11

Uso de encapsulación y constructores

Visión general

- Encapsulación significa ocultar los campos de objeto mediante la conversión de todos los campos en privados:
  - Utilizar los métodos getter y setter.
  - En los métodos setter, utilice el código para asegurarse de que los valores son válidos.
- La encapsulación exige una programación a la interfaz:
  - El tipo de dato del campo es irrelevante para el método de llamada.
  - La clase se puede cambiar mientras que la interfaz sea la misma.
- La encapsulación fomenta un diseño orientado a objetos (OO) correcto.

Modificador `public`

```
public class Elevator {
   public boolean doorOpen=false;
   public int currentFloor = 1;
   public final int TOP_FLOOR = 10;
   public final int MIN_FLOOR = 1;

   ... < code omitted > ...

 public void goUp() {
    if (currentFloor == TOP_FLOOR) {
      System.out.println("Cannot go up further!");
    }
    if (currentFloor < TOP_FLOOR) {
      currentFloor++;
      System.out.println("Floor: " + currentFloor);
    }
  }
}
```

Riesgos del acceso a un campo `public`

```
Elevator theElevator = new Elevator();

theElevator.currentFloor = 15;
```

Puede causar un problema.

Modificador `private`

```
public class Elevator {
   private boolean doorOpen=false;
   private int currentFloor = 1;
   private final int TOP_FLOOR = 10;
   private final int MIN_FLOOR = 1;

    ... < code omitted > ...

 public void goUp() {
    if (currentFloor == TOP_FLOOR) {
      System.out.println("Cannot go up further!");
    }
    if (currentFloor < TOP_FLOOR) {
      currentFloor++;
      System.out.println("Floor: " + currentFloor);
    }
  }
}
```

No se puede acceder a ninguno de estos campos desde otra clase con una notación de puntos.

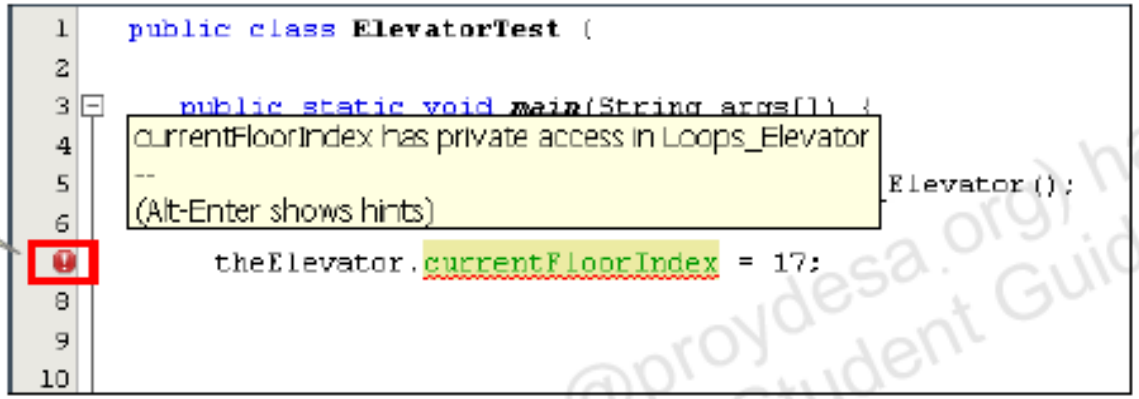
Intento de acceso a un campo `private`

```
Elevator theElevator = new Elevator();

theElevator.currentFloor = 15;
```

← No permitido

NetBeans mostrará un error. Puede obtener una explicación si coloca aquí el cursor.



Modificador `private` en los métodos

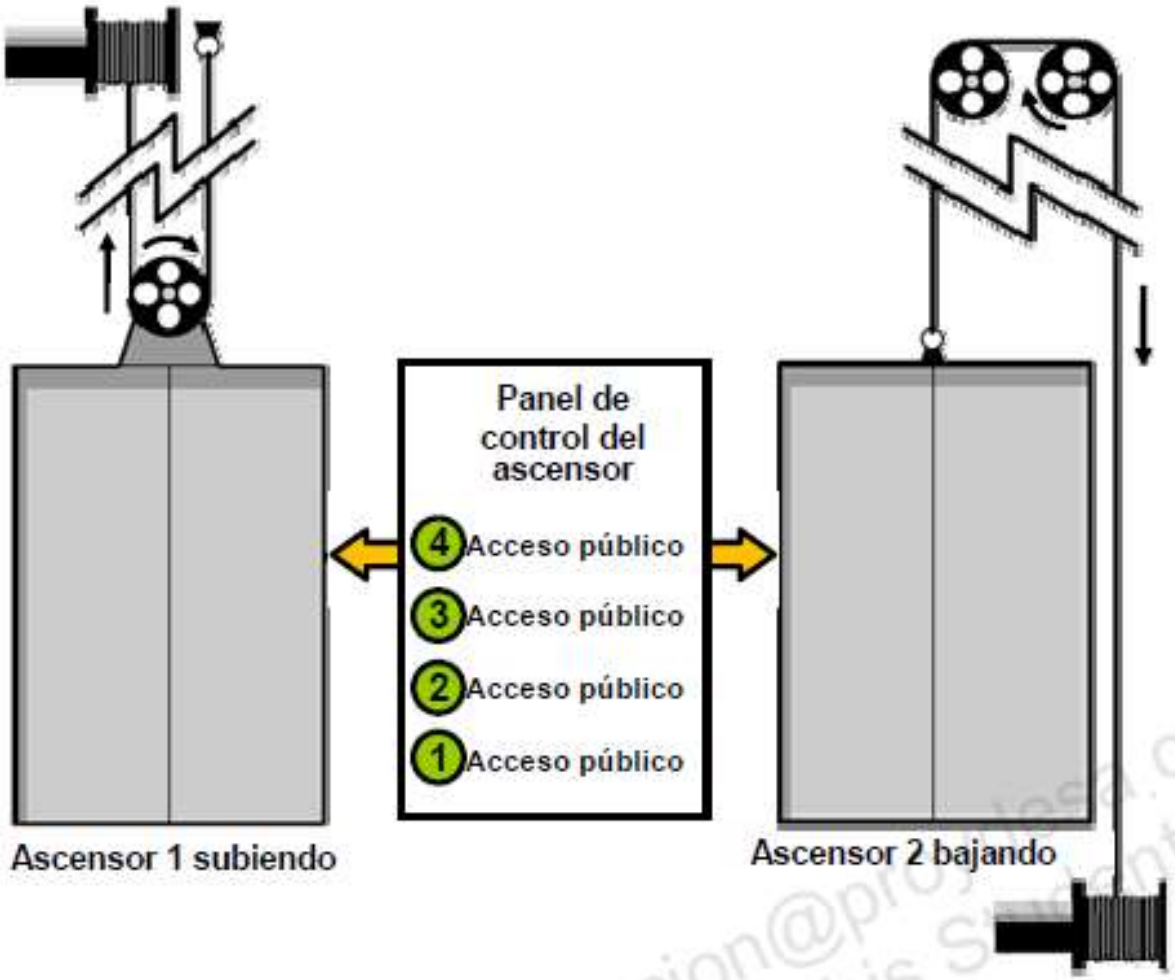
```
public class Elevator {
   ... < code omitted > ...

 private void setFloor() {
    int desiredFloor = 5;
    while (  currentFloor != desiredFloor  ){
       if (currentFloor < desiredFloor) {
          goUp();
       } else {
          goDown();
       }
    }
 }

 public void requestFloor(int desiredFloor) {
    ... < contains code to add requested floor to a queue > ...
 }
```

¿Debe ser privado este método?

Interfaz e implantación



Métodos get y set

```
public class Shirt {
  private int shirtID = 0; // Default ID for the shirt
  private String description = "-description required-"; // default
  // The color codes are R=Red, B=Blue, G=Green, U=Unset
  private char colorCode = 'U';
  private double price = 0.0; // Default price for all items

  public char getColorCode() {
    return colorCode;
  }
  public void setColorCode(char newCode) {
    colorCode = newCode;
  }
  // Additional get and set methods for shirtID, description,
  // and price would follow

} // end of class
```

Uso de los métodos setter y getter

```
public class ShirtTest {
    public static void main (String[] args) {
    Shirt theShirt = new Shirt();
    char colorCode;

    // Set a valid colorCode
    theShirt.setColorCode('R');
    colorCode = theShirt.getColorCode();
    // The ShirtTest class can set and get a valid colorCode
    System.out.println("Color Code: " + colorCode);

    // Set an invalid color code
    theShirt.setColorCode('Z');
    colorCode = theShirt.getColorCode();
    // The ShirtTest class can set and get an invalid colorCode
    System.out.println("Color Code: " + colorCode);
}
```

No es un código de color válido.

Método setter con comprobación

```
public void setColorCode(char newCode) {
    switch (newCode) {
        case 'R':
        case 'G':
        case 'B':
        colorCode = newCode;
        break;
        default:
        System.out.println("Invalid colorCode. Use R, G, or B");
    }
}
```

Uso de los métodos setter y getter

```
public class ShirtTest {
   public static void main (String[] args) {
   Shirt theShirt = new Shirt();
   System.out.println("Color Code: " + theShirt.getColorCode());

   // Try to set an invalid color code
   Shirt1.setColorCode('Z');
   System.out.println("Color Code: " + theShirt.getColorCode());
}
```

No es un código de color válido.

Salida:

```
Color Code: U
Invalid colorCode. Use R, G, or B
Color Code: U
```

Llamada anterior a setColorCode(): muestra un valor por defecto.

La llamada a setColorCode imprime un mensaje de error.

colorCode no modificado por un argumento no válido transferido a setColorCode()

Inicialización de un objeto `Shirt`

```
public class ShirtTest {
    public static void main (String[] args) {
    Shirt theShirt = new Shirt();

    // Set values for the Shirt
    theShirt.setColorCode('R');
    theShirt.setDescription("Outdoors shirt");
    theShirt.price(39.99);

}
```

Constructores

- Los constructores son estructuras similares a un método de una clase:
  - Tienen el mismo nombre que la clase.
  - Se suelen utilizar para inicializar campos en un objeto.
  - Pueden recibir argumentos.
  - Se pueden sobrecargar.
- Todas las clases tienen al menos un constructor:
  - Si no hay constructores explícitos, el compilador Java proporciona un constructor sin argumentos por defecto.
  - Si hay más de un constructor explícito, no se proporcionará ningún constructor por defecto.

Creación de constructores

Sintaxis:

```
[modifiers] class ClassName {

    [modifiers] ClassName([arguments]) {
      code_block
    }
}
```

Creación de constructores

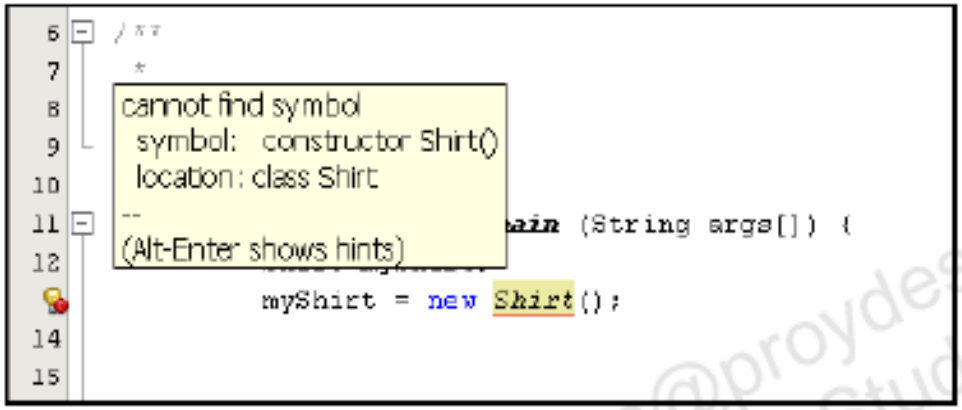
```
public class Shirt {
  public int shirtID = 0; // Default ID for the shirt
  public String description = "-description required-"; // default
  // The color codes are R=Red, B=Blue, G=Green, U=Unset
  private char colorCode = 'U';
  public double price = 0.0; // Default price all items

 // This constructor takes one argument
  public Shirt(char colorCode ) {
      setColorCode(colorCode);
  }
```

Inicialización de un objeto `Shirt` con un constructor

```
public class ShirtTest {
    public static void main (String[] args) {
    Shirt theShirt = new Shirt('G');

    theShirt.display();
}
```



Varios constructores

```
public class Shirt {
  ... < declarations for field omitted > ...

  // No-argument constructor
  public Shirt() {
     // You could add some default processing here
  }
  // This constructor takes one argument
  public Shirt(char colorCode ) {
      setColorCode(colorCode);
  }
  public Shirt(char colorCode, double price) {

      this(colorCode);
      setPrice(price);
  }
```

Si es necesario, se debe agregar explícitamente.

Encadenamiento de constructores.