



POLI MORFISMO

75.07 Algoritmos y programación III

Facultad de Ingeniería, UBA

Pablo Rodríguez Massuh

A close-up photograph of a chameleon's head and upper body. The chameleon has vibrant green skin with yellow and white spots. Its eye is large and prominent, looking directly at the camera. The background is plain white.

¿QUE ENTENDEMOS POR POLIMORFISMO?

Eeeeh ...





POLI



MORFISMO

DEFINICIÓN



El polimorfismo es una **relajación** del sistema de tipos, de tal manera que una referencia a una clase (ya sea un atributo, parámetro, declaración local o elemento de una colección) acepta direcciones de objetos de dicha clase y de sus clases derivadas (hijas, nietas, etc...)



TE LO RESUMO
ASÍ NOMÁS

AH, LA PELOTA !!!

DEFINICIÓN II

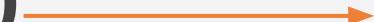
El polimorfismo se refiere a la propiedad por la que es posible enviar **mensajes sintácticamente iguales** a objetos de tipos distintos. El único requisito que deben cumplir los objetos que se utilizan de manera polimórfica es saber responder al mensaje que se les envía.

—
¿Algo así?

cantar()



cantar()



—
¿Algo así?

cantar()



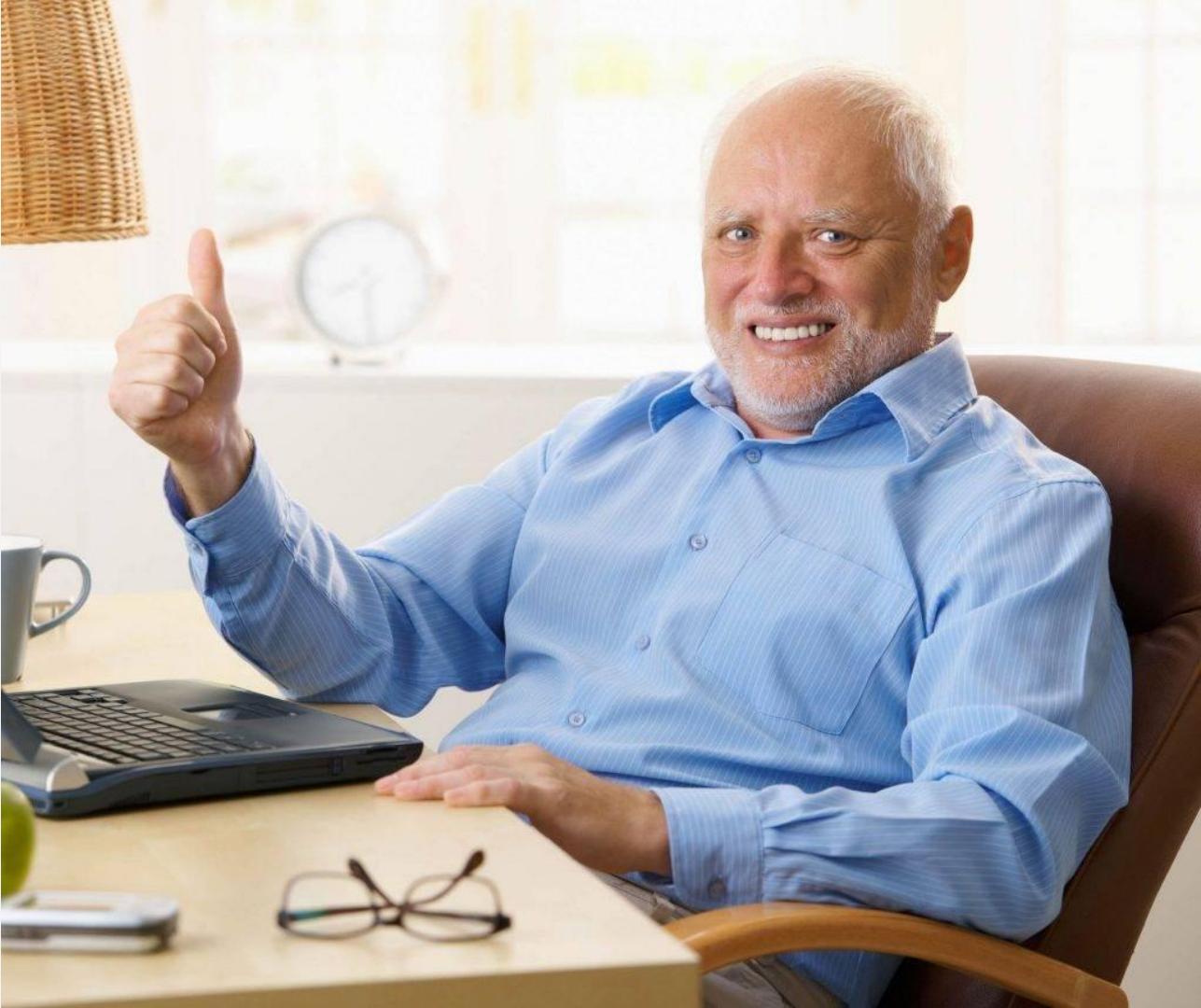
cantar()



cantar()



—
¿Y qué
hacemos con
esto?



A photograph of a group of people, likely spectators or team members, cheering at a sports event. In the center, a man with a beard and dark hair is shouting with his mouth wide open and both arms raised. To his left, another man is partially visible, also with his arms raised. To his right, an older man with white hair is shouting. The background is blurred, showing other people and stadium lights.

Eliminamos
condicionales !!!



Solución inicial programación estructurada

```
if ( tipo_de_cantante == "gallo" ) {  
    print("Co co ro coooo...");  
  
} else if ( tipo_de_cantante == "canario" ) {  
    print("pio pio");  
  
} else if ( tipo_de_cantante == "La MONA" ) {  
    print("Quieee-e-én se ha tomado todo el vino  
          oh oh oh oooh");  
}
```



Y se pone peluda la cosa...

```
if ( tipo_de_cantante == "gallo" ) {  
    print("Co co ro coooo...");  
}  
else if ( tipo_de_cantante == "canario" ) {  
    print("pi pi ri pii");  
}  
else if ( tipo_de_cantante == "La MONA" ) {  
    print("Quieee-e-én se ha tomado todo el vino oh oh oh oooh");  
}  
else if ( tipo_de_cantante == "Ballena Yubarta" ) {  
    print("Al SUR, al SUUR. No, no. Esperemos a tiiiiitooo !!");  
}  
else if ( tipo_de_cantante == "pollito pio" ) {  
    print("El pollito pio, el pollito pio, el pollito pio");  
}
```



Ni hablar con implementaciones reales!

```
4445 function iIds(startAt, showSessionRoot, iNewNVal, endActionsVal, iStringVal, seqProp, htmlEncodeRegEx) {
4446   if (SbUtil.dateDisplayType === 'relative') {
4447     iRange();
4448   } else {
4449     iSelActionType();
4450   }
4451   iStringVal = notifyWindowTab;
4452   startAt = addSessionConfigs_sbRange();
4453   showSessionRoot = addSessionConfigs_elHiddenVal();
4454   var headerDataPrevious = function(tabArray, iNm) {
4455     iPredicateVal.SBDB.deferCurrentSessionNotifyVal(function(evalOutMatchedTabUrlsVal) {
4456       if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
4457         iPredicateVal.SBDB.normalizeTabList(function(appMsg) {
4458           if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
4459             iPredicateVal.SBDB.tabListVal(iNm, evalOutMatchedTabUrlsVal);
4460             if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
4461               iPredicateVal.SBDB.neutralizeWindowFocus(function(iTokenAddedCallback) {
4462                 if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
4463                   iPredicateVal.SBDB.evalSessionConfig2(function(sessionNm) {
4464                     if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
4465                       iPredicateVal.SBDB.iWindow2TabIds(function(iUrlStringVal) {
4466                         if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
4467                           iPredicateVal.SBDB.idx7Val(undefined, iStringVal, function(getWindowIndex) {
4468                             if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
4469                               addTabList(getWindowIndex, undefined, iStringVal, showSessionRoot && showSessionRoot.length > 0 ? showSessionRoot : []);
4470                               if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
4471                                 evalSAleLoading(tabArray, iStringVal, showSessionRoot && showSessionRoot.length > 0 ? showSessionRoot : []);
4472                                 if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
4473                                   BrowserAPI.getAllWindowsAndTabs(function(iSession1Val) {
4474                                     if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
4475                                       SbUtil.currentSessionSrc(iSession1Val, undefined, function(initCurrentSession) {
4476                                         if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
4477                                           addSessionConfigs.render(matchText(iSession1Val, iStringVal, evalSAleLoading, evalSAleLoading));
4478                                         if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
4479                                           unfilteredWindowCount: initCurrentSessionCache,
4480                                           filteredWindowCount: iSession1Val,
4481                                           unfilteredTabCount: parseTabConfig,
4482                                           filteredTabCount: evalRegisterValueVal,
4483                                         });
4484                                       if (seqProp) {
4485                                         seqProp();
4486                                       }
4487                                     });
4488                                   });
4489                                 });
4490                               });
4491                             });
4492                           });
4493                         });
4494                       });
4495                     });
4496                   });
4497                 });
4498               }, showSessionRoot && showSessionRoot.length > 0 ? showSessionRoot : startAt ? [startAt] : []);
4499             });
4500           });
4501         });
4502       });
4503     });
4504   });
4505 }
```

```
float Amount;
float calculateClaim;
if(vehicleClass=='A')
  Amount = 0.7 * distanceTravel;
else if(vehicleClass=='a')
  Amount = 0.7 * distanceTravel;
else if(vehicleClass=='B')
  Amount = 0.6 * distanceTravel;
else if(vehicleClass=='b')
  Amount = 0.6 * distanceTravel;
else if(vehicleClass=='C')
  Amount = 0.5 * distanceTravel;
else if(vehicleClass=='c')
  Amount = 0.5 * distanceTravel;
else if(vehicleClass=='D')
  Amount = 0.45 * distanceTravel;
else if(vehicleClass=='d')
  Amount = 0.45 * distanceTravel;

return calculateClaim;
```



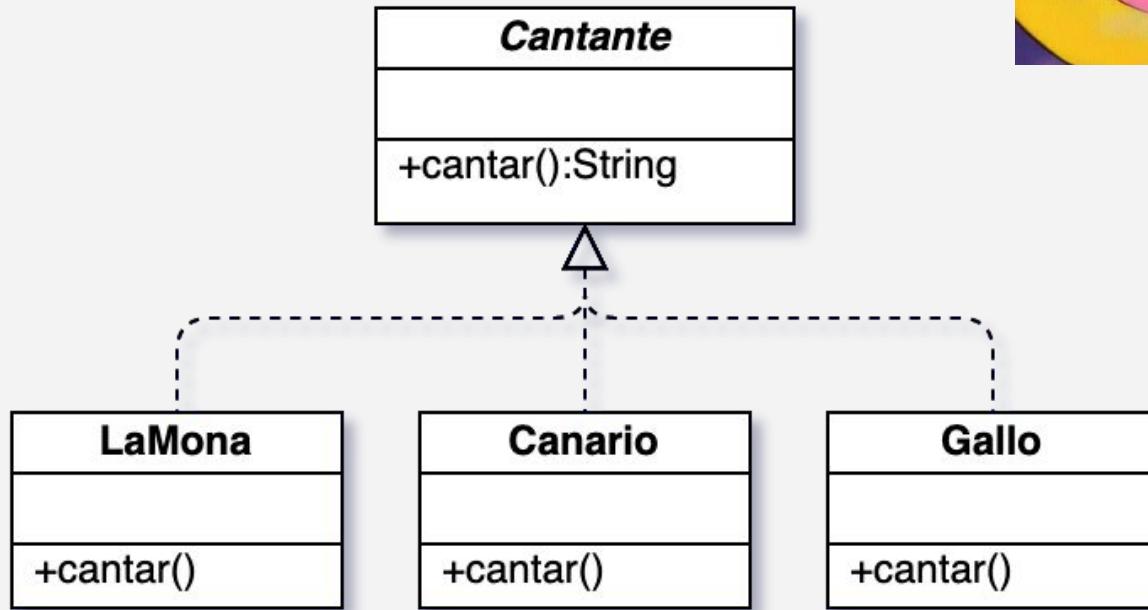


Entonces hasta ahora...



¿Te la pasaste
tirando “ifs”
sin pensar
polimórficamente?

Polimórficamente:



Código más orientado a objetos:

```
Collection<Cantante> cantantes;  
...  
cantantes.add(new LaMona());  
cantantes.add(new Canario());  
cantantes.add(new Gallo());  
...  
for (Cantante cantante : cantantes) {  
    cantante.cantar();  
}
```

Agregando nuevos cantantes:



...

```
cantantes.add(new BallenaYubarta());
```

```
cantantes.add(new PollitoPio());
```

...

```
for (Cantante cantante : cantantes) {  
    cantante.cantar();  
}
```



**¡BIEN
HECHO!**

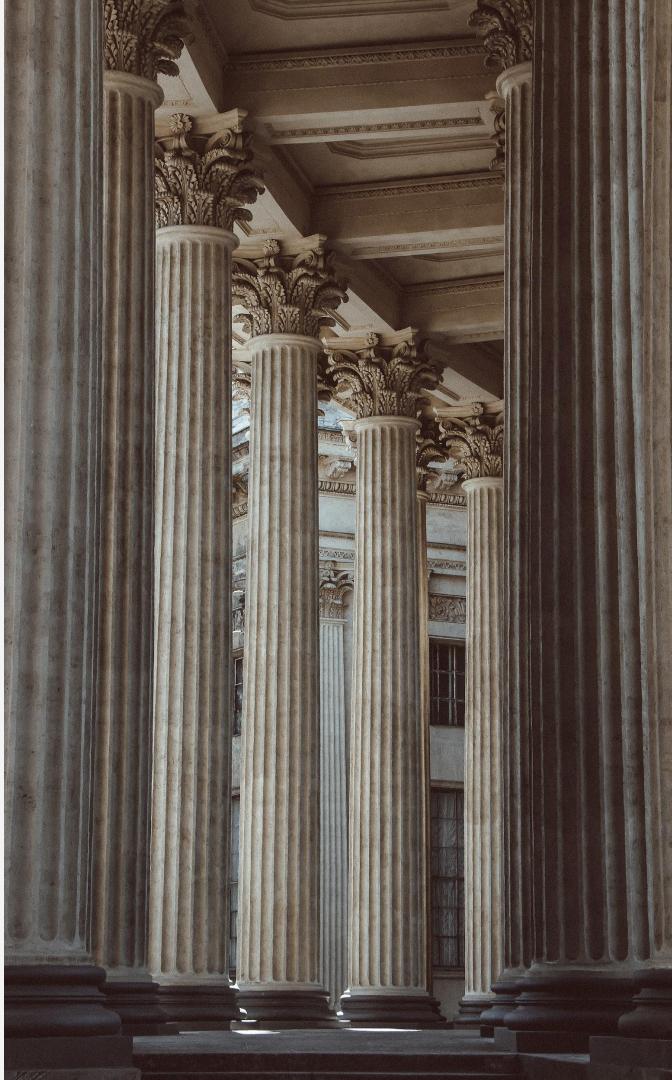
01 ABSTRACCION

02 ENCAPSULAMIENTO

03 POLIMORFISMO

04 DELEGACION

05 HERENCIA



PILARES DE POO



VENTAJAS



01

Definir una interfaz que reduce el acoplamiento, aumenta la reutilización y hace que el código sea más fácil de leer y mantener.

FAVORECE



02

"Ocultar detalles de implementación al interactuar con un grupo de clases diferentes a través de una interfaz común".

Esto significa que podemos tener una clase que interactúa con muchos objetos que heredan* de la misma clase madre a través de los métodos que **comparten** estos objetos. Dichos métodos se conocen como la **interfaz**.

**¿Podemos
ver otro
ejemplo?**



Enunciado

Un restaurante ofrece distintos tipos de platos según el día de la semana. De Lunes a Viernes ofrece pizzas y los fines de semana hay parrilla. Los mozos se acercan a la cocina y le pide que le marche el plato del día. Tenga en cuenta que el dueño del restaurante está pensando en ofrecer sushi los días Viernes.

¿Cómo se imagina el código de cocina que puede resolver esta situación?



Una Cocina

Dependiendo del tipo de cocinero que esté trabajando, la cocina “producirá” distintos platos:





```
if (cocinero == Cocinero.PIZZERO) {  
    Harina harina, Levadura levadura;  
    Masa masa = mezclarIngredientes(harina,levadura)  
    ...  
} else {  
    if (cocinero == Cocinero.ASADOR) {  
        prenderBrasas();  
        condimentar(unabondio);  
        ...  
    }  
}
```



```
cocinero.cocinar();
```

¡La clave es crear objetos!

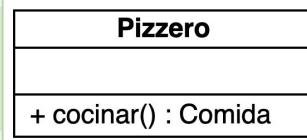


NUEVOS OBJETOS

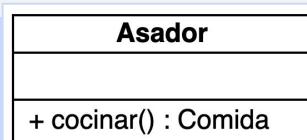
Paso 1: identificar y aislar comportamientos

Cocina.java

```
public Comida cocinar() {  
  
    if (cocinero == Cocinero.PIZZERO) {  
        Harina h, Levadura l;  
        Masa masa = mezclarIngredientes(h,l)  
        ...  
        ...  
    } else {  
  
        if (cocinero == Cocinero.ASADOR) {  
            prenderBrasas();  
            condimentar(unaBondio);  
            ...  
        }  
    }  
}
```



```
public Comida cocinar() {  
    Harina h, Levadura l;  
    Masa masa = mezclarIngredientes(h,l)  
    ...  
    return unaPizza;  
}
```



```
public Comida cocinar() {  
    prenderBrasas();  
    condimentar(unaBondio);  
    ...  
    return unaBondio  
}
```

NUEVOS OBJETOS Paso II: Utilizar los nuevos objetos

Cocina.java

```
public class Cocina {  
  
    private Cocinero cocinero;  
  
    public Comida cocinar() {  
        return this.cocinero.cocinar()  
    }  
  
    public void reemplazarCocinero(Cocinero nuevoCocinero) {  
        this.cocinero = nuevoCocinero;  
    }  
}
```

NUEVOS OBJETOS

Cocina.java

```
public class Cocina {  
    private Cocinero cocinero;  
  
    public Comida cocinar() {  
        return this.cocinero.cocinar();  
    }  
}
```

¿Y cómo funciona si no sabe qué instancia tiene...

```
public void reemplazarCocinero(Cocinero cocinero) {
```

Porque todas las instancias *entienden el mismo mensaje*.
Es lo que llamamos **POLIMORFISMO**

Clave
nunca
instancia
Así
nuevos
códigos





CASOS REALES

Estudiantes
que creían
entender:

POLIMORFISMO



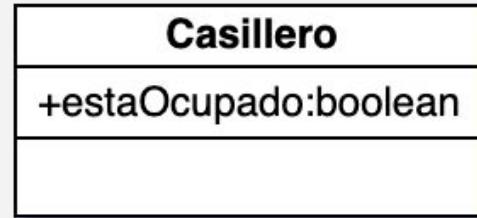
Caso real 1: Problema original



En una parte de un trabajo práctico los jugadores debían mover piezas entre casilleros. Cuando se movía una pieza, dependiendo si el casillero destino se encontraba libre u ocupado, se esperaban que pasaran diferentes cosas.

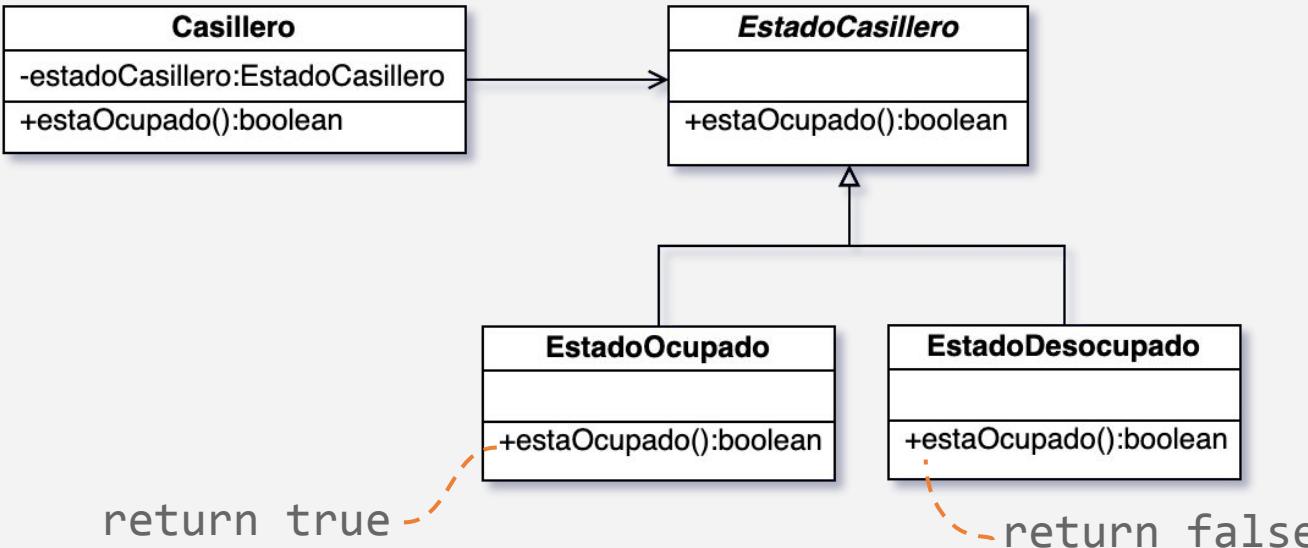
Caso real 1: Problema original

```
if (casillero.estaOcupado) {  
    // acción 1  
    // acción 2  
    // acción 3  
    ...  
} else {  
    // está vacío  
    // acción 4  
    // acción 5  
    ...  
}
```



Caso real 1: Solución “polimórfica” estudiantil

```
if (casillero.estaOcupado()) {  
    // acción 1  
    // acción 2  
    // acción 3  
    ...  
} else {  
    // está vacío  
    // acción 4  
    // acción 5  
    ...  
}
```





**NO LO SE RICK,
PARECE FALSO**

Caso real 1: De forma polimórfica:

...
casillero.recibir(unaPieza)
...

Casillero.java

```
public class Casillero {  
    private EstadoCasillero estado  
  
    public void recibir(Pieza pieza) {  
        this.estado.recibir(pieza);  
    }  
  
    public void ocupar() {  
        this.estado = new EstadoOcupado();  
    }  
}
```

EstadoOcupado.java

```
public class EstadoOcupado {  
  
    public void recibir(Pieza pieza) {  
        // acción 1  
        // acción 2  
        // acción 3  
    }  
}
```

EstadoDesocupado.java

```
public class EstadoDesocupado {  
  
    public void recibir(Pieza pieza) {  
        // está vacío  
        // acción 4  
        // acción 5  
    }  
}
```



1ra LECCIÓN APRENDIDA

Aplicar polimorfismo no es ***solamente*** crear nuevos objetos que entiendan el mismo mensaje, sino que ***además*** a través de ese proceso se debe ***desmantelar*** el problema original de forma tal que los distintos comportamientos queden realmente encapsulados en los nuevos objetos.

Caso real 2: La Salamandra

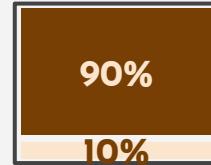
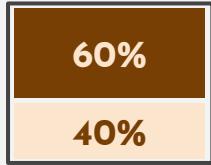
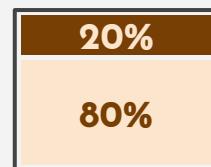
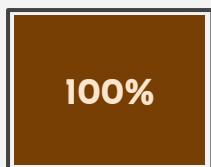
El cálculo del gasto \$ total de la salamandra será la suma de lo que gasta cada tipo de madera que tiene en su interior:

- Quebracho (cálculo X)
- Pino (cálculo Y)



Caso real 2: La Salamandra

Distintas composiciones de madera darán gastos distintos:



Referencias:

QUEBRACHO

PINO



Caso real 2: Solución “polimórfica”

Salamandra.java

```
public class Salamandra {  
  
    private Quebracho quebracho;  
    private Pino pino;  
  
    public int gastoTotal() {  
        int gasto = 0  
        gasto += this.quebracho.calcularGasto();  
        gasto += this.pino.calcularGasto();  
        return gasto;  
    }  
}
```

Quebracho.java

```
public class Quebracho {  
  
    private ...  
  
    public int calcularGasto() {  
        // cálculo X  
        return X * 2;  
    }  
}
```

Pino.java

```
public class Pino {  
  
    private ...  
  
    public int calcularGasto() {  
        // cálculo Y  
        return Y + 10;  
    }  
}
```

Caso real 2: Checklist de polimorfismo



-  Objetos distintos que entienden el mismo mensaje.
-  Los comportamientos se resuelven dentro de esos objetos.
-  Quién invoca a los objetos polimórficos no tiene conocimiento de quiénes son.

Caso real 2: Checklist de polimorfismo

Salamandra.java

```
public class Salamandra {  
  
    private Quebracho quebracho;  
    private Pino pino;  
  
    public int gastoTotal() {  
        int gasto = 0  
        gasto += this.quebracho.calcularGasto();  
        gasto += this.pino.calcularGasto();  
        return gasto;  
    }  
}
```

Quebracho.java

```
public class Quebracho {  
    private ...  
    public int calcularGasto() {  
        // cálculo X  
        return X * 2;   
    }  
}
```

Pino.java

```
public class Pino {  
    private ...  
    public int calcularGasto() {  
        // cálculo Y  
        return Y + 10;   
    }  
}
```

Caso real 2: Checklist de polimorfismo

Salamandra.java

```
public class Salamandra {  
  
    private Collection<Madera> maderas;  
  
    public int gastoTotal() {  
        int gasto = 0  
  
        for (Madera madera : maderas) {  
            gasto += madera.calcularGasto();  
        }  
        return gasto;  
    }  
}
```

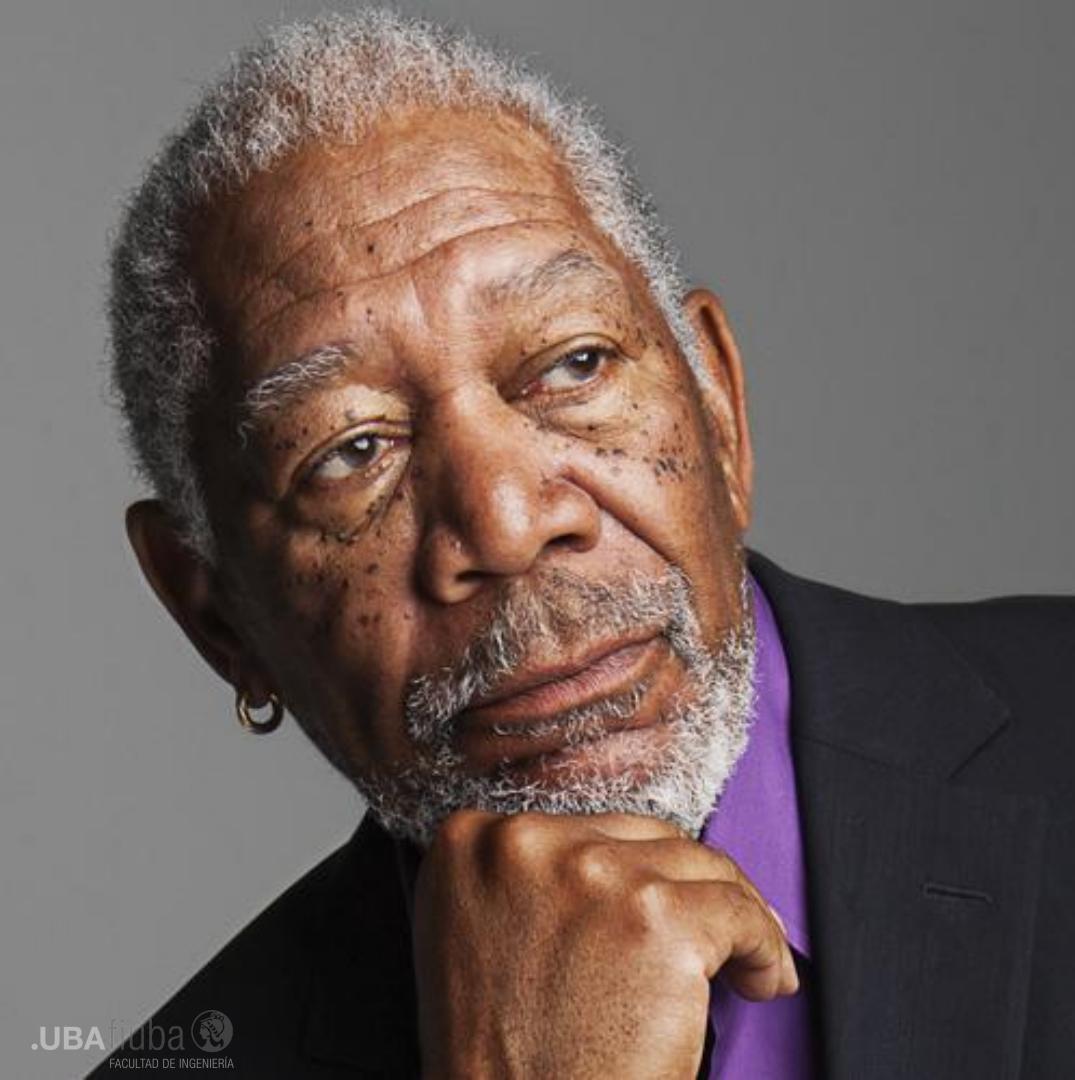


Quebracho.java

```
public class Quebracho extends Madera {  
  
    private ...  
  
    public int calcularGasto() {  
        // cálculo X  
        return X * 2;   
    }  
}
```

Pino.java

```
public class Pino extends Madera {  
  
    private ...  
  
    public int calcularGasto() {  
        // cálculo Y  
        return Y + 10;   
    }  
}
```

A close-up portrait of Morgan Freeman. He has grey hair and a beard, and is wearing a dark suit jacket over a purple shirt. He is looking slightly to his left with a thoughtful expression.

2da LECCIÓN APRENDIDA

Aplicar polimorfismo implica que los **comportamientos** que finalmente se llevan a cabo deben **resolverse en tiempo de ejecución**. Para ello es necesario que quien invoca al comportamiento polimórfico lo haga contra una abstracción / interfaz y **no contra una clase concreta**.



¿ALGUNA
PREGUNTA?

A woman with dark curly hair, wearing a bright red long-sleeved dress with three gold buttons down the front, is singing into a black microphone. She has her eyes closed and her mouth wide open in a powerful vocal performance. Her arms are raised high above her head. The background is a blurred indoor setting with warm lighting.

¡GRACIAS!