UTEC

CIENCIA DE LA COMPUTACIÓN

Unidad 13: Polimorfismo

CS1102 – PROGRAMACIÓN ORIENTADA A OBJETOS 1 CICLO 2019-2
UTEC

Logros de la sesión

- Al finalizar la sesión los alumnos entienden el concepto de polimorfismo

Logros de la sesión

- Al finalizar la sesión los alumnos entienden el concepto de polimorfismo
- Conoceran los distintos tipos de Polimorfismo soportado en el lenguaje C++.

Logros de la sesión

- Al finalizar la sesión los alumnos entienden el concepto de polimorfismo
- Conoceran los distintos tipos de Polimorfismo soportado en el lenguaje C++.
- Conocerán las relaciones entre el Polimormismo y otros tipos de abstracción en el lenguaje.

Logros de la sesión

- Al finalizar la sesión los alumnos entienden el concepto de polimorfismo
- Conoceran los distintos tipos de Polimorfismo soportado en el lenguaje C++.
- Conocerán las relaciones entre el Polimormismo y otros tipos de abstracción en el lenguaje.
- Utiliza el polimorfismo para representar comportamiento dinámico en C++

Polimorfismo

¿Quien puede adoptar la forma de cualquier humano?

Figure: Morfeo de Matrix

Figure: Dios griego Morfeo

Figure: Una entidad multiples formas

¿Polimorfismo en biología?

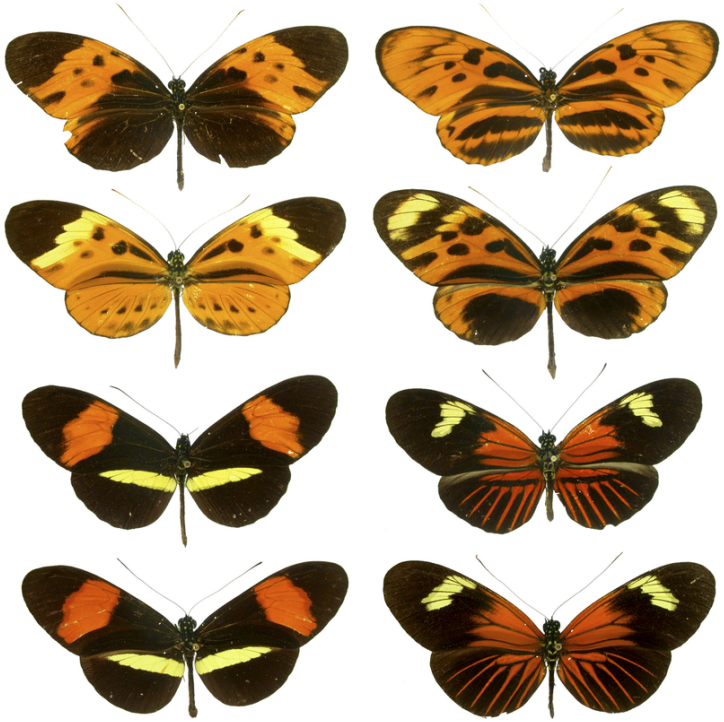

Figure: Lepidoptera

Figure: ¿Es la misma especie o especies diferentes?

¿Que es el Polimorfismo en la programación?

- Polimorfismo significa que una entidad puede tomar muchas formas.
- En la programación convencional cada valor o variable puede ser de un solo tipo y valor. Esto se conoce como monomorfo.
- En lenguajes que soportan polimorfismo, un objeto polimórfico es una entidad, como una variable o argumento de función, que puede tener valores de tipos diferentes en el curso de ejecución.
- Las funciones polimórficas son funciones que tienen argumentos polimórficos.
- Los tipos polimórficos son tipos cuyas operaciones son aplicables a valores de más de un tipo.

Clasificación del Polimorfismo(1)

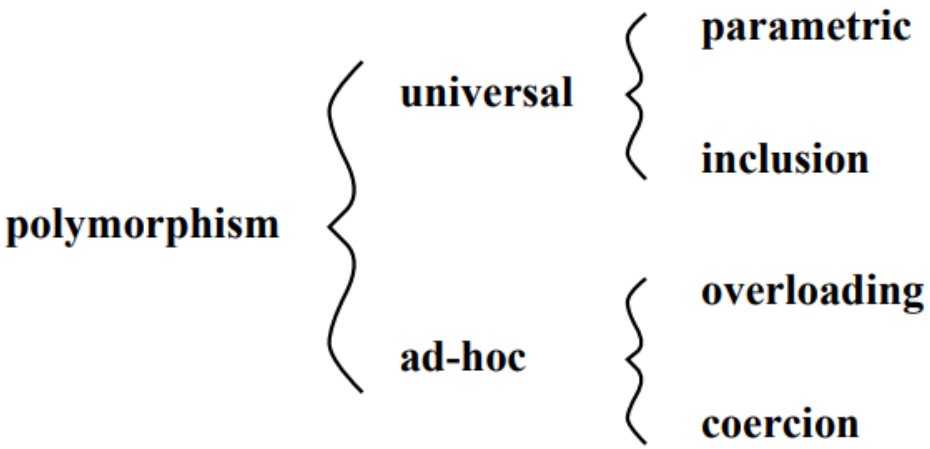

Figure: Clasificación del Polimorfismo

Luca Cardelli and Peter Wegner. "On Understanding Types, Data Abstraction, and Polymorphism". In: *ACM Comput. Surv.* 17.4 (Dec. 1985), pp. 471–523. ISSN: 0360-0300. DOI: 10.1145/6041.6042. URL: http://doi.acm.org/10.1145/6041.6042

Clasificación del Polimorfismo(2)

- El polimorfismo A) universal se divide en: 1) **parametric**, el cual se obtiene cuando una función funciona de manera uniforme en una variedad de tipos: estos tipos normalmente exhiben alguna estructura común.
- Por otra parte 2) **inclusion** se refiere a la herencia y los subtipos. (Visto en las clases de Herencia)
- El polimorfismo B) **ad-hoc** o no universal, se obtiene cuando una función trabaja, o parece funcionar, con varios tipos diferentes (que pueden no exhibir una estructura común) y pueden comportarse en formas no relacionadas para cada tipo.
- También hay dos tipos principales de polimorfismo ad-hoc. El 3)**overloading** el mismo nombre de variable esto se usa para denotar diferentes funciones, y el contexto se usa para decidir qué función se denota mediante un instancia particular del nombre.
- Y 4) **coercion** es, en cambio, una operación semántica que se necesita para convertir un argumento para el tipo esperado por una función, en una situación que de lo contrario resultaría en un error de tipo.
- En el caso del polimorfismo universal, uno puede afirmar con confianza que algunos valores (es decir, funciones polimórficas) tienen muchos tipos, mientras que en el polimorfismo ad-hoc esto es más difícil de sostener, ya que uno puede tomar la posición de que una función polimórfica ad-hoc es realmente un pequeño conjunto de funciones monomórficas.Cardelli1985

Los 4 tipos de Polimorfismo en C++

- De los 4 tipos de polimorfismo, ¿como se pueden representar en C++?
- 1) **parametric**, es conocido como **compile-time polymorphism**.
- 2) **inclusion** se refiere a **runtime polymorphism** o **subtype** herencia de tipos.
- El 3)**overloading** lleva el mismo nombre.
- Y 4) **coercion** es, conocido como (implicito o explicito) **casting**.

1) Parametric o Compile-Time Polymorphism

- Provee un medio para ejecutar el mismo código para cualquier tipo. En C++ esto se implementa vía plantillas.

```
1  #include <iostream>
2  #include <string>
3
4    template <class T>
5    T max(T a, T b) {
6      return a > b ? a:b;
7    }
8
9    int main() {
10     std::cout << ::max(9, 6) << "\n";
11     std::cout << ::max(12.6, 3.0) << "\n";
12     return 0;
13   }
```

¿Con que tipos de datos funciona la función max?

2) Inclusion o Subtype Polymorphism (1)

- Es lo que conocemos como el uso de clases derivadas vía punteros y referencias. También llamado Runtime Polymorphism.
- En el siguiente ejemplo usamos una relación de clases para la familia de felinos.

```
1  class Felino {
2    public:
3      virtual void maullido() = 0;
4  };
5
6  class Gato: public Felino {
7    public:
8      void maullido() {
9        std::cout << "Maullido como un gato\n";
10     }
11 }
12
13 class Tigre: public Felino {
14   public:
15     void maullido() {
16       std::cout << "Maullido como un tigre!!\n";
17     }
18 }
19 ...
```

2) Inclusion o Runtime o Subtype Polymorphism(2)

- Veamos ahora un programa que usa la jerarquía de clases de Felinos.

```
1
2  #include <iostream>
3  #include "felinos.h"
4
5  void do_maullido(Felino *gato) {
6    gato->maullido();
7  }
8
9  int main() {
10
11   Gato gato;
12   Tigre tigre;
13   Puma puma;
14   do_maullido(&gato);
15   do_maullido(&tigre);
16   do_maullido(&puma);
17 }
```

- ¿Que resultado dará la llamada a la función?
- ¿En que momento se determina el tipo correspondiente del parámetro?

3) Overloading o Sobrecarga

- Esto permite tener varias funciones con el mismo nombre que se comportan diferente de acuerdo al tipo de dato que reciben.
- Por ejemplo la función suma con parámetros enteros y con cadenas.

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int add(int a, int b) {
5   return a + b;
6  }
7  std::string add(string a, string b) {
8   std::string result(a);
9   result += b;
10  return result;
11 }
12
13 int main() {
14  std::cout << add(5, 9) << "\n";
15  std::cout << add("hello ", "world") << "\n";
16 }
```

- ¿Que resultado se obtiene?

4) Coercion o Casting (1)

- Casting o conversión ocurre cuando un objeto o variable es convertido a otro tipo de objeto o tipo de dato.
- Por ejemplo una variable entera asignada por un valor real implicitamente se convierte a entero el valor real.

```
1
2 float b = 6;
3 int a = 9.99;
4
5 int x = (int) 3.1415;
```

4) Coercion o Casting (2)

- La coerción también ocurre si el constructor de una clase no se establece de forma explícita.

```
1  #include "tipos.h"
2  class Alumno {
3      entero nota1, nota2;
4      caracter* name;
5  public:
6      Alumno(entero i) { };
7      Alumno(const caracter* n, entero j = 0) { };
8  };
9
10 void add(Alumno) { };
11
12 int main() {
13
14     Alumno obj1 = 2; // obj1 = Y(2)
15
16     Alumno obj2 = "somestring"; // obj1 = Y(2)
17
18     obj1 = 10;    // obj1 = Y(10)
19
20     add(5);    // add(Y(5))
21 }
```

Ejercicio 1

Enunciado

Escribir un algoritmo en C++ que construya una clase Array Polimórfico de tipo paramétrico usando plantillas que permita crear un arreglo de cualquier tipo de dato:

- Construya un Array de enteros e inicialicelos.
- Construya una función que imprima los datos almacenados.

```
1
2  template <typename T>
3  class Array {
4  private:
5      T *ptr;
6      int size;
7  public:
8      Array(T arr[], int s);
9      void print();
10 };
```

Ejercicio 1 (Solución)

```
1
2  #include <iostream>
3  using namespace std;
4
5  template <typename T>
6  class Array {
7  private:
8    T *ptr;
9    int size;
10 public:
11   Array(T arr[], int s);
12   void print();
13 };
14
15 template <typename T>
16 Array<T>::Array(T arr[], int s) {
17   ptr = new T[s];
18   size = s;
19   for(int i = 0; i < size; i++)
20     ptr[i] = arr[i];
21 }
```

Ejercicio 1 (Solución)

```
1  template <typename T>
2  void Array<T>::print() {
3    for (int i = 0; i < size; i++)
4      cout<<" "<<*(ptr + i);
5    cout<<endl;
6  }
7  int main() {
8    int arr[5] = {1, 2, 3, 4, 5};
9    Array<int> a(arr, 5);
10   a.print();
11   return 0;
12 }
```

Ejercicio 2

Enunciado

Diseñe un algoritmo en C++ que use polimorfismo de (inclusion) o subtipo.El programa debe representar la jerarquía de clases para una imagen y sus clases derivadas JPG y PNG.

```
1  class Imagen {
2  private:
3      entero alto;
4      entero ancho;
5  public:
6      Imagen(entero _alto, entero _ancho):alto{_alto}, ancho{_ancho} {}
7      virtual void guardar(const string &Nombre) = 0;
8  };
9  class JPEG : public Imagen {
10 public:
11     JPEG(entero _alto, entero _ancho):Imagen(_alto, _ancho){}
12     void guardar(const string &Nombre) override ;
13 };
14 class PNG: public Imagen {
15 public:
16     PNG(entero _alto, entero _ancho):Imagen(_alto, _ancho){}
17     void guardar(const string &Nombre) override ;
};
```

Ejercicio 2 (Solución)

```
1  void PNG::guardar(const string &Nombre) {
2      std::cout << "Guardando un archivo PNG " << Nombre;
3  }
4  void JPEG::guardar(const string &Nombre){
5      std::cout << "Guardando un archivo JPEG" << Nombre;
6  }
7
8  void guardarImagen(Imagen *img, const string &Nombre)
9  {
10     img->guardar(Nombre);
11 }
12
13 int main() {
14
15     JPEG jpg(10, 20);
16     PNG png(30, 30);
17     string Nombre1 = "rombo.jpg";
18     string Nombre2 = "cubo.png";
19     guardarImagen(&jpg, Nombre1);
20     guardarImagen(&png, Nombre2);
21 }
```

Ejercicio 3

Enunciado

Escribir un programa en C++ que use sobrecarga(overloading).

Dos físicos quieren saber cual es el diametro de una pompa de jabón cuando se junta en el aire con otra. Para ello se necesitan sobrecargar un operador de suma y asignación.

La sobrecarga de operadores es equivalente a la sobrecarga de una función, en este caso se sobrecarga el operador +=.

Realice la simulación y ejecute una operación de suma de pompas de jabón.

```
1
2  #include "Tipos.h"
3  class Pompa {
4    private:
5      real radio;
6    public:
7      Pompa();
8  };
```

Ejercicio 3 (Solución)

http://bit.ly/33h9e0d

```
1
2  #include "Tipos.h"
3  class Pompa {
4    private:
5      real radio;
6    public:
7      Pompa();
8      Pompa(real pradio):radio{pradio}{}
9      void setRadio(real pradio){radio = pradio;}
10     real Diametro() const {return radio*2;}
11     real Radio() const {return radio;}
12
13     Pompa& operator+=(Pompa z){radio+=z.Radio(); return *this;}
14 };
15 int main(){
16   Pompa p1(10);
17   Pompa p2(15);
18   p1 += p2;
19   std::cout << p1.Diametro();
20 }
```

Ejercicio 4

Enunciado

Construya una clase Persona en los que se observe el casting por constructor.

```
1  class Persona {
2  private:
3      real peso;
4      entero edad;
5      caracter* nombre;
6  public:
7      Persona(entero _edad):edad{_edad}, peso{0.0}, nombre{""} {}
8      Persona(caracter * _nombre):edad{0}, peso{0}, nombre{_nombre}{}
9      void print(){std::cout << "peso:" << peso << " edad:" << edad << "
         nombre:" << nombre; }
10 };
11 int main() {
12     Persona p1 = 5;
13     Persona p2 = "Juan";
14     p1.print();
15     p2.print();
16 }
```

Cierre

En esta sesión aprendiste

A conocer el concepto de comportamiento polimórfico.

A conocer la implementación en C++ de los distintos tipos de polimorfismo.