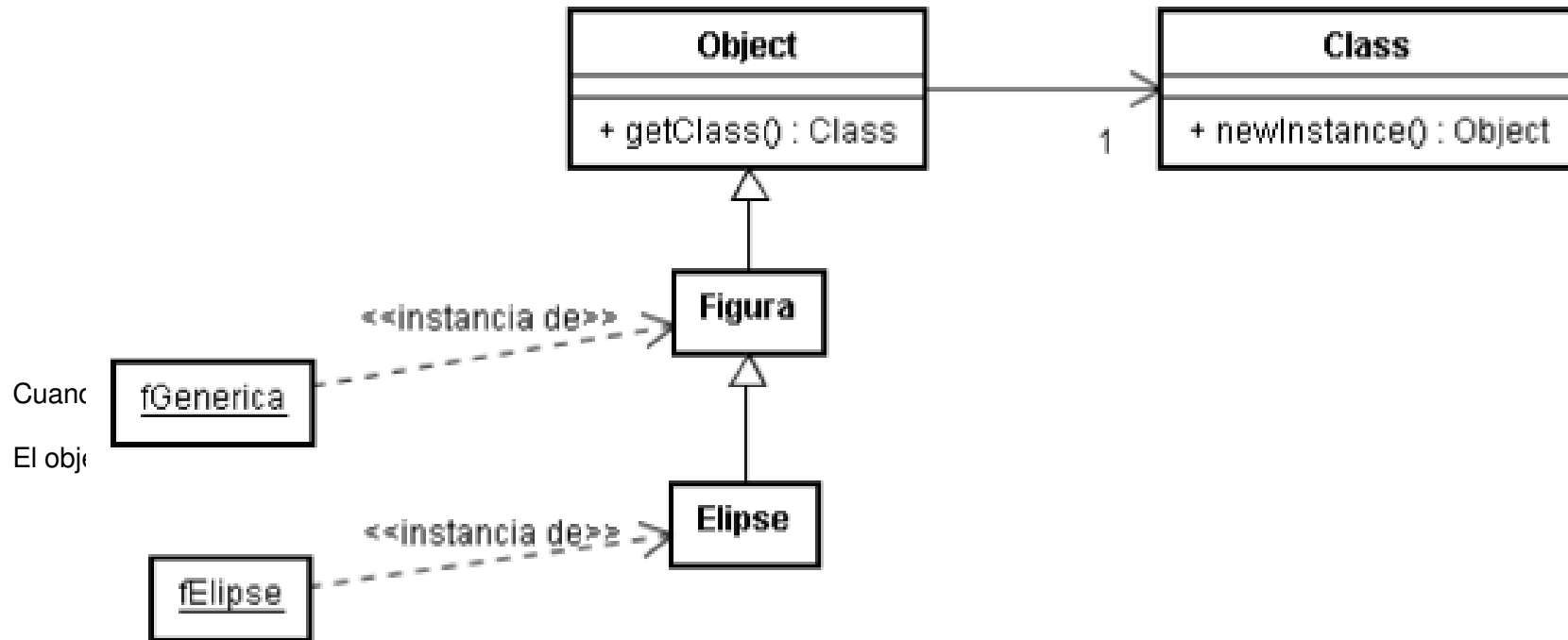
Evita el polimorfismo

```
if (x instanceof Elipse) ((Elipse)x).dibujar( );
```

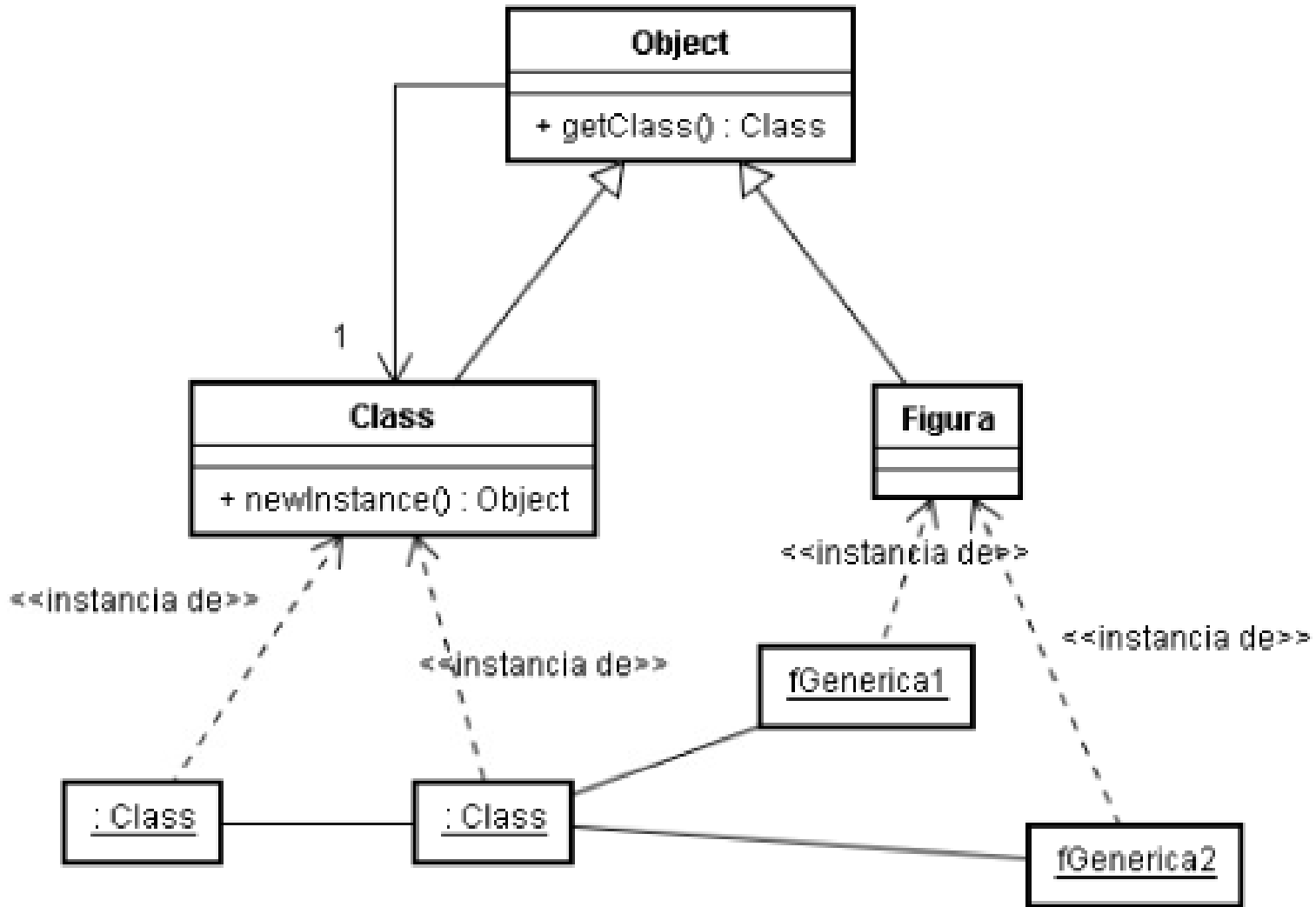
Muy contrario a POO en general



RTTI en Java (1)



RTTI en Java (2)



RTTI en Smalltalk: principios

Las clases son objetos

Las clases son instancias de una metacalse

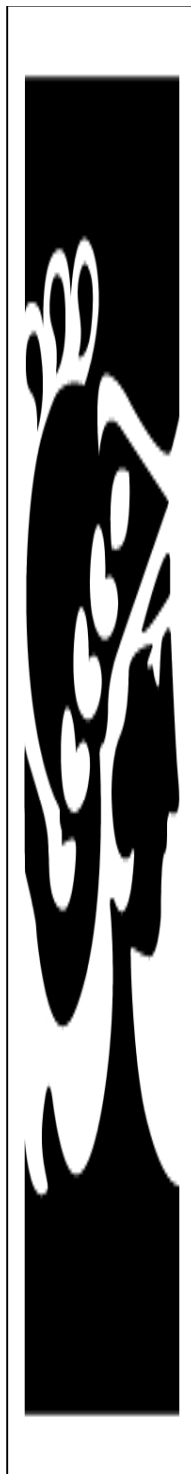
Hay una jerarquía de metaclases paralela a la de las clases

Cada metacalse hereda de Class

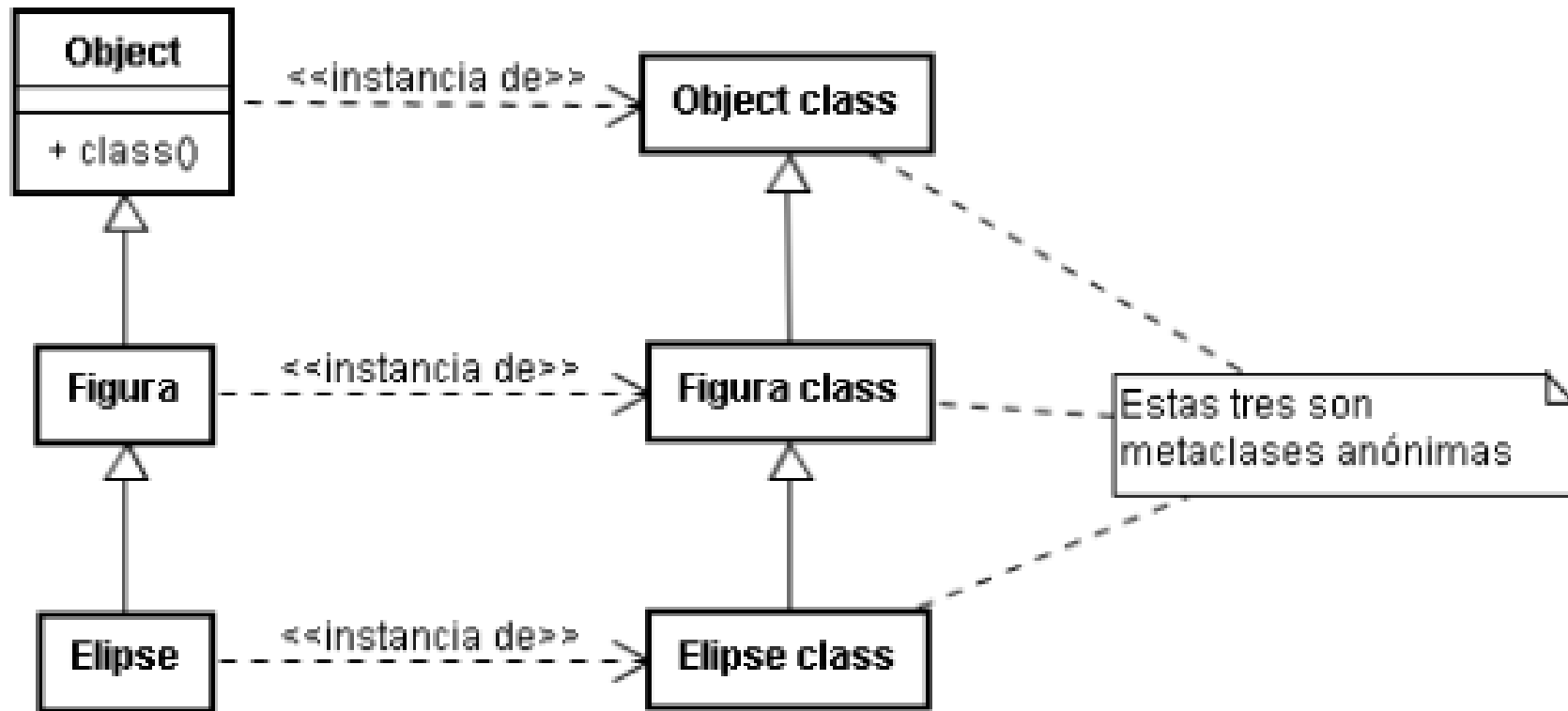
Class hereda de Behavior

Las metaclases son instancias de Metaclass

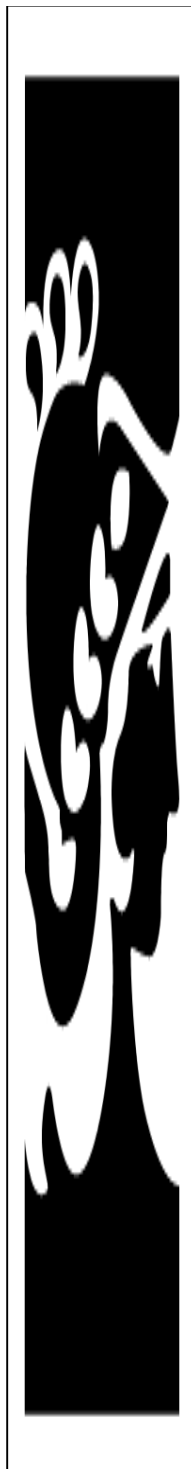
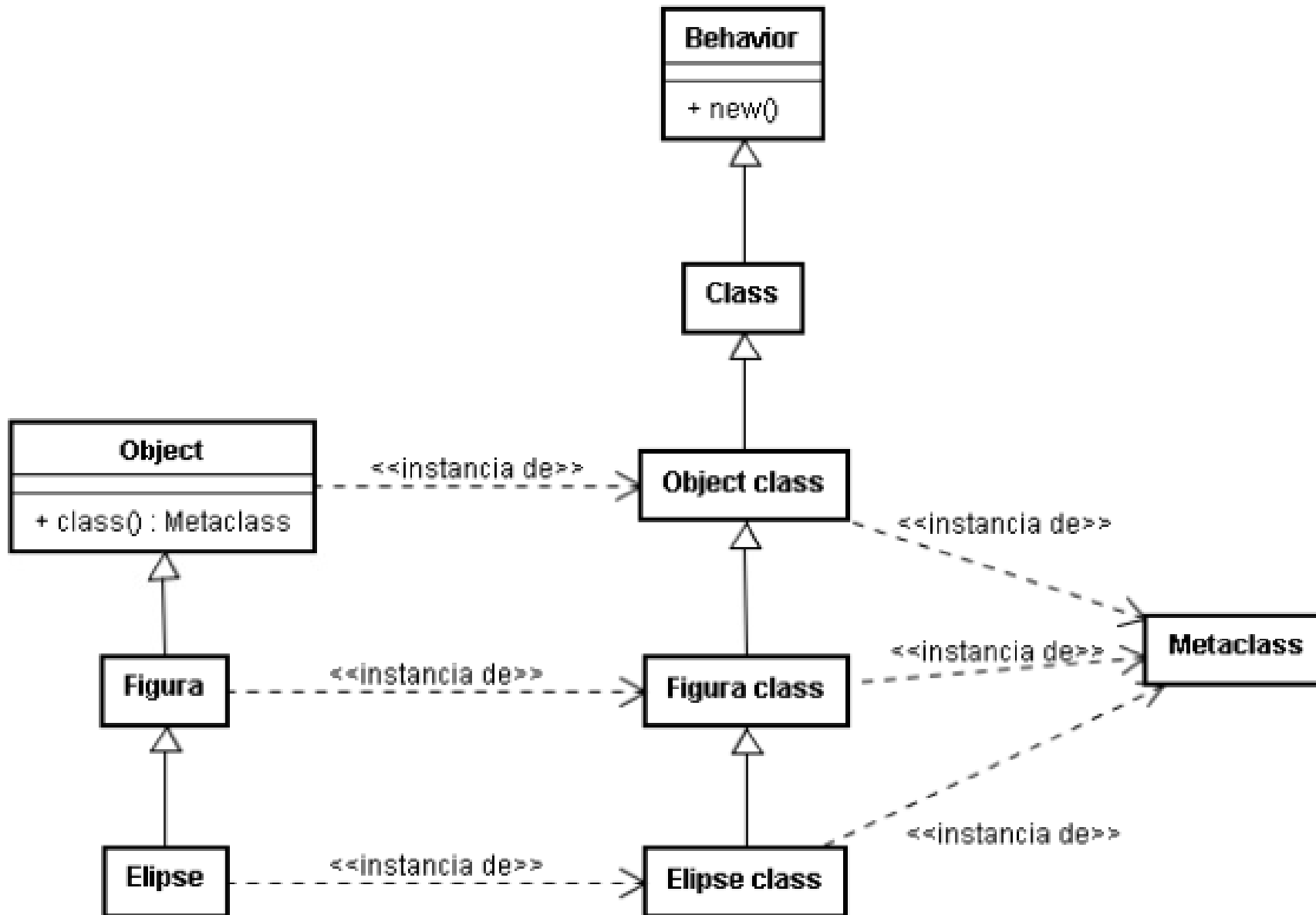
La metacalse de la clase Metaclass es una instancia de Metaclass



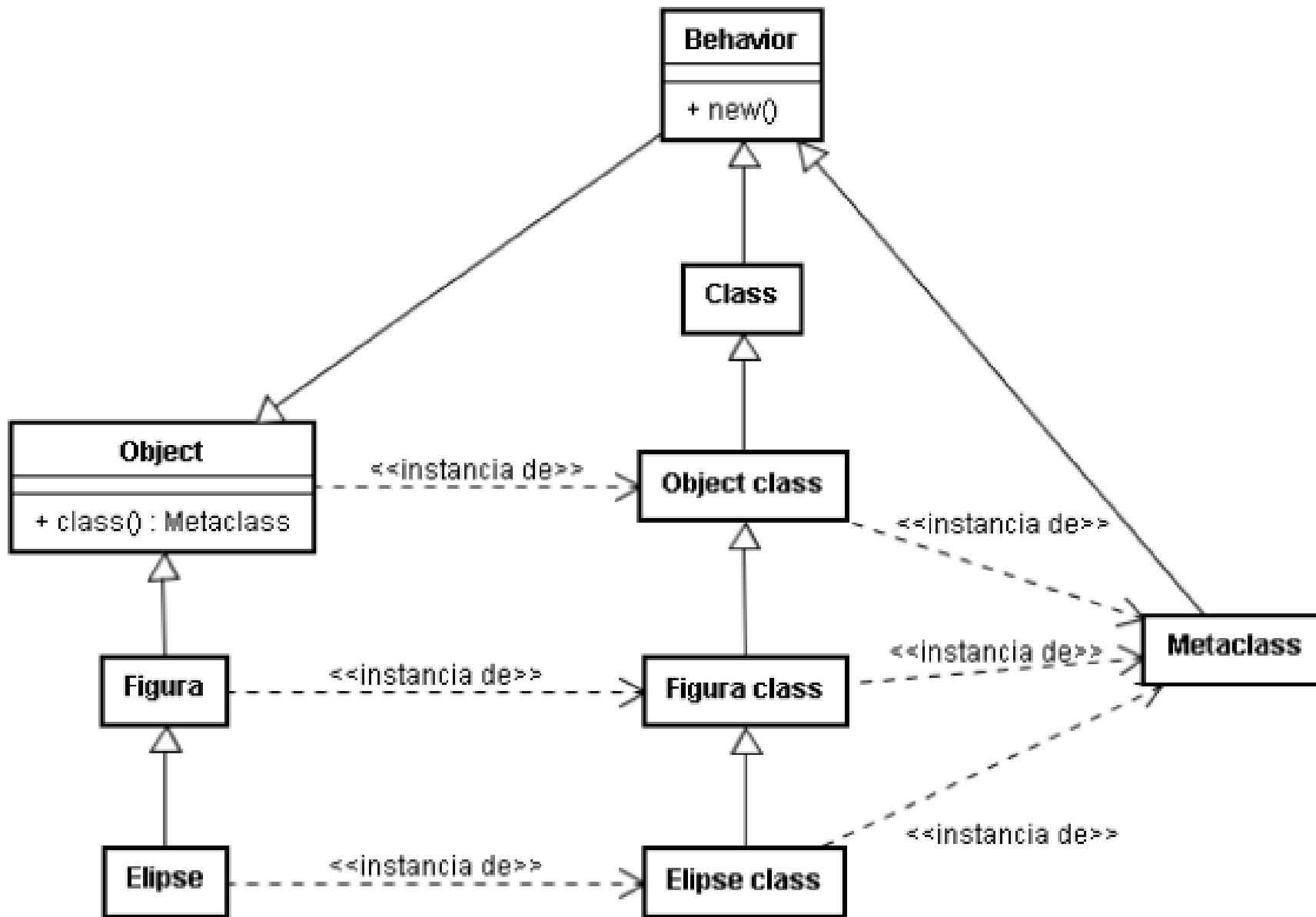
Smalltalk: las clases son objetos



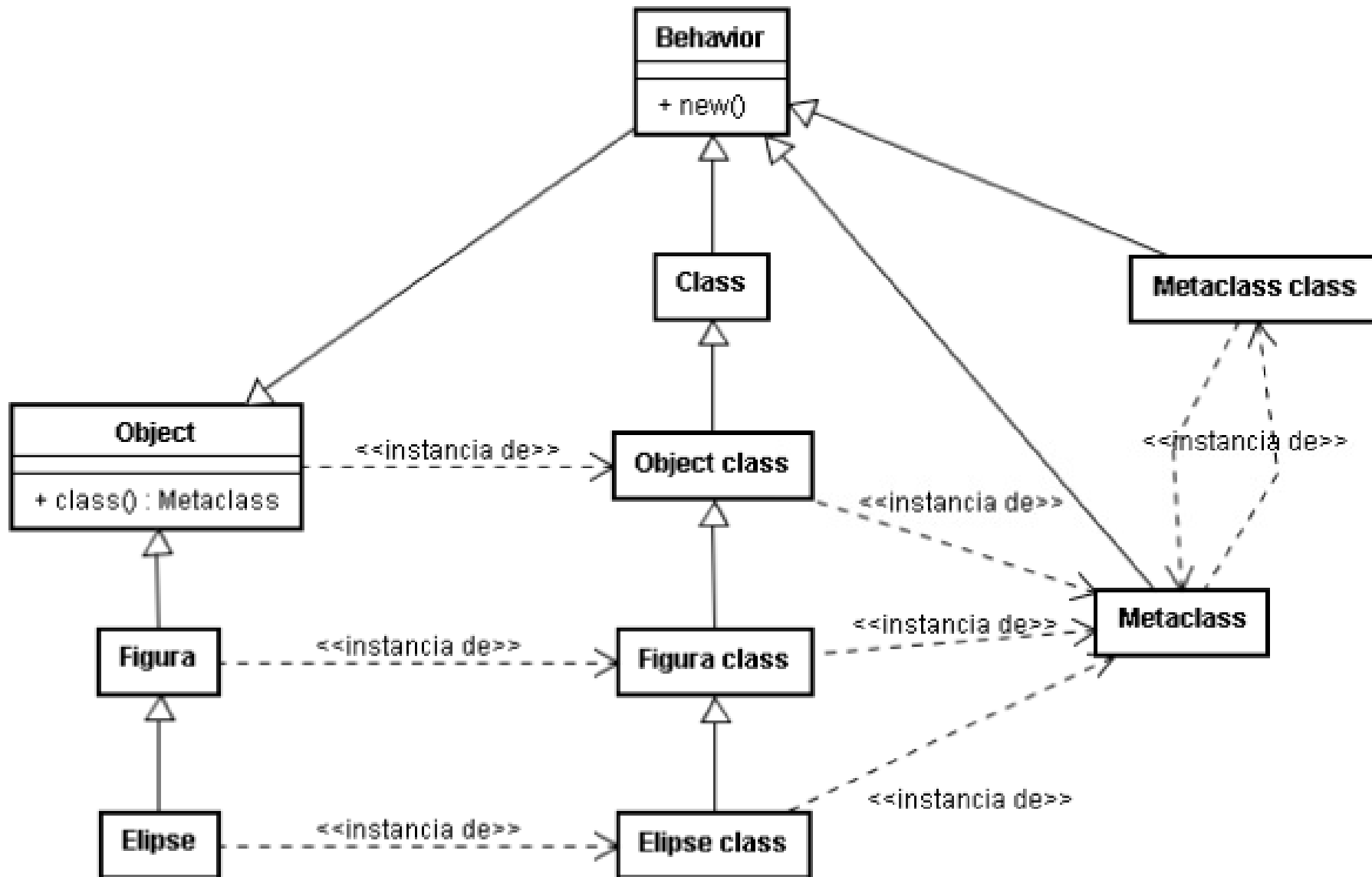
Smalltalk: jerarquía de metaclasses



Smalltalk: todo cierra



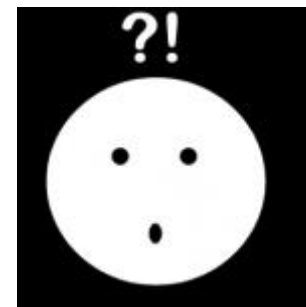
Smalltalk: clases de metaclasses



Reflexión en Java (1)

Cosas raras (faltan excepciones):

```
public void metodoRaro (Object o1) throws Exception {  
    Class claseAncestro = o1.getSuperclass();  
    Object o2 = claseAncestro.newInstance();  
    Method m = o2.getClass().getConstructors()[0];  
    m.invoke();  
}
```



Reflexión en Java (2)

RTTI: el compilador debe conocer los tipos

¿Qué pasa si recibo un objeto por una red?

Puedo obtener el objeto Class y preguntar por métodos, atributos, interfaces, etc.

Paquete `java.lang.reflect`

La información debe estar en tiempo de ejecución

En general se maneja en forma automática

Interrogación a componentes para saber qué eventos

soporta



Reflexión en Java (3)

Clases

Method, Constructor, Field

Métodos

getMethods(), getConstructors(), getFields(),
getInterfaces(), getSuperclass()

newInstance()

En clase Field: get() y set()

En clase Method: invoke()

Mucho más



Ejemplo: creación de objetos

```
Fraccion f = new Fraccion();
```

La clase está especificada en el código y se conoce en tiempo de compilación

¿Qué pasaría si hiciésemos?:

```
Serializable f = x.crearObjeto(); // Serializable es una interfaz
```

El método podría ser:

```
public Serializable crearObjeto ( ) {  
    String nombreClase = leerArchivoConfiguracion();  
    Class claseInstanciar = Class.forName(nombreClase);  
    Object nuevo = claseInstanciar.newInstance();  
    return (Serializable) nuevo;  
}
```

Ahora la clase se conoce recién en tiempo de ejecución



Reflexión: ¿ya la usamos?

Framework JUnit

¿Cómo funciona?



Usa polimorfismo

Métodos `setUp()` y `tearDown()`

Y reflexión

Métodos “`public void testXxx()`”

Algo bastante común en todos los frameworks



Reflexión en Smalltalk

Más potente que en Java

También más compleja

2 niveles:

Introspección

- Similar a Java, con agregados

Intersección

- Actuación sobre el entorno de ejecución

Admite la metaprogramación

Ver “Pharo By Example”, capítulo 14



Introspección

Hay más control sobre el entorno de ejecución
que en Java

Todos son objetos

Workspace, Debugger, Inspector, Transcript

Hay más información

Subclases

Instancias existentes

Toda la jerarquía de herencia

Uso de objetos desde métodos

Referencias cruzadas entre métodos



Intersección y metaprogramación

Control total de los objetos durante la ejecución

Permite la metaprogramación

Cambios de comportamiento en forma dinámica

Por ejemplo, `doesNotUnderstand`

O generar métodos que no existen

Ejemplo del Proxy



Ojo con reflexión

Podemos terminar generando cualquier cosa
Síndrome del elefante alado

Difícil de testear
Difícil de leer
No cualquiera la usa bien



Modelo de objetos y lenguajes

Todo es un objeto

Las clases son objetos

Todo ocurre debido al envío de un mensaje

Los objetos se acceden a través de referencias

Recolección automática de basura

Tipeo estático o dinámico

Todo objeto es instancia de una clase

Toda clase tiene una clase madre

Polimorfismo

Encapsulamiento

Herencia múltiple

Genericidad

Diseño por contrato

Reflexión



Todo es un objeto

Smalltalk 😊

C# 😊

Aún los tipos primitivos se comportan como tipos de objetos, pero son tipos cerrados para herencia

Java ↔

Salvo por constantes, variables de tipos primitivos y arreglos primitivos

Python 😊

Eiffel 😊

C++ ●

1c2011 La POO es sólo uno de los paradigmas soportados por C++



Las clases son objetos

Smalltalk 😊

Y son instancias de metaclasses

C# ↔

Hay una clase Type que simula esta propiedad

Java ↔

Hay una clase Class que simula esta propiedad

Python 😊

Eiffel ●

C++ ●



Todo ocurre debido al envío de un mensaje

Smalltalk 😊

C# ↔

Las operaciones son métodos también

Las estructuras de control no

Java ↔

Operaciones y estructuras de control no

Python ↔

Las estructuras de control no

Eiffel ↔

Las estructuras de control no



Los objetos se acceden a través de referencias

Smalltalk 😊

C# 😊

Salvo tipos por valor

Java 😊

Salvo tipos primitivos

Python 😊

Eiffel 😊

C++ ●

Sólo usando punteros



Recolección automática de basura

Smalltalk 😊

C# 😊

Salvo para tipos por valor

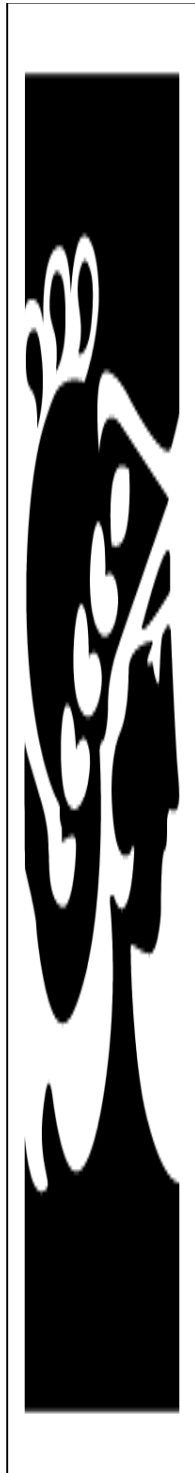
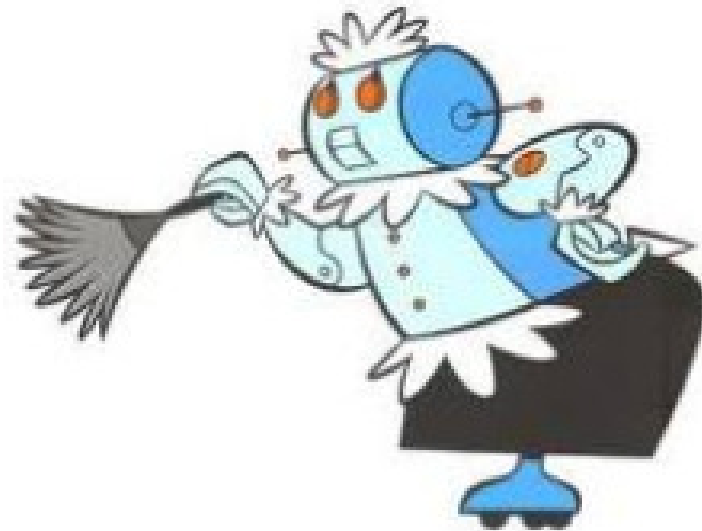
Java 😊

Salvo para tipos primitivos

Python 😊

Eiffel 😊

C++ ●



Tiempo estático o dinámico

Smalltalk: dinámico

C#: estático

Java: estático

Python: dinámico

Eiffel: estático

C++: estático



Todo objeto es instancia de una clase

Smalltalk 😊

C# 😊

Java 😊

Python 😊

Eiffel 😊

C++ 😊



Toda clase tiene una clase madre

Smalltalk 😊

C# 😊

Java 😊

Python ●

Aunque hay una
clase “object”

Eiffel 😊

C++ ●



Polimorfismo

Smalltalk 😊

C# ↔

Métodos virtuales explícitos

Java 😊

Métodos virtuales por defecto

Python 😊

Eiffel 😊

C++ ↔

Métodos virtuales explícitos



Clases y métodos abstractos

Smalltalk ↔

Sólo convencionalmente

C# 😊

Java 😊

Python ●

Eiffel 😊

Los llama diferidos (deferred)

C++ ↔

Métodos abstractos sí

Una clase es abstracta si tiene métodos abstractos



Encapsulamiento

Smalltalk ↔

Atributos son protegidos

C# 😊

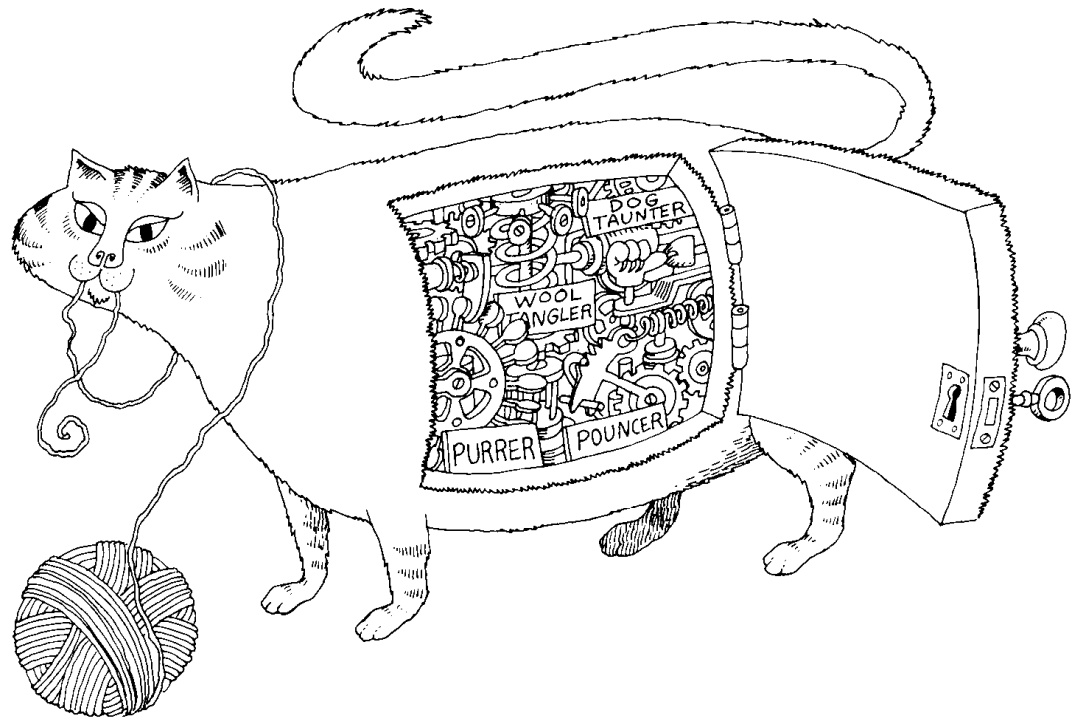
Java 😊

Python ●

Atributos son públicos

Eiffel 😊

C++ 😊



Herencia múltiple

Smalltalk ●

C# ↔

Sólo con interfaces

Java ↔

Sólo con interfaces

Python 😊

Eiffel 😊

C++ 😊



Genericidad

Smalltalk ●

C# 😊

Java 😊

Python ●

Eiffel 😊

C++ ↔

Sólo plantillas de tipos



Diseño por contrato incorporado

Smalltalk ●

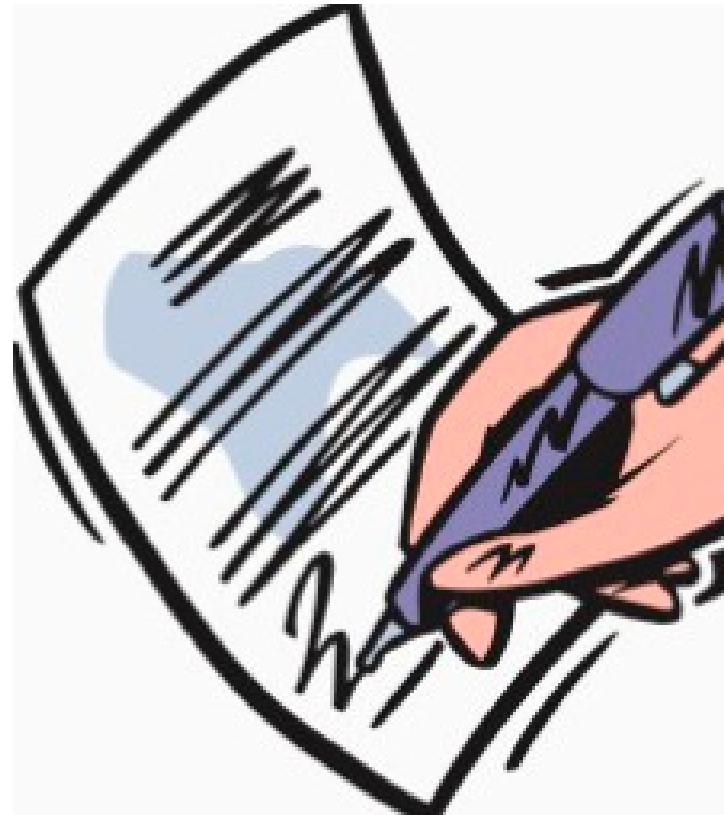
C# ●

Java ●

Python ●

Eiffel 😊

C++ ●



Reflexión

Smalltalk 😊

C# 😊

Java 😊

Python 😊

Eiffel ●

C++ ●



Claves

Todo objeto conoce su clase: RTTI

Se le puede preguntar de todo a un objeto:
reflexión

Usar RTTI y reflexión con medida

En Smalltalk todo es un objeto: ¡ya lo sabíamos!

Los demás lenguajes también son OO, pero tienen
una implementación menos estricta



Lecturas opcionales

Pharo By Example

Capítulo 13, “Classes and Metaclasses”

Capítulo 14, “Reflection”

Thinking in Java, Bruce Eckel

Capítulo 4, “Initialization & Cleanup”

Capítulo 12, “Run-time Type Identification”

Apéndice A, “Passing & Returning Objects”

Está en biblioteca

Hay versión castellana

Orientación a objetos, diseño y programación, Carlos Fontela 2008

Capítulo 20 “Los datos, los tipos y la memoria”

