

Documentación Video 202301005_1

Durante la grabación se explican y resuelven dudas sobre temas específicos y ejercicios en concreto. La principal abarca la explicación de **Constructores, uso de `super()` y uso de `this()`**. Adicional se menciona la explicación de **Raíz de todas las clases `Object`**. Problemas y su solución como sobre *si es posible compilar un fragmento de código entero dentro de una clase abstracta y manera de inicializar de arrays multidimensionales*.

Constructores

En Java, un constructor es un bloque de código especial que se utiliza para inicializar objetos cuando se crean. Los constructores permiten establecer valores iniciales para los atributos del objeto y preparar el objeto para su uso.

```
1 public class Pato{
2     public static void main(String[] args) {
3         Pato p = new Pato();
4         System.out.println(p);
5     }
6 }
```

Aquí se aprecia bastante el cómo podemos llamar a nuestro constructor *Pato* ya que estamos creando una un objetivo de tipo *Pato*, aunque explícitamente no haya constructor como tal. Sí hace llamado del constructor default de *Pato*. Tal como se ve en este ejemplo.

```
1 Pato(){
2     super();
3     System.out.println("Constructor default");
4 }
```

Aunque no esté definido, internamente existe y es como el código anterior. Sin embargo, podemos hacer el llamado de más constructores, mediante la sobrecarga de constructores, para ello tendremos este código.

```
1 public class Pato{
2
3     String nombre; //null
4     int edad; //0
5
6     // Sobrecargar de constructores
7     Pato(){
8         //super(); <-- Primero es super
9         this("Nombre Default"); // <-- Segundo this, este redirecciona a otro constructor
10        System.out.println("Constructor default");
11    }
12
13    public Pato(String nombre){
14        super(); // <-- Se decide si llamarlo o no, siempre y cuando haya parametros
15        this.nombre = nombre;
16        System.out.println("Constructor con un string");
17    }
18
19    public Pato(String nombre, int edad){
20        super();
21        this.nombre = nombre;
22        this.edad = edad;
23    }
24 }
```

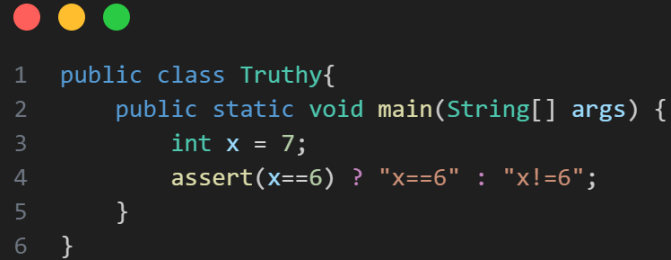
Aquí se aprecia varios constructores que hacen el uso de sobrecargar cada uno, y para ello declaramos variables como lo son el nombre y edad. En cada uno se puede ver que reciben parámetros de acuerdo a las variables que se pueden acceder mediante ellos mismos. Todo esto mediante el uso de *this()*; y para hacer una diferencia entre los **this**. Este *this()*; es para llamar a los constructores que estén dentro de la clase y este *this*. es para acceder a las variables de referencia declaradas. Todo el código de ejemplo quedaría de la siguiente manera con su respectiva salida.

```
1 // El default llama/extiende a Object
2 public class Pato{
3
4     String nombre; //null
5     int edad; //0
6
7     // Sobrecargar de constructores
8     Pato(){
9         //super(); <-- Primero es super
10        this("Donald"); // <-- Segundo this, este redirecciona a otro constructor
11        System.out.println("Constructor default");
12    }
13
14    public Pato(String nombre){
15        //super();
16        this(nombre, 8); // <-- Se llama siempre y cuando se pueda acceder
17        this.nombre = nombre;
18        System.out.println("Constructor con un string");
19    }
20
21    public Pato(String nombre, int edad){
22        //super();
23        this.nombre = nombre;
24        this.edad = edad;
25        System.out.println("Constructor con un string y un int");
26    }
27
28    @Override
29    public String toString() {
30        return "Pato [nombre= " + nombre + " edad= " + edad + " ]";
31    }
32
33    public static void main(String[] args) {
34        Pato p = new Pato();
35        System.out.println(p);
36    }
37 }
```

```
Constructor con un string y un int
Constructor con un string
Constructor default
Pato [nombre= Donald edad= 8]
```

Problema Assert

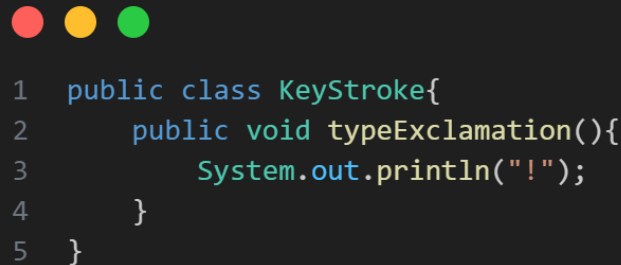
Tenemos el siguiente problema con sus siguientes respuestas:



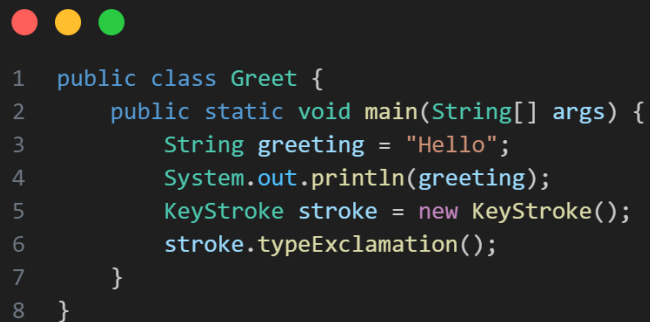
```
1 public class Truthy{
2     public static void main(String[] args) {
3         int x = 7;
4         assert(x==6) ? "x==6" : "x!=6";
5     }
6 }
```

1. Truthy.java does NOT compile ✓✓
2. Truthy.java compiles and the output is x != 6
3. Truthy.java compiles and an AssertionError is thrown with no additional output
4. Truthy.java compiles and an AssertionError is thrown with x != 6 as additional output

Problema Keystroke



```
1 public class KeyStroke{
2     public void typeExclamation(){
3         System.out.println("!");
4     }
5 }
```



```
1 public class Greet {
2     public static void main(String[] args) {
3         String greeting = "Hello";
4         System.out.println(greeting);
5         KeyStroke stroke = new KeyStroke();
6         stroke.typeExclamation();
7     }
8 }
```


Problema Ciclos Anidados



```
1  public static void main(String[] args) {  
2      int j=0, k=0;  
3      for (int i = 0; i < x; i++) {  
4          do {  
5              k=0;  
6              while (k<z){  
7                  k++;  
8                  System.out.print(k + "");  
9              }  
10             System.out.println("");  
11             j++;  
12         } while (j < y);  
13         System.out.println("---");  
14     }  
15 }
```

1. - int x = 2, y = 3, z = 4
2. - int x = 2, y = 3, z = 4
3. - int x = 2, y = 3, z = 4
4. - Doesn't Compile ✓✓

Problema de Parse



```
1 public class Align{
2     public static void main(String[] args) throws ParseException{
3         String[] sa = {"111.234", "222.5678"};
4
5         NumberFormat nf = NumberFormat.getInstance();
6         nf.setMaximumFractionDigits(3);
7
8         for (String s : sa) {
9             System.out.println(nf.parse(s));
10        }
11    }
12 }
```