

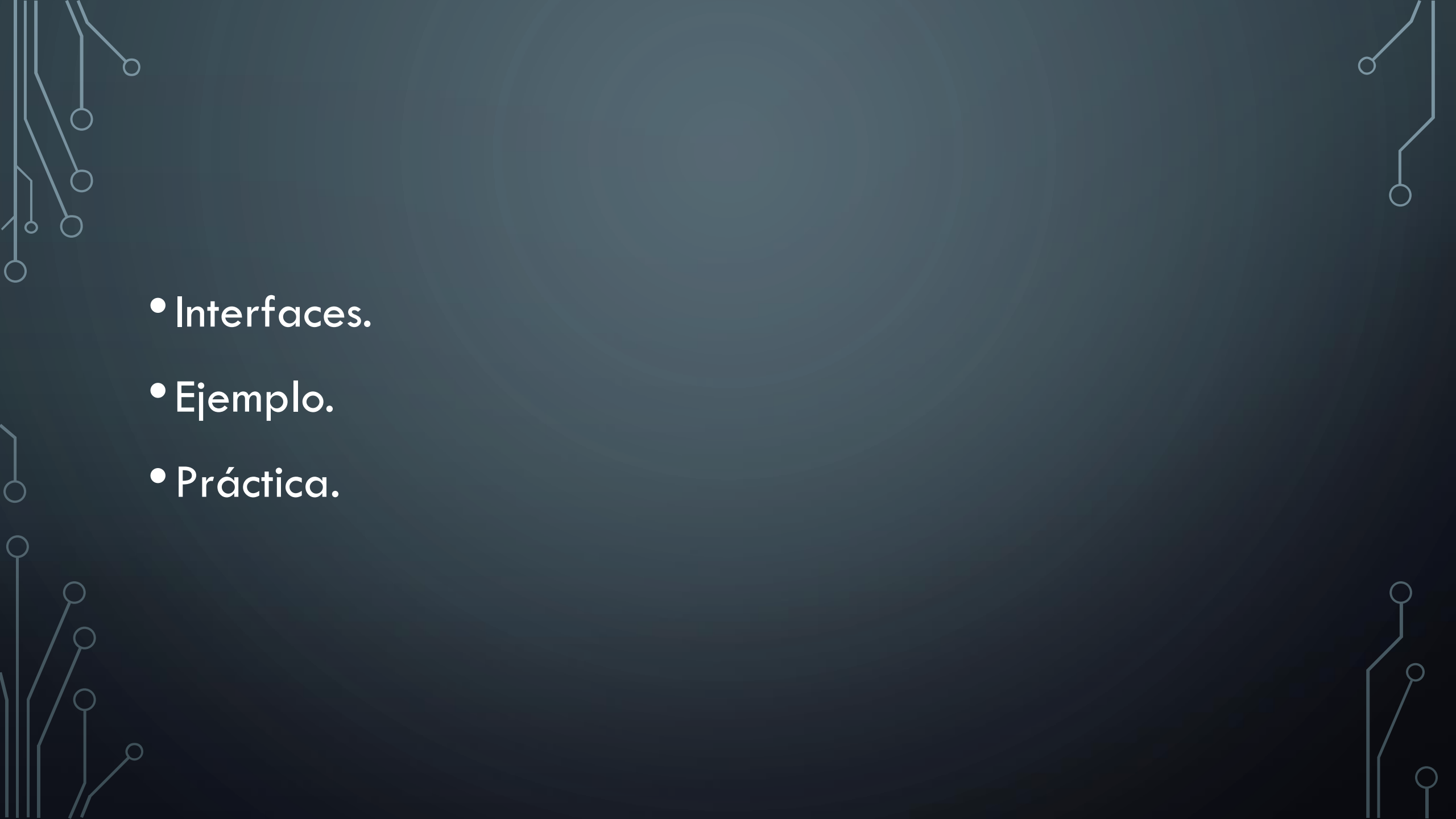


HERENCIA EN JAVA

CAMILO ANDRÉS URREGO ACOSTA

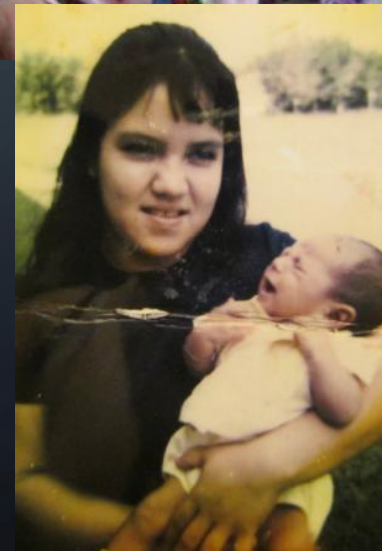
AGENDA

- ¿Qué es herencia?
- Extensión sobre una clase.
- Sobre escribir Atributos.
- Sobre Escribir métodos.
- La palabra reservada Super.
- Clases Abstractas.

- 
- The background is a dark blue gradient with a large, faint, light blue circle in the center. The corners are decorated with white circuit-like lines and small circles, resembling a PCB layout.
- Interfaces.
 - Ejemplo.
 - Práctica.

¿QUÉ ES HERENCIA?

- Utilizando la herencia, se puede derivar una nueva clase a partir de una antigua, y la nueva heredará todos los métodos y miembros de datos de la antigua. La clase nueva se llama **clase derivada** y la clase original **clase base**.



EXTENSIÓN SOBRE UNA CLASE

- Java sólo admite herencia simple. Una clase no puede tener más de una clase padre.
- Supongamos que queremos ampliar la definición de una Persona para que contenga datos de su lugar de residencia, como son la Provincia y la Población y también la Edad, y llamemos a esta nueva clase Ciudadano.

```
1 public class Ciudadano extends Persona {  
2     // Definición de la nueva clase extendida  
3 }
```

```
1 package tipos;  
2  
3 public class Ciudadano extends Persona {  
4     private String poblacion;  
5     private String provincia;  
6     private int edad;  
7  
8     public Ciudadano() {  
9         super();  
10        iniciaAtributos();  
11    }  
12  
13    @Override  
14    protected void iniciaAtributos () {  
15        setNombre("Un nombre");  
16        edad = 0;  
17    }  
18  
19    public String getPoblacion() {  
20        return poblacion;  
21    }  
22
```

```
1 Ciudadano ciudadano = new Ciudadano("José", "García", "555 123 456", "  
    Alcorcón", "Madrid", 40;  
2 System.out.println("Nombre: " + ciudadano.getNombre());
```

```
1 Ciudadano ciudadano = new Ciudadano();  
2 Persona persona = ciudadano; // Perfectamente válido.  
3 persona.getNombre(); // No hay problema, getNombre() está definido en  
  Persona.  
4 persona.getEdad(); // Error!!!, getEdad() está definido en Ciudadano.
```

```
1 Persona persona = new Persona();  
2 Ciudadano ciudadano = persona; // Error!!!
```

Una referencia de una clase padre admite una referencia a cualquiera de sus clase hijas, pero nunca al contrario.

SOBREESCRIBIR ATRIBUTOS

- En algunas circunstancias, podemos vernos en la necesidad de definir un atributo en una clase hija con el mismo nombre que en su clase padre, como muestra el siguiente código de ejemplo:

```
1 // Esta es la clase padre
2 public class Distancia {
3     float distancia;
4
5     public Distancia() {
6         distancia = 0;
7     }
8
9     public Distancia(float distancia) {
10         this.distancia = distancia;
11     }
12 // Sigue la definición de esta clase.
```



```
13 }  
14  
15 // Esta es la clase hija  
16 public class DistanciaDoblePrecision extends Distancia {  
17     // Este es el atributo sobrescrito  
18     double distancia;  
19  
20     public DistanciaDoblePrecision() {  
21         distancia = 0;  
22     }  
23  
24     public DistanciaDoblePrecision(double distancia) {  
25         this.distancia = distancia;  
26     }  
27     // Sigue la definición de esta clase.  
28 }
```

Cuando una clase hija sobrescribe algún atributo de su clase padre, el atributo de la clase padre queda oculto , de modo que si aparece el nombre del atributo en la clase hija se utilizará el atributo definido en esta clase y no el definido en la clase padre.

SOBREESCRIBIR MÉTODOS

- Para que una clase hija sobrescriba un método de su clase padre es necesario que ambos métodos tengan la misma signatura y el mismo tipo de retorno, de lo contrario no se sobrescribe el método.

```
8 public DistanciaDoblePrecision(double distancia) {  
9     this.distancia = distancia;  
10 }  
11  
12 @Override  
13 void incrementaDistancia(float incremento) {  
14     distancia += incremento;  
15 }  
16 // Sigue la definición de esta clase.  
17 }
```

LA PALABRA RESERVADA SUPER

- Existen casos en los que, desde una clase hija, nos interesa acceder a los métodos o atributos sobrescritos en la clase padre. Si escribimos en la clase hija simplemente el nombre del atributo o del método estaremos haciendo uso de la de acción dada para ellos en la clase hija. ¿Cómo accedemos a los miembros sobrescritos desde la clase hija?. La respuesta es haciendo uso de la palabra reservada super.



- La palabra reservada **super** es una referencia a la clase padre, del mismo modo que la palabra reservada **this** es una referencia a la propia clase.

CLASES ABSTRACTAS

- A veces es útil simplemente declarar los métodos en una clase padre, sin dar ninguna implementación para ellos, y delegar la implementación a las clases hijas que la extiendan. Esta técnica usa el concepto de polimorfismo.

- Si queremos indicar que no vamos a dar ninguna implementación para algún método declarado en la clase, debemos modificarlo con la palabra reservada **abstract**, con la restricción de que si una clase que contiene un método **abstract** ésta también debe ser **abstract**, también podemos declarar una clase como **abstract** sin que ninguno de sus métodos lo sea.
- Si una clase es declarada como **abstract** no se podrán crear instancias de la misma.

```
1 public abstract class Figura {
2     public abstract void dibujate();
3     // Sigue la definición de la clase
4 }
5
6 public class Triangulo extends Figura {
7     public void dibujate() {
8         // Código para dibujar un triángulo
9     }
10    // Sigue la definición de la clase
11 }
12
13 public class Cuadrado extends Figura {
14     public void dibujate() {
15         // Código para dibujar un cuadrado
16     }
17    // Sigue la definición de la clase
18 }
19
20 public class Circulo extends Figura {
21     public void dibujate() {
22         // Código para dibujar un círculo
23     }
24    // Sigue la definición de la clase
25 }
```

```
1 Figura figura = new Circulo();
2 figura.dibujate(); // Dibujará un círculo
3 figura = new Triangulo();
4 figura.dibujate(); // Dibujará un triángulo
5 figura = new Cuadrado();
6 figura.dibujate(); // Dibujará un cuadrado
```


INTERFACES

- Los **interface** son una nueva construcción del lenguaje Java que da un paso más allá en las clases **abstract**. Se puede pensar que un interface es como una clase **abstract** en las que todos sus métodos son **abstract**.
- Siguiendo con el ejemplo de las figuras.....

```
1 public interface Dibujable {  
2     public void dibuja();  
3 }
```

- Las clases no extienden a los **interfaces** si no que los implementan y esto se indica con el uso de la palabra reservada **implements**:

```
1 public class Triangulo implements Dibujable {
2     @Override
3     public void dibuja() {
4         // Código para dibujar un triángulo
5     }
6     // Sigue la definición de la clase
7 }
8
9 public class Cuadrado implements Dibujable {
10     @Override
11     public void dibuja() {
12         // Código para dibujar un cuadrado
13     }
14     // Sigue la definición de la clase
15 }
16
17 public class Circulo implements Dibujable {
18     @Override
19     public void dibuja() {
20         // Código para dibujar un círculo
21     }
22     // Sigue la definición de la clase
23 }
```

- En la definición de un **interface** podemos declarar cualquier número de métodos y también cualquier número de constantes:

```
1 public class interface Dibujable {  
2     public static final Color BLANCO = new Color(255,255,255);  
3     public static final Color NEGRO = new Color(0,0,0);  
4     public void dibuja();  
5 }
```

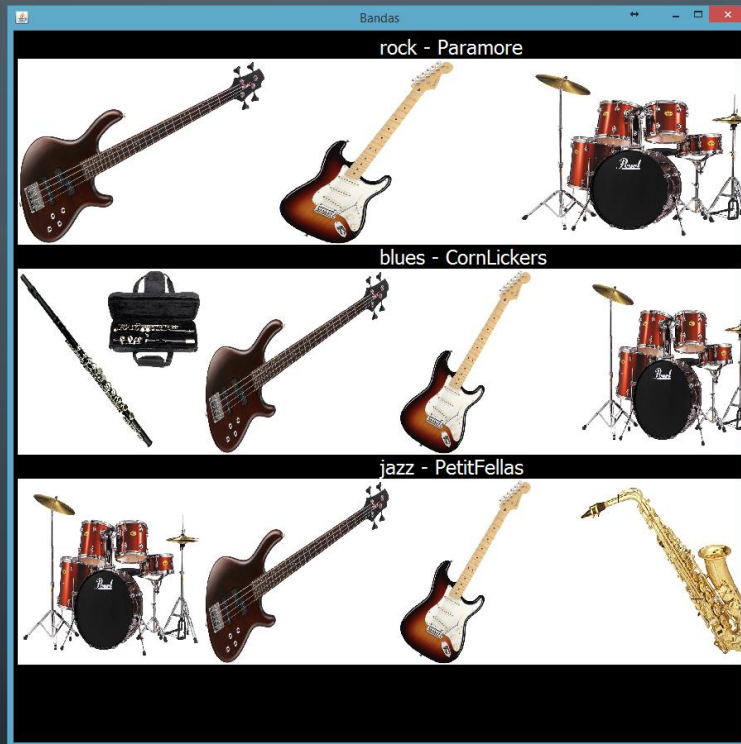
- Los **interface** al igual que las clases se pueden extender, y además un interface puede extender a más de un interface, **la herencia simple es una restricción en el ámbito de las clases, los interface no poseen esta restricción:**

```
1 public interface Figura extends Dibujable , Transformable {  
2     public void gira(float angulo);  
3 }
```

EJEMPLO

- Un sello discográfico quiere desarrollar una aplicación en lenguaje Java que muestre los artistas con los cuales se tiene firmado un contrato, en la primera versión se requiere que aparezcan en pantalla el nombre de la banda, el género que tocan y los instrumentos que usan, en versiones posteriores se añadirán datos de los integrantes y el teléfono del manager.
- Se requiere hacer uso de la herencia en Java para ahorrar tiempo, reutilizar código y facilitar la adición de nuevos elementos en un futuro.

EJEMPLO DE PANTALLA



EJERCICIO

- Una empresa de comida rápida desea incrementar la velocidad en la que se despachan los pedidos, para ello se propone la creación de una aplicación que en su primera versión de prueba permitirá al usuario observar los diferentes tipos de pizzas, cada una tendrá 3 o más ingredientes (algunos pueden ser repetidos).
- Se debe hacer uso de interfaces o clases abstractas para la construcción de los diferentes tipos de pizza.
- La interfaz debe ser similar a la presentada en el ejemplo de la banda y no será necesario agregar tipos de pizza en tiempo de ejecución, puede agregarse mediante el código o usando algún cuadro de diálogo.