

Conceptos básicos de la Programación Orientada a Objetos

Cuando se escribe un programa en un lenguaje orientado a objetos, definimos una plantilla o clase que describe las características y el comportamiento de un conjunto de objetos similares. La clase automóvil describe las características comunes de todos los automóviles: sus atributos y su comportamiento. Los atributos o propiedades se refieren a la marca o fabricante, el color, las dimensiones, si tienen dos, tres, cuatro o más puertas, la potencia, si utiliza como combustible la gasolina o gasoil, etc. El comportamiento se refiere a la posibilidad de desplazarse por una carretera, frenar, acelerar, cambiar de marcha, girar, etc.

Luego, tenemos automóviles concretos, por ejemplo, el automóvil propio de una determinada marca, color, potencia, etc, el automóvil del vecino de otra marca, de otro color, etc., el automóvil de un amigo, etc.

- Una clase es por tanto una plantilla implementada en software que describe un conjunto de objetos con atributos y comportamiento similares.
- Una instancia u objeto de una clase es una representación concreta y específica de una clase y que reside en la memoria del ordenador.

Atributos

Los atributos son las características individuales que diferencian un objeto de otro y determinan su apariencia, estado u otras cualidades. Los atributos se guardan en variables denominadas de instancia y cada objeto particular puede tener valores distintos para estas variables.

Las variables de instancia también denominados miembros dato, son declaradas en la clase pero sus valores son fijados y cambiados en el objeto.

Además de las variables de instancia hay variables de clase, las cuales se aplican a la clase y a todas sus instancias. Por ejemplo, el número de ruedas de un automóvil es el mismo cuatro, para todos los automóviles.

Comportamiento

El comportamiento de los objetos de una clase se implementa mediante funciones miembro o métodos. Un método es un conjunto de instrucciones que realizan una determinada tarea y son similares a las funciones de los lenguajes estructurados.

Del mismo modo que hay variables de instancia y de clase, también hay métodos de instancia y de clase. En el primer caso, un objeto llama a un método para realizar una determinada tarea, en el segundo, el método se llama desde la propia clase.

El proyecto

El proyecto consta de dos archivos, el primero contiene la clase *Rectangulo* que se guarda en el archivo *Rectangulo.java* y no tiene el método *main*.

La otra clase, es la que describe la aplicación *RectanguloApp1* y se guarda en el archivo *RectanguloApp1.java*, esta clase tiene que tener el método *main*.

Un proyecto puede constar de varias clases (normalmente se sitúa cada clase en un archivo) pero solamente una tiene el método *main* y representa la aplicación. Para distinguir la clase que describe la aplicación de las demás le hemos añadido el sufijo **App**.

La clase

Para crear una clase se utiliza la palabra reservada **class** y a continuación el nombre de la clase. La definición de la clase se pone entre las llaves de apertura y cierre. El nombre de la clase empieza por letra mayúscula.

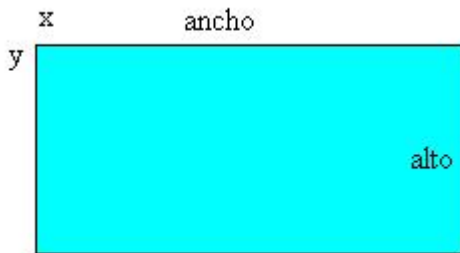
```
class Rectangulo{  
    //miembros dato  
    //funciones miembro  
}
```

Los miembros dato

Los valores de los atributos se guardan en los miembros dato o variables de instancia. Los nombres de dichas variables comienzan por letra minúscula.

Vamos a crear una clase denominada *Rectangulo*, que describa las características comunes a estas figuras planas que son las siguientes:

- El origen del rectángulo: el origen o posición de la esquina superior izquierda del rectángulo en el plano determinado por dos números enteros *x* e *y*.
- Las dimensiones del rectángulo: *ancho* y *alto*, otros dos números enteros.



```
class Rectangulo{  
    int x;  
    int y;  
    int ancho;
```

```

        int alto;
        //faltan las funciones miembro
    }

```

Las funciones miembro

En el lenguaje C++ las funciones miembro se declaran, se definen y se llaman. En el lenguaje Java las funciones miembro o métodos solamente se definen y se llaman.

El nombre de las funciones miembro o métodos comienza por letra minúscula y deben sugerir acciones (mover, calcular, etc.). La definición de una función tiene el siguiente formato:

```

tipo nombreFuncion(tipo parm1, tipo parm2, tipo parm3){
    //...sentencias
}

```

Entre las llaves de apertura y cierre se coloca la definición de la función. *tipo* indica el tipo de dato que puede ser predefinido **int**, **double**, etc, o definido por el usuario, una clase cualquiera.

Para llamar a un función miembro o método se escribe

```

retorno=objeto.nombreFuncion(arg1, arg2, arg3);

```

Cuando se llama a la función, los argumentos *arg1*, *arg2*, *arg3* se copian en los parámetros *parm1*, *parm2*, *parm3* y se ejecutan las sentencias dentro de la función. La función finaliza cuando se llega al final de su bloque de definición o cuando encuentra una sentencia **return**.

Cuando se llama a la función, el valor devuelto mediante la sentencia **return** se asigna a la variable *retorno*.

Cuando una función no devuelve nada se dice de tipo **void**. Para llamar a la función, se escribe

```

objeto.nombreFuncion(arg1, arg2, arg3);

```

Se ha de tener en cuenta que las funciones miembro tienen acceso a los miembros dato, por tanto, es importante en el diseño de una clase decidir qué variables son miembros dato, qué variables son locales a las funciones miembro y qué valores les pasamos a dichas funciones. Los ejemplos nos ayudarán a entender esta distinción.

Hemos definido los atributos o miembros dato de la clase *Rectangulo*, ahora le vamos añadir un comportamiento: los objetos de la clase *Rectangulo* o rectángulos sabrán calcular su área, tendrán capacidad para trasladarse a otro punto del plano, sabrán si contienen en su interior un punto determinado del plano.

La función que calcula el área realizará la siguiente tarea, calculará el producto del ancho por el alto del rectángulo y devolverá el resultado. La función devuelve un entero

es por tanto, de tipo **int**. No es necesario pasarle datos ya que tiene acceso a los miembros dato *ancho* y *alto* que guardan la anchura y la altura de un rectángulo concreto.

```
class Rectangulo{
    int x;
    int y;
    int ancho;
    int alto;
    int calcularArea(){
        return (ancho*alto);
    }
}
```

A la función que desplaza el rectángulo horizontalmente en *dx* y verticalmente en *dy*, le pasamos dichos desplazamientos y a partir de estos datos, actualizará los valores que guardan sus miembros dato *x* e *y*. La función no devuelve nada es de tipo **void**.

```
class Rectangulo{
    int x;
    int y;
    int ancho;
    int alto;
    void desplazar(int dx, int dy){
        x+=dx;
        y+=dy;
    }
}
```

La función que determina si un punto está o no en el interior del rectángulo, devolverá **true** si el punto se encuentra en el interior del rectángulo y devolverá **false** si no se encuentra, es decir, será una función del tipo **boolean**. La función necesitará conocer las coordenadas de dicho punto. Para que un punto de coordenadas *x1* e *y1* esté dentro de un rectángulo cuyo origen es *x* e *y*, y cuyas dimensiones son *ancho* y *alto*, se deberá cumplir a la vez cuatro condiciones

x1>*x* y a la vez *x1*<*x*+*ancho*

También se debe cumplir

y1>*y* y a la vez *y1*<*y*+*alto*

Como se tienen que cumplir las cuatro condiciones a la vez, se unen mediante el operador lógico AND simbolizado por **&&**.

```
class Rectangulo{
    int x;
    int y;
    int ancho;
    int alto;
    boolean estaDentro(int x1, int y1){
        if ( (x1>x) && (x1<x+ancho) && (y1>y) && (y1<y+alto) ) {
            return true;
        }
        return false;
    }
}
```

```
    }  
}
```

En el lenguaje Java, si la primera condición es falsa no se evalúan las restantes expresiones ya que el resultado es **false**. Ahora bien, si la primera es verdadera **true**, se pasa a evaluar la segunda, si ésta es falsa el resultado es **false**, y así sucesivamente.

Los constructores

Un objeto de una clase se crea llamando a una función especial denominada constructor de la clase. El constructor se llama de forma automática cuando se crea un objeto, para situarlo en memoria e inicializar los miembros de datos declarados en la clase. El constructor tiene el mismo nombre que la clase. Lo específico del constructor es que no tiene tipo de retorno.

```
class Rectangulo{  
    int x;  
    int y;  
    int ancho;  
    int alto;  
    Rectangulo(int x1, int y1, int w, int h){  
        x=x1;  
        y=y1;  
        ancho=w;  
        alto=h;  
    }  
}
```

El constructor recibe cuatro números que guardan los parámetros *x1*, *y1*, *w* y *h*, y con ellos inicializa los miembros de datos *x*, *y*, *ancho* y *alto*.

Una clase puede tener más de un constructor. Por ejemplo, el siguiente constructor crea un rectángulo cuyo origen está en el punto (0, 0).

```
class Rectangulo{  
    int x;  
    int y;  
    int ancho;  
    int alto;  
    Rectangulo(int w, int h){  
        x=0;  
        y=0;  
        ancho=w;  
        alto=h;  
    }  
}
```

Este constructor crea un rectángulo de dimensiones nulas situado en el punto (0, 0),

```
class Rectangulo{  
    int x;  
    int y;  
    int ancho;  
    int alto;  
    Rectangulo(){  
        x=0;
```

```

        y=0;
        ancho=0;
        alto=0;
    }
}

```

Con estas porciones de código definimos la clase, y la guardamos en un archivo que tenga el mismo nombre que la clase *Rectangulo* y con extensión .java.

```

public class Rectangulo {
    int x;
    int y;
    int ancho;
    int alto;
    public Rectangulo() {
        x=0;
        y=0;
        ancho=0;
        alto=0;
    }
    public Rectangulo(int x1, int y1, int w, int h) {
        x=x1;
        y=y1;
        ancho=w;
        alto=h;
    }
    public Rectangulo(int w, int h) {
        x=0;
        y=0;
        ancho=w;
        alto=h;
    }
    int calcularArea(){
        return (ancho*alto);
    }
    void desplazar(int dx, int dy){
        x+=dx;
        y+=dy;
    }
    boolean estaDentro(int x1, int y1){
        if((x1>x) && (x1<x+ancho) && (y1>y) && (y1<y+ancho)) {
            return true;
        }
        return false;
    }
}

```

Los objetos

Para crear un objeto de una clase se usa la palabra reservada **new**.

Por ejemplo,

```
Rectangulo rect1=new Rectangulo(10, 20, 40, 80);
```

new reserva espacio en memoria para los miembros dato y devuelve una referencia que se guarda en la variable *rect1* del tipo *Rectangulo* que denominamos ahora objeto. Dicha sentencia, crea un objeto denominado *rect1* de la clase *Rectangulo* llamando al segundo constructor en el listado. El rectángulo estará situado en el punto de coordenadas $x=10$, $y=20$; tendrá una anchura de *ancho*=40 y una altura de *alto*=80.

```
Rectangulo rect2=new Rectangulo(40, 80);
```

Crea un objeto denominado *rect2* de la clase *Rectangulo* llamando al tercer constructor, dicho rectángulo estará situado en el punto de coordenadas $x=0$, $y=0$; y tendrá una anchura de *ancho*=40 y una altura de *alto*=80.

```
Rectangulo rect3=new Rectangulo();
```

Crea un objeto denominado *rect3* de la clase *Rectangulo* llamando al primer constructor, dicho rectángulo estará situado en el punto de coordenadas $x=0$, $y=0$; y tendrá una anchura de *ancho*=0 y una altura de *alto*=0.

Acceso a los miembros

Desde un objeto se puede acceder a los miembros mediante la siguiente sintaxis

objeto.miembro;

Por ejemplo, podemos acceder al miembro dato *ancho*, para cambiar la anchura de un objeto rectángulo.

```
rect1.ancho=100;
```

El rectángulo *rect1* que tenía inicialmente una anchura de 40, mediante esta sentencia se la cambiamos a 100.

Desde un objeto llamamos a las funciones miembro para realizar una determinada tarea. Por ejemplo, desde el rectángulo *rect1* llamamos a la función *calcularArea* para calcular el área de dicho rectángulo.

```
rect1.calcularArea();
```

La función miembro *area* devuelve un entero, que guardaremos en una variable entera *medidaArea*, para luego usar este dato.

```
int medidaArea=rect1.calcularArea();  
System.out.println("El área del rectángulo es "+medidaArea);
```

Para desplazar el rectángulo *rect2*, 10 unidades hacia la derecha y 20 hacia abajo, escribiremos

```
rect2.desplazar(10, 20);
```

Podemos verificar mediante el siguiente código si el punto (20, 30) está en el interior del rectángulo *rect1*.

```

        if(rect1.estaDentro(20,30)){
            System.out.println("El punto está dentro del
rectángulo");
        }else{
            System.out.println("El punto está fuera del
rectángulo");
        }

```

rect1.dentro() devuelve **true** si el punto (20, 30) que se le pasa a dicha función miembro está en el interior del rectángulo *rect1*, ejecutándose la primera sentencia, en caso contrario se ejecuta la segunda.

Como veremos más adelante no siempre es posible acceder a los miembros, si establecemos controles de acceso a los mismos.

```

public class RectanguloAppl {
    public static void main(String[] args) {
        Rectangulo rect1=new Rectangulo(10, 20, 40, 80);
        Rectangulo rect2=new Rectangulo(40, 80);
        Rectangulo rect3=new Rectangulo();
        int medidaArea=rect1.calcularArea();
        System.out.println("El área del rectángulo es "+medidaArea);

        rect2.desplazar(10, 20);

        if(rect1.estaDentro(20,30)){
            System.out.println("El punto está dentro del
rectángulo");
        }else{
            System.out.println("El punto está fuera del
rectángulo");
        }
    }
}

```

La vida de un objeto

En el lenguaje C++, los objetos que se crean con **new** se han de eliminar con **delete**. **new** reserva espacio en memoria para el objeto y **delete** libera dicha memoria. En el lenguaje Java no es necesario liberar la memoria reservada, el recolector de basura (garbage collector) se encarga de hacerlo por nosotros, liberando al programador de una de las tareas que más quebraderos de cabeza le producen, olvidarse de liberar la memoria reservada.