Programación Orientada a Objetos

Polimorfismo

Fundamentos de la POO

- Abstracción
- Encapsulamiento y ocultación de datos
- Herencia (generalización o especialización)
- Polimorfismo
- Reutilización

Java
Polimorfismo

POLIMORFISMO

- Este termino se utiliza en la POO para "**referirse a la propiedad por la que es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos**"
- El polimorfismo es un concepto de la programación orientada a objetos que nos permite programar en **forma general**, y posteriormente hacerlo en las clases hija de manera **específica.**
- Así también, nos sirve para programar objetos con **características comunes y que todos estos compartan la misma superclase en una jerarquía de clases**, como si todas fueran objetos de la superclase. Esto nos simplifica la programación.

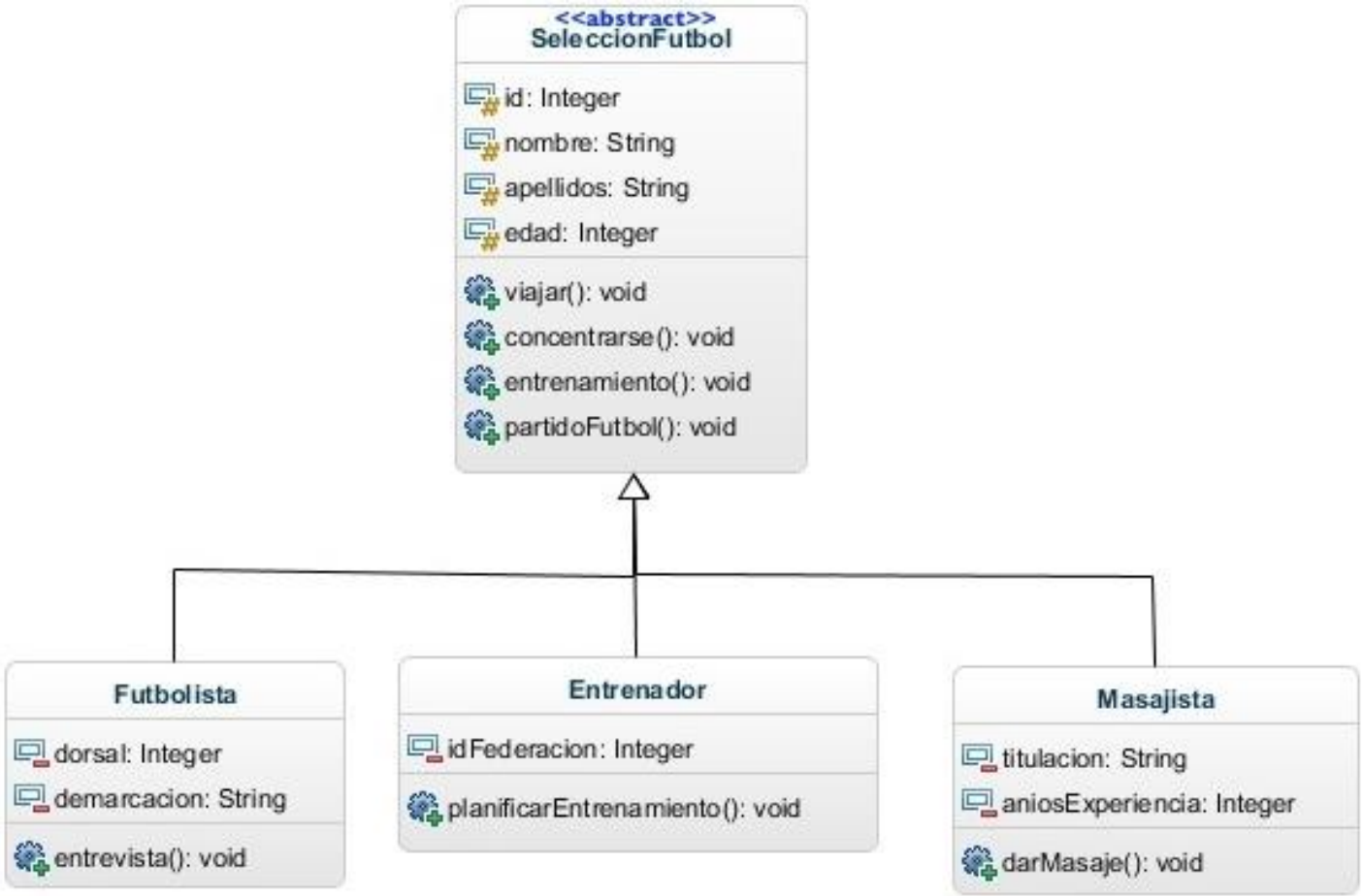
Existen varias formas de polimorfismo:

- Cuando invocamos el **mismo nombre de método** sobre instancias **de distinta clase**.
- Cuando creamos **múltiples constructores.**
- Cuando vía subtipo asignamos una instancia de una **subclase a una referencia a la clase base**.
- Por lo cual se dice que para implementar en Java se establece el polimorfismo:
  - Herencia y métodos en distintas clases.
  - Herencia usando Clases Abstractas y métodos abstractos

Polimorfismo en java

Usando Clases abstractas y métodos abstractos y herencia entre clases

POLIMORFISMO



Polimorfismo en Java

- En java el polimorfismo usa las clases abstractas

Clase abstracta

```
Abstract public class SelecciónFutbol
{
        //Variables de instancia //atributos
        //Variables de clase
        //Constantes//final
        //Métodos de la clase
        public void Mostrar()
        {.....}
        //Métodos abstractactos de la clase sin cuerpo que se definen en
        //la clase hija o derivada
        abstract public void Nombre_método();
}
```

Clases Abstractas

- Una clase abstracta es una clase **de la que no se puede** crear objetos.
- La **utilidad** de estas clases estriba en que otras clases hereden de ésta, por lo que con ello conseguiremos **reutilizar código**, siendo **más eficiente** la programación.
- Para declarar una clase como abstracta utilizamos la palabra clave: **abstract.**

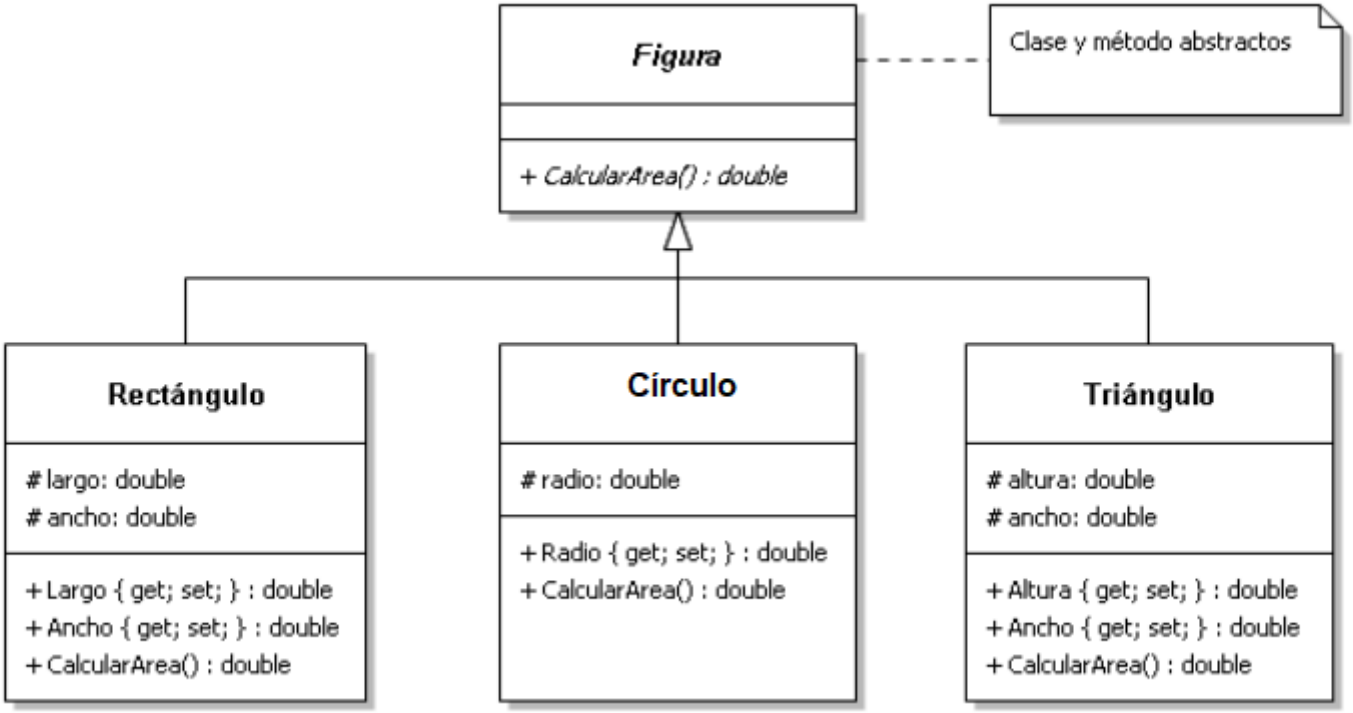
Características de una Clase Abstracta

- Pueden tener variables de instancia y métodos propios de la clase.
- Tienen al menos un método abstracto declarado.
- Todas las subclases (hijas o dervidas) que hereden de una clase abstracta tendrán que redefinir los métodos abstractos dándoles una implementación (cuerpo/instrucciones del método)
- En caso de que se tengan varios niveles jerárquicos, en el nivel bajo, en la clase hija o derivada se implementa el método abstracto.

Métodos Abstractos

- Son aquellos que se declaran en un clase abstracta y no se implementan en dicha clase.
- Los métodos abstractos solo se pueden implementar en las clases hijas o derivadas, es decir que no son abstractas, y que recibe este método por medio de la herencia
- Los métodos abstractos su cuerpo o conjunto de instrucciones se reescriben en la clase hija o derivada para realizar las acciones o tareas para demostrar los comportamientos de los objetos o instancias.

Ejemplo 1. Figura



Pasos para el elaboración:

a) Crear una clase Figura, abstracta
b) Añadirle los métodos abstractos calcularArea(); NO debe llevar ningún cuerpo o instrucciones, recuerda termina en ";"
c) Crear la clase Rectángulo y realizar la relación de herencia de Figura
d) Añadir los atributos de la clase Rectángulo
e) Anadir el método calcularArea() y el cuerpo del método, coloca las instrucciones de la formula
f) Crear la clase Circulo y realizar la relación de herencia de Figura (extends)
g) Añadimos atributo radio protegido
h) Añadimos el método calcularArea() y coloca las instrucciones de la formula
i) Creamos clase EjecutaFigura, con método main() y el cuerpo del método
j) Instancia tres objetos de cada clase

```
abstract public class Figura
{

public Figura()
{
}

// método asbtracto declarado solamente
public  abstract double calcularArea();

}
```

```
public class Rectangulo extends Figura
{
 double largo, ancho;

public Rectangulo()
{
}

public void setLargo(double largo1)
{
    largo=largo1;

}

 public double  getLargo()
{
    return largo;


}

public void setancho(double ancho1)
{
       ancho=ancho1;

}
public double getancho ()
{
    return ancho;

}
// método abstracto implementado

 public  double calcularArea()
 {
    return (largo * ancho );
 }

}
```

```
public class Circulo extends Figura
{
  // variables de instancia
  double radio;

  public Circulo()
  {
  }

  // método de la propia clase
  public void setRadio(double radio1)
  {
      radio=radio1;


  }

 public double getRadio()
  {
      return radio;


  }
  // método abstracto implementado
   public  double calcularArea()
   {
      return Math.PI*(radio*radio);
   }
```

```
public class Triangulo extends Figura

{
  //variables de instancia
  double altura, base;

   public Triangulo()
   {
   }

   public void setAltura(double altura1)
   {
        altura=altura1;

   }

    public double getAltura()
   {
       return altura;


   }

   public void setBase(double base1)
   {
          base=base1;


   }
   public double getBase ()
   {
       return base;

   }

    public  double calcularArea()
    {
       return (altura * base )/2;
    }

}
```

```java
public class EjecutaFiguras
{

    public static void main(String[] args)
    {
        Rectangulo r1 = new Rectangulo();
        r1.setancho(3.0);
        r1.setLargo(6.0);
        Rectangulo r2 = new Rectangulo();
        r2.setancho(4.0);
        r2.setLargo(5.0);
        System.out.println("RECTANGULOS");
        System.out.println("Area");
        System.out.println("Rectángulo base " + r1.getancho()+ " y altura " +   r1.getLargo() + " es " + r1.calcularArea());
        System.out.println("Rectángulo base " + r2.getancho()+ " y altura " +   r2.getLargo() + " es " + r2.calcularArea());
        Circulo c1 = new Circulo();
        c1.setRadio(2.0);
        Circulo c2 = new Circulo();
        System.out.println("CIRCULOS" );
        c2.setRadio(3.0);
        System.out.println("Area");
        System.out.println("Circulo radio: "+ c1.getRadio()+ " :" +c1.calcularArea());
        System.out.println("Circulo radio: "+ c2.getRadio()+ " :" +c2.calcularArea());
        Triangulo t1= new Triangulo();
        t1.setBase (6.5);
        t1.setAltura(8.5);
        Triangulo t2= new Triangulo();
        t2.setBase (9);
        t2.setAltura(2);
        System.out.println("TRIANGULOS");
        System.out.println("Area");
        System.out.println("Triángulo base " + t1.getBase()+ " y altura " + t1.getAltura() + " es " + t1.calcularArea());
        System.out.println("Triángulo base " + t2.getBase()+ " y altura " + t2.getAltura() + " es " + t2.calcularArea());
    }
```

```
RECTANGULOS
Area
Rectángulo base 3.0 y altura 6.0 es 18.0
Rectángulo base 4.0 y altura 5.0 es 20.0
CIRCULOS
Area
Circulo radio: 2.0 :12.566370614359172
Circulo radio: 3.0 :28.274333882308138
TRIANGULOS
Area
Triángulo base 6.5 y altura 8.5 es 27.625
Triángulo base 9.0 y altura 2.0 es 9.0

Process completed.
```

Realiza los cambios

- Agrega el método calcularPerimetro( ) de forma abstracta en Figuras.
- Implementa el método en cada clase hija
- Crea una clase de tipo figura e instanciar en EjecutaFigura
- Rediseña el diagrama de clases UML

Ejemplo 2. Figuras Cuadriláteros

```
public abstract class FiguraCuadrilatero
{
protected int x, y, ancho, alto;
public void cambiaX(int x)
{  this.x =  x;
}
public void cambiaY(int y)
{
this.y =  y;
}
public void cambiaAncho(int ancho)
{
this.ancho =  ancho;
}
public void cambiaAlto(int alto)
{
this.alto =  alto;
}
public abstract float obtenPerimetro();
public abstract double obtenArea();
}
```

Clase Cuadrado

```
public class Cuadrado extends FiguraCuadrilatero
{

    public float obtenPerimetro()

    {  return 4 *  ancho;
    }

    public double obtenArea()
    {  return ancho * ancho;}
}
```

Clase Rectangulo

```
public class Rectangulo extends Figura
{

        public float obtenPerimetro()
        {
        return 2 *  ancho +  2 * alto;
        }
        public float obtenArea()
        {
        return ancho * alto;
        }
}
```

Clase Principal

```
public class EjecutaFigurasCuadrilateras
 {
public static void main(String[] args)
{
        Cuadrado c= new Cuadrado(); c.cambiaAncho(10);
        c.cambiaAlto(10);
        Rectangulo r = new Rectangulo(); r.cambiaAncho(20);
        r.cambiaAlto(30);
       System.out.println("Perimetro Cuadrado = " + c.obtenPerimetro());
       System.out.println("Area Cuadrado = " + c.obtenArea());
       System.out.println("Perimetro Rectangulo = " + r.obtenPerimetro());
       System.out.println("Area Rectangulo = " + r.obtenArea());
}
}
```

Realizar los cambios:

- Agrega otro dos cuadriláteros
- Obtener tanto perímetro como el área
- Realiza el diagrama de clases en UML
- Realiza una versión 2, para implementar la lectura de datos usando scanner.

Polimorfismo en Java

Usando métodos con el mismo nombre sobre objetos de distinta clase y herencia

Ejemplo: Polimorfismo

```
public class Animal
{
   // variables de instancia
   private int peso = 0;
     public void cambiaPeso(int peso)
     {  this.peso = peso;
     }
     public int obtenPeso()
     {  return peso;}

     public String habla()
      {return "Los animales no hablan";}
}
```

Clase Vaca

```
public class Vaca extends Animal
{
      public String habla()
     {
     return "MUU";
     }
}
```

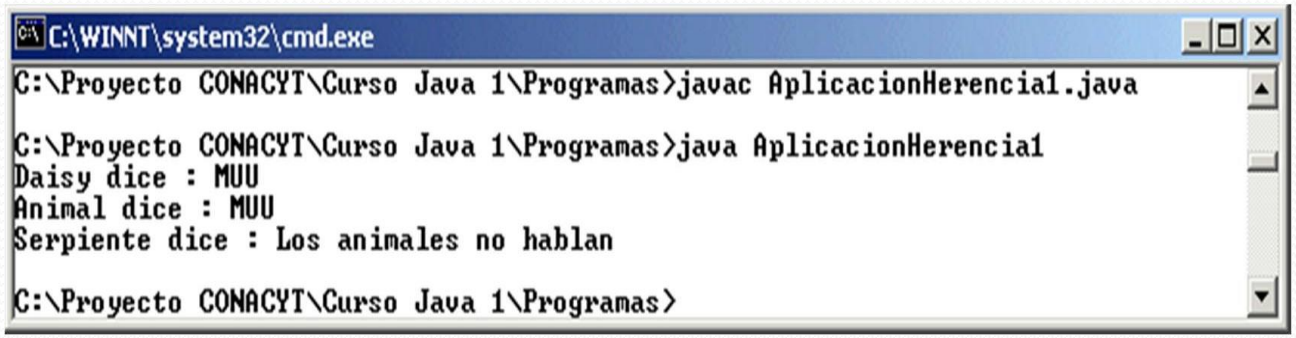
Clase Cerdo

```
public class Cerdo extends Animal
{
        public String habla()
      {  return "OINC";
      }
}
```

Clase Serpiente

```
public class Serpiente extends Animal
{
}
```

```
import java.io.*;
public class AplicacionHerencia1
{
public static void main(String[] args)
{ Vaca daisy = new Vaca();
Animal animal = new Vaca();
Serpiente serpiente = new Serpiente();
System.out.println("Daisy dice : " + daisy.habla() );
System.out.println("Animal dice : " + animal.habla() );
System.out.println("Serpiente dice : " + serpiente.habla() );
}
}
```



Realiza las modificaciones:

- Agrega otra dos clases y dos métodos que puedan demostrar el polimorfismo en todas las clases diseñadas
  - Ejemplo de métodos: comer( )…. dormir( )….

- Realiza el diagrama de clases en UML

Ejercicio: Relaciona cada concepto

Polimorfismo

100 PUNTOS

Abstracto

MÉTODOS

Herencia

Abstractas

ATRIBUTOS

- Es la propiedad que nos permite programar en forma general, en lugar de hacerlo en forma específica

  _____

- El polimorfismo usa clases_____
- Se le llama método _____ya que se redefine en las clases hijas
- Los atributos y métodos de la clase abstracta se pueden accederse de la clase hija por _____
- Se heredan _____ y _____

Reflexionamos y contesta

- ¿Qué es una clase abstracta?
- ¿Qué es un método abstracto?
- ¿Cómo se representa una clase abstracta en UML?

Actividad Colaborativa: Reporte Colaborativo

Realiza los siguientes diagramas de clases:

- Modela la Clase Persona que tiene tres atributos: Nombre, Edad y Genero
- Modela la Clase Cliente que tiene tres atributos: No_cliente, dirección y teléfono
- Modela la Clase Vendedor que tiene tres atributos: No_vendedor, comision, departamento
- Todos tienen métodos set/get para cada atributo
- Todos muestran los datos de cada cliente
- Todos leen los datos de cada clase usando el constructor
- Modela la clase EjecutaPersonas

Realiza la implementación en Java