ESCOLA SUPERIOR DE ENXEÑARÍA INFORMÁTICA

Universidade Vigo

TEMA 2: Encapsulación

Programación II - 2017/2018

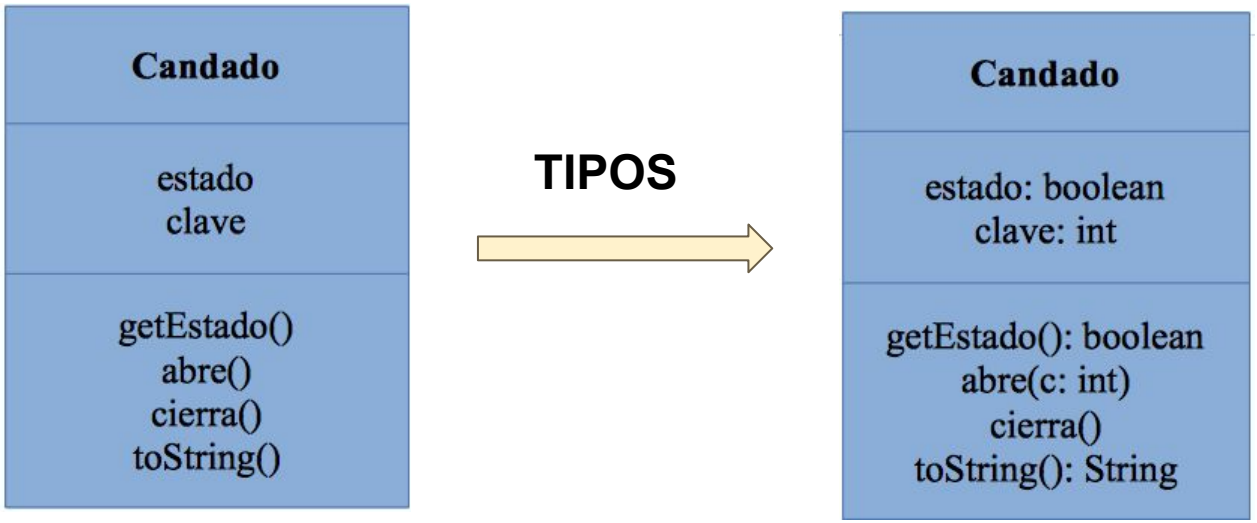
Pedro Cuesta Morales, Baltasar García Pérez-Schofield,
Encarnación González Rufino (Dpto. de Informática)

Índice

2. Identificación de clases

- Orientación a objetos → análisis textual: identificar clases dominio del problema (atributos y métodos)
  - Nombres: clases y atributos
  - Verbos: métodos

Un candado electrónico se abre mediante una clave (una secuencia de números), mientras que se puede cerrar sin ella. Se visualiza como "*candado: abierto*" o "*candado: cerrado*". Es posible obtener el estado abierto o cerrado. La clave no puede ser cambiada, se asigna en el momento de crear el candado.



Clase Candado

```
class Candado {
	private int clave;
	private boolean estado;

	public Candado(int k)
	{
		clave = k;
		estado = false;
	}

	public boolean getEstado()
	{
		return estado;
	}

	public void abre(int k)
	{
		if ( clave == k ) {
			estado = true;
		}
	}

	public void cierra()
	{
		estado = false;
	}

	public String toString()
	{
		String toret = "abierto";

		if ( !getEstado() ) {
			toret = "cerrado";
		}

		return "Candado: " + toret;
	}
}
```

estado = true → abierto
estado = false → cerrado

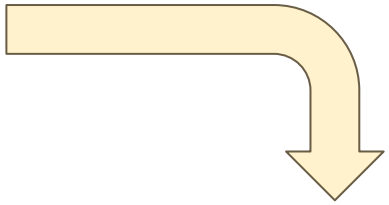
Clase Candado

```java
public static void main (String[] args)
{
    Candado c1= new Candado( 123 );

    System.out.println( c1 );
    c1.abre( 456 );
    System.out.println( c1 );

    c1.abre( 123 );
    System.out.println( c1 );

    c1.cierra();
    System.out.println( c1 );
}
```



stdout

```
Candado: cerrado
Candado: cerrado
Candado: abierto
Candado: cerrado
```

2.1. Discusión sobre visibilidad

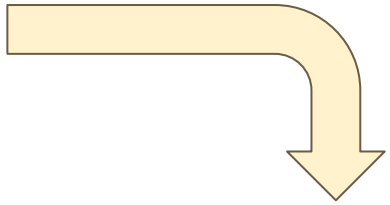
1) public int clave;

3) ~~getClave();~~

```
Candado c1= new Candado( 123 );

System.out.println( c1 );

int copiaClave = c1.clave;
c1.abre( copiaClave );
System.out.println( c1 );
```



stdout

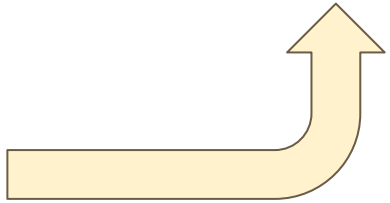
```
Candado: cerrado
Candado: abierto
```

2) public boolean estado;

```
Candado c1= new Candado( 123 );

System.out.println( c1 );

c1.estado = true;
System.out.println( c1 );
```



2.2. Qué pasa cuando no se indica visibilidad

```
class Candado {
      int clave;
      boolean estado;

      public Candado(int k)
      {
            clave = k;
            estado = false;
      }
```

clave y *estado* accesibles desde el exterior de la clase

Accesibles desde el mismo paquete (inaccesibles desde otros paquetes)

package: colecciones de clases

proyecto de programación: varios paquetes

3. Documentación

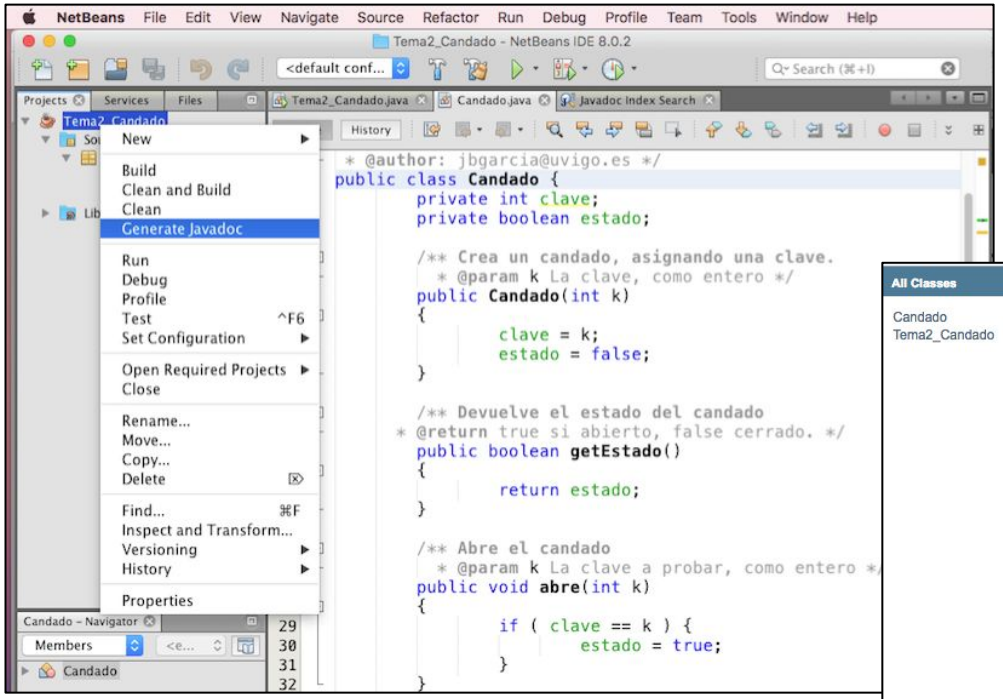
Escribir código (sobre todo en métodos) es importante añadir comentarios

JavaDoc: herramienta que permite obtener automáticamente la documentación con referencias cruzadas, a partir de comentarios en el código:

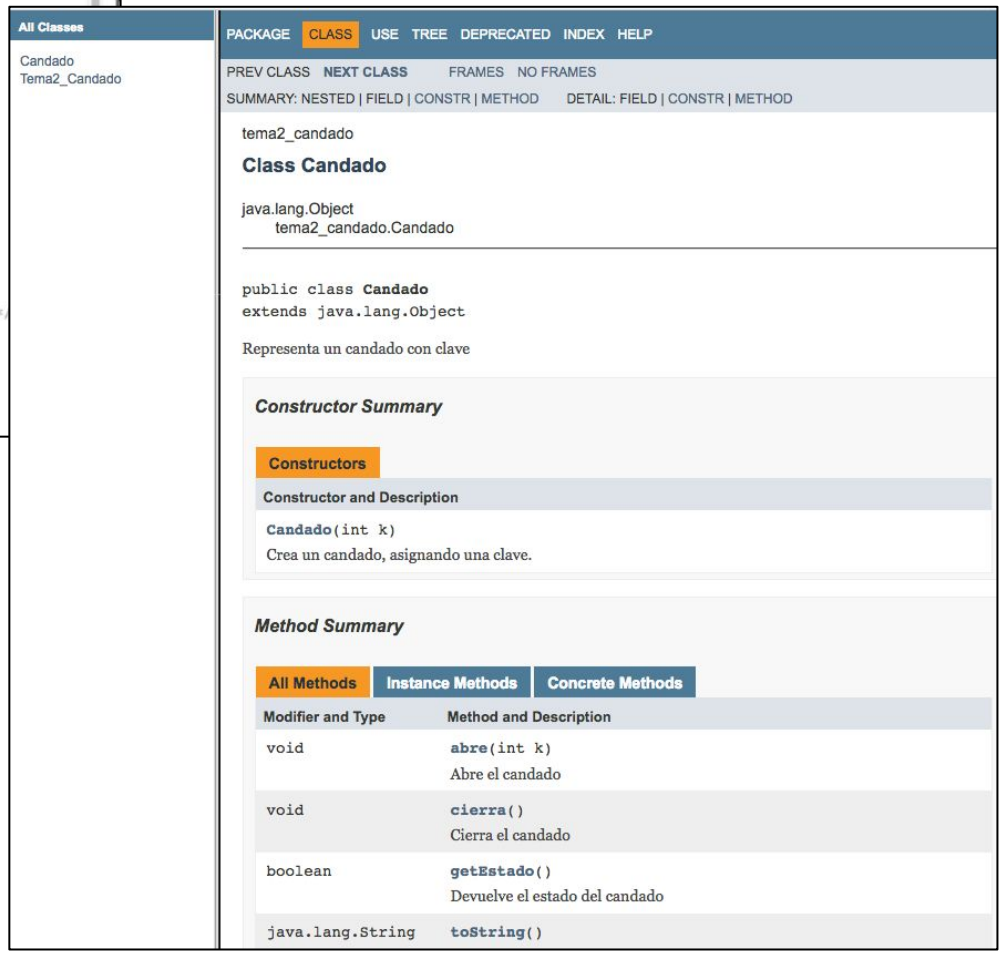
- Comienzan con /** en vez de /*
- uso de etiquetas → comienzan con @

Etiqueta	Explicación
`@param <parámetro> <explicación>`	Proporciona documentación sobre un parámetro en concreto.
`@return <explicación>`	Proporciona documentación sobre el retorno de un método.
`@see <nombre>`	Permite relacionar la documentación de este miembro con otro.

JavaDoc en Netbeans



- HTML
- Estilo API JAVA



http://ideone.com/PH12Oe

4. Setters

http://ideone.com/IBZQkQ

Métodos para modificar el valor de los atributos de un objeto

```java
/** modifica la coordenada x del punto
 * @param la nueva coordenada x del punto, como entero */
public void setX( int x )
{
   this.x = x;
}

/** modifica la coordenada y del punto
 * @param la nueva coordenada y del punto, como entero */
public void setY( int y )
{
   this.y = y;
}
```

```java
public static void main(String[] args)
{
    Punto p1 = new Punto( 5, 5 );

    System.out.println( "La distancia de "
                    + p1 + " a "
                    + Punto.ORIGEN + " es "
                    + p1.calculaDistanciaOrigen() );

    // se modifican las coordenadas del punto p1 y se vuelve a mostrar
    p1.setX( 10 );
    p1.setY( 10 );
    System.out.println( "La distancia de "
                    + p1 + " a "
                    + Punto.ORIGEN + " es "
                    + p1.calculaDistanciaOrigen() );

}
```

stdout

```
La distancia de (5, 5) a (0, 0) es 7.0710678118654755
La distancia de (10, 10) a (0, 0) es 14.142135623730951
```

4. Setters

Usar setters cuando sea necesario

Alternativa: crear un nuevo objeto (***no hay setters)***

```
public static void main(String[] args)
{
    Punto p1 = new Punto( 5, 5 );

    System.out.println( "La distancia de "
                    + p1 + " a "
                    + Punto.ORIGEN + " es "
                    + p1.calculaDistanciaOrigen() );

    p1 = new Punto( 10, 10 );

    System.out.println( "La distancia de "
                    + p1 + " a "
                    + Punto.ORIGEN + " es "
                    + p1.calculaDistanciaOrigen() );
```

Sobreescribe la referencia:
p1 referencia al punto (5,5)
p1 referencia al punto (10,10)
Recolector automático de basura en JAVA
En otros lenguajes como C++ no sería correcto

4. Setters

http://ideone.com/bf8Dmi

Usar setters cuando sea necesario

```
/** Representa a personas */
class Persona {
      private String nombre;
      private int dni;
      private int anhoNacimiento;

      /** Crea un nuevo objeto Persona
          @param dni El dni, como int
          @param nombre El nombre, como String
          @param anhoNacimiento El año de nacimiento, como int
      */
      public Persona(int dni, String nombre, int anhoNacimiento)
      {
            this.dni = dni;
            this.nombre = nombre;
            this.anhoNacimiento = anhoNacimiento;
      }
```

setNombre() ?

setDni() ?

setAnoNacimiento() ?

Usar getters devolver un valor que no es el de un atributo

```
/** @return La edad, como int */
public int getEdad()
{
      return Calendar.getInstance().get( Calendar.YEAR )
                  - this.getAnoNacimiento();
}
```

4.1. Getters/setters en lugar de atributos públicos

```
class Punto {
      public int x;            // ERROR: implementación de la clase expuesta
      public int y;            // ERROR: implementación de la clase expuesta
}
```

No escala bien:

- Código que use la clase Punto va a estar fuertemente acoplado a la implementación de la clase
- Cambios no sólo en la clase sino en el código que la usa

Ejemplo: si tuviéramos que cambiar coordenadas de int a double

Solución: **atributos siempre privados**

5. Sobrecarga de métodos

Varias versiones de un método que actúe en función de los parámetros que recibe

JAVA: varios métodos con el mismo nombre, siempre y cuando se diferencien en el tipo y/o número de parámetros.

Constructor de un clase → tipo de métodos

```
/** Crea un candado, asignando una clave.
  * @param k la clave, como entero */
public Candado(int k)
{
      clave = k;
      estado = false;
}

/** Crea un candado, asignando una clave.
  * @param a Primer dígito de la clave, como entero.
  * @param b Segundo dígito de la clave, como entero.
  * @param c Tercer dígito de la clave, como entero. */
public Candado(int a, int b, int c)
{
      clave = ( ( a % 10 ) * 100 ) + ( ( b % 10 ) * 10 ) + ( c % 10 );
      estado = false;
}
```

```
Candado c1 = new Candado( 1, 2, 3 );
c1.toString();
```

5. Sobrecarga de métodos

```
/** Abre el candado
  * @param k La clave a probar, como entero */
public void abre(int k)
{
    if ( clave == k ) {
        estado = true;
    }
}

/** Abre el candado
  * @param a Primer dígito de la clave, como entero.
  * @param b Segundo dígito de la clave, como entero.
  * @param c Tercer dígito de la clave, como entero. */
public void abre(int a, int b, int c) {
    if ( clave == convierteClave( a, b, c ) ) {
        estado = true;
    }
}
/**
 * Convierte una clave proporcionada en tres dígitos a un solo entero.
 * @param a Primer dígito de la clave, como entero.
 * @param b Segundo dígito de la clave, como entero.
 * @param c Tercer dígito de la clave, como entero. */
private int convierteClave(int a, int b, int c)
{
    return ( ( a % 10 ) * 100 ) + ( ( b % 10 ) * 10 ) + ( c % 10 );
}
```

6. Vectores y matrices primitivos

Crear mediante el operador *new:* devuelve una referencia

Constante pública *length:* devuelve la longitud del vector

Posiciones del vector indexadas desde cero (última posición *length-1*

```
int suma = 0;
int[] v = new int[]{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

for(int i = 0; i < v.length; ++i) {
        int x = v[ i ];
        suma += x;
}
```

```
int[ ] v = new int[10];
int suma = 0;
for (int i = 0; i < v.length; ++i) {
        v[ i ] = i+1;
        suma += v[ i ];
}
```

Ejemplo Clase Candado:
http://ideone.com/BTtDzh

7. Miembros static

Atributos pertenecientes a la clase

P.e. contabilizar el número de objetos de la clase creados

```
/** Representa un candado con clave */
class Candado {
    private static int numCandados = 0;

    private int clave;
    private boolean estado;

    /** Crea un candado, asignando una clave.
      * @param k La clave, como entero */
    public Candado(int k) {
        clave = k;
        estado = false;
        ++numCandados;
    }

    /** Devuelve el num. de candados creados hasta el momento.
     *  @return El num. de candados, como entero.
     */
    public static int getNumCandados() {
        return numCandados;
    }
```

8. Enumerados

Tipo de dato definido por el usuario que solo puede tomar como valores los definidos en una lista consecutiva de constantes :

[modificador] enum TipoEnumerado {VALOR1, VALOR2, ...};

- Constantes se indican en mayúsculas
- Se les asigna un valor que comienza en cero y se va incrementando de izquierda a derecha
- Para hacer referencia a las constantes es necesario cualificarlas con el nombre del tipo

```
Class Candado {
    public static enum Estado { CERRADO, ABIERTO };
    private Estado estado;
    ... }
```

```
Estado.CERRADO
Candado.Estado.CERRADO
```

8. Enumerados

http://ideone.com/BTtDzh

+: Tipos enumerados son mucho más informativos

-: No se pueden leer directamente

Métodos disponibles sobre enumerados:

- *values()*: devuelve un vector de Strings que contiene todas las constantes de la enumeración
- *name()*: devuelve un String con el nombre de la constante que contiene
- *ordinal()*: devuelve un entero con la posición de la constante
- *toString()*: devuelve un String con el nombre de la constante que contiene
- *equals()*: devuelve true si el valor de la variable enum es igual al objeto que recibe

10. String

Las cadenas de caracteres en Java no son tipos primitivos, sino objetos de la clase **String**

```
String s = new String( “hola” ); // es lo mismo que String s = “hola”;
```

Cada cadena en java es un objeto y es **inmutable**: cualquier operación que se realice sobre una cadena devuelve una nueva cadena, no modifica la cadena original

```
String s = "  ESEI  ";
System.out.println(s.trim());
System.out.println(s);
```

s no se modifica

```
String s = "  ESEI  ";
s = s.trim();
System.out.println(s);
```

es necesario hacerlo explícitamente

10. String

String s = " ESEI ";

Método	Explicación	Ejemplo
trim()	Devuelve una nueva cadena eliminando los espacios al comienzo y al final	s = s.trim(); // “ESEI”
charAt(i)	Devuelve el carácter en la posición dada.	char c = s.charAt(0); // c == 'E'
toUpperCase()	Devuelve una nueva cadena conviertiendo los caracteres a mayúsculas.	String s2 = s.toUpperCase(); // “ESEI”
toLowerCase()	Devuelve una nueva cadena conviertiendo los caracteres a minúsculas.	String s3 = s.toLowerCase(); // “esei”
equals()	Devuelve true si dos cadenas son iguales.	s2.equals(s3); // false

10. String

String s = " ESEI ";

Método	Explicación	Ejemplo
equalsIgnoreCase()	Devuelve true si dos cadenas son iguales, ignorando mayúsculas y minúsculas.	s2.equalsIgnoreCase(s3); // true
length()	Devuelve la longitud de la cadena.	s2.length(); // 4
indexOf(c)	Devuelve la posición de c en la cadena.	s2.indexOf('E'); // 0 s2.indexOf('j'); // -1
concat(s)	Concatena dos cadenas.	s = s2.concat(s3); // s es “ESEIesei”
split(expreg)	Divide la cadena en función de las coincidencias de la expresión regular dada. Devuelve un vector de String	String s3 = “Escola Superior de Enxeñaría Informática”; String s4 = s3.split(“ “)[0] // S4 es “Escola”

10. String

```
public class Ppal {
    public static String lista(Candado[] candados)
    {
        String toret = "";

        for(int i = 0; i < candados.length; ++i) {
            toret += candados[ i ].toString() + '\n';
        }

        return toret;
    }
}
```

```
public class Ppal {
    public static String lista(Candado[] candados)
    {
        StringBuilder toret = new StringBuilder();

        for(int i = 0; i < candados.length; ++i) {
            toret.append( candados[ i ].toString() );
            toret.append( '\n' );
        }

        return toret.toString();
    }
}
```

String

inmutable:

crean objetos intermedios que ocupan memoria (recolección)

StringBuilder

mutable:

es modificada cada vez que se hace una operación sobre ella