CICLO 01

[FORMACIÓN POR CICLOS]

Fundamentos de Programación

Clase derivada de la clase vector

Ingeni@ Soluciones TIC

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Programación orientada a objetos

Clases derivadas

Herencia

La definición en Python de nuestra clase *altaPrecision* es:

```
class    altaPrecision(vector):
    def __init__(self, n):
        vector.__init__(self, n)
        self.V[0] = n
```

La diferencia básica con respecto al constructor de la clase base (la clase vector) es que el dato de la posición 0 es n.

Los datos en la clase derivada los manejamos ajustados a la derecha.

Si queremos representar el número 31628476977 en un vector de 20 elementos, la representación es:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
V	9										3	1	6	2	8	4	7	6	9	7	7

n = 20

El dato en la posición 0 indica que el número que se está representando, su primer dígito significativo está en la posición V[0]+1

Se representa el número de a dígito por posición, quedando el dígito menos significativo en la posición n.

Teniendo definido el constructor procedemos a definir los métodos propios de la clase altaPrecision:
sumar, restar, multiplicar, dividir, etc.

Analicemos aquí el método para mostrar los datos del vector.

Recordemos el método para mostrar los datos de un vector

```
def   imprimeVector(self, mensaje="vector sin nombre: "):
        print("\n", mensaje, end=" ")
        for i in range(1, self.V[0]+1):
              print(self.V[i], end=", ")
        print()
```

Consideremos ahora el método para mostrar un número de *altaPrecision*

```
def   imprimeVector(self, mensaje="vector sin nombre: "):
        print("\n", mensaje)
        for  i  in range(self.V[0]+1, self.n+1):
              print(self.V[i], end = "")
        print()
```

La diferencia entre ambos métodos es la forma de recorrer el arreglo:
en la clase vector se recorre desde 1 hasta V[0] inclusive.
en la clase altaPrecision se recorre desde V[0]+1 hasta n inclusive

Ambos métodos tienen el mismo nombre.
El uno pertenece a la clase base, el otro a la clase derivada.

Esto se denomina **polimorfismo dinámico**.

El programa, en tiempo de ejecución, identifica cuál de los dos métodos es el que debe ejecutar.

Lo hace dependiendo del objeto que invoque el método.

Recordemos el parámetro "mensaje":

Si la llamada a imprimeVector es: **vec.imprimeVector()**
escribirá el mensaje: "vector sin nombre"

Si la llamada a imprimeVector es: **vec.imprimeVector("Vector de prueba")**
escribirá el mensaje: "Vector de prueba"