PROGRAMACIÓN ORIENTADA A OBJETOS

Índice de contenido

1.

Clases.

1. Clases.

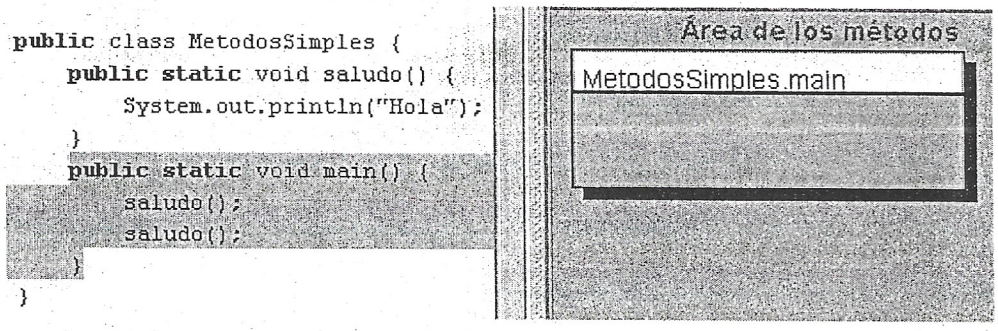
1.1. Métodos Simples.

1.1.1. Código.

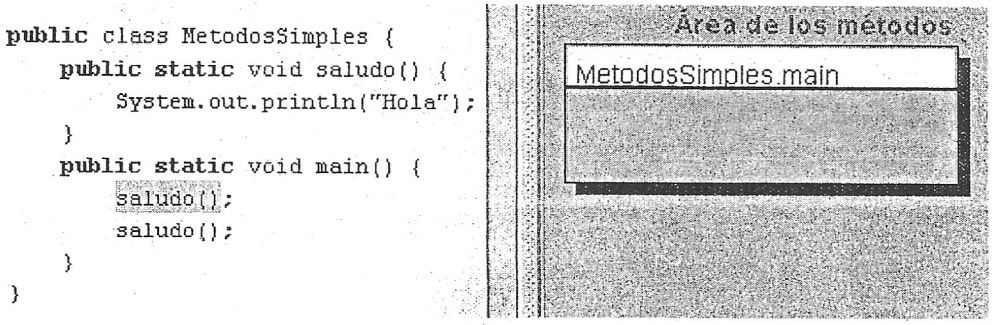
```
1 public class MetodosSimples {
2     public static void saludo() {
3         System.out.println("Hola");
4     }
5     public static void main() {
6         saludo();
7         saludo();
8     }
9 }
```

1.1.2. Ejecución.

1 Se crea un registro de activación para main.

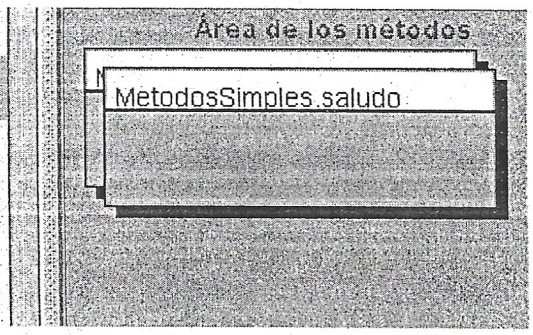


2 Se invoca el método saludo.



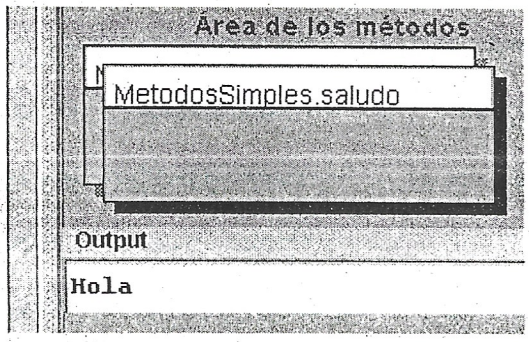
2.1 Se crea un registro de activación para saludo por encima del registro de activación para main.

```
public class MetodosSimples {
    public static void saludo() {
        System.out.println("Hola");
    }
    public static void main() {
        saludo();
        saludo();
    }
}
```



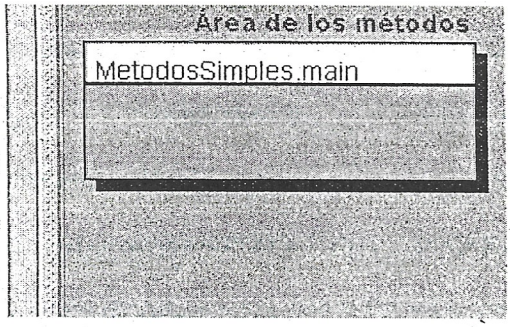
2.2 Muestra el mensaje "Hola" y hace un salto de línea.

```
public class MetodosSimples {
    public static void saludo() {
        System.out.println("Hola");
    }
    public static void main() {
        saludo();
        saludo();
    }
}
```



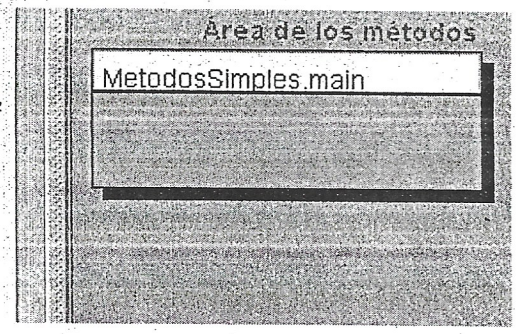
2.3 Elimina el registro de activación para saludo y regresa a main.

```
public class MetodosSimples {
    public static void saludo() {
        System.out.println("Hola");
    }
    public static void main() {
        saludo();
        saludo();
    }
}
```



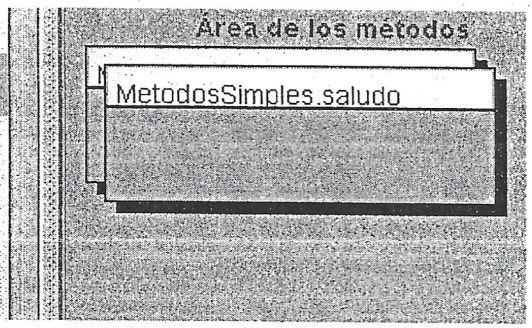
3 Se ejecuta la siguiente invocación de saludo.

```
public class MetodosSimples {
    public static void saludo() {
        System.out.println("Hola");
    }
    public static void main() {
        saludo();
        saludo();
    }
}
```



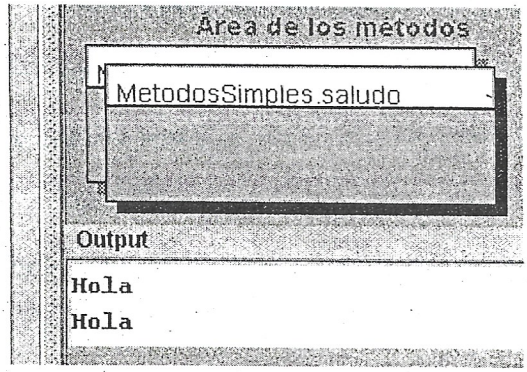
3.1 Se crea un registro de activación para saludo.

```
public class MetodosSimples {
    public static void saludo() {
        System.out.println("Hola");
    }
    public static void main() {
        saludo();
        saludo();
    }
}
```



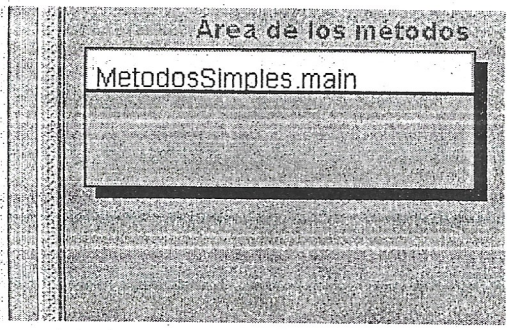
3.2 Muestra "Hola" y realiza un salto de línea.

```
public class MetodosSimples {
    public static void saludo() {
        System.out.println("Hola");
    }
    public static void main() {
        saludo();
        saludo();
    }
}
```



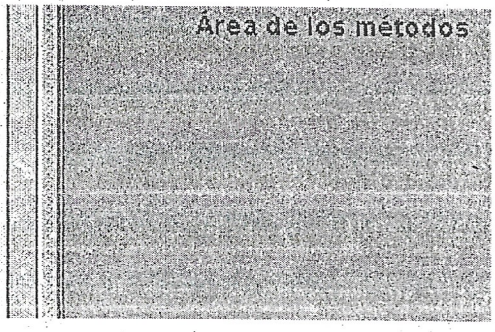
3.3 Elimina el registro de activación para saludo y regresa a main.

```
public class MetodosSimples {
    public static void saludo() {
        System.out.println("Hola");
    }
    public static void main() {
        saludo();
        saludo();
    }
}
```



4 Elimina el registro de activación para main y termina el programa.

```
public class MetodosSimples {
    public static void saludo() {
        System.out.println("Hola");
    }
    public static void main() {
        saludo();
        saludo();
    }
}
```



1.1.3. Comentarios.

Cuando un programa es muy largo se hace difícil de comprender. Para solucionar este problema, las instrucciones se pueden dividir en varios métodos cuyo nombre indique claramente lo que hacen.

Los métodos también se emplean cuando hay instrucciones que se repiten constantemente a lo largo del programa. Su uso es similar al mostrado en el ejemplo.

1.2. Anidación de Métodos.

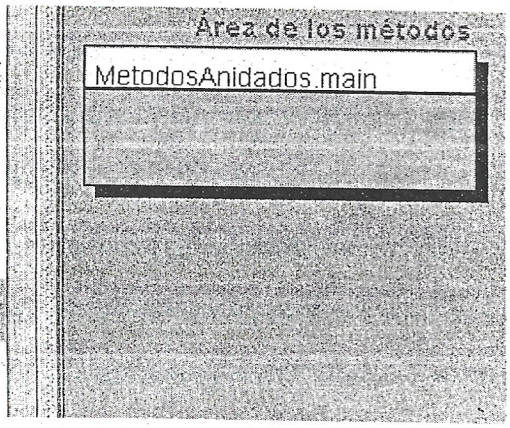
1.2.1. Código.

```
1  public class MetodosAnidados {
2      public static void despedida() {
3          cool();
4      }
5      public static void cool() {
6          String a = "Cool";
7          System.out.println(a);
8      }
9      public static void main() {
10         despedida();
11     }
12 }
```

1.2.2. Ejecución.

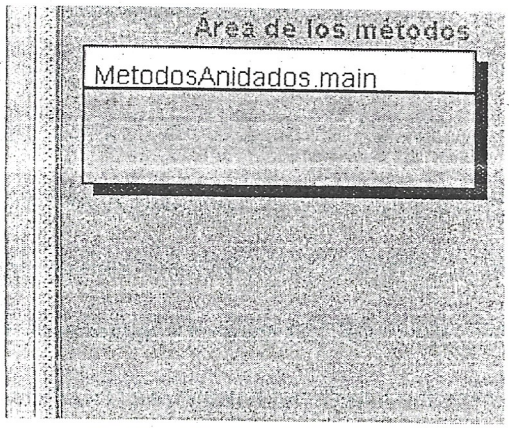
1 Se crea el registro de activación para main.

```
public class MetodosAnidados {
    public static void despedida() {
        cool();
    }
    public static void cool() {
        String a = "Cool";
        System.out.println(a);
    }
    public static void main() {
        despedida();
    }
}
```



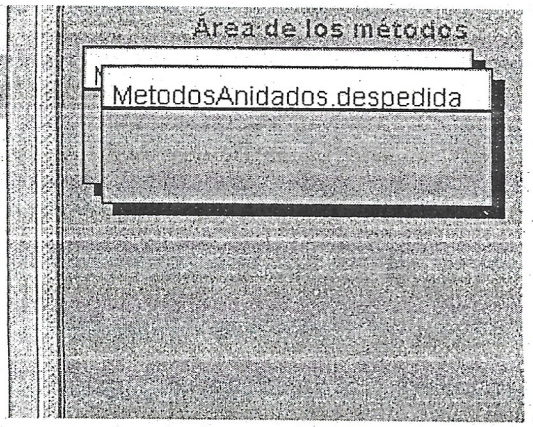
2 Se invoca el método despedida.

```
public class MetodosAnidados {
    public static void despedida() {
        cool();
    }
    public static void cool() {
        String a = "Cool";
        System.out.println(a);
    }
    public static void main() {
        despedida();
    }
}
```



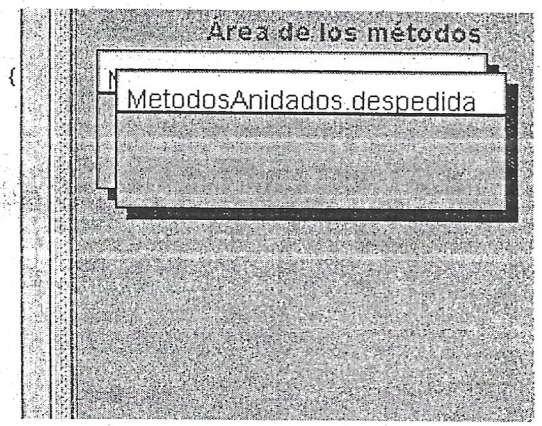
2.1 Se crea un registro de activación para despedida sobre el de main.

```
public class MetodosAnidados {
    public static void despedida() {
        cool();
    }
    public static void cool() {
        String a = "Cool";
        System.out.println(a);
    }
    public static void main() {
        despedida();
    }
}
```



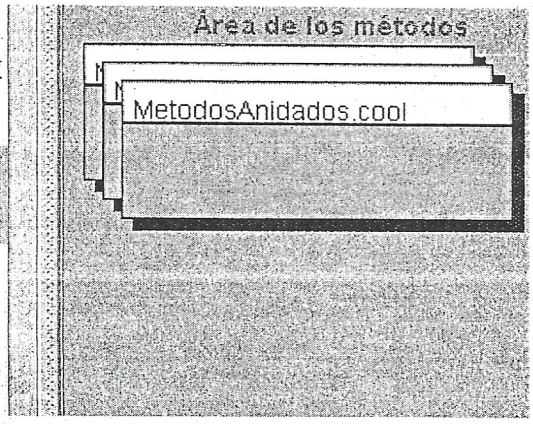
2.2 Invoca el método cool.

```
public class MetodosAnidados {
    public static void despedida() {
        cool();
    }
    public static void cool() {
        String a = "Cool";
        System.out.println(a);
    }
    public static void main() {
        despedida();
    }
}
```



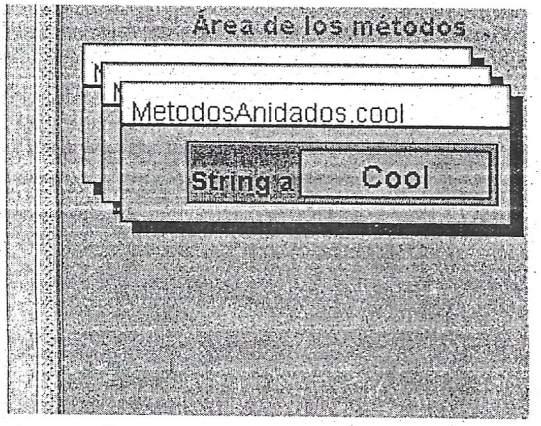
2.2.1 Se crea un registro de activación para cool sobre el de despedida.

```
public class MetodosAnidados {
    public static void despedida() {
        cool();
    }
    public static void cool() {
        String a = "Cool";
        System.out.println(a);
    }
    public static void main() {
        despedida();
    }
}
```



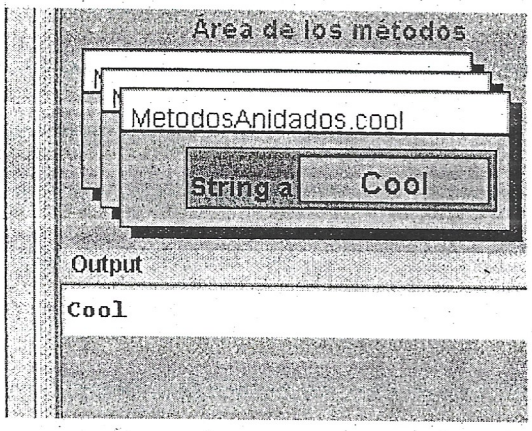
2.2.2 Se crea la variable "a" de tipo "String" con el valor "String" en el registro de activación de cool.

```
public class MetodosAnidados {
    public static void despedida() {
        cool();
    }
    public static void cool() {
        String a = "Cool";
        System.out.println(a);
    }
    public static void main() {
        despedida();
    }
}
```



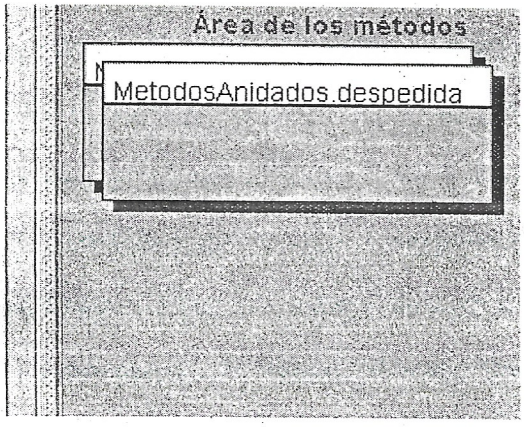
2.2.3 Se imprime el valor de "a", o sea "Cool" y realiza un salto de línea.

```
public class MetodosAnidados {
    public static void despedida() {
        cool();
    }
    public static void cool() {
        String a = "Cool";
        System.out.println(a);
    }
    public static void main() {
        despedida();
    }
}
```



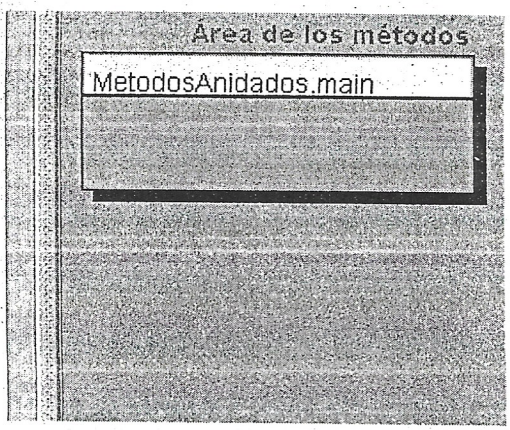
2.2.4 Se elimina el registro de activación junto con sus variables y regresa al registro de activación que esté hasta arriba, o sea despedida.

```
public class MetodosAnidados {
    public static void despedida() {
        cool();
    }
    public static void cool() {
        String a = "Cool";
        System.out.println(a);
    }
    public static void main() {
        despedida();
    }
}
```



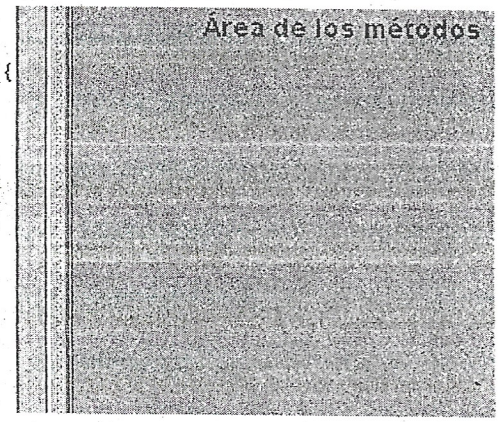
2.3 Se elimina el registro de activación para despedida y regresa a main.

```
public class MetodosAnidados {
    public static void despedida() {
        cool();
    }
    public static void cool() {
        String a = "Cool";
        System.out.println(a);
    }
    public static void main() {
        despedida();
    }
}
```



3 Se elimina el registro de activación para main y termina el programa.

```
public class MetodosAnidados {
    public static void despedida() {
        cool();
    }
    public static void cool() {
        String a = "Cool";
        System.out.println(a);
    }
    public static void main() {
        despedida();
    }
}
```



1.2.2. Comentarios.

Los métodos a su vez pueden invocar a otros métodos. Para controlar la invocación de estos, la computadora utiliza el área de métodos apilando los registros de activación uno sobre el otro. Cuando termina una invocación, se destruye el registro de activación que está hasta arriba. Continúa entonces la ejecución con el registro que quede más arriba, como si fuera una pila de platos. Por esta razón el área de métodos se conoce comunmente con el nombre de **stack**, que es la palabra en inglés para aquellas cosas que se comportan como una pila de platos.

Cuando un registro de activación se elimina, también se destruyen las variables que tenga definidas.

Los métodos pueden invocarse a sí mismos. Este comportamiento se conoce como **recursividad**. Para cada invocación se crea otro registro de activación con sus propias variables, independientes del que invoca. El diseño de métodos recursivos debe realizarse con cuidado, pues un error puede llevar a un ciclo sin fin de invocaciones con muchos registros de activación sobre otros, hasta agotar la memoria asignada al área de métodos.

Para coordinar el comportamiento de los métodos en ocasiones se requiere intercambiar información. A continuación se muestran dos mecanismos que nos permiten hacerlo: paso de parámetros y atributos de clase.

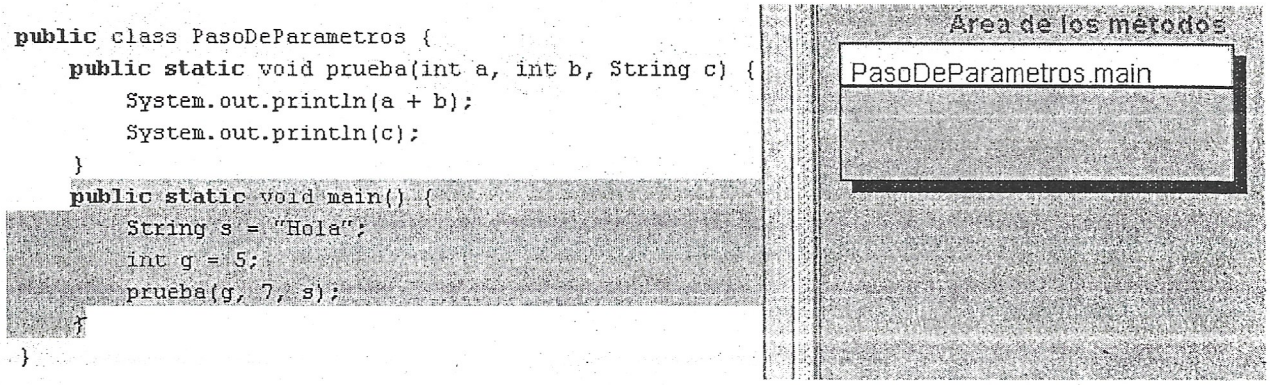
1.3. Paso de Parámetros.

1.3.1. Código.

```
 1 public class PasoDeParametros {
 2     public static void prueba(int a, int b, String c) {
 3         System.out.println(a + b);
 4         System.out.println(c);
 5     }
 6     public static void main() {
 7         String s = "Hola";
 8         int g = 5;
 9         prueba(g, 7, s);
10     }
11 }
```

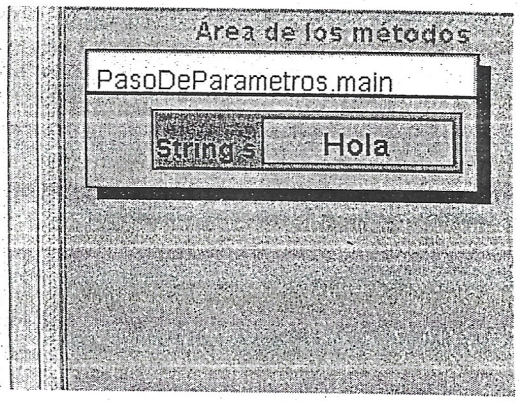
1.3.2. Ejecución.

1 Se crea un registro de activación para main.



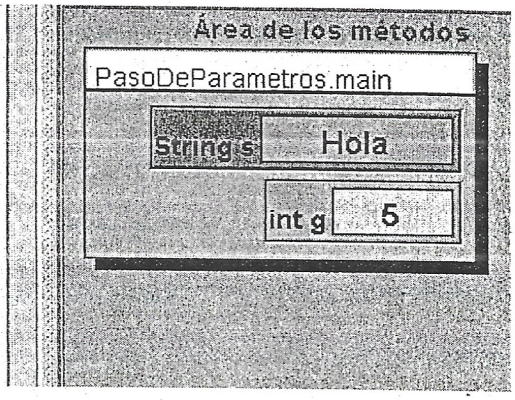
2 Crea la variable "s" de tipo "String" con el valor "Hola".

```
public class PasoDeParametros {
    public static void prueba(int a, int b, String c) {
        System.out.println(a + b);
        System.out.println(c);
    }
    public static void main() {
        String s = "Hola";
        int g = 5;
        prueba(g, 7, s);
    }
}
```



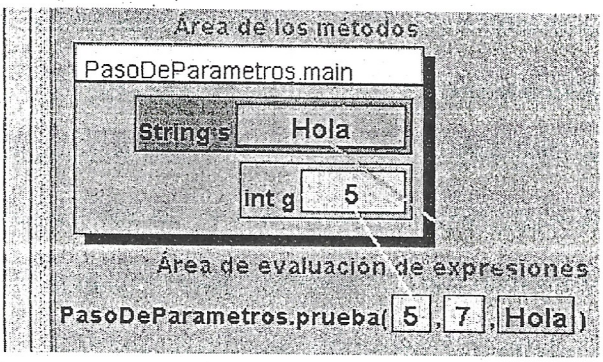
3 Crea la variable entera "g" con el valor "5".

```
public class PasoDeParametros {
    public static void prueba(int a, int b, String c) {
        System.out.println(a + b);
        System.out.println(c);
    }
    public static void main() {
        String s = "Hola";
        int g = 5;
        prueba(g, 7, s);
    }
}
```



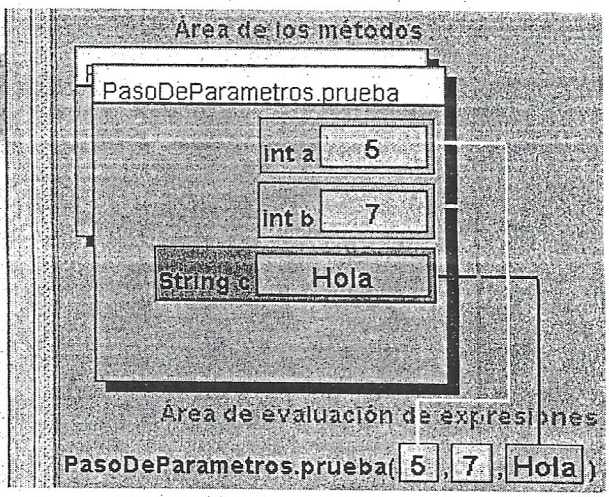
4 Se invoca el método prueba y se envían los siguientes valores: el valor de "g", que es "5", 7 y el valor de "s", o sea "Hola".

```
public class PasoDeParametros {
    public static void prueba(int a, int b, String c) {
        System.out.println(a + b);
        System.out.println(c);
    }
    public static void main() {
        String s = "Hola";
        int g = 5;
        prueba(g, 7, s);
    }
}
```



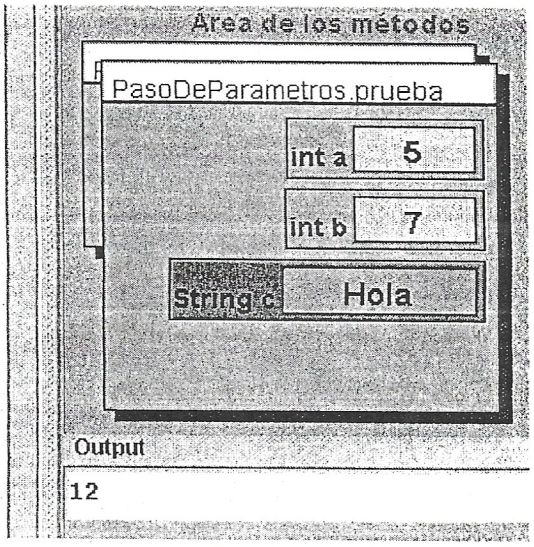
4.1 Se crea un registro de activación para prueba que contiene los parámetros declarados. Se le asignan, de izquierda a derecha los valores que se enviaron al invocar. Los tipos de los parámetros deben coincidir en la invocación y en la declaración.

```
public class PasoDeParametros {
    public static void prueba(int a, int b, String c) {
        System.out.println(a + b);
        System.out.println(c);
    }
    public static void main() {
        String s = "Hola";
        int g = 5;
        prueba(g, 7, s);
    }
}
```



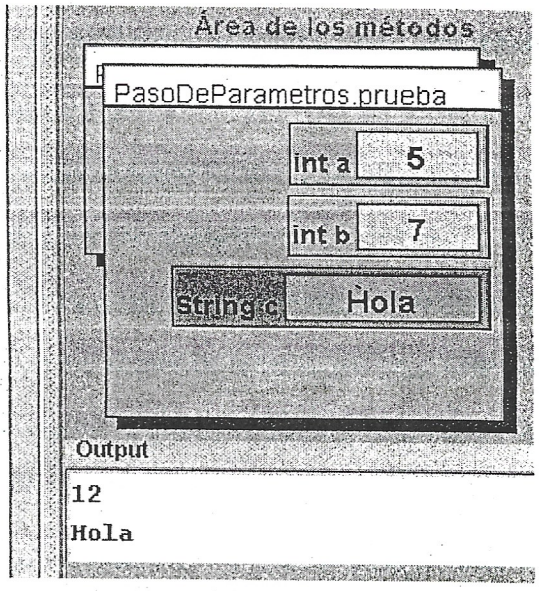
4.2 Se muestra el resultado de la expresión (a + b), o sea 12 y realiza un salto de línea.

```
public class PasoDeParametros {
    public static void prueba(int a, int b, String c) {
        System.out.println(a + b);
        System.out.println(c);
    }
    public static void main() {
        String s = "Hola";
        int g = 5;
        prueba(g, 7, s);
    }
}
```



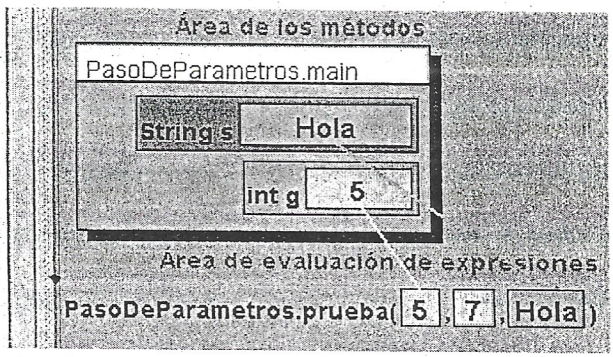
4.3 Se muestra el resultado de la expresión (a + b), o sea 12 y realiza un salto de línea.

```
public class PasoDeParametros {
    public static void prueba(int a, int b, String c) {
        System.out.println(a + b);
        System.out.println(c);
    }
    public static void main() {
        String s = "Hola";
        int g = 5;
        prueba(g, 7, s);
    }
}
```



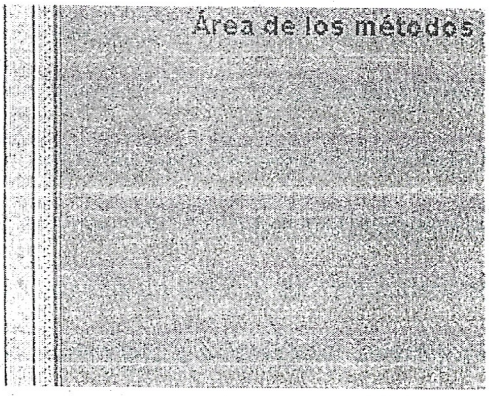
4.4 Se elimina el registro de activación de prueba y regresa a main.

```
public class PasoDeParametros {
    public static void prueba(int a, int b, String c) {
        System.out.println(a + b);
        System.out.println(c);
    }
    public static void main() {
        String s = "Hola";
        int g = 5;
        prueba(g, 7, s);
    }
}
```



5 Se elimina el registro de activación para main y termina el programa.

```
public class PasoDeParametros {
    public static void prueba(int a, int b, String c) {
        System.out.println(a + b);
        System.out.println(c);
    }
    public static void main() {
        String s = "Hola";
        int g = 5;
        prueba(g, 7, s);
    }
}
```



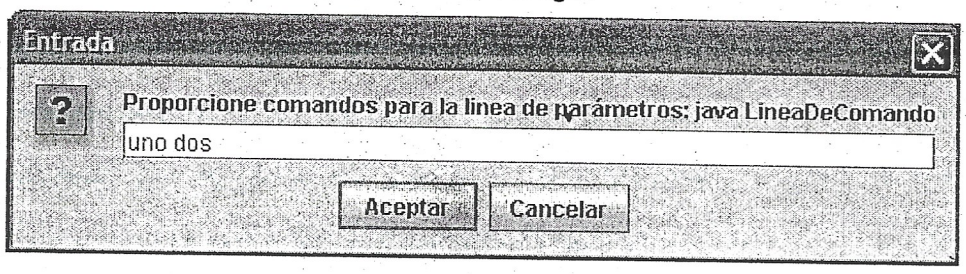
1.4. Línea de Parámetros.

1.4.1. Código.

```
1 public class LineaDeComando {
2     public static void main(String[] args) {
3         for (int i = 0; i < args.length; i++) {
4             System.out.println(args[i]);
5         }
6     }
7 }
```

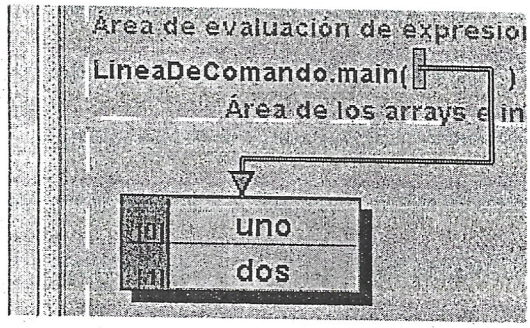
1.4.2. Ejecución.

1. Se toman los parámetros de la línea de comando del sistema operativo. En este ejemplo, se introducen en un cuadro de diálogo.



2. Invoca a main y se le envía un arreglo que contenga los parámetros.

```
public class LineaDeComando {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }
    }
}
```

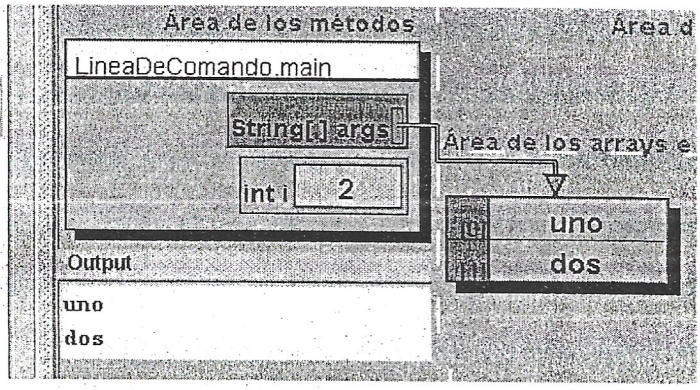


3. Crea el registro de activación para main y crea args, que es la referencia al arreglo recibido.

```
public class LineaDeComando {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }
    }
}
```

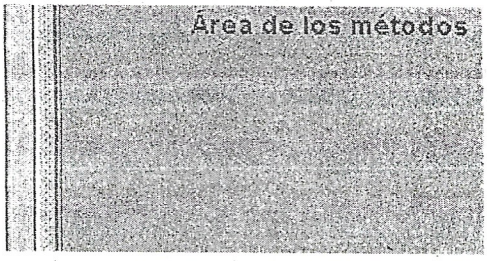
4. El ciclo for imprime los valores contenidos en args.

```
public class LineaDeComando {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }
    }
}
```



5. Se elimina el registro de activación, el arreglo y termina el programa.

```
public class LineaDeComando {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }
    }
}
```



1.4.3. Comentarios.

El parámetro args es utilizado por el sistema operativo para enviar información al programa. Por ejemplo, en el caso de los procesadores de texto se recibe el nombre del archivo a editar.

Cuando ejecutes programas fuera de Jeliot debes declarar el parámetro args en el método main.

1.5. Valor de Retorno.

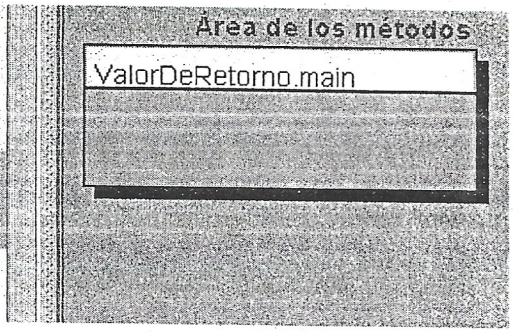
1.5.1. Código.

```
1 public class ValorDeRetorno {
2     public static int suma(int op1, int op2) {
3         return (op1 + op2);
4     }
5     public static void main() {
6         int g = suma(7, 3);
7         System.out.println(g);
8     }
9 }
```

1.5.2. Ejecución.

1 Se crea un registro de activación para main.

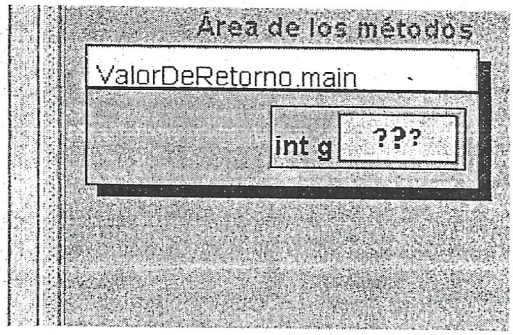
```
public class ValorDeRetorno {
    public static int suma(int op1, int op2) {
        return (op1 + op2);
    }
    public static void main() {
        int g = suma(7, 3);
        System.out.println(g);
    }
}
```



2 Ejecuta el método suma enviando los parámetros 7 y 3. El resultado se guardará en la variable "g".

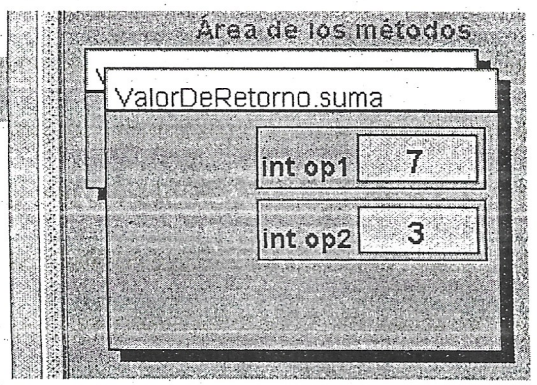
2.1 Se crea la variable "g" sin asignarle valor.

```
public class ValorDeRetorno {
    public static int suma(int op1, int op2) {
        return (op1 + op2);
    }
    public static void main() {
        int g = suma(7, 3);
        System.out.println(g);
    }
}
```



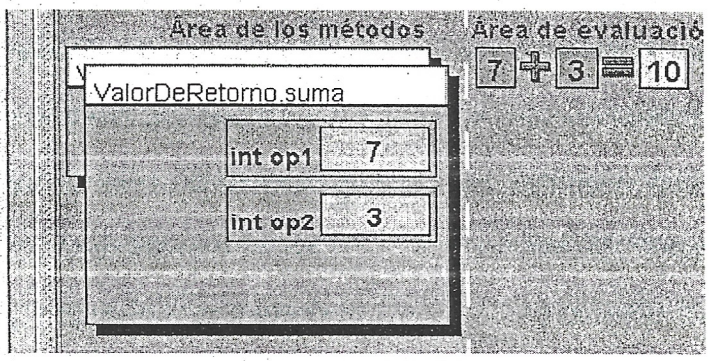
2.2 Se crea el registro de activación para suma; se le envía "7" al parámetro "op1" y "3" al parámetro "op2".

```
public class ValorDeRetorno {
    public static int suma(int op1, int op2) {
        return (op1 + op2);
    }
    public static void main() {
        int g = suma(7, 3);
        System.out.println(g);
    }
}
```



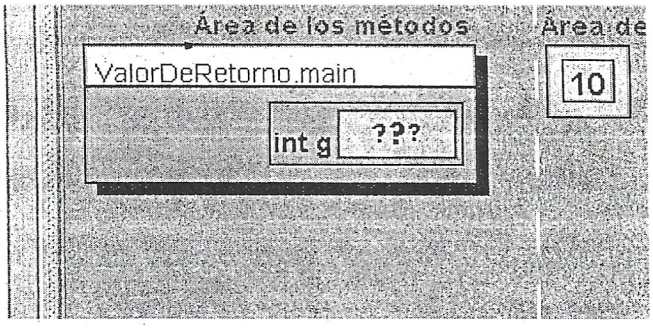
2.3 Realiza la operación (op1 + op2) obteniendo 10.

```
public class ValorDeRetorno {
    public static int suma(int op1, int op2) {
        return (op1 + op2);
    }
    public static void main() {
        int g = suma(7, 3);
        System.out.println(g);
    }
}
```



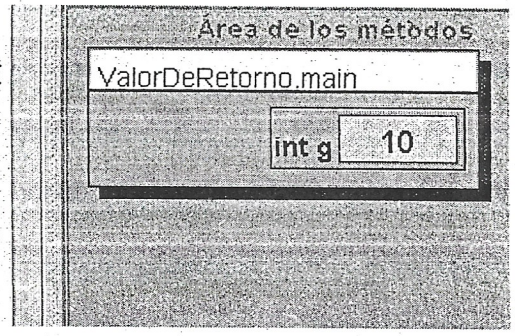
2.4 El valor obtenido, o sea "10" se guarda como valor de retorno. Se elimina el registro de activación para suma y regresa a main.

```
public class ValorDeRetorno {
    public static int suma(int op1, int op2) {
        return (op1 + op2);
    }
    public static void main() {
        int g = suma(7, 3);
        System.out.println(g);
    }
}
```



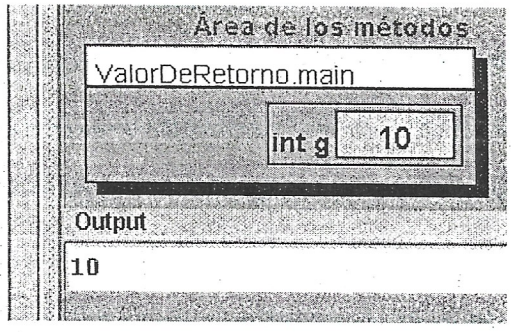
2.5 El valor de retorno, o sea "10" se asigna a la variable "g".

```
public class ValorDeRetorno {
    public static int suma(int op1, int op2) {
        return (op1 + op2);
    }
    public static void main() {
        int g = suma(7, 3);
        System.out.println(g);
    }
}
```



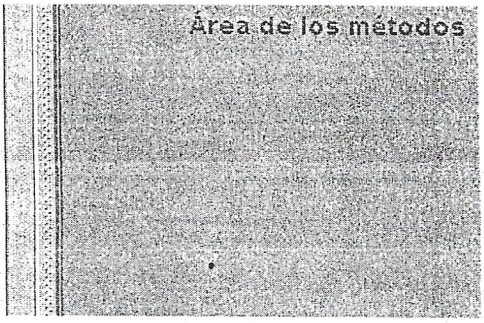
3 Se muestra el valor de "g", o sea "10" y realiza un salto de línea.

```
public class ValorDeRetorno {
    public static int suma(int op1, int op2) {
        return (op1 + op2);
    }
    public static void main() {
        int g = suma(7, 3);
        System.out.println(g);
    }
}
```



4 Se elimina el registro de activación para main, el arreglo y termina el programa.

```
public class ValorDeRetorno {
    public static int suma(int op1, int op2) {
        return (op1 + op2);
    }
    public static void main() {
        int g = suma(7, 3);
        System.out.println(g);
    }
}
```



1.5.3. Comentarios.

El valor de retorno se utiliza para indicar el resultado del trabajo realizado por un método. La palabra void indica que el método no devuelve datos.

La expresión return termina la ejecución de los métodos. Cuando devuelven datos, se utiliza de la siguiente forma:

```
return expresión;
```

Cuando el método no regresa información se usa así:

```
return;
```

Los programas también pueden devolver información al sistema operativo con la instrucción

```
System.exit(expresión entera);
```

Cuando se invoca termina la ejecución del programa. Si el parámetro que recibe es cero, indica que el programa terminó correctamente. Si el valor es diferente indica que el programa no terminó correctamente y el valor entero puede ser diferente para indicar distintos tipos de errores.

Cuando el programa termina sin invocar "System.exit", el programa devuelve un cero, indicando que terminó adecuadamente.

1.6. Atributos.

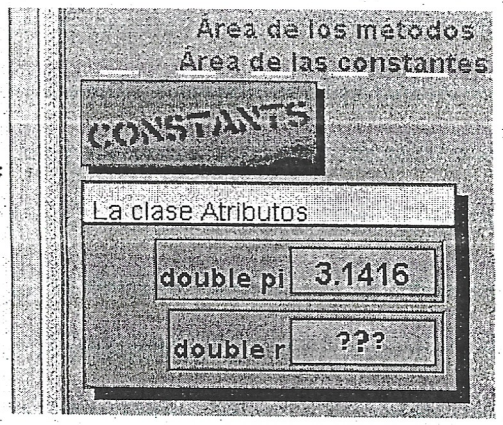
1.6.1. Código.

```
1  public class Atributos {
2      public static final double pi = 3.1416;
3      public static double r;
4      public static void calcula() {
5          System.out.println("a = " + (pi * r * r));
6      }
7      public static void main() {
8          r = 2.1;
9          calcula();
10     }
11 }
```

1.6.2. Ejecución.

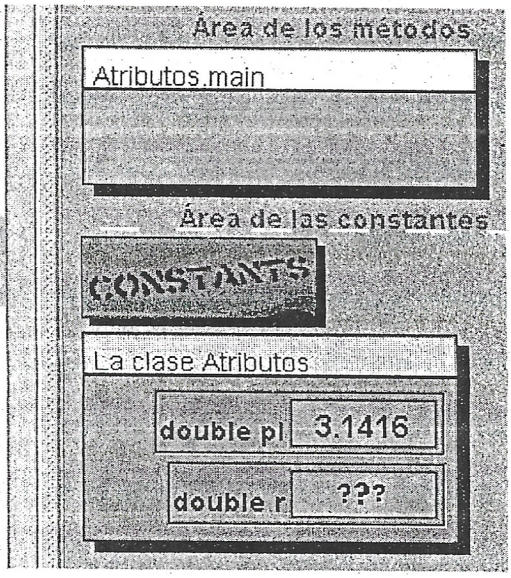
1 Crea la constante de clase pi con el valor 3.1416 y la variable de clase r sin valor en el área de variables de clase.

```
public class Atributos {
    public static final double pi = 3.1416;
    public static double r;
    public static void calcula() {
        System.out.println("a = " + (pi * r * r));
    }
    public static void main() {
        r = 2.1;
        calcula();
    }
}
```



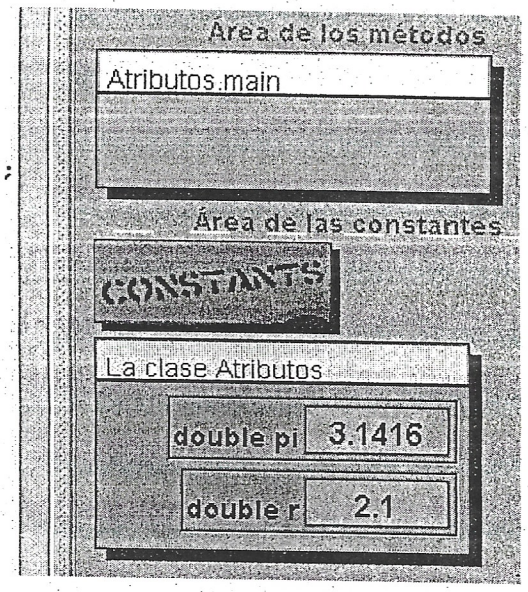
2 Se crea un registro de activación para main.

```
public class Atributos {
    public static final double pi = 3.1416;
    public static double r;
    public static void calcula() {
        System.out.println("a = " + (pi * r * r));
    }
    public static void main() {
        r = 2.1;
        calcula();
    }
}
```



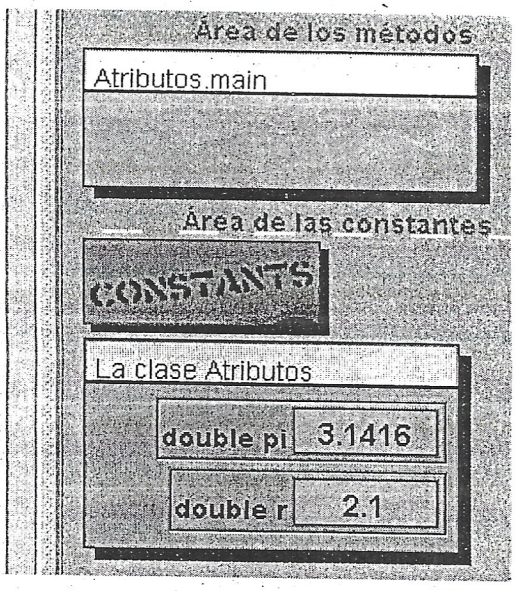
3 Asigna el valor "2.1" a la variable de clase r.

```
public class Atributos {
    public static final double pi = 3.1416;
    public static double r;
    public static void calcula() {
        System.out.println("a = " + (pi * r * r));
    }
    public static void main() {
        r = 2.1;
        calcula();
    }
}
```



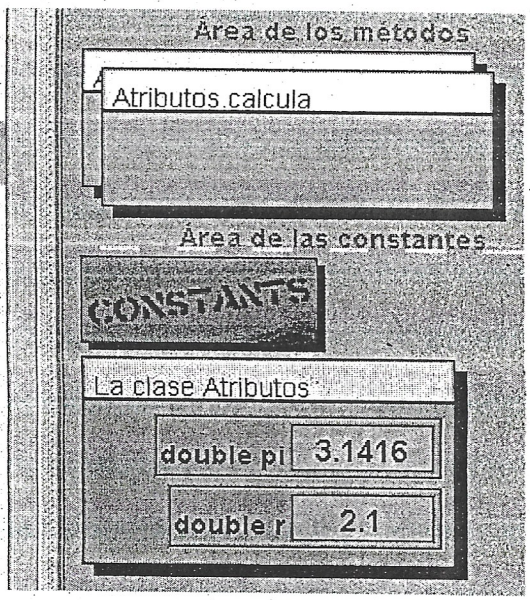
4 Se invoca el método calcula.

```
public class Atributos {
    public static final double pi = 3.1416;
    public static double r;
    public static void calcula() {
        System.out.println("a = " + (pi * r * r));
    }
    public static void main() {
        r = 2.1;
        calcula();
    }
}
```



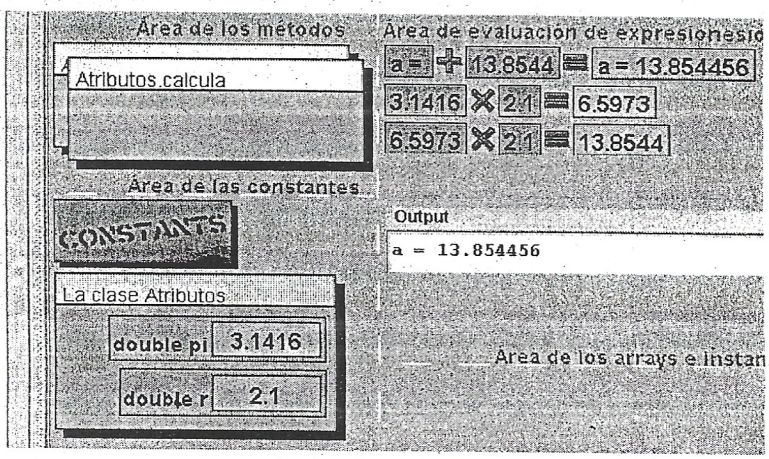
4.1 Se crea un registro de activación para calcula.

```
public class Atributos {
    public static final double pi = 3.1416;
    public static double r;
    public static void calcula() {
        System.out.println("a = " + (pi * r * r));
    }
    public static void main() {
        r = 2.1;
        calcula();
    }
}
```



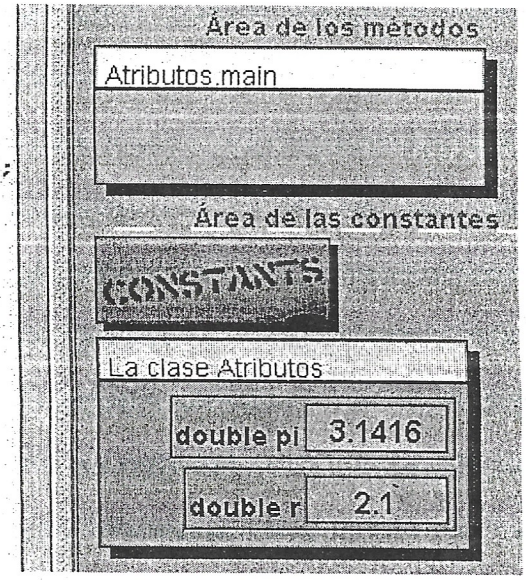
4.2 Muestra el resultado de ("a = " + (pi * r * r)), que es: "a = 13.854456".

```
public class Atributos {
    public static final double pi = 3.1416;
    public static double r;
    public static void calcula() {
        System.out.println("a = " + (pi * r * r));
    }
    public static void main() {
        r = 2.1;
        calcula();
    }
}
```



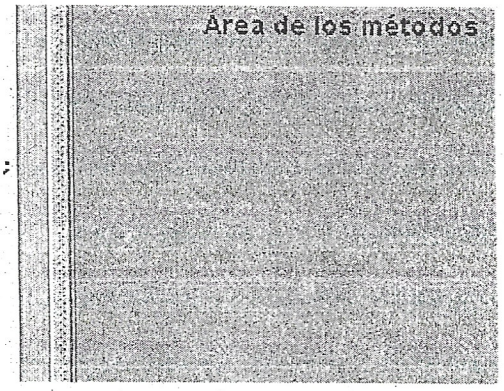
4.3 Elimina el registro de activación de calcula y regresa a main.

```
public class Atributos {
    public static final double pi = 3.1416;
    public static double r;
    public static void calcula() {
        System.out.println("a = " + (pi * r * r));
    }
    public static void main() {
        r = 2.1;
        calcula();
    }
}
```



5 Se elimina el registro de activación de main, los atributos "r" y "pi". Finaliza el programa.

```
public class Atributos {
    public static final double pi = 3.1416;
    public static double r;
    public static void calcula() {
        System.out.println("a = " + (pi * r * r));
    }
    public static void main() {
        r = 2.1;
        calcula();
    }
}
```



1.6.3. Comentarios.

Los atributos de clase solo se destruyen al terminar el programa y por lo mismo se pueden usar para enviar datos a un método o devolver valores. Aún así, un mal uso de ellos puede complicar la comprensión de los programas. Solo se recomienda su empleo cuando el paso de parámetros y valores de regreso se complican o bien para declarar constantes. En los lenguajes que no son orientados a objetos, los atributos de clase son equivalentes a las variables globales.

2.

Objetos.

2. Objetos.

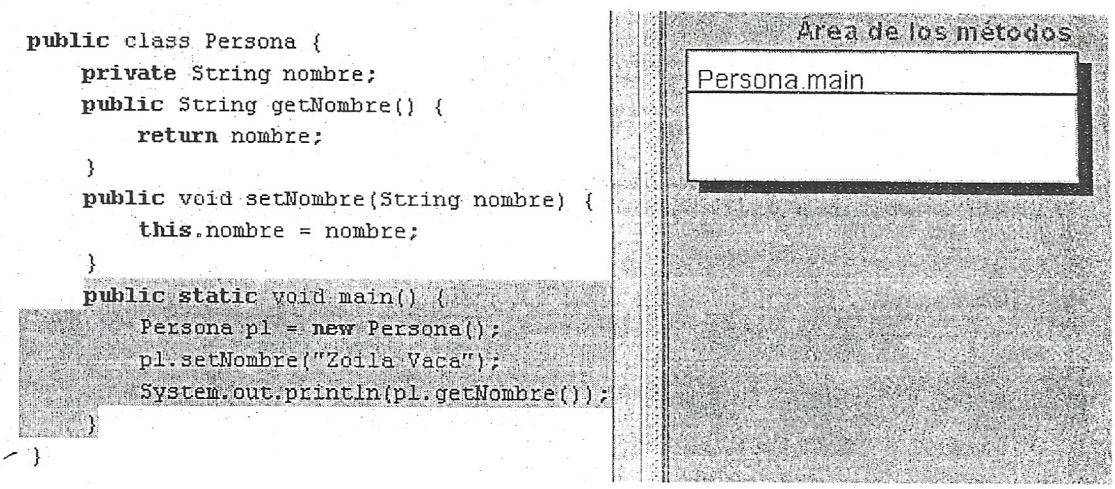
2.1. Objetos.

2.1.1. Código.

```
1  public class Persona {
2      private String nombre;
3      public String getNombre() {
4          return nombre;
5      }
6      public void setNombre(String nombre) {
7          this.nombre = nombre;
8      }
9      public static void main() {
10         Persona p1 = new Persona();
11         p1.setNombre("Zoila Vaca");
12         System.out.println(p1.getNombre());
13     }
14 }
```

2.1.2. Ejecución.

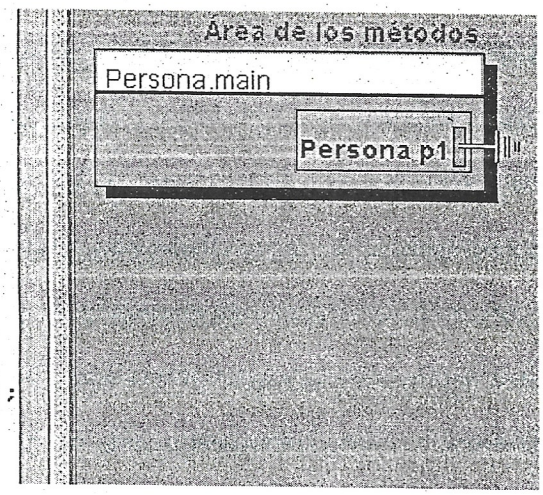
1 Se crea un registro de activación para main.



2 Se crea un objeto de la clase persona.

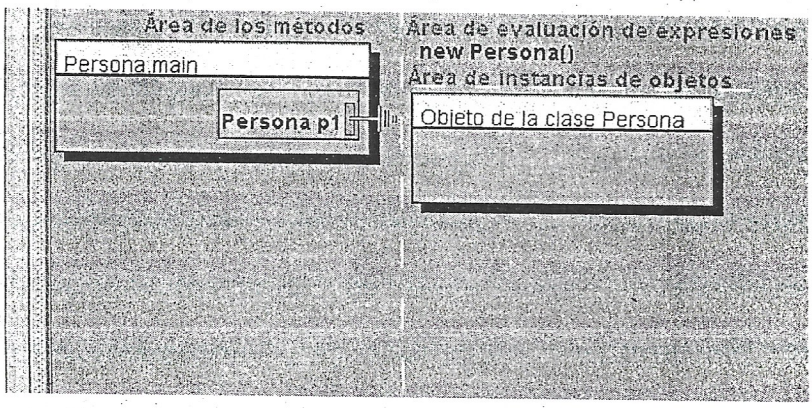
2.1 Se crea una referencia a un objeto de la clase Persona, pero aún no le asigna valor.

```
public class Persona {
    private String nombre;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public static void main() {
        Persona p1 = new Persona();
        p1.setNombre("Zoila Vaca");
        System.out.println(p1.getNombre());
    }
}
```



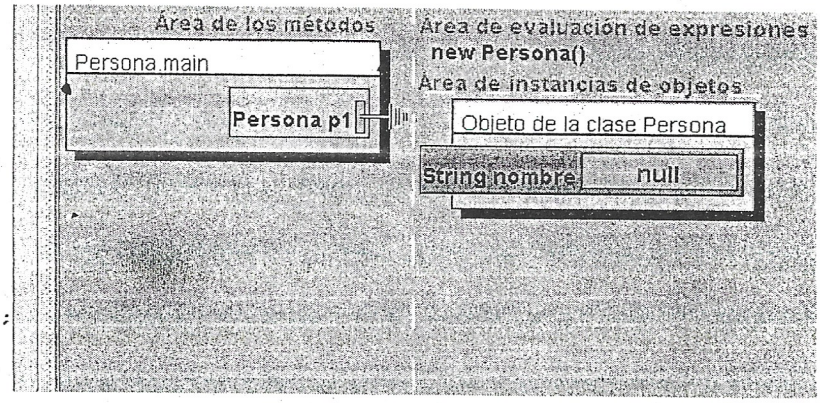
2.2 Se crea un objeto de la clase persona.

```
public class Persona {
    private String nombre;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public static void main() {
        Persona p1 = new Persona();
        p1.setNombre("Zoila Vaca");
        System.out.println(p1.getNombre());
    }
}
```



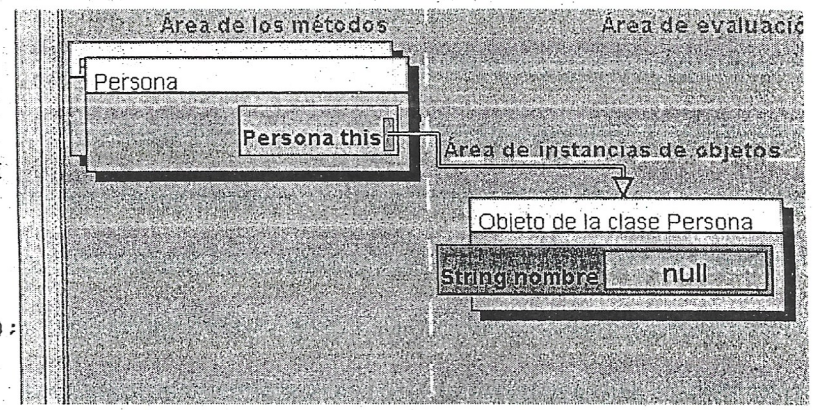
2.3 Al objeto se añade una copia de todos los atributos y métodos que no tengan la marca static.

```
public class Persona {
    private String nombre;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public static void main() {
        Persona p1 = new Persona();
        p1.setNombre("Zoila Vaca");
        System.out.println(p1.getNombre());
    }
}
```



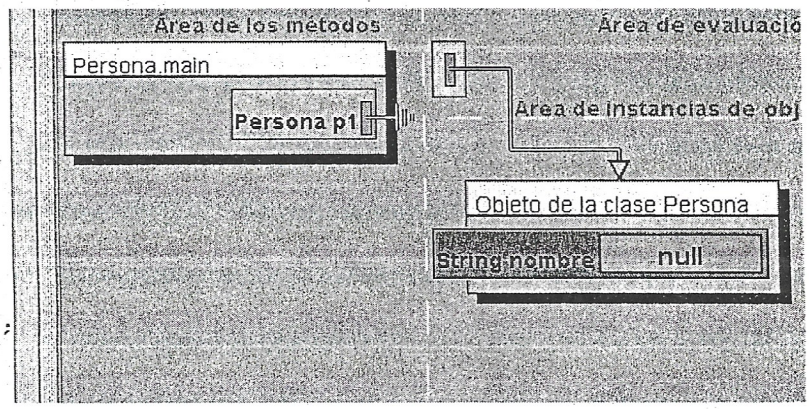
2.4 Se invoca el constructor, que es un método de objeto (es decir, no tiene la marca static) cuyo nombre es igual al de la clase. Cuando no lo encuentra, se crea uno que no tiene instrucciones y se conoce como **default constructor** en Inglés. Todos los métodos de objeto reciben una referencia al objeto que manejan y se llama **this**.

```
public class Persona {
    private String nombre;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public static void main() {
        Persona p1 = new Persona();
        p1.setNombre("Zoila Vaca");
        System.out.println(p1.getNombre());
    }
}
```



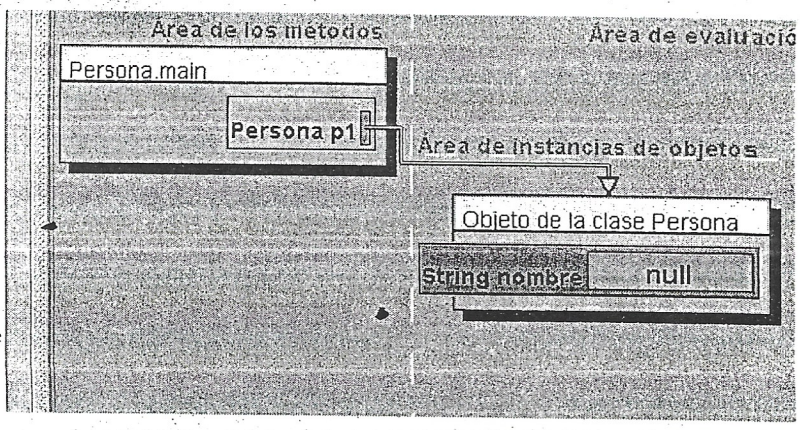
2.5 Se realizan las instrucciones del constructor. Como en nuestro caso no tiene, se termina y se elimina el registro de activación. Como resultado de la invocación del constructor se obtiene una referencia al objeto.

```
public class Persona {
    private String nombre;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public static void main() {
        Persona p1 = new Persona();
        p1.setNombre("Zoila Vaca");
        System.out.println(p1.getNombre());
    }
}
```



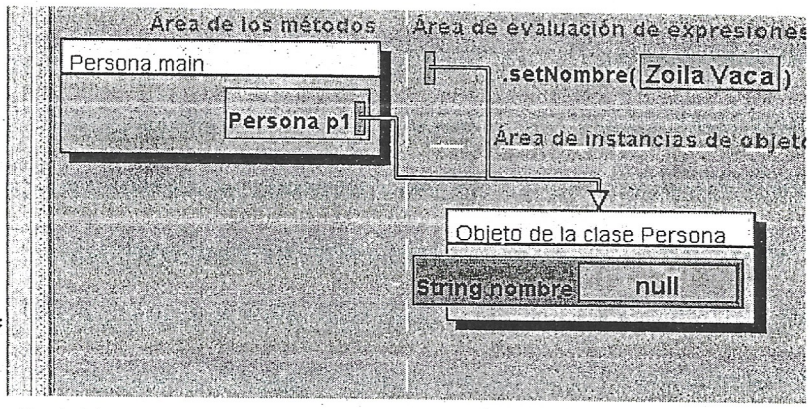
2.6 Asigna la referencia a la variable p1.

```
public class Persona {
    private String nombre;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public static void main() {
        Persona p1 = new Persona();
        p1.setNombre("Zoila Vaca");
        System.out.println(p1.getNombre());
    }
}
```



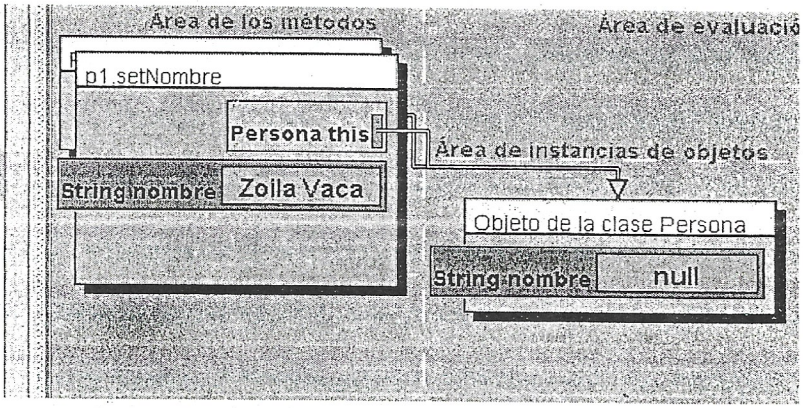
3 Invoca el método setNombre del objeto apuntado por p1, pasándole el parámetro "Zoila Vaca".

```
public class Persona {
    private String nombre;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public static void main() {
        Persona p1 = new Persona();
        p1.setNombre("Zoila Vaca");
        System.out.println(p1.getNombre());
    }
}
```



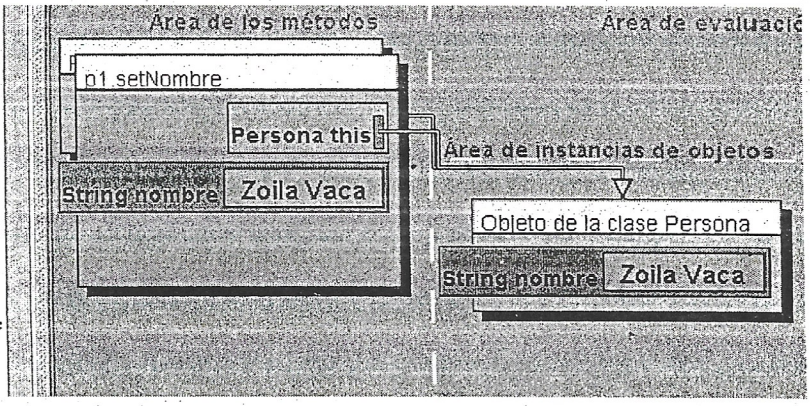
3.1 Se crea el registro de activación para p1.setNombre. El parámetro nombre recibe el valor "Zoila Vaca". Recuerda que todos los parámetros reciben la referencia this, que apunta al objeto que contiene el método invocado.

```
public class Persona {
    private String nombre;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public static void main() {
        Persona p1 = new Persona();
        p1.setNombre("Zoila Vaca");
        System.out.println(p1.getNombre());
    }
}
```



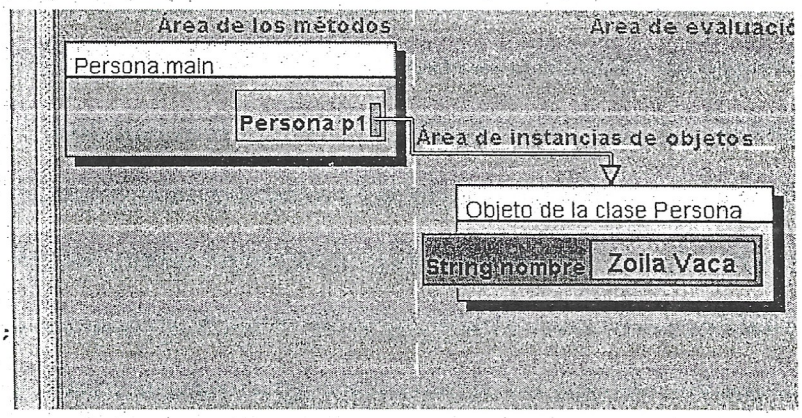
3.2 Se asigna el valor del parámetro nombre al atributo llamado igual. Para no confundirlos se emplea la referencia this.

```
public class Persona {
    private String nombre;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public static void main() {
        Persona p1 = new Persona();
        p1.setNombre("Zoila Vaca");
        System.out.println(p1.getNombre());
    }
}
```



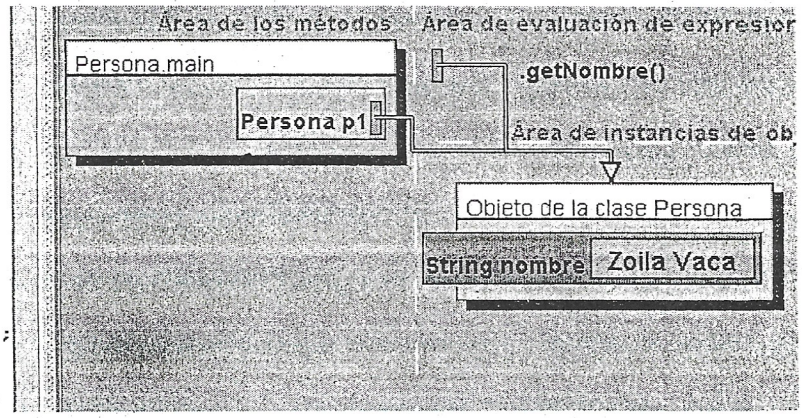
3.3 Elimina el registro de activación de p1.setNombre y regresa a main.

```
public class Persona {
    private String nombre;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public static void main() {
        Persona p1 = new Persona();
        p1.setNombre("Zoila Vaca");
        System.out.println(p1.getNombre());
    }
}
```



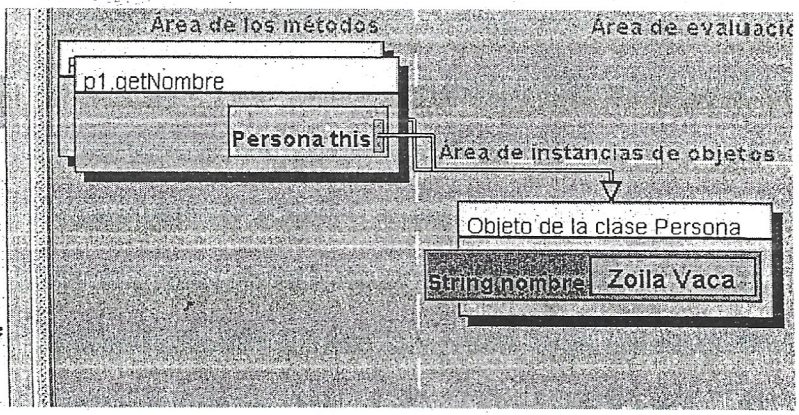
4 Invoca el método getNombre del objeto apuntado por p1.

```
public class Persona {
    private String nombre;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public static void main() {
        Persona p1 = new Persona();
        p1.setNombre("Zoila Vaca");
        System.out.println(p1.getNombre());
    }
}
```



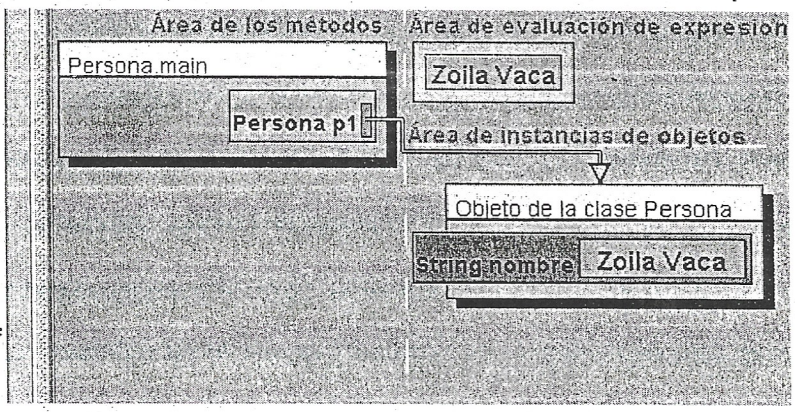
4.1 Se crea el registro de activación para el método de objeto getNombre.

```
public class Persona {
    private String nombre;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public static void main() {
        Persona p1 = new Persona();
        p1.setNombre("Zoila Vaca");
        System.out.println(p1.getNombre());
    }
}
```



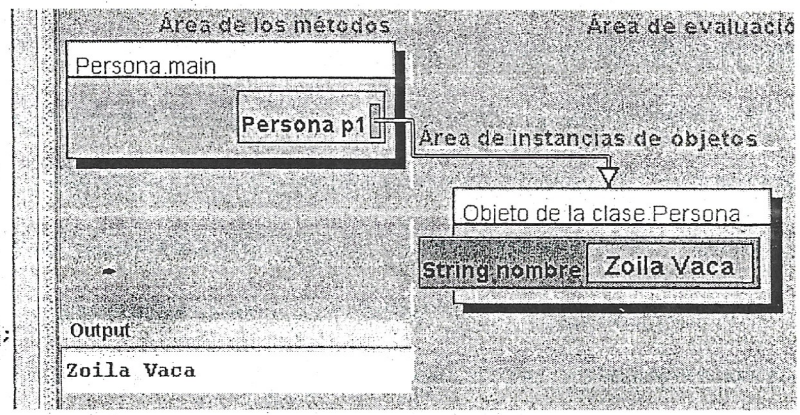
4.2 Termina el método y devuelve el valor de nombre. Lo busca en el registro de activación y como no lo encuentra lo busca en el objeto apuntado por this, donde lo encuentra. Devuelve el valor "Zoila Vaca" y regresa a main.

```
public class Persona {
    private String nombre;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public static void main() {
        Persona p1 = new Persona();
        p1.setNombre("Zoila Vaca");
        System.out.println(p1.getNombre());
    }
}
```



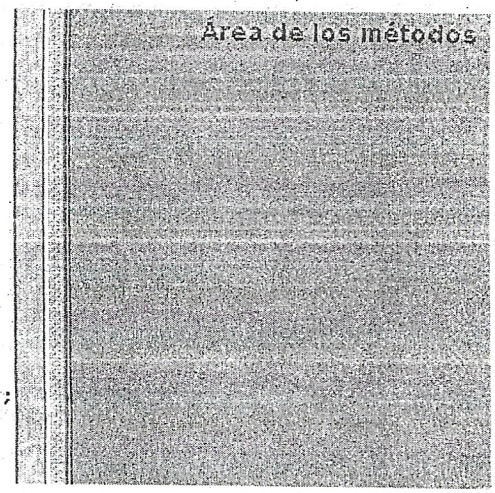
4.3 Se muestra el valor devuelto y realiza un salto de línea.

```
public class Persona {
    private String nombre;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public static void main() {
        Persona p1 = new Persona();
        p1.setNombre("Zoila Vaca");
        System.out.println(p1.getNombre());
    }
}
```



5 Se elimina el registro de activación, el objeto y termina el programa.

```
public class Persona {
    private String nombre;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public static void main() {
        Persona p1 = new Persona();
        p1.setNombre("Zoila Vaca");
        System.out.println(p1.getNombre());
    }
}
```



2.2. Más Objetos.

2.2.1. Código.

```
1  public class Persona2 {
2      private String nombre;
3      public String getNombre() {
4          return nombre;
5      }
6      public void setNombre(String nombre) {
7          this.nombre = nombre;
8      }
9      // 1. Crea registro de activación para main.
10     public static void main() {
11         // 2. Se crea un objeto de la clase Persona2 con una copia del
12         //    atributo nombre, setNombre y getNombre. Se asigna a la
13         //    referencia p1.
14         Persona2 p1 = new Persona2();
15         // 3. Se asigna valor al atributo nombre del objeto apuntado por
16         //    p1. Para ello se invoca setNombre del objeto apuntado por
17         //    p1.
18         p1.setNombre("Amanda Zapata");
19         // 4. Se crea otro objeto de la clase Persona con su propia copia
20         //    del atributo nombre, setNombre y getNombre. Se asigna a la
21         //    referencia p2.
22         Persona2 p2 = new Persona2();
23         // 5. Se asigna valor al atributo nombre del objeto apuntado por
24         //    p2. Para ello se invoca setNombre del objeto apuntado por
25         //    p2.
26         p2.setNombre("Armando Pacheco");
27         // 6. Se muestra el valor del atributo nombre asociado con p1
28         //    ("Amanda Zapata").
29         System.out.println("p2 = " + p2.getNombre());
30         // 7. Se muestra el valor del atributo nombre asociado con p2
```

```
31          //    ("Armando Pacheco");
32          System.out.println("p1 = " + p1.getNombre());
33          // 8. Se elimina el registro de activación y termina.
34      }
35  }
```

2.2.2. Comentarios.

El objetivo de este programa es que notes como cada objeto tiene sus propios atributos y métodos, aunque sean de la misma clase. Es diferente de los métodos y atributos de clase, que son únicos y por cierto accesibles a todos los objetos.

2.3. Herencia.

2.3.1. Código.

```
1  class Deportista {
2      // Los atributos marcados con protected también se pueden usar en las
3      // clases derivadas y las que pertenezcan al mismo paquete.
4      protected String nombre;
5      public String getNombre() {
6          return nombre;
7      }
8      public void setNombre(String nombre) {
9          this.nombre = nombre;
10     }
11 }
12 public class Futbolista extends Deportista {
13     // Los atributos marcados como private solo se pueden usar en las clases
14     // donde se definen.
15     private String posición;
16     public String getDatos() {
17         return "Futbolista: " + nombre + ", " + posición;
18     }
19     public void setPosición(String posición) {
20         this.posición = posición;
21     }
22     // 1. Crea registro de activación para main y recibe los parámetros de
23     //    línea de comando.
24     public static void main(String[] args) {
25         // 2. Se crea un objeto de la clase Futbolista con una copia de
26         //    nombre, posición, setNombre, setPosición, getNombre y
27         //    getPosición. Se asigna a la referencia f.
28         Futbolista f = new Futbolista();
29         // 3. Se asigna valor al atributo nombre del objeto apuntado por
30         //    f. Para ello se invoca setNombre del objeto apuntado por f.
31         f.setNombre("Ronaldinho");
32         // 4. Se asigna valor al atributo posición del objeto apuntado por
33         //    f. Para ello se invoca setPosición del objeto apuntado por f.
34         f.setPosición("medio");
35         // 5. Se muestra el valor del atributo nombre asociado con f
```

```
36          //      ("Ronaldinho").
37          System.out.println("Nombre: " + f.getNombre());
38          // 6. Imprime "Futbolista: Ronaldinho, medio".
39          System.out.println(f.getDatos());
40          // 7. Se elimina el registro de activación y termina.
41      }
42  }
```

2.3.2. Comentarios.

La clase que sirve como base para crear otra se conoce como superclase y la derivada también se conoce como subclase.

Cuando no quieras que una clase tenga subclases, puedes usar la palabra **final** antes de class.

2.4. Constuctores.

2.4.1. Código.

```
1  public class Persona3 {
2      // 2.1. En el objeto creado se añade una copia del atributo nombre.
3      private String nombre;
4      // 2.4. Como ahora si tenemos constructor definido, se crea un
5      //      registro de activación para él.
6      public Persona3(String nombre) {
7          // 2.5. se copia el valor del parámetro al objeto.
8          this.nombre = nombre;
9          // 2.6. Se elimina el registro de activación y se devuelve una
10         //      referencia al objeto. Regres a main.
11     }
12     // 2.2. En el objeto creado se añade una copia del método getNombre.
13     // 3.1.1. Crea un registro de activación para p1.getNombre. La
14     //        referencia this apunta al objeto indicado por p1.
15     public String getNombre() {
16         // 3.1.2. Busca "nombre" en el registro de activación. No lo
17         //        encuentra y lo busca en el objeto. Destruye el registro
18         //        de activación, regresa a main y devuelve "Zoila Vaca".
19         return nombre;
20     }
21     // 4.1. Crea un registro de activación para p1.setNombre con
22     //      nombre = "Zoi". La referencia this apunta al objeto indicado
23     //      por p1.
24     public void setNombre(String nombre) {
25         // 4.2. "nombre" es el parámetro. "this.nombre" es el atributo
26         //      dentro del objeto. Asigna el valor del parámetro ("Zoi")
27         //      al atributo nombre, descartando su valor anterior.
28         this.nombre = nombre;
29         // 4.3. destruye el registro de activación y regresa a main.
30     }
```

```
31      // 1. Crea registro de activación para main.
32      public static void main() {
33          // 2. Crea un objeto de la clase persona.
34          Persona3 p = new Persona3("Zoila Vaca");
35          // 3. Muestra el nombre del objeto.
36          // 3.1. Invoca el método getNombre.
37          // 3.2. Muestra en pantalla "Zoila Vaca".
38          System.out.println(p.getNombre());
39          // 4. Invoca setNombre del objeto apuntado por p.
40          p.setNombre("Zoi");
41          // 5. Muestra el nuevo valor de nombre, o sea "Zoi".
42          System.out.println(p.getNombre());
43          // 6. Destruye el registro de activación para main y termina.
44      }
45  }
```

2.5. Constructores y Herencia.

2.5.1. Código.

```
 1  class Deportista2 {
 2      protected String nombre;
 3      // Cuando se declara explícitamente un constructor, el constructor por
 4      // defecto no se declara. Si lo necesitas, debes declararlo.
 5      public Deportista2() {}
 6      // 2.2.2.1. Se crea un registro de activación para f.Deportista2.
 7      public Deportista2(String nombre) {
 8          // 2.2.2.2. Se asigna el valor del parámetro al atributo.
 9          this.nombre = nombre;
10          // 2.2.2.3. Se elimina el registro y regresa a f.Deportista2.
11      }
12  }
13  public class Futbolista2 extends Deportista2 {
14      private String posición;
15      // 2.2.1. Se crea un registro de activación para f.Futbolista2.
16      public Futbolista2(String nombre, String posición) {
17          // 2.2.2. Invoca al constructor de la clase padre, o sea Deportista2.
18          //        Se le envía el valor del parámetro nombre, que es
19          //        "Ronaldinho". La invocación a "super" siempre debe ser la
20          //        primera instrucción del constructor.
21          super(nombre);
22          // 2.2.3. Se asigna el valor del parámetro al atributo.
23          this.posición = posición;
24          // 2.2.4. Se elimina el registro y regresa a main.
25      }
26      public String getDatos() {
27          return "Futbolista: " + nombre + ", " + posición;
28      }
29      // 1. Crea un registro de activación para main.
30      public static void main() {
31          // 2. Crea un objeto y lo asigna a f.
32          // 2.1. Se crea un objeto, poniéndole una copia de nombre,
```

```
33          //      posición, getNombre y getPosición.
34          // 2.2. Se invoca al constructor.
35          Futbolista2 f = new Futbolista2("Ronaldinho", "medio");
36          // 3. Imprime "Futbolista: Ronaldinho, medio".
37          System.out.println(f.getDatos());
38          // 4. Elimina el registro de activación y termina.
39      }
40  }
```

3.

Temas

Avanzados.

3. Temas Avanzados.

3.1. Interfaces.

3.1.1. Código.

```
1  interface Cronista {
2      void gol();
3      void penalti();
4  }
5  class ImplementacionDeCronista implements Cronista {
6      public void gol() {
7          System.out.println("!!!GOOOOOLL, GOOOOLL!!!!");
8          System.out.println("!!!GOOOOOLL, GOOOOLL!!!!");
9      }
10     public void penalti() {
11         System.out.println("!!PEEEEE-NAAAL-TIIII!!!");
12     }
13 }
14 public class Interfaces {
15     public static void main() {
16         Cronista raul = new ImplementacionDeCronista();
17         raul.gol();
18         raul.penalti();
19     }
20 }
```

3.1.2. Comentarios.

Las interfaces se utilizan cuando quieres obligar a que varias clases contengan los mismos métodos. Cuando una clase implementa una interfaz, debe tener definidos todos los objetos indicados en la interfaz.

3.2. Clases Anónimas.

3.2.1. Código.

```
1  interface Mensaje {
2      void muestra();
3  }
4  public class Anonimas {
5      public static void main() {
6          Mensaje m = new Mensaje() {
7              public void muestra() {
8                  System.out.println("HOLA");
9              }
10         };
11         m.muestra();
```

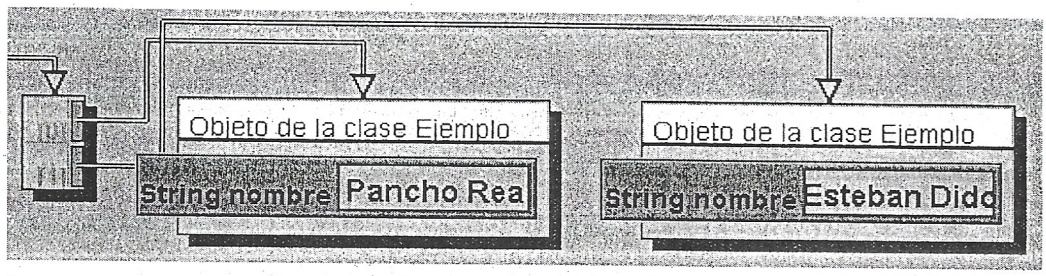
```
12      }
13      // La clase anónima esequivalente a crear
14      // una clase como la siguiente:
15      public class Implementacion implements Mensaje {
16          public void muestra() {
17              System.out.println("HOLA");
18          }
19      }
20  }
```

3.3. Arreglos de Objetos.

3.3.1. Código.

```
1   class Ejemplo {
2       private String nombre;
3       public Ejemplo(String nombre) {
4           this.nombre = nombre;
5       }
6       public String toString() {
7           return "Ejemplo: " + nombre;
8       }
9   }
10  public class Arreglos {
11      public static void main() {
12          Ejemplo[] arr = new Ejemplo[2];
13          arr[0] = new Ejemplo("Pancho Rea");
14          arr[1] = new Ejemplo("Esteban Dido");
15          for (int i = 0; i < arr.length; i++) {
16              System.out.println(arr[i]);
17          }
18      }
19  }
```

3.3.2. Estructura del arreglo.



3.3.3. Comentarios.

Todos los objetos son derivados de otra clase que se llama **Object**, aunque no esté declarado. En ella está declarado el método **toString**, que se utiliza para desplegar objetos.

String es una clase y cada vez que deseas mostrar una cadena, se invoca toString.

3.4. For each y Clases Internas Estáticas.

3.4.1. Código.

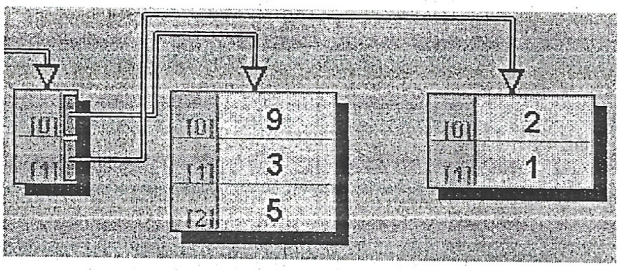
```
1  // ESTE PROGRAMA NO FUNCIONA EN JELIOT.
2  public class ForEach {
3      public static class Ejemplo {
4          private String nombre;
5          public Ejemplo(String nombre) {
6              this.nombre = nombre;
7          }
8          // @Override indica que este método debe estar definido en la
9          //           superclase y si no lo está se marca error.
10         @Override
11         public String toString() {
12             return "Ejemplo: " + nombre;
13         }
14     }
15     public static void main(String[] args) {
16         Ejemplo[] arr = new Ejemplo[2];
17         arr[0] = new Ejemplo("Pancho Rea");
18         arr[2] = new Ejemplo("Esteban Dido");
19         for (Ejemplo e : arr) {
20             System.out.println(e);
21         }
22     }
23 }
```

3.5. Arreglos Bidimensionales.

3.5.1. ArreglosBidimensionales.

```
1  public class ArreglosBidimensionales {
2      public static void main() {
3          int[][] b = {{9, 3, 5}, {2, 1}};
4          // Muestra todos lo elementos del arreglo.
5          for (int i = 0; i < b.length; i++) {
6              for (int j = 0; j < b[i].length; j++) {
7                  System.out.print(b[i][j]);
8              }
9              System.out.println();
10         }
11     }
12 }
```

3.5.2. Estructura del arreglo bidimensional.



3.5.3. ArreglosBidimensionales2.

```
1  // ESTE PROGRAMA NO FUNCIONA EN JELIOT.
2  public class ArreglosBidimensionales2 {
3      private static void muestra(int[][] arr) {
4          for (int[] is : arr) {
5              for (int i : is) {
6                  System.out.print(i);
7              }
8              System.out.println();
9          }
10     }
11     public static void main(String[] args) {
12         int[][] a = new int[2][3];
13         // Llena el arreglo.
14         for (int i = 0; i < a.length; i++) {
15             for (int j = 0; j < a[i].length; j++) {
16                 a[i][j] = i + j;
17             }
18         }
19         muestra(a);
20     }
21 }
```

3.5.4. Renglones y Columnas.

```
1  public class RenglonesYColumnas {
2      public static void main() {
3          int[][] b = {{9, 3, 5}, {2, 1, 5}};
4          int sumaRenglon0 = 0;
5          int sumaColumna2 = 0;
6          int suma = 0;
7          for (int i = 0; i < b.length; i++) {
8              sumaColumna2 += b[i][2];
9          }
10         for (int j = 0; j < b[0].length; j++) {
11             sumaRenglon0 += b[0][j];
12         }
13         for (int i = 0; i < b.length; i++) {
14             for (int j = 0; j < b[i].length; j++) {
15                 suma += b[i][j];
```

```
16                  }
17              }
18          }
19  }
```