



# TEMA 4

## 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO



## 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

### Código Autoexplicativo. Formato, Comentarios y Nombrado

#### Legibilidad

Software	autoexplicativo	consistente	y mínimo
<p>“Una línea de código se escribe una vez y se lee cientos de veces</p> <p>— Tom Love</p>	<ul style="list-style-type: none"><li>• Formato</li><li>• Comentarios</li><li>• Nombrado</li></ul>	<ul style="list-style-type: none"><li>• Estándares</li><li>• Consistencia</li><li>• Alertas</li></ul>	<ul style="list-style-type: none"><li>• Código Muerto</li><li>• YAGNI</li><li>• DRY</li></ul>

# 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

## Código Autoexplicativo. Formato, Comentarios y Nombrado

### Legibilidad

Software	autoexplicativo	consistente	y mínimo
“Una línea de código se escribe una vez y se lee cientos de veces” — Tom Love	<ul style="list-style-type: none"><li>• Formato</li><li>• Comentarios</li><li>• Nombrado</li></ul>	<ul style="list-style-type: none"><li>• Estándares</li><li>• Consistencia</li><li>• Alertas</li></ul>	<ul style="list-style-type: none"><li>• Código Muerto</li><li>• YAGNI</li><li>• DRY</li></ul>

### Formato

Justificación	Implicaciones	Violaciones <a href="#">TicTacToe</a>
<ul style="list-style-type: none"><li>• Formateo de código es importante. Es <b>demasiado importante como para ignorarlo y es demasiado importante como para tratarlo religiosamente</b>. El formateo de código trata sobre la comunicación y la comunicación es de primer orden para los desarrolladores profesionales</li></ul> <p>“Un equipo de desarrolladores debe ponerse de acuerdo sobre <b>un único estilo de formato</b> y luego todos los miembros de ese equipo debe usar ese estilo.”</p> <p>— Martin R.</p>	<ul style="list-style-type: none"><li>• Un código es una jerarquía. Hay información que pertenece al archivo como un todo, a las clases individuales dentro del archivo, a los métodos dentro de las clases, a los bloques dentro de los bloques. Cada nivel de esta jerarquía es un ámbito en el que los nombres pueden ser declarados y en la que las declaraciones y sentencias ejecutables se interpretan. Para hacer esta jerarquía visible, hay que <b>sangrar la líneas de código fuente de forma proporcional a su posición en la jerarquía</b>.</li><li>• Utilizamos el <b>espacio en blanco horizontal para asociar</b> las cosas que están fuertemente relacionadas y disociar las cosas que están más débilmente relacionadas y para acentuar la precedencia de operadores</li><li>• Una <b>línea entre grupos lógicos</b> (atributos y cada método).</li><li>• Los <b>atributos deben declararse al principio</b> de la clase</li><li>• Las <b>funciones dependientes</b> en que una llama a otra, deberían estar verticalmente cerca: primero la llamante y luego la llamada</li><li>• <b>Grupos de funciones</b> que realizan operaciones parecidas, deberían permanecer juntas</li><li>• Las variables deberían declararse tan cerca como sea posible de su utilización, <b>minimizar el intervalo de vida de una variable</b></li><li>• Los programadores prefieren <b>líneas cortas</b> (~40, máximo 80/120)</li></ul>	<ul style="list-style-type: none"><li>• <b>No uses tabuladores</b> entre los tipos y las variables para una disposición por columnas</li><li>• <b>Nunca rompas las reglas de sangrado</b> por muy pequeñas que sean las líneas</li><li>• ...</li></ul>

# 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

## Código Autoexplicativo. Formato, Comentarios y Nombrado

### Legibilidad

Software	autoexplicativo	consistente	y mínimo
“Una línea de código se escribe una vez y se lee cientos de veces” — Tom Love	<ul style="list-style-type: none"><li>• Formato</li><li>• Comentarios</li><li>• Nombrado</li></ul>	<ul style="list-style-type: none"><li>• Estándares</li><li>• Consistencia</li><li>• Alertas</li></ul>	<ul style="list-style-type: none"><li>• Código Muerto</li><li>• YAGNI</li><li>• DRY</li></ul>

### Comentarios

- Justificación	- Implicaciones	- Violaciones <a href="#">TicTacToe</a>
<ul style="list-style-type: none"><li>• Nada puede ser tan útil como un comentario bien colocado.</li><li>• Nada puede ser tan perjudicial como un enrevesado comentario desactualizado que propaga mentiras y desinformación</li><li>• Nada puede estorbar más encima de un módulo que frívolos <b>comentarios dogmáticos</b>.<ul style="list-style-type: none"><li>◦ Es simplemente una tontería tener una regla que dice que cada variable debe tener un comentario o que cada función debe tener un javadoc a a no ser que sea publicado como biblioteca#</li></ul></li></ul> <p>“No comentes código malo, reescribelo” — Kernighan &amp; Plaugher</p>	<ul style="list-style-type: none"><li>• <b>Comentario legal:</b><ul style="list-style-type: none"><li>◦ <i>copyright, license,</i></li></ul></li><li>• <b>Comentario aclarativo:</b><ul style="list-style-type: none"><li>◦ <i>// format matched</i> <i>kk:mm:ssEEE, MMM</i> <i>dd, yyyy;</i></li><li>◦ <i>String format =</i> <i>"  d*:  d*:  d*  w*;</i> <i>  w*  d*  d*";</i></li></ul></li><li>• Código claro y expresivo con algunos <b>pocos comentarios</b> es muy superior al código desordenado y complejo con un montón de comentarios. En muchos casos es simplemente una cuestión de <b>crear un método, clase, ... con el nombre que diga lo mismo que el comentario.</b></li></ul>	<ul style="list-style-type: none"><li>• <b>Comentarios redundantes.</b><ul style="list-style-type: none"><li>◦ <i>intdayOfMonth; //the day of themonth,</i></li></ul></li><li>• <b>Comentarios de sección.</b><ul style="list-style-type: none"><li>◦ <i>//_Actions////////////////////////////////_</i></li></ul></li><li>• <b>Comentarios no mantenidos</b>, con valores que no se actualizará.<ul style="list-style-type: none"><li>◦ <i>// portis7077</i></li></ul></li><li>• <b>Comentarios excesivos</b>, como el historial de interesantes discusiones de diseño</li><li>• <b>Comentarios confusos.</b> Si nuestro único recurso es examinar el código en otras partes del sistema para averiguar lo que está pasando.</li><li>• <b>Comentarios inexactos.</b> Un programador hace una declaración en sus comentarios que no es lo suficientemente precisa para ser exacta</li><li>• <b>Comentarios de atribución</b>, cuando para eso está el control de versiones cuando haga realmente falta<ul style="list-style-type: none"><li>◦ <i>// Added by Luis</i></li></ul></li><li>• <b>Código comentado</b>, cuando para eso está el control de versiones</li><li>• ...</li></ul>

## 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

### Código Autoexplicativo. Formato, Comentarios y Nombrado

#### Legibilidad

Software	autoexplicativo	consistente	y mínimo
“Una línea de código se escribe una vez y se lee cientos de veces” — Tom Love	<ul style="list-style-type: none"><li>• Formato</li><li>• Comentarios</li><li>• Nombrado</li></ul>	<ul style="list-style-type: none"><li>• Estándares</li><li>• Consistencia</li><li>• Alertas</li></ul>	<ul style="list-style-type: none"><li>• Código Muerto</li><li>• YAGNI</li><li>• DRY</li></ul>

#### Nombrado

Sinónimos	Synonyms	Libro	Autor
Elige nombres descriptivos	Choose descriptive names	Clean Code	Robert Martin
Elige nombres al nivel de abstracción apropiado	Choose names at the appropriate level of abstraction	Clean Code	Robert Martin
Usa nomenclatura estándar donde sea posible	Use standard nomenclature where possible	Clean Code	Robert Martin
Nombre no ambiguos	Unambiguous name	Clean Code	Robert Martin
Usa nombres largos para ámbitos largos	Use long names for long scopes	Clean Code	Robert Martin
Evita codificaciones	Avoid Encodings	Clean Code	Robert Martin
Los nombre deberían describir los efectos laterales	The names should describe the side effects	Clean Code	Robert Martin

# 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

## Código Autoexplicativo. Formato, Comentarios y Nombrado

### Legibilidad

Software	autoexplicativo	consistente	y mínimo
“Una línea de código se escribe una vez y se lee cientos de veces” — Tom Love	<ul style="list-style-type: none"><li>• Formato</li><li>• Comentarios</li><li>• Nombrado</li></ul>	<ul style="list-style-type: none"><li>• Estándares</li><li>• Consistencia</li><li>• Alertas</li></ul>	<ul style="list-style-type: none"><li>• Código Muerto</li><li>• YAGNI</li><li>• DRY</li></ul>

Sinónimos	Synonyms	Libro	Autor
Elige nombres descriptivos	Choose descriptive names	Clean Code	Robert Martin
Elige nombres al nivel de abstracción apropiado	Choose names at the appropriate level of abstraction	Clean Code	Robert Martin
Usa nomenclatura estándar donde sea posible	Use standard nomenclature where possible	Clean Code	Robert Martin
Nombre no ambiguos	Unambiguous name	Clean Code	Robert Martin
Usa nombres largos para ámbitos largos	Use long names for long scopes	Clean Code	Robert Martin
Evita codificaciones	Avoid Encodings	Clean Code	Robert Martin
Los nombre deberían describir los efectos laterales	The names should describe the side effects	Clean Code	Robert Martin

- Los nombre deben revelar su intención.  
Deberían revelar por qué existe, qué hace, y cómo se utiliza para facilitar la legibilidad para el desarrollo y el mantenimiento correctivo, perfectivo y adaptativo
- La elección de buenos nombres lleva tiempo, pero ahorra más de lo que consume.
- Se pueden cambiar cuando encuentres otros mejores.
- El código se debe leer como si fueran párrafos y oraciones.

## 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

### Código Autoexplicativo. Formato, Comentarios y Nombrado



Legibilidad

Software	autoexplicativo	consistente	y mínimo
<i>“Una línea de código se escribe una vez y se lee cientos de veces”</i> — Tom Love	<ul style="list-style-type: none"><li>• Formato</li><li>• Comentarios</li><li>• Nombrado</li></ul>	<ul style="list-style-type: none"><li>• Estándares</li><li>• Consistencia</li><li>• Alertas</li></ul>	<ul style="list-style-type: none"><li>• Código Muerto</li><li>• YAGNI</li><li>• DRY</li></ul>

Nombrado

Sinónimos	Synonyms	Libro	Autor
Elige nombres descriptivos	Choose descriptive names	Clean Code	Robert Martin
Elige nombres al nivel de abstracción apropiado	Choose names at the appropriate level of abstraction	Clean Code	Robert Martin
Usa nomenclatura estándar donde sea posible	Use standard nomenclature where possible	Clean Code	Robert Martin
Nombre no ambiguos	Unambiguous name	Clean Code	Robert Martin
Usa nombres largos para ámbitos largos	Use long names for long scopes	Clean Code	Robert Martin
Evita codificaciones	Avoid Encodings	Clean Code	Robert Martin
Los nombre deberían describir los efectos laterales	The names should describe the side effects	Clean Code	Robert Martin

- Los nombres deben de ser **pronunciables** que permitan mantener una conversación
- **Mayúsculas** en los caracteres **inicio de palabra** (CamelCase).
- Nombres del **dominio del problema** y de la **solución**.
- Elige una **palabra** para un **concepto abstracto** y aferrarte a él.
- Nombres de **paquetes** deben ser **sustantivos** y comenzar en **minúsculas**. (models.customers)
- Nombres de **clases** deben ser **sustantivos** y comenzar en **mayúsculas**. (Controller, no Control)
- Nombres de **métodos** deben ser verbos o una frase con verbo y comenzar en minúsculas.
- Nombres de **métodos** de acceso deben anteponer **get**(is para lógicos) y **/set** o **put**

## 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

### Código Autoexplicativo. Formato, Comentarios y Nombrado

#### Legibilidad

Software	autoexplicativo	consistente	y mínimo
<p>“Una línea de código se escribe una vez y se lee cientos de veces</p> <p>— Tom Love</p>	<ul style="list-style-type: none"><li>• Formato</li><li>• Comentarios</li><li>• Nombrado</li></ul>	<ul style="list-style-type: none"><li>• Estándares</li><li>• Consistencia</li><li>• Alertas</li></ul>	<ul style="list-style-type: none"><li>• Código Muerto</li><li>• YAGNI</li><li>• DRY</li></ul>

#### Nombrado

Sinónimos	Synonyms	Libro	Autor
Elige nombres descriptivos	Choose descriptive names	Clean Code	Robert Martin
Elige nombres al nivel de abstracción apropiado	Choose names at the appropriate level of abstraction	Clean Code	Robert Martin
Usa nomenclatura estándar donde sea posible	Use standard nomenclature where possible	Clean Code	Robert Martin
Nombre no ambiguo	Unambiguous name	Clean Code	Robert Martin
Usa nombres largos para ámbitos largos	Use long names for long scopes	Clean Code	Robert Martin
Evita codificaciones	Avoid Encodings	Clean Code	Robert Martin
Los nombres deberían describir los efectos laterales	The names should describe the side effects	Clean Code	Robert Martin

- Si un nombre **requiere un comentario**, el nombre no revela su intención.
- Utilizar **separadores de palabras** como guiones o subrayados en la comunidad Java, en otra comunidad, depende!!!
- **Constantes numéricas** que son difíciles de localizar y mantener
- Nombres de **una letra** y muy en particular, ‘O’ y ‘I’ que se confunden con 0 y 1.
- Un **contador de bucle** puede ser nombrado i o j o k (pero nunca !!) si su alcance es muy pequeño y no hay otros nombres que pueden entrar en conflicto con él.
- Nombres **acrónimos** a no ser que sean internacionales.
- Nombres con **códigos de tipo o información del ámbito** (notación Húngara y similares). Ej.: `int iAge` -> `age`;
- Nombre con **palabras vacías de significado** o redundantes como Object, Class, Data, Inform.
- Nombre **en serie** `player1, player2, ... mejor players o winnerPlayer y loserPlayer`
- Nombres **desinformativos** que no son lo que dicen.
- Nombres **indistinguibles** (`XYZControllerForEfficientHandlingOfStrings` y `XYZControllerForEfficientStorageOfStrings`)
- Nombres **polisémicos** en un mismo contexto. (`book` como registro en un hotel y libro;)



## 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

### Código Consistente. Estándares y Alertas

#### Legibilidad

Software	autoexplicativo	consistente	y mínimo
<p>“Una línea de código se escribe una vez y se lee cientos de veces</p> <p>— Tom Love</p>	<ul style="list-style-type: none"><li>• Formato</li><li>• Comentarios</li><li>• Nombrado</li></ul>	<ul style="list-style-type: none"><li>• Estándares</li><li>• Consistencia</li><li>• Alertas</li></ul>	<ul style="list-style-type: none"><li>• Código Muerto</li><li>• YAGNI</li><li>• DRY</li></ul>

# 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

## Código Consistente. Estándares y Alertas

### Legibilidad

Software	autoexplicativo	consistente	y mínimo
“Una línea de código se escribe una vez y se lee cientos de veces” — Tom Love	<ul style="list-style-type: none"><li>• Formato</li><li>• Comentarios</li><li>• Nombrado</li></ul>	<ul style="list-style-type: none"><li>• Estándares</li><li>• Consistencia</li><li>• Alertas</li></ul>	<ul style="list-style-type: none"><li>• Código Muerto</li><li>• YAGNI</li><li>• DRY</li></ul>

### Estándares

Sinónimos	Synonyms	Libro	Autor
Seguir las convenciones estándar	Follow Standard Conventions	Clean Code	Robert Martin
<ul style="list-style-type: none"><li>• Siga las <b>convenciones estándar basadas en normas comunes de la industria</b></li><li>• Cada miembro del equipo debe ser lo suficientemente maduro como para darse cuenta de que no importa un ápice donde pones tus llaves, siempre y cuando <b>todos estén de acuerdo</b> en dónde ponerlos.</li></ul>		<ul style="list-style-type: none"><li>• Debe especificar cosas como <b>dónde declarar</b> variables de instancia; <b>cómo nombrar</b> las clases, métodos y variables; <b>dónde poner paréntesis</b>, llaves; ...</li><li>• <b>No se necesita un documento</b> para describir estos convenios porque su código proporciona los ejemplos.</li></ul>	

# 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

## Código Consistente. Estándares y Alertas

### Legibilidad

Software	autoexplicativo	consistente	y mínimo
“Una línea de código se escribe una vez y se lee cientos de veces” — Tom Love	<ul style="list-style-type: none"><li>• Formato</li><li>• Comentarios</li><li>• Nombrado</li></ul>	<ul style="list-style-type: none"><li>• Estándares</li><li>• Consistencia</li><li>• Alertas</li></ul>	<ul style="list-style-type: none"><li>• Código Muerto</li><li>• YAGNI</li><li>• DRY</li></ul>

### Consistencia

Sinónimos	Synonyms	Libro	Autor
Inconsecuencia	Inconsistency	Clean Code	Robert Martin

Justificación	Ejemplos
<ul style="list-style-type: none"><li>• Si haces algo de cierta manera, <b>haz todas las cosas similares de la misma forma</b><ul style="list-style-type: none"><li>◦ Una simple consistencia como esta, cuando se aplica de forma fiable, se puede conseguir <b>código más fácil de leer y modificar</b>.</li><li>◦ Tenga <b>cuidado con los convenios</b> que decides, y una vez elegido, tenga cuidado de seguirlos.</li></ul></li></ul>	<ul style="list-style-type: none"><li>• Si dentro de una función se utiliza una variable "interval", a continuación, utilizar el mismo nombre de variable en las otras funciones que utilizan variables "interval".</li><li>• Si se nombra un método "processVerificationRequest", a continuación, utiliza un nombre similar, como "processDeletionRequest", para los métodos que procesan otros tipos de solicitudes.</li></ul>

# 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

## Código Consistente. Estándares y Alertas

Alertas

Legibilidad

Software	autoexplicativo	consistente	y mínimo
“Una línea de código se escribe una vez y se lee cientos de veces” — Tom Love	<ul style="list-style-type: none"><li>• Formato</li><li>• Comentarios</li><li>• Nombrado</li></ul>	<ul style="list-style-type: none"><li>• Estándares</li><li>• Consistencia</li><li>• Alertas</li></ul>	<ul style="list-style-type: none"><li>• Código Muerto</li><li>• YAGNI</li><li>• DRY</li></ul>

Sinónimos	Synonyms	Libro	Autor
Seguridad anulada	Overridden Safeties	Clean Code	Robert Martin

Justificación	Ejemplos
<ul style="list-style-type: none"><li>• Es <b>arriesgado desactivar ciertas advertencias</b> del compilador (o todas las advertencias!) aunque puede ayudarle a obtener la sensación de tener éxito pero con el riesgo de sesiones de depuración sin fin.</li></ul>	<ul style="list-style-type: none"><li>• el desastre de Chernobylse debió a que el gerente de la planta hizo caso omiso de cada uno de los mecanismos de seguridad de uno en uno. Los dispositivos de seguridad estaban siendo inconvenientes para ejecutar un experimento. El resultado fue que el experimento no consiguió realizarse y el mundo vio su primera gran catástrofe civil nuclear.</li></ul>

## 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

### Código Mínimo. Código Muerto, YAGNI y DRY

#### Legibilidad

Software	autoexplicativo	consistente	y mínimo
<p>“Una línea de código se escribe una vez y se lee cientos de veces</p> <p>— Tom Love</p>	<ul style="list-style-type: none"><li>• Formato</li><li>• Comentarios</li><li>• Nombrado</li></ul>	<ul style="list-style-type: none"><li>• Estándares</li><li>• Consistencia</li><li>• Alertas</li></ul>	<ul style="list-style-type: none"><li>• Código Muerto</li><li>• YAGNI</li><li>• DRY</li></ul>

# 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

Código Mínimo.  
Código Muerto,  
YAGNI y DRY

Legibilidad

Software	autoexplicativo	consistente	y mínimo
🔗		<ul style="list-style-type: none"><li>Estándares</li><li>Consistencia</li><li>Alertas</li></ul>	<ul style="list-style-type: none"><li>Código Muerto</li><li>YAGNI</li><li>DRY</li></ul>

Código Muerto

Sinónimos	Synonyms	Libro	Autor
Flujo de lava	Lava Flow		

Justificación	Implicaciones	Violaciones <a href="#">TicTacToe</a>
<ul style="list-style-type: none"><li>Según el código muerto se anquilosa y se endurecen, rápidamente se hace <b>imposible documentar</b> el código o entender suficientemente su arquitectura para hacer mejoras.</li><li>Si no se elimina el código muerto, <b>puede continuar proliferando</b> según se reutiliza código en otras áreas</li><li>Puede haber crecimiento exponencial según los sucesivos desarrolladores, demasiado apremiados o intimidados por analizar los códigos originales, seguirán produciendo <b>nuevos flujos secundarios</b> en su intento de evitar los originales.</li></ul>	<ul style="list-style-type: none"><li><b>Revisiones de código manual</b><ul style="list-style-type: none"><li>Obteniendo referencias mediante el entorno</li></ul></li><li><b>Herramientas de calidad por análisis estático</b><ul style="list-style-type: none"><li>Understad, Metrics, SonarQube, ...</li></ul></li></ul>	<ul style="list-style-type: none"><li>Fragmentos de código injustificables, inexplicables u obsoletas en el sistema: interfaces, clases, funciones o segmentos de código complejo con aspecto importante pero que <b>no están relacionados con el sistema</b></li><li>Bloques de <b>código comentado</b> sin explicación o documentación</li><li>Bloques de <b>código con comentarios</b><ul style="list-style-type: none"><li>//TODO: “proceso de cambio”, “para ser reemplazado”, ...</li></ul></li></ul>

# 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

Código Mínimo.  
Código Muerto,  
YAGNI y DRY

## Legibilidad

### YAGNI

Sinónimos	Synonyms	Libro	Autor
No lo vas a necesitar	You aren't going to need it o You ain't gonna need it		
Generalidad Especulativa	Speculative Generality	Smell Code(Refactoring)	Martin Fowler

Software	autoexplicativo	consistente	y mínimo
“	autoexplicativo	• Estándares • Consistencia • Alertas	• Código Muerto • YAGNI • DRY

Justificación	Implicaciones	Violaciones <a href="#">TicTacToe</a>
<ul style="list-style-type: none"><li>Las <b>características innecesarias son inconveniente</b> por:<ul style="list-style-type: none"><li>El <b>tiempo gastado se toma para la adición, la prueba o la mejora de funcionalidad innecesaria</b>. Y posteriormente, las nuevas características <b>deben depurarse, documentarse y mantenerse</b>.</li><li>Conduce a la <b>hinchazón de código</b> y el software se hace más grande y más complicado. Añadir nuevas características puede sugerir otras nuevas características. Si estas nuevas funciones se implementan así, esto podría resultar en un <b>efecto bola de nieve</b></li></ul></li></ul>	<ul style="list-style-type: none"><li>Siempre se implementan cosas cuando realmente se necesitan, no cuando se prevén que se necesiten. Por tanto, <b>no se debe agregar funcionalidad hasta que se considere estrictamente necesario</b>.</li></ul>	<ul style="list-style-type: none"><li>Hasta que la característica es realmente necesaria, es difícil definir completamente lo que debe hacer y probarla. <b>Si la nueva característica no está bien definida y probada, puede que no funcione correctamente, incluso si eventualmente se necesitara</b>. A menos que existan especificaciones y algún tipo de control de revisión, la función no puede ser conocida por los programadores que podrían hacer uso de ella.</li><li><b>Cualquier nueva característica impone restricciones en lo que se puede hacer en el futuro</b>, por lo que una característica innecesaria puede interrumpir características necesarias que se agreguen en el futuro.</li></ul>

# 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

Código Mínimo.  
Código Muerto,  
YAGNI y DRY

DRY

Legibilidad

Software	autoexplicativo	consistente	y mínimo
“Una línea de código se escribe	• Formato	• Estándares	• Código Muerto

Sinónimos	Sinónimos	Libro	Autor
No te repitas	DRY(Don't Repeat Yourself)		
Fuente Única de la Verdad	Single Source of Truth		

Antónimos	Antonyms	Libro	Autor
Código Duplicado	Duplicate code	Smell Code -(Refactoring)	Martin Fowler
Duplicación	Duplication	Smell Code(Clean Code)	Robert Martin
Copiar y Pegar	Copy + Paste	Antipatrón de Desarrollo	William Brown et al
Escribe todo dos veces o disfrutamos tecleando	Write everything twice or we enjoy typing WET		

# 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

Legibilidad

Código Mínimo.  
Código Muerto,  
YAGNI y DRY

Software	autoexplicativo	consistente	y mínimo
		Estándares	• Código Muerto
		Consistencia	• YAGNI
		Alertas	• DRY
Justificación	Implicaciones	Violaciones <a href="#">TicTacToe</a>	
<ul style="list-style-type: none"> <li>Evitar re-analizar, re-diseñar, re-codificar, re-probar y re-documentar soluciones que complican enormemente el mantenimiento correctivo, perfectivo y adaptativo</li> <li>El efecto 2000 paralizó la producción de software y los gobiernos subvencionaron con el dinero de los impuestos a las empresas privadas para reaccionar ante el problema</li> </ul>	<ul style="list-style-type: none"> <li>Cada pieza de conocimiento debe tener una <b>única, inequívoca y autoritativa representación</b> en un sistema.</li> <li>El objetivo es <b>reducir la repetición de la información de todo tipo</b>, lo que hace que los sistemas de software sean más fácil de mantener</li> <li>La consecuencia es que una modificación de cualquier elemento individual de un sistema <b>no requiere un cambio en otros elementos</b> lógicamente no relacionados.</li> <li><b>Aplicable a todo:</b> la programación, esquemas de bases de datos, planes de prueba, el sistema de construcción, análisis y diseños, incluso la documentación.</li> </ul>	<ul style="list-style-type: none"> <li>Obviamente, código repetido con la misma semántica. <ul style="list-style-type: none"> <li><b>Cuidado!</b> La misma línea en ámbitos diferentes puede ser diferente código por la declaraciones en las que se apoya.</li> <li><code>this.add(element);</code> puede ser completamente diferente en dos clases</li> </ul> </li> <li>Código semánticamente repetido pero con <b>nombres de variables cambiadas, algún orden de sentencias, ...</b></li> <li>Bloque de código que podría sustituirse por llamadas a <b>otros métodos que ya desarrollan esa funcionalidad</b></li> </ul>	

# 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

## Acoplamiento y los principales problemas que detectados

### Inapropiada Intimidad

Sinónimos	Synonyms	Libro	Autor
Intimidad Inapropiada	Inappropriate Intimacy	Smell Code ((Refactoring)	Martin Fowler

Justificación	Solución	
<ul style="list-style-type: none"><li>Una <b>relación bi-direccional complica el desarrollo</b>, las pruebas, la legibilidad, ...</li></ul>	<ul style="list-style-type: none"><li>La sobre-intimidad necesita ser rota:<ul style="list-style-type: none"><li>Debes <b>arreglar relaciones bidireccionales por unidireccionales</b>. Mueve métodos y atributos para separar las piezas que reduzcan la intimidad</li><li>Si las <b>clases tienen interes es encomún, extrae en una nueva clase poniendo lo común</b> a salvo y haz que las demás sean honestas sobre ella.</li><li>La <b>herencia a menudo puede conducir a la sobre-intimidad</b>. Las subclases van a conocer más de sus padres de lo que a sus padres les gustaría que ellos conocieran. <b>Se puede sustituir por Delegación</b></li></ul></li></ul>	<p>“<i>Algunas clases llegan a alcanzar demasiada intimidad y gastan mucho tiempo ahondando en las partes privadas de otras clases. Nosotros pensamos que <b>nuestras clases deberían ser estrictas con reglas puritanas</b></i>”</p> <p>— Fowler Refactoring. 1999</p>

## 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

### Acoplamiento y los principales problemas que detectados

```
class Candidate {  
  
    void printJobAddress(Job job) {  
  
        System.out.println("This is your position address");  
  
        System.out.println(job.address().street());  
        System.out.println(job.address().city());  
        System.out.println(job.address().zipCode());  
  
        if (job.address().country() == job.country()) {  
            System.out.println("It is a local job");  
        }  
    }  
}
```

```
final class Address {  
    void print() {  
        System.out.println(this.street);  
        System.out.println(this.city);  
        System.out.println(this.zipCode);  
    }  
  
    bool isInCountry(Country country) {  
        return this.country == country;  
    }  
}
```

```
class Job {  
    void printAddress() {  
  
        System.out.println("This is your position address");  
  
        this.address().print();  
  
        if (this.address().isInCountry(this.country()) {  
            System.out.println("It is a local job");  
        }  
    }  
}
```

```
class Candidate {  
    void printJobAddress(Job job) {  
        job.printAddress();  
    }  
}
```

## 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

### Acoplamiento y los principales problemas que detectados

#### Leyes de Demeter

Sinónimos	Synonyms	Libro	Autor
No hablescon extraños	Do not talk to strangers	xxx	Lieberherr
Cadena de Mensajes	Chain of Message	Smell Code (Refactoring)	Martin Fowler

Justificación	Solución
<ul style="list-style-type: none"><li>Controlar el <b>bajo acoplamiento restringieno a qué objetos enviar mensajes</b> desde un método</li></ul>	<ul style="list-style-type: none"><li>Enviar únicamente a:<ul style="list-style-type: none"><li><i>this</i> Ley de Demeter: un objeto no debería conocer las</li><li><b>Paré</b> entrañas de otros objetos con los que interactúa</li><li>Atributos</li><li>Objetos locales</li></ul></li><li>No enviar <b>nunca a otros objetos indirectos obtenidos como resultado de un mensaje a un objeto de conocimiento directo.</b></li></ul>

## 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

### Acoplamiento y los principales problemas que detectados

Código Sucio por Librería Incompleta

Sinónimos	Synonyms	Libro	Autor
Clase de biblioteca incompleta	Incomplete Library Class	Smell Code ((Refactoring)	Martin Fowler

Justificación	Solución
<ul style="list-style-type: none"><li>Los <b>desarrolladores de clases de biblioteca son raramente omniscientes</b>. No los culpamos por eso, después de todo, rara vez podemos imaginar un diseño hasta que su mayoría ha sido construida, así que los desarrolladores de la biblioteca tienen un trabajo muy duro.</li><li>El problema es que a menudo es de mala educación, y por lo general imposible, <b>modificar una clase de biblioteca para hacer algo que te gustaría que hiciera</b>.</li></ul>	<ul style="list-style-type: none"><li><b>Crea una clase con los métodos extra adecuados a tus necesidades</b></li></ul>

## 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

### Tamaño y los principales problemas que detectados

Código Sucio por Listas de Parámetros Largas

Sinónimos	Synonyms	Libro	Autor
Lista de Parámetros Larga	Long Parameter List	Smell Code (Refactoring)	Martin Fowler
Demasiados Argumentos	Too Many Arguments	Smell Code (Clean Code)	Robert Martin

Justificación	Solución	Violaciones
<ul style="list-style-type: none"><li>Son <b>difíciles de entender</b></li><li>Son <b>difíciles de probar</b> todas la combinaciones de argumentos</li></ul>	<ul style="list-style-type: none"><li>Eliminar el parámetro cuando puedes <b>obtenerlo a partir de algún objeto que ya conoces</b></li><li>Eliminar <b>varios parámetros suministrando un objeto</b> que los facilite</li><li>Crear <b>un objeto que agrupe varios parámetros</b> y asigna responsabilidad a sus clase</li></ul>	<ul style="list-style-type: none"><li>Funciones deberán tener un <b>número pequeño de argumentos</b>.<ul style="list-style-type: none"><li><b>Sin argumentos es lo mejor</b>,</li><li>seguido por uno, dos.</li><li><b>Tres debería evitarse</b> y <b>más de tres es muy cuestionable</b> y debe considerarse como un prejuicio.</li></ul></li></ul>

## 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

### Tamaño y los principales problemas que detectados

#### Código Sucio por Métodos Largos

Sinónimos	Synonyms	Libro	Autor
Métodos largos	Long Method	Smell Code (Refactoring)	Martin Fowler

Justificación	Solución	Violaciones
<ul style="list-style-type: none"><li>Desde los principios de la programación, los programadores se han dado cuenta de que <b>cuanto más largo es un procedimiento, más difícil es de entender</b>.<ul style="list-style-type: none"><li>Los <b>viejos lenguajes conllevaban una sobrecarga en las llamadas a subrutinas</b>, de tal forma que se persuadía de escribir métodos pequeños.</li><li>Los <b>nuevos programadores orientados a objetos a menudo sienten que la computación no se hace en ninguna parte</b>, que los programas son secuencias sin fin de delegación. Cuando has vivido con un programa como tal por unos años, aprendes cómo de valorable son todos esos pequeños métodos. Todos los costes de indirección – explicación, compartición y selección– son respaldadas por pequeños métodos</li></ul></li></ul>	<ul style="list-style-type: none"><li>El 99% de las veces, se tiene que <b>acortar un método extrayendo otro</b>.<ul style="list-style-type: none"><li>Buscar una parte del método que parezca ir bien junta y hacerlo un nuevo método</li></ul></li></ul>	<ul style="list-style-type: none"><li>Una buena técnica es mirar los comentarios o líneas en blanco para separar partes. Son señales de esta clase de distancia semántica. Un <b>bloque de código con un comentario dice que debes reemplazar el bloque con un método cuyo nombre está basado en el comentario</b></li></ul>

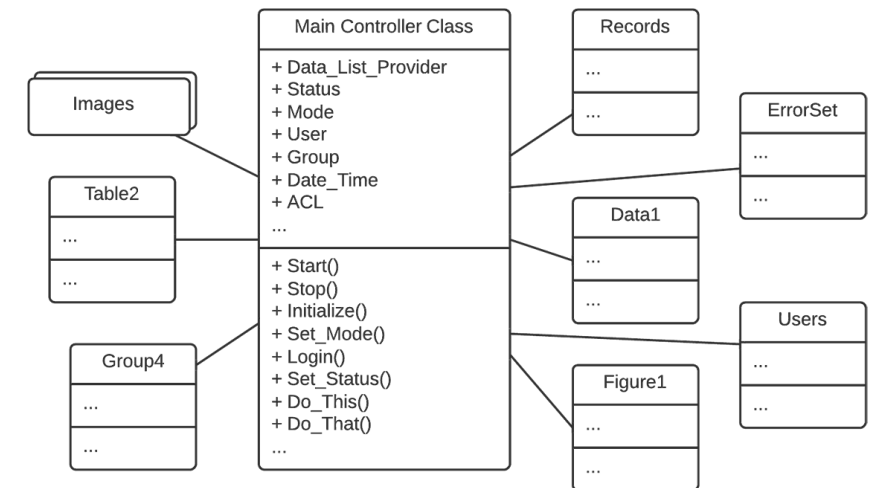
## 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

### Tamaño y los principales problemas que detectados

#### Código Sucio por Clases Grandes

Sinónimos	Synonyms	Libro	Autor
Objeto gigante	<b>Big Large Object(BLOB)</b>	Antipatrón de Desarrollo	William H.Brown et al
Large Class	Clase grande	Smell Code (Refactoring)	Martin Fowler
Demasiada información	Too much Information	Smell Code (Clean Code)	Robert Martin (Uncle Bob)

Justificación	Violaciones	Solución
<ul style="list-style-type: none"><li>Los <b>archivos pequeños son generalmente más fáciles de entender</b> que archivos de gran tamaño.</li></ul>	<ul style="list-style-type: none"><li>Cuando <b>una clase está tratando de hacer demasiado</b>, a menudo aparece con demasiadas variables de instancia.<ul style="list-style-type: none"><li>En tal caso, el <b>código duplicado no puede estar muy lejos</b>.</li></ul></li></ul>	<ul style="list-style-type: none"><li><b>Descomponer la clase otorgando grupos de atributos relacionados a otras clases</b></li><li>Si es una clase de interfaz <b>separa los datos y cálculos del dominio en una clase de entidad</b></li></ul>



## 4.4. DISEÑO ORIENTADO A OBJETOS AVANZADO

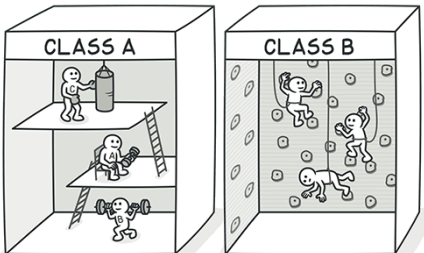
### Tamaño y los principales problemas que detectados

Código Sucio por Atributos Temporales

Sinónimos	Synonyms	Libro	Autor
Campos temporales	Temporary Fields	Smell Code (Refactoring)	Martin Fowler

Justificación	Violaciones	Solución
<ul style="list-style-type: none"><li>A veces se ve un objeto en el que una variable de instancia se establece sólo en ciertas circunstancias. Tratar de entender por qué una variable está allí cuando no parece ser usada puede crear complejidad innecesaria.<ul style="list-style-type: none"><li>Tal código es <b>difícil de comprender porque tu esperas que un objetos necesite todas sus variables.</b></li></ul></li></ul>	<ul style="list-style-type: none"><li>Un caso común de atributo temporal se produce cuando un algoritmo complicado necesita varias variables. Debido a que el ejecutor <b>no quería pasar una lista de parámetros enorme, se ponen en atributos.</b><ul style="list-style-type: none"><li>Pero los atributos son válidas sólo durante el algoritmo; en otros contextos son simplemente confusos.</li></ul></li></ul>	<ul style="list-style-type: none"><li>En este caso, se puede <b>extraer en una clase los atributos y los métodos que lo requieran.</b><ul style="list-style-type: none"><li>El nuevo objeto es un <b>objeto método</b> [Beck]</li></ul></li></ul>

```
function nameToObject (name) {  
  var fullName = name.split(' ');  
  var firstName = fullName[0];  
  var lastName = lastName[1];  
  
  var name = {  
    firstName: firstName,  
    lastName: lastName  
  };  
  
  return name;  
}
```



<https://refactoring.guru/images/refactoring/content/smells/temporary-field-01.png?id=5e30a8144171693ee4894091762b9742>

```
function nameToObject (name) {  
  var fullName = name.split(' ');  
  
  return {  
    firstName: fullName[0],  
    lastName: fullName[1]  
  };  
}
```