UNCUYO UNIVERSIDAD NACIONAL DE CUYO

FACULTAD DE INGENIERÍA

Programación Orientada a Objetos

Herencia y Polimorfismo

Esp. Ing. César Aranda

cesar.aranda@ingenieria.uncuyo.edu.ar

Ingeniería en Mecatrónica

Objetivos y contenidos

- Conocer la sintaxis de los lenguajes C++ y Python relacionados con la Herencia y el Polimorfismo
- Discutir aspectos básicos de implementación para manejar relaciones de herencia y mecanismos polimórficos.
- Implementar una aplicación de baja complejidad

Sintaxis de Herencia en C++

```
class Subclase : public SuperClase {
    // lista de miembros públicos y privados
};
```

```
5   class Motor {
6
7     public:
14    private:
17  };
```

```
4   #include "Motor.h"
5
6
7   class MotorPaP : public Motor {
8
9     public:
13    private:
15  };
```

Ing. César Aranda 3

Ejemplo C++

```
8   class Motor{
9
10    public:
11      virtual string arrancar();
12      virtual string parar();
13
14    private:
15        float tension;
16  };
```

```
8   class MotorPaP : public Motor {
9
10    public:
11       virtual float avanzar(int cantidad);
12
13    private:
14       float anguloPaso;
15  };
```

Ing. César Aranda 4

Interfaz de las clases Motor y MotorPaP

Motor.h

```
#include <string>
using namespace std;

class Motor {
public:
    Motor(float tension);
    float getTension() const;
    string arrancar() const;
    string parar() const;

private:
    float tension;
};
```

Clase MotorPaP que hereda de la clase Motor

MotorPaP.h

```
#include "Motor.h"

class MotorPaP : public Motor {
public:

    MotorPaP::MotorPaP(float tension, float anguloPaso) const;
    float getAnguloPaso() const;
    float avanzar(int cantidad) const;

private:
    float anguloPaso;
};
```

Ing. César Aranda

Definición de las Funciones Miembro

Motor.cpp

```
Motor::Motor(float tension){
    this->tension = tension;
}
float Motor::getTension() const {
    return tension;
}
string Motor::arrancar()const
//...
{
    return "Motor Andando";
}
//...
string Motor::parar()const
//...
{
    return "Motor Detenido";
}
```

Constructor de MotorPaP que hace uso del constructor de Motor

MotorPaP.cpp

```
#include "MotorPaP.h"

MotorPaP::MotorPaP(float tension, float anguloPaso):
                                    Motor(tension) {
    this->anguloPaso = anguloPaso;
}

float MotorPaP::getAnguloPaso() const {
    return anguloPaso;
}

float MotorPaP::avanzar(int cantidad)const
//...
{
    return (cantidad * this->anguloPaso);
}
```

Ing. César Aranda

Ejemplo de uso de Motor y MotorPaP

```
Motor Base
Tension: 12 V
Estado 1: Motor Andando
Estado 2: Motor Detenido

Motor PAP
Tension: 2.5 V
Paso: 1.8 °
Estado 1: Motor Andando
Ha rotado: 54 °
Estado 2: Motor Detenido
```

```
7  int main(int argc, char** argv) {
8      Motor *pmot1;
9      MotorPaP *pmotpap1;
10
11     pmot1 = new Motor(12);
12     cout << "Motor Base " << endl;
13     cout << "Tension: " << pmot1->getTension() << " V" << endl;
14     cout << "Estado 1: " << pmot1->arrancar() << endl;
15     cout << "Estado 2: " << pmot1->parar() << endl;
16
17     pmotpap1 = new MotorPaP(2.5, 1.8);
18     cout << endl << "Motor PAP " << endl;
19     cout << "Tension: " << pmotpap1->getTension() << " V" << endl;
20     cout << "Paso: " << pmotpap1->getAnguloPaso() << " °" << endl;
21     cout << "Estado 1: " << pmotpap1->arrancar() << endl;
22     cout << "Ha rotado: " << pmotpap1->avanzar(30) << " °" << endl;
23     cout << "Estado 2: " << pmotpap1->parar() << endl;
24     return 0;
25 }
```

7

Funciones Miembro Constantes (FMC)

```
class Motor {
public:
    Motor(float tension);
    float getTension() const;
    string arrancar() const;
    string parar();

private:
    float tension;
};
```

- Las FMC no modifican la representación interna de los objetos (datos miembro)
- A un objeto constante sólo se le pueden enviar mensajes usando FMC

```
25      const Motor *pmot2 = new Motor(220);
26      cout << "Estado 1: " << pmot2->arrancar() << endl;
27      cout << "Estado 2: " << pmot2->parar() << endl;
```

herencia_motores.cpp

```
24
25      passing `const Motor' as `this' argument of `std::string Motor::parar()' discards qualifiers
        ----
26      (Alt-Enter shows hints)
        cout << "Estado 2: " << pmot2->parar() << endl;
```

Error de compilación

Ing. César Aranda 8

Algunas Restricciones en C++

- Hay algunos elementos de la clase base que ***no*** pueden ser heredados:
  - Constructores
  - Destructores
  - Funciones friend
  - Funciones y datos estáticos de la clase Operador de Asignación (=) sobrecargado
- Una clase derivada no puede acceder directamente a los miembros private de su clase base.

Ing. César Aranda 9

Sobreescritura

- Una clase derivada puede sobreponer una función miembro de su clase base indicando una nueva versión de dicha función con la misma firma. Cuando esto ocurre, lo más frecuente es que la versión de la clase derivada llame a la versión de la clase base a fin de que realice parte de la nueva tarea.
- No utilizar el operador de resolución de ámbito (::) para referenciar la función miembro de la clase base hace que la función miembro de la clase derivada se llame a sí misma, provocando esto una recursión infinita.

Ing. César Aranda 10

Sobreescritura en C++ (1)

```
7  class Motor {
8  public:
9      Motor(float tension);
10     float getTension() const;
11     string arrancar();
12     string parar();
13     string toString() const;
14 private:
15     float tension;
16 };
```

```
1  #include <sstream>
2  using namespace std;
3
4  #include "Motor.h"
5
33 string Motor::toString() const {
34     string aux = "Soy un motor de ";
35     stringstream ss;
36     ss << this->getTension();
37     aux += ss.str();
38     aux += string(" voltios");
39     return aux;
40 }
```

```
28     cout << endl << endl << endl;
29     cout << "El Motor Basico dice: " << pmot1->toString() << endl;
30     cout << "El Motor PaP dice: " << pmotpap1->toString() << endl;
```

```
El Motor Basico dice: Soy un motor de 12 voltios
El Motor PaP dice: Soy un motor de 2.5 voltios
```


Sobreescritura en C++ (2)

```
6  class MotorPaP : public Motor {
7  public:
8
9      MotorPaP::MotorPaP(float tension,
10                        float anguloPaso);
11     float getAnguloPaso() const;
12     float avanzar(int cantidad) const;
13     string toString() const;
14
15 private:
16     float anguloPaso;
17 };
```

Función que sobreescribe la definición heredada

```
1  #include <sstream>
2  using namespace std;
3
4  #include "MotorPaP.h"
5
25 string MotorPaP::toString() const {
26     string aux = "Soy un motor PaP de ";
27     stringstream ss;
28     ss << this->getTension();
29     aux += ss.str();
30     aux += string(" voltios y Paso de ");
31     ss << this->getAnguloPaso();
32     aux += ss.str();
33     aux += string(" grados");
34     return aux;
35 }
```

```
28     cout << endl << endl << endl;
29     cout << "El Motor Basico dice: " << pmot1->toString() << endl;
30     cout << "El Motor PaP dice: " << pmotpap1->toString() << endl;
```

```
El Motor Basico dice: Soy un motor de 12 voltios
El Motor PaP dice: Soy un motor PaP de 2.5 voltios y Paso de 2.51.8 grados
```

Desarrollo en 2 capas (M+VC): Parte M

```
7  class Motor {
8  public:
9      Motor(float tension);
10     float getTension() const;
11     string arrancar();
12     string parar();
13     string toString() const;
14 private:
15     float tension;
16 };
```

```
6  class MotorPaP : public Motor {
7  public:
8
9      MotorPaP::MotorPaP(float tension,
10             float anguloPaso);
11     float getAnguloPaso() const;
12     float avanzar(int cantidad) const;
13     string toString() const;
14
15 private:
16     float anguloPaso;
17 };
```

```
1  #include <sstream>
2  using namespace std;
3
4  #include "Motor.h"
5
33 string Motor::toString() const {
34     string aux = "Soy un motor de ";
35     stringstream ss;
36     ss << this->getTension();
37     aux += ss.str();
38     aux += string(" voltios");
39     return aux;
40 }
```

```
1  #include <sstream>
2  using namespace std;
3
4  #include "MotorPaP.h"
5
25 string MotorPaP::toString() const {
26     string aux = "Soy un motor PaP de ";
27     stringstream ss;
28     ss << this->getTension();
29     aux += ss.str();
30     aux += string(" voltios y Paso de ");
31     ss << this->getAnguloPaso();
32     aux += ss.str();
33     aux += string(" grados");
34     return aux;
35 }
```

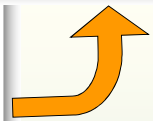
Ing. César Aranda 13

Desarrollo en 2 capas (M+VC): Parte VC

```
Motor Base
Tension: 12 V
Estado 1: Motor Andando
Estado 2: Motor Detenido

Motor PAP
Tension: 2.5 V
Paso: 1.8 °
Estado 1: Motor Andando
Ha rotado: 54 °
Estado 2: Motor Detenido
```

```
7  int main(int argc, char** argv) {
8      Motor *pmot1;
9      MotorPaP *pmotpap1;
10
11     pmot1 = new Motor(12);
12     cout << "Motor Base " << endl;
13     cout << "Tension: " << pmot1->getTension() << " V" << endl;
14     cout << "Estado 1: " << pmot1->arrancar() << endl;
15     cout << "Estado 2: " << pmot1->parar() << endl;
16
17     pmotpap1 = new MotorPaP(2.5, 1.8);
18     cout << endl << "Motor PAP " << endl;
19     cout << "Tension: " << pmotpap1->getTension() << " V" << endl;
20     cout << "Paso: " << pmotpap1->getAnguloPaso() << " °" << endl;
21     cout << "Estado 1: " << pmotpap1->arrancar() << endl;
22     cout << "Ha rotado: " << pmotpap1->avanzar(30) << " °" << endl;
23     cout << "Estado 2: " << pmotpap1->parar() << endl;
24     return 0;
25 }
```



14

Ejemplo Polimorfismo por Sobrecarga (1)

```
13  class Conversion {
14      int numero;
15  public:
16      Conversion();
17      Conversion(const Conversion& orig);
18      virtual ~Conversion();
19
20      void setNumero(float);
21      void setNumero(string);
22      void setNumero(char *);
23      int getNumero();
24  };
```

```
16  void Conversion::setNumero(float f) {
17      this->numero = (int) f;
18  };
19
20  void Conversion::setNumero(string s) {
21      stringstream(s) >> this->numero;
22  };
23
24  void Conversion::setNumero(char *a) {
25      int aux = 0;
26      int n = strlen(a);
27      for (int i = 0; i < n
28              && a[i] != '\0'; i++) { // ctrl redundante
29          aux += (a[i] - 48) * (int) pow(10, n - i - 1);
30      }
31      this->numero = aux;
32  };
33
34  int Conversion::getNumero() { return numero; }
```

Ing. César Aranda 15

Ejemplo Polimorfismo por Sobrecarga (2)

```
16      Conversion i;
17      i.setNumero(20.43);
18      cout << "Si flotante: " << i.getNumero() << endl;
19
20      i.setNumero(string("15"));
21      cout << "Si cadena: " << i.getNumero() << endl;
22
23      char arreglo[] = {'1', '6', '5', '\0'};
24      i.setNumero(arreglo);
25      cout << "Si array char: " << i.getNumero() << endl;
26
27      int a;
28      i.setNumero(20.43);
29      a = i.getNumero();
30      i.setNumero(string("15"));
31      a += i.getNumero();
32      i.setNumero(arreglo);
33      a += i.getNumero();
34      cout << "Suma de los anteriores: " << a << endl;
```

```
Si flotante: 20
Si cadena: 15
Si array char: 165
Suma de los anteriores: 200
```

Ing. César Aranda 16

Polimorfismo por Herencia y Enlaz.Dinám. (1)

```
class Persona {
protected:
    string nombre;
public:
    Persona();
    Persona(string);
    Persona(const Persona& orig);
    virtual ~Persona();

    string GetNombre();
    virtual string quienEsUd();
};
```

```
class Empleado : public Persona {
protected:
    string categoria;
public:
    Empleado();
    Empleado(string);
    Empleado(const Empleado& orig);
    virtual ~Empleado();
    void SetCategoria(string);
    virtual string quienEsUd();
};
```

Ejemplo de miembro de datos friendly

```
class Empleado_Compras : public Empleado{
    string Sector;
public:
    Empleado_Compras(string);
    Empleado_Compras(const Empleado_Compras& orig);
    virtual ~Empleado_Compras();
    virtual string quienEsUd();
};
```

Ing. César Aranda 17

Polimorfismo por Herencia y Enlaz.Dinám. (2)

```
#include "Persona.h"

Persona::Persona(){}

Persona::Persona(string nombre){
    this->nombre = nombre;
}
string Persona::GetNombre(){
    return nombre;
}

string Persona::quienEsUd(){
    string msg("Soy ");
    msg+=this->nombre;
    return msg;
}
```

```
#include "Empleado.h"

Empleado::Empleado() {
}

Empleado::Empleado(string nombre) {
    this->nombre = nombre;
}
void Empleado::SetCategoria(string categoria) {
    this->categoria = categoria;
}

string Empleado::quienEsUd() {
    string msg("Soy ");
    msg += this->nombre + ", el " + this->categoria;
    return msg;
}
```

```
Empleado_Compras::Empleado_Compras(string nombre) {
    this->nombre = nombre;
    this->Sector = "Compras";
}
string Empleado_Compras::quienEsUd() {
    string msg("Soy ");
    msg += this->nombre + ", el " + this->categoria +
           '\n' + " del Departamento de " + this->Sector;
    return msg;
}
```

Ing. César Ar 18

Polimorfismo por Herencia y Enlaz.Dinám. (3)

```
22      Persona op1("Angela");
23      Empleado op2("Rodrigo");
24      Empleado_Compras op3("Agustina");
25      Persona *op;
26
27      op2.SetCategoria("JEFE");
28      op3.SetCategoria("Administrativo");
29      op = &op1;
30      cout << op->quienEsUd() << endl;
31      op = &op2;
32      cout << op->quienEsUd() << endl;
33      op = &op3;
34      cout << op->quienEsUd() << endl;
35
36      cout << endl << "Con coleccion es" << endl;
37      Persona * pers[] = {&op1, &op2, &op3};
38      for (int i = 0; i < 3; i++)
39          cout << pers[i]->quienEsUd() << endl;
```

```
Soy Angela
Soy Rodrigo, el JEFE
Soy Agustina, el Administrativo
 del Departamento de Compras

Con coleccion es
Soy Angela
Soy Rodrigo, el JEFE
Soy Agustina, el Administrativo
 del Departamento de Compras
```

El modificador virtual indica al compilador que genere código que mire al tipo del objeto en tiempo de ejecución y use esta información para seleccionar la versión apropiada de la función.

Ing. César Aranda 19

Herencia y Polimorfismo en Python

Atributo de Clase e Instancia. Visibilidad.

```
Auto 1:  Ford
Auto 2:  Audi
Auto 4:  Toyota
Auto: Toyota
La distancia del Toyota es de 192.00
Total de autos:  3
o bien         :  3
o bien         :  3
```

```
class Auto:
    """Clase para autos (ejemplo) // Archivo

    num_autos = 0       # Numero total de au

    def __init__(self, marca):
        self.marca = marca
        Auto.num_autos += 1

    def __str__(self):
        return "Auto: {}".format(self.marca)

    def set_veloc(self, veloc):
        self.veloc = veloc

    def set_tiempo(self, tiempo):
        self.tiempo = tiempo

    def get_dist(self, aceler = None):
        if aceler is not None:
            distancia = aceler * self.tiempo^2 + self.veloc * self.tiempo
            print "La distancia del {} es de {:.2f}".format(self.marca, distancia)
        else:
            print "La distancia del {} es de {:.2f}".format(self.marca, self.veloc*self.tiempo)
```

```
#!/usr/bin/python
# -*- coding: ut

from auto import
from autoMejorado

if __name__ == "

    auto1 = Auto('Ford')
    auto2 = Auto('Audi')
    auto4 = Auto('Toyota')
    print "Auto 1: ", auto1.marca
    print "Auto 2: ", auto2.marca
    print "Auto 4: ", auto4.marca
    print(auto4)
    auto4.set_veloc(100)
    auto4.set_tiempo(2)
    auto4.get_dist(5)
    print "Total de autos: ", auto4.get_num_autos()
    print "o bien        : ", auto4.num_autos
    print "o bien        : ", Auto.num_autos
```


Herencia, Polimorfismo x Sobreescritura

```
Auto: Ford
Auto: Fiat, Modelo: Argos
La distancia del Fiat es de 180.00
Total de autos:  3
o bien         :  3
o bien         :  3
```

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from auto import Auto
from autoMejorado import AutoMejorado

if __name__ == "__main__":

    auto1 = Auto('Ford')
    auto2 = Auto('Audi')
    auto3 = AutoMejorado('Fiat', 'Argos')

    print(auto1)
    print(auto3)
    auto3.set_veloc(120)
    auto3.set_tiempo(1.5)
    auto3.get_dist()
    print "Total de autos: ", auto3.get_num_autos()
    print "o bien        : ", auto3.num_autos
    print "o bien        : ", Auto.num_autos
```

```
from auto import Auto

class AutoMejorado(Auto):
    """Clase para Autos Mejorados (ejem
    def __init__(self, marca, modelo):
        Auto.__init__(self, marca)
        self.modelo = modelo

    def __str__(self):
        return "Auto: {}, Modelo: {}".format(self.marca, self.modelo)
```

Práctica usando C++ y Python

- Objetivo:
  - Implementar una aplicación de consola, usando OO
- Consigna:

  El programa debe crear 1 móvil (manual) dentro de una grilla plana. El móvil posee un nombre único y conoce su posición actual, la secuencia de órdenes recibidas y la distancia total recorrida.

  Las dimensiones del plano y las órdenes de movimiento son dadas por el operador (para moverse puede usar el código estructurado visto en las diapositivas 20-21 de la presentación C vs C++).

  Crear también 3 objetos móviles autónomos (obtenidos a partir del anterior). Cada móvil está alimentado por una batería de carga inicial variable al azar entre 200-300 mAh y cuenta con un indicador de carga de la misma. Por cada unidad de distancia recorrida se consumen entre 20 a 40 mAh (constante al azar para móvil).

  El reporte debe mostrar, en cada ciclo, la posición, la cadena de órdenes recibidas, la distancia, el tiempo que el móvil estuvo en movimiento desde su creación hasta que el operador eligió TERMINAR. Además del nivel de carga y consumo si corresponden.

Ing. César Aranda 23

Referencias C++

- DEITEL, H. M. y DEITEL, P. J. (2009): Cómo Programar en C/C++, 6ª edicion, Prentice-Hall
- ACERA GARCIA, M.A. (2010): C/C++, Anaya Multimedia
- http://www.cplusplus.com/doc/tutorial/classes/
- http://es.wikipedia.org/wiki/C%2B%2B
- http://www.zator.com/Cpp/index.htm
- http://plagatux.es/category/programacion/
- http://mindview.net/Books/TICPP/ThinkingInCPP2e.html
- http://arco.esi.uclm.es/~david.villa/pensar_en_C++/vol1/index.html

Ing. César Aranda 24

Referencias Python

- Von Rossum, G. (2017): El tutorial de Python 3. Editorial Python Software Foundation. Disponible en URL http://tutorial.python.org.ar/, accedido en 2017.
- http://pyspanishdoc.sourceforge.net/lib/lib.html
- Chazalet, S. (2017): Python3, los fundamentos del lenguaje. 2da edición. Ediciones Eni.
- Pérez castaño, A. (2016): Python fácil. Editorial Marcombo
- Beazley, D. y Jones, B. (2013). Python Cookbook. 3ra edición. O'Reilly Media. California.
- Hinojoza Gutierrez, A.P. (2016): Python, paso a paso. Editorial Ra-Ma. Madrid
- González Duque, R. (2008). Python para todos. UPR http://mundogeek.net/tutorial-python/, accedido en 2015.
- Hinojoza Gutierrez, A.P. (2016): Taller de Python. En URL https://www.psicobyte.com/descargas/taller_python.pdf , accedido en 2016