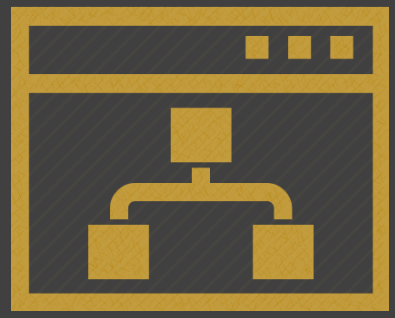
Herencia en Java



Profesor: Andrés Guzmán F

¿Qué es la herencia?

Es una relación de parentesco entre dos clases, una es padre de la otra, una es hija de la otra!

Características

Es el mecanismo de la programación orientada a objetos para implementar relaciones de jerarquía de clases

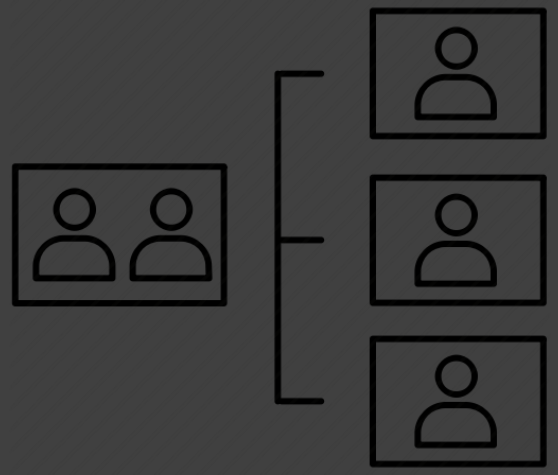
Una subclase hereda el estado y el comportamiento de todos sus ancestros

Es una de las bases de reutilización de código y polimorfismo

También es un principio fundamental de la POO

Parentesco entre clases

No puede existir herencia si no existe alguna relación familiar entre ambas clases!



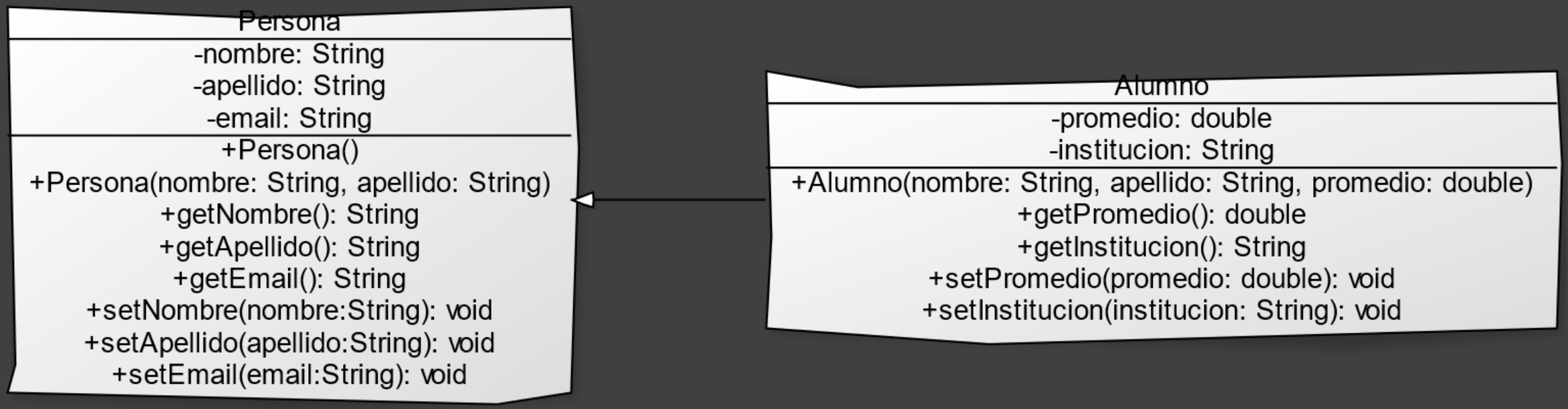
Ejemplos de herencia

```
class Persona {...}
class Profesor extends Persona {...}
class Director extends Persona {...}
class Alumno extends Persona {...}
class AlumnoInternacional extends Alumno {...}


class FigurasGeometricas {...}
class Cuadrado extends FigurasGeometricas {...}
class Triangulo extends FigurasGeometricas {...}
```

El término superclase se refiere a la clase que es el ancestro más directo, así como a todas las clases ascendentes

Relación de Generalización



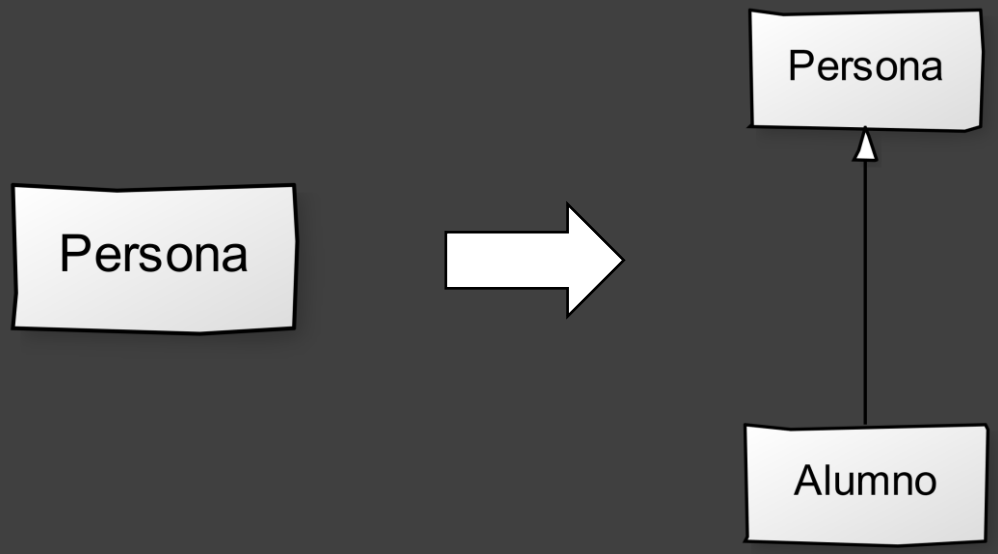
En UML es una flecha continua que va desde la clase hija hasta la clase padre, similar a la relación de asociación, pero con la diferencia que termina con una punta en forma de triángulo

Hay dos formas de herencia

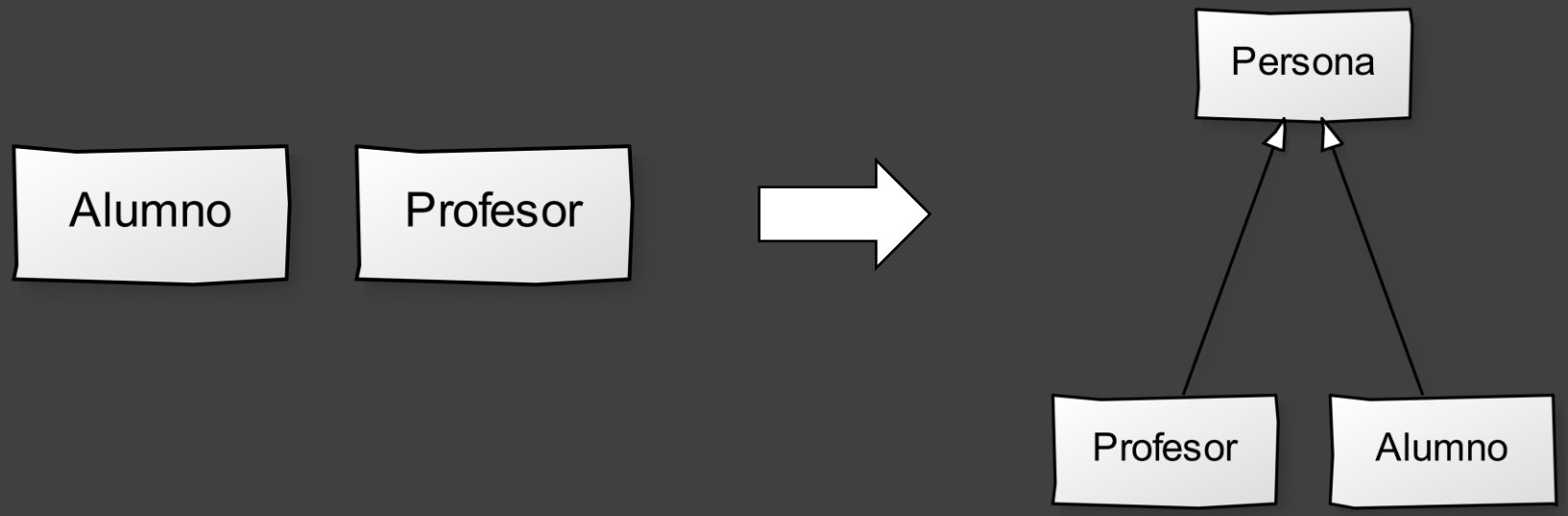
Hay dos formas distintas para el proceso de creación de jerarquías de clases o herencia:

Generalización	Especialización

Especialización



Generalización



Constructor y la palabra super

```
public class Persona {
    private String nombre;
    private String apellido;

    public Persona(String nombre, String apellido) {
        this.nombre = nombre;
        this.apellido = apellido;
    }
}
```

Constructor y la palabra super

```java
public class Persona {
    private String nombre;
    private String apellido;

    public Persona(String nombre, String apellido) {
        this.nombre = nombre;
        this.apellido = apellido;
    }
}
```

```java
public class Alumno extends Persona{
    private double promedio;

    public Alumno(String nombre, String apellido, double promedio) {
        super(nombre, apellido);
        this.promedio = promedio;
    }
}
```

```java
public class EjemploHerencia {
    public static void main(String[] args) {

        Alumno a = new Alumno("Andres", "Guzman", 6.5);
    }
}
```

Sobreescritura de métodos

- Otra característica de la herencia es poder sobre escribir un método que heredamos del padre, se puede hacer redefiniendo en la case hija un método con el mismo nombre:

```
public class Persona {
 [ ... ]
    public void metodoDelPadre() {
       // hacer algo importante
    }
}
```

Sobreescritura de métodos

- Otra característica de la herencia es poder sobre escribir un método que heredamos del padre, se puede hacer redefiniendo en la case hija un método con el mismo nombre:

```
public class Persona {
 [ ... ]
    public void metodoDelPadre() {
       // hacer algo importante
    }
}
```

```
public class Alumno extends Persona {

 [ ... ]
    @Override
    public void metodoDelPadre() {
       [ ... ]
    }
}
```

Sobreescritura de métodos

- Si quisiéramos usar el comportamiento original y solo modificar parte en la clase hija, deberíamos hacer lo mismo que vimos anteriormente en el constructor, invocando desde super:

```
public class Alumno extends Persona {

[ ... ]
    @Override
    public void metodoDelPadre() {
        super.metodoDelPadre();
        [ ... ]
    }
}
```

Sobreescritura toString()

- Si quisiéramos reusar el toString() del padre lo invocamos con super:

```
public class Persona {
 [ ... ]
    @Override
    public String toString() {
        return this.nombre + " " + this.apellido;
    }
}
```

Sobreescritura toString()

- Si quisiéramos reusar el toString() del padre lo invocamos con super:

```
public class Persona {
 [ ... ]
    @Override
    public String toString() {
       return this.nombre + " " + this.apellido;
    }
}
```

```
public class Alumno extends Persona {
 [ ... ]
    @Override
    public String toString() {
       return super.toString() + ", promedio: " + this.promedio;
    }
}
```

Restringir la herencia y sobreescritura de métodos

- Para impedir la herencia usamos la palabra final:

Restringir la herencia y sobreescritura de métodos

- Para impedir la herencia usamos la palabra final:

```
final public class Persona {
    [ ... ]
}
```

Restringir la herencia y sobreescritura de métodos

- Para impedir la herencia usamos la palabra final:

```
final public class Persona {
    [ ... ]
}
```

- Impedir la sobreescritura de un método que pueda ser heredado:

Restringir la herencia y sobreescritura de métodos

- Para impedir la herencia usamos la palabra final:

```
final public class Persona {
    [ ... ]
}
```

- Impedir la sobreescritura de un método que pueda ser heredado:

```
public class Persona {
[ ... ]
    final public void metodoDelPadre() {
        // hacer algo importante
    }
}
```