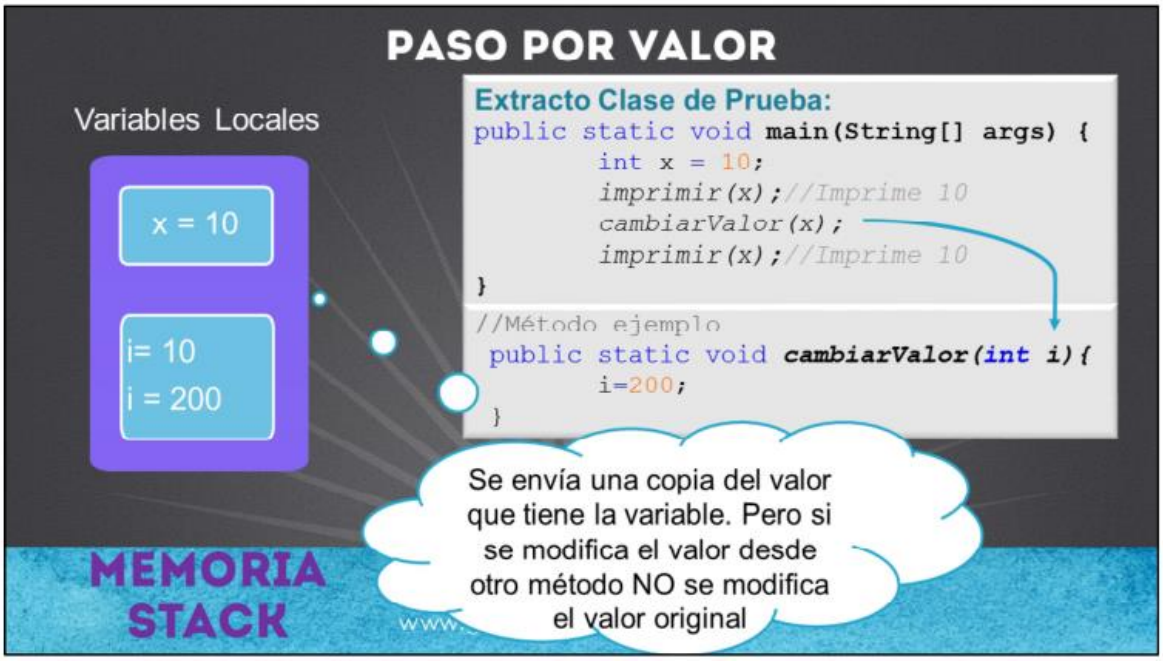
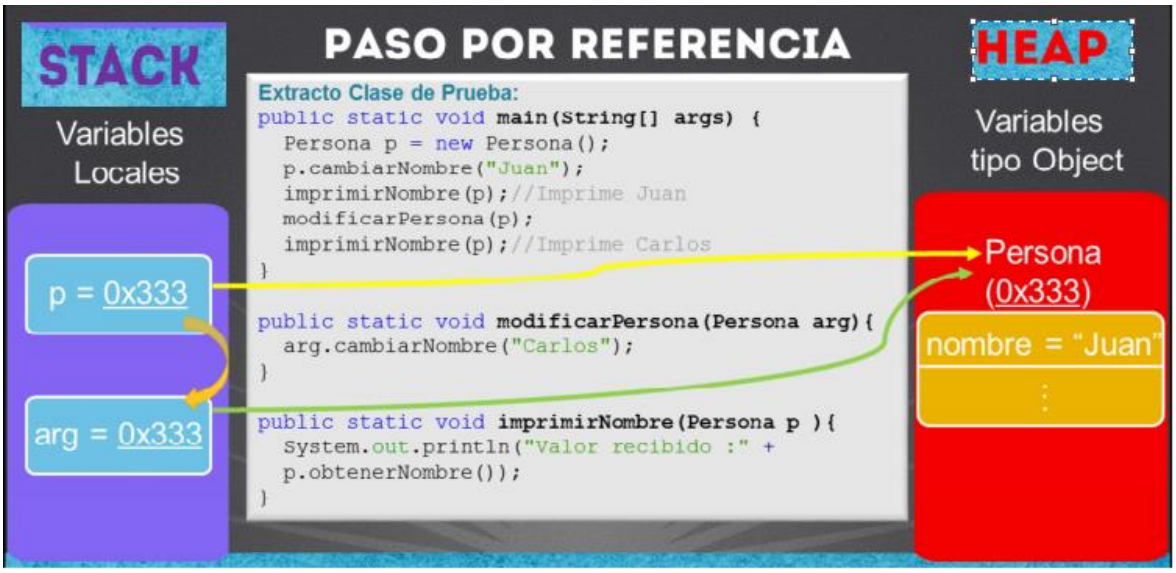
# PASO POR VALOR Y REFERENCIA




Hasta ahora hemos utilizado tipos primitivos al enviar valores a nuestros métodos, esto se conoce como paso por valor, ya que como vimos en lecciones anteriores, sólo estamos copiando el valor del tipo primitivo, pero cada variable contiene su propia copia.

Por ejemplo en el código mostrado podemos observar un ejemplo de paso por valor, también conocido como valores de tipo primitivo. Lo que observamos en el código es que al crear una variable, en este caso x, y asignarle un valor, por ejemplo 10, esta variable al ser modificada en otro método su valor original no se ve modificado, ya que solamente paso una copia de su valor original.

La manera de cambiar el valor de la variable x sería cambiándolo dentro del mismo método. Sin embargo en este ejemplo el método llamado cambiarValor intenta realizar un cambio a la variable x, pero como está fuera del alcance de este método acceder a esta variable, únicamente puede cambiar el valor de su variable local llamada i. Y al cambiar el valor de esta variable y regresar al método que hizo la llamada original




Además del paso por valores, en Java tenemos el paso por referencia, esto básicamente que utilizaremos objetos como parámetros en lugar de tipos primitivos. Al utilizar un objeto como parámetro y enviarlo a un argumento de un método lo que estamos haciendo es enviando la referencia del objeto que se está deseando utilizar, y en lugar de utilizar una copia del valor del objeto, lo que realmente estamos haciendo es apuntar al mismo objeto, con la finalidad de modificarlo directamente, sin necesidad de hacer una copia.

Esto se debe a que imaginemos que un objeto contiene 50 atributos o más, y si se realizar una copia del objeto, lo que estaríamos generando realmente es una copia de estos 50 valores. Por ello, cuando trabajamos con objetos, y los enviamos como argumentos a un método, sólo estamos enviando el valor de la referencia donde se ubica este objeto, de tal manera que podemos modificar directamente el objeto, ya sea desde la variable original que apunta al objeto, o desde la variable local que funciona como argumento en la llamada del método.

En el código podemos observar la variable p de tipo Persona, esta clase la crearemos en la sección de ejercicios, pero básicamente tiene un atributo llamado nombre de tipo String, y lo que podemos observar es que se modifica el valor del atributo nombre, esto debido a que la variable local llamada arg apunta al mismo objeto creado en la memoria Heap de tipo Persona, y por lo tanto al modificar el atributo del mismo objeto, entonces se modifica el objeto creado originalmente, y podemos acceder a este cambio incluso desde otros métodos.

Esto trae lugar a discusiones, ya que se dice que Java únicamente pasa parámetros por valor, y no por referencia, y de alguna manera es cierto, ya que como podemos observar en la memoria stack, el valor de p se "Copia" a la variable arg al llamar al método modificarPersona, pero debido a que lo que recibe arg es la dirección de memoria del objeto Persona creado originalmente, entonces la variable arg puede acceder a los atributos y métodos del objeto p, y de esta manera es que es posible modificar el estado o contenido del objeto original. Y al terminar de ejecutar el método modificarPersona y regresar al método main es posible acceder a estos cambios, y en lugar de mandar a imprimir Juan que era el nombre original, se manda a imprimir el valor de Carlos, que es el valor modificado por la variable arg.

De esta manera podemos observar como en Java cuando trabajamos con objetos, realmente lo que estamos proporcionando es el valor de la referencia de memoria, en este caso el valor de 0x333, y con esta referencia accedemos directamente al objeto, y así podemos modificarlo. Finalmente estas modificaciones las podremos acceder incluso fuera del método donde fue creado originalmente el objeto, o desde el método donde se creó dicho objeto.