T- POO

Contenido

6

1. Recordatorio:

- Una clase sirve para instanciar objetos, es decir para crear objetos de esa clase.
- Los atributos de una clase serán privados, accesibles a través de los métodos set (para asignarle valor) y get (para recuperar el valor)
- Una clase tendrá un método constructor para crear nuevas instancias y un destructor para eliminarlas.

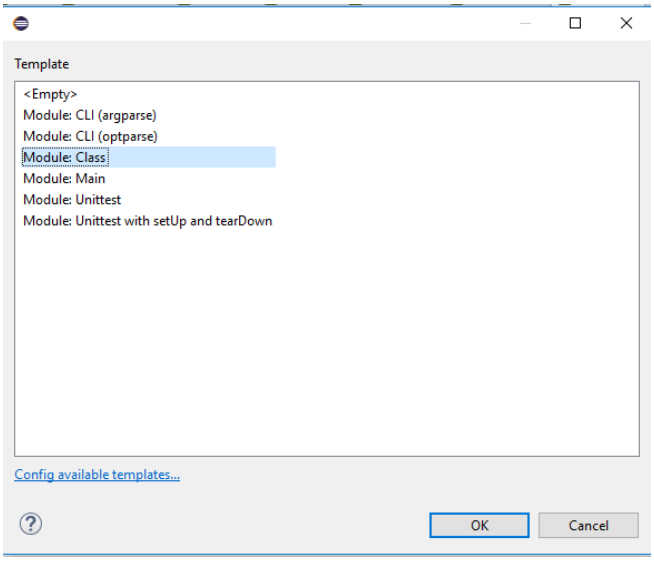
Plantilla Definición clase:

```
Clase
        Atributo1
        Atributo2
        ……

        Constructor
        Destructor
        Métodos get y set por atributo.
```

2. Python

Al crear con Eclipse, indicar que va a ser una Class



2.1 Definicion de una clase:

class Nombreclase:

El nombre de la clase con la Inicial en mayúsculas.

2.2 Constructor:

No existe el constructor vacío. Cuando se crea un objeto se le pasan todos los valores

a) Definición

```
def__init__(self, argumento1, argumento2, argumento3,…):
        self.atributo1=argumento1 #Atributo publico
        self.atributo2=argumento2 #Atributo publico
        self.__atributo3= argumento3 #Atributo privado con 2 guiones
delante.
        ……
```

b) Invocación:

```
NombreObjeto=Nombreclase  (argumento1, argumento2, argumento3,…)
```

2.3 Metodos get y set por cada atributo:

a) Definición

```
def getAtributo1(self):
        return self.atributo1
def setAtributo1(self,valor)
        self.atributo1=valor
```

b) Invocación

```
Objeto.getAtributo1()

Objeto.setAtributo1(valor)
```

NOTA: Python permite el acceso a atributos privados sin necesidad de utilizar el método get. Se hace con la sentencia: Objeto.NombreCLase_atributoprivado

2.4 Destructor

Python incorpora un recolector de basura (garbage collector) que se encarga de llamar a esté destructor y liberar memoria cuándo el objeto no es necesario. Este proceso es transparente y no es necesario invocar al destructor para liberar memoria. Sin embargo, esté método destructor puede realizar acciones adicionales si lo sobreescribimos en nuestras clases. Su nombre es __del__() y, habitualmente, nunca se le llama explícitamente.

Debemos tener en cuenta que la función integrada del() no llama directamente al destructor de un objeto, sino que está sirve para decrementar el contador de referencias al objeto. Por otro lado, el método especial __del__() es invocado cuándo el contador de referencias es igual a cero

a) Definición:
   No hace falta definirlo, pero si se quiere puede hacerse:

```
def __del__(self): #Destructor
            print "Objeto borrado"
```

b) Invocación (esté o no definido el destructor en la clase):

```
del Objeto
```

3. Herencia

La herencia se define al lado del nombre de la clase entre corchetes

Class NombreClase(ClasePadre)

A la hora de llamar al constructor, la propia clase deberá llamar al constructor padre de la siguiente manera:

ClasePadre.__init__(self, argumentos)

Tendrá métodos get y set solamente para los atributos propios, para los atributos que hereda del padre, hereda también sus métodos. Lo que sí se podría escribir sería un método que sobrescriba un método del padre (no que sobrecargue, ya que no está permitida la sobrecarga, utiliza el último método definido con el mismo nombre, no mira los parámetros).

3.1 Herencia Múltiple

a) Definición:
```
class Clase3 (Clase1, Clase2)
```

b) Constructor

```
def __init__(self, *lista):
        '''
        Constructor
        '''
        self.__at1=lista[0]
        Clase1.__init__(self, lista[1], lista[2]) #Llamada constructor
Clase1
        Clase2.__init__(self, lista[3], lista[4], lista[5]) #Llamada
constructor Clase2
```

c) Atributos con mismo nombre en las clases Padre.
Python no sabe distinguir entre dos atributos con el mismo nombre (Si Clase1 y Clase2 tienen ambos atributos llamados atributo1 y atributo2).
Elige el atributo que sea el último en inicializarse, en el ejemplo el de Clase2.
Si cambio la llamada de los constructores de orden:

```
def __init__(self, *lista):
        '''
        Constructor
        '''
        self.__at1=lista[0]
        Clase2.__init__(self, lista[3], lista[4], lista[5]) #Llamada
constructor Clase2

        Clase1.__init__(self, lista[1], lista[2]) #Llamada constructor
      Clase1
```

el que devolvería sería el de Clase1

4. Ejemplo Python

- *Módulo Clase.py*

```
class Clase:

    def __init__(self,arg1,arg2,arg3):#Constructor
        self.__atributo1=arg1 #Atributos precedidos con __ indican que son
privados
        self.__atributo2=arg2
        self.__atributo3=arg3
        self.__atributo4=5
        self.__atributo5=''
        self.atributo6="publico" #atributo publico

    def getaributo1(self):
        return (self.__atributo1)
```

```
    def setatributo1(self,valor):
        self.__atributo1=valor

    def getaributo2(self):
        return (self.__atributo2)
    def setatributo2(self,valor):
        self.__atributo2=valor

    def getaributo3(self):
        return (self.__atributo3)
    def setatributo3(self,valor):
        self.__atributo3=valor

    def getaributo4(self):
        return (self.__atributo4)
    def setatributo4(self,valor):
        self.__atributo4=valor

    def getaributo5(self):
        return (self.__atributo5)
    def setatributo5(self,valor):
        self.__atributo5=valor
```

- *Módulo Herencia: hereda de Clase*

```
class Herencia(Clase):
    def __init__(self,params,arg1,arg2,arg3):
        self.__atributoN=params #Atributo propio de esta clase
        Clase.__init__(self,arg1,arg2,arg3) #Llamada al constructor padre

    def getatributoN(self):
        return self.__atributoN
    def setatributoN(self,valor):
        self.__atributoN=9999

    #Aquí no están definidos más métodos get y set porque los hereda de la
    clase padre.

    def setatributo4(self): #Método que sobreescribe el método setatributo4
    que heredaría del padre.
        self.__atributo4=”VALOR 4 en hijo”
```

- *Módulo Main.py*

```
from CrearClase import *

if __name__ == '__main__':
    Nueva=Clase(3,4,5) #Llamada al constructor

    print Nueva.getaributo1()
    Nueva.setatributo1(6)
    print Nueva.getaributo1()
    print "Atributo público-valor:",Nueva.atributo6
    # print Nueva.__atributo1 <-- Sentencia errónea. El atributo no es
    público, no es accesible
    #Acceso a un atributo privado si utilizar el método get()
    print Nueva._Clase_atributo4
```

```
del Nueva
    #print Nueva.getaributo1() <-- Sentencia errónea. Ha sido borrado el
objeto


#HERENCIA
NuevaHerencia=Herencia(8888,9,7,6)
print NuevaHerencia.getatributo1()
print NuevaHerencia.getatributoN()
#Cuando el programa acaba el garbage collector elimina los objetos.
```

- Ejecución de Main:

```
3
6
Atributo público-valor: público
5
Objeto borrado
9
8888
Objeto borrado
```

5. Métodos especiales en las clases

- `def __str__(self): # Método para representar el objeto como una cadena. Será llamado al hacer print (Objeto)`
- `def __cmp__ (self,objeto2): # Método para comparar 2 objetos. Será llamado al utilizar el operador ==`
- `def __len__(self): # Método para determinar la longitud de la clase. Será llamado al hacer len(Objeto)`
- `Objeto1 is Objeto2  # Operador que indica si son el mismo objeto.`

Ejemplos:

- Definición de los métodos en la clase:

```
#Métodos especiales
    def __str__(self):
    # Metodo para representar el objeto como una cadena. Será llamado
    al hacer print (Objeto)

        cadena=self.__atributo1,self.__atributo5,self.atributo6

        #cadena2=self.atributo6 --> Si lo que se devolviera fuera un
        atributo que ya es string, no haría falta utilizar str
        #return cadena2

        return str(cadena) #Hay que asegurar que lo que devuelve este
        método sea una cadena. Utilizar método str.


    def __cmp__ (self,objeto2):
    # Método para comparar 2 objetos. Será llamado al utilizar el
    operador ==

        if self.__atributo4==objeto2.__atributo4:
```

```
            return 0 #Cuando la condicion de comparación se cumple
            SIEMPRE return 0
        else:return 1


    def __len__(self):
    # Método para determinar la longitud de la clase. Será llamado al
    hacer len(Objeto)
        return (self.__atributo2)
```

- Llamada de estos métodos desde el main

```
print "\r METODOS ESPECIALES".center(100)
    Una=Clase(1,4,5) #Llamada al constructor
    Dos=Clase(2,8,9) #Llamada al constructor
    if Una==Dos: #Llamada al metodo __cmp__. Si retorna 0, significa
    que se cumple.
        print "Iguales"
    else: print "Distintos"

    print len(Dos) # Llamada al método __len__(self)
    print Una #Llamada al método __str__(self)

    print Una is Dos # Comprueba si el objeto es el mismo. Son
    diferentes → False

    Tres=Una
     print Una is Tres # Es el mismo objeto, tan sólo es una referencia
      a él→ True

    Cuatro=Clase(4,4,5) # Crea un nuevo objeto que es igualito a Una,
    tiene los mismos valores.
    print Una is Cuatro # Como es otro objeto, la respuesta es False.
```

- Resultado de la ejecución de esta parte en main:

```
METODOS ESPECIALES
Iguales
8
(1, '', 'p\xc3\xbablico')
False
True
False
```