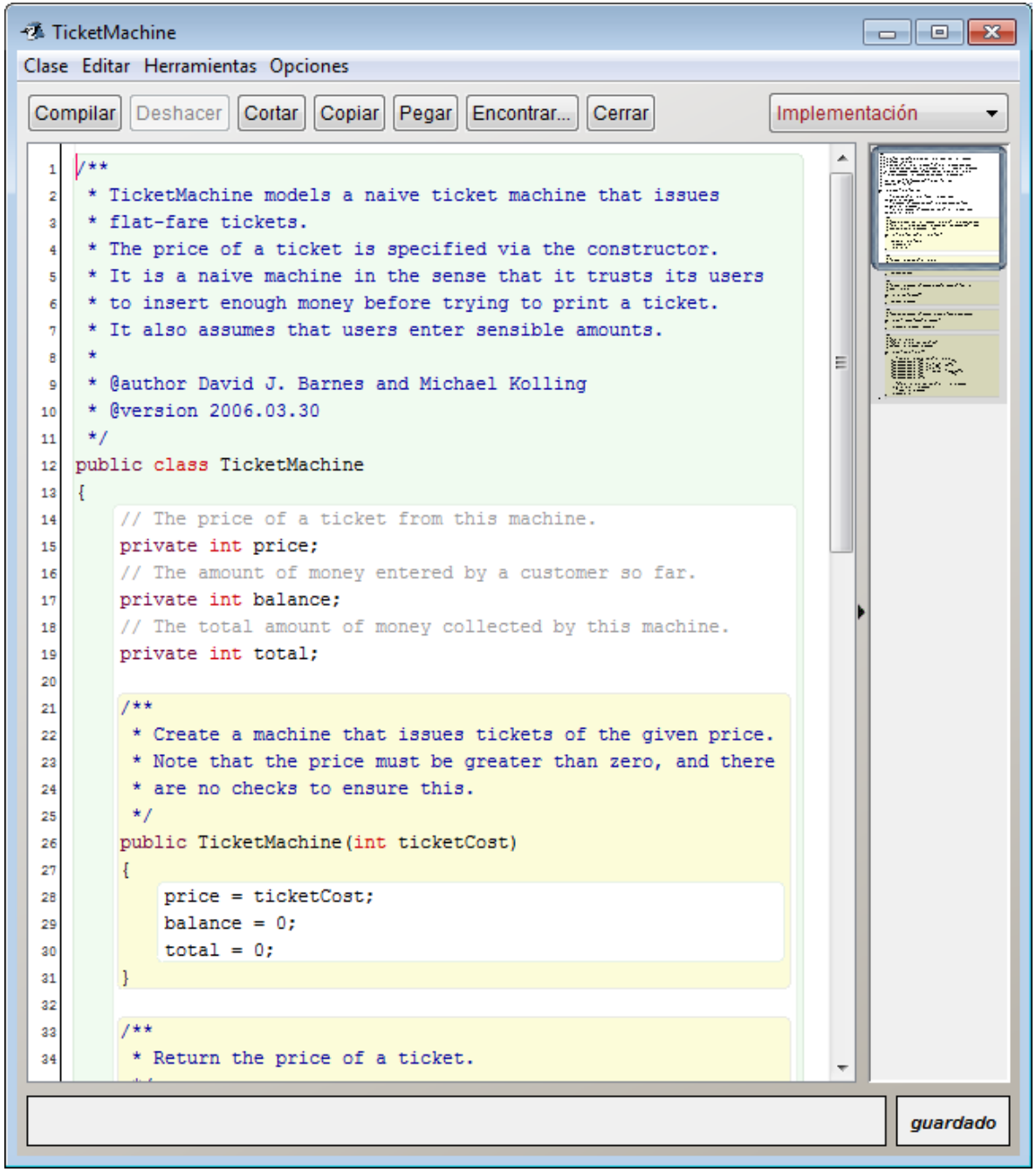
Programación orientada a objetos

Capítulo 2

Comprender las definiciones de clase

TEMA 5: Definición de clases. Semana 3	

1. El concepto de Clase..
2. Campos, constructores y métodos.
3. Paso de datos mediante parámetros.
4. Asignación de valores.
5. Tipos de métodos:
  1. Métodos de acceso: get()
  2. Métodos de modificación: set()
  3. Método main()
4. Impresión desde métodos.
5. Estructuras de control: la sentencia condicional if.
6. Campos, parámetros y variables locales.
	1. Estudiar el Capítulo 2 y leer los Apéndices B, C y D del libro base para la "Unidad Didáctica II".
	2. Leer los documentos accesibles a través de Curso virtual en el apartado "Clases y Objetos" del "Material de Estudio" de la "Unidad Didáctica II
	3. Realizar los ejercicios en el entorno BLUEJ sugeridos en el libro base.



2.1 Máquina expendedora

- Los clientes insertan el dinero y la máquina imprime billete
- Lleva el control del dinero acumulado

2.2 Examinar la definición de clase

Ver código furente del ejemplo MaquinaDeBoletos (naive-ticket-machine)

```
public class MaquinaDeBoletos
{
    Se omite la parte interna de la clase
}
```

2.4 Campos, constructores y métodos

```
public class NombreDeClase
{
Campos
Constructores
Métodos
}
```

Los campos almacenan los datos para que cada objeto los use

Los constructores permiten que cada objeto se configuren adecuadamente cuando es creado. Inicialización del objeto

Los métodos implementan el comportamiento de los objetos

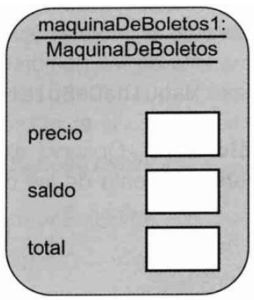
2.4.1 Campos

Código 2.3
Los campos de la clase `MaquinaDeBoletos`

Concepto

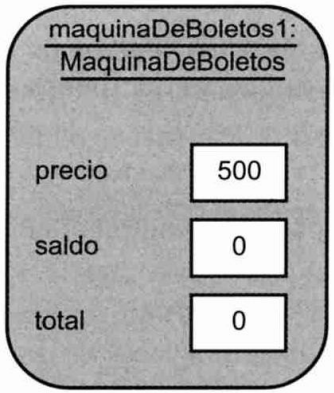
Los **campos** almacenan datos para que un objeto los use. Los campos también son conocidos como variables de instancia.

```
public class MaquinaDeBoletos
{
    private int precio;
    private int saldo;
    private int total;

    Se omitieron el constructor y los métodos.

}
```



2.4.2 Constructores

Los **constructores** permiten que cada objeto sea preparado adecuadamente cuando es creado.




```
public class MaquinaDeBoletos
{
Se omitieron los campos
          /**
           * Crea una máquina que vende boletos de un
             determinado precio.
           * Observe que el precio debe ser mayor que cero
             y que no hay
           * controles que aseguren esto.
           */
public MaquinaDeBoletos (int precioDelBoleto)
{
precio = precioDelBoleto;
saldo = 0;
total = 0;
}
   Se omitieron los métodos
}
```

Los **constructores** permiten que cada objeto sea preparado adecuadamente cuando es creado. Esta operación se denomina *inicialización*. El constructor inicializa el objeto en un estado razonable.

Uno de los rasgos distintivos de los constructores es que tienen el mismo nombre que la clase en la que son definidos.

Los campos del objeto se inicializan en el constructor, bien con valores fijos, o bien con parámetros del propio constructor.

Nota: en Java, todos los **campos** son inicializados automáticamente con un valor por defecto, si es que no están inicializados explícitamente. El valor por defecto para los **campos enteros** es 0. Sin embargo, es preferible escribir explícitamente las asignaciones. No hay ninguna desventaja en hacer esto y sirve para documentar lo que está ocurriendo realmente.

2.5 Pasar datos mediante parámetros

- Los constructores y los métodos reciben valores mediante los parámetros

Los parámetros se definen en el encabezado de un constructor o un método:

```
public MaquinaDeBoletos (int precioDelBoleto)
```

Distinguimos entre nombres de los parámetros dentro de un constructor o un método, y valores de los parámetros fuera de un constructor o un método: hacemos referencia a los nombres como *parámetros formales* y a los valores como *parámetros actuales*.

El **alcance** de una variable define la sección de código en la que la variable puede ser accedida.

Un parámetro formal está disponible para un objeto sólo dentro del cuerpo del constructor o del método que lo declara. Decimos que el *alcance* de un parámetro está restringido al cuerpo del constructor o del método en el que es declarado. En cambio, el alcance de un campo es toda la clase y puede ser accedido desde cualquier lugar en la misma clase.

El **tiempo de vida** de una variable describe cuánto tiempo continuará existiendo la variable antes de ser destruida.

Un concepto relacionado con el alcance de una variable es el *tiempo de vida* de la variable. El tiempo de vida de un parámetro se limita a una sola llamada de un constructor o método. Una vez que completó su tarea, los parámetros formales desaparecen y se pierden los valores que contienen. En otras palabras, cuando un constructor termina su ejecución, se elimina el espacio del constructor (*véase* Figura 2.4) junto con las variables parámetro que contiene.

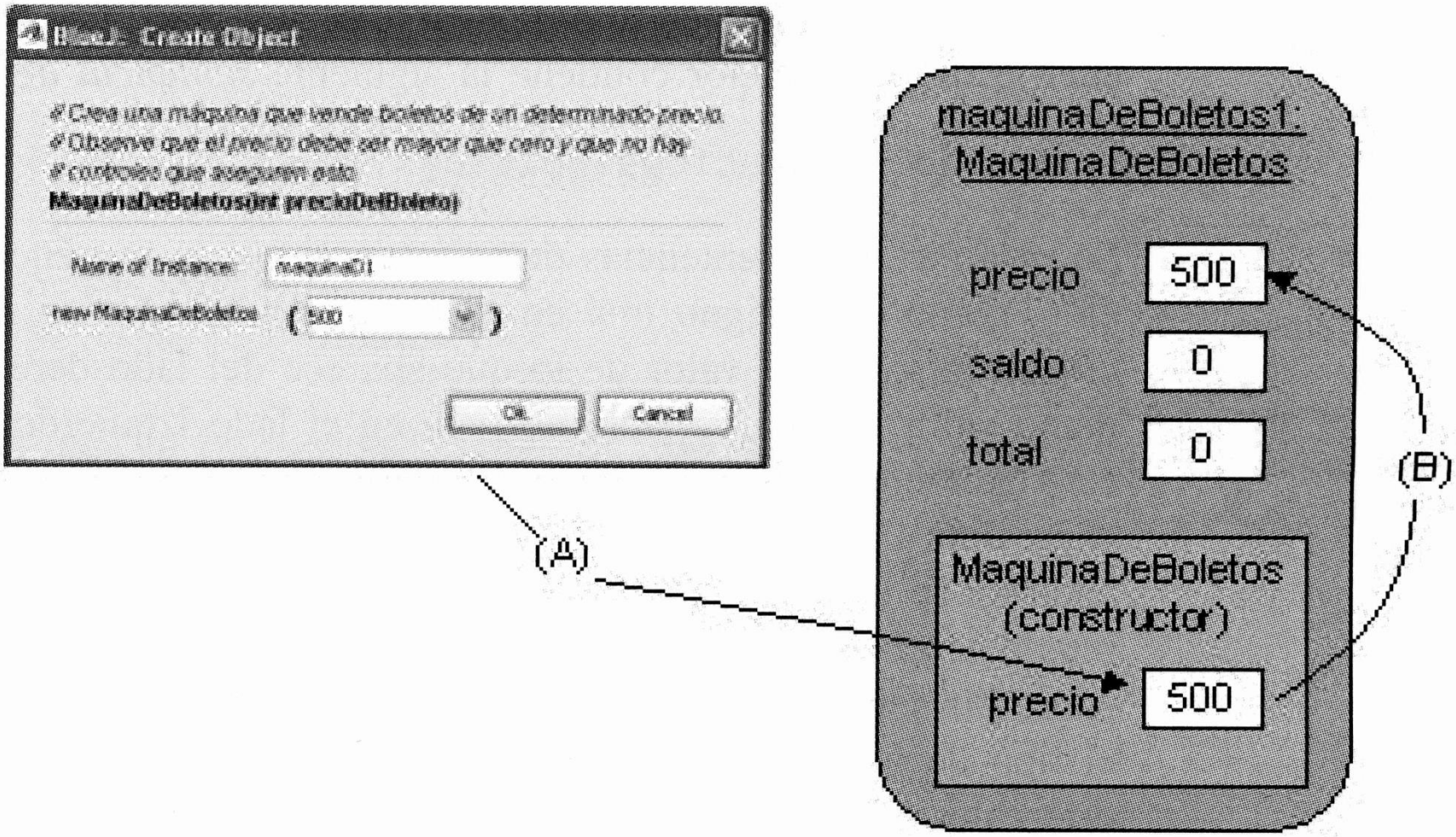
2.6 Asignación

Las **sentencias de asignación** almacenan el valor representado por el lado derecho de la sentencia en una variable nombrada a la izquierda.

```
precio = precioDelBoleto;
```

Ejercicio 2.20 *Desafío* ¿Cuál es el error en la siguiente versión del constructor de la clase `MaquinaDeBoletos`?

```
public MaquinaDeBoletos(int precioDelBoleto)
{
  int precio = precioDelBoleto;
  saldo = 0;
  total = 0;
}
```


Create Object
Crea una máquina que vende boletos de un determinado precio
Observe que el precio debe ser mayor que cero y que no hay
controles que aseguren esto
MaquinaDeBoletos(int precioDelBoleto)
Name of Instance
maquina01
new MaquinaDeBoletos
(500)
OK
Cancel
maquinaDeBoletos1:
MaquinaDeBoletos
precio
500
saldo
0
total
0
MaquinaDeBoletos
(constructor)
precio
500
(A)
(B)

Los **comentarios** se insertan en el código de una clase para proporcionar explicaciones a los lectores humanos. No tienen ningún efecto sobre la funcionalidad de la clase.

Comentarios

*Los **comentarios** se insertan en el código de una clase para proporcionar explicaciones a los lectores humanos. No tienen ningún efecto sobre la funcionalidad de la clase.*

Se introduce una sola línea de comentario mediante los dos caracteres «//».

Los comentarios más detallados, que frecuentemente ocupan varias líneas, se escriben generalmente en la forma de comentarios multilínea: comienzan con el par de caracteres «/*» y terminan con el par «*/».

Comentarios de Documentación:

```
/**
 * (descripción principal) objetivo general de la clase o método
 * (sección de etiqueta)
 */
```

Sección etiqueta:

Puede usarse de dos maneras:

*En bloques de etiquetas----------> la más importante.

**Etiquetas de sólo una linea.....(ver Javadoc, Tools an Utilities)*

Hay alrrededor de 20, pero las más importantes:

Comentarios de clase e interfaz:

@author
@version
@see

Comentarios de metodos y constructores:

@param
@throws
@see
@return (solo para métodos)

@see referencia cruzada:

Adopta varias formas:

**@see "The Java Language Specification" (encierra un texto en forma de cadena sin hipervículo)*

**@see <a href=http://www.bluej.org/> (hipervinculo)*

**@see #estaVivo (vinculo a la documentación del método estaVivo de la misma clase)*

**@see java.util.ArrayList#add (vincula la documentación del método add cn la clase java.util.ArrayList)*

2.7 Métodos de acceso

Los métodos implementan el comportamiento de los objetos

Los métodos se componen de dos partes: un encabezado y un cuerpo.

```
public class MaquinaDeBoletos
{
    Se omitieron los campos.
    Se omitieron los constructores.
/**
* Devuelve el precio de un boleto.
*/
public int obtenerPrecio()
{
    return precio;
}
    Se omitieron los restantes métodos.
}
```

Existen, por lo menos, dos diferencias significativas entre las signaturas del constructor `MaquinaDeBoletos` y del método `obtenerPrecio`:

```
public MaquinaDeBoletos (int precioDelBoleto)
public int obtenerPrecio()
```

- El método tiene un *tipo de retorno* `int` pero el constructor no tiene tipo de retorno. El tipo de retorno se escribe exactamente antes del nombre del método.
- El constructor tiene un solo parámetro formal, `precioDelBoleto`, pero el método no tiene ninguno, sólo un par de paréntesis vacíos.

En el cuerpo de `obtenerPrecio` hay una sola sentencia:

```
return precio;
```

Esta es una *sentencia return* y es la responsable de devolver un valor entero que coincida con el tipo de retorno `int` de la signatura del método. Cuando un método contiene una sentencia return, siempre es la última sentencia del mismo porque una vez que se ejecutó esta sentencia no se ejecutarán más sentencias en el método.

Los **métodos de acceso** devuelven información sobre el estado de un objeto.

2.8 Métodos selectores y mutadores

Los **métodos de modificación** cambian el estado de un objeto.

```
/**
* Recibe de un cliente una cantidad de dinero en centavos.
*/
public void ingresarDinero(int cantidad)
{
saldo = saldo + cantidad;
}
```

Nota: convenciones Java sobre métodos de acceso y de modificación7

En Java, los nombres de los métodos de acceso suelen comenzar con la palabra «get» en lugar de la palabra «obtener» y los nombres de los métodos de modificación, con la palabra «set» en lugar de «poner».

Por ejemplo:

`getPrecio`, `getSaldo` son métodos de acceso a las variables precio y saldo.

`setPrecio`, `setSaldo` son métodos de modificación de las variables precio y saldo.

De aquí en adelante, usaremos esta convención para los nombres de los métodos de modificación y de acceso.

Ejercicio 2.29 ¿Qué elementos del encabezado de `ponerPrecio` nos indican que es un método y no un constructor?

```
public void ponerPrecio (int precioDelBoleto)
```

2.9 Imprimir desde métodos

El método **System.out.println** imprime su parámetro en la terminal de texto.

```
/**
* Imprime un boleto y pone el saldo actual en cero
*/
public void imprimirBoleto()
{
// Simula la impresión de un boleto.
System.out.println("##################");
System.out.println("# Línea BlueJ");
System.out.println("# Boleto");
System.out.println("# " + precio + " cvos.");
System.out.println("##################");
System.out.println();
// Actualiza el total recaudado con el saldo.
total = total + saldo;
// Limpia el saldo.
saldo = 0;
}
```

Ejercicio 2.42

Ejercicio 2.42 Provea a la clase de dos constructores: uno debe tomar un solo parámetro que especifique el precio del boleto, y el otro no debe tener parámetros y debe establecer el precio como un valor fijo por defecto, el que usted elija. Pruebe su implementación creando máquinas mediante los dos constructores diferentes.

```
public TicketMachine()
{
    price = 1000;
    balance = 0;
    total = 0;
}

public TicketMachine(int ticketCost)
{
    price = ticketCost;
    balance = 0;
    total = 0;
}
```

2.12 Máquina mejorada

```
    /**
     * Recibe del cliente una cantidad de dinero en
centavos.
     * Controla que la cantidad tenga sentido.
     */
    public void ingresarDinero(int cantidad)
    {
        if(cantidad > 0) {
            saldo = saldo + cantidad;
        }
        else {
            System.out.println("Debe ingresar una cantidad
positiva: " +
                                                cantidad);
        }
    }
```

Ver código de Better-ticket-machine

2.13 Sentencia condicional

Una **sentencia condicional** realiza una de dos acciones posibles basándose en el resultado de una prueba.

Las **expresiones booleanas** tienen sólo dos valores posibles: verdadero o falso. Se las encuentra comúnmente controlando la elección entre los dos caminos posibles de una sentencia condicional.

```
if(se lleva a cabo alguna prueba que da un resultado
verdadero o falso) {
Si la prueba dio resultado verdadero, ejecutar estas sentencias
}
else {
Si el resultado dio falso, ejecutar estas sentencias
}
```

```
if(cantidad > 0) {
saldo = saldo + cantidad;
}
else {
System.out.println("Debe ingresar una cantidad positiva: " +
cantidad);
}
```

2.14 Ejemplo avanzado de sentencia condicional

El método `imprimirBoleto` contiene un ejemplo más avanzado de una sentencia condicional. Aquí está su esquema:

```
if(saldo >= precio) {
Se omitieron los detalles de impresión.
// Actualiza el total recaudado con el precio.
total = total + precio;
// Decrementa el saldo en el valor del precio.
saldo = saldo - precio;
}
       else {
            System.out.println("Debe ingresar como mínimo: "
                                    + (precio - saldo) + "
cvos más.");
       }
```

2.15 Representación visual del ámbito

```
/**
 * TicketMachine models a ticket machine that issues
 * flat-fare tickets.
 * The price of a ticket is specified via the constructor.
 * Instances will check to ensure that a user only enters
 * sensible amounts of money, and will only print a ticket
 * if enough money has been input.
 *
 * @author David J. Barnes and Michael Kölling
 * @version 2011.07.31
 */
public class TicketMachine
{
    // The price of a ticket from this machine.
    private int price;
    // The amount of money entered by a customer so far.
    private int balance;
    // The total amount of money collected by this machine.
    private int total;

    /**
     * Create a machine that issues tickets of the given price.
     */
    public TicketMachine(int cost)
    {
        price = cost;
        balance = 0;
        total = 0;
    }

    /**
     * @Return The price of a ticket.
     */
```

2.16 Variables locales

Una **variable local** es una variable que se declara y se usa dentro de un solo método. Su alcance y tiempo de vida se limitan a los del método.

El método `reintegrarSaldo` contiene tres sentencias y una declaración. La declaración ilustra una nueva clase de variable:

```
public int reintegrarSaldo()
{
int cantidadAReintegrar;
cantidadAReintegrar = saldo;
saldo = 0;
return cantidadAReintegrar;
}
```

```
int cantidadAReintegrar = saldo;
```

2.17 Campos, parámetros y variables locales

- Las tres clases de variables pueden almacenar un valor acorde a su definición de tipo de dato. Por ejemplo, una variable definida como de tipo `int` permite almacenar un valor entero.
- Los campos se definen fuera de los constructores y de los métodos.
- Los campos se usan para almacenar datos que persisten durante la vida del objeto, de esta manera mantienen el estado actual de un objeto. Tienen un tiempo de vida que finaliza cuando termina el objeto.
- El alcance de los campos es la clase: la accesibilidad de los campos se extiende a toda la clase y por este motivo pueden usarse dentro de cualquier constructor o método de clase en la que estén definidos.
- Como son definidos como privados (`private`), los campos no pueden ser accedidos desde el exterior de la clase.

Campos, parámetros y variables locales

- Los parámetros formales y las variables locales persisten solamente en el lapso durante el cual se ejecuta un constructor o un método. Su tiempo de vida es tan largo como una llamada, por lo que sus valores se pierden entre llamadas. Por este motivo, actúan como lugares de almacenamiento temporales antes que permanentes.
- Los parámetros formales se definen en el encabezado de un constructor o de un método. Reciben sus valores desde el exterior, se inicializan con los valores de los parámetros actuales que forman parte de la llamada al constructor o al método.
- Los parámetros formales tienen un alcance limitado a su definición de constructor o de método.
- Las variables locales se declaran dentro del cuerpo de un constructor o de un método. Pueden ser inicializadas y usadas solamente dentro del cuerpo de las definiciones de constructores o métodos. Las variables locales deben ser inicializadas antes de ser usadas en una expresión, no tienen un valor por defecto.
- Las variables locales tienen un alcance limitado al bloque en el que son declaradas. No son accesibles desde ningún lugar fuera de ese bloque.

2.20 Revisión de lab-classes

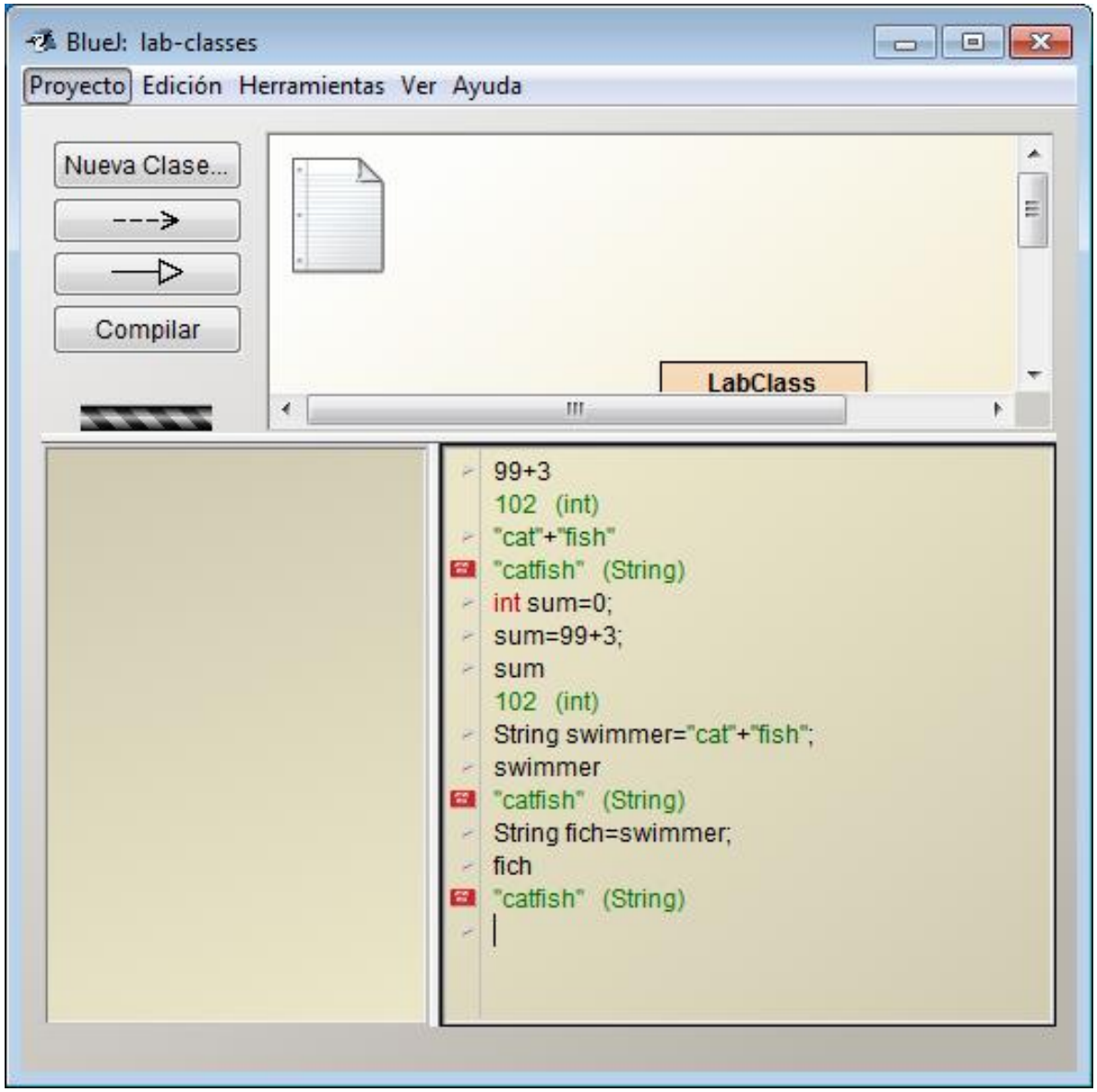
- Ver código de lab-classes (tema 01)

```
/**
 * The Student class represents a student in a student administration system.
 * It holds the student details relevant in our context.
 *
 * @author Michael Kölling and David Barnes
 * @version 2011.07.31
 */
public class Student
{
    // the student's full name
    private String name;
    // the student ID
    private String id;
    // the amount of credits for study taken so far
    private int credits;

    /**
     * Create a new student with a given name and ID number.
     */
    public Student(String fullName, String studentID)
    {
        name = fullName;
        id = studentID;
        credits = 0;
    }

    /**
     * Return the full name of this student.
     */
    public String getName()
    {
        return name;
    }

    /**
     * Set a new name for this student.
     */
    public void changeName(String replacementName)
    {
        name = replacementName;
    }
```

2.21 Invocación de métodos

```
/**
    * Return the login name of this student. The
login name is a combination
    * of the first four characters of the student's
name and the first three
    * characters of the student's ID number.
    */
   public String getLoginName()
   {
      return name.substring(0,4) +
id.substring(0,3);
   }
```

2.22 Experimentación con expresiones: Code Pad



Resumen de conceptos

- **campo** Los campos almacenan datos para que un objeto los use. Los campos se conocen como variables de instancia.
- **comentario** Los comentarios se insertan dentro del código de una clase para brindar explicaciones a los lectores humanos. No tienen efecto sobre la funcionalidad de la clase.
- **constructor** Los constructores permiten que cada objeto sea preparado adecuadamente cuando es creado.
- **alcance** El alcance de una variable define la sección de código desde donde la variable puede ser accedida.
- **tiempo de vida** El tiempo de vida de una variable describe el tiempo durante el cual la variable continúa existiendo antes de ser destruida.
- **asignación** Las sentencias de asignación almacenan el valor representado del lado derecho de la sentencia en la variable nombrada en el lado izquierdo.

Resumen de conceptos

- **método** Los métodos están compuestos por dos partes: un encabezado y un cuerpo.
- **método de acceso** Los métodos de acceso devuelven información sobre el estado de un objeto.
- **métodos de modificación** Los métodos de modificación cambian el estado de un objeto.
- **println** El método `System.out.println(...)` imprime su parámetro en la terminal de texto.
- **condicional** Una sentencia condicional realiza una de dos acciones posibles basándose en el resultado de una prueba.
- **expresión booleana** Las expresiones booleanas tienen sólo dos valores posibles: verdadero y falso. Se las encuentra comúnmente controlando la elección entre los dos caminos de una sentencia condicional.
- **variable local** Las variables locales son variables que se declaran y usan dentro de un único método. Su alcance y tiempo de vida están limitados por el método.