

Variables de Tipo Referencia en Clases

Informática II

Variables primitivas

Las variables de tipo primitivas tienen un valor por default:

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

fuente:

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

Variables de tipo Referencia

Todas las variables de tipo referencia (objetos, arreglos, Strings) comienzan con un valor inicial de **null**.

¿Qué es null?

Null no es una dirección de memoria o un valor particular, **null** implica que la referencia simplemente está vacía.

El valor de null depende del procesador para el que esté compilado el código:

- **32 bits es 4 bytes**
- **64 bits es 8 bytes**

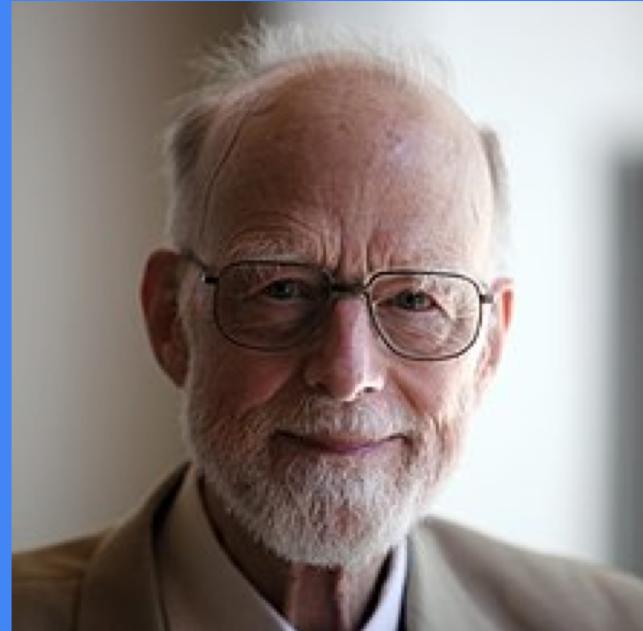
Variables de tipo referencia

A cualquier variable de tipo referencia se le puede hacer una asignación del valor **null** para indicar que la referencia aún no está inicializada.

Si intentamos acceder a algún método o variable de un objeto no inicializado, se generará la excepción **NullPointerException**.

```
public static void main(String[] args) {  
  
    Scanner s1 = null;  
    int x = s1.nextInt();  
  
}
```

```
run:  
[-]Exception in thread "main" java.lang.NullPointerException  
        at primitive_reference.Primitive_Reference.main(Primitive\_Reference.java:21)  
Java Result: 1  
BUILD SUCCESSFUL (total time: 0 seconds)
```



“Null references were created in 1964 - how much have they cost? (...)

This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.”

Sir Charles Anthony Richard Hoare

Variables de tipo referencia

Instanciar una variable con un valor vacío es distinto a inicializarlo con un valor de tipo nulo.

NOTA: null es una palabra reservada.

```
int[] c1 = null;  
int[] c2 = new int[0];  
  
if (c1 == c2) {  
    System.out.println("Iguales");  
} else {  
    System.out.println("Diferentes");  
}
```

run:

Diferentes

BUILD SUCCESSFUL (total time: 0 seconds)

Variable de tipo referencia

Es importante implementar validaciones de este tipo cuando utilizamos métodos.

Veamos los siguientes ejemplos:

```
public static void main(String[] args) {  
    int[] x = null;  
    printArray(x);  
}  
  
public static void printArray(int[] array) {  
    for(int i = 0; i<array.length; i++) {  
        System.out.println(array[i]);  
    }  
}
```

Al momento de intentar acceder a la variable **length** del objeto **array**, se levantará la excepción **NullPointerException**!

```
run:  
Exception in thread "main" java.lang.NullPointerException  
    at primitive_reference.Primitive_Reference.printArray(Primitive_Reference.java:25)  
    at primitive_reference.Primitive_Reference.main(Primitive_Reference.java:21)  
Java Result: 1  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Variable de tipo referencia

¿Cómo podemos evitarlo?

Respuesta: Incluyendo una validación de la validez del arreglo recibido.

```
public static void main(String[] args) {  
    int[] x = null;  
    printArray(x);  
}  
  
public static void printArray(int[] array) {  
  
    //check pointer validity  
    if (array == null){  
        System.out.println("Imposible imprimir.");  
        return;  
    }  
  
    for(int i = 0; i<array.length; i++) {  
        System.out.println(array[i]);  
    }  
}
```

Programación a la defensiva!

Ejercicio

Clase de Estudiantes

Queremos representar un salón de clases de estudiantes de preparatoria.

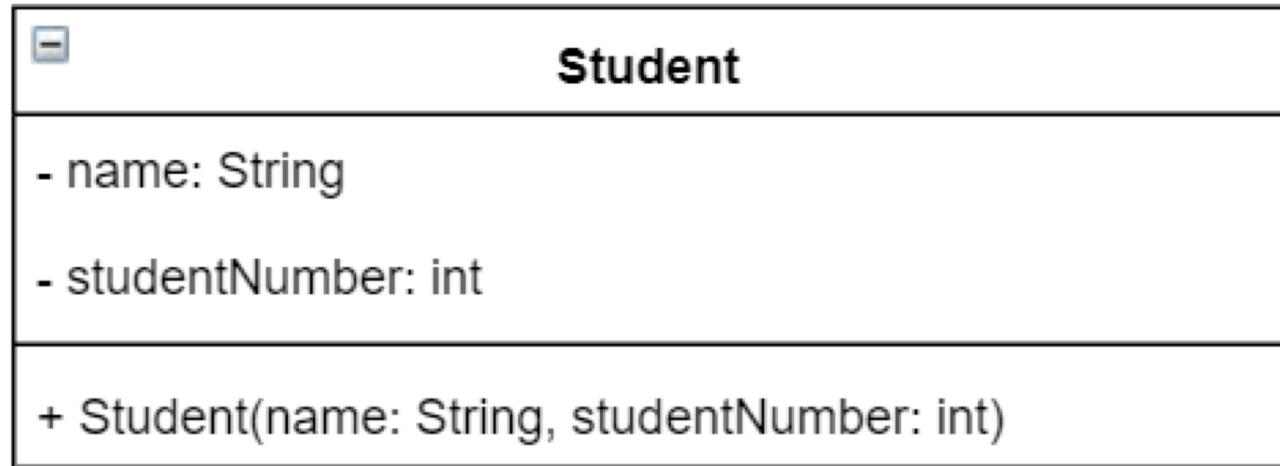
Cada estudiante deberá poder almacenar nombre y número de estudiante.

Cada salón de clases deberá poder almacenar un arreglo con los estudiantes inscritos, y el número de salón en donde se lleva a cabo la clase.

Ejercicio: Salón de Clases

Diseñamos una primera clase Student que pueda almacenar:

- Nombre
- Número de estudiante



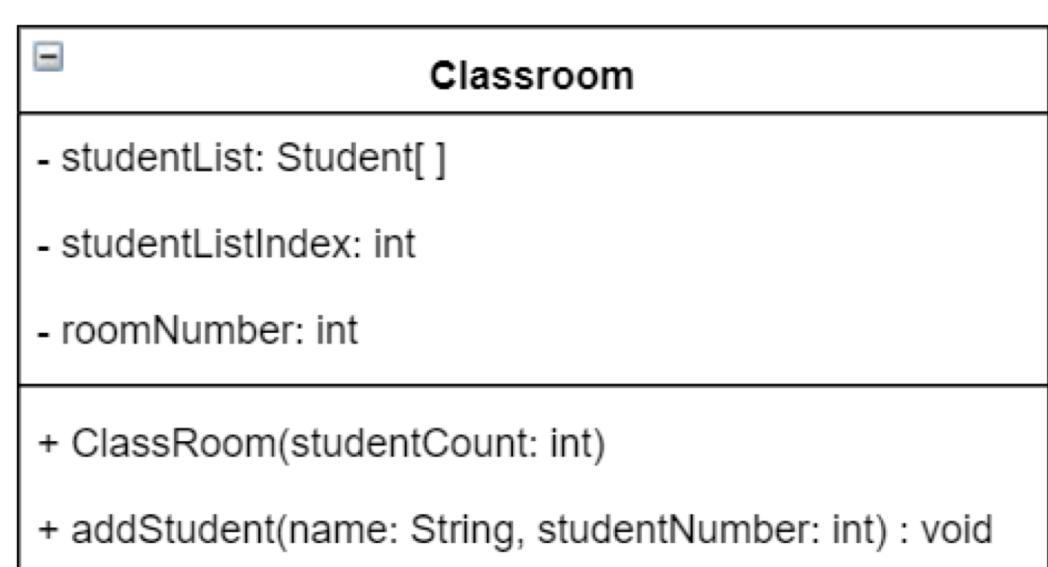
Ejercicio: Salón de Clases

```
public class Student {  
  
    private String name;  
    private int studentNumber;  
  
    public Student(String name, int studentNumber) {  
        this.name = name;  
        this.studentNumber = studentNumber;  
    }  
}
```

Ejercicio: Salón de Clases

Posteriormente diseñamos una segunda clase Classroom que pueda almacenar:

- Lista de Estudiantes
- Índice del último estudiante registrado en el arreglo
- Número de salón de clases



Ejercicio: Salón de Clases

```
public class Classroom{  
    private Student[] studentList;  
    private int studentListIndex;  
    private int roomNumber;  
  
    public Classroom(int studentCount, int roomNumber) {  
        if (studentCount <= 0 || roomNumber < 0) {  
            System.out.println("Error, studentCount (" + studentCount + ") invalido.");  
            System.exit(0);  
        }  
  
        this.studentList = new Student[studentCount];  
        this.studentListIndex = 0;  
        this.roomNumber = roomNumber;  
    }  
  
    public void addStudent(String name, int studentNumber){  
        //only insert student into the list if list has space for new student  
        if (this.studentListIndex < this.studentList.length ) {  
            this.studentList[this.studentListIndex] = new Student(name, studentNumber);  
            this.studentListIndex++;  
        } else {  
            System.out.println("No hay espacio para " + name + "!");  
        }  
    }  
}
```

Casos de Prueba

```
public static void main(String[] args) {  
  
    Classroom informatica = new Classroom(5, 123);  
    informatica.addStudent("Omar", 1234);  
    informatica.addStudent("Jose", 1234);  
    informatica.addStudent("Eduardo", 1234);  
    informatica.addStudent("Marco", 1234);  
    informatica.addStudent("Santiago", 1234);  
    informatica.addStudent("Pepe", 1234); //No hay espacio para Pepe!  
  
    Classroom matematicas = new Classroom(-5, 944); //Error!  
  
}
```

OUTPUT

No hay espacio para Pepe!
Error, studentCount (-5) invalido