INDICE

1. PARADIGMA DE LA POO

La **programación orientada a objetos** intenta llevar al máximo nivel los conceptos de abstracción, encapsulamiento y ocultación de la información.

La programación estructurada aparece como una forma de crear un código que dispone de una estructura y es más fácil de leer y modificar. Pero cuando el código necesario para resolver un problema se hace muy grande ocupa tantas líneas que se hace inmanejable.

La programación modular descompone un problema en diferentes módulos y cada uno se ocupa de resolver una tarea determinada. En este paradigma, cuanto más independiente sea un módulo más fácil es realizar el mantenimiento de la aplicación y del trabajo en equipo. El problema es que los módulos no ocultan del todo la información, necesitando datos globales conocidos por todos y restando independencia a los módulos.

La programación orientada a objetos divide el problema en objetos, de forma que cada objeto actúa de forma independiente al resto. ***Cada objeto integra sus propios datos y los métodos que manipulan dichos datos.*** Sólo se puede acceder a los datos que definen un objeto a través de los métodos que se han definidos para ello.

Un objeto es un conjunto de variables (datos) y métodos (funciones) relacionados entre sí que permiten modelar entidades del mundo real.

Por ejemplo, en un programa que cree y manipule una agenda de amigos, los amigos son los objetos; estos objetos poseen unas características, que definen al amigo, y que se ocultan al mundo; cada amigo dispone de una serie de operaciones (métodos) que acceden a los datos y muestran la información deseada.

Las características de los objetos amigos son los atributos que nos interesa conocer de cada uno de ellos. Por ejemplo: nombre, apellidos, dirección, teléfono y descripción.

La operaciones que se pueden realizar sobre cada objeto amigo son: crear amigo, consultar amigo, eliminar amigo, modificar teléfono o modificar dirección.

Una vez definido y creado el objeto amigo no es posible acceder a las características de un amigo si no es a través del método *consultar amigo*.

Las características que definen la POO son:

- **Abstracción.** Proceso por el que se extraen las características esenciales para el problema que nos ocupa, ignorando las que no son relevantes. Otras características que puede tener un amigo son color de pelo, trabajo, estudios, etc., pero estas características no son relevantes para la agenda.
- **Modularidad.** Proceso de descomposición de un sistema en un conjunto de módulos o piezas independientes y con significado propio.
- **Encapsulamiento.** Proceso por el que se ocultan las características esenciales de una abstracción.
- **Herencia.** Propiedad por la que una clase de objetos nueva se crea a partir de otra clase ya existente, pudiendo utilizar las características existentes y añadir otras características nuevas. Por ejemplo, habiendo definido la clase trabajador se pueden definir otras clases que la amplían como Directivo, Contable, Administrador, etc.
- **Polimorfismo.** Propiedad por la que es posible que varios métodos de una misma clase tengan el mismo nombre pero que se compartan de forma diferente en función de los parámetros utilizados durante su invocación.

2. CLASES Y OBJETOS

Una clase es un tipo de datos genérico definido por el usuario. Podemos decir que es un patrón que define el estado (atributos o características) y comportamiento (métodos o funciones) de un tipo particular de objetos.

En la definición de una clase se indica cuáles son los atributos, determinando su tipo y nombre, y cuáles son los métodos que modifican el estado de dichos atributos.

Un objeto de una determinada clase se crea en el momento en el que se declara una variable de dicha clase (instancia de una clase).

Los atributos son una característica fundamental de cada objeto en una clase de objetos; tienen un valor para cada objeto.

Un objeto tiene:

- Una entidad que se puede referenciar con un nombre.
- Un estado, que es el conjunto de propiedades o atributos del objeto.
- Un comportamiento, que es el conjunto de operaciones (métodos) que se pueden realizar sobre los atributos.

Objeto = estado + comportamiento

Ejemplo. Una ventana de Windows es un objeto. El color de fondo, la altura y la anchura es su estado y las rutinas que permiten maximizar, cerrar minimizar la ventana son los métodos que definen el comportamiento.

Definición de una clase

La implementación de una clase se almacena en un fichero en formato texto con la extensión **.java** y con un nombre idéntico a la clase implementada. El nombre de una clase y el nombre del fichero que la contiene empezará siempre en mayúsculas.

Clase coche en fichero coche.java (Incorrecto)
Clase coche en fichero Coche.java (Incorrecto)
Clase Coche en fichero coche.java (Incorrecto)
Clase Cocheen fichero Coche.java (Correcto)

El nombre de una clase se realiza se realiza mediante la palabra reservada **class** seguida del nombre de la clase.

La implementación de la clase, atributos y métodos deben ir encerrados entre llaves ({ y }).

*La palabra reservada **class** puede ir precedida de un modificador de acceso que altera el comportamiento de la clase en algunos aspectos.*

Los modificadores de clase son:

- **public.** La clase es accesible desde cualquier lugar del programa. Si no aparece este modificador antes de la palabra reservada class, la clase solo puede ser referenciada por otras clases definidas dentro del mismo paquete, pero no puede ser referenciada por ninguna clase que no esté en el mismo paquete.
- **abstract.** Define una clase de la que no se pueden crear objetos. Se utiliza para que otras clases hereden de ella.
- **final.** Indica que no pueden crearse subclases de esa clase.

Estructura general de una clase

La implementación de una clase incluye sus atributos y sus métodos:

```
[modificador] class NombreClase
{
    //declaración de atributos
        …
    //declaración de métodos
        …
}
```

Ejemplo:

```
public class Persona
{
    //declaración de atributos
        private String nombre;
        private String apellidos;
        private int edad;

    //declaración de métodos
        public Persona() // constructor
        {
        //cuerpo del método constructor
        }
        public String getNombre()
        {
        //cuerpo del método
        }
    …
}
```

Los atributos:

- Se definen dentro de la clase y fuera de cualquier método.
- El tipo de los atributos pueden ser un tipo primitivo (int, float, char, etc.) o un tipo complejo (array, matriz, objeto,..).
- El nombre de los atributos comienzan con minúsculas
- Se inicializan implícitamente, si no se ha hecho explícitamente, para cada objeto según su tipo. Son globales a todos los métodos de la clase.

Los atributos pueden ir precedidos en su declaración de uno o varios modificadores:

Modificadores de acceso: (se indican en primer lugar)

- **public.** El atributo se puede referenciar desde cualquier otra clase. (no es conveniente)
- **private.** El atributo solo puede ser referenciado por los métodos de la clase.
- **protected.** (Es el modificador por defecto) El atributo puede ser accedido desde la clase, desde sus subclases, y desde otras clases que se encuentren en el mismo paquete.

Otros modificadores: (se indican después de los modificadores de acceso)

- **final:** el atributo se comporta como una constante.
- **static:** su valor es compartido por todos los objetos de la clase.

Estos dos modificadores se pueden combinar entre sí y con los modificadores de acceso.

Ejemplo:

```
public class Coche
{
    //Declaración de atributos
        private int codigo;
        public String color;
        protected Motor motor;
        private static final int numeroRuedas=4;
        private static final int numeroRetovisores=3;
        …….
    //Declaración de métodos
    ……
}
```

Los métodos

Por convenio, los nombres de los métodos comienzan con minúsculas. Si el nombre está formado por varias palabras, la primera comienza por minúsculas y el resto por mayúsculas.

Desde el cuerpo de los métodos de una clase se puede acceder directamente a los atributos.

Los métodos pueden ir precedidos en su declaración de uno o varios modificadores de acceso:

Modificadores de acceso:

- **public:** el método se puede referenciar directamente desde cualquier otra clase. (no recomendado)
- **private:** el método no puede ser invocado desde ninguna otra clase. Se emplea para realizar tareas auxiliares que no es necesario ver desde fuera.
- **protected (por defecto):** se puede acceder a ellos desde la clase, las subclases y por todas la clases que se encuentren en un mismo paquete.

Otros modificadores:

- **static.** Se pueden ejecutar sin crear un objeto de la clase. Se invocan anteponiendo el nombre de la clase no el del objeto. No dependen de ningún objeto en particular. Se denominan métodos de clase.
- **final.** No puede ser sobrescrito por las subclases de la clase a la que pertenece (los veremos en el tema de herencia). Solo los métodos static pueden ser final.
- **abstract.** Define un método abstracto (los veremos en el tema de herencia).

En Java los **argumentos de tipos primitivos** (int, float, etc.) se pasan siempre **por valor**. Las **referencias a objetos** se pasan también por valor. Es decir, que **como son referencias**, los objetos referenciados **se pueden modificar**.

3. CONSTRUCTORES

Existe un tipo de métodos especiales que se denominan constructores que:

- Permiten crear objetos de la clase a la que pertenecen.
- En su implementación se suele inicializar los atributos del objeto que se crea.
- Su declaración es similar a la de los métodos normales pero destacan dos diferencias:
  - ✓ No tienen tipo de retorno.
  - ✓ Se llaman igual que la clase.
- Si en una clase no se definieran constructores el compilador añade uno (sin parámetros) por defecto.
- Dentro de una clase, los constructores sólo pueden ser llamados por otros constructores o por métodos **static** (métodos de clase). No pueden ser llamados por los métodos de objeto de la clase.

Java permite definir métodos distintos con el mismo nombre (se diferencian en el número y/o tipo de los argumentos). Se les denomina **métodos sobrecargados**. Es normal sobrecargar el constructor de una clase para que un objeto pueda ser creado de varias formas distintas. Para sobrecargar un constructor:

¿Qué significa que una clase tenga más de un constructor? Que podemos crear un objeto de esa clase de varias formas distintas. Ejemplo:

```
public class Amigo {
    private String nombre;
    private String direccion;
    private string telefono;

    public Amigo(String nombre, String direccion, String telefono) {
       this.nombre = nombre;
       this.direccion = direccion;
       this.telefono = telefono;
     }

    public Amigo(String nombre, String telefono) {
       this.nombre = nombre;
       this.direccion = null;
       this.telefono = telefono;
     }
}
```

Al crear un amigo podemos crearlo con todos sus atributos a **null** si no le proporcionamos valores en la llamada, podemos crearlo proporcionándole todos los parámetros o proporcionándole simplemente el nombre y el teléfono.

Al acceder a variables de instancia de una clase, la palabra clave **this** hace referencia a los miembros de la propia clase en el objeto actual; es decir, **this** se refiere al objeto actual sobre el que está actuando un método determinado y se utiliza siempre que se quiera hace referencia al objeto actual de la clase. En el ejemplo anterior, los atributos se denominan igual que los parámetros, por lo que puede haber ambigüedad sobre quien es quien. Esta ambigüedad se soluciona con **this**, el nombre asociado a **this** se refiere al atributo de la clase y el que no tiene asociado el **this** se refiere al parámetro.

4. OBJETOS

Los objetos en Java son instancias de las clases. Se pueden crear varios objetos de la misma clase. Cuando se crea un objeto se reserva espacio en memoria. El tipo del objeto es el de la clase de la que se ha instanciado.

Para declarar un objeto se indica:

NombreClase IdentificadorObjeto;

Ejemplo:

Persona p1; // en este punto p1 vale null

Se declara un objeto p1 que pertenece a la clase Persona. En este momento se ha declarado el objeto pero no se ha creado. Al no estar creado solo se ha reservado espacio en memoria para su identificador, que tiene el valor **null**.

Para crear un objeto se escribe la sentencia:

identificador = new Clase(…);

En el momento en que se crea un objeto se reserva espacio en memoria para sus atributos y sus métodos.

Persona p1= new Persona();

Ahora ya existe en memoria el objeto **p1** y podemos acceder a él a través de sus métodos.

Se puede declarar y crear un objeto resumidamente en una sola línea:

Clase identificador = new Clase(…);

Persona p1= new Persona();

EJEMPLO 1

Queremos modelar una casa con muchas bombillas, de forma que cada bombilla se puede encender o apagar individualmente. Para ello:

a. Haremos una clase Bombilla con una variable privada que indique si está encendida o apagada, así como un método que nos diga si la bombilla está encendida.

b. Además queremos poner un interruptor general de la luz, tal que si saltan los fusibles, todas las bombillas quedan apagadas. Cuando el fusible se repara, las bombillas vuelven a estar encendidas o apagadas, según estuvieran antes del percance. Para modelar en Java esta característica usaremos una variable de clase (static) que nos diga si hay luz en el edificio o no. Cada objeto Bombilla se enciende y se apaga individualmente; pero sólo responde que está encendida si su interruptor particular está activado y además hay luz general.

La clase tendrá dos atributos:

- Un interruptor general que indica si los fusibles están activados o desactivados. Este atributo es compartido por todos los objetos de la clase.
- Un interruptor particular que indica si la bombilla esté encendida o apagada.

Los métodos de la clase serán:

- Un constructor sin parámetros
- Un método que active los fusibles
- Un método que desactive los fusibles
- Un método que encienda la bombilla
- Un método que devuelva verdadero si la bombilla está encendida o falso en caso contrario.

Diseña la clase bombilla y realiza un programa de prueba. El programa creará tres bombillas: una para el dormitorio, otra para el salón y otra para la cocina.

- El programa mostrará un menú con las siguientes opciones:
- Encender habitación
- Encender Salón
- Encender cocina
- Apagar habitación
- Apagar Salón
- Apagar cocina
- Activar interruptor general
- Desactivar interruptor general
- Consultar estado de las bombillas
- Salir

Pedirá una de las opciones y ejecutará la opción indicada. Realiza las siguientes pruebas:

- Enciende la cocina y el salón.
- Comprueba el estado de las bombillas
- Activa el interruptor general
- Comprueba el estado de las bombillas

Vamos a tener dos tipos de interruptores, uno cuyo estado afecte a todos los objetos de la clase y otro cuyo estado sólo afecte al objeto al que pertenece, a la propia bombilla.

Cuando necesitemos un atributo cuyo valor sea compartido por todos los objetos creados de dicha clase se utiliza una variable con los modificadores **private static**.

Por lo tanto esta clase necesita dos atributos:

- Uno con acceso **private static boolean** que simule el interruptor general. Si un objeto pone este atributo a false, su valor será false para todos los objetos creados de la clase.
- Otro con acceso **private boolean** que simule el interruptor particular de la bombilla. Cuando un objeto modifique el valor de esta atributo no afectará al resto de los objetos.

Definición de la clase Bombilla

```
public class Bombilla
{
   //Atributos
   /**
    * Atributo de clase que simula el comportamiento del fusible. Apagado por defecto
    */
   private static boolean intgen = false; // interruptor general. Apagado por defecto

    /**
    * Atributo de objeto que simula el comportamiento del interruptor de la bombilla
    */
   private boolean intpar; // interruptor particular

   /**
    * Constructor por defecto de la clase Bombilla
    */
   public Bombilla()
   {
       intpar = false; // creamos la bombilla apagada
    }
```

```
 // Métodos
   /**
    * Enciende los "fusibles" este método debe ser static porque modifica un atributo static
    */
   public static void activaGeneral ()
   {
      intgen = true;
   }

   /**
    * Apaga los "fusibles"
    */
   public static void desactivaGeneral ()
   {
      intgen = false;
   }


 /**
    * Enciende el interruptor particular de la bombilla
    */
   public void enciende ()
   {
      this.intpar = true;
   }

   /**
    * Apaga el interruptor particular de la bombilla
    */
   public void apaga ()
   {
      this.intpar = false;
   }

   /**
    * Devuelve true si la bombilla está encendida, false en otro caso
    */
   public boolean encendida ()
   {
      return this.intpar && intgen;
   }
}
```

Clase de prueba de la clase Bombilla

```
import java.util.Scanner;
public class PruebaBombilla2
{
//Metodo que muestra el menú y devuelve la opción elegida
   public static int menu()
   {
      Scanner leer=new Scanner(System.in);
      int op;
      System.out.println();
         System.out.println("Selecciona una opcion");
         System.out.println("----------------------");
         System.out.println(" 1- Encender bombilla de la habitación");
         System.out.println(" 2- Encender bombilla del salon");
         System.out.println(" 3- Encender bombilla de la cocina");
         System.out.println(" 4- Apagar bombilla de la habitación");
         System.out.println(" 5- Apagar bombilla del salon");
```

```
        System.out.println(" 6- Apagar bombilla de la cocina");
        System.out.println(" 7- Activar interruptor general");
        System.out.println(" 8- Desactivar interruptor general");
        System.out.println(" 9- Consultar estado de las bombillas");
        System.out.println(" 0- Salir");
        System.out.print("Introduce una opcion: ");
        op=leer.nextInt();
        System.out.println();

        return op;

}
```

//Método que ejecuta la opción elegida

```
public static void ejecutarOpcion(int op, Bombilla salon, Bombilla habitacion, Bombilla cocina)
{
  switch (op)
      {
         case 1:
         {
            System.out.println("*** Se enciende el interruptor de la bombilla de la habitacion ***");
            habitacion.enciende();
            break;
         }
         case 2:
         {
            System.out.println("*** Se enciende el interruptor de la  bombilla de salon ***");
            salon.enciende();
            break;
         }
         case 3:
         {
            System.out.println("*** Se enciende el interruptor de la bombilla de la cocina ***");
            cocina.enciende();
            break;
          }
          case 4:
          {
             System.out.println("*** Se apaga el interruptor de la bombilla de la habitacion***");
             habitacion.apaga();
             break;
          }
          case 5:
          {
             System.out.println("*** Se apaga el interruptor de la bombilla de salon ***");
             salon.apaga();
             break;
          }
          case 6:
          {
             System.out.println("*** Se apaga el interruptor de la bombilla de la cocina ***");
             cocina.apaga();
             break;
          }
          case 7:
          {
             System.out.println(" *** Interruptor general activado ***");
             Bombilla.activaGeneral();
```

```
            break;
          }
          case 8:
          {
            System.out.println("*** Interruptor general desactivado ***");
            Bombilla.desactivaGeneral();
            break;
          }
          case 9:
          {
            System.out.println();
            System.out.println("*** Consultando el estado de las bombillas ***");
            System.out.println("---------------------------------------------");
            System.out.print("-  La bombilla de la babitación esta ");
            if (habitacion.encendida())
              System.out.println("encendida");
            else
              System.out.println("apagada");

           System.out.print("-  La bombilla del salón esta ");
           if (salon.encendida())
              System.out.println("encendida");
           else
              System.out.println("apagada");

           System.out.print("-  La bombilla de la cocina esta ");
           if (cocina.encendida())
              System.out.println("encendida");
           else
              System.out.println("apagada");

           break;
          }
          case 0:
          {
              System.out.println("Fin del programa");
              break;
          }
          default:
              System.out.println("Selecciona otra opcion");

      }
}


//Programa principal
public static void main(String args[])
{
   Scanner leer=new Scanner(System.in);
   int op;

   System.out.println("Se crean las bombillas de prueba");
   Bombilla bombilla_habitacion=new Bombilla();
   System.out.println("Creada la bombilla de la habitacion");
   Bombilla bombilla_salon=new Bombilla();
   System.out.println("Creada la bombilla del salon");
   Bombilla bombilla_cocina=new Bombilla();
   System.out.println("Creada la bombilla de la cocina");
```

```
    do
    {
       op=menu();

       ejecutarOpcion(op,bombilla_salon,bombilla_habitacion,bombilla_cocina);

    }while (op!=0);


  }
}
```

RECUERDA. Un método que modifique un atributo estático DEBE ser estático

EJEMPLO 2

Vamos a desarrollar tres clases que se asociarán para desarrollar un programa:

Desarrollar una clase llamada **Asignatura** que:

- Tenga tres atributos private:
  - El identificador de la asignatura (tipo int)
  - El nombre de la asignatura (tipo String)
  - La calificación en la asignatura (tipo double)
- Tenga un constructor con dos parámetros: recibe un identificador y un nombre.
- Tenga un método get para cada uno de los atributos
- Tenga un método set para la calificación.

Desarrollar una clase llamada **Alumno** que:

- Tenga un atributo nombre.
- Tenga un atributo que almacene las calificaciones de las asignaturas donde está matriculado (array de **Asignatura**)
- Tenga un atributo índice para controlar en qué posición del array de calificaciones se puede insertar. Los objetos **Asignatura** se introducirán en el array calificaciones en la posición indicada por índice.
- Tenga un constructor con dos parámetros que indiquen el nombre y el número de asignaturas donde está matriculado.
- Tenga un método **get** para obtener el nombre.
- Tenga un método **insertarCalificacion** para insertar una calificación. El método insertará un objeto **Asignatura** en el array calificaciones. (en este método se deberá comprobar que no se añadan más calificaciones que el número máximo de asignaturas en las que está matriculado el alumno).
- Tenga un método **obtenerCalificaciones** para obtener las calificaciones (retorna un array de **Asignatura**).

Desarrollar una clase **Profesor** que:

- Un atributo nombre
- Tenga un método **ponerCalificaciones** que recibe un parámetro de tipo Alumno. Pondrá una calificación aleatoria a cada una de las asignaturas del alumno.
- Tenga un método **calcularMedia** que recibe parámetro de tipo Alumno y devuelve un **double**.

Nota: Para calcular números aleatorios se emplea el método **random** de la clase **Math**.

Cree un Alumno matriculado en tres asignaturas.

- Cree tres objetos de tipo **Asignatura** y se los añada al alumno creado.
- Cree un Profesor que le ponga calificaciones y muestre por pantalla la media del alumno.
- Mostrar la calificación de cada una de las asignaturas (indicado el nombre y calificación de cada una) y también el nombre del alumno.

Definición de la clase Asignatura

```
public class Asignatura
{
        //Atributos
        private int identificador;
        private String nombre;
        private double calificacion;

        //Constructor
        public Asignatura(int ident, String nom)
        {
                this.identificador = ident;
                this.nombre = nom;
        }


        //métodos get
        public int obtenerIdentificador()
        {
           return this.identificador;
        }

        public String obtenerNombre()
        {
           return this.nombre;
        }

        public double obtenerCalificacion()
        {
           return this.calificacion;
        }


        //metodo set
        public void ponerCalificacion(double calif)
        {
           this.calificacion = calif;
        }
}//fin de la clase Asignatura
```

Definición de la clase Alumno

```
public class Alumno
{

//Atributos. Cada alumno puede estar matriculado en una cantidad determinada de asignaturas
//Las asignaturas en las que esta matriculado el alumno se almacenan en un array que contiene objetos de tipo
//Asignatura. Al declarar los atributos se define el array pero no se crea hasta que no sepamos el número de
asignaturas
//en las que se matricula el alumno
   private String nombre;
   private Asignatura [] calificaciones; //array que almacena las asignaturas en las que se matricula el alumno
   private int indice;   //índice que siempre apuntará a la primera posición libre del array de asignaturas

//El constructor recibe el nombre del alumno y el número de asignaturas en las que se matricula
//Cuando se recibe el número de asignaturas en las que se matricula el alumno se crea el array
//El índice se inicializa a cero, en esta posición se almacenará la primera asignatura en la que se matricula el alumno
```

```
   public Alumno(String nombre, int num)
   {
      this.nombre = nombre;
      this.indice = 0;   // primera posicion libre del array
      calificaciones = new Asignatura[num];   // array para guardar las asignaturas
   }

   public String obtenerNombre()
   {
      return this.nombre;
    }


//Método que recibe un objeto de tipo Asignatura, en la que se matricula el alumno, y se almacena en el array de
//asignaturas. La asignatura se almacenará en la primera posición libre del array, la que indique el índice. Sólo
// almacena la asignatura si en el array quedan posiciones libres

   public void insertarAsignatura(Asignatura asig)
    {
           if (indice < calificaciones.length)  //Si el array de calificaciones no está completo
           {
             this.calificaciones[indice] = asig;  //almacena la asignatura
             indice++;   //avanza el índice una posición
            }
           else
           {
             System.out.println("Vector asignaturas lleno");
           }
     }
           //Método que devuelve el array de asignaturas en las que está matriculado el alumno

           public Asignatura [] obtenerCalificaciones()
           {
              return this.calificaciones;
           }
}//fin de la clase Aumno
```

Definición de la clase Profesor

```
import java.lang.Math;

public class Profesor
{
   //atributos
   private String nombre;

   //Constructor
   public Profesor(String nombre)
   {
      this.nombre = nombre;
   }

//Un profesor está relacionado directamente con un alumno e indirectamente con las asignaturas del alumno, puesto
//que solo dispone de so métodos: calcular la nota media de las calificaciones obtenidas por el alumno en cada
//asignatura en la que está matriculado y poner las notas a las asignaturas en las que está matriculado el alumno
//método que calcula la media de las notas de un alumno. En este caso, primero se obtiene el array que contiene las
// asignaturas en las que está matriculado el alumno, utilizando el método de la clase alumno obtenerCalificaciones()
// guarda el array en un array local al método y lo recorre para calcular la media
```

```
   public double calcularMedia(Alumno alum)
   {
      Asignatura notas[]=alum.obtenerCalificaciones();  //descarga las asignaturas del alumno
      double acum=0;

//recorre el array para sumar las notas
      for (int i=0; i<notas.length;i++)
      {
         acum = acum + notas[i].obtenerCalificacion();
      }

//devuelve la nota media
      return acum/notas.length;
   }

//Para que un profesor pueda poner las calificaciones a un alumno necesita acceder al array de asignaturas
//en las que está matriculado el alumno, pero no existe ningún método que permita realizar esta acción
//Para resolver este problema podemos hacer que los atributos indice y calificaciones del alumno sean protected.
//De esta forma todas las clases que se encuentren en el mismo paquete de la clase Alumno podrán acceder a estos
//atributos anteponiendo al atributo el nombre del objeto.
//La solución sería entonces

//metodo que pone las calificaciones de un alumno
   public void ponerCalificaciones(Alumno alum)
   {
      int i;
      double nota;
      for (i=0; i<alum.indice;i++)  //Mientras que i sea menor que el tamaño del array
      {
         nota = Math.random()*10; //Calculamos la nota
         alum.calificaciones[i].ponerCalificacion(nota); //ponemos la nota de la signatura
      }
   }
}//Fin de la clase Profesor
```

Clase de Prueba

```
public class Prueba
{
  public static void main ()
  {
    Asignatura notas[];  //array que recogerá las notas de un alumno para mostrar sus calificaciones
    //Creamos tres objetos Asignatura
    Asignatura as1 = new Asignatura(1,"Sistemas");
    Asignatura as2 = new Asignatura(2, "Análisis");
    Asignatura as3 = new Asignatura(3, "Programación");

    //creamos un objeto Alumno
    Alumno al1 = new Alumno("Pablo", 3);

    //creamos un objeto profesor
    Profesor p1 = new Profesor("Luis");

    //Matriculamos al alumno en las asignaturas creadas
    al1.insertarAsignatura(as1);
    al1.insertarAsignatura(as2);
    al1.insertarAsignatura(as3);

    //El profesor pone las calificaciones del alumno
    p1.ponerCalificaciones(al1);
```

```
    //Se muestran las calificaciones del alumno
    System.out.println(" las notas de " + al1.obtenerNombre() + " son:.....");

    //Obtenemos las notas del alumno
    notas=al1.obtenerCalificaciones();

   //Se muestran una a una las calificaciones
    for (int i=0; i<notas.length; i++)
    {
       System.out.println(" la nota de " + notas[i].obtenerNombre() + " es " + notas[i].obtenerCalificacion());
    }

   //Se muestra la nota media del alumno
    System.out.println(" la media es " + p1.calcularMedia(al1));
  }
}
```

RECUERDA. Cuando declaramos un atributo con el modificador protected todas las clases que pertenezcan al mismo paquete o al mismo proyecto pueden acceder directamente al atributo de la forma nombreObjeto.nombreAtributo

Si declaramos al atributo como public se puede acceder al atributo desde cualquier clase poniendo NombreClase.nombreAtributo

EJEMPLO 3

Se pretende desarrollar una aplicación en Java que simule, de una forma simplificada, un partido de tenis entre dos jugadores.

Construir clase Jugador cuyas instancias (objetos) representen jugadores de tenis con su puntuación en el ranking de la ATP (puntuación total que tienen en la Asociación de Tenistas Profesionales. Cada vez que un jugador gana un partido en una competición se aumenta en 1 su puntuación). Esta clase deberá disponer de:

- Un constructor con dos argumentos: el nombre del jugador y su puntuación en el ranking.
- Métodos para consultar el nombre y la puntuación.
- Un método para modificar la puntuación

Construir una clase Partido, cuyos objetos mantendrán referencias a los dos jugadores participantes, y al ganador del partido. Esta clase deberá disponer de:

- Un constructor con dos argumentos de tipo Jugador que genere un partido sin jugar.
- Métodos de consulta a cada uno de los jugadores del partido.
- Un métodos jugar(), que haga que el partido se juegue generando el ganador de forma aleatoria. El jugador con mayor puntuación en la ATP debe tener mayores posibilidades de ganar. Para ello se puede emplear, por ejemplo, este mecanismo:

```
resultado1= Math.random() * puntos_ATP_jugador1;
resultado2= Math.random() * puntos_ATP_jugador2;
// el que tenga mayor resultado ganará el partido
// Tened en cuenta que si el resultado es empate, se deben
//volver a generar resultados hasta que exista un ganador.
```

- El jugador que gane el partido debe aumentar su puntuación ATP en 1 punto.
- Dos métodos nombreDelGanador() y nombreDelPerdedor() que devuelvan respectivamente el ganador y el perdedor del partido. Si el partido no se ha jugado, deben hacer que se juegue antes de devolver nada.

Construir la clase Prueba que cree dos jugadores y un partido. Jugar el partido e indicar el nombre del ganador y del perdedor. Mostrar por pantalla las nuevas puntuaciones en el ranking de cada jugador. Ofrecer un menú que permita jugar tantos partidos como se desee.