Agenda

Pasaje de Parámetros en JAVA: ¿cómo funciona?

- -Tipos primitivos como parámetros
- -Objetos como parámetros
- -Tipos “inmutables” como parámetros

Parámetros de métodos en JAVA

En JAVA cuando invocamos a métodos y le pasamos parámetros, el tipo de pasaje de parámetros que usamos se llama “Pasaje de parámetros por valor o por copia”: **el método invocado recibe una copia del parámetro enviado.**

¿Cómo funciona el pasaje de parámetros por valor en JAVA?

-**Si los parámetros son datos primitivos** (int, double, float, char, boolean) los cambios que se hagan del parámetro en el método invocado sólo se reflejan dentro del método y cuando el método retorna estos cambios se pierden.

-Si los parámetros son objetos los cambios que se hagan en los valores de los objetos dentro del método (sus variables de instancia) se mantendrán cuando el método retorne. En este caso, el parámetro del método contiene una copia de la “referencia” (puntero) del parámetro enviado. Ambos “referencias” apuntan al mismo objeto.

Tipos Primitivos como Parámetros

Se copia el contenido de la variable x en el parámetro p.

El parámetro p es una variable local del método invocarMetodo() que inicialmente toma el mismo valor que x, es decir 3

```
package ayed2019.parametros;
public class DatosPrimitivosPorValor {
        public static void main(String[] args) {
        int x = 3;
        // invocamos a invocarMetodo()
        // con x como argumento
        System.out.println("Antes de invocar a invocarMetodo(), x = " + x);
        invocarMetodo(x);
        // imprimimos x para ver si su valor cambio
        System.out.println("Después de invocar a invocarMetodo(), x = " + x);
}
// cambia el parámetro en invocarMetodo()
public static void invocarMetodo(int p) {
        p = 10;
}
}
```

Antes de invocar a invocarMetodo(), x = 3
Después de invocar a invocarMetodo(), x = 3

Objetos como Parámetros

```
package ayed2019.parametros;
public class TestCirculo {
public static void main(String[] args) {
   Circulo miCirculo=new Circulo(10,15);
   TestCirculo test=new TestCirculo();
    System.out.println("Antes de invocar a moverCirculo el centro del círculo es:
     ("+ miCirculo.getX()+","+ miCirculo.getY()+")");
    test.moverCirculo(miCirculo, 23, 56);
    System.out.println("Después de invocar a moverCirculo el centro del círculo es:
      ("+ miCirculo.getX()+","+ miCirculo.getY()+")");
}
public void moverCirculo(Circulo c, int deltaX, int deltaY) {
// mueve el origen del círculo a x+deltaX, y+deltaY
    int origenX=c.getX();
    int origenY=c.getY();
    c.setX( origenX+ deltaX);
    c.setY(origenY+ deltaY);
}
}
```

Circulo

-int x
-int y
+Circulo (int cx, int cy)
+void setX(int x)
+int getX()
+void setY(int y)
+int getY()

Se copia el contenido de la variable miCirculo en el parámetro c: miCirculo y c apuntan a la misma dirección de memoria

Antes de invocar a moverCirculo() el centro del círculo es: (10,15)
Después de invocar a moverCirculo() el centro del círculo es: (33,71)

Tipos “Inmutables” como Parámetros

Tipos Wrapper y los String son innmutables

Se copia el contenido de la variable x en el parámetro p y el de str en msg. En este caso x y p apuntan a direcciones de memoria diferente que tienen inicialmente el mismo contenido. Idem para str y msg

Las variables de tipo Integer y String son “inmutables”, sus valores no cambian, por eso a pesar de ser objetos, sus valores no cambian. Los parámetros de tipo Wrapper y String funcionan de manera similar a los tipos primitivos.

```
package ayed2019.parametros;
public class DatosInmutablesPorValor {
public static void main(String[] args) {
 Integer x = 3;
 String str= "Hola";
 System.out.println("Antes de invocar a invocarMetodo(), x = " + x+ "  str= " + str );
invocarMetodo(x, str);
      // imprimimos x y srt para ver si cambiaron sus valores
 System.out.println("Después de invocar a invocarMetodo(), x = " + x + "  str= " + str );
}
// cambia el parámetro en invocarMetodo()
public static void invocarMetodo(Integer p, String msg) {
       p = 10;
     msg=msg+ "  "+ "AyED 2019 ";
}
}
```

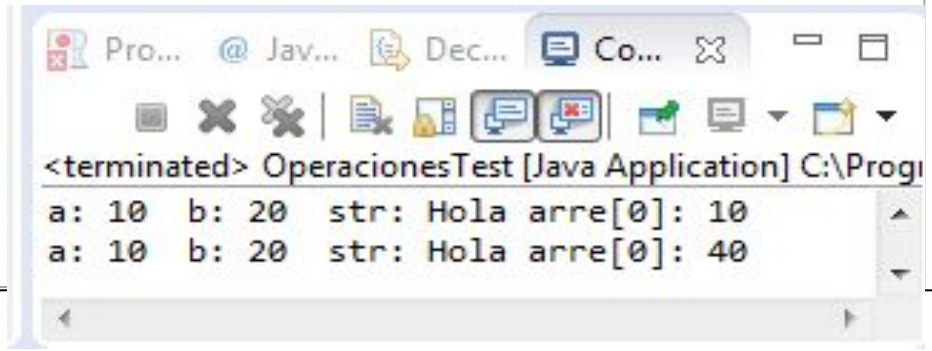
Antes de invocar a invocarMetodo(), x = 3, str= Hola
Después de invocar a invocarMetodo(), x = 3, str= Hola

Wrapper

- Integer
- Double
- Float
- Boolean
- Long
- Short
- Byte
- Character

TODO Junto

```
package parametros;
public class Operaciones {
 public static void cambiar(int x, Integer y, String palabra, int[] arre)   {
    x = x + 20;
    y =  y + 20;
    palabra = palabra + "Juana";
    arre[0] = arre[0] + 30;
 }
}
```




```
package parametros;
public class OperacionesTest {
 public static void main(String[] args) {
  int a = 10;
  Integer b = new Integer(20);
  String str = new String("Hola");
  int[] arre = {10};
  System.out.println("a: "+ a +"  b: "+b +"  str: "+str+ "arre: " + arre[0]);

  Operaciones.cambiar(a, b, str, arre);
  System.out.println("a: "+a +"  b: "+b +"  str: "+str+ "arre: " + arre[0]);
  }
}
```

¿Cómo hago para obtener un valor actualizado?

Mediante la sentencia **return**

```
package ayed2019.parametros;
public class DatosPrimitivosPorValor {
        public static void main(String[] args) {
        int x = 3;
        // invocamos a invocarMetodo()  con x como argumento
        System.out.println("Antes de invocar a invocarMetodo(), x = " + x);
        int y= invocarMetodo(x);
        // imprimimos x para ver si su valor cambió
        System.out.println("Después de invocar a invocarMetodo(), x = " + x);
}

public static int invocarMetodo(int p) {
        p = 10;
      return p;
}
}
```

¿Cómo hago para obtener un valor actualizado?

Guardando el valor actualizado en una **variable de instancia**.

```
public class Maximo {
private int max;
    public void buscarMaximo(int[] a) {
       max = 0;   // variable de instancia
       for (int i=0; i<a.length; ++i)
       if (a[i] > max) {
              max = a[i];
       }
}
     public void imprimirMaximo(int[] a) {
            buscarMaximo(a);
            System.out.print(max);
}
}
```

No abusar de esta opción.

OJO: esta solución podría llegar a ensuciar la estructura cuando se pide que implementen los algoritmos en los árboles, grafos, etc.