**SECCIÓN 20**

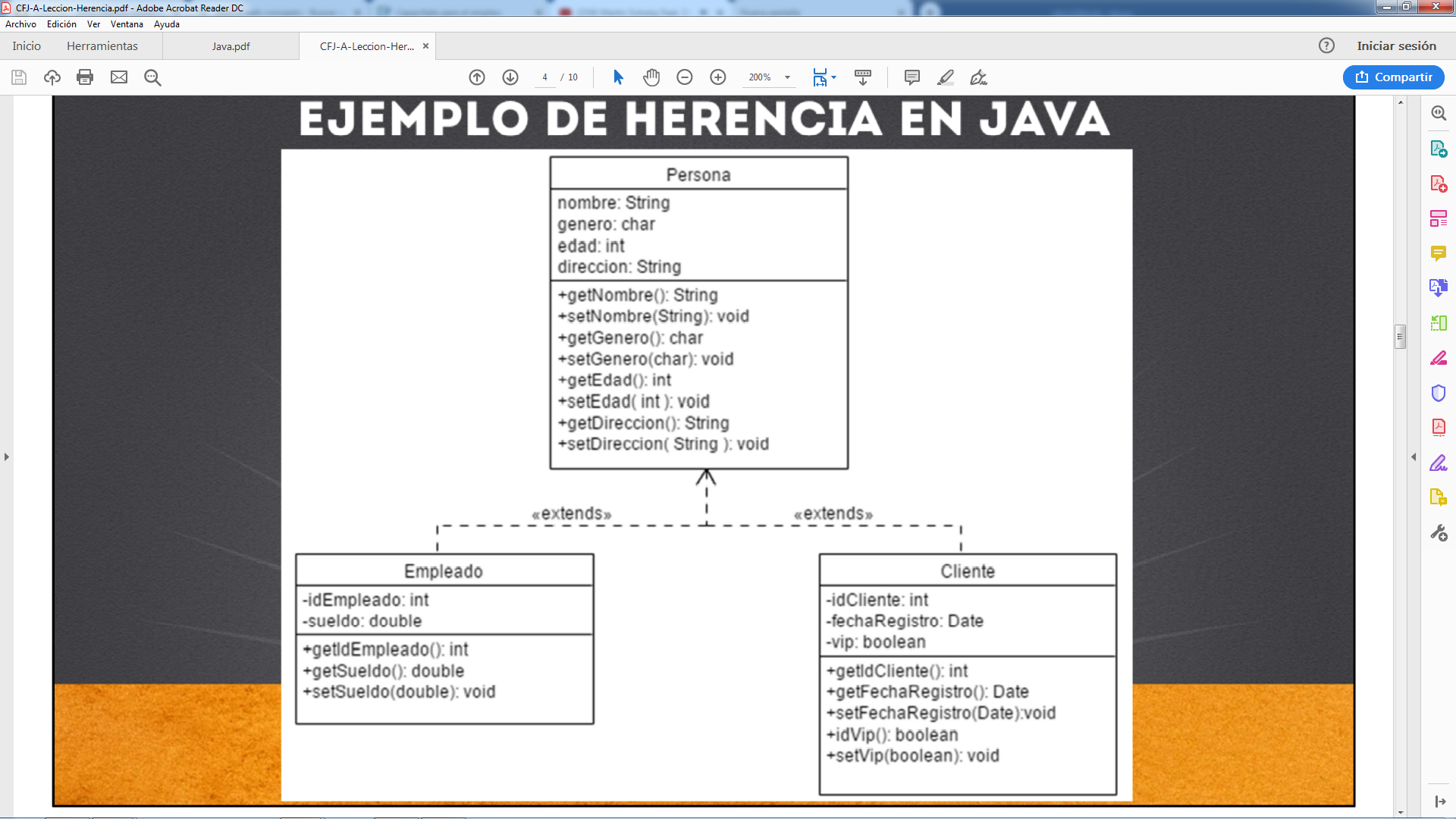
**HERENCIA EN JAVA**

Una de las formas en que entendemos la herencia, son las características que se mantienen de generación en generación en una familia. Por ejemplo, los abuelos tenían cabello rubio, y uno de los hijos hereda esta característica, a su vez el hijo tiene a su vez un hijo que también hereda el cabello rubio.

En la programación orientada a objetos, el concepto de herencia es exactamente igual. Definiremos una jerarquía de clases que nos permitirán heredar características entre clases Padre y clases Hijas.

La herencia nos permitirá representar características o comportamiento en común entre clases, permitiendo definir en la clase Padre los atributos o métodos que sean comunes a las clases hijas, las cuales heredarán estos atributos o métodos definidos en la clase Padre.

Lo anterior permite evitar duplicar el código entre la clase Padre y las clases Hijas, por lo que cumplimos con la reutilización de código que es uno de los principales objetivos de la POO.



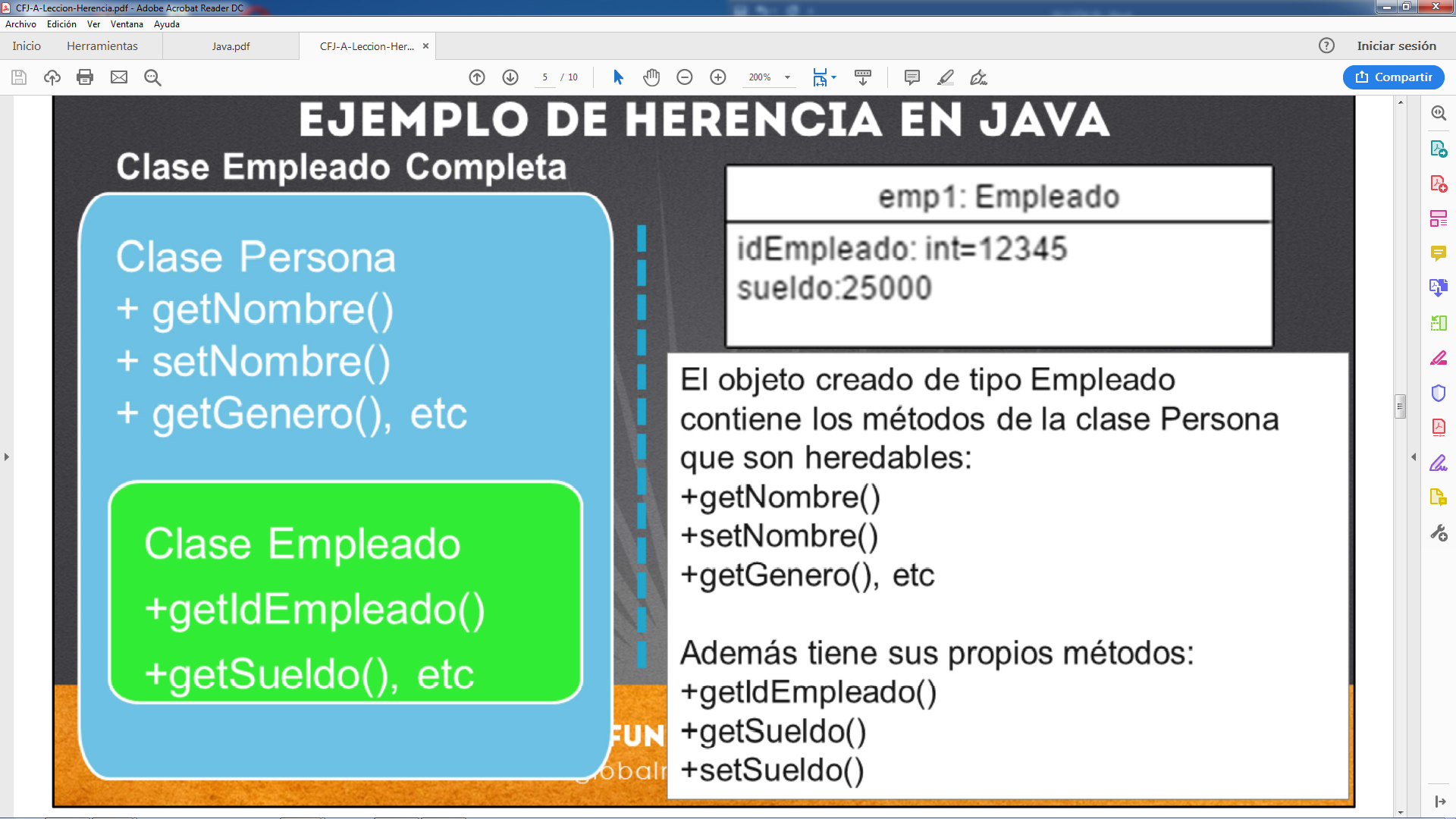
Vamos a ver un ejemplo de herencia en Java. Podemos observar una clase llamada Persona, la cual como cualquier persona tiene muchas características, pero nos vamos a enfocar a unas cuantas. Una persona tiene un nombre, un genero, una edad y una dirección, vamos a capturar estas características en cada uno de los atributos que corresponden en la clase Java, puede tener muchos más, pero nos vamos a enfocar en estos para simplificar el ejemplo.

Por otro lado, si estamos tratando de modelar un sistema de ventas, podríamos tener algunas variantes de una persona en nuestro sistema, por ejemplo la clase de Empleado y Cliente.

En este ejemplo no estamos indicando cómo llegamos al diseño de estas clases, sin embargo el proceso general es identificar los actores u objetos (entidades que generalmente serán nuestras clases) que interactuarán en nuestro sistema, y a partir de las necesidades de nuestro sistema podemos ir anotando cuales son las características (atributos) y funcionalidad (métodos) que debe tener cada entidad registrada.

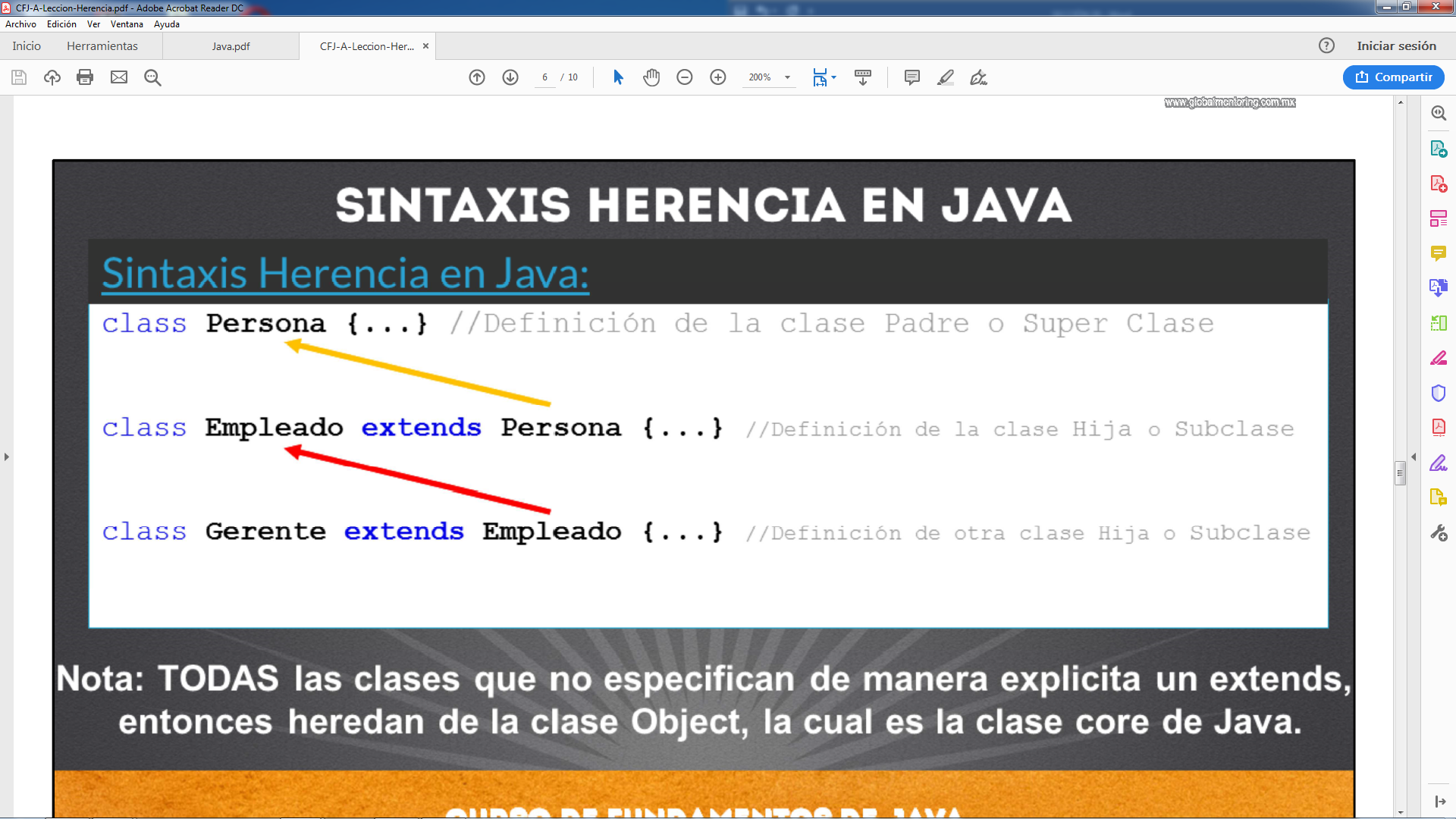
Una vez que se han encontrando estas entidades con sus atributos y métodos, vemos si existen características o funcionalidad en común, y allí es precisamente como llegamos a este tipo de diseños. Podemos observar que la clase Persona es la clase Padre y que las clases Empleado y Cliente extienden de la clase Persona, es decir, que heredan las características de la clase Persona.

Esto permite que no tengamos que repetir el código ya definido en la clase Persona, y una vez que creamos un objeto de tipo Empleado o Cliente, en automático también tendrá las características de la clase Persona.



Lo que visualizamos es que la clase Persona tiene atributos privados y varios métodos públicos. Los atributos privados no se heredan, al igual que los métodos privados. Es decir, mientras usemos el modificador de acceso private no se heredara el atributo o método marcado con este modificador.

En resumen, cuando visualicemos una clase que herada de otra, podemos imaginarnos a esta clase como el conjunto de sus propios atributos y métodos, así como la suma de los atributos o métodos hererables de la clase o clases Padre, ya que puede recibir herencia de una clase Padre, o una clase “Abuelo”, es decir, una clase Padre de su propia clase Padre.



Para definir la herencia en Java basta con utilizar la palabra extends en la definición de la clase Java e indicar el nombre de

la clase Padre.

Ej. class Empleado extends Persona

Cabe aclarar que en Java la herencia es simple, esto quiere decir que sólo podemos heredar de una clase a la vez. Puede haber una jerarquía de clases hacia arriba, por ejemplo Clase Abuelo, Padre e Hija, pero una clase no puede heredar por decir de una clase Padre y una clase Madre, es decir, de dos clases al mismo tiempo, únicamente por jerarquía de clases.

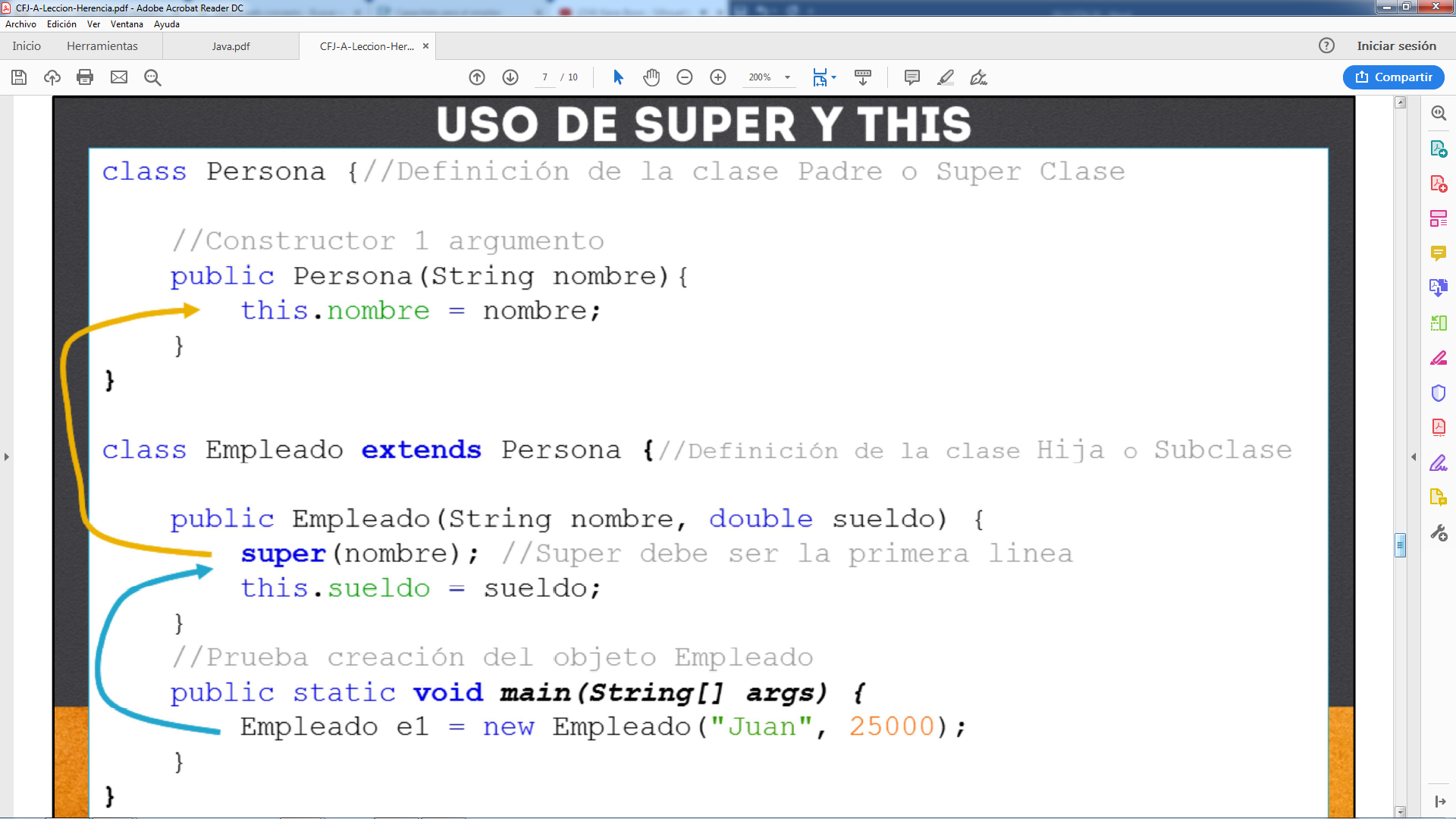
Esto NO está permitido (NO se puede heredar de dos clases a la vez, en Java la herencia es simple):

class Gerente extends Empleado, Persona

Es importante indicar que TODAS las clases en Java heredan de la clase Object, de manera indirecta o directa. En el caso de la clase Persona, como no está indicando que hereda de alguna clase, es decir, no usa extends en la definición de su

clase, se entiende que en automático hereda de la clase Object. Pero las clases Empleado o Gerente, debido a que extienden de otra clases, heredarán los métodos de la clase Object de manera indirecta, pero todas las clases en Java de

una manera u otra heredan las características de la clase Object. Finalmente, la clase Object es la clase core o base de Java, y pertenece al paquete java.lang. Posteriormente hablaremos del tema de paquetes en Java.



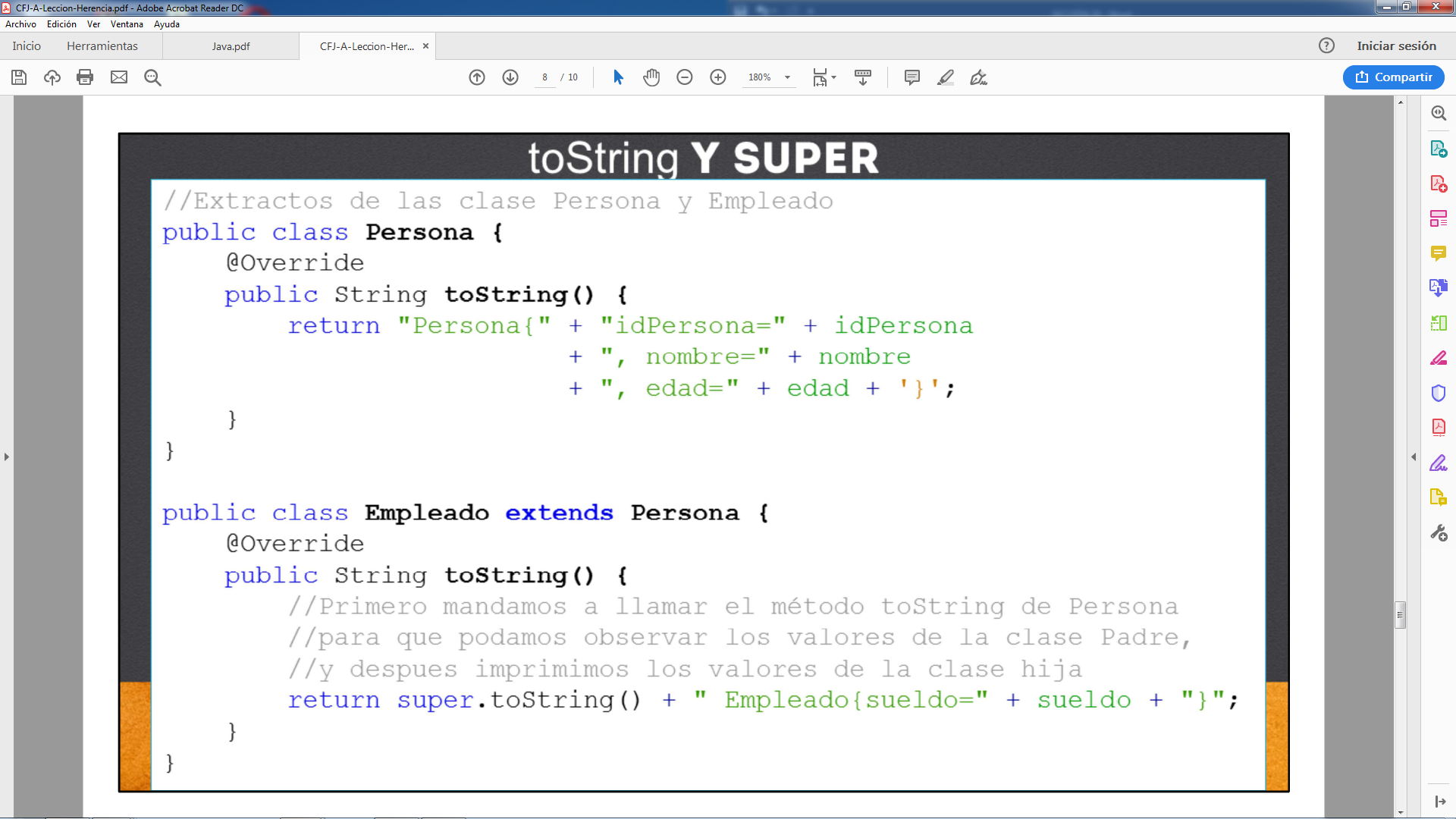
Si aplicamos el concepto de herencia la clase Hija o Subclase puede acceder a los atributos, métodos o constructores permitidos de la clase padre solo con utilizar la palabra super.

De esta manera podemos aprovechar código heredado y aprovecharlo por ejemplo para iniciar un objeto de la clase hija. Esto es muy común utilizarlo dentro del constructor para llamar desde el constructor de la clase hija al constructor de la clase Padre.

De hecho la palabra this se puede utilizar de manera idéntica pero para llamar a un constructor de la misma clase, y la palabra super para llamar a los constructores de la clase Padre. Ojo, super no solo permite llamar constructores de la clase padre, sino cualquier atributo, método o constructor heredable

de la clase Padre.

En el ejemplo podemos observar que utilizamos la palabra super para mandar a llamar el constructor de la clase Padre desde la clase Hija.



En la lámina podemos observar un ejemplo para el uso del método toString( ). El método toString( ) según hemos comentado, lo vamos a utilizar para mostrar el estado de un objeto, es decir, los valores de los atributos en cierto momento del tiempo de vida del objeto.

El método toString es un método heredado de la clase object. La clase Object es la clase Padre de todas las clases Java, ya sea de manera explícita o implícita, según hemos comentado. Por ejemplo, en el caso de la clase Empleado, la clase Object sería la clase abuela de la clase Empleado, debido a que se ha definido que de manera explícita la clase Persona como su clase Padre, y debido a que la clase Padre no ha indicado que extiende de alguna clase, entonces se entiende que su clase padre es la clase Object. De esta manera la clase Empleado también recibe los métodos y atributos que llegase a heredar de la clase Object.

Para indicar que un método pertenece a una clase padre o de nivel superiores, como la clase Object, entonces debemos agregar la anotación @Override, esto quiere decir que estamos sobreescribiendo el comportamiento de un método de la clase padre o clases superiores.

Ahora, este método toString( ) nos sirve según hemos comentado, para convertir el estado de un objeto a una cadena. Pero para el caso que estamos mostrando, la clase Persona tiene su método toString( ) y lo ideal es que reutilizaramos este código para completar el método toString de la clase Empleado, ya que si combinamos el método toString de la clase Persona con el de la clase Empleado, entonces podremos mostrar el estado de todos los atributos de la clase Empleado y de su clase padre Persona.

Cómo hacemos esto? Como observamos en el código, lo que hacemos es utilizar la palabra super, y debido a que toString es un método público redefinido (sobreescrito) en la clase Persona, entonces podemos acceder a este método de la clase padre por medio de la palabra super, el operador punto y el nombre del método que deseamos acceder del padre. Esto nos va a regresar una cadena con el estado del objeto Persona, y con ello solo basta concatenar el valor que aún no hemos colocado de la clase Empleado, por ello concatenamos el valor de la variable sueldo y ahora si podemos regresar la cadena completa para que ahora

el método toString de la clase Empleado pueda mostrar tanto su propio estado como el estado de cada atributo de su clase padre que es Persona.