## Universidad Internacional de Valencia (VIU)

15GIIN – Estructuras de Datos y Algoritmos

Primer Portafolio – ACT1

**Alumno:** Gagliardo Miguel Angel

## **Ejercicio 1**

En el codigo del método **findMax()** podemos ver que declara un retorno de tipo **AnyType** (clase genérica de Java):

## public static <AnyType> AnyType findMax(AnyType [] a) {

El problema es que podemos ver que en el for loop que implementa el método, compara el elemento **a[i]** del array con **a[maxIndex]**, usando **compareTo** para verificar que devuelva 0, que significa que son iguales.

El problema es que el tipo **AnyType** (generic) no implementa **Comparable** comparable, por tanto es imposible que el programa siquiera compile, por eso veremos el mensaje de error:

Exception in thread "main" java.lang.Error: Unresolved compilation problem: The method compareTo(AnyType) is undefined for the type AnyType

Por tanto, para que este programa funcione, tenemos como opciones:

1) Implementar Comparable en AnyType, por tanto en la definición del método:

public static <AnyType extends Comparable<AnyType>> AnyType findMax(AnyType[] a) {

Con esto, AnyType puede ahora hacer **compareTo**, el resto del método se mantiene igual.

2) Otra opción es aprovechar el uso de **overloading**. El código del método se mantiene idéntico pero la declaración cambia, entonces tendremos:

Para type Integer:

public static Integer findMax(Integer[] a) {

Para type String:

public static String findMax(String[] a) {

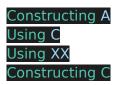
Para type Double:

public static Double findMax(Double[] a) {

Si bien es verdad que hacer **overloading** es repetir código, la declaración del mismo es mucho mas prolija y legible.

## Ejercicio 2

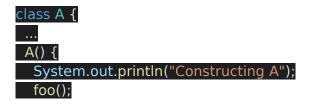
El output de este ejercicio es el siguiente:



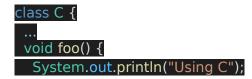
La razón es que la clase **Prueba** genera una instancia de la clase **C y la misma hereda de A.**Aunque no estén explicitamente declarados, todos los constructores llaman por defecto al constructor de la clase superior a través de una llamada a **super()**. Esto es debido a que los constructores no se heredan entre jerarquías de clases. Por lo tanto, la palabra **super()** siempre es la primera línea de un constructor e invoca al **constructor de la clase superior**.

Por tanto la explicación es:

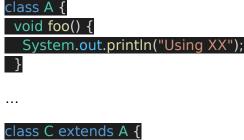
- 1. Problema → Crea una instancia de C
- 2. Constructor de **C por defecto y sin ser declarado llama a super()**, que es el constructor de su clase superior **A**
- 3. A tiene un constructor declarado que tiene un mensaje para imprimir por pantalla el primer mensaje que vemos: Constructing A



4. Dado que **A** en su constructor luego llama al método **foo()**, que también esta implementado en la **clase C** primero vemos el mensaje: **Using C** 



5. Luego vemos el mensaje Using XX porque en el constructor de C estamos llamando a super.foo(), donde super es una palabra reservada para llamar al método de la clase heredada, o sea estamos llamando al método foo() de la clase A:



class C extends A {
 C() {
 super.foo();

6. El ultimo mensaje que veremos es  $\frac{\text{Constructing C}}{\text{de la clase C}}$  que es el que se imprime en el constructor de la  $\frac{\text{Clase C}}{\text{Clase C}}$ 

class C extends A {
 C() {
 super.foo();
 System.out.println("Constructing C");
}