Programación orientada a objetos

Capítulo 3

Interacción de objetos

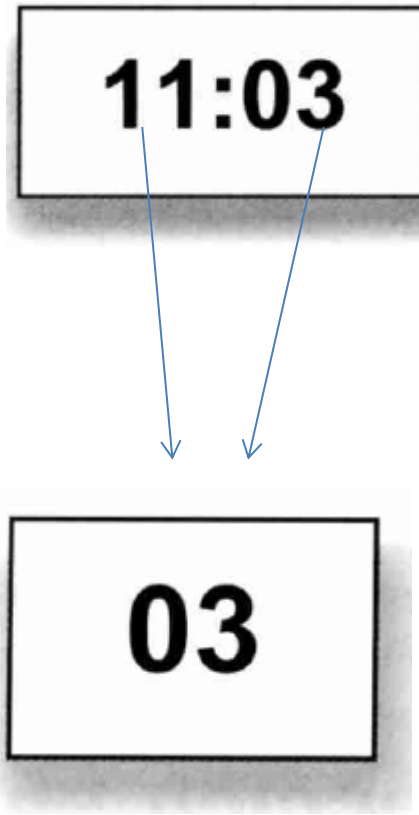
TEMA 6: Interacción entre objetos. Semana 4	

1. Abstracción .
2. Modularidad.
3. Comparación de diagramas de clases con diagramas de objetos.
4. Tipos primitivos y tipos objeto..
5. Objetos que crean objetos.
6. Constructores múltiples.
7. Llamadas a métodos.
  1. Llamadas a métodos internos.
  2. Llamadas a métodos externos.
8. Referencia a parámetros del propio objeto: la palabra clave this.
9. Depuración de código.
	1. Estudiar el Capítulo 3 del libro base para la "Unidad Didáctica II".
	2. Leer los Apéndices B y F del libro base para la "Unidad Didáctica II". Realizar los ejercicios propuestos en el libro base.
	3. Definir los campos y métodos necesarios para la resolución de la práctica.

3.2 Abstracción y modularización

Concepto

La **abstracción** es la habilidad de ignorar los detalles de las partes para centrar la atención en un nivel más alto de un problema.



Concepto

La **modularización** es el proceso de dividir un todo en partes bien definidas que pueden ser construidas y examinadas separadamente, las que interactúan de maneras bien definidas.

3.5 Implementación de la pantalla del reloj

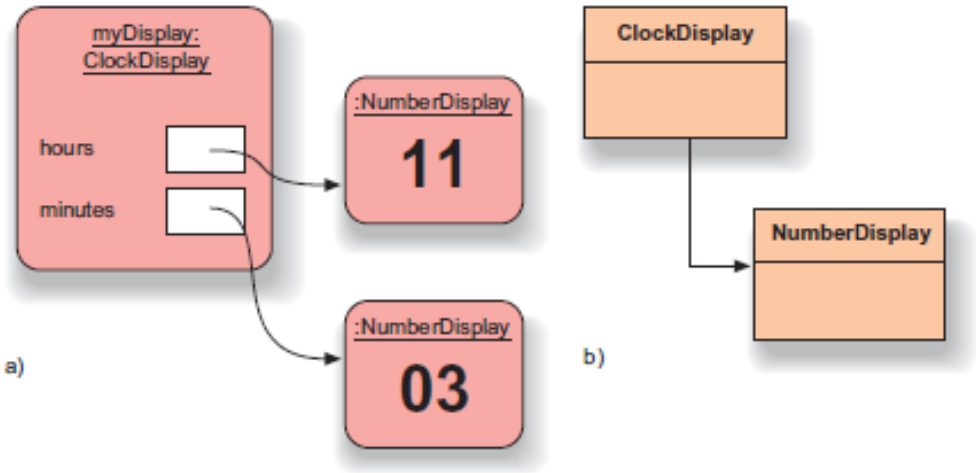
Concepto

Las clases definen tipos. El nombre de una clase puede ser usado como el tipo de una variable. Las variables cuyo tipo es una clase pueden almacenar objetos de dicha clase.

```
public class VisorDeNumeros
{
private int limite;
private int valor;
  Se omitieron los constructores y los métodos.
}
```

```
public class VisorDeReloj
{
private VisorDeNumeros horas;
private VisorDeNumeros minutos;
  Se omitieron los constructores y los métodos.
}
```

3.6 Diagramas de clases y diagramas de objetos



El **diagrama de clases** muestra las clases de una aplicación y las relaciones entre ellas. Da información sobre el código. Representa la vista estática de un programa.

El **diagrama de objetos** muestra los objetos y sus relaciones en un momento dado de la ejecución de una aplicación. Da información sobre los objetos en tiempo de ejecución. Representa la vista dinámica de un programa.

Cuando una variable almacena un objeto, el objeto no se almacena directamente en la variable, lo que la variable contiene es una referencia al objeto

Concept:

Object references. Variables of **object types** store references to objects.

3.7 Tipos primitivos y tipos de objetos

Los **tipos primitivos** en Java son todos los tipos que no son objetos. Los tipos primitivos más comunes son los tipos `int`, `boolean`, `char`, `double` y `long`. Los tipos primitivos no poseen métodos.

3.8 ver código de ClokDisplay

Clase “NumberDisplay”

```
public class NumberDisplay
{
   private int limit;
   private int value;

   /**
   * Constructor for objects of class NumberDisplay.
   * Set the limit at which the display rolls over.
   */
   public NumberDisplay(int rollOverLimit)
   {
      limit = rollOverLimit;
      value = 0;
   }
```

```
  /**
     * Return the current value.
     */
    public int getValue()
    {
       return value;
    }
```

```
public String getDisplayValue()
   {
      if(value < 10) {
         return "0" + value;
      }
      else {
         return "" + value;
      }
   }
   /**
    * Set the value of the display to the new specified value. If the new
    * value is less than zero or over the limit, do nothing.
    */
   public void setValue(int replacementValue)
   {
      if((replacementValue >= 0) && (replacementValue < limit)) {
         value = replacementValue;
      }
   }
  /**
   * Increment the display value by one, rolling over to zero if the
   * limit is reached.
   */
   public void increment()
   {
      value = (value + 1) % limit;
   }
}
```

Operadores Lógicos

Los operadores lógicos operan con valores booleanos (verdadero o falso) y producen como resultado un nuevo valor booleano. Los tres operadores lógicos más importantes son «y», «o» y «no». En Java se escriben:

&& (y)

|| (o)

! (no)

La expresión

a && b

es verdadera si tanto **a** como **b** son verdaderas, en todos los otros casos es falsa. La expresión

a || b

es verdadera si alguna de las dos es verdadera, puede ser **a** o puede ser **b** o pueden ser las dos; si ambas son falsas el resultado es falso. La expresión

!a

es verdadera si **a** es falso, y es falsa si **a** es verdadera.

Concatenación de cadenas

El operador suma (+) tiene diferentes significados dependiendo del tipo de sus operandos. Si ambos operandos son números, el operador + representa la adición tal como esperamos. Por lo tanto,

```
42 + 12
```

suma esos dos números y su resultado es 54. Sin embargo, si los operandos son cadenas, el significado del signo más es la concatenación de cadenas y el resultado es una única cadena compuesta por los dos operandos. Por ejemplo, el resultado de la expresión

```
"Java" + "con BlueJ"
```

es una sola cadena que es

```
“Javacon BlueJ”
```

El operador módulo

El último método de la clase `VisorDeNumeros` incrementa el valor del visor en 1 y cuida que el valor vuelva a ser cero cuando alcanza el límite:

```
public void incrementar()
      {
            valor = (valor + 1) % limite;
      }
```

Este método usa el operador *módulo* (%). El operador módulo calcula el resto de una división entera. Por ejemplo, el resultado de la división

```
27/4
```

puede expresarse en números enteros como

```
resultado = 6, resto = 3
```

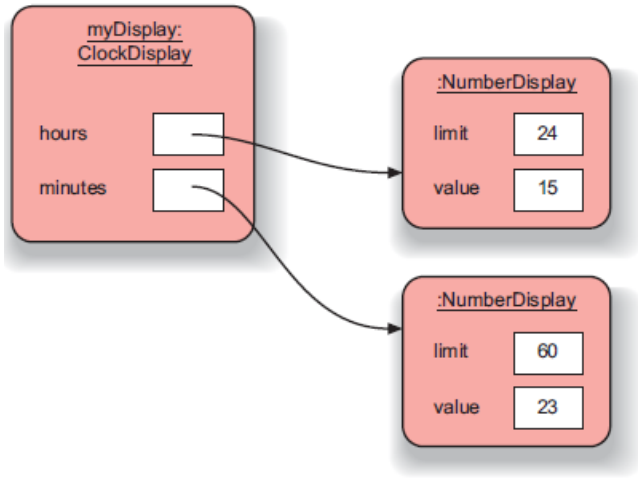
La operación módulo justamente devuelve el resto de la división, por lo que el resultado de la expresión (27%4) será 3.

Clase “ClockDisplay”

```
 */
public class ClockDisplay
{
   private NumberDisplay hours;
   private NumberDisplay minutes;
   private String displayString;    // simulates the
actual display

   /**
    * Constructor for ClockDisplay objects. This
constructor
    * creates a new clock set at 00:00.
    */
   public ClockDisplay()
   {
      hours = new NumberDisplay(24);
      minutes = new NumberDisplay(60);
      updateDisplay();
   }
```

```
   /**
    * Constructor for ClockDisplay objects. This
constructor
    * creates a new clock set at the time specified
by the
    * parameters.
    */
   public ClockDisplay(int hour, int minute)
   {
      hours = new NumberDisplay(24);
      minutes = new NumberDisplay(60);
      setTime(hour, minute);
   }
```

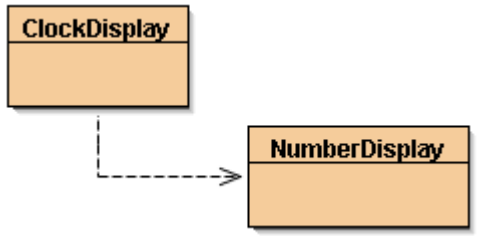


```
    /**
     * This method should get called once every minute
- it makes
     * the clock display go one minute forward.
     */
    public void timeTick()
    {
        minutes.increment();
        if(minutes.getValue() == 0) {  // it just rolled over!
            hours.increment();
        }
        updateDisplay();
    }


     /**
         * Update the internal string that represents the
     display.
         */
        private void updateDisplay()
        {
            displayString = hours.getDisplayValue() + ":" +
                        minutes.getDisplayValue();
        }
     }
```

```
     /**
         * Set the time of the display to the specified
hour and
         * minute.
         */
        public void setTime(int hour, int minute)
        {
            hours.setValue(hour);
            minutes.setValue(minute);
            updateDisplay();
        }

        /**
         * Return the current time of this display in
the format HH:MM.
         */
        public String getTime()
        {
            return displayString;
        }
```



3.9 Objetos que crean objetos

Creación de objetos. Los objetos pueden crear otros objetos usando el operador **new**.

```
public class VisorDeReloj
{
    private VisorDeNumeros horas;
    private VisorDeNumeros minutos;

    Se omitieron los restantes campos.

    public VisorDeReloj()
    {
        horas = new VisorDeNumeros(24);
        minutos = new VisorDeNumeros(60);
        actualizarVisor();
    }

    Se omitieron los métodos.
}
```

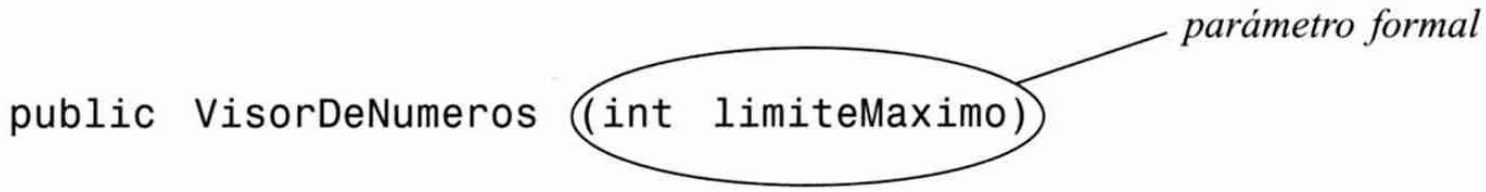
```
new NombreDeClase (lista-de-parámetros)
```

Parámetro formal parámetro actual

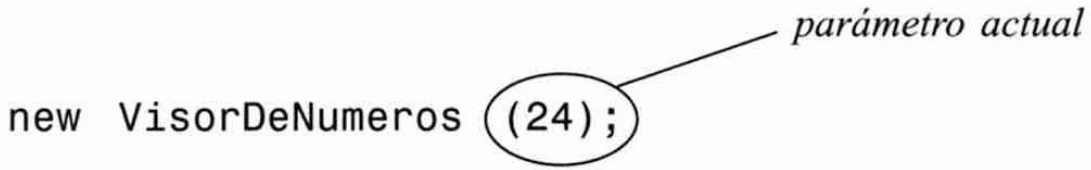
La operación new hace dos cosas:

1. Crea un nuevo objeto de la clase nombrada (en este caso, `VisorDeReloj`).
2. Ejecuta el constructor de dicha clase.

Si el constructor de la clase tiene parámetros, los parámetros actuales deben ser proporcionados en la sentencia new. Por ejemplo, el constructor de la clase `VisorDeNumeros` fue definido para esperar un parámetro de tipo entero:



Por lo tanto, la operación new sobre la clase `VisorDeNumeros` que invoca a este constructor, debe proveer un parámetro actual de tipo entero para que coincida con el encabezado que define el constructor:



3.10 Constructores múltiples

Sobrecarga. Una clase puede contener más de un constructor o más de un método con el mismo nombre, siempre y cuando tengan distintos conjuntos de parámetros que se diferencien por sus tipos.

Al crear objetos `visorDeReloj`, seguramente habrá notado que el menú contextual ofrece dos formas de hacerlo:

```
new VisorDeReloj()
new VisorDeReloj(hora, minuto)
```

Es así porque la clase contiene dos constructores que proveen formas alternativas de inicializar un objeto `VisorDeReloj`. Si se usa el constructor que no tiene parámetros, la primer hora que se mostrará en el reloj será 00:00. Por otra parte, si desea tener una hora inicial diferente, puede establecerla usando el segundo constructor. Es común que las declaraciones de clases contengan versiones alternativas de constructores o métodos que proporcionan varias maneras de llevar a cabo una tarea en particular mediante diferentes conjuntos de parámetros. Este punto se conoce como *sobrecarga* de un constructor o método.

3.11 Llamadas a métodos

3.11.1 Llamadas a métodos Internos

Los métodos pueden llamar a otros métodos de la misma clase como parte de su implementación. Esto se denomina ***llamada a método interno***.

```
actualizarVisor();
```

Esta sentencia es una *llamada a un método*. Como hemos visto anteriormente, la clase `VisorDeReloj` tiene un método con la siguiente signatura:

```
private void actualizarVisor()
```

La llamada a método que mostramos en la línea anterior, justamente invoca a este método. Dado que este método está ubicado en la misma clase en que se produce su llamada, decimos que es una *llamada a un método interno*. Las llamadas a métodos internos tienen la siguiente sintaxis:

```
nombreDelMétodo (lista-de-parámetros)
```

3.11.2 Llamada a métodos externos

Los métodos pueden llamar a métodos de otros objetos usando la notación de punto: se denomina ***llamada a método externo***.

```
public void ticTac()
      {
            minutos.incrementar();
            if(minutos.getValor() == 0) {   // ¡alcanzó el
límite!
                  horas.incrementar();
            }
            actualizarVisor();
      }
```

Si este visor se conectara a un reloj real, este método sería invocado una vez cada 60 segundos por el temporizador electrónico del reloj. Por ahora, lo llamamos nosotros para probar el visor.

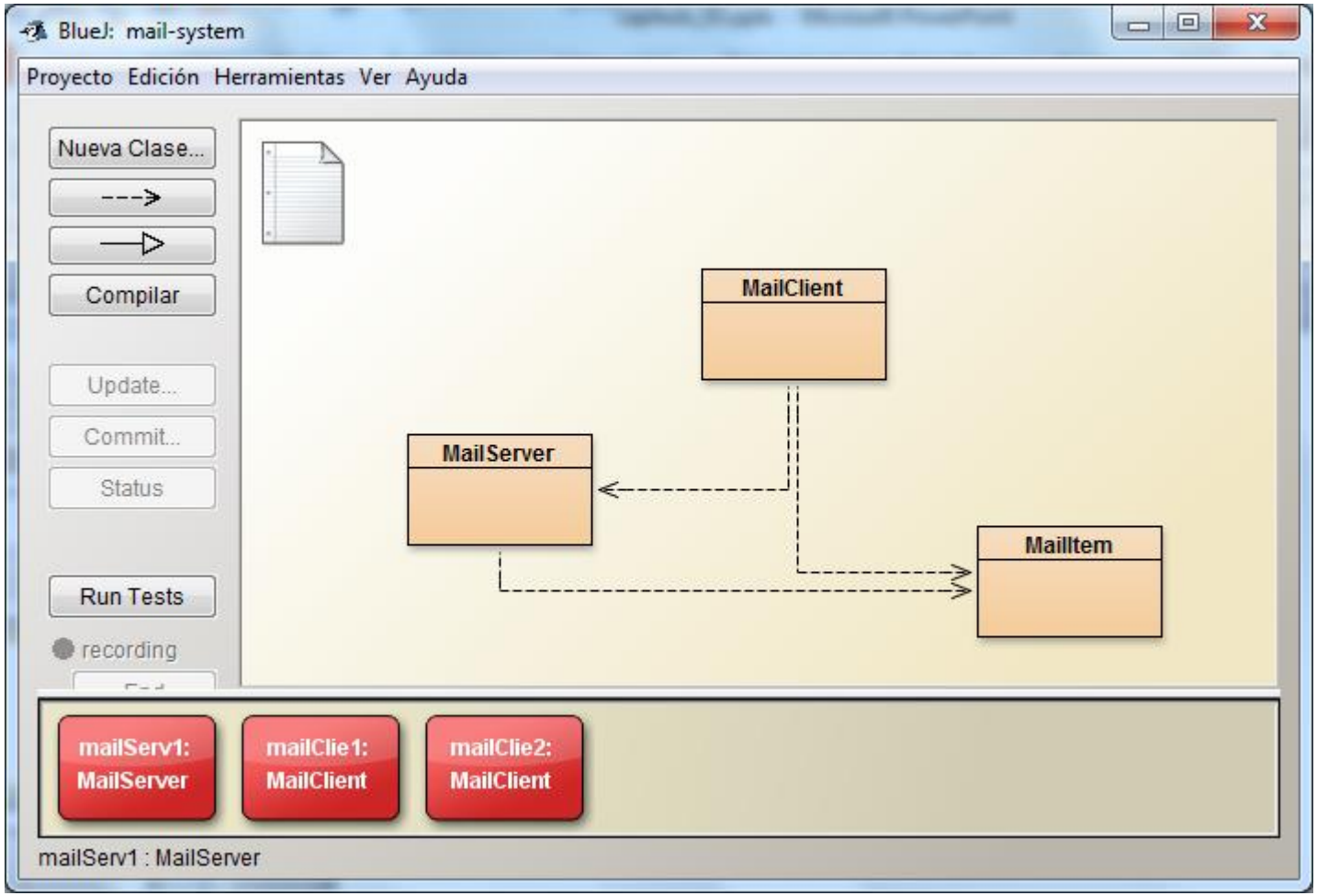
Cuando es llamado, el método `ticTac` ejecuta primero la sentencia

```
minutos.incrementar();
```

Esta sentencia llama al método `incrementar` del objeto `minutos`. Cuando se llama a uno de los métodos del objeto `VisorDeReloj`, este método a su vez llama a un método de otro objeto para colaborar en la tarea. Una llamada a método desde un método de otro objeto se conoce como *llamada a un método externo*. La sintaxis de una llamada a un método externo es

```
objeto.nombreDelMétodo (lista-de-parámetros)
```

Proyecto mail-system



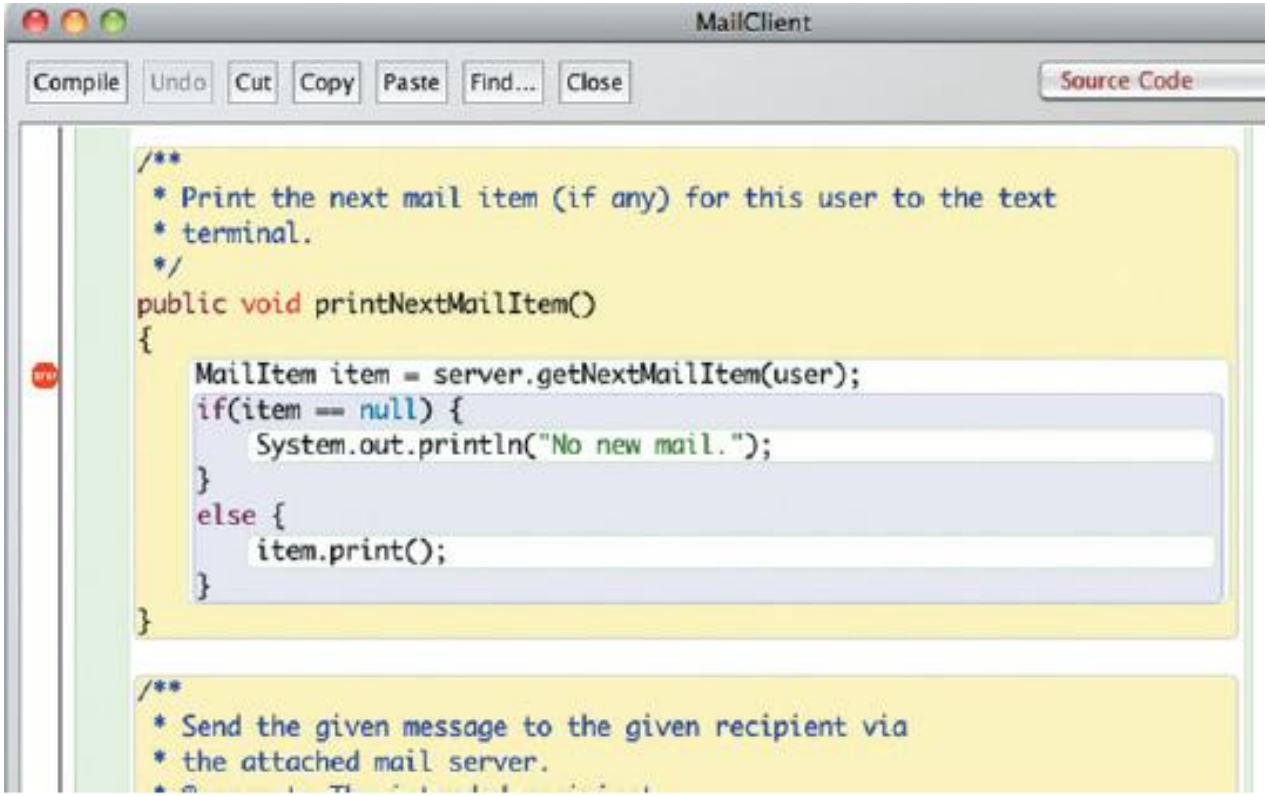
Palabra clave “this”

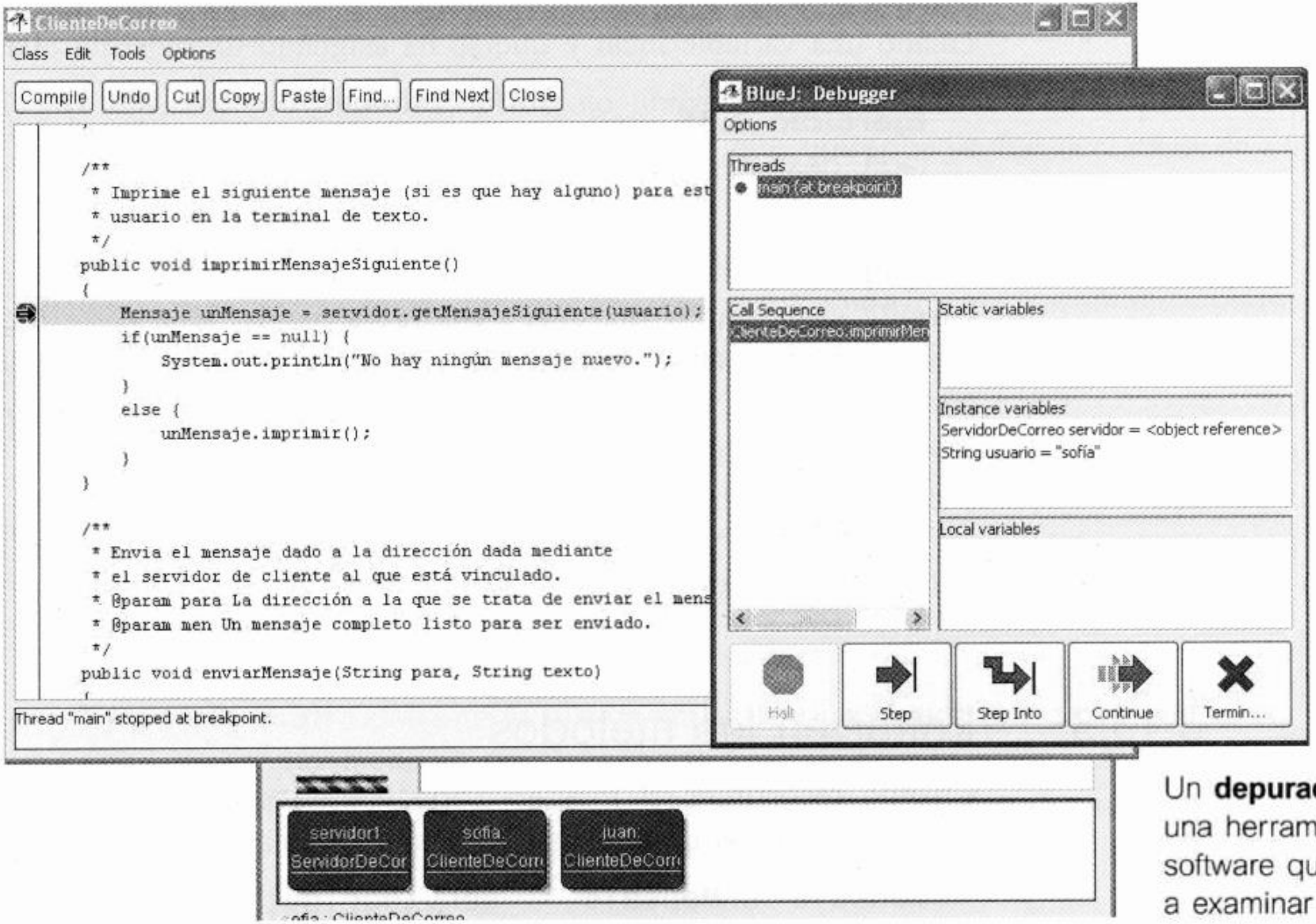
```
public class Mensaje
{
    // El remitente del mensaje.
    private String de;
    // El destinatario del mensaje.
    private String para;
    // El texto del mensaje.
    private String texto;
    /**
     * Crea un mensaje de correo del remitente para un
destinatario
     * dado, que contiene el texto especificado.
     * @param de      El remitente de este mensaje.
     * @param para    El destinatario de este mensaje.
     * @param texto El texto del mensaje que será enviado.
     */
    public Mensaje(String de, String para, String texto)
    {
        this.de = de;
        this.para = para;
        this.texto = texto;
    }
    Se omitieron los métodos.
}
```

`this.de = de;` *campo de nombre* “de” = *parámetro de nombre* “de”;

3.13 Uso del depurador

- Establecemos el punto de interrupción, pulsando sobre el número de la línea





Un **depurador** es una herramienta de software que ayuda a examinar cómo se ejecuta una aplicación. Puede usarse para encontrar problemas.

Glosario

- **abstracción** La abstracción es la habilidad de ignorar los detalles de las partes para enfocar la atención en un nivel más alto de un problema.
- **modularización** La modularización es el proceso de dividir una totalidad en partes bien definidas que podemos construir y examinar separadamente y que interactúan de maneras bien definidas.
- **las clases definen tipos** Puede usarse un nombre de clase para el tipo de una variable. Las variables que tienen una clase como su tipo pueden almacenar objetos de dicha clase.
- **diagrama de clases** Los diagramas de clases muestran las clases de una aplicación y las relaciones entre ellas. Dan información sobre el código. Representan la vista estática de un programa.
- **diagrama de objetos** Los diagramas de objetos muestran los objetos y sus relaciones en un momento dado, durante el tiempo de ejecución de una aplicación. Dan información sobre los objetos en tiempo de ejecución. Representan la vista dinámica de un programa.
- **referencias a objetos** Las variables de tipo objeto almacenan referencias a los objetos.

- **tipo primitivo** Los tipos primitivos en Java no son objetos. Los tipos `int`, `boolean`, `char`, `double` y `long` son los tipos primitivos más comunes. Los tipos primitivos no tienen métodos.
- **creación de objetos** Los objetos pueden crear otros objetos usando el operador `new`.
- **sobrecarga** Una clase puede contener más de un constructor o más de un método con el mismo nombre, siempre y cuando tengan un conjunto de tipos de parámetros que los distinga.
- **llamada a método interno** Los métodos pueden llamar a otros métodos de la misma clase como parte de su implementación. Esto se denomina llamada a método interno.
- **llamada a método externo** Los métodos pueden llamar a métodos de otros objetos usando la notación de punto. Esto se denomina llamada a método externo.
- **depurador** Un depurador es una herramienta de software que ayuda a examinar cómo se ejecuta una aplicación. Puede usarse para encontrar errores.