Clases Abstractas

Definición

Ejemplos

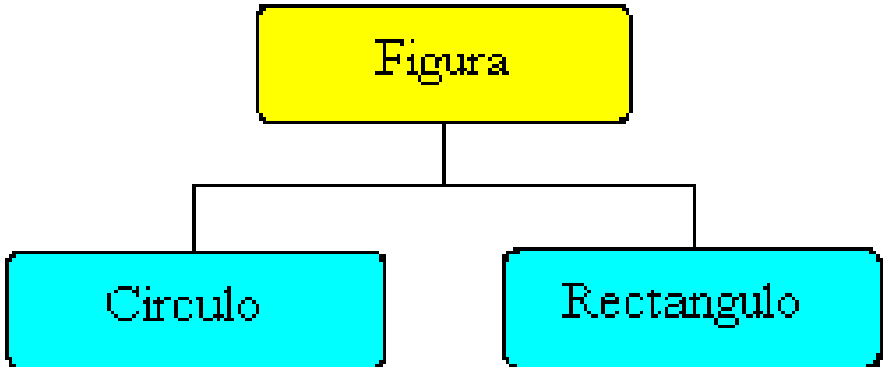
Definición

- Una clase abstracta es una clase de la que NO se pueden crear objetos.
- Su utilidad es permitir que otras clases deriven de ella, proporcionándoles un modelo a seguir y algunos métodos de utilidad general.
- Se declaran anteponiendo la palabra abstract:

  public abstract class Geometria{.....}
- Una clase abstract puede tener métodos declarados como abstract, en cuyo caso no se da definición del método.
- Si una clase tiene algún método abstract, es obligatorio que la clase sea abstract.

Ejemplo: Figuras planas

- El rectángulo y el círculo tienen características comunes: posición de la figura, área,..
- Podemos diseñar una jerarquía de clases, tal que la clase base denominada *Figura*, tenga las características comunes y cada clase derivada las específicas. La relación jerárquica se muestra en la figura:



Ejemplo: Figuras planas

- La clase ***Figura*** es la que contiene las características comunes a dichas figuras concretas. Declararemos ***Figura*** como una clase abstracta, y tendrá la función miembro ***area*** **abstract**.
- Las clases abstractas solamente se pueden usar como clases base para otras clases.

Ejemplo: La clase Figura

- La definición de la clase abstracta *Figura*, contiene la posición *x* e *y* de la figura particular, y la función *area*, que se va a definir en las clases derivadas para calcular el área de cada figura en particular.

```
public abstract class Figura {
    protected int x;
    protected int y;
    public Figura(int x, int y) {
            this.x=x; this.y=y; }
    public abstract double area( );
}
```

Ejemplo: Clase Rectángulo

- Las clases derivadas heredan los miembros dato *x* e *y* de la clase base, y definen la función *area*, declarada **abstract** en la clase base *Figura*, ya que cada figura particular tiene una fórmula distinta para calcular su área.
- Por ejemplo, la clase derivada *Rectangulo*, tiene como datos su anchura *ancho* y altura *alto*.

```
class Rectangulo extends Figura{
    protected double ancho, alto;
    public Rectangulo(int x, int y, double ancho, double alto){
            super(x,y);
            this.ancho=ancho;
            this.alto=alto; }
    public double area(){
    return ancho*alto;
    }
}
```

Ejemplo: Clase círculo

```
class Circulo extends Figura{
   protected double radio; public Circulo(int x,
   int y, double radio){
        super(x,y);
        this.radio=radio;
   }
   public double area(){
        return Math.PI*radio*radio;
   }
}
```

Uso de la jerarquía de clases

- Creamos un objeto *c* de la clase *Circulo* situado en el punto (0, 0) y de 5.5 unidades de radio. Calculamos y mostramos el valor de su área.

```
Circulo c=new Circulo(0, 0, 5.5);
System.out.println("Area del círculo "+c.area());
```

- Creamos un objeto *r* de la clase *Rectangulo* situado en el punto (0, 0) y de dimensiones 5.5 de anchura y 2 unidades de largo. Calculamos y mostramos el valor de su área.

```
Rectangulo r=new Rectangulo(0, 0, 5.5, 2.0);
System.out.println("Area del rectángulo "+r.area());
```

- Veamos ahora, una forma alternativa, guardamos el valor devuelto por **new** al crear objetos de las clases derivadas en una variable *f* del tipo *Figura* (clase base).

```
Figura f=new Circulo(0, 0, 5.5);
System.out.println("Area del círculo "+f.area());
f=new Rectangulo(0, 0, 5.5, 2.0);
System.out.println("Area del rectángulo "+f.area());
```

Polimorfismo

- Creamos un array de la clase base *Figura,* guardamos en sus elementos los valores devueltos por **new** al crear objetos de las clases derivadas.

  Figura[] fig=new Figura[4];
  fig[0]=new Rectangulo(0,0, 5.0, 7.0);
  fig[1]=new Circulo(0,0, 5.0);
  fig[2]=new Circulo(0, 0, 7.0);
  fig[3]=new Rectangulo(0,0, 4.0, 6.0);
- La sentencia fig[i].area(); ¿a qué función *area* llamará?.
- La respuesta será, según sea el índice *i*. Si *i* es cero, el primer elemento del array guarda una referencia a un objeto de la clase *Rectangulo*, luego llamará a la función miembro *area* de *Rectangulo*. Si *i* es uno, el segundo elemento del array guarda una referencia un objeto de la clase *Circulo*, luego llamará también a la función *area* de *Circulo*, y así sucesivamente.
- Pero podemos introducir el valor del índice *i*, a través del teclado, o seleccionando un control en un applet, en el momento en el que se ejecuta el programa. Luego, la decisión sobre qué función *area* se va a llamar se retrasa hasta el tiempo de ejecución.

El polimorfismo en acción

- Supongamos que deseamos saber la figura que tiene mayor área independientemente de su forma. Primero, programamos una función que halle el mayor de varios números reales positivos.

```
double valorMayor(double[ ] x){
    double mayor=0.0;
    for (int i=0; i<x.length; i++){
       if(x[i]>mayor){
            mayor=x[i];
      }
     }
    return mayor;
 }
```

- Ahora, la llamada a la función ***valorMayor***

```
double numeros[ ]={3.2, 3.0, 5.4, 1.2};
System.out.println("El valor mayor es "+valorMayor(numeros));
```

- La función ***areaMayor*** que compara el área de figuras planas es semejante a la función ***valorMayor*** anteriormente definida, se le pasa el array de objetos de la clase base ***Figura***. La función devuelve un valor double que indica el área mayor.

El polimorfismo en acción

- Ahora, la llamada a la función *valorMayor*

```
double numeros[ ]={3.2, 3.0, 5.4, 1.2};
System.out.println("El valor mayor es "+valorMayor(numeros));
```

- La función *areaMayor* que compara el área de figuras planas es semejante a la función *valorMayor* anteriormente definida, se le pasa el array de objetos de la clase base *Figura*. La función devuelve una referencia al objeto cuya área es la mayor.

```
static double areaMayor(Figura[] figuras){
        double areaMayor=0.0;
        for(int i=0; i<figuras.length; i++){
                if(figuras[i].area()>areaMayor){
                        areaMayor=figuras[i].area();
                }
         }
         return areaMayor;
}
```

El polimorfismo en acción

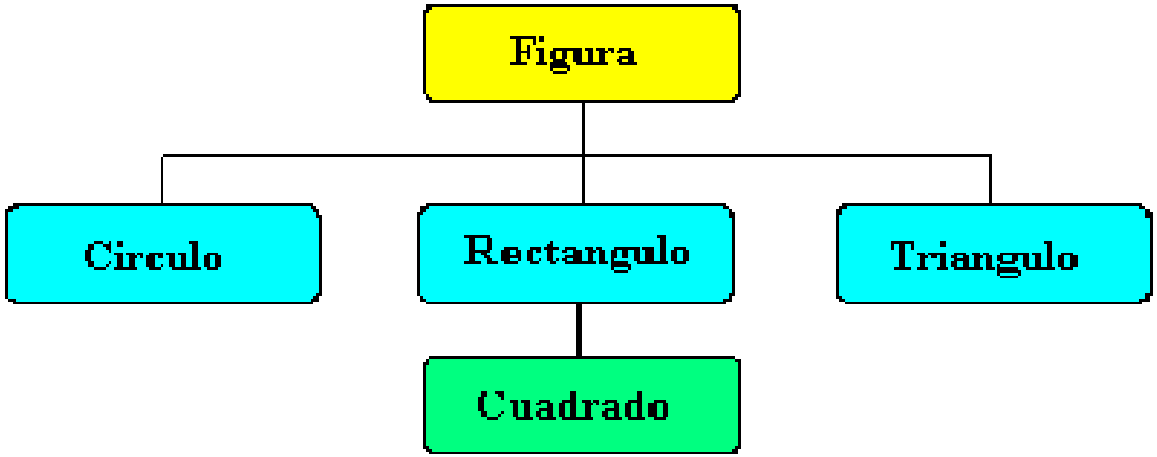
- Ventaja: La función *areaMayor* está definida en términos de variable *figuras* de la clase base *Figura*, por tanto, trabaja no solamente para una colección de círculos y rectángulos, sino también para cualquier otra figura derivada de la clase base *Figura*.
- Si se deriva *Triangulo* de *Figura*, y se añade a la jerarquía de clases, la función *areaMayor* podrá manejar objetos de dicha clase, sin modificar para nada el código de la misma.
- Veamos ahora la llamada a la función *areaMayor*

```
Figura[] fig=new Figura[4];
fig[0]=new Rectangulo(0,0, 5.0, 7.0);
fig[1]=new Circulo(0,0, 5.0);
fig[2]=new Circulo(0, 0, 7.0);
fig[3]=new Rectangulo(0,0, 4.0, 6.0);
double fMayor=areaMayor(fig);
System.out.println("El área mayor es "+fMayor);
```

El polimorfismo es, por tanto, la técnica que permite pasar un objeto de una clase derivada a funciones que conocen el objeto solamente por su clase base.

Ejemplo: Ampliación de clases

- Ampliamos el árbol jerárquico de las clases que describen las figuras planas regulares, para acomodar a dos clases que describen las figuras planas, triángulo y cuadrado. La relación jerárquica es:



Clase Cuadrado

- La clase *Cuadrado* es una clase especializada de *Rectangulo*, ya que un cuadrado tiene los lados iguales.
- El constructor solamente precisa de tres argumentos los que corresponden a la posición de la figura y a la longitud del lado.

```
class Cuadrado extends Rectangulo{
   public Cuadrado(int x,int y, double dimension){
         super(x, y, dimension, dimension);
    }
}
```

Clase Triangulo

- La clase derivada *Triángulo*, tiene como datos, aparte de su posición *(x, y)* en el plano, la base y la altura del triángulo.

```
class Triangulo extends Figura{
    protected double base, altura;
    public Triangulo(int x, int y, double base, double
    altura){
          super(x, y);
          this.base=base;
          this.altura=altura;
    }
    public double area(){
          return base*altura/2;
     }
}
```

Utilización de todas las clases

```
Figura[] fig=new Figura[4];
fig[0]=new Rectangulo(0,0, 5.0, 2.0);
fig[1]=new Circulo(0,0, 3.0);
fig[2]=new Cuadrado(0, 0, 5.0);
fig[3]=new Triangulo(0,0, 7.0, 12.0);
double fMayor=areaMayor(fig);
System.out.println("El área mayor es
   "+fMayor);
```

Clases finales

- Se puede declarar una clase como ***final***, cuando no nos interesa crear clases derivadas de dicha clase. La clase ***Cuadrado*** se puede declarar como **final**, ya que no se espera que ningún programador necesite crear clases derivadas de *Cuadrado*.

```
final class Cuadrado extends Rectangulo{
   public Cuadrado(int x, double dimension){
        super(x, x, dimension, dimension);
   }
}
```

- Un método es final si NO se puede sobresecribir.

Resumen

- La herencia es la propiedad que permite la creación de nuevas clases a partir de clases ya existentes. La clase derivada hereda los datos y las funciones miembro de la clase base, y puede redefinir algunas de las funciones miembro o definir otras nuevas, para ampliar la funcionalidad que ha recibido de la clase base.
- Para crear un objeto de la clase derivada se llama primero al constructor de la clase base mediante la palabra reservada **super**. Luego, se inicializa los miembros dato de dicha clase derivada.
- El polimorfismo se implementa por medio de las funciones abstractas, en las clases derivadas se declara y se define una función que tiene el mismo nombre, el mismo número de parámetros y del mismo tipo que en la clase base, pero que da lugar a un comportamiento distinto, específico de los objetos de la clase derivada.
- Enlace dinámico significa que la decisión sobre la función a llamar se demora hasta el tiempo de ejecución del programa.
- No se pueden crear objetos de una clase abstracta pero si se pueden declarar referencias en las que guardamos el valor devuelto por **new** al crear objetos de las clases derivadas. Esta peculiaridad nos permite pasar un objeto de una clase derivada a una función que conoce el objeto solamente por su clase base. De este modo podemos ampliar la jerarquía de clases sin modificar el código de las funciones que manipulan los objetos de las clases de la jerarquía.