LENGUAJES DE PROGRAMACIÓN

AYUDANTÍA N°3

Álvaro Hernández I.

Clase Abstracta

- Tiene al menos un método abstracto.
- Un método abstracto no presenta implementación
- La idea es que otra clase la extienda e implemente sus métodos.
- No puede ser instanciada

```
abstract class MiClaseAbstracta {
    public int Suma(int Num1, int Num2) {
            return Num1 + Num2;
     }
     public abstract int Multiplicacion(int Num1, int Num2);
}
```

INTERFACES

Solo contiene métodos abstractos

```
interface iMiInterfaz {
        int Sumar(int Num1, int Num2);
        int Multiplicar(int Num1, int Num2);
}
```

Clase abstracta v/s Interfaz

- No pueden ser instanciadas
- La primera puede o no tener métodos no implementados (abstractos)
- La segunda es completamente abstracta

- Una clase puede implementar una clase abstracta a la vez
- Mientras que puede implementar varias interfaces.

- Ambas son ampliamente utilizadas por librerías externas para establecer el uso de componentes de esta, que no están completamente definidos, resultando más práctico el uso de clases abstractas
- Cuando el nivel de abstracción es mayor, por ejemplo para patrones de diseño, se deben utilizar interfaces (con la misma finalidad).

Polimorfismo en Java

- Patrones de Diseño (más adelante)

- A través de la instanciación de dos objetos de la misma clase padre pero distinta clase hijo.
- Ambos subclases deben definir el mismo método pero con comportamientos distintos
- De manera de abstraer (UpCast) los objetos hacia la clase padre y ejecutar el mismo método en ambos, obteniendo distintos resultados.

UPCAST & DOWNCAST

```
Cat c1 = new Cat();
Animal a = c1; //automatic upcasting to Animal
Cat c2 = (Cat) a; //manual downcasting back to a Cat
```

- La utilidad de ambos radica principalmente en:
- *Hacer coincidir el tipo de dato a ser utilizado como parámetro en alguna función u otro*
  - Por ejemplo:
    - **Upcast** comúnmente se utiliza para llenar listas con el mismo “tipo” de objeto
    - **Downcast** no es utilizado comúnmente, pero es muy útil para acceder todos los métodos y atributos de la subclase (public), que no estén definidos en el padre

Es importante considerar que no es un cast común, no se cambia el tipo de objeto, si no más bien su “etiqueta”.

PATRONES DE DISEÑO

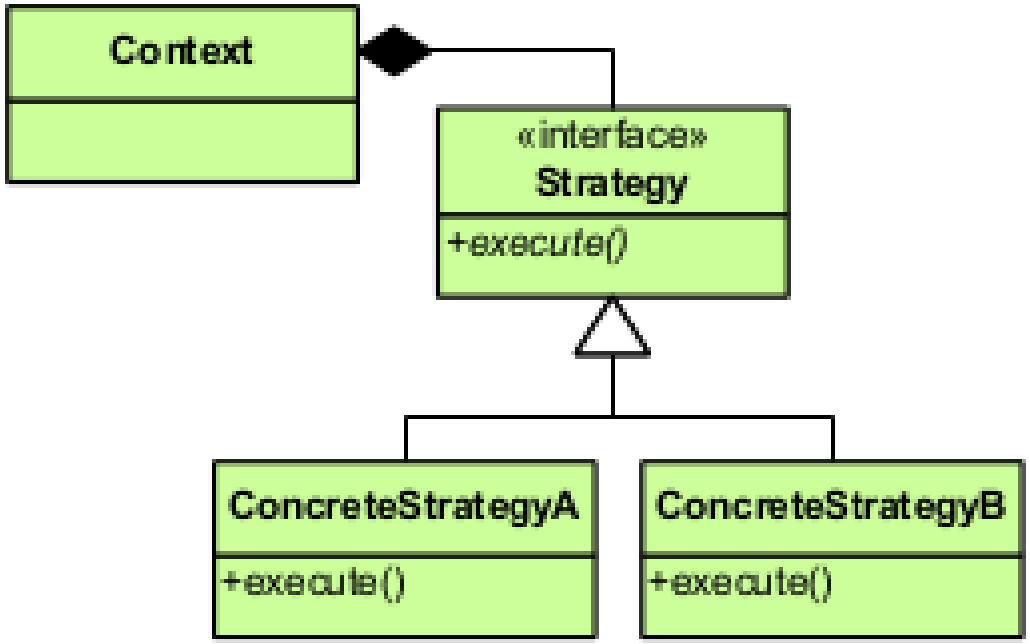
"Los **patrones de diseño** son un esqueleto de soluciones a problemas comunes en el desarrollo de software."

Fábrica Abstracta (*Abstract Factory*)
Creación de diferentes familias de objetos

Estrategia (*Strategy*):
Utilizado para manejar la selección de un algoritmo.

Singleton
Restringe la instanciación de una clase o valor de un tipo a un solo objeto.

STRATEGY PATTERN



```java
import java.io.IOException;
/** The classes that implement a concrete strategy should implement this.
* The Context class uses this to call the concrete strategy. */
interface Strategy {
    int execute(int a, int b);
}

/** Implements the algorithm using the strategy interface */
class Add implements Strategy {
    public int execute(int a, int b) {
        System.out.println("Called Add's execute()");
        return a + b;  // Do an addition with a and b
    }
}

class Subtract implements Strategy {
    public int execute(int a, int b) {
        System.out.println("Called Subtract's execute()");
        return a - b;  // Do a subtraction with a and b
    }
}

class Multiply implements Strategy {
    public int execute(int a, int b) {
        System.out.println("Called Multiply's execute()");
        return a * b;   // Do a multiplication with a and b
    }
}
```

```java
// Configured with a ConcreteStrategy object and maintains
// a reference to a Strategy object
class Context {
    private Strategy strategy;

    public Context(Strategy strategy) {
        this.strategy = strategy;
    }

    public int executeStrategy(int a, int b) {
        return this.strategy.execute(a, b);
    }
}

/** Tests the pattern */
class StrategyExample {
    public static void main(String[] args) {
        Context context;

        // Three contexts following different strategies
        context = new Context(new Add());
        int resultA = context.executeStrategy(3,4);

        context = new Context(new Subtract());
        int resultB = context.executeStrategy(3,4);

        context = new Context(new Multiply());
        int resultC = context.executeStrategy(3,4);

        System.out.println("Result A : " + resultA );
        System.out.println("Result B : " + resultB );
        System.out.println("Result C : " + resultC );
    }
}
```

PASO POR VALOR Y POR REFERENCIA

Java:
Los datos primitivos se pasan por valor
La referencia a los objetos se pasa por valor

C:
Por defecto se realiza por valor.
Para pasar por referencia se debe utilizar & en el argumento al llamar la funció, y * para recibirlo.

STATIC

```
class UnicaInstancia {
   private static UnicaInstancia instancia;

   static {
      instancia=new UnicaInstancia();
   }

   public UnicaInstancia() {}

   public static UnicaInstancia getInstancia() {
      return instancia;
   }
}
```

STATIC

```
class Ave{
   private static int numero;

   public static UnicaInstancia getInstancia() {
      return numero;
   }
}
```

FINAL. FINALIZE

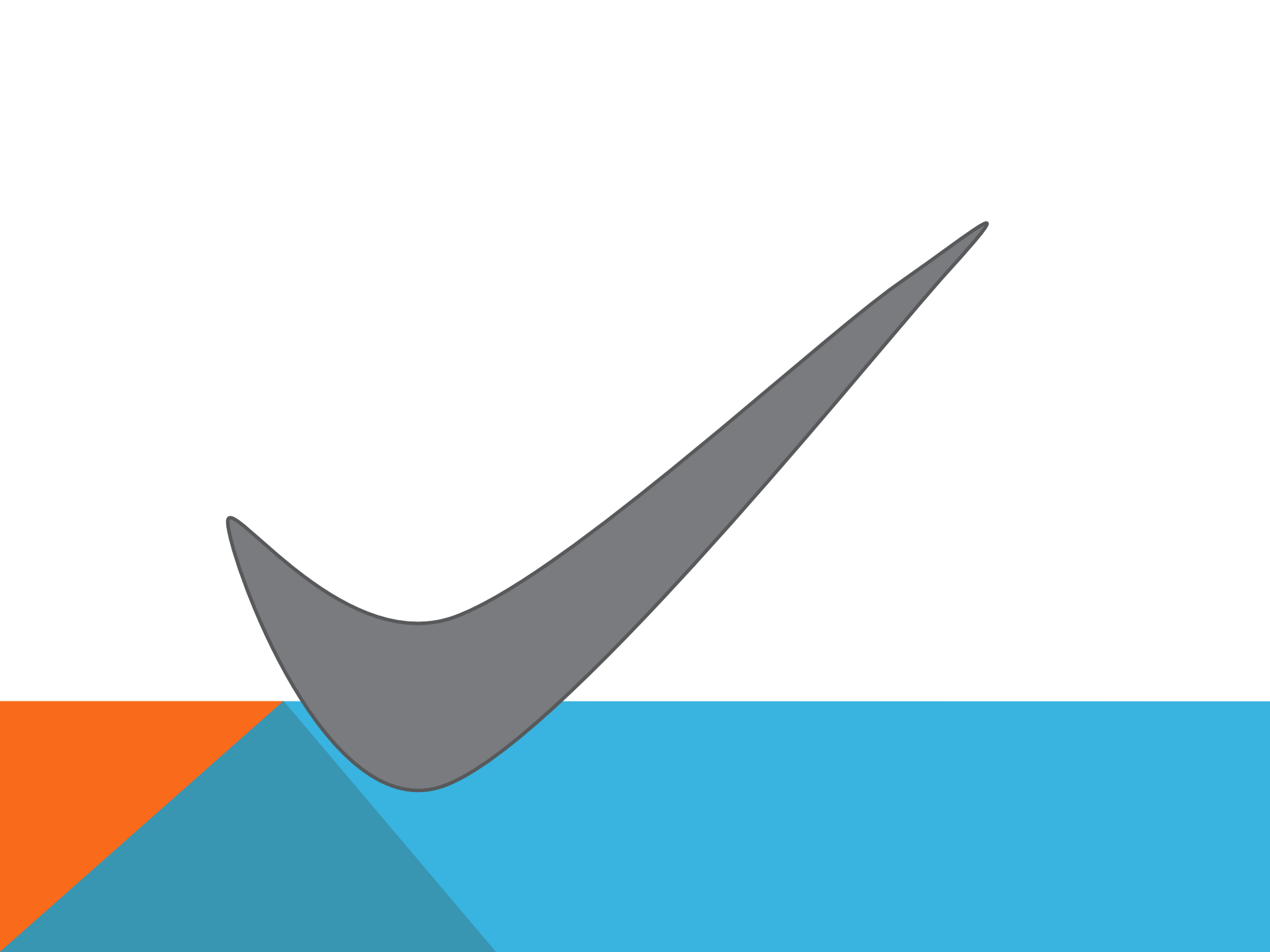
No puede ser reasignada

```
class Clase{
        public final int x = 100;

        public int getX(){
                return this.x;
        }

        protected void finalize(){
                System.out.println("x = " + x);
        }
}
```

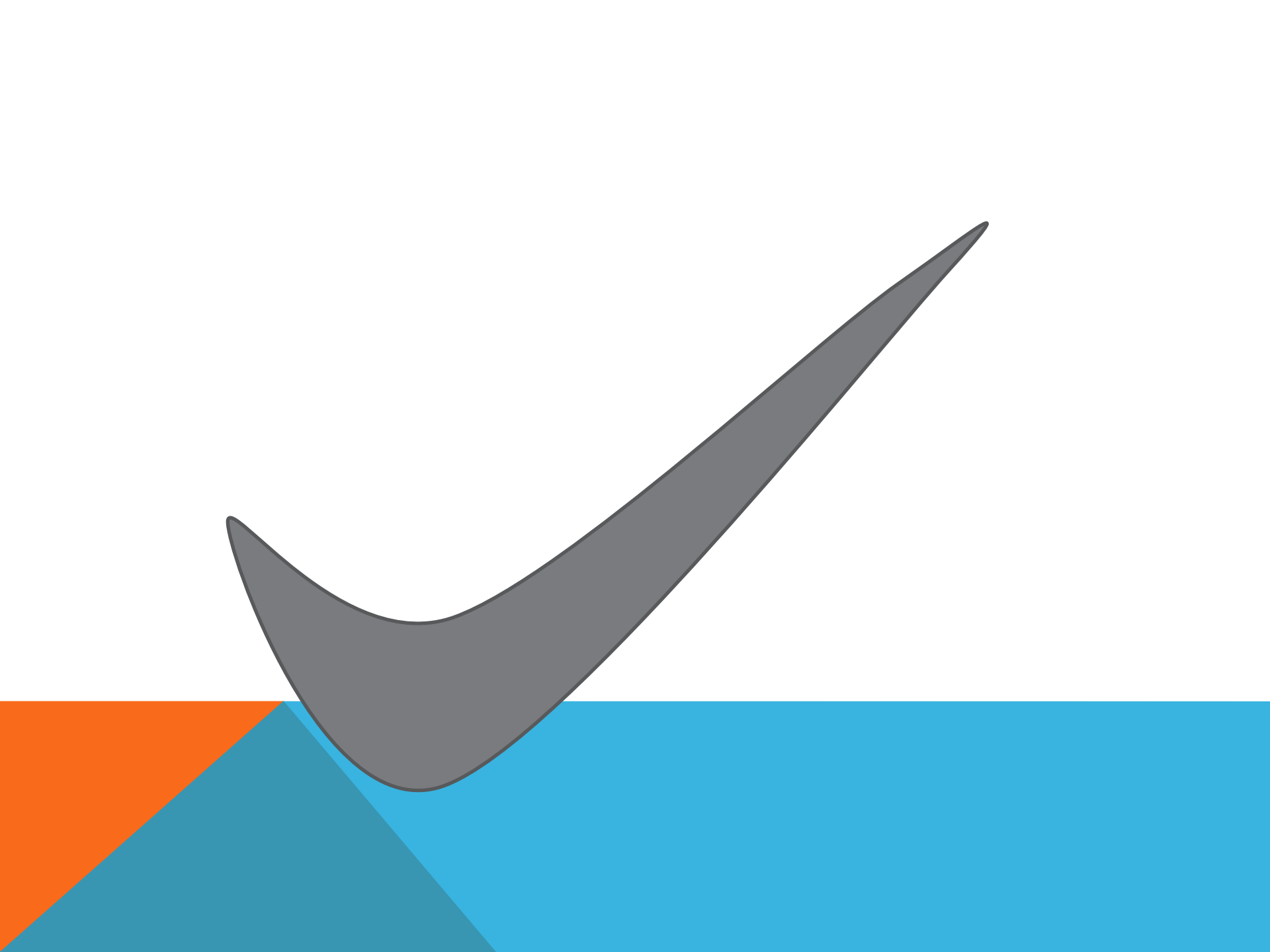
Llamado por el garbage colector, heredado de Object

AL TENER UNA SUBCLASE CON LA MISMA FIRMA DEL MÉTODO DE LA SUPERCLASE PODEMOS DECIR QUE ESTAMOS ANTE UN CASO DE SOBRECARGA DE MÉTODOS.

EN JAVA UNA O MÁS CLASES PUEDEN IMPLEMENTAR UNA MISMA INTERFAZ, ESTABLECIENDO UN CONTRATO DE PROGRAMACIÓN SIN REVELAR DETALLES DE LA IMPLEMENTACIÓN.

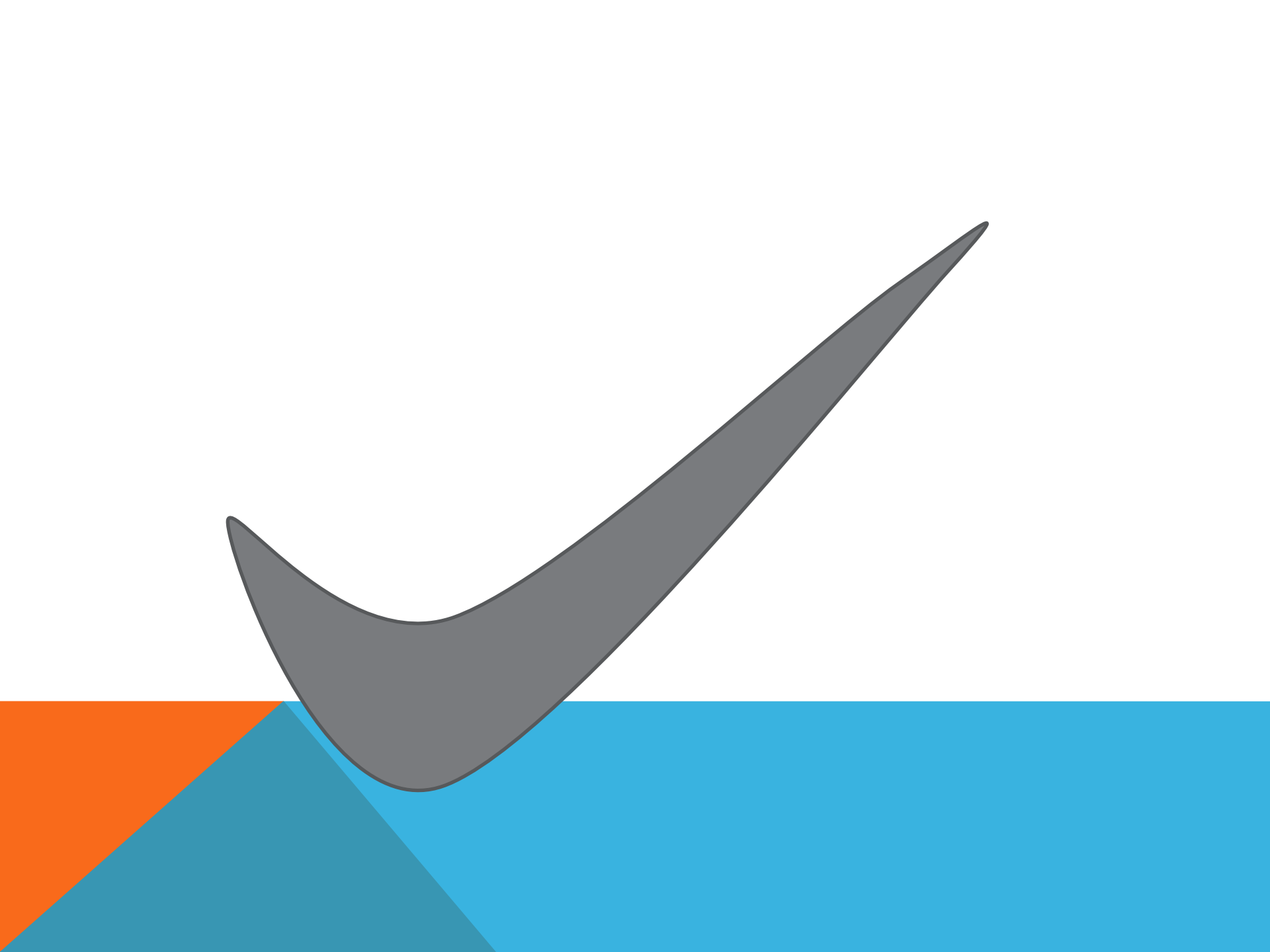


LOS ÍNDICES DE ARREGLOS EN JAVA SIEMPRE PARTEN DE 0.



UNA VARIABLE DE INSTANCIA DECLARADA EN UNA CLASE ES CREADA LA PRIMERA VEZ EN CUANDO UNA INSTANCIA DE LA CLASE ES ACCESADA.

UNA VARIABLE DE INSTANCIA DECLARADA EN UNA CLASE ES CREADA CUANDO UNA INSTANCIA DE LA CLASE ES CREADA.



CUANDO UN OBJETO ES CREADO, NO HAY CASOS DE VARIABLES DE INSTANCIAS QUE AL NO TENER ASIGNADO UN VALOR DECLARADO SIGUEN SIENDO INDEFINIDO.

Dado el siguiente código en Java,
¿cómo se debería invocar el constructor Base para que muestre por la salida estándar el *string* "base constructor"?

```
class Base{
    Base(int i){
        System.out.println("base constructor");
    }
    Base(){
    }
}
public class Sup extends Base{
    public static void main(String argv[]){
        Sup s= new Sup();
        //Uno
    }
    Sup()
    {
        //Dos
    }
    public void derivada()
    {
        //Tres
    }
}
```

Indique la salida por pantalla de la ejecución del siguiente bloque de código:

```
public class eReader {
        public static int a = 3;
        public int b = 4;
        public int getData(int a){
          return this.a;
         }
public static void main(String[] args) {
        int a = 1; int b = 2;
        eReader unObjeto = new eReader();
        eReader otroObjeto = new eReader();
        unObjeto.a += 1;   unObjeto.b += 1;
        otroObjeto.a += 2;   otroObjeto.b += 3;
        a = a + unObjeto.getData(a);
        b = b + otroObjeto.getData(b);
        System.out.println("a: "+ a + " b:" + b );
        }
}
```

Qué ocurre con el siguiente bloque de código:

```
public class NFE {
    public static void main(String [] args)     {
    String s = "42";
        try    {
            s = s.concat(".5");  /* Line 8 */
            double d = Double.parseDouble(s);
            s = Double.toString(d);
            int x = (int) Math.ceil(Double.valueOf(s).doubleValue());
            System.out.println(x);
        }
        catch (NumberFormatException e)        {
            System.out.println("numero incorrecto");
        }
    }
}
```

Qué ocurre con el siguiente bloque de código:

```
class Arbol { }
class Pino extends Arbol { }
class Planta extends Arbol { }
public class Bosque1 {
    public static void main (String [] args)    {
        Arbol arbol = new Pino();
        if( arbol instanceof Pino )
            System.out.println ("Pino");
        else if( arbol instanceof Arbol )
            System.out.println ("Arbol");
        else if( arbol instanceof Planta )
            System.out.println ( "Planta" );
        else
            System.out.println ("Oops ");
    }
}
```