DuocUC® INFORMÁTICA Y TELECOMUNICACIONES

EXPERIENCIA DE APRENDIZAJE 2
HERENCIA Y COLECCIONES

Herencia

DSY1102-Desarrollo Orientado a Objetos

CONTENIDO

DuocUC®

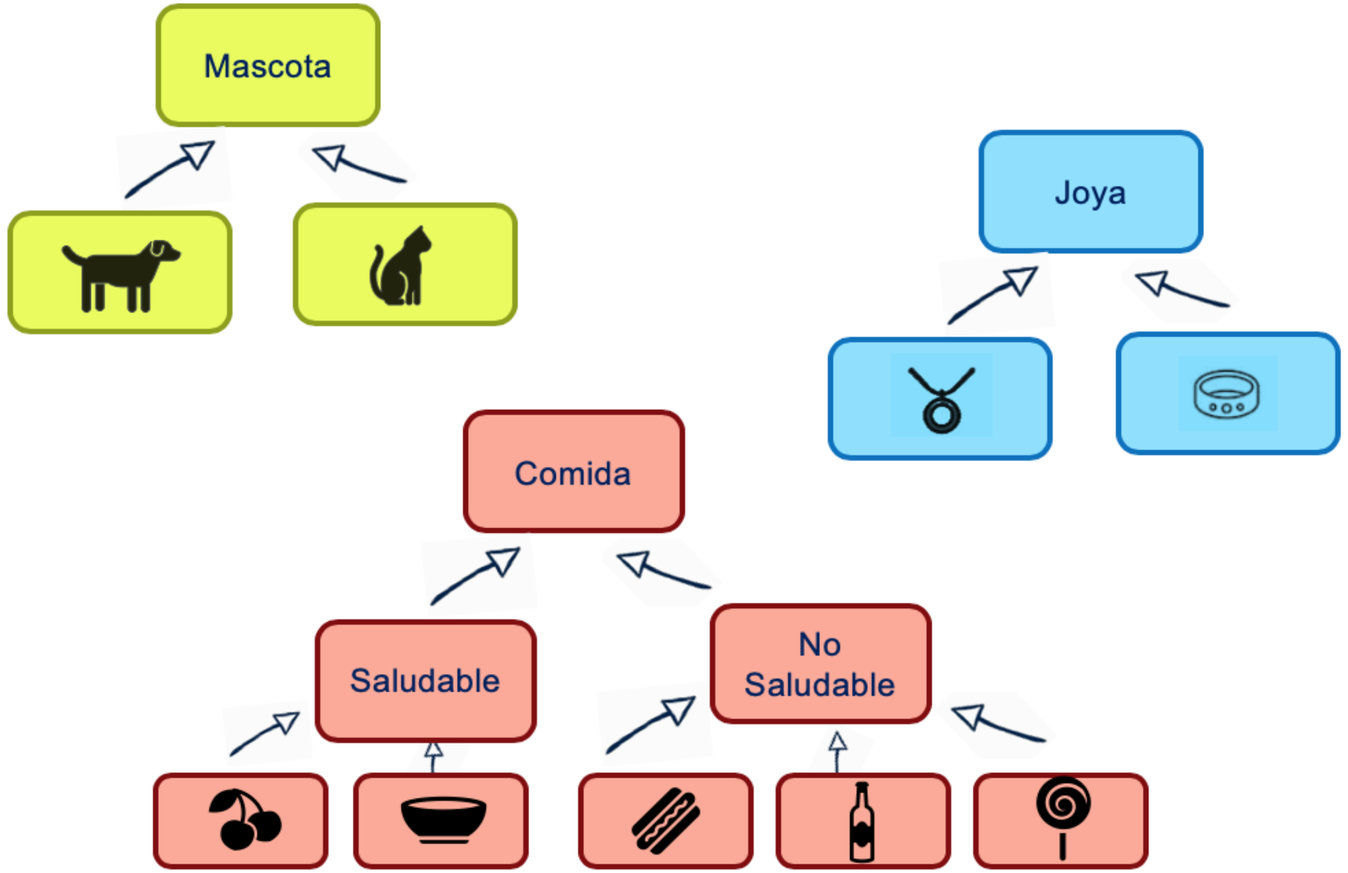
01

HERENCIA

DuocUC®

Herencia

Observa las siguientes imágenes…

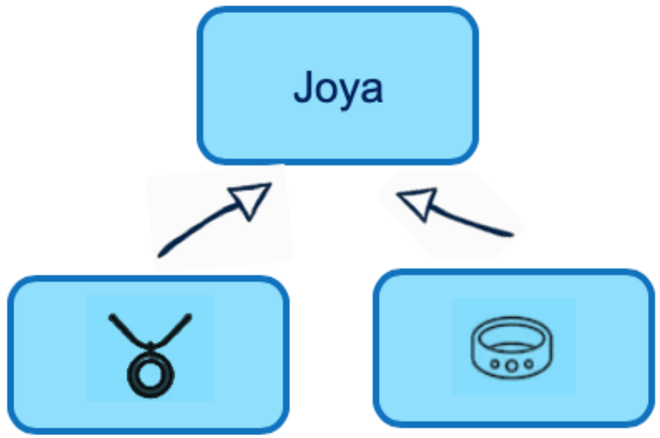


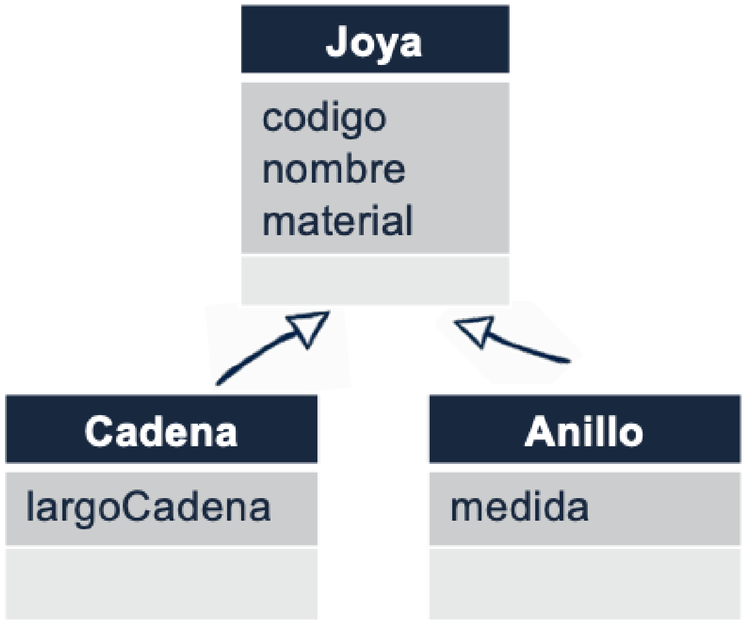
¿Qué crees que es herencia?

DuocUC®

Herencia

Es la capacidad de **crear clases** que **adquieran** de manera automática los **atributos y métodos** de otras clases existentes, logrando al mismo tiempo añadir atributos y métodos propios.



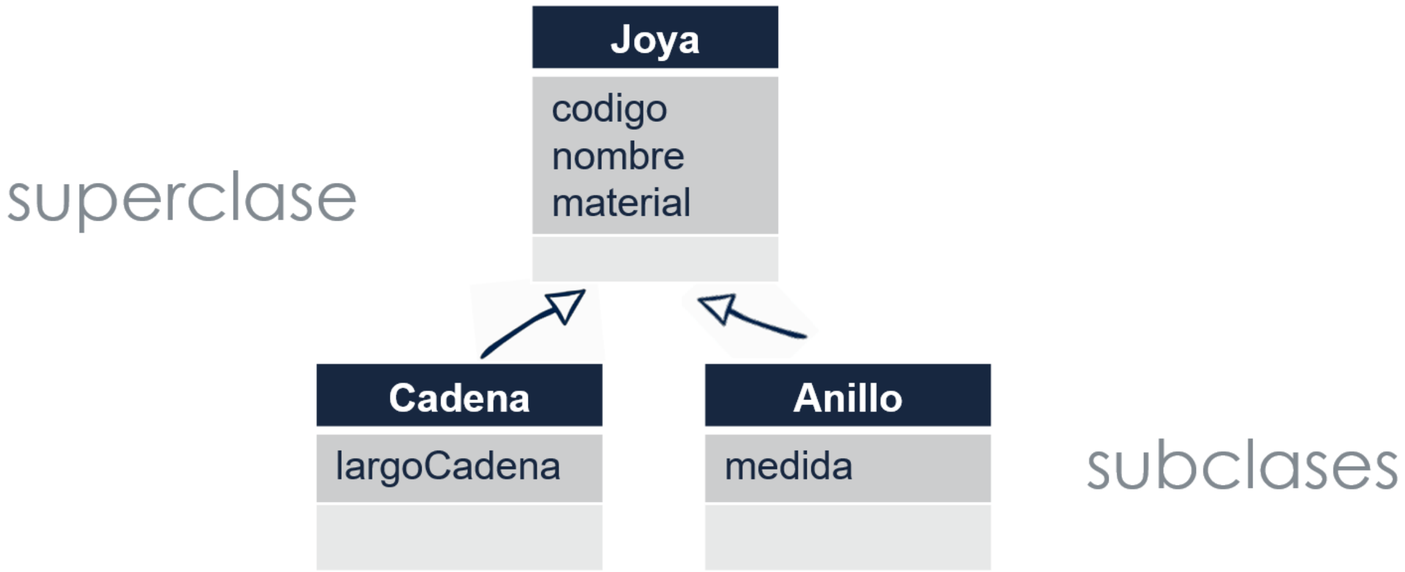


En este ejemplo, **Joya** tiene los atributos **codigo, nombre y material**. **Cadena**, hereda los atributos de Joya, por lo que tiene **codigo, nombre, material y largoCadena**. **Anillo**, tiene **codigo, nombre, material y medida**.

DuocUC®

Herencia

La clase creada se llama **subclase** y la clase existente **superclase**.



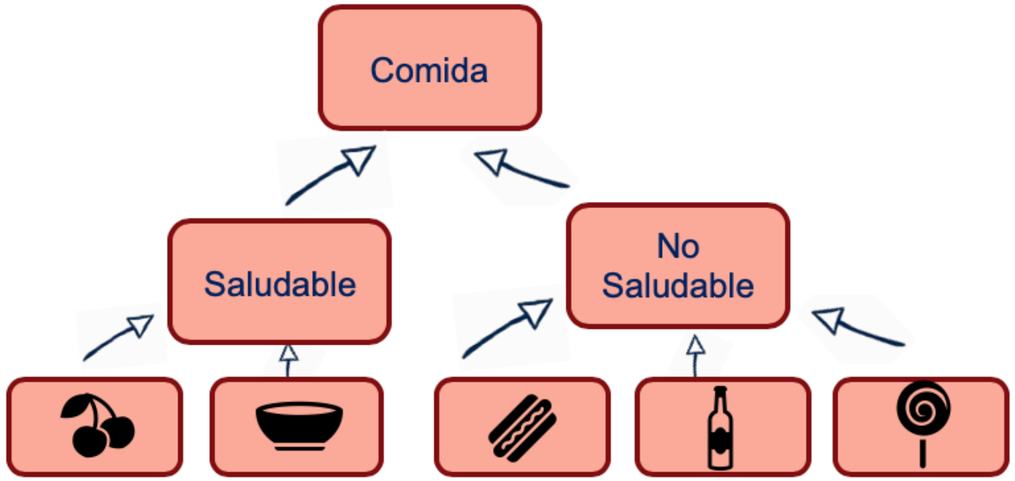
La herencia permite la especialización, es decir, extender las características de un objeto a otro más particular, permitiéndole heredar las características del objeto padre.

DuocUC®

Herencia

Con la herencia, todas las clases están clasificadas en una jerarquía estricta. Cada clase tiene su superclase (la clase superior en su jerarquía) y cada clase puede tener una o más subclases (las clases inferiores en su jerarquía).

¿Cuáles serían las superclases y subclases?



Comida es superclase de Saludable y NoSaludable. Saludable es subclase de Comida y superclase de Fruta y Sopa.

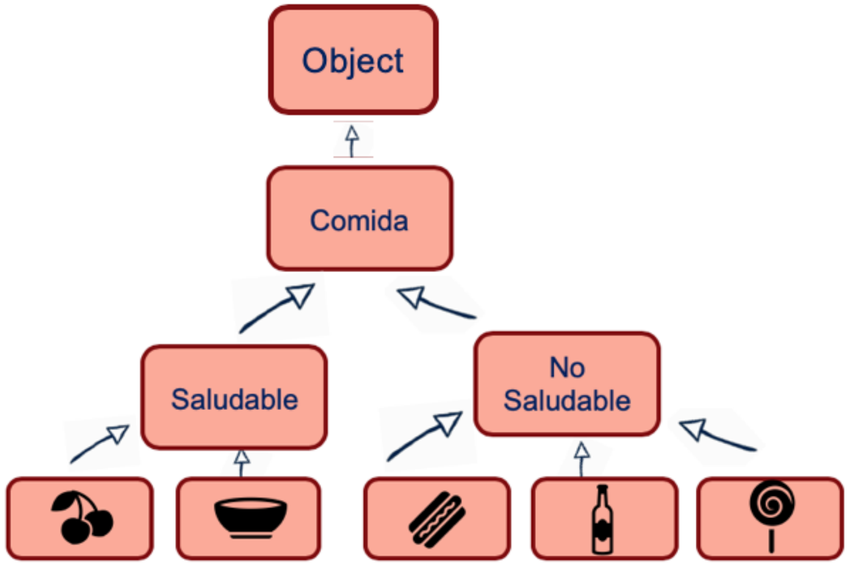
y… ¿NoSaludable?

Cada subclase puede convertirse en la superclase de futuras subclases.

DuocUC®

Herencia

En Java, la jerarquía de clases empieza con la clase **Object** (del paquete java.lang), a partir de la cual se extienden (o “heredan”) todas las clases en Java, ya sea en forma directa o indirecta.



Entonces, Comida es superclase de Saludable y NoSaludable y además es subclase de Object.

Revisa los métodos de la clase Object:

https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

DuocUC®

Herencia

¿Cuál es el objetivo de crear estas clases?

Reutilizar código

Ahorrar tiempo en programar

Las **superclases** tienden a ser "**más generales**" y las **subclases** "**más específicas**"

Con la herencia, **los atributos y los métodos comunes** de todas las clases en la jerarquía **se declaran en una superclase**. Cuando se requieren modificaciones para estas características comunes, los desarrolladores de software sólo necesitan realizar las modificaciones en la superclase; así **las clases derivadas heredan los cambios**. Sin la herencia, habría que modificar todos los archivos de código fuente que contengan una copia del código en cuestión.

Herencia

Una **subclase** se crea igual que una clase normal, agregando la clase que se está extendiendo. Se utiliza la palabra **extends**:

```
public class Saludable extends Comida{
//cuerpo de la subclase
}

public class NoSaludable extends Comida{
//cuerpo de la subclase
}
```

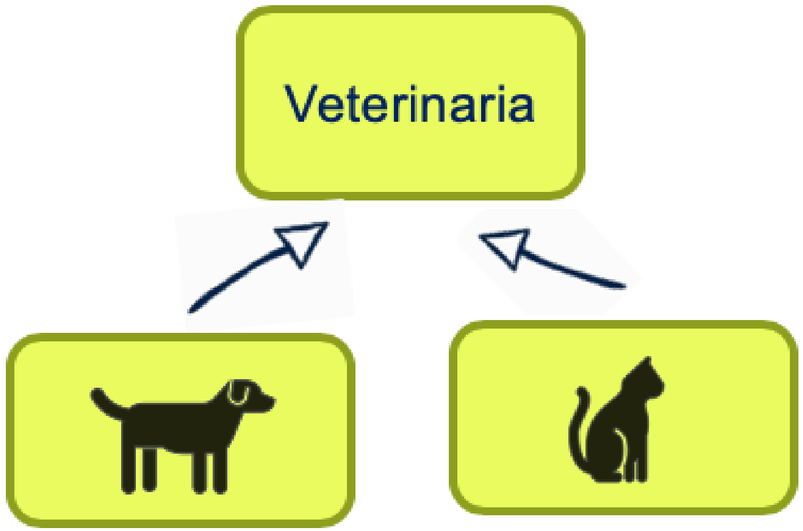
Saludable **“es una”** Comida.
NoSaludable **“es una”** Comida.

La herencia entre dos clases establece una relación entre las mismas de tipo “**es un**”.

DuocUC®

Herencia

Entonces, ¿es correcta la definición de herencia en la imagen?



¿El Perro “**es una**” veterinaria?

¿Hay que hacer alguna modificación?

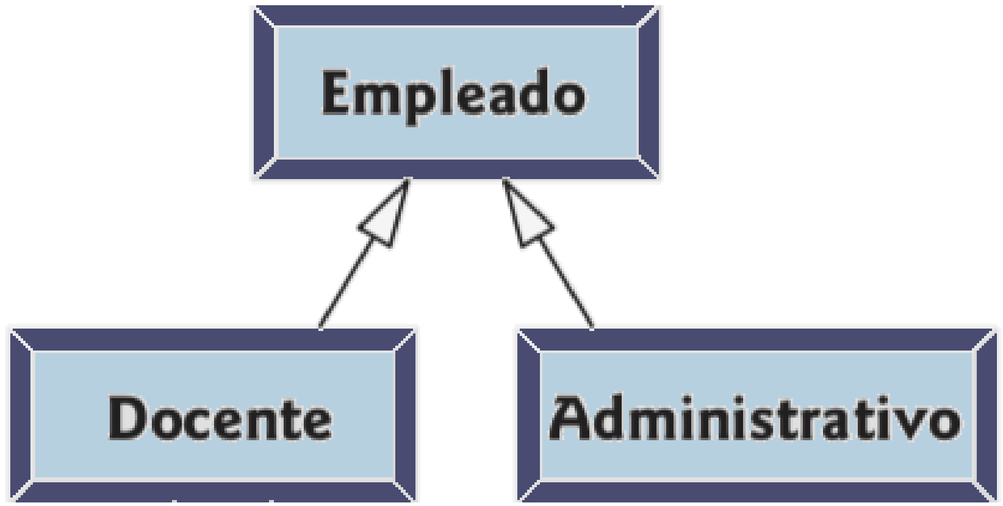
DuocUC®

Herencia

Relación “es un” y “tiene un”

La relación “**es un**” representa a la herencia. En este tipo de relación, un objeto de una subclase puede tratarse también como un objeto de su superclase. Por ejemplo, un docente es un Empleado.

La relación “**tiene un**” identifica a la composición. En este tipo de relación, un objeto contiene referencias a objetos como atributos. Por ejemplo, un docente tiene un cargo (atributo).

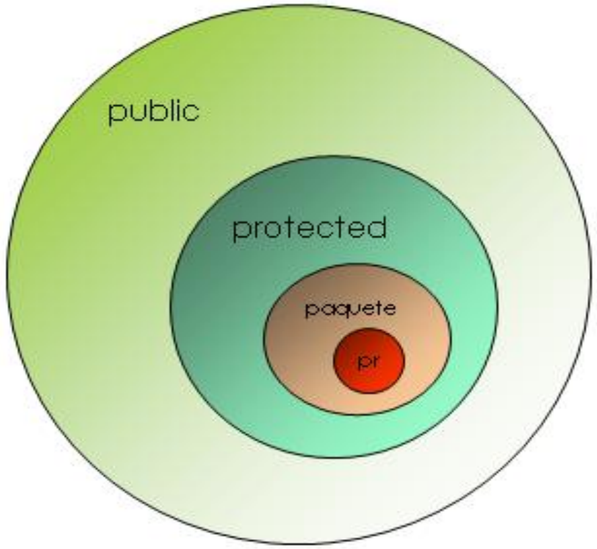


DuocUC®

Herencia

Modificador de acceso

public
protected
private



Los miembros **public** de una clase son accesibles en cualquier parte del programa.

Los miembros **private** de una clase son accesibles sólo dentro de la misma clase. Los miembros private de una superclase no son heredados por sus subclases.

DuocUC®

Herencia

Sobreescribir **@override** Volver a escribir

Significa **reescribir** en la subclase el método heredado de la superclase. La subclase puede modificar atributos y métodos de la superclase.

Sobrescribir un método significa volver a declararlo en la subclase:

- Con el mismo tipo de **retorno** (o subtipo)
- Con el mismo **nombre**
- Con la misma lista de **parámetros**

Si se declara de diferente manera, será un método distinto (sobrecargado).

Herencia

Sobreescribir **@override**

```
public class Comida{
        ✓   ○   ➢
        public void cocinar(int grados){
                temperatura = grados;
        }
}

public class NoSaludable extends Comida{
        ✓   ○   ➢
        public void cocinar(int grados){
                freir(grados);
        }
}
```

superclase

subclase

✓ mismo tipo de retorno
○ mismo nombre
➢ mismos parámetros

Sobreescribir

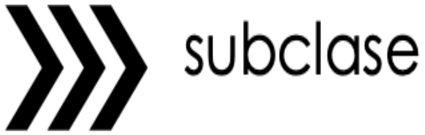
Herencia

El método cocinar de la superclase Comida, es sobrescrito en la subclase NoSaludable.

```
public class Comida{
        public void cocinar(int grados){
                temperatura = grados;
        }
}
```

✓ mismo tipo de retorno: void
○ mismo nombre: cocinar
➢ mismos parámetros: int grados

```
public class NoSaludable extends Comida{
        public void cocinar(int grados){
                freir(grados);
        }
}
```



La sobreescritura siempre se implementa en una subclase, reescribiendo el método de la superclase

DuocUC®

Herencia

¿Por qué sobreescribir un método?

Depende de lo que queremos hacer. Si quiero agregar algo más a un método que sea específico para la clase hija, no podría hacerlo sólo heredando, ya que tendría que modificarlo en la clase padre y dicha funcionalidad sería común para todas las clases hijas que hereden de la superclase.

Por la razón anterior, la sobreescritura nos permite extender la funcionalidad de un método heredado para hacerlo especifico a lo que necesitemos, pudiendo implementar la lógica que queramos en nuestras clases hijas para el mismo método.

Una clase padre podría heredar un método a todas sus clases hijas menos a una que tenga un comportamiento específico.

Herencia

Sobrecargar Métodos distintos

Un método sobrecargado se utiliza para **reutilizar el nombre** de un método pero con diferentes parámetros (opcionalmente un tipo diferente de retorno).

Las reglas para sobrecargar un método son:

- Los métodos sobrecargados **deben cambiar la lista de parámetros**.
- Pueden cambiar el tipo de retorno (opcional).
- Pueden cambiar el modificador de acceso (opcional)..
- Un método puede ser sobrecargado en la misma clase o en una subclase.

Herencia

Sobrecargar

```
public class Motor{
                ✓       ○       ➢
        public void encender(char tipoMotor){
            if(tipoMotor == 'E'){
                    prenderElectrico();
            }else{
                    prenderCombustion();
             }
        }

                ✓       ○       ➢
        public String encender(){
                    return "ON"
        }
}
```

superclase

- ✓ tipo de retorno puede cambiar o no
- ○ mismo nombre
- ➢ distintos parámetros

DuocUC®

Sobreescribir

Herencia

El método encender de la clase Motor, es sobrecargado en la misma clase.

```
public class Motor{
                    ✓       ○       ➢
        public void encender(char tipoMotor){
            if(tipoMotor == 'E'){
                    prenderElectrico();
            }else{
                    prenderCombustion();
             }
        }

                    ✓       ○       ➢
        public String encender(){
                    return "ON"
        }
}
```

- ✓**tipo de retorno puede o no cambiar: void y String**
- ○ **mismo nombre: encender**
- ➢**distintos parámetros: char tipoMotor y ninguno**

La sobrecarga se puede implementar en la superclase o subclase

DuocUC®

Herencia

¿Por qué sobrecargar un método?

Permite declarar métodos que se llamen igual pero que reciban parámetros diferentes (no pueden haber 2 métodos con el mismo nombre y los mismos parámetros).

En el ejemplo, podemos ver que los 2 métodos se llaman igual pero poseen parámetros diferentes, así cuando sean llamados, dependiendo del parámetro enviado se accede al método.

¿Cómo se invocaría el primer método? ¿y el segundo método?

Herencia

Llamando a la superclase

Para acceder a métodos o atributos de la superclase, se utiliza **super**, el cual es una referencia hacia la clase padre.

Se puede invocar al constructor de la clase padre, a métodos o atributos, siempre que el modificador de acceso lo permita.

super.atributo;
super.método();

Llamando a atributos y métodos del padre

this.atributo:
this.método();

Llamando a atributos y métodos de la misma clase

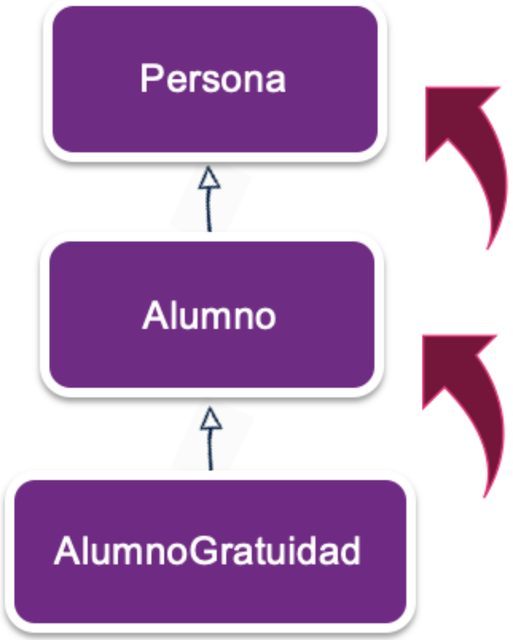
DuocUC®

02 CONSTRUCTOR EN LA HERENCIA

DuocUC®

Constructor en la Herencia

Los constructores no se heredan. De hecho, la primera tarea del constructor de cualquier subclase es llamar al constructor de su superclase directa, ya sea en forma explícita o implícita (si no se especifica una llamada al constructor), para asegurar que las variables de instancia heredadas de la superclase se inicialicen en forma apropiada.



1 Constructor Persona
2 Constructor Alumno
3 Constructor AlumnoGratuidad

DuocUC®

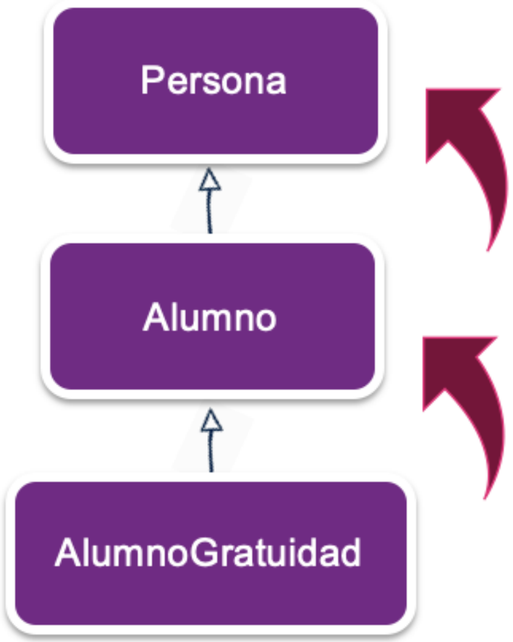
Constructor en la Herencia

Considere el siguiente código:

```
public class Persona{
    public Persona(){
        System.out.println("Soy Persona")
}
```

```
public class Alumno extends Persona{
    public Alumno(){
        System.out.println("Soy Alumno")
}
```

```
public class AlumnoGratuidad extends Alumno{
    public AlumnoGratuidad(){
        System.out.println("Soy Alumno con Gratuidad")
}
```



¿Qué arroja el siguiente código?

AlumnoGratuidad ag = new AlumnoGratuidad();

DuocUC®

Constructor en la Herencia

- Soy Persona
- Soy Alumno
- Soy Alumno con Gratuidad

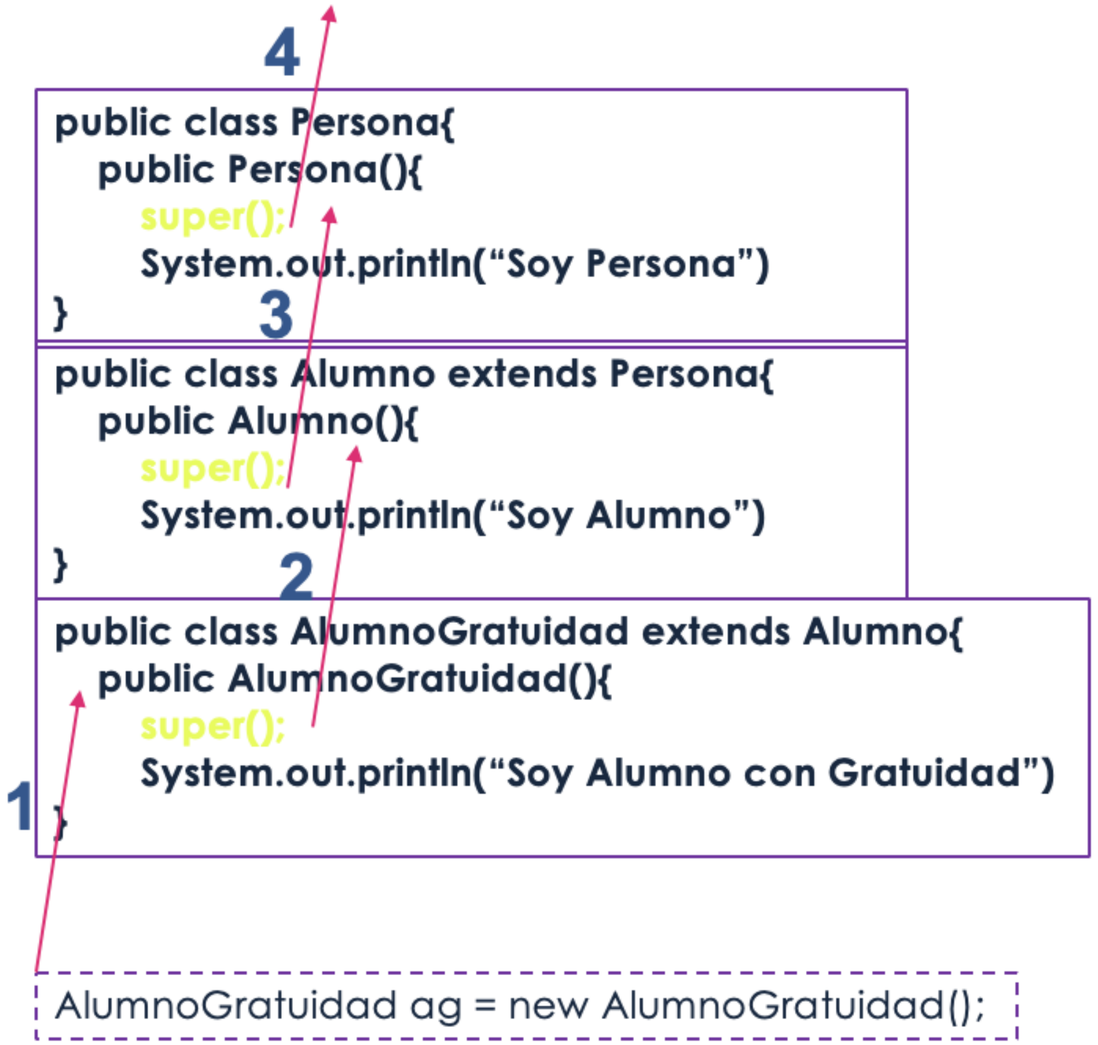
¿por qué?

Porque el compilador Java añade como primera línea de código en todos los constructores de una clase, la instrucción:

```
super();
```

Esta instrucción provoca una llamada al constructor sin parámetros de la superclase.

Constructor en la Herencia



¿Qué hemos aprendido?

✓Reconocer qué es herencia.
✓Utilizar una estructura de herencia: superclases y subclases.
✓Implementar atributos y métodos en la herencia.
✓Utilizar modificadores de acceso en la herencia.
✓Implementar la sobreescritura y sobrecarga de los métodos.

¿Qué te resultó difícil entender?

DuocUC®