TEMA 3: Clases y Objetos

3.1 Clase.

Una clase es un tipo definido por el usuario que describe los atributos y los métodos de los objetos que se crearán a partir de ella. El estado del objeto viene determinado por los atributos y los métodos son las operaciones que definen su comportamiento. Dentro de las clases también se encuentran los constructores que permiten inicializar un objeto.
Los atributos y los métodos se denominan en general miembros de la clase.
La definición de una clase consta de dos partes: el nombre de la clase precedido por la palabra reservada **class** y el cuerpo de la clase entre llaves. La sintaxis queda:

```
class nombre-clase{
    cuerpo de la clase
}
```

Dentro del cuerpo de la clase se puede encontrar atributos y métodos.

Ej:
```
class Circunferencia{
    private double x,y, radio;
     public Circunferencia(){ }
     public Circunferencia(double cx, double cy, double r){
          x=cx; y=cy; radio=r;
     }
     public void ponRadio(double r){
         radio=r;
     }
    public double longitud(){
       return 2*Math.PI*radio;
    }
}
```

En el ejemplo se define la clase Circunferencia, que puede ser usada dentro de un programa de la misma manera que cualquier otro tipo. Un objeto de esta clase tiene tres atributos (coordenadas del centro y valor del radio), dos constructores y un método.

Los constructores se distinguen fácilmente porque tienen el mismo nombre que la clase.

Los atributos se declaran de la misma manera que cualquier variable. En una clase, cada atributo debe tener un nombre único.

Siguiendo las recomendaciones de la programación orientada a objetos, cada clase se debe implementar en un fichero *.java*, de esta manera es más sencillo modificar la clase.

3.2 Métodos

Los métodos forman lo que se denomina interfaz de los objetos, definen las operaciones que se pueden realizar con los atributos. Desde el punto de vista de la Programación Orientada a Objetos, el conjunto de métodos se corresponde con el conjunto de mensajes que los objetos de una clase pueden responder.
Los métodos permiten al programador modularizar sus programas.

Todas las variables declaradas en las definiciones de métodos son variables locales, sólo se conocen en el método que las define. Casi todos los métodos tienen una lista de parámetros que permiten comunicar información.

La sintaxis para definir un método es:

```
<modificador-acceso> tipoR Nombre-método(<parámetros>){
        <cuerpodelmétodo>
 }
```

donde <modificador-acceso> indica como es el acceso a dicho método: public, private, protected y sin modificador.
<tipoR> es el tipo de dato que retorna el método. Es obligatorio indicar un tipo. Para los métodos que no devuelven nada se utiliza la palabra reservada void.
Nombre-método es como el programador quiere llamar a su método y <parámetros> son los datos que se van a enviar al método para trabajar con ellos, no son obligatorios.
Para retornar el valor se utiliza el operador **return**.

Ej:

```
int Suma(int x, int y){
  return x+y;
}
```

En este ejemplo el método retorna un valor de tipo int, recibe 2 parámetros también de tipo int y realiza su suma.

```
void Imprimir(){
  System.out.println("Este método no devuelve nada y tampoco
                      recibe parámetros");
}
```

En este ejemplo el método no retorna ningún valor, ni recibe paráme-tros, aun así es necesario poner los paréntesis vacíos.

3.2.1 Métodos Estáticos o de Clase

Se cargan en memoria en tiempo de compilación y no a medida que se ejecutan las líneas de código del programa. Van precedidos del modificador static.
Para invocar a un método estático no se necesita crear un objeto de la clase en la que se define. Si se invoca desde la clase en la que se encuentra definido, basta con escribir su nombre.
Si se le invoca desde una clase distinta, debe anteponerse a su nombre, el de la clase en que se encuentra seguido del operador punto (.) La sintaxis es:

```
< NombreClase> .metodoEstatico( );
```

Suelen emplearse para realizar operaciones comunes a todos los objetos de la clase. No afectan a los estados de los mismos.

Por ejemplo, si se necesita un método para contabilizar el número de objetos creados de una clase, se define estático ya que su función, aumentar el valor de una variable entera, se realiza independientemente del objeto empleado para invocarle.

No es conveniente usar muchos métodos estáticos, pues si bien se aumenta la rapidez de ejecución, se pierde flexibilidad, no se hace un uso efectivo de la memoria y no se trabaja según los principios de la POO.

A veces es necesario crear un método que se utiliza fuera del contexto de cualquier instancia, para ello hay que declarar estos métodos como static. Los métodos estáticos sólo pueden llamar a otros métodos static directamente, y no se pueden referir a this o super de ninguna manera.

Las variables también se pueden declarar como static, y es equivalente a declararlas como variables globales, que son accesibles desde cualquier fragmento de código.

Se puede declarar un bloque static que se ejecuta una sola vez si se necesitan realizar cálculos para inializar las variables static.

Ej:

```
class Estatica{
   static int a=3,b;
   static{
      System.out.println("Bloque static inicializado");
       b=a*4;
    }
    static void metodo2(int x){
         System.out.println("x= "+x);
         System.out.println("a= "+a);
         System.out.println("b= "+b);
    }
   public static void main(String[] args){
      metodo2(42);
   }
}
```

En el ejemplo la clase que tiene dos variables static, un bloque de inicialización static y un método static. La salida del programa es:

```
Bloque static inicializado
x = 42
a = 3
b = 12
```

3.3 Objetos

Un objeto consta de una estructura interna (los atributos) y de una interfaz que permite acceder y manipular dicha estructura (los métodos). Para construir un objeto de una clase cualquiera hay que llamar a un método de iniciación, el constructor. Para ello se utiliza el operador new. La sintaxis es la siguiente:

```
Nombre-clase nombre-objeto=new  Nombre-clase(<valores>);
```

donde Nombre-clase es el nombre de la clase de la cual se quiere crear el objeto, nombre-objeto es el nombre que el programador quiere dar a ese objeto y <valores> son los valores con los que se inicializa el objeto. Dichos valores son opcionales.

Ej: Circunferencia circ = new Circunferencia();

Se crea el objeto circ, de la clase Circunferencia, con los valores predeterminados.

Cuando se crea un objeto, Java hace lo siguiente:

- Asignar memoria al objeto por medio del operador new.
- Llamar al constructor de la clase para inicializar los atributos de ese objeto con los valores iniciales o con los valores predeterminados por el sistema: los atributos numéricos a cero, los alfanuméricos a nulos y las referencias a objetos a null.

Si no hay suficiente memoria para ubicar el objeto, el operador new lanza una excepción OutOfMemoryError.

Para acceder desde un método de una clase a un miembro de un objeto de otra clase diferente se utiliza la sintaxis: objeto.miembro. Cuando el miembro accedido es un método se entiende que el objeto ha recibido un mensaje, el especificado por el nombre del método y responde ejecutando ese método.

3.3.1 Asignación

Una vez el objeto está creado, se tiene una referencia a ese objeto. Si se realiza la asignación de un objeto a otro los dos harán referencia al mismo objeto.

```
Ej:   Circunferencia c1 = new Circunferencia(0,0,15);
      Circunferencia c2 = c1;
      c1.ponRadio(25);
      System.out.println(c2.radio);
```

En el ejemplo la variable c1 apunta a un objeto de la clase Circunferencia. Debido a la asignación, la variable c2 apunta al mismo objeto. Después se modifica el valor del radio del objeto apuntado por c1. Y por último, se visualiza c2 que será el valor 25.

3.3.2 Igualdad

Creamos dos objetos iguales (con los mismos valores de sus variables miembro), al preguntar si las variables que apuntan a esos objetos son iguales nos devolverá false, pues aunque tengan el mismo valor no son el mismo objeto.

```
Ej:   Circunferencia c1 = new Circunferencia(0,0,15);
      Circunferencia c2 = new Circunferencia(0,0,15);
      if (c1==c2) /* Esto es falso*/
```

3.4 Constructor.

Un Constructor es un método especial en Java empleado para inicializar valores en instancias de objetos. A través de este tipo de métodos es posible generar diversos tipos de instancias para la clase en cuestión.

Los métodos constructores tienen las siguientes características:

- Se llaman igual que la clase.
- No devuelve nada, ni siquiera void.
- Puede haber varios constructores, que deberán distinguirse por el tipo de valores que reciba.
- De entre los que existan, sólo uno se ejecutará al crear el objeto.
- El código de un constructor, generalmente, suele ser inicializaciones de variables y objetos, para conseguir que el objeto sea creado con dichos valores iniciales.
- Si no se define ningún constructor el compilador crea uno por defecto sin parámetros que, al ejecutarse, inicializa el valor de cada atributo de la nueva instancia a 0, false o null, dependiendo de si el atributo es numérico, alfanumérico o una referencia a otro objeto respectivamente, pero dicho constructor desaparece en el mismo momento en que se defina otro constructor, por lo que si se quiere tener el de por defecto habrá que definirlo.

La declaración de constructores sigue la siguiente sintaxis:

```
<modificadordeVisibilidad> NombredelaClase ( <argumentos> ) {
        < declaraciones>
 }
```

donde <modificadorvisibilidad> es el tipo de modificador de acceso del constructor, <Nombredelaclase> es el nombre del constructor y debe coincidir con el de la clase y <argumentos> son las variables que recibe el constructor y contienen los valores con los que se inicializaran los atributos.

Ej:
```
class Circunferencia{
   private double x,y, radio;

   public Circunferencia(){ }

   public Circunferencia(double cx, double cy, double r){
        x=cx; y=cy; radio=r;
   }
   ……..
}
```

En este ejemplo existen 2 constructores, el primero que es el de por defecto y el segundo que inicializa los atributos x, y y radio con los valores cx, cy y r, respectivamente.

3.5 Referencia this

Java incluye un valor de referencia especial llamado this, que se utiliza dentro de cualquier método para referirse al objeto actual. El valor this se refiere al objeto sobre el que ha sido llamado el método actual.
Se puede utilizar this siempre que se requiera una referencia a un objeto del tipo de una clase actual. Si hay dos objetos que utilicen el mismo código, seleccionados a través de otras instancias, cada uno tiene su propio valor único de this.
Normalmente, dentro del cuerpo de un método de un objeto se puede referir directamente a las variables miembros del objeto. Sin embargo, algunas veces no se querrá tener ambigüedad sobre el nombre de la variable miembro y uno de los argumentos del método que tengan el mismo nombre.

Ej:
```
class Circunferencia{
    private double x,y, radio;

    public Circunferencia(double x, double y, double radio){
       this.x=x; this.y=y;  this.radio=radio;
    }
```

En el ejemplo el constructor de la clase inicializa las variables con los argumentos pasados al constructor. se. Se debe utilizar this en este constructor para evitar la ambigüedad entre el los argumentos y las variables miembro.

También se puede utilizar this para llamar a uno de los métodos del objeto actual. Esto sólo es necesario si existe alguna ambigüedad con el nombre del método y se utiliza para intentar hacer el código más claro.

3.6 Arrays.

3.6.1 Conceptos básicos

Un arreglo unidimensional es un grupo de valores (llamados elementos o componentes) que son del mismo tipo. Los arreglos son objetos, por lo que se consideran como tipos de referencia. Los elementos de un arreglo pueden ser tipos primitivos o de referencias (incluyendo arreglos). Para hacer referencia a un elemento específico en un arreglo se debe especificar el nombre del arreglo y el número de la posición del elemento en el arreglo.
El número de posición del elemento se conoce formalmente como el índice o subíndice del elemento en el arreglo. Todos los elementos de un arreglo deben ser del mismo tipo.

3.6.2 Declaración y creación de arreglos Unidimensionales

Para crear un arreglo, hay que crear una variable de arreglo del tipo deseado. La forma general de un arreglo unidimensional es:

Tipo nombre-de-variable[ ] ;

Los objetos arreglo ocupan espacio en memoria. Todos los objetos deben de crearse con la palabra clave new. El programador especifica el tipo de cada elemento y el número de elementos que se requieren para el arreglo. La siguiente declaración crea 12 elementos para el arreglo de enteros de nombre c:

```
int c[ ] ; / / declara la variable arreglo
c = new int [12] ; / / crea el arreglo
```

Esta tarea también puede realizarse en un paso:

```
int c[ ] = new int[12] ;
```

Al crear un arreglo cada uno de sus elementos recibe un valor predeterminado de cero para los elementos numéricos de tipos primitivos, falso para los elementos boléanos y nulo para las referencias (cualquier tipo no primitivo).

Al declararse un arreglo, su tipo y los corchetes pueden combinarse al principio de las declaraciones para indicar que todos los identificadores en la declaración son referencias a arreglos.

```
Ej:   double[ ] arreglo1, arreglo2;
      arreglo1 = new double[10] , arreglo2 = new double[20] ;
```

Un programa puede declarar arreglos de cualquier tipo. Todo elemento de un arreglo de tipo primitivo es una variable del tipo declarado del arreglo.
En un arreglo que sea de tipo de referencia, cada elemento del arreglo es una referencia a un objeto del tipo declarado de ese arreglo.
Cada uno de los elementos de un arreglo String es una referencia a un objeto String.

Un programa puede hacer referencia a cualquiera de estos elementos mediante una expresión de acceso a un arreglo que incluye el nombre del arreglo, seguido por el índice del elemento especifico encerrado entre ([]) corchetes.
El primer elemento en cualquier arreglo tiene el índice cero (lo que se denomina como elemento cero).

Por lo tanto, el primer elemento del arreglo c es c[0], el segundo elemento es c[1], el séptimo elemento del arreglo es c[6] y, en general, el i-esimo elemento del arreglo c es[c-i]. Los nombres de los arreglos siguen las mismas convenciones, que los demás nombres de las variables.

Un índice debe ser un entero positivo o una expresión entera que pueda promoverse a un int. Si un programa utiliza una expresión como índice, el programa evalúa la expresión para determinar el índice.

Ej.: Supóngase que la variable a es 5 y que b es 6, entonces la instrucción: C [a + b] += 2; suma 2 al elemento c[11] del arreglo.

El nombre del arreglo con subíndice es una expresión de acceso al arreglo. Dichas expresiones pueden utilizarse en el lado izquierdo de una asignación, para colocar un nuevo valor en un elemento del arreglo.

Operaciones

Para realizar cualquier operación con un arreglo, normalmente, se usa un for que comenzara en 0 y terminara en n-1, donde n es el número de elementos del array.

Ej: int c[ ] = new int[12] ;

```
for (int i=0; i<12;i++)
    c[i]= (int) (Math.random()*10)+1;    // Crea de manera aleatoria un array de 12 componentes

for (int i=0; i<12;i++)
   System.out.print(c[i]+ " " );         // Escribe en pantalla el contenido de las componentes
```

3.6.3 Arreglos Bidimensionales

Se define como un conjunto de datos del mismo tipo organizados en filas y en columnas, es decir, en una matriz o tabla. Los arreglos con dos dimensiones se utilizan a menudo para representar tablas de valores, que constan de información ordenada en filas y columnas.
El primer índice indica la fila y el segundo la columna. Para identificar un elemento de una tabla, se deben especificar los dos índices.

```
Ej:  int[ ] [ ] M = {{1,2,3,4},{5,6,7,8}{9,10,11,12}};
     for( int i=0;i<3;i++){
        for(int j=0;j<4;j++)
           System.out.print(M[i][j]+ " " );
        System.out.println();
     }
```

1	2	3	4
5	6	7	8
9	10	11	12