TEMA 4: P.O.O. Clases y Objetos

"El buen código
es su mejor documentación"

La programación orientada a objetos es un paradigma de programación diferente al método clásico de programación. Sus principales ventajas:

- Aumenta la modularidad de los programas: el código fuente de una clase puede mantenerse y reescribirse sin que haya que modificar otras clases.
- Aumenta la reutilización de los programas: es sencillo utilizar clases de otras personas sin tener que conocer con detalle la codificación. A este concepto se le denomina encapsulamiento.
- Facilidad de testeo: si una clase da problemas sólo testeamos ese código

Utiliza técnicas nuevas: herencia, polimorfismo,……

Los lenguajes de programación de última generación permiten tanto la POO como la programación clásica.

Clases y Objetos

Un programa java consta de una o más clases interdependientes.
Una **clase** permite describir las propiedades y habilidades de los objetos de la vida real. Es un conjunto de datos y operaciones que actúan sobre esos datos. Son los moldes de los cuales se generan los objetos.

Por ejemplo podríamos tener una clase Persona, por cada persona tenemos la siguiente información:
sexo, edad, casado...; además una persona puede realizar las siguientes acciones : cumplirAnyos, casarse...

Las propiedades y habilidades de una clase se llaman en Java **campos (o atributos)** y **métodos**, respectivamente, y se conocen conjuntamente como **miembros** de la clase.

Un **objeto** es una realización concreta de una clase. (La instancia de un tipo es una variable, la de una clase es un objeto). Por ejemplo dada la clase Persona, podemos

crear un objeto que se llame Antonio, este objeto tendrá todas las habilidades y propiedades de la clase persona.

Cuando se escribe un programa, se definen las clases (propiedades y métodos) y cuando se ejecute ese programa se crearán los objetos donde almacenaremos los datos (propiedades) y con ellos operaremos (métodos).

Ejemplo de clase:

```
class Persona{
	private char sexo;  // 'H' o 'M'
	private int edad;
	private boolean casado;
	private int numHijos;

	public void cumplirAnnos(){
		edad++;
		}
	public boolean casarse(){
		if (casado)
			return false;

		casado=true;
		return true;
		}

	public void tenerHijo(int n){
		numHijos+=n;
	}

}
```

Los **campos** de una clase pueden ser de cualquier tipo, incluso pueden ser a su vez objetos.
Un **método** es el equivalente a una función. Como tal tiene un retorno (void si no retorna nada) y un número de parámetros.

El nombre de la clase y el fichero que la contiene deberá ser el mismo. Guardaremos una clase en cada fichero.

Para declarar un objeto de la clase Persona, hacemos:

```
	Persona Antonio;
	Persona Maria;
```

Cada uno de los objetos que creamos tiene los datos especificados en la clase Persona.

Un objeto es único y diferente de otro objeto. La Persona Antonio es diferente de cualquier otra, aunque tengan los mismos datos guardados

Se puede mezclar el orden en que aparecen los campos y métodos de una clase.

Para hacer referencia a un campo o un método de un objeto, fuera de los métodos de la clase de dicho objeto, se hace anteponiendo el nombre del objeto luego un punto y a continuación el nombre del campo o del método. Sólo puedo acceder a los miembros públicos.

```
Antonio.tenerHijo()
Maria.casarse()
```

Constructores

Un método especial en una clase es el **constructor.** Un constructor crea un objeto y lo inicializa.
Tiene el nombre de la clase y no se le pone ningún tipo de retorno, ni void.
Para una misma clase puede haber varios constructores, que sólo se diferencian en el orden y/o tipo de los parámetros. Se dice que están sobrecargados.

Si una clase no tiene constructor el compilador crea uno por defecto que no tiene argumentos e inicializa los atributos por defecto: numéricos=0, boolean=false, char='\0'. Aún así conviene que exista un método constructor.

Ejemplo:

```
public Persona(char s ){
        sexo=s;
        casado=false;
        edad=0;
        numHijos=0;
}

public Persona(char s, int e, boolean c, int nh){
        sexo=s;
        casado=c;
        edad=e;
        numHijos=nh;
}
```

Un objeto se crea invocando a un constructor de su clase. En nuestro ejemplo tenemos dos constructores distintos, podemos hacer lo siguiente:

```
Persona juan = new Persona('H');
Persona maria = new Persona('M',30,true,1);
```

La palabra clave *new* se usa para crear una instancia de la clase. Antes de ser instanciada con *new* no consume memoria, simplemente es una declaración de tipo. El valor de un objeto antes de ser instanciado es null. Después de ser instanciado un nuevo objeto juan, el valor de sexo juan será igual a 'H'.

NOTA: Puedo inicializar los atributos en la definición, no da error de compilación, pero no se debe, si lo hago ahí y en el constructor, este último sobreescribe los valores anteriores.

Ejemplos:

1. Añadir a la clase persona un método que nos diga si una persona es mayor de edad.
2. Implementar una clase coche para que la guardamos la velocidad. Tendremos dos métodos uno para acelerar (aumentamos su velocidad) y otro para frenar (disminuimos la velocidad). Generar en el main dos coches, acelerar o frenar para probar la clase.

3. Implementar una clase satélite para la que se guardan: paralelo y meridiano en los que está el satélite y su distancia a la tierra. Implementar dos métodos, uno para cambiar de posición al satélite y otro para printar en pantalla su posición.

Métodos get y set

Se utilizan para devolver el valor de un atributo get, o para modificarlo set. No son obligatorios.

Ejemplo:

```
class Persona{


        public int getEdad(){
                return edad;
        }

        public  void setEdad(int e){
                edad=e;
        }

}
```

Acceso a los miembros de una clase

Dentro de una clase se puede acceder a todos los atributos y métodos de dicha clase. Fuera de la clase, depende del modificador del miembro, este puede ser:

public: Accesible fuera de la clase.
private: Sólo accesible dentro de la clase.

Los atributos suelen ser private y los métodos public.

Para hacer referencia a un miembro público de una clase, fuera de ésta, se antepone el nombre del objeto, un punto y luego el nombre del atributo o método.

Ejemplo

```
class Rectangulo{
        private float base;
        private float altura;

        public Rectangulo(float b,float a){
                base=b;
                altura=a;
                }
        public float area(){
                return base*altura;
                }
        public float perimetro(){
                return (2*base)+(2*altura);
                }
        public void escribe(){
                System.out.println("Base:" + base + " altura:" + altura + " area:" +
area() + "Perímetro:" + perímetro());
                }
}
```

Observar como en el método escribe hemos invocado al método area y perímetro.

Fuera de la clase los invocaríamos así:

```
Rectangulo r = new Rectángulo(2,4);
Float a = r.area();
Float p = r.perimetro();
```

Más sobre variables

Un método puede tener variables locales, y estas no estarán inicializadas por defecto. Una variable, local o miembro, puede tener el modificador final, esto significa que se trata de una constante.

Ejemplo:

```
class Circulo{
		private float radio;

		Circulo( float r){
			r=radio;
		}

		public float area(){
			float ar;
			final float PI=3.1416;

			ar=PI * radio * radio;
			return ar;
		}

		public float perimetro(){
			final float PI=3.1416;

			return 2*PI*radio;
		}

		public escribe(){
			System.out.println("Circulo de radio:" + radio + " perímetro:" +
perimetro() + " area:" + area() );
		}
}
```

Miembros static

Cuando se crea un objeto se crea una copia de todos los campos de la clase para ese objeto.
A veces nos puede interesar que un campo sea común a todos los objetos de una clase. Es decir que sólo se almacene ese valor en una zona de memoria y lo compartan todos los objetos de esa misma clase.
En nuestra clase Persona, nos puede interesar llevar una variable numTotalHijos donde se anote el número total de hijos de nuestras personas. Normalmente estas variables

se suelen declarar private para mantener el encapsulado de la clase, pero en ocasiones se declaran public.
Para declarar una variable de este tipo utilizamos el modificador static:

```
private static int numTotalHijos;
```

Estas variables se inicializan al empezar el programa con el valor por defecto. Podemos inicializarla en la declaración, por ej:

```
private static int numTotalHijos=0;
```

Podemos modificar su valor en los métodos de esa clase. En el constructor:

```
Persona(char s, int e, boolean c,int nh){
        ...
        numTotalHijos=numTotalHijos+ nh;
        }
```

El método tenerHijo queda de la siguiente forma:

```
void tenerHijo(){
        numHijos++;
        numTotalHijos++;
}
```

Dentro de su clase nos referimos a estas variables con el nombre simplemente.
Cuando se usa desde otra clase, podemos acceder a ella si es pública, anteponiendo el nombre de la clase:

```
        Persona.numTotalHijos;
```

Del mismo modo podemos declarar un método de clase, precedido del modificador static. Estos métodos no pueden acceder a los campos no estáticos de una clase, ya que no se les llama desde un objeto. Por tanto sólo pueden modificar los campos estáticos. Dicho de otra forma, un método es static si los datos que usa de su clase son estáticos.

Podemos declarar un método que si en total hay más de 20 hijos saque un mensaje en pantalla:

```
static void informe(){
        if (numTotalHijos > 20)
                System.out.println("\nSe ha sobrepasado el límite");
```

```
}
```

Podemos invocar a este método desde el método tenerHijo que quedaría como sigue:

```
void tenerHijo(){
        numHijos++;
        numTotalHijos++;
        informe();
}
```

Podemos declarar variables constantes estáticas por ejemplo:

Ej.

```
static final float pi=3.1416
```

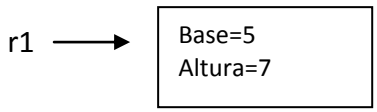
Las clases que sólo tienen miembros static no necesitan constructores(p.ej la clase dónde está el main, la clase Math o Character).

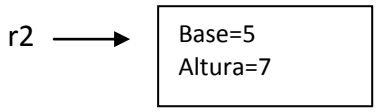
Constructor copia

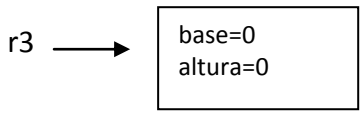
Construye un nuevo objeto como copia de uno que se pasa por parámetro.

```
class Rectangulo{
                private float base;
                private float altura;

                public Rectangulo(){
                        }

                public Rectangulo(float b,float a){
                        base=b;
                        altura=a;
                        }
/* Constructor copia */
                public Rectangulo( Rectangulo r){
                        this.base=r.base;
                        this.altura=r.altura;
                        }
}
```

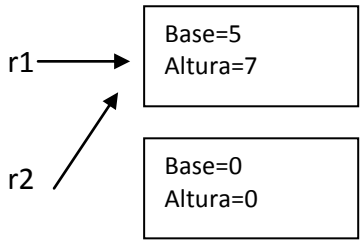
```
Rectangulo r1 = new Rectangulo(5,7);
Rectangulo r2= new Rectangulo(r1);
Rectangulo r3= new Rectangulo();
```








```
r2=r1;
```



Identificador this

Nos proporciona una referencia al objeto con el que estamos trabajando, es decir el que está ejecutando el método. La mayoría de las veces no es necesario ponerlo puesto que se sobreentiende que objeto está invocando al método. En algunas ocasiones nos sirve para resolver ambigüedades o para devolver referencias al propio objeto.

Usamos this cuando un método tiene un campo local o un parámetro con el mismo nombre que el de un miembro.

Ejemplo:

```
class Mio{
        int d;
        char a;
        Mio() {
                d=7;
                a='m';
                }
        void multiplica(int d){
                this.d = d*2; }
        }
```

this.d se refiere al dato de la clase, mientras que d es el parámetro.

Igualdad y Clonación

Como los objetos se guardan como referencias, si comparo dos objetos directamente, estoy preguntando por sus referencias.

```
Mio c1=new Mio();
Mio c2=new Mio();
if (c1==c2)
  …….
```

Así estoy comparando sus direcciones de memoria. Para comparar sus valores utilizamos el método equals, cada usuario debe definirse este método para cada clase nueva.

Ejemplo:

Yo me puedo definir de este modo la igualdad entre dos objetos de la clase Café:

```
boolean equals ( Mio c){
        return c.a==a;}
```

así preguntaría entonces si dos objetos de la clase Mio son iguales:

```
        c1.equals(c2)
```

Lo mismo sucede con la asignación, si yo hago c1=c2, estoy copiando las referencias, es decir, ahora c1 y c2 apuntan al mismo objeto. Para copiar los valores hay que clonar el objeto usando un método nuevo clone que de nuevo debe definirse el usuario:

```
class Mio implements Cloneable {
        public Object clone(){
                Mio x=new Mio();
                x.d=d;
                x.a=a;
                return x;
                }
        }
```

Esto es una alternativa al constructor copia.
Para hacer una copia de un objeto hago lo siguiente:

```
Mio primero=new Mio();
Mio segundo=(Mio)primero.clone();
```

Comparación entre variables de tipos básicos y de objetos:

	creación	guarda	copia	comparación
típo básico	tipo	valor	=	==
objeto		referencia	clone	equals

Diferencia entre objetos y tipos básicos

Una variable puede contener un tipo básico o un objeto. Si contiene:

- Un tipo básico: memoria para un valor. Cada tipo tiene un conjunto predefinido de operaciones.
- Un objeto: en memoria se guarda un conjunto de valores. Sobre ese conjunto de valores opera un conjunto de métodos. Se almacena como una referencia (una dirección de memoria). Las tablas, al ser objetos se almacenan mediante una referencia.

Ejemplo

```
class Cafe{
        String nombre;
        double precio;
        Lote L[]=new Lote[4];
        double existencias;
        }

class Lote{
        double  cantidad;
        Fecha fechalimite;
        }
```

Objetos nulos

Los objetos que sólo son declarados y no construidos toman el valor null.
Podemos comprobar si un objeto es nulo así:

```
if (s==null)
```

Si tratamos de utilizar un objeto nulo se lanza la excepción nullPointerException.

Paso de un objeto como parámetro a un método

Si paso un objeto a un método y modifico dentro sus campos, el objeto queda modificado:

Ejemplo

Si me defino el siguiente método:

```
static void modificar( Mio par){
        par.d=5;
        par.a='z';
}
```

El siguiente código

```
Mio x = new Mio();
modificar(x);
```

modifica los datos de x.

Objeto como retorno de un método

Para devolver un solo valor desde un método usamos return.
Para devolver varios valores, se hace construyendo un objeto dentro del método y retornando a continuación dicho objeto:

```
static Mio m2(){

        Mio local=new Mio();
        local.d=3;
        local.a='b';
        return local;
        }
```

La siguiente llamada crea un objeto de la clase Mio y lo guarda en x:

```
Obj x= m2();
```