UNIVERSIDAD DE LIMA

POLIMORFISMO Y CLASES ABSTRACTAS

UNIDAD 3: HERENCIA Y POLIMORFISMO

SEMANA 9

Temario

- Definición de polimorfismo.
- Compatibilidad de tipos.
- Métodos polimórficos.
- Definición de una clase abstracta.
- Declaración de clases abstractas y métodos abstractos.
- Implementación de métodos abstractos.
- Colecciones heterogéneas.

UNIVERSIDAD DE LIMA

MOTIVACIÓN

Cómo vamos con la oferta laboral

1. Definición de polimorfismo

UNIVERSIDAD DE LIMA

POLIMORFISMO = «poli» (muchos) + «morfé» (forma)

- Un método es polimórfico si tiene "**muchas formas**".
- En O-O esto significa que el mismo método puede aplicarse a diferentes clases de objetos.
- Cada clase puede implementarlo de manera diferente (muchas formas).

Ejemplo:

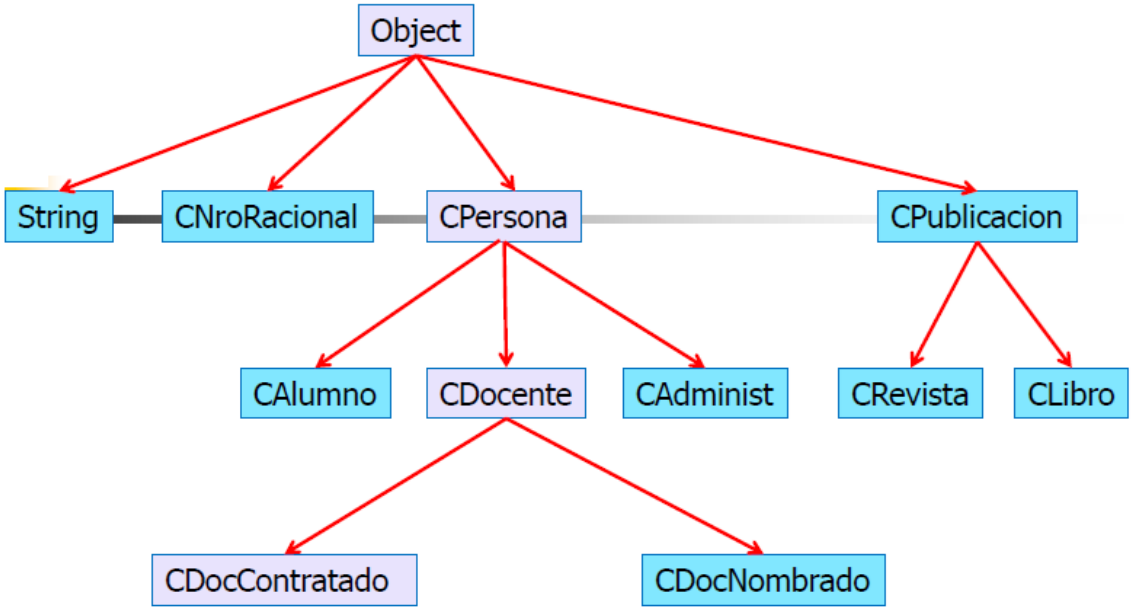
```
Object x = new String( "poly" );
out.print( x.toString() );  // "poly"
x = new Date( );
out.print( x.toString() );  // "Mon Dec 14 09:00.."
```

1. ¿Qué es polimorfismo?

- En POO se refiere al hecho de que un mismo identificador se puede comportar de diferentes formas, accediendo a objetos de diferentes clases.

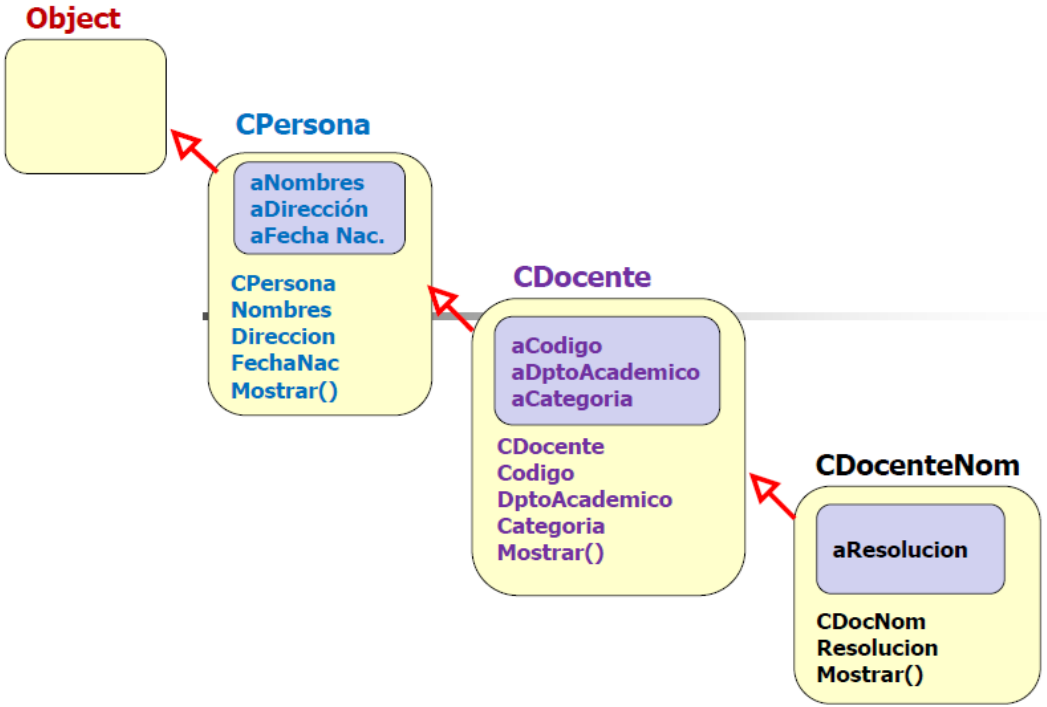
1. ¿Qué es polimorfismo?

- El polimorfismo es posible en una jerarquía de herencias, en virtud a que una subclase tiene todos los elementos de su superclase, por tanto, puede comportarse como tal. Por ejemplo: Un objeto de la clase **CDocContratado** puede comportarse como un objeto de la clase **CDocente**, **CPersona** u **Object**.



UNIVERSIDAD DE LIMA

1. ¿Qué es polimorfismo?



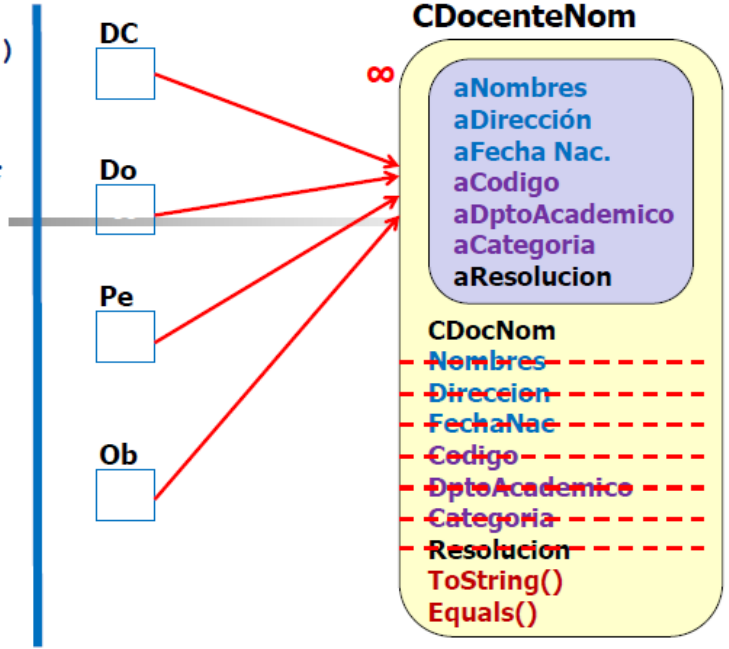
- En POO el polimorfismo se refiere a la posibilidad de definir clases diferentes dentro de una misma jerarquía de herencias que tienen métodos o atributos denominados de forma idéntica, pero que se comportan de manera distinta.
- Por ejemplo el método **Mostrar()** tiene un comportamiento específico en cada clase, porque muestra diferente número de atributos en cada caso.

UNIVERSIDAD DE LIMA

1. ¿Qué es polimorfismo?

- Para entender polimorfismo primero se debe entender la compatibilidad de tipos entre identificadores de objetos.
- A un objeto de una subclase se puede acceder con identificadores de objetos de todas sus superclases.

```
public static void Main()
{
   CDocNom DC;
   DC = new CDocNom(...);
   CDocente Do;
   Do = DC;
   CPersona Pe;
   Pe = DC;
   Object Ob;
   Ob = Pe;

}
```

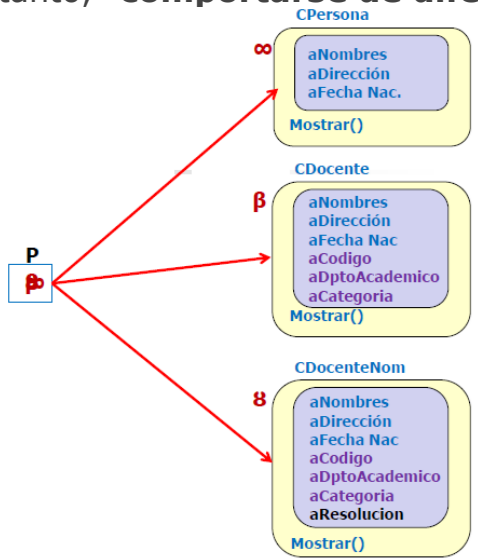


UNIVERSIDAD DE LIMA

1. ¿Qué es polimorfismo?

- El polimorfismo es el mecanismo que permite que un identificador de un objeto, pueda referenciar a diferentes tipos de objetos, por tanto, "**comportarse de diferentes formas**".

```
public static void Main()
{
   CPersona P;
   P = new CPersona(...);
   P.Mostrar();
   P = new CDocente(...);
   P.Mostrar();
   P = new CDocenteNom(...);
   P.Mostrar();


}
```



En este ejemplo, el identificador **P** tiene un comportamiento polimórfico, porque primero referencia a un objeto de la clase **CPersona**, luego a un objeto de la clase **CDocente** y finalmente a un objeto de la clase **CDocenteNom**. En cada caso ejecuta un método con el mismo nombre, pero, el resultado obtenido es distinto.

UNIVERSIDAD DE LIMA

2. Conversión de tipos

- Considerando la conversión de tipos primitivos:

```
int entero = 4;
double real;
real = entero;
```

Conversión implícita o automática

```
int entero;
double real = 5.34;
entero = (int) real;
```

Conversión explícita o casting

UNIVERSIDAD DE LIMA

2. Conversión de tipos (Continua ..)

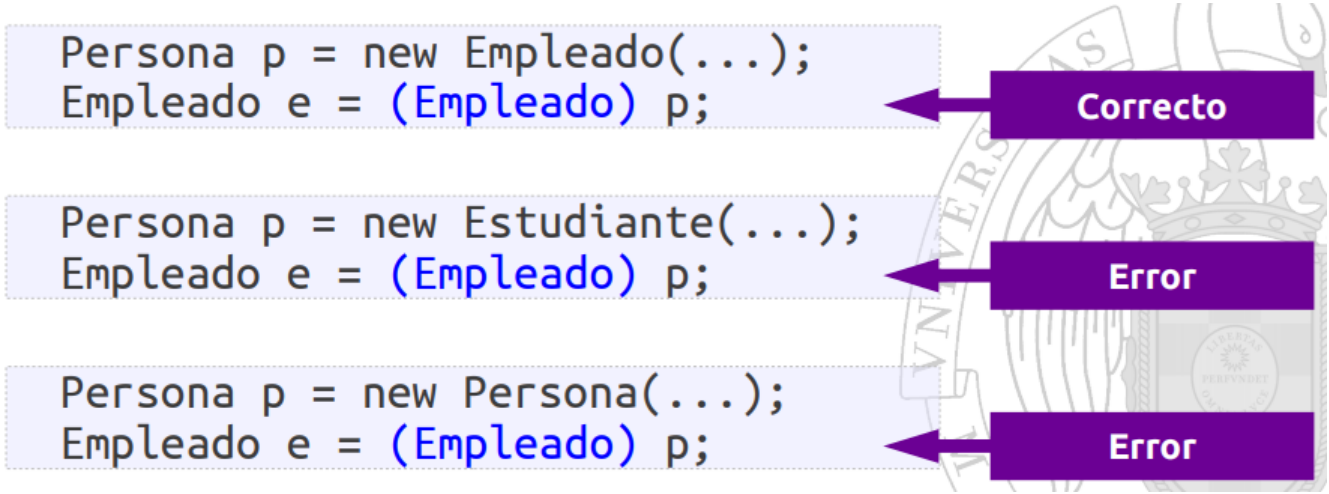
- Similarmente, podemos asignar un objeto de una clase B a una variable de tipo A, si A es superclase (directa o indirecta) de B.
- Esta forma de declarar se denomina**: Upcasting (Conversión hacia arriba)**

```
Persona p;
Empleado e = new Empleado(...);
p = e;
```

P es de tipo persona, y sólo podrá acceder a los métodos de la clase Persona

2. Conversión de tipos (Continua ..)

- Es posible asignar una variable de un tipo A a otra de un tipo B, si B es subclase de A. Pero esta conversión no siempre es correcta, y ha de indicarse explícitamente**: Downcasting (Conversión hacia abajo)**



http://dalila.sip.ucm.es/~manuel/JSW1/Slides/Polimorfismo.pdf

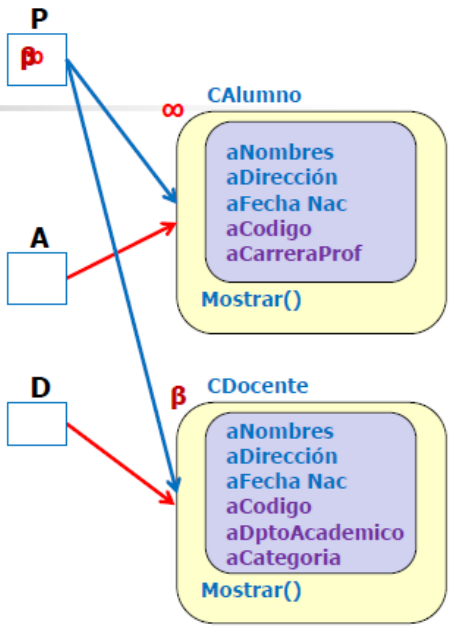
UNIVERSIDAD DE LIMA

3. Métodos polimórficos.

- La ventaja del polimorfismo es que permite optimizar código, mediante la implementación de métodos únicos que puedan trabajar con diferentes tipos de objetos.

```
public static void MostrarEnPantalla(CPersona P)
{
        P.Mostrar();
}
```

```
public static void Main()
{
    CAlumno A;

    A = new CAlumno(...);

    MostrarEnPantalla(A);

    CDocente D;

    D = new CDocente(...);

    MostrarEnPantalla(D);

}
```



En este ejemplo, la ventaja del polimorfismo es que se implementó un único método (**MostrarEnPantalla**) para mostrar los atributos tanto del objeto **Alumno** como del objeto **Docente**.

UNIVERSIDAD DE LIMA

UNIVERSIDAD DE LIMA

MOTIVACIÓN

https://youtu.be/Eo76x0uyTIE

4. Clase Abstracta

- Clase cuya descripción es incompleta y no puede instanciarse.
- Declara métodos, pero no tiene que implementarlos todos. Se utiliza para definir subclases.
  - Estas subclases pueden derivar de sólo una clase abstracta.
- ¿Cuándo se usa?
  - Cuando deseamos definir una abstracción que englobe objetos de distintos tipos y queremos hacer uso del polimorfismo.

5. Declaración de clases abstractas y métodos abstractos

- Es un método aplicable a todos los objetos de la super clase.
- La implementación del método es completamente diferente en cada subclase
- Para declarar un método como abstracto, se pone delante la palabra reservada **abstract** y no define un cuerpo:
- abstract tipo nombreMétodo (....);
- Luego en cada subclase se define un método con la misma cabecera y distinto cuerpo.

UNIVERSIDAD DE LIMA

5. Declaración de clases abstractas y métodos abstractos

- Si una clase contiene al menos un método abstracto, entonces es una clase abstracta.
- Una **clase abstracta** es una clase de la que no se pueden crear objetos, pero puede ser utilizada como clase padre para otras clases.

- Declaración:

```
public abstract class NombreClase {
.............
}
```

UNIVERSIDAD DE LIMA

6. Implementación de métodos abstractos

UNIVERSIDAD DE LIMA

6. Implementación de métodos abstractos

UNIVERSIDAD DE LIMA

```
1   package poo.claseabstracta;
2   public abstract class Forma {
3       private String nombre;
4       public Forma() {
5       }
6       public Forma(String nombre) {
7           this.nombre = nombre;
8       }
9       public void imprimirDatos(){
10          System.out.println("Nombre: "+nombre);
11          System.out.println("Area: "+area());
12      }
        public abstract double area();
14  }
```

6. Implementación de métodos abstractos

UNIVERSIDAD DE LIMA

```
1   package poo.claseabstracta;
2   public class Cuadrado extends Forma {
3       private double lado;
4       public Cuadrado() {
5       }
6       public Cuadrado(String nombre, double lado) {
7           super(nombre);
8           this.lado = lado;
9       }
10      @Override
        public double area() {
12          return lado*lado;
13      }
14  }
```

6. Implementación de métodos abstractos

UNIVERSIDAD DE LIMA

```
1   package poo.claseabstracta;
2   public class Rectangulo extends Forma{
        private double lado1;
        private double lado2;
5       public Rectangulo(String nombre, double lado1, double lado2) {
6           super(nombre);
7           this.lado1 = lado1;
8           this.lado2 = lado2;
9       }
10      @Override
        public double area(){
12          return lado1*lado2;
13      }
14  }
```

6. Implementación de métodos abstractos

UNIVERSIDAD DE LIMA

```
1   package poo.claseabstracta;
2   public class Circulo extends Forma{
3       private double r;
4       public Circulo(String nombre, double r) {
5           super(nombre);
6           this.r = r;
7       }
8       @Override
9       public double area() {
10          return Math.PI*r*r;
11      }
12  }
```

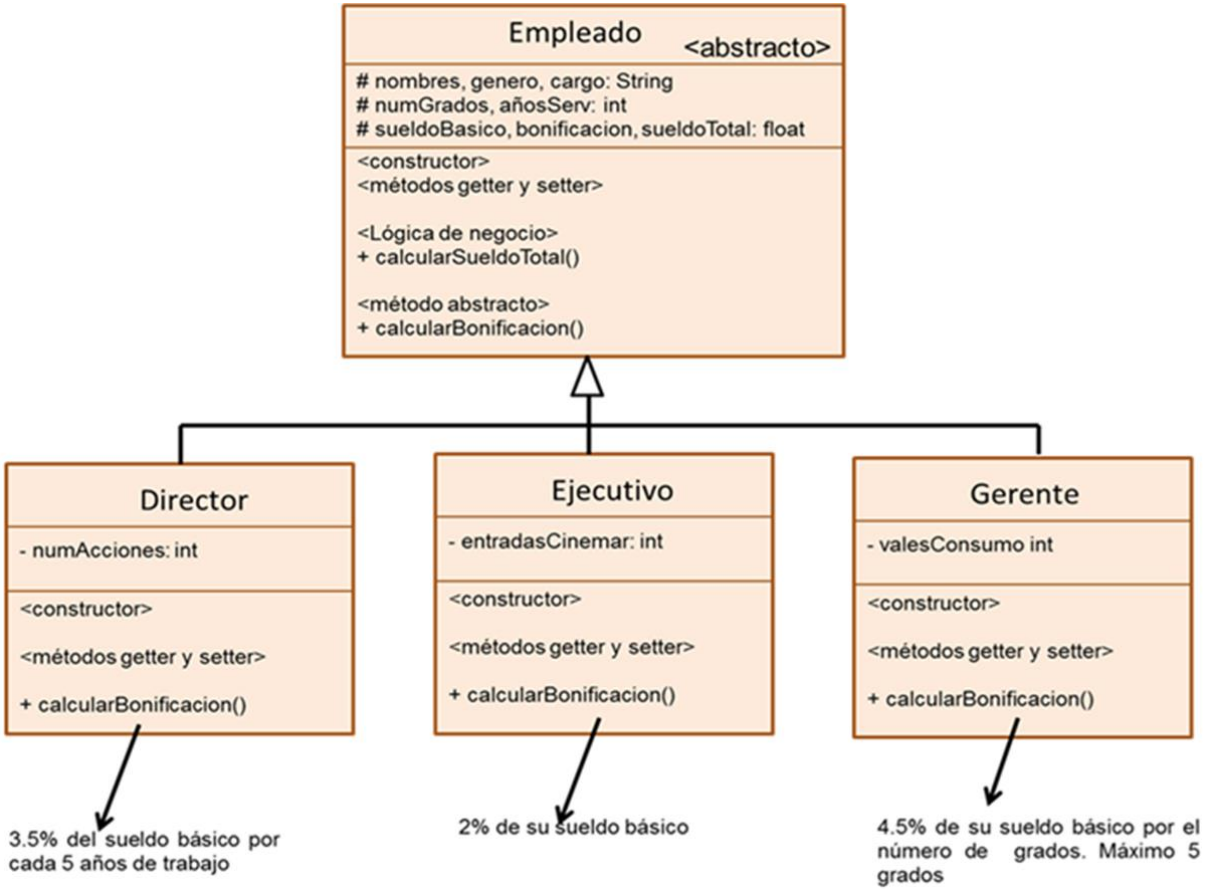
7. Colecciones heterogéneas

UNIVERSIDAD DE LIMA

```
1   package poo.claseabstracta;
2   import java.util.LinkedList;
3   public class AppForma {
4       public static void mostrarFormas(LinkedList<Forma> formas){
5           for (Forma f : formas){
6               f.imprimirDatos();
7           }
8       }
9       public static void main(String[] args){
10          Forma f1 = new Cuadrado("Cuadrado",5);
11          Forma f2 = new Rectangulo("Rectangulo",5,3);
12          Forma f3 = new Circulo("Circulo",1);
13          LinkedList<Forma> formas = new LinkedList<Forma>();
14          formas.add(f1);
15          formas.add(f2);
16          formas.add(f3);
17          mostrarFormas(formas);
18      }
19  }
```

Ejercicio en Clase

UNIVERSIDAD DE LIMA



Referencias

- Deitel, H. M. (2016). Java: como programar.
- Paul S. Wang, Java con programación orientada a objetos y aplicaciones en la WWW, México, 2000.
- Sun microsystem, Fundamentals of the Java™ Programming Language SL-110-SE6
- A.M. Vozmediano, Java para novatos, 2017.