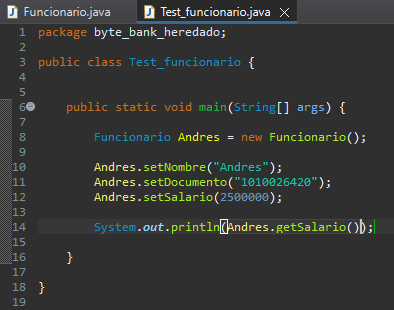
Curso de  
**Java Polimorfismo: Entendiendo herencia e interfaces**

**INTRODUCCION A HERENCIA:**

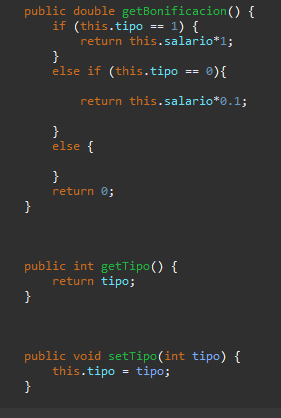
Clase funcionario



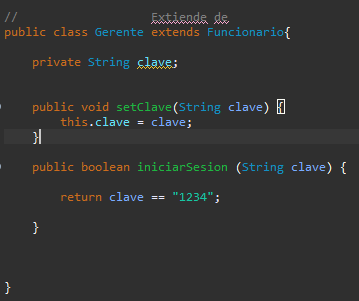
Clase test funcionario:



Clase gerente:



Comenzando con la herencia:



Lo que aprendimos:

En esta aula comenzamos a hablar de herencia y aprendimos:

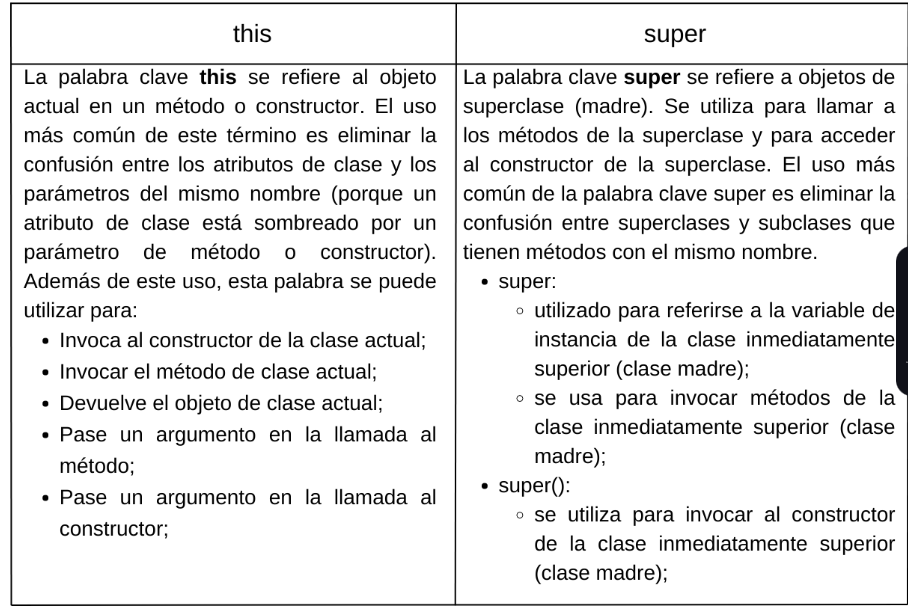
* Cuáles son los problemas que la herencia resuelve.
* Cómo usar la herencia en Java a través de la palabra llave *extends*
* Al heredar la clase hija gana todas las características (atributos) y todas las funcionalidades (métodos) de la clase madre.
* Conocimos el primer beneficio de la herencia: *La reutilización de código*.

En la próxima clase veremos más detalles sobre la herencia como la palabra llave *super*, *protected* y la sobreescritura de métodos.

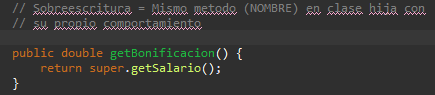
SUPER Y REESCRITA DE METODOS:

Método Super:





Sobreescritura:



Modificadoresd e acceso:

**Modificadores de acceso**

Los modificadores de acceso o accesibilidad son algunas palabras claves utilizadas en el lenguaje Java para definir el nivel de accesibilidad que los elementos de una clase (atributos y métodos) e incluso la propia clase puede tener los mismos elementos de otra clase.

**Public**

Este es el modificador menos **restrictivo** de todos. De esta manera, cualquier componente puede acceder a los miembros de la clase, las clases y las interfaces.

**Protected**

Al usar este modificador de acceso, los miembros de la clase y las clases son accesibles para otros elementos siempre que estén dentro del mismo package o, si pertenecen a otros packages, siempre que tengan una relación extendida (herencia), es decir, las clases secundarias pueden acceder a los miembros de su clase principal (o clase de abuelos, etc.).

**Private**

Este es el modificador de acceso **más restrictivo** de todos. Solo se puede acceder a los miembros definidos como privados desde dentro de la clase y desde ningún otro lugar, independientemente del paquete o la herencia.

Lo que aprendimos:

En esta clase entramos más a fondo en la herencia. Aprendimos:

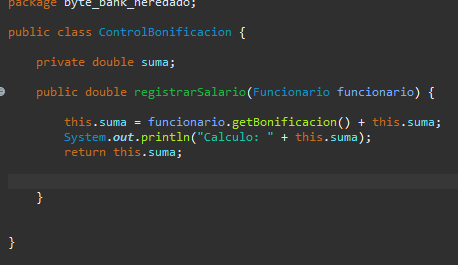
* que la clase madre es llamada de super o base class.
* que la clase hija también es llamada de sub class.
* como aumentar la visibilidad de un miembro (atributo, método) a través de *protected*.
* cómo acceder o llamar un miembro (atributo, método) a través de *super*.
* cómo redefinir un método a través de la sobreescritura. En la próxima clase veremos un nuevo beneficio de la herencia, el **Polimorfismo**. ¡Aguarda!

ENTENDIENDO POLIMORFISMO:

Introducción al polimorfismo:



Extends:



Lo que aprendimos:

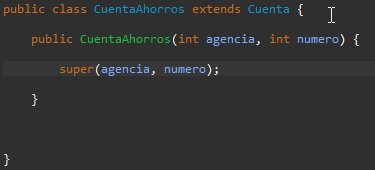
En esta clase aprendimos que:

* los objetos no cambian de tipo;
* la referencia puede cambiar, y ahí es donde entra el polimorfismo;
* el polimorfismo permite utilizar referencias más genéricas para comunicarse con un objeto;
* el uso de referencias más genéricas permite desacoplar sistemas.

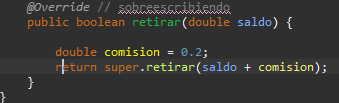
En el siguiente vídeo, hablaremos sobre cómo se comportan los constructores en la herencia.

HERENCIA Y EL USO DE CONSTRUCTORES:

Mas sobre polimorfismo:



Sobreescritura de métodos:



Lo que aprendimos:

En esta clase, vimos:

* Conceptos de herencia, constructores y polimorfismo
* Usando la anotación @Override
* Los constructores no se heredan
* Se puede llamar a un constructor de clase madre mediante super()

CLASES Y METODOS ABSTRACTOS:

Clase abstracta:

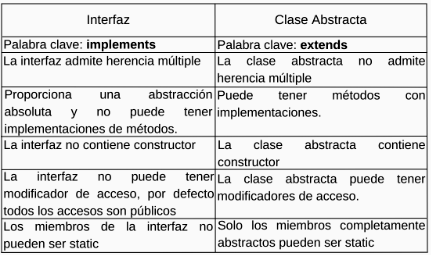


Método abstracto:



Obliga a los hijos a implementar el método que sea abstracto.

Cuenta abstracta:



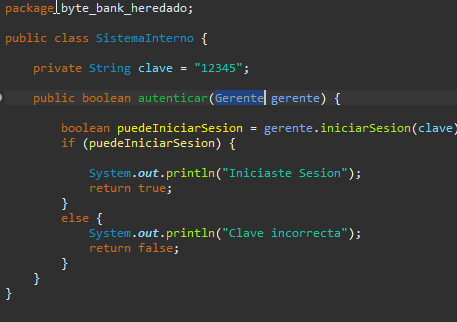
Lo que aprendimos:

En esta clase aprendimos:

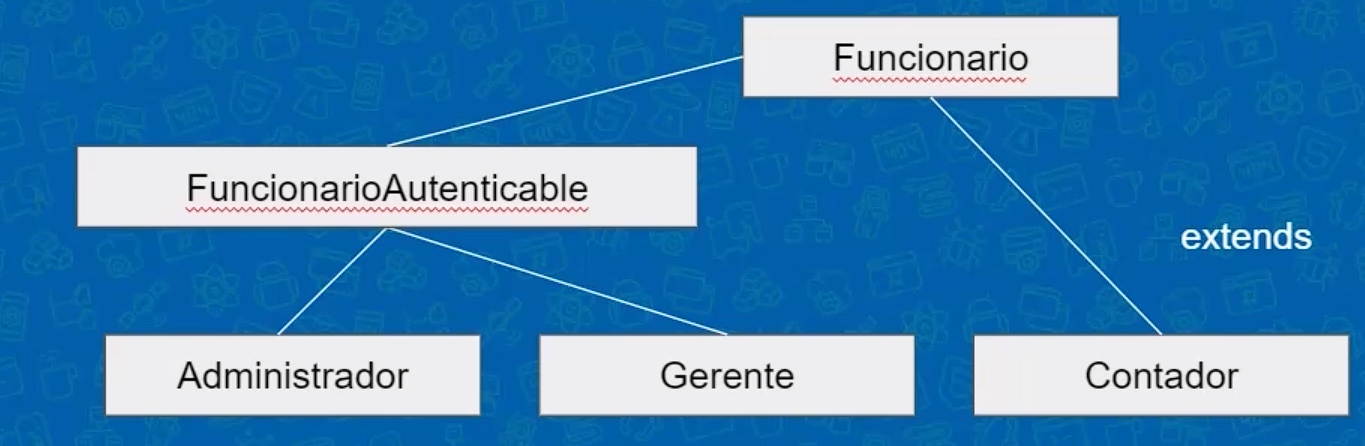
* Qué son las clases abstractas
* Para qué sirven las clases abstractas
* Qué son los métodos abstractos
* Para qué sirven los métodos abstractos

INTERFACES:

Herencia múltiple:



Modificando la estructura:



Interfaces:

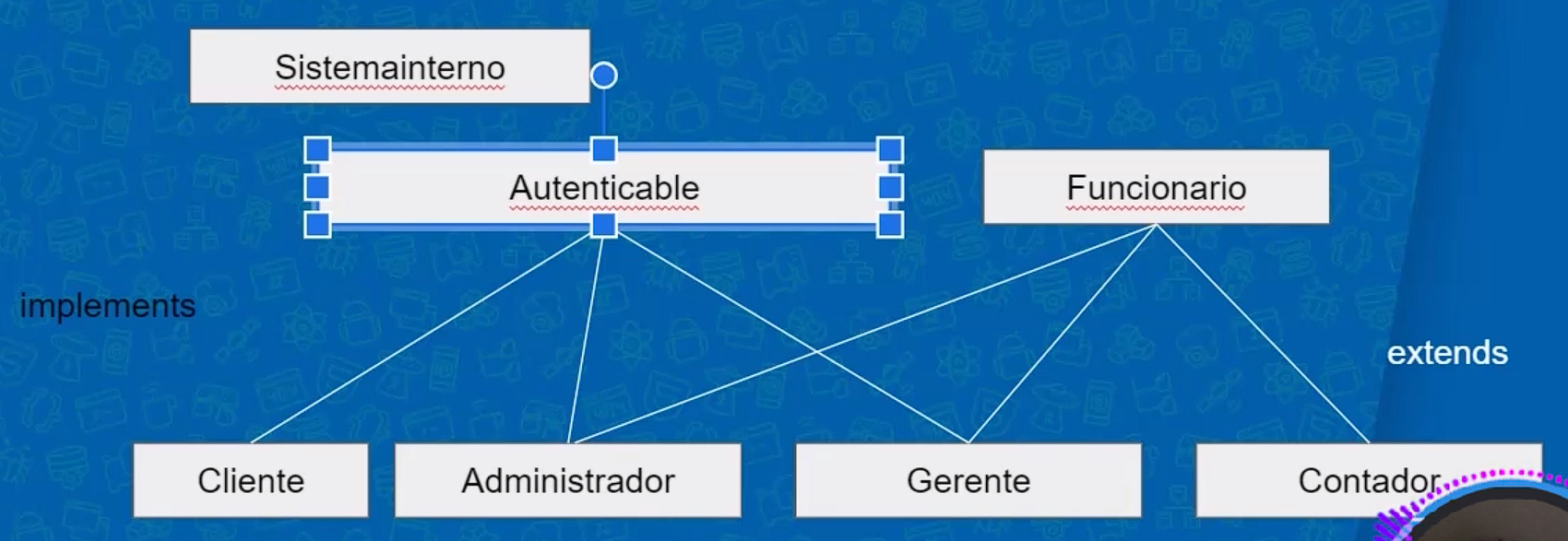
La interfaz no tiene métodos implementados.

Todos los métodos son abstractos.

La interfaz no puede tener atributos privados.

Implementación de interfaces:

Resumen:



En cuanto al concepto de polimorfismo, marque las alternativas correctas:

Principio del formulario

* Alternativa correta



Es la capacidad de un objeto de ser referenciado por varios tipos.

Correcto. Podemos comunicarnos con un objeto a través de diferentes tipos de variables.

Por ejemplo, si hay una clase *Gerente* que sea hija de *Funcionario*. Un objeto de tipo *Gerente* puede ser referenciado como tipo *Funcionario* también.

* Alternativa correta



Tenemos polimorfismo cuando una clase se extiende de otra o también cuando una clase implementa una interfaz.

Correcto, tenemos polimorfismo por herencia o interfaz.

Final del formulario

Lo que aprendimos:

En esta clase aprendimos que:

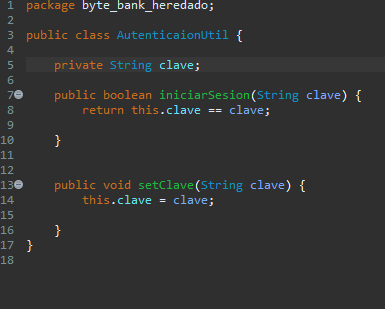
* No hay herencia múltiple en Java.
* Conceptos de interfaz.
* Diferencias entre clases abstractas e interfaces.
* Las interfaces son una alternativa a la herencia con respecto al polimorfismo

PRACTICANDO HERENCIA DE INTERFACES:

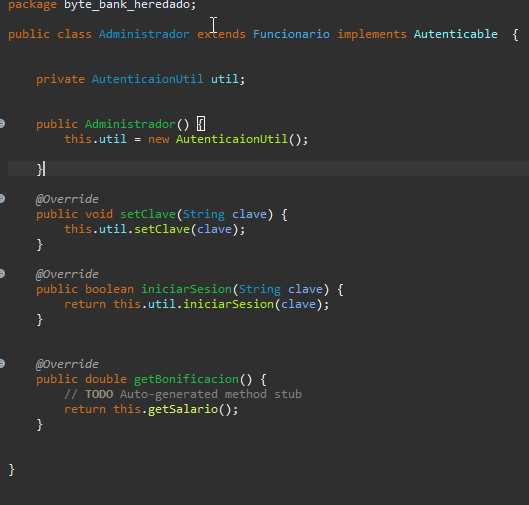
Overview Herencia:



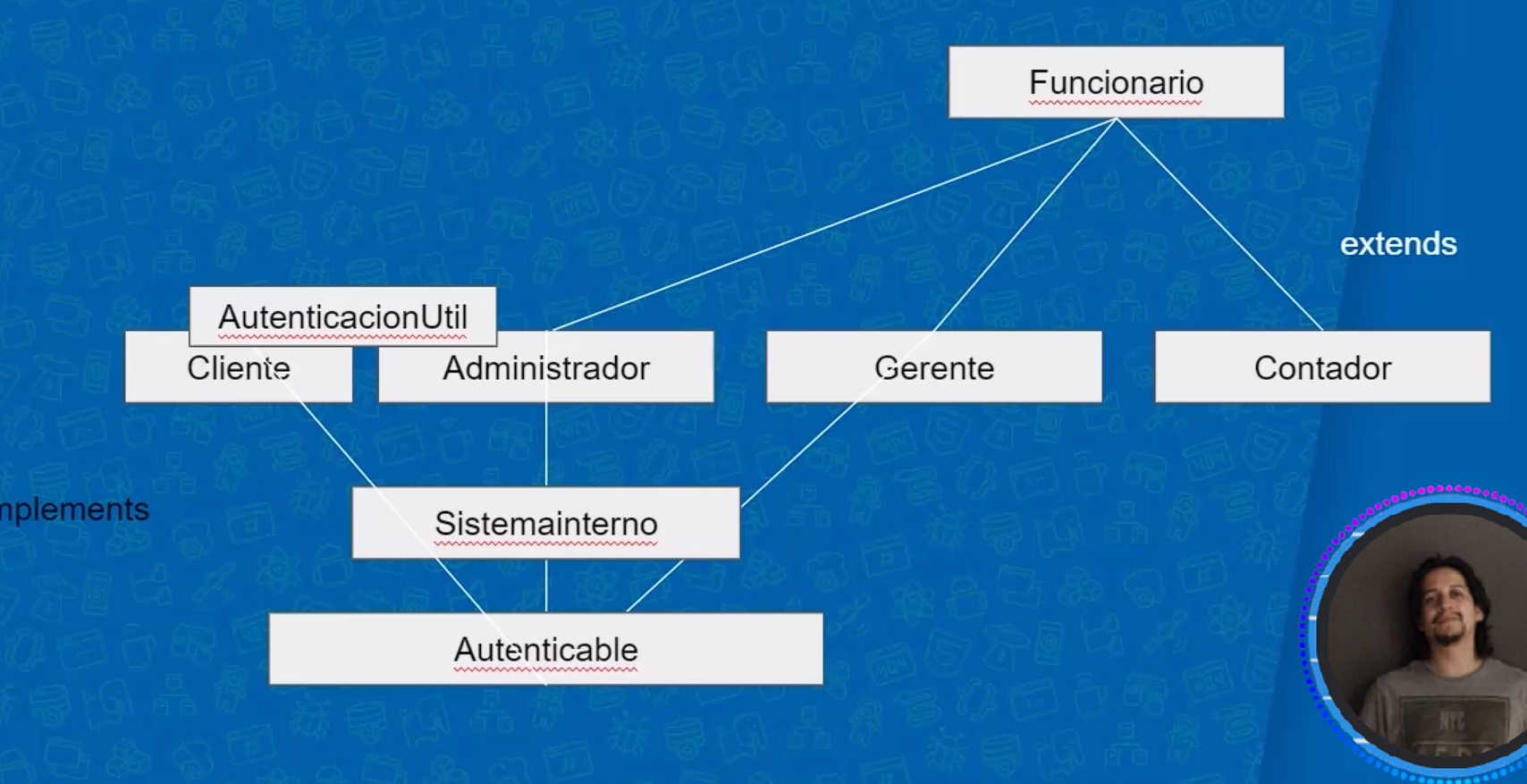
Usando herencia e interfaces:



Reviendo la composición de objetos:



Composición objetos:



Lo que aprendimos:

En esta clase aprendemos:

* Más en profundidad sobre el uso de interfaces
* Trabajamos más profundamente con la herencia
* Vimos otras aplicaciones de herencia e interfaz