

# ¿Compila o no Compila?

Poniendo a prueba mis conocimientos en Java



# Hola mundo!

## Soy Sarina Bolaños

Estoy aquí porque me gusta Java y quiero compartirlo contigo.



**Sarina20Lives**



**sarina20lives**

# Acerca de mi..

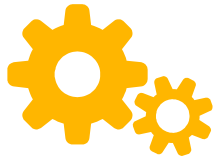
1. Actualmente estudio Ingeniería en Ciencias y Sistemas.
2. Laboré como auxiliar de cátedra en la Facultad de Ingeniería, USAC.
3. Trabajo como desarrolladora de software en Nabenik, utilizamos Java, Kotlin, JavaScript y Android.
4. Formo parte de StackOverLove



**1**

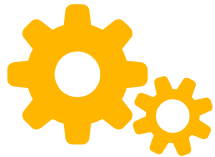
**Hagamos un test inicial...**

**¿Listos?**



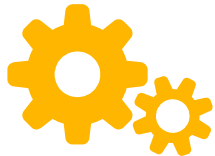
```
public static void firstQuestion() {  
    short a = 2 + 3 * 2;  
    short c = a + 12;  
    if(c > 20) {  
        System.out.println(c);  
    }  
}
```

- A.** No compila
- B.** 22
- C.** 20
- D.** Compila, no da error y no imprime nada



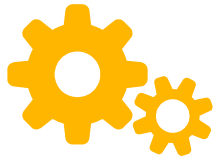
```
public static void secondQuestion(){  
    String var1 = "Hello";  
    String var2 = new String("Hello");  
    String var3 = var1;  
    System.out.println(var1 == var3);  
    System.out.println(var1.equal(var2));  
    System.out.println(var1 == var2);  
}
```

- A.** No compila por comparar dos String con ==
- B.** No compila por comparar dos String con equal
- C.** Si compila y se imprime: true, true, false
- D.** Si compila y se imprime: false, true, false



```
public static void thirdQuestion() {  
    boolean $$ = false;  
    if ($$=true) {  
        System.out.println("is true");  
    } else if (!$$) {  
        System.out.println("is false");  
    }  
}
```

- A.** No compila, identificador de variable inválido.
- B.** No compila, no posee sección else.
- C.** is true
- D.** Is false



```
public static void fourthQuestion() {  
    int a[ ], b = 0;  
    int[ ] _$a, _$b[] = new int[3][ ];  
    for(int i=0; i<_$b.length; i++)  
        _$b[i] = new int[2];  
    System.out.println(_$b.length + a.length);  
}
```

- A.** No compila, la declaración de variables no es válida
- B.** No compila, no se ha especificado la segunda dimensión del arreglo \_\$b
- C.** No compila por otra razón
- D.** 3null



# Declaraciones válidas

Identificadores válidos, declaraciones múltiples y declaración de arreglos.



# Identificadores

1. El nombre debe empezar con letra, \$ o \_
2. Los siguientes caracteres también pueden contener números
3. No se permite el uso de palabras reservadas como identificadores

## Advertencia:

Usar el símbolo `_` (únicamente) como identificador, podría no ser soportado en lanzamientos después de Java SE 8





NABENIK

## Recuerda:

Java es Case Sensitive, es sensible a mayúsculas y minúsculas.





# Declaraciones válidas

1. `int a[ ], b = 0;`

2. `int[ ] _$a, _$b[] = new int[3][ ];`

3. `String var1=""; int var2=9;`



# Declaraciones inválidas

1. `int a[ ], int b = 0;`
2. `int [ ] _$a, _$b = new int[3][];`
3. `String var1="", int var2=9;`

# Promoción numérica

Al sobrepasar el rango permitido de valores (según tamaño por tipo de dato) conlleva a una conversión con pérdida en los datos. Para evitar esto Java trabaja de la siguiente manera.

## Primer Regla:

Si se hace una operación entre dos variables numéricas de diferente tipo, Java automáticamente promueve el resultado al tipo con mayor tamaño.

## Ejemplo:

```
long var1 = 10;
```

```
int var2 = 2;
```

```
long var3 = var1 * var2;
```



```
int var4 = var1 * var2;
```





## Segunda Regla:

Si se operan un dato de tipo entero con un dato con punto flotante, Java promueve el valor entero a valor con punto flotante automáticamente.

## Ejemplo:

```
float var1 = 10.5f;  
int var2 = 2;  
double var3 = var1 * var2;  
float var4 = var1 * var2;
```



## Tercera Regla:

Tipos de datos pequeños como byte, short y char son promovidos a int cuando estos se encuentran dentro de alguna operación aritmética con operadores binarios aunque ninguno de ellos sea int.

## Ejemplo:

```
short var1 = 10;
```

```
short var2 = 2;
```

```
int var3 = var1 + var2;
```

```
short var4 = var1 + var2;
```



## Cuarta Regla:

Al terminar la operación, el resultado posee el mismo tipo de dato que el tipo de dato resultante de la promoción.

## Ejemplo:

```
short var1 = 10;  
short var2 = 2;  
float var3 = var1 + var2 + 2.4f;  
double var4 = var1 + var2 + 2.4f;  
short var5 = 5 + 20;
```



## Exception:

Al aplicar operadores unarios, la promoción no es aplicada automáticamente.

## Ejemplo:

```
short var1 = 10;  
short var2 = var1++;  
int var3 = var1++;
```





```
public static void firstQuestion() {  
    short a = 2 + 3 * 2;  
    short c = a + 12;  
    if(c > 20) {  
        System.out.println(c);  
    }  
}
```

## A. No compila

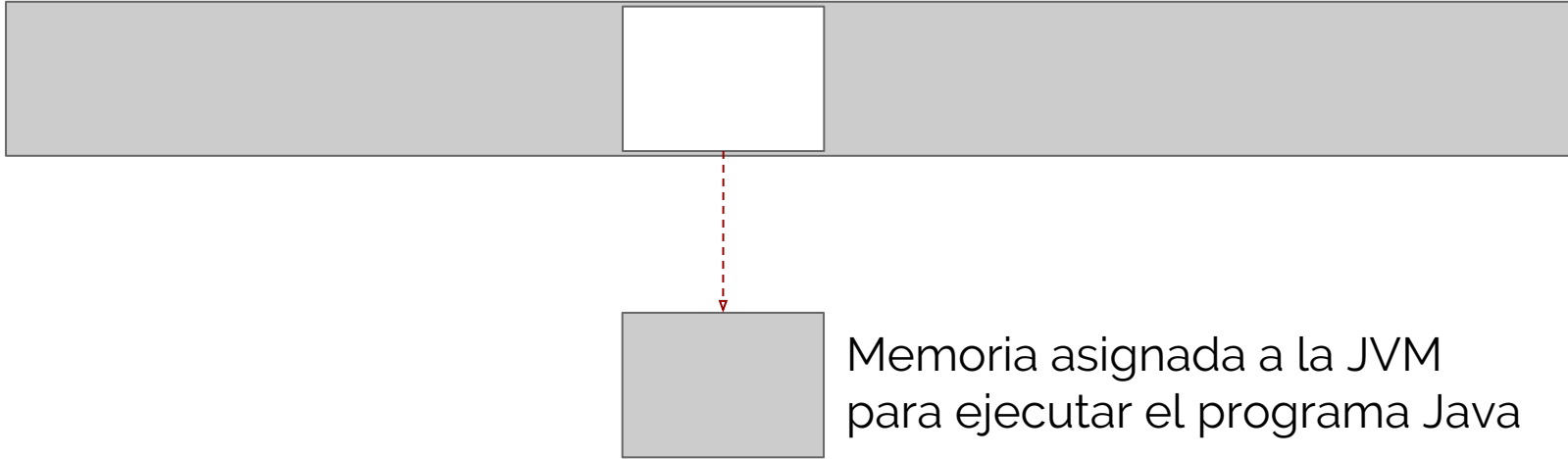
La promoción numérica me indica que al operar un short en una operación binaria es promovido a int, por tanto para poder asignar el resultado a un short es necesario realizar un casteo.

# String vs String Builder

Distribución de memoria en Java, Heap, Stack y String Pool,  
Inmutabilidad.

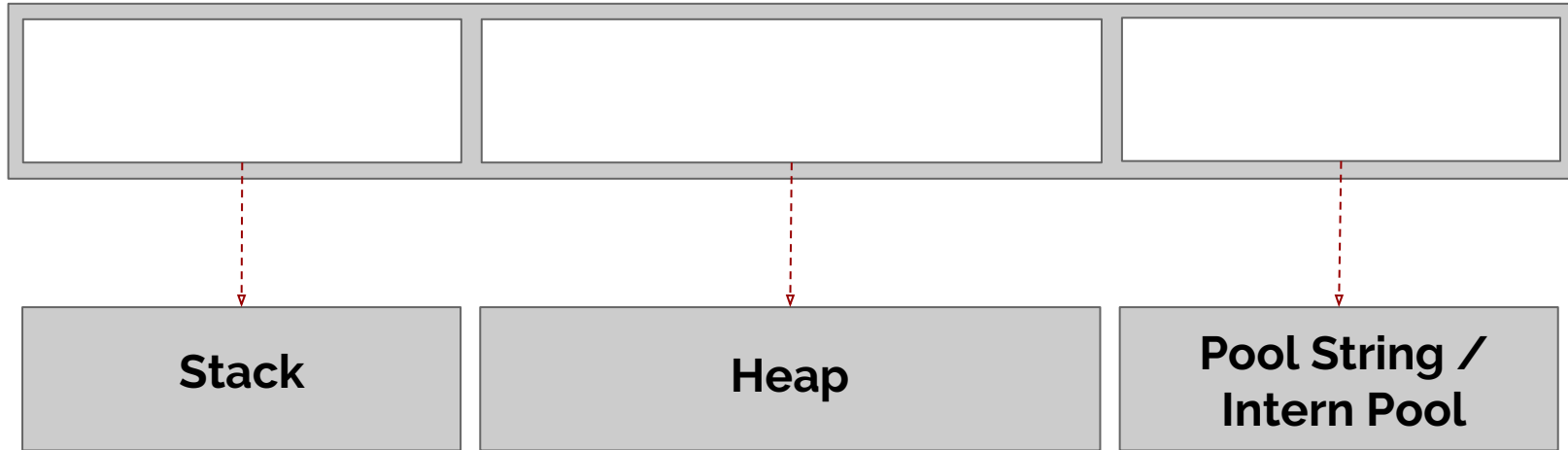
# Distribución de la memoria

Memoria disponible del computador:

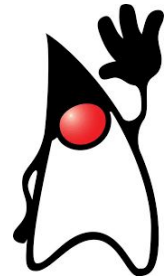


# Distribución de la memoria

Memoria asignada a la JVM para ejecutar el programa Java:







# String

Sucesión de caracteres.

```
String nickname = "esvux";
```

```
String nickname = new String("esvux");
```

Un String es **IMUTABLE**, la inmutabilidad es una propiedad que indica que el objeto una vez creado no puede cambiar.

```
String nickname = "esvux";
```

Se conoce como String Literal y son almacenadas en el String Pool.

```
String nickname = new String("esvux");
```

Usar el new String es decirle al compilador que no deseamos usar el String Pool, que queremos utilizar el Heap.

*"No, JVM. I really don't want you to use the string pool. Please create a new object for me even though it is less efficient" - OCA Study Guide*



NABENIK

```
String var1 = "0";  
for(int i= 0; i<10; i++){  
    var1 += i+1;  
}
```

```
System.out.println(var1);
```

out: 012345678910

Para lo cual se ha necesitado 11 instancias y 10 ya no serán usadas quedando disponibles para el Garbage Collection

```
String var1 = "0";  
String var2 = "01";  
String var3 = "012";  
String var4 = "0123";  
String var5 = "01234";  
  
...  
String var10 = "0123456789";  
String var11 = "012345678910";  
System.out.println(var11);  
out: 012345678910
```



NABENIK

```
1. String var1 = "ESVUX ";  
2. var1.trim().toLowerCase().replace("e", "E");  
3. System.out.println(var1);
```

out: ESVUX

No olvidemos que String es inmutable, por tanto var1 sigue apuntando a la misma sección de memoria, su valor original no se puede modificar.



# String Builder

Posee un arreglo con capacidad por defecto de 16 caracteres, se dice que String Builder no es inmutable.

```
StringBuilder sb1 = new StringBuilder();  
StringBuilder sb2 = new StringBuilder("esvux");  
StringBuilder sb3 = new StringBuilder(10);
```

```
StringBuilder var1 = new StringBuilder("0");  
for(int i= 0; i<10; i++){  
    var1.append(i+1);  
}  
System.out.println(var1);
```

out: 012345678910

Para lo cual se ha necesitado 1 instancia.

Posiciones ->	0	1	2	3	4	5	6	7	8	9	10	11	12	14	15
Valores ->	0	1	2	3	4	5	6	7	8	9	1	0			



```
public static void secondQuestion(){  
    String var1 = "Hello";  
    String var2 = new String("Hello");  
    String var3 = var1;  
    System.out.println(var1 == var3);  
    System.out.println(var1.equal(var2));  
    System.out.println(var1 == var2);  
}
```

## **B. No compila por comparar dos String con equal**

El método equal no está definido, debería de llamarse equals, si esta línea cambia, la respuesta sería la opción C. Compila e imprime true, true, false.



# Valores por defecto

Hablemos de valores por defecto en Java para Packages, Constructor, Herencia, Variables de instancia



NABENIK

## Package:

```
import java.lang.*;
```

<https://docs.oracle.com/javase/8/docs/api/java/lang/package-summary.html>

## Herencia:

Toda clase por defecto hereda de la clase Object y como tal, posee los métodos: equals(), toString(), hashCode(), etc.



NABENIK

## Constructor:

El constructor por defecto existe sin necesidad de declararse. A excepción cuando se establezca otro constructor.

```
String var1 = new String();
```

```
StringBuilder var2 = new StringBuilder();
```

```
MyClass var3 = new MyClass();
```



NABENIK

```
public class TestDefaultValues{  
    public static void main(String...args) {  
        Person person = new Person();  
        System.out.println(person);  
    }  
}
```

```
class Person{  
    int age;  
}
```

out: Person@6d06d69c



NABENIK

```
public class TestDefaultValues{  
    public static void main(String...args){  
        Person person = new Person();  
        System.out.println(person);  
    }  
}  
  
class Person{  
    int age;  
    public Person(int age){this.age = age;}  
}
```

error: constructor Person in class Person cannot be applied ...



NABENIK

## Variables de instancia:

Variable de instancia es toda aquella variable declarada a nivel de clase. Variables locales son aquellas que se declaran a nivel de método. Los valores por defecto para las **variables de instancia** según su tipo son:

## Variables de instancia:

<b>Tipo</b>	<b>Valor</b>
<code>boolean</code>	<code>false</code>
<code>byte, short, int, long</code>	<code>0</code>
<code>float, double</code>	<code>0.0</code>
<code>char</code>	
<code>Toda referencia de objeto</code>	<code>null</code>



```
public static void fourthQuestion() {  
    int a[ ], b = 0;  
    int[ ] _$a, _$b[] = new int[3][ ];  
    for(int i=0; i<_$b.length; i++)  
        _$b[i] = new int[2];  
    System.out.println(_$b.length + a.length);  
}
```

### C. No compila por otra razón

El arreglo con referencia "a" no está inicializado, es variable local y por tanto no posee un valor por defecto



# Extras

**Retorno de asignación y Garbage Collection**



NABENIK

## Retorno de una asignación:

En Java una asignación es una operación que también retorna su valor de asignación.

Ejemplo:

```
boolean var1 = false;  
boolean var2 = (var1 = true);  
System.out.println(var1 + "," + var2);
```

Out: true, true



NABENIK

## Garbage Collection:

Garbage Collection puede ser llamado con el método **System.gc()**, sin embargo es solo una sugerencia para Java de que es momento de ejecutar el GC, sin embargo **Java es libre de ignorarla.**

Cuando una variable deja de ser referenciada es candidata para ser eliminada por el GC, pero no es seguro afirmar que sea de manera inmediata.



```
public static void thirdQuestion() {  
    boolean $$ = false;  
    if ($$=true) {  
        System.out.println("is true");  
    } else if (!$$) {  
        System.out.println("is false");  
    }  
}
```

### C. is true

La variable **\$\$** empieza siendo false, sin embargo en la condición del if, su valor es cambiado a true y como la asignación retorna su valor, la sentencia if es ejecutada.

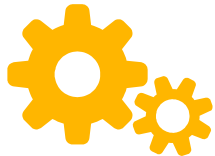


# Gracias!

Esta presentación así como el código de los ejemplos utilizados los puedes encontrar en el siguiente repositorio.

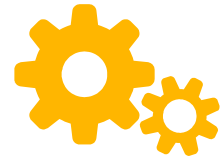


[goo.gl/P4wSe2](https://goo.gl/P4wSe2)



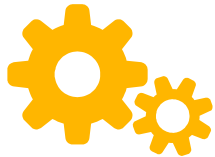
```
public class Exam{  
    private String title;  
    private boolean isEmpty;  
    public static void main(String...args) {  
        Exam exam = new Exam();  
        System.out.println(exam.title+" "+exam.isEmpty);  
    }  
}
```

- A. No compila, las variables no están inicializadas
- B. "", false
- C. **null, false**
- D. Compila, no da error y no imprime nada



¿Cuáles son identificadores válidos?

- A. A\$B
- B. \_helloWorld
- C. true
- D. java.lang
- E. Public
- F. 1980\_s

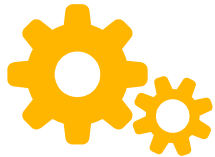


¿Qué tipo de dato es permitido en el siguiente trozo de código?

```
byte x = 5;  
byte y = 10;  
_____ z = x * y;
```

- A. int
- B. long
- C. boolean
- D. double
- E. short
- F. byte





¿Cuál es la salida del siguiente código?

```
int x1 = 50, x2 = 75;  
boolean b = x1 > x2;  
if(b = true) System.out.println("Success");  
else System.out.println("Failure");
```

- A. Success**
- B. Failure**
- C. El código no compila**
- D. El código compila pero no muestra nada**