RELACIONES ENTRE CLASES

DOCENTE. YOLANDA MOYAO MARTINEZ

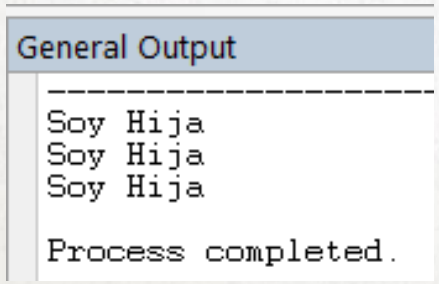
POLIMORFISMO

- **Polimorfismo** hace referencia a un conjunto de Métodos con diferentes funcionalidades con el mismo **Nombre e igual Número de Parámetros y Tipos**, pero que se encuentran definidos en diferentes Clases.
- Un objeto puede tomar diferentes formas de comportarse, es decir, que las subclases de una clase pueden definir su propio comportamiento.
- Sobreescribir es cuando la clase hija tiene el mismo método que su clase padre, aunque con código distinto.

```
public class Figura {
    public  void Dibujar(); }

public class Triangulo extendes  Figura {
     public  void Dibujar()
          { //Aqui dibujar un Triangulo } }

public class Cuadrado extends Figura {
       public  void Dibujar()
          { //Aqui dibujar un cuadrado } }
```

EJEMPLO

```
// clase padre con metodo()
class Padre {
  public void metodo() {
     System.out.println("Soy Padre");
  }
}

// Clase hija que sobreescribe metodo()
class Hija extends Padre {

@Override
    public void metodo() {
     System.out.println("Soy Hija");
  }
}
```




```
public class Polimorfirmo {

 public static void main (String[] args) {

Hija hija = new Hija();
hija.metodo();
 // Aquí sí usamos polimorfismo, puesto que la
//variable es de tipo Padre y guarda una instancia de Hija
Padre padre[] = {new Hija(), new Hija()};
padre[0].metodo();
padre[1].metodo();  }}
```

EJEMPLO

```
class Profesionista {

   private String especialidad;

   public Profesionista(String especialidad){
      this.especialidad = especialidad;
   }

   public void printMensaje(){
      System.out.println("Soy un Profesionista con
especialidad: " + this.getEspecialidad());
   }
   public String getEspecialidad() {
      return especialidad;
   }
   public void setEspecialidad(String especialidad)
{
      this.especialidad = especialidad;
   }
}
```

```
class Doctor extends Profesionista{

   private String nombre;

   public Doctor(String especialidad, String nombre){
      super(especialidad);
      this.nombre = nombre;
   }
   @Override
   public void printMensaje(){
      super.printMensaje();
      System.out.println("Me llamo " + getNombre() + " y soy doctora que
cura pacientes");    }
   public String getNombre() {
      return nombre;
   }
   public void setNombre(String nombre) {
      this.nombre = nombre;
   }
}
```

EJEMPLO

```
class Arquitecto extends Profesionista{

   private String nombre;

   public Arquitecto(String especialidad, String nombre){
      super(especialidad);
      this.nombre = nombre;
   }
    @Override
   public void printMensaje(){
      super.printMensaje();
      System.out.println("Me llamo " + getNombre() + " y
soy arquitecto que diseña planos");    }
   public String getNombre() {
      return nombre;
   }
   public void setNombre(String nombre) {
      this.nombre = nombre;
   }
}
```

```
public class PolimorfismoProfesion{

   public static void main(String []args){

//se declaran profesionistas
      Profesionista Carla, Miguel;
// se instancia Doctor y Arquitecto
      Carla = new Doctor("Doctora", "Carla");
      Miguel = new Arquitecto("Arquitecto", "Miguel");
      Carla.printMensaje();
      Miguel.printMensaje();

   }
}
```

Se comprueba herencia pues los hijos muestran comportamiento del padre y de ellos mismos

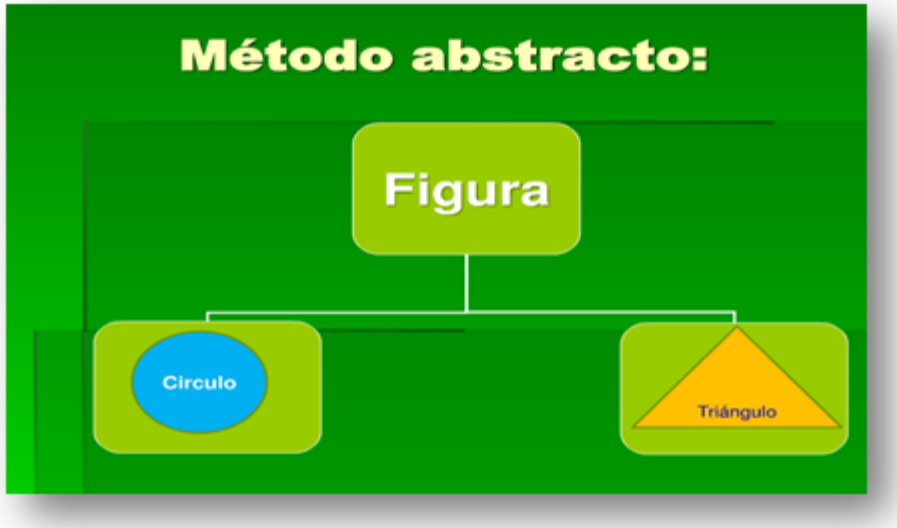
```
General Output
----------------------Configuration: <Default>----------
Soy un Profesionista con especialidad de: Doctora
Me llamo Carla y soy doctora que cura pacientes
Soy un Profesionista con especialidad de: Arquitecto
Me llamo Miguel y soy arquitecto que diseña planos

Process completed.
```

CLASE ABSTRACTA

- Jerarquía de clases en que algún comportamiento está presente en todas ellas pero se materializa de forma distinta para cada una.
- Para resolver esta problemática Java proporciona las clases y métodos abstractos. Un método abstracto es un método declarado en una clase para el cual esa clase no proporciona la implementación (el código).
- Una clase abstracta es una clase que tiene al menos un método abstracto.
- No se pueden definir constructores abstractos
- No se pueden definir métodos estáticos abstractos.
- Las subclases de una clase abstracta deben:
  - Sobreescribir todos los métodos abstractos de la superclase
  - O ser declaradas también como clases abstractas

CLASE ABSTRACTA

- El método área de circuloy triangulo se implementa de manera diferente
- No importa qué forma tiene un objeto, aplicándole el método área devolverá el resultado correcto



Fuente: http://lia1620.blogspot.com/2013/07/polimorfismo-y-clases-abstractas.html

SINTAXIS

```
acceso abstract  class nombre_clase {

……..

}
```

Ejemplo

```
 public abstract  class animal{

      public abstract void comer();

}
```

EJEMPLO

```
public abstract class Forma {
    private int xpos, ypos;
    private Color color;
    // ...
    public abstract void dibuja();   // no tiene cuerpo
    public void setColor(Color c){ /*...*/ };
 }
public class Circulo extends Forma{
  private int radio;
  // ...
  public void dibuja(){ /*...*/ };   // debe tener cuerpo o ser abstracta
  public void setRadio(int){/*...*/};
 }


public class triangulo extends Forma{
 private int altura, anchura;
 // ...
 public void dibuja(){ /*...*/ };   // debe tener cuerpo o ser abstracta
 public void setAltura(int){/*...*/};
}
```

EJEMPLO

```
abstract class Figura {
 protected int x, y;
 public void mostrarOrigen() {
 System.out.println("x= "+x+" y= "+y);}
 public abstract double area(); // No tiene implementación
 public abstract void mostrarNombre(); //No tiene implementación
}
public class Triangulo extends Figura {
 protected int base, altura;
 public Triangulo (int ba, int al) { base=ba; altura=al; }
 public double area() { return base*altura/2; }
 public void mostrarNombre() { System.out.println("triangulo"); }
}
public class Cuadrado extends Figura {
 protected int lado;
 public Cuadrado (int lado) { this.lado=lado; }
 public double area() { return lado*lado; }
 public void mostrarNombre() { System.out.println("cuadrado");}
 }
```

```
public class SinCuerpo{

   public static void main(String
[]args){
      //Figura x = new Figura();  error
pues Figura no puede ser instanciada
      Triangulo ft = new Triangulo(4,7);
      Cuadrado fc = new Cuadrado(5);
      ft.mostrarNombre();
      fc.mostrarNombre();

   }
}
```

```
General Output
--------------------Configura
Soy la Figura triangulo
Soy la Figura cuadrado

Process completed.
```

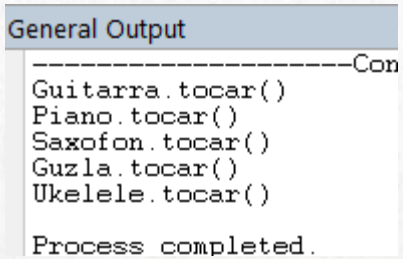
EJEMPLO

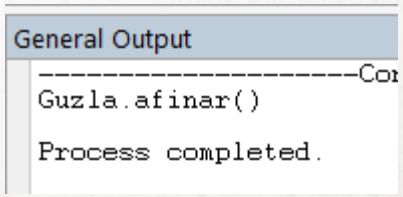
```
import java.util.*;
 abstract  class Instrumento {
   public abstract void tocar();
   public String tipo() {
return "Instrumento";
   }
   public abstract void afinar();
}
class Guitarra extends Instrumento {
 public void tocar() {
System.out.println("Guitarra.tocar()");
   } @Override
   public String tipo() { return "Guitarra"; }
   public void afinar(){}
}
class Piano extends Instrumento {
   public void tocar() {
System.out.println("Piano.tocar()");
   } @Override
   public String tipo() { return "Piano"; }
  public void afinar() {}
}
```

```
class Saxofon extends Instrumento {
   public void tocar() {
            System.out.println("Saxofon.tocar()");
   } @Override
   public String tipo() { return "Saxofon"; }
   public void afinar() {}
}
// Un tipo de Guitarra
class Guzla extends Guitarra {@Override
      public void tocar() {
       System.out.println("Guzla.tocar()");
      }      @Override
      public void afinar() {
       System.out.println("Guzla.afinar()");
      }
     }
```

```
// Un tipo de Guitarra
class Ukelele extends Guitarra {@Override
   public void tocar() {
System.out.println("Ukelele.tocar()");
   } @Override
   public String tipo() { return "Ukelele"; }
}
public class Concierto {
   // No importa el tipo de Instrumento,
   // seguirá funcionando debido a Polimorfismo:
   static void afinar(Instrumento i) {
// ...
i.tocar();
   }
   static void afinarTodo(Instrumento[] e) {
            for(int i = 0; i < e.length; i++)
   afinar(e[i]);
   }
```

```
public static void main(String[] args) {
    // Declarar un Arreglo SIN INSTANCIAS es valido en Clases
Abstractas
Instrumento[] orquesta = new Instrumento[5];
    // Generar una INSTANCIA de una la Clase Abstracta no es valido
    // Instrumento nuevo = new Instrumento();
          int i = 0;
          orquesta[i++] = new Guitarra();
          orquesta[i++] = new Piano();
          orquesta[i++] = new Saxofon();
          orquesta[i++] = new Guzla();
          orquesta[i++] = new Ukelele();
          afinarTodo(orquesta);
 } }
```





DESCRIPCIÓN

Clase Concierto

En el código fuente de Concierto.java son diseñadas diversas Clases que demuestran el uso de Polimorfismo:

1. Instrumento: Es utilizada como la Clase Base para el resto de Clases y en ella son definidos tres métodos: tocar, tipo y afinar.
2. Guitarra, Piano y Saxofon: Heredan de la Clase Instrumento y redefinen los métodos de ésta.
3. Guzla y Ukelele: Heredan de la Clase Guitarra y redefinen los métodos de ésta.
4. Las definiciones de la Clase principal Conciewrto son descritas a continuación:
   a) El método afinar, toma como valor de entrada una referencia del tipo Instrumento, sobre la cual es invocado el método tocar.
   b) El método afinarTodo, toma como valor de inicio un arreglo de Instrumento, el cual es procesado por un ciclo que a su vez manda llamar el método afinar con los respectivos valores del arreglo.

- Dentro del método principal se define lo siguiente:
  1. Primero se genera un arreglo de Instrumento para 5 Objetos.
  2. Se inicializa la variable i con un valor de cero.
  3. A través de la referencia orquesta son asignados distintos Objetos al arreglo, nótese que aunque el arreglo es de tipo Instrumento es posible asignar los Objetos: Guitarra, Piano, Saxofon, Guzla y Ukelele.
  4. Finalmente se invoca el método afinarTodo con la referencia que representa el arreglo de Instrumento.

- Lo interesante de este resultado reside en la manera que fue realizada la invocación de métodos, el método que realiza las llamadas hacia las diversas clases es afinar dentro de la Clase Concierto, observe que este método toma como valor inicial una referencia del tipo Instrumento y se invoca el método tocar.

- Sin embargo, al ser invocado el método tocar a pesar de ser a través de una referencia Instrumento es invocado el método de Clase acordemente, esto es, se llama el método de la Clase especifica; lo anterior es una labor de Polimorfismo ya que ocurre el comportamiento adecuado a pesar de realizarse las llamadas a través de una clase general (Instrumento).

ACTIVIDAD 9

1. Editar y probar clase Padre diapositiva 3
2. Editar y probar clase Profesionista diapositiva 4
3. Editar y probar clase Figura diapositiva 10
4. Editar y probar clase Concierto diapositiva 11
5. Terminar ejercicio 2 de la práctica 8.1, la clase cuenta bancaria