

Ocultamiento de información

Jorge I. Meza

jimezam@autonoma.edu.co



Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

This license requires that reusers give credit to the creator. It allows reusers to distribute, remix, adapt, and build upon the material in any medium or format, for noncommercial purposes only. If others modify or adapt the material, they must license the modified material under identical terms.

① **BY:** Credit must be given to you, the creator.

Ⓜ **NC:** Only noncommercial use of your work is permitted.

Noncommercial means not primarily intended for or directed towards commercial advantage or monetary compensation.

Ⓢ **SA:** Adaptations must be shared under the same terms.

Contenidos

- Motivación
- Cuál es el problema
- Concepto de ocultamiento de información
- Modificadores (4p)
- Caso de ejemplo
 - Acceso directo
 - Con ocultamiento
 - Con *setters* y *getters*
 - Con validaciones

Motivación

- Actualmente permitimos modificar arbitrariamente los atributos de los objetos que creamos e invocarle todos sus métodos sin restricción.
- ¿Podría ser esto un problema de "seguridad"?
- Entendiendo *seguridad* como permitir un mal uso y con este, la pérdida de consistencia con la realidad modelada.

Ejemplo

```
class Fecha {  
    int dia;  
    int mes;  
    int año;  
}
```

- $1 \leq \text{dia} \leq 31$
- $1 \leq \text{mes} \leq 12$
- $\text{año} \geq 0$

¡Se ve bien! 🤔

El nacimiento de Shakira

```
Fecha shaki = new Fecha();
```

```
shaki.dia = 3;
```

```
shaki.mes = 2;
```

```
shaki.año = 1977;
```



El nacimiento de Shakira

```
Fecha shaki = new Fecha();
```

```
shaki.dia = -3;  
shaki.mes = 200;  
shaki.año = -75;
```



Los desarrolladores malos

El nacimiento de Shakira

```
Fecha shaki = new Fecha();
```

```
shaki.dia = -3;  
shaki.mes = 200;  
shaki.año = -75;
```

¿Es **válida** esta asignación?



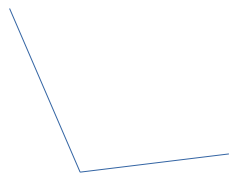
Los malos desarrolladores

¿Es *válida* esta asignación?

- Desde el punto de vista computacional, **si**. Tanto `dia`, `mes` y `año` son atributos de tipo `int` y su rango es `-2.147.483.648` a `2.147.483.647`.
- Desde el punto de vista del concepto Fecha (lógica de negocio), **no**. Los días deben estar entre 1 y 31, los meses entre 1 y 12, y los años deben ser mayores a cero.

¿Es válida esta asignación?

- Si, compila sin errores.
- No nos sirve, ya que la fecha "-3/200/-75" no es válida.



Nuestro modelo (clase) se puede vulnerar (hacer funcionar mal). En pocas palabras, está mal hecho.

¿Cómo podemos arreglar esta situación?



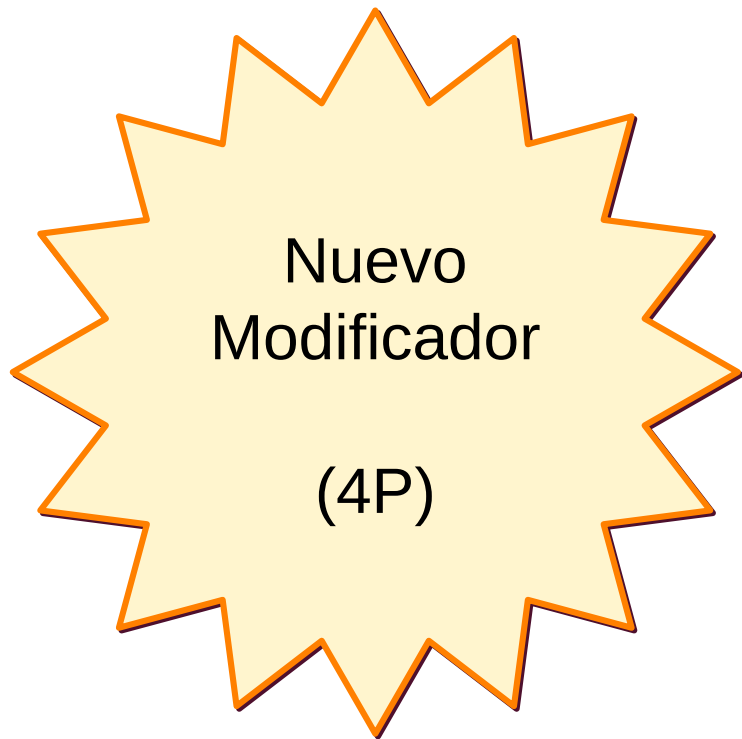
Ocultamiento de información



Ocultamiento de información

- Es la característica de los lenguajes orientados a objetos modernos que le permite al desarrollador determinar "quien puede acceder" a los diferentes miembros de una clase.
- Recuérdesse que al hablar de *miembros* de una clase, estamos hablando de *atributos* y *métodos* de manera indistinta.

Ocultamiento de información



- `public`
- *"package"*
- `protected`
- `private`

Modificador `public`

- "Todo el mundo" puede acceder o invocar a ese miembro.
- Por "todo el mundo" consideramos a cualquier objeto, de cualquier clase, en cualquier paquete.

Modificador "*package*"

- Es similar al `public`, sólo que estos miembros no pueden ser accedidos o invocados por *todos* sino solamente por los objetos, instancias de las clases que estén en el mismo paquete.
- El modificador "*package*" no existe como palabra reservada, en Java si un miembro no tiene un modificador específico, por defecto se considera "*package*".

Modificador `private`


- "Nadie" puede acceder o invocar a ese miembro, salvo su propietario.

Modificador `protected`

- "Nadie" puede acceder o invocar a ese miembro, salvo su propietario y sus hijos, es decir, los objetos de las clases que hereden de esta clase en cuestión.

Ejemplo


```
class Camion {  
    public String placa;  
    String color;  
    private double velocidad;  
    protected int cantidadEjes;  
}
```



Nótese como `color` no tiene un modificador de ocultamiento explícito, entonces es "*package*".

Ejemplo

```
class Camion {  
    public double acelerar(double velocidad) {}  
    void frenar(double velocidad) {}  
    private void encender() {}  
    protected void apagar() {}  
}
```



Nótese como `frenar(double)` no tiene un modificador de ocultamiento explícito, entonces es "*package*".

Volviendo al problema

```
Fecha shaki = new Fecha();
```

```
shaki.dia = -3;  
shaki.mes = 200;  
shaki.año = -75;
```



Los desarrolladores malos

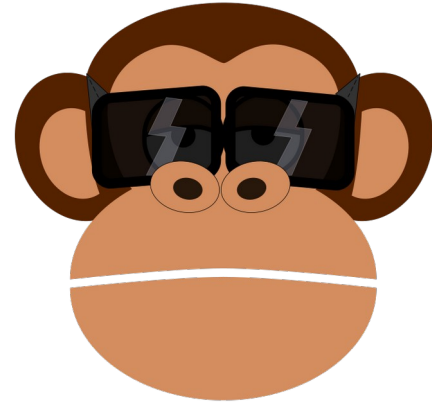
Ejemplo

```
class Fecha {  
    int dia;  
  
    int mes;  
  
    int año;  
}
```

¿Cómo podemos
proteger a shaki?

Ejemplo

```
class Fecha {  
    private int dia;  
    private int mes;  
    private int año;  
}
```



Protección total

El problema ya no existe

```
Fecha shaki = new Fecha();
```

```
shaki.dia = -3; ❌
```

```
shaki.mes = 200; ❌
```

```
shaki.año = -75; ❌
```



Los desarrolladores malos

Tenemos un nuevo problema

```
Fecha shaki = new Fecha();
```

```
shaki.dia = 3; ❌
```

```
shaki.mes = 2; ❌
```

```
shaki.año = 1977; ❌
```



Tenemos un nuevo problema

- Al hacer los atributos `private`, ya no se pueden acceder directamente.
- El acceso directo queda bloqueado, tanto para valores inválidos como para válidos.
- ¿De qué sirve un atributo que no se puede consultar o modificar?
- ¿Qué podemos hacer?

Recordemos los tipos de métodos

- Constructores
- ~~Destructores~~
- ***Setters* (modificadores)**
- ***Getters* (analizadores)**
- Otros métodos definidos por el usuario

Ejemplo

```
class Fecha {  
    private int dia;  
    private int mes;  
    private int año;  
  
    public void setDia(int d) {  
        dia = d;  
    }  
  
    public int getDia() {  
        return dia;  
    }  
}
```

Ejemplo

```
class Fecha {  
    private int dia;  
    private int mes;  
    private int año;  
  
    public void setDia(int dia) {  
        this.dia = dia;  
    }  
  
    public int getDia() {  
        return dia;  
    }  
}
```

Ejemplo

```
class Fecha {  
    private int dia;  
    private int mes;  
    private int año;  
  
    public void setDia(int dia) {  
        this.dia = dia;  
    }  
  
    public int getDia() {  
        return dia;  
    }  
}
```

```
    public void setMes(int mes) {  
        this.mes = mes;  
    }  
  
    public int getMes() {  
        return mes;  
    }  
  
    public void setAño(int año) {  
        this.año = año;  
    }  
  
    public int getAño() {  
        return año;  
    }  
}
```

Utilizando *setters* y *getters*

```
Fecha shaki = new Fecha();
```

```
shaki.setDia(3);
```

```
shaki.setMes(2);
```

```
shaki.setAño(1977);
```

```
System.out.println("Shakira  
nació en: " + shaki.getAño());
```



¿Solucionamos el problema ... o no?



El problema sigue existiendo

```
Fecha shaki = new Fecha();
```

```
shaki.setDia(-3);  
shaki.setMes(200);  
shaki.setAño(-75);
```



Los desarrolladores malos

¿Qué hacemos?

¿Propuestas?



¡Validaciones!

Todo dato externo que entre al programa,
¡debe ser validado!

Lo que sabemos

```
class Fecha {  
    int dia;  
    int mes;  
    int año;  
}
```

- $1 \leq \text{dia} \leq 31$
- $1 \leq \text{mes} \leq 12$
- $\text{año} \geq 0$

¡Se ve bien! 🤔

Mejorando nuestro código

```
class Fecha {  
  
    private int dia;  
    private int mes;  
    private int año;  
  
    public void setDia(int dia) {  
        if(dia >= 1 & dia <= 31)  
            this.dia = dia;  
    }  
}
```

```
    public void setMes(int mes) {  
        if(dia >= 1 & dia <= 12)  
            this.mes = mes;  
    }  
  
    public void setAño(int año) {  
        if(dia >= 0)  
            this.año = año;  
    }  
  
    // ... getters ...  
}
```

Mejorando nuestro código ... aún más

```
class Fecha {  
  
    private int dia;  
    private int mes;  
    private int año;  
  
    public boolean setDia(int dia) {  
        if(dia < 1 | dia > 31)  
            return false;  
  
        this.dia = dia;  
        return true;  
    }  
}
```

```
        public boolean setMes(int mes) {  
            if(dia < 1 | dia > 12)  
                return false;  
  
            this.mes = mes;  
            return true;  
        }  
  
        public boolean setAno(int ano) {  
            if(dia < 0)  
                return false;  
  
            this.ano = ano;  
            return true;  
        }  
  
        // ... getters ...  
    }  
}
```

El problema se soluciona elegantemente

```
Fecha shaki = new Fecha();
```

```
shaki.setDia(-3);
```

```
shaki.setMes(200);
```

```
shaki.setAño(-75);
```

Como los datos son inválidos, los *setters* retornan *false* y no modifican los atributos, protegiéndolos.



Los desarrolladores malos

El problema se soluciona elegantemente

```
Fecha shaki = new Fecha();
```

```
shaki.setDia(3);
```

```
shaki.setMes(2);
```

```
shaki.setAño(1977);
```

Como los datos son válidos, los *setters* retornan `true` y modifican los atributos



El problema se soluciona elegantemente

```
Fecha miFecha = new Fecha();

boolean control = miFecha.setDia(dia);

if(control)
    System.out.println("Cambio de día exitoso");
else
    System.out.println("Error, el día es inválido,
        inténtelo nuevamente");
```



¿Alguna

pregunta?