CAPÍTULO 2

© F.J.Ceballos/RA-MA

PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos (POO) es un modelo de programación que utiliza objetos, ligados mediante mensajes, para la solución de problemas. Puede considerarse como una extensión natural de la programación estructurada en un intento de potenciar los conceptos de modularidad y reutilización del código.

¿A qué objetos nos referimos? Si nos paramos a pensar en un determinado problema que intentamos resolver podremos identificar entidades de interés, las cuales pueden ser objetos potenciales que poseen un conjunto de *propiedades* o atributos, y un conjunto de *métodos* mediante los cuales muestran su comportamiento. Y no sólo eso, también podremos ver, a poco que nos fijemos, un conjunto de interrelaciones entre ellos conducidas por mensajes a los que responden mediante métodos.

Veamos un ejemplo. Considere una entidad bancaria. En ella identificamos entidades que son cuentas: cuenta del cliente 1, cuenta del cliente 2, etc. Pues bien, una cuenta puede verse como un objeto que tiene unos atributos, *nombre*, *número de cuenta* y *saldo*, y un conjunto de métodos como *IngresarDinero*, *RetirarDinero*, *AbonarIntereses*, *SaldoActual*, *Transferencia* etc. En el caso de una transferencia:

```
cuenta01.Transferencia(cuenta02);
```

Transferencia sería el mensaje que el objeto *cuenta02* envía al objeto *cuenta01*, solicitando le sea hecha una transferencia, siendo la respuesta a tal mensaje la ejecución del método *Transferencia*. Trabajando a este nivel de abstracción, manipular una entidad bancaria resultará algo muy sencillo.

MECANISMOS BÁSICOS DE LA POO

Los mecanismos básicos de la programación orientada a objetos son: *objetos*, *mensajes*, *métodos* y *clases*.

Objetos

Un programa orientado a objetos se compone solamente de *objetos*, entendiendo por objeto una encapsulación genérica de datos y de los métodos para manipularlos. Dicho de otra forma, un *objeto* es una entidad que tiene unos atributos particulares, las *propiedades*, y unas formas de operar sobre ellos, los *métodos*.

Por ejemplo, una ventana de una aplicación Windows es un objeto. El color de fondo, la anchura, la altura, etc. son propiedades. Las rutinas, lógicamente transparentes al usuario, que permiten maximizar la ventana, minimizarla, etc. son métodos.

Mensajes

Cuando se ejecuta un programa orientado a objetos, los objetos están recibiendo, interpretando y respondiendo a *mensajes* de otros objetos. Esto marca una clara diferencia con respecto a los elementos de datos pasivos de los sistemas tradicionales. En la POO un *mensaje* está asociado con un método, de tal forma que cuando un objeto recibe un mensaje la respuesta a ese mensaje es ejecutar el método asociado.

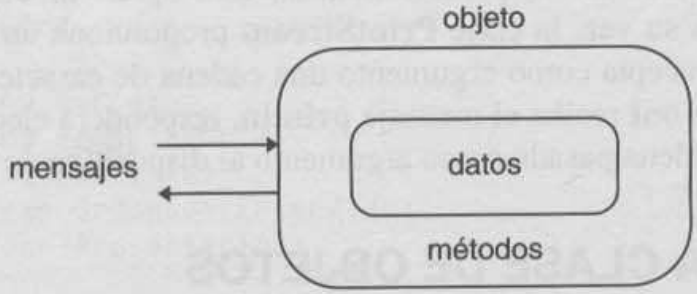
Por ejemplo, cuando un usuario quiere maximizar una ventana de una aplicación Windows, lo que hace simplemente es pulsar el botón de la misma que realiza esa acción. Eso, provoca que Windows envíe un mensaje a la ventana para indicar que tiene que maximizarse. Como respuesta a este mensaje se ejecutará el método programado para ese fin.

Métodos

Un *método* se implementa en una *clase* de objetos y determina cómo tiene que actuar el objeto cuando recibe el *mensaje* vinculado con ese método. A su vez, un *método* puede también enviar *mensajes* a otros objetos solicitando una acción o información.

En adición, las *propiedades* (atributos) definidas en la clase permitirán almacenar información para dicho objeto.

Cuando se diseña una clase de objetos, la estructura más interna del *objeto* se oculta a los usuarios que lo vayan a utilizar, manteniendo como única conexión con el exterior, los *mensajes*. Esto es, los datos que están dentro de un objeto solamente podrán ser manipulados por los *métodos* asociados al propio objeto.



Según lo expuesto, podemos decir que la ejecución de un programa orientado a objetos realiza fundamentalmente tres cosas:

1. Crea los objetos necesarios.
2. Los mensajes enviados a unos y a otros objetos dan lugar a que se procese internamente la información.
3. Finalmente, cuando los objetos no son necesarios, son borrados, liberándose la memoria ocupada por los mismos.

Clases

Una *clase* es un tipo de objetos definido por el usuario. Una *clase* equivale a la generalización de un tipo específico de objetos. Por ejemplo, piense en un molde para hacer flanes; el molde es la clase y los flanes los objetos.

Un *objeto* de una determinada clase se crea en el momento en que se define una variable de dicha *clase*. Por ejemplo, la siguiente línea declara el objeto *cliente01* de la clase o tipo *CCuenta*.

```
CCuenta cliente01 = new CCuenta(); // nueva cuenta
```

Algunos autores emplean el término instancia (traducción directa de *instance*), en el sentido de que una instancia es la representación concreta y específica de una clase; por ejemplo, *cliente01* es un instancia de la clase *CCuenta*. Desde este punto de vista, los términos instancia y objeto son lo mismo. El autor prefiere utilizar el término *objeto*, o bien *ejemplar*.

Cuando escribe un programa utilizando un lenguaje orientado a objetos, no se definen objetos verdaderos, se definen clases de objetos, donde una clase se ve como una plantilla para múltiples objetos con características similares.

Afortunadamente no tendrá que escribir todas las clases que necesite en su programa, porque Java proporciona una biblioteca de clases estándar para realizar las operaciones más habituales que podamos requerir. Por ejemplo, en el capítulo anterior, vimos que la clase **System** tenía un atributo **out** que era un objeto de la clase **PrintStream** que, de forma predeterminada, está ligado al dispositivo de salida (a la pantalla). A su vez, la clase **PrintStream** proporciona un método denominado **println** que acepta como argumento una cadena de caracteres. De esta forma, cuando el objeto **out** reciba el mensaje **println**, responderá ejecutando este método, que envía la cadena pasada como argumento al dispositivo de salida.

CÓMO CREAR UNA CLASE DE OBJETOS

Según lo expuesto hasta ahora, un objeto contiene, por una parte, atributos que definen su estado, y por otra, operaciones que definen su comportamiento. También sabemos que un objeto es la representación concreta y específica de una clase. ¿Cómo se escribe una clase de objetos? Como ejemplo, podemos crear una clase *COrdenador*. Abra su entorno de programación integrado favorito y escriba paso a paso el ejemplo que a continuación empezamos a desarrollar:

```
class COrdenador
{
  // ...
}
```

Observamos que para declarar una clase hay que utilizar la palabra reservada **class** seguida del nombre de la clase y del cuerpo de la misma. El cuerpo de la clase incluirá entre { y } los atributos y los métodos u operaciones que definen su comportamiento.

Los atributos son las características individuales que diferencian un objeto de otro. El color de una ventana Windows, la diferencia de otras; el D.N.I. de una persona la identifica entre otras; el modelo de un ordenador le distingue entre otros; etc.

La clase *COrdenador* puede incluir los siguientes atributos:

- Marca: Mitac, Toshiba, Ast
- Procesador: Intel, AMD
- Pantalla: TFT, DSTN, STN

Los atributos también pueden incluir información sobre el estado del objeto; por ejemplo, en el caso de un ordenador, si está encendido o apagado, si la presentación en pantalla está activa o inactiva, etc.

◊ Dispositivo: encendido, apagado
◊ Presentación: activa, inactiva

Todos los atributos son definidos en la clase por variables:

```
class COrdenador
{
  String Marca;
  String Procesador;
  String Pantalla;
  boolean OrdenadorEncendido;
  boolean Presentación;

  // ...
}
```

Observe que se han definido cinco atributos: tres de ellos, *Marca*, *Procesador* y *Pantalla*, pueden contener una cadena de caracteres (una cadena de caracteres es un objeto de la clase **String** perteneciente a la biblioteca estándar). Los otros dos atributos, *OrdenadorEncendido* y *Presentación*, son de tipo **boolean** (un atributo de tipo **boolean** puede contener un valor **true** o **false**; verdadero o falso). Debe respetar las mayúsculas y las minúsculas.

No vamos a profundizar en los detalles de la sintaxis de este ejemplo ya que el único objetivo ahora es entender la definición de una clase con sus partes básicas. El resto de la sintaxis y demás detalles se irán exponiendo poco a poco en sucesivos capítulos.

El comportamiento define las acciones que el objeto puede emprender. Por ejemplo, pensando acerca de un objeto de la clase *COrdenador*, esto es, de un ordenador, algunas acciones que éste puede hacer son:

◊ Ponerse en marcha
◊ Apagarse
◊ Desactivar la presentación en la pantalla
◊ Activar la presentación en la pantalla
◊ Cargar una aplicación

Para definir este comportamiento hay que crear métodos. Los métodos son rutinas de código definidas dentro de la clase, que se ejecutan en respuesta a alguna acción tomada desde dentro de un objeto de esa clase o desde otro objeto de la misma o de otra clase. Recuerde que los objetos se comunican mediante mensajes.

Como ejemplo, vamos a agregar a la clase *COrdenador* un método que responda a la acción de ponerlo en marcha:

```
void EncenderOrdenador()
{
  if (OrdenadorEncendido == true) // si está encendido...
    System.out.println("El ordenador ya está en marcha.");
  else // si no está encendido, encenderlo.
  {
    OrdenadorEncendido = true;
    System.out.println("El ordenador se ha encendido.");
  }
}
```

Como se puede observar un método consta de su nombre precedido por el tipo del valor que devuelve cuando finalice su ejecución (la palabra reservada **void** indica que el método no devuelve ningún valor) y seguido por una lista de parámetros separados por comas y encerrados entre paréntesis (en el ejemplo, no hay parámetros). Los paréntesis indican a Java que el identificador (*EncenderOrdenador*) se refiere a un método y no a un atributo. A continuación se escribe el cuerpo del método encerrado entre { y }. Usted ya conoce algunos métodos, llamados en otros contextos funciones; seguro que conoce la función logaritmo que devuelve un valor real correspondiente al logaritmo del valor pasado como argumento.

El método *EncenderOrdenador* comprueba si el ordenador está encendido; si lo está, simplemente visualiza un mensaje indicándolo; si no lo está, se enciende y lo comunica mediante un mensaje.

Agreguemos un método más para que el objeto nos muestre su estado:

```
void Estado()
{
  System.out.println("\nEstado del ordenador:" +
                     "\nMarca " + Marca +
                     "\nProcesador " + Procesador +
                     "\nPantalla " + Pantalla + "\n");
  if (OrdenadorEncendido == true) // si el ordenador está encendido...
    System.out.println("El ordenador está encendido.");
  else // si no está encendido...
    System.out.println("El ordenador está apagado.");
}
```

El método *Estado* visualiza los atributos específicos de un objeto. La secuencia de escape *\n*, así se denomina, introduce un retorno de carro más un avance de línea (caracteres ASCII, CR LF).

En este instante, si nuestras pretensiones sólo son las expuestas hasta ahora, ya tenemos creada la clase *COrdenador*. Para poder crear objetos de esta clase y trabajar con ellos, tendremos que escribir un programa, o bien añadir a esta clase

el método **main**. Siempre que se trate de una aplicación (no de un *applet*) es obligatorio que la clase que define el comienzo de la misma incluya un método **main**. Cuando se ejecuta una clase Java compilada que incluye un método **main**, éste es lo primero que se ejecuta.

Hagamos lo más sencillo, añadir el método **main** a la clase *COrdenador*. El código completo, incluyendo el método **main**, se muestra a continuación:

```
class COrdenador
{
  String Marca;
  String Procesador;
  String Pantalla;
  boolean OrdenadorEncendido;
  boolean Presentación;

  void EncenderOrdenador()
  {
    if (OrdenadorEncendido == true) // si está encendido...
      System.out.println("El ordenador ya está encendido.");
    else // si no está encendido, encenderlo.
    {
      OrdenadorEncendido = true;
      System.out.println("El ordenador se ha encendido.");
    }
  }

  void Estado()
  {
    System.out.println("\nEstado del ordenador:" +
                       "\nMarca " + Marca +
                       "\nProcesador " + Procesador +
                       "\nPantalla " + Pantalla + "\n");
    if (OrdenadorEncendido == true) // si el ordenador está encendido...
      System.out.println("El ordenador está encendido.");
    else // si no está encendido...
      System.out.println("El ordenador está apagado.");
  }

  public static void main (String[] args)
  {
    COrdenador MiOrdenador = new COrdenador();
    MiOrdenador.Marca = "Ast";
    MiOrdenador.Procesador = "Intel Pentium";
    MiOrdenador.Pantalla = "TFT";
    MiOrdenador.EncenderOrdenador();
    MiOrdenador.Estado();
  }
}
```

El método **main** siempre se declara público y estático, no devuelve un resultado y tiene un parámetro *args* que es una matriz de una dimensión de cadenas de caracteres. En los capítulos siguientes aprenderá para qué sirve. Analicemos el método **main** para que tenga una idea de lo que hace:

- La primera línea crea un objeto de la clase *COrdenador* y almacena una referencia al mismo en la variable *MiOrdenador*. Esta variable la utilizaremos para acceder al objeto en las siguientes líneas. Ahora quizás empiece a entender por qué anteriormente decíamos que un programa orientado a objetos se compone solamente de objetos.

- Las tres líneas siguientes establecen los atributos del objeto referenciado por *MiOrdenador*. Se puede observar que para acceder a los atributos o propiedades del objeto se utiliza el operador punto (.). De esta forma quedan eliminadas las ambigüedades que surgirían si hubiéramos creado más de un objeto.

- En las dos últimas líneas el objeto recibe los mensajes *EncenderOrdenador* y *Estado*. La respuesta a esos mensajes es la ejecución de los métodos respectivos, que fueron explicados anteriormente. Aquí también se puede observar que para acceder a los métodos del objeto se utiliza el operador punto.

En general, para acceder a un miembro de una clase (atributo o método) se utiliza la sintaxis siguiente:

nombre_objeto.nombre_miembro

Guarde la aplicación con el nombre *COrdenador.java*. Después compílela y ejecútela. Se puede observar que los resultados son los siguientes:

```
El ordenador se ha encendido.

Estado del ordenador:
Marca Ast
Procesador Intel Pentium
Pantalla TFT

El ordenador está encendido.
```

Otra forma de crear objetos de una clase y trabajar con ellos es incluir esa clase en el mismo fichero fuente de una clase aplicación, entendiendo por clase aplicación una que incluya el método **main** y cree objetos de otras clases. Por ejemplo, volvamos al instante justo antes de añadir el método **main** a la clase *COrdenador* y añadamos una nueva clase pública denominada *CMiOrdenador* que incluya el método **main**. El resultado tendrá el esqueleto que se observa a continuación:

```
public class CMiOrdenador
{
  public static void main (String[] args)
  {
    // ...
  }
}

class COrdenador
{
  // ...
}
```

En el capítulo anterior aprendimos que una aplicación está basada en una clase cuyo nombre debe coincidir con el del programa fuente que la contenga, respetando mayúsculas y minúsculas. Por lo tanto, guardemos el código escrito en un fichero fuente denominado *CMiOrdenador.java*. Finalmente, completamos el código como se observa a continuación, y compilamos y ejecutamos la aplicación. Ahora es la clase *CMiOrdenador* la que crea un objeto de la clase *COrdenador*. El resto del proceso se desarrolla como se explicó en la versión anterior. Lógicamente, los resultados que se obtengan serán los mismos que obtuvimos con la versión anterior.

```
public class CMiOrdenador
{
  public static void main (String[] args)
  {
    COrdenador MiOrdenador = new COrdenador();
    MiOrdenador.Marca = "Ast";
    MiOrdenador.Procesador = "Intel Pentium";
    MiOrdenador.Pantalla = "TFT";
    MiOrdenador.EncenderOrdenador();
    MiOrdenador.Estado();
  }
}

class COrdenador
{
  String Marca;
  String Procesador;
  String Pantalla;
  boolean OrdenadorEncendido;
  boolean Presentación;

  void EncenderOrdenador()
  {
    if (OrdenadorEncendido == true) // si está encendido...
      System.out.println("El ordenador ya está encendido.");
```

```
    else // si no está encendido, encenderlo.
    {
      OrdenadorEncendido = true;
      System.out.println("El ordenador se ha encendido.");
    }
  }

  void Estado()
  {
    System.out.println("\nEstado del ordenador:" +
                       "\nMarca " + Marca +
                       "\nProcesador " + Procesador +
                       "\nPantalla " + Pantalla + "\n");
    if (OrdenadorEncendido == true) // si el ordenador está encendido...
      System.out.println("El ordenador está encendido.");
    else // si no está encendido...
      System.out.println("El ordenador está apagado.");
  }
}
```

La aplicación *CMiOrdenador.java* que acabamos de completar tiene dos clases: la clase aplicación *CMiOrdenador* y la clase *COrdenador*. Observe que la clase aplicación es pública (**public**) y la otra no. Cuando incluyamos varias clases en un fichero fuente, sólo una puede ser pública y su nombre debe coincidir con el del fichero donde se guardan. Al compilar este fichero, Java creará tanto ficheros *.class* como clases separadas hay.

Según lo expuesto hasta ahora, en esta nueva versión también tenemos un fichero, el que almacena la aplicación, que tiene el mismo nombre que la clase que incluye el método **main**, que es por donde se empezará a ejecutar la aplicación.

CARACTERÍSTICAS DE LA POO

Las características fundamentales de la POO son: *abstracción*, *encapsulamiento*, *herencia* y *polimorfismo*.

Abstracción

Por medio de la abstracción conseguimos no detenernos en los detalles concretos de las cosas que no interesen en cada momento, sino generalizar y centrarse en los aspectos que permitan tener una visión global del problema. Por ejemplo, el estudio de un ordenador podemos realizarlo a nivel de funcionamiento de sus circuitos electrónicos, en términos de corriente, tensión, etc., o a nivel de transferencia entre registros, centrándose así el estudio en el flujo de información entre las unidades que lo componen (memoria, unidad aritmética, unidad de control, registros,

etc.), sin importarnos el comportamiento de los circuitos electrónicos que componen estas unidades.

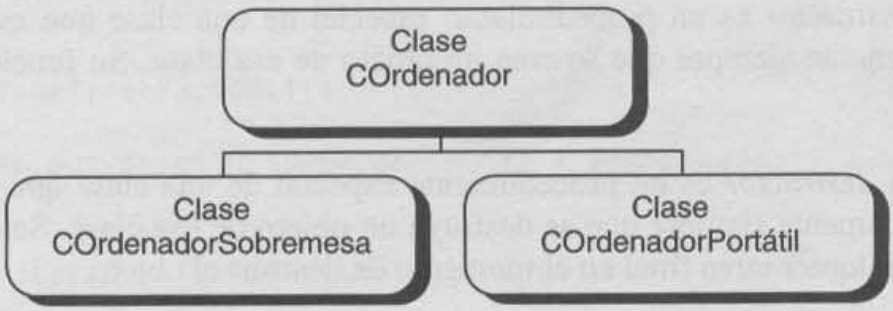
Encapsulamiento

Esta característica permite ver un objeto como una caja negra en la que se ha introducido de alguna manera toda la información relacionada con dicho objeto. Esto nos permitirá manipular los objetos como unidades básicas, permaneciendo oculta su estructura interna.

La abstracción y la encapsulación están representadas por la *clase*. La *clase* es una abstracción, porque en ella se definen las propiedades o atributos de un determinado conjunto de objetos con características comunes, y es una encapsulación porque constituye una caja negra que encierra tanto los datos que almacena cada objeto como los métodos que permiten manipularlos.

Herencia

La herencia permite el acceso automático a la información contenida en otras clases. De esta forma, la reutilización del código está garantizada. Con la herencia todas las clases están clasificadas en una jerarquía estricta. Cada clase tiene su superclase (la clase superior en la jerarquía), y cada clase puede tener una o más subclases (las clases inferiores en la jerarquía).



Las clases que están en la parte inferior en la jerarquía se dice que *heredan* de las clases que están en la parte superior en la jerarquía.

El término heredar significa que las subclases disponen de todos los métodos y propiedades de su superclase. Este mecanismo proporciona una forma rápida y cómoda de extender la funcionalidad de una clase.

En Java cada clase sólo puede tener una superclase, lo que se denomina *herencia simple*. En otros lenguajes orientados a objetos, como C++, las clases pue-

den tener más de una superclase, lo que se conoce como *herencia múltiple*. En este caso, una clase comparte los métodos y propiedades de varias clases. Esta característica, proporciona un poder enorme a la hora de crear clases, pero complica excesivamente la programación, por lo que es de escasa o nula utilización. Java, intentando facilitar las cosas, soluciona este problema de comportamiento compartido utilizando *interfaces*.

Una *interfaz* es una colección de nombres de métodos, sin incluir sus definiciones, que puede ser añadida a cualquier clase para proporcionarla comportamientos adicionales no incluidos en los métodos propios o heredados.

Todo esto será objeto de un estudio amplio en capítulos posteriores.

Polimorfismo

Esta característica permite implementar múltiples formas de un mismo método, dependiendo cada una de ellas de la clase sobre la que se realice la implementación. Esto hace que se pueda acceder a una variedad de métodos distintos (todos con el mismo nombre) utilizando exactamente el mismo medio de acceso. Más adelante, cuando estudie en profundidad las clases y subclases, estará en condiciones de entender con claridad la utilidad de esta característica.

CONSTRUCTORES Y DESTRUCTORES

Un *constructor* es un procedimiento especial de una clase que es llamado automáticamente siempre que se crea un objeto de esa clase. Su función es iniciar el objeto.

Un *destructor* es un procedimiento especial de una clase que es llamado automáticamente siempre que se destruye un objeto de esa clase. Su función es realizar cualquier tarea final en el momento de destruir el objeto.

EJERCICIOS RESUELTOS

Para practicar con una aplicación más, escriba el siguiente ejemplo y pruebe los resultados. Hágalo primero desde la línea de órdenes y después con el entorno de desarrollo integrado preferido por usted. El siguiente ejemplo muestra una clase para representar números racionales. Esta clase puede ser útil porque muchos números no pueden ser representados exactamente utilizando un número fraccionario. Por ejemplo, el número racional *1/3* representado como un número

fraccionario sería *0,333333*, valor más fácil de manipular, pero a costa de perder precisión. Evidentemente, *1/3 * 3 = 1*, pero *0,333333 * 3 = 0,999999*.

Pensando en un número racional como si de un objeto se tratara, es fácil deducir que sus atributos son dos: el *numerador* y el *denominador*. Y los métodos aplicables sobre los números racionales son numerosos: suma, resta, multiplicación, simplificación, etc. Pero en base a los conocimientos adquiridos, sólo añadiremos dos métodos sencillos: uno, *AsignarDatos*, para establecer los valores del numerador y del denominador; y otro, *VisualizarRacional*, para visualizar un número racional.

Edición

Abra el procesador de textos o el editor de su entorno integrado y edite la aplicación propuesta, como se muestra a continuación:

```
class CRacional
{
  int Numerador;
  int Denominador;

  void AsignarDatos(int num, int den)
  {
    Numerador = num;
    if (den == 0) den = 1; // el denominador no puede ser cero
    Denominador = den;
  }

  void VisualizarRacional()
  {
    System.out.println(Numerador + "/" + Denominador);
  }

  public static void main (String[] args)
  {
    // Punto de entrada a la aplicación
    CRacional r1 = new CRacional(); // crear un objeto CRacional

    r1.AsignarDatos(2, 5);
    r1.VisualizarRacional();
  }
}
```

Una vez editado el programa, guárdelo en el disco con el nombre *CRacional.java*.

¿Qué hace esta aplicación?

Fijándonos en el método principal, **main,** vemos que se ha declarado un objeto *r1* de la clase *CRacional.*

```
CRacional r1 = new CRacional();
```

En el siguiente paso se envía el mensaje *AsignarDatos* al objeto *r1*. El objeto responde a este mensaje ejecutando su método *AsignarDatos* que almacena el valor 2 en su numerador y el valor 5 en su denominador; ambos valores han sido pasados como argumentos.

```
r1.AsignarDatos(2, 5);
```

Finalmente, se envía el mensaje *VisualizarRacional* al objeto *r1*. El objeto responde a este mensaje ejecutando su método *VisualizarRacional* que visualiza sus atributos numerador y denominador en forma de quebrado; en nuestro caso, el número racional 2/5.

```
r1.VisualizarRacional();
```

Para finalizar, compile, ejecute la aplicación y observe que el resultado es el esperado.

EJERCICIOS PROPUESTOS

1. Añada a la aplicación *COrdenador.java* el método *ApagarOrdenador*.

2. Diseñe una clase *CCoche* que represente coches. Incluya los atributos *marca*, *modelo* y *color*; y los métodos que simulen, enviando mensajes, las acciones de arrancar el motor, cambiar de velocidad, acelerar, frenar y parar el motor.