# Java Polimorfismo: Entendiendo herencia e interfaces

## Conclusión 1: Introducción a herencia

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.

|  |
| --- |
|  |

## Conclusión 2: Super y reescrita de Métodos (intercambiar modulo 5 con 3)

* 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!

|  |
| --- |
|  |

## Conclusión 3: Entendiendo Polimorfismo

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.

|  |
| --- |
|  |

## Conclusión 4: Herencia y el uso de constructores

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()

¡En el siguiente vídeo hablaremos sobre cómo se comportan las clases y métodos abstractos! Espere :)

|  |
| --- |
|  |

## Conclusión 5: Clases y métodos abstractos

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

¡En la siguiente clase veremos sobre el uso de interfaces!

|  |
| --- |
|  |

## Conclusión 6: Interfaces

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

¡En el próximo capítulo practicaremos un poco más sobre herencia e interfaces!

|  |
| --- |
|  |

## Conclusión 7: Practicando herencia e interfaces

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

|  |
| --- |
|  |

## Anexos:

* <https://docs.oracle.com/javase/8/docs/api/java/util/Map.html>
* <https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html>

## Código:

* Extends: extiende clases a otros grupos de datos, solo se puede extender un solo padre al hijo
* Super: da acceso a los métodos de la clase padre
* Abstract: Representación conceptual de la clase y no física. No te obliga a ingresar un cuerpo, se vuelve libre porque no tiene implementación.

Una clases abstracta puede tener campos o variables, también puede referenciar objetos no abstractos, puede tener elementos estáticos, puede tener contructores y contructores por deefectos, get y set. Solo no permite instancias

* Protected: permite el acceso a las clases hijas
* Interface: es muy similar a una clase abstracta solo que no puede tener métodos ya implementados y con cuerpo. En la interface todos los métodos son abstractos por defecto. Una interface no puede extender de una clase. No puede tener variables a diferencia de la clase abst. Las interfaces son abstractas aunque no se declare.

No se pueden generar hijos de una interfaces, porque no es una clase (es decir no se puede extender o heredar). A cambio puedes llamar a implements. Ayudo a desacoplar todo lo que es autenticable de funcionario.

No representa un familiaridad, pero si un carácter en común.

* Default: permite implementar métodos con cuerpo en interfaces abstractas

## Vocabulario:

* Sobre-escritura de método: override
* Paquete
* Herencia: ayuda a reutilizat código con un significado
* Polimorfismos:
* Firma del método
* Escalable
* Composición de objetos: para que nunca inicie con null
* Desacoplar
* CamelCase
* Acoplar