Valor vs Referencia en Java

Valor y Referencia

En Java, el paso de argumentos a un método siempre se produce por valor.

Si el argumento es de tipo primitivo, se hace una copia del valor en el Stack. Este nuevo valor es completamente independiente del valor pasado inicialmente. Es una copia. Cualquier modificación que hagamos sobre esta nueva copia dentro del método no afectará de ninguna manera al valor original.

Si el argumento es un objeto, se hace una copia de la referencia en el Stack que apunta al mismo objeto (en el Heap) pasado como argumento al método.

Si el objeto pasado es un objeto inmutable (String, Wrapper) y lo "modificamos" se creará una nueva referencia que hará que ahora tengamos dos objetos completamente independientes en el Heap.

Si el objeto pasado es un objeto mutable (ArrayList etc.) las modificaciones que hagamos en él (por ejemplo añadir un elemento) afectarán al objeto pasado en el argumento ya que es el mismo objeto.

Paso de argumentos de tipos primitivos

1 Se crea la variable variable1 en el Stack con un valor de 8

```
int variable1 = 8;
```

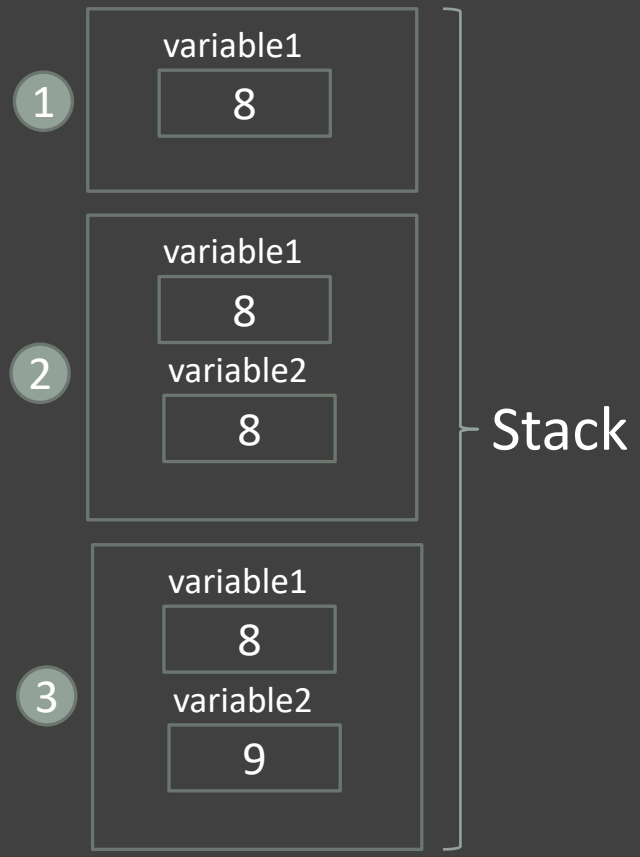
```
elMetodo(variable1);
```

2 Se crea la variable variable2 en el Stack con un valor de 8

```
public void elMetodo(int variable2){
    variable2=9;
};
```

3 Modificamos el valor de variable2 en el Stack con un valor de 9

variable2 es una variable local del método



Paso de argumentos de objetos mutables I

1 Se crea la variable miList en el Stack con un valor de 0X00454 que apunta a un objeto de tipo ArrayList en el heap

```
List miList = new ArrayList();
miList.add("hola");
```

```
elMetodo(miList);
```

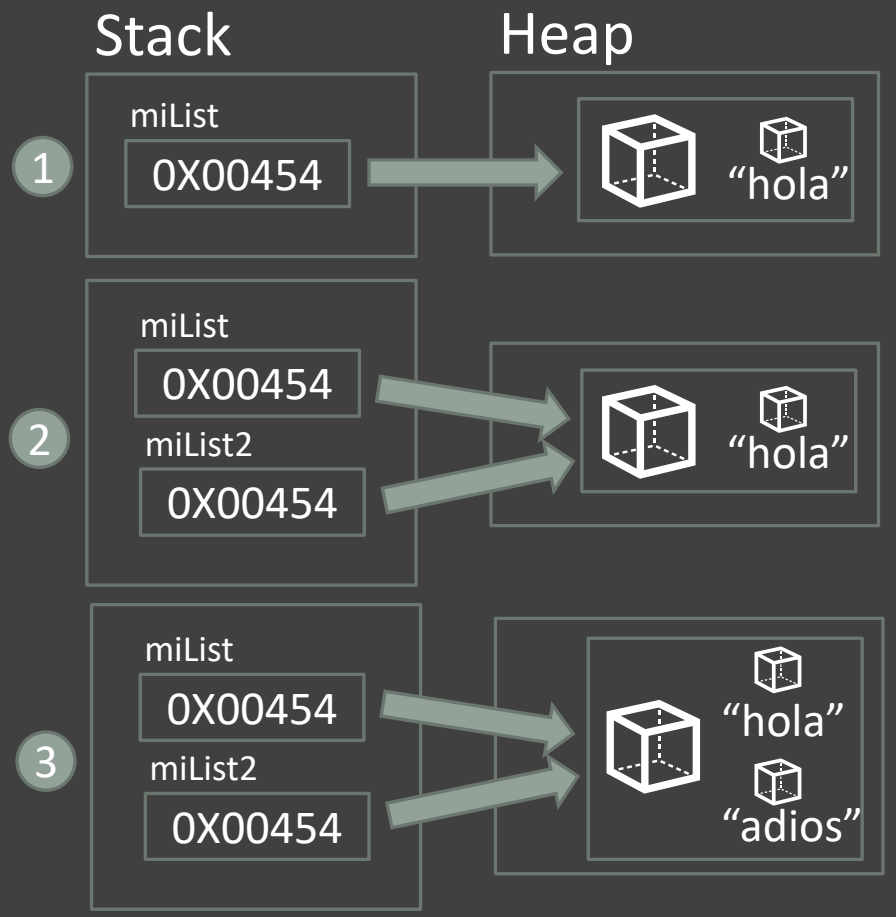
Se crea la variable miList2 en el Stack con un valor de 0X00454 (se copia el valor de la referencia)

2

```
public void elMetodo(List miList2){
    miList2.add=("adios");
};
```

3 Añadimos un elemento a miList2

miList2 es una variable local del método



Paso de argumentos de objetos mutables II

1 Se crea la variable miList en el Stack con un valor de 0X00454 que apunta a un objeto de tipo ArrayList en el heap

```
List miList = new ArrayList();
miList.add("hola");
```

```
elMetodo(miList);
```

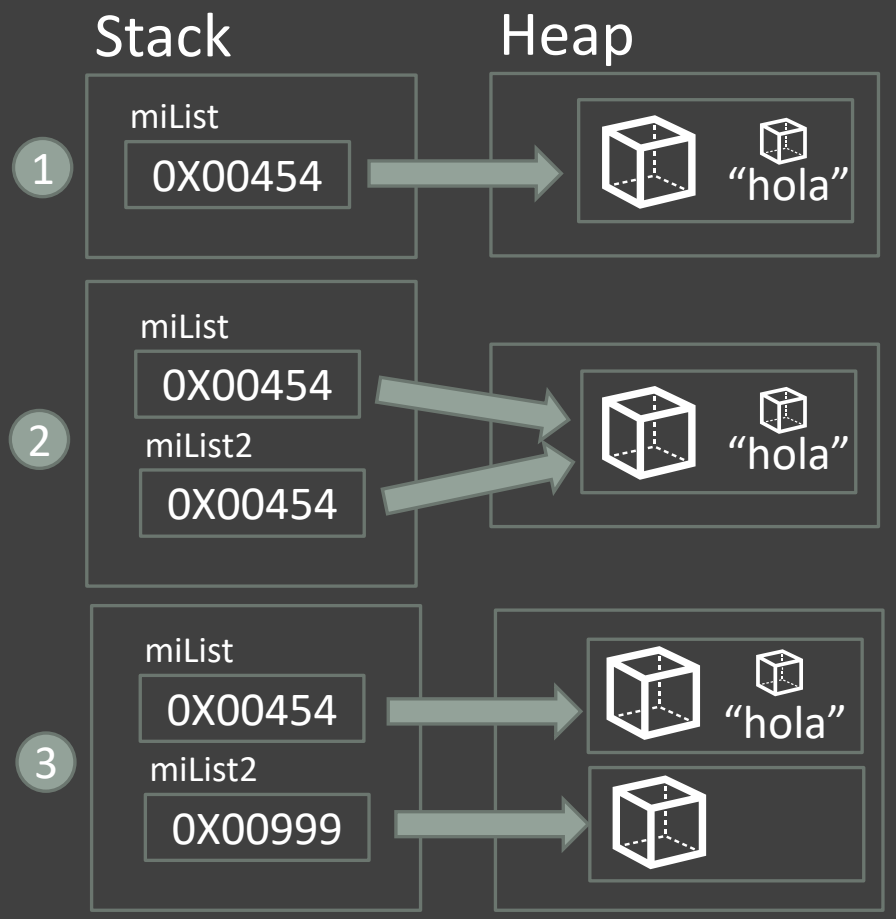
Se crea la variable miList2 en el Stack con un valor de 0X00454 (se copia el valor de la referencia)

2

```
public void elMetodo(List miList2){
    miList2 = new ArrayList()
};
```

3 Creamos un nuevo objeto en la variable miList2

miList2 es una variable local del método



Strings, Wrappers…

Paso de argumentos de objetos inmutables

1 Se crea la variable miString en el Stack con un valor de 0X00454 que apunta a un objeto de tipo String en el heap

```
String miString = "hola";
```

```
elMetodo(miString);
```

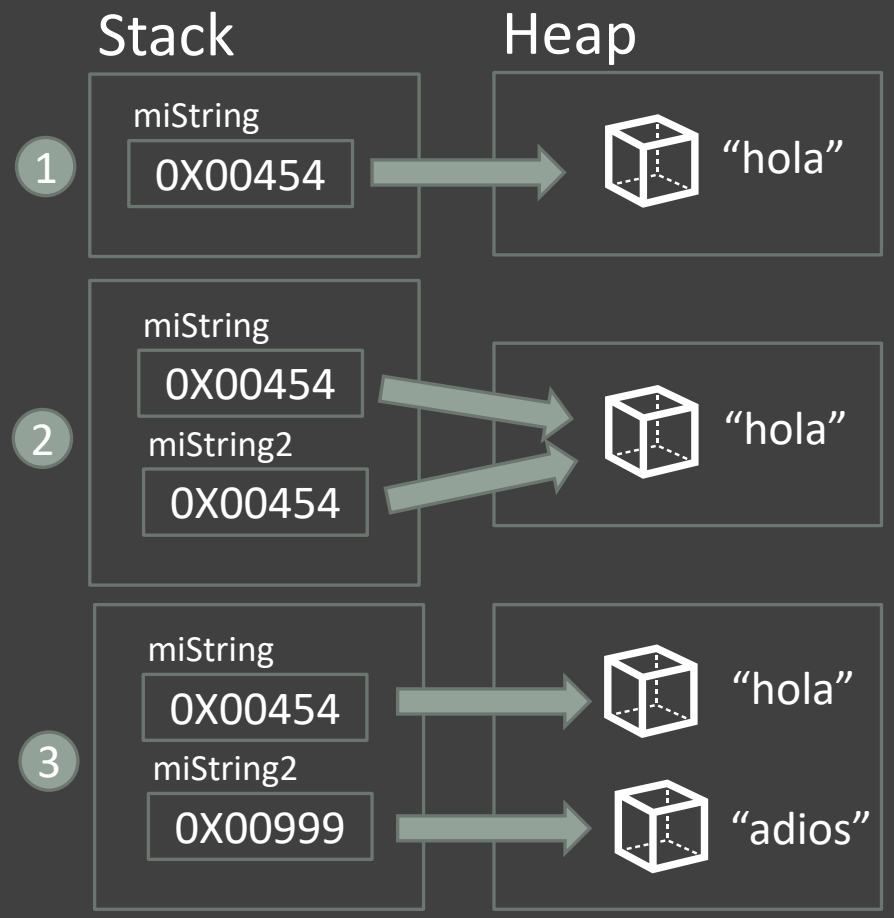
Se crea la variable miString2 en el Stack con un valor de 0X00454 (se copia el valor de la referencia)

2

```
public void elMetodo(String miString2){
    miString2="adios";
};
```

3 "Modificamos" miString2. Esto es imposible porque los Strings son objetos inmutables. Por lo tanto se crea una nueva referencia.

miString2 es una variable local del método



Paso de argumentos en Java

En Java el paso de argumentos a los métodos siempre es por VALOR

Tenemos que entender que una referencia es un valor en si mismo!!!

Tips finales 

1) La modificación de valores de tipos primitivos, dentro de un método, nunca afectan a la variable original.

2) Si se cambia la referencia al objeto dentro del método nunca afecta a la referencia original, sin embargo esto crea un nuevo objeto en la memoria Heap.

3) Modificar los atributos de un objeto pasado como argumento se refleja fuera del método.

4) Modificar colecciones y maps siempre se reflejan fuera del método.