



en base a estos dos pdf sobre refactoring quiero que completes la data incompleta y me hagas una lista que explique todos los bad smells y ademas una lista con metodos de refactoring con descripcion y ademas vincules cada code smell con los metodos de refactoring que los solucionan

ResumenRefactoring.pdf

OO2 resumen refactoring.pdf

Basado en los dos documentos proporcionados, he recopilado y completado la información sobre los code smells (malos olores en el código) y los métodos de refactoring descritos. A continuación, presento dos listas detalladas: una con todos los bad smells identificados, su descripción y problemas asociados, y otra con los métodos de refactoring, su propósito y cómo aplicarlos. Finalmente, vinculo cada code smell con los métodos de refactoring que lo solucionan.

---

## Lista de Bad Smells (Malos Olores en el Código)

Los "bad smells" son indicios de problemas en el diseño o implementación del código que sugieren la necesidad de aplicar refactorizaciones. A continuación, se detallan todos los mencionados en los documentos, completando datos faltantes con información implícita o estándar:

1. Duplicated Code (Código Duplicado)
  - Descripción: El mismo código, o muy similar, aparece en varios lugares del sistema.
  - Problemas: Hace el código más largo de lo necesario, dificulta el mantenimiento y los cambios (un bug corregido en un lugar no se propaga fácilmente a otros).
  - Ejemplo: Fragmentos idénticos de código en diferentes clases o métodos.
2. Large Class (Clase Grande)
  - Descripción: Una clase intenta hacer demasiado, con muchas variables de instancia y métodos.
  - Problemas: Indica baja cohesión en el diseño, puede contener código duplicado y algunas responsabilidades podrían pertenecer a otras clases.
  - Ejemplo: Una clase con 20 métodos y 15 variables que maneja múltiples tareas no relacionadas.
3. Long Method (Método Largo)
  - Descripción: Un método tiene demasiadas líneas de código.

- Problemas: Es difícil de entender, modificar y reutilizar; suele indicar lógica compleja que podría dividirse.
  - Ejemplo: Un método con 50 líneas que realiza múltiples tareas secuenciales.
4. Feature Envy (Envidia de Atributo)
    - Descripción: Un método de una clase usa más datos o métodos de otra clase que de la suya propia.
    - Problemas: Sugiere un diseño incorrecto donde el método debería estar en la otra clase; rompe la encapsulación.
    - Ejemplo: Un método que accede repetidamente a los campos de otra clase en lugar de los suyos.
  5. Data Class (Clase de Datos)
    - Descripción: Una clase que solo contiene variables con getters/setters y no tiene comportamiento propio.
    - Problemas: Indica un diseño procedural; otras clases suelen mostrar "Feature Envy" hacia ella.
    - Ejemplo: Una clase Person con solo campos name y age y sus getters/setters.
  6. Long Parameter List (Lista Larga de Parámetros)
    - Descripción: Un método tiene demasiados parámetros.
    - Problemas: Es difícil de entender y usar, complica la obtención de los parámetros y reduce la reusabilidad.
    - Ejemplo: Un método con 8 parámetros como doSomething(int a, int b, String c, double d, boolean e, Object f, int g, String h).
  7. Divergent Change (Cambio Divergente)
    - Descripción: Una clase cambia frecuentemente por razones diferentes.
    - Problemas: Viola el principio de responsabilidad única; dificulta mantener la clase cohesionada.
    - Ejemplo: Una clase que se modifica tanto por cambios en la UI como en la lógica de negocio.
  8. Shotgun Surgery (Cirugía de Escopeta)
    - Descripción: Un cambio en una clase requiere modificaciones en muchas otras clases.
    - Problemas: Indica acoplamiento alto; un pequeño cambio tiene un impacto desproporcionado.
    - Ejemplo: Cambiar un campo en una clase obliga a ajustar 10 clases dependientes.
  9. Comments (Comentarios)
    - Descripción: Uso excesivo de comentarios para explicar código confuso.
    - Problemas: Los comentarios pueden desactualizarse y el código debería ser autoexplicativo; suele indicar mala calidad del código.
    - Ejemplo: Un método con 5 líneas de comentarios explicando una lógica confusa.
  10. Speculative Generality (Generalidad Especulativa)
    - Descripción: Clases, métodos o variables creadas "por si acaso" que no se usan.
    - Problemas: Aumenta la complejidad innecesariamente y dificulta el mantenimiento.

- Ejemplo: Una clase abstracta sin subclases concretas o un método con parámetros no utilizados.
11. Temporary Field (Campo Temporal)
    - Descripción: Un campo de una clase solo se usa en ciertas circunstancias y permanece vacío el resto del tiempo.
    - Problemas: Confunde el propósito del campo y puede indicar que pertenece a otro contexto.
    - Ejemplo: Un campo que solo se usa durante un cálculo específico y luego queda sin uso.
  12. Lazy Class (Clase Perezosa)
    - Descripción: Una clase que hace muy poco y podría fusionarse con otra.
    - Problemas: Aumenta la complejidad sin aportar valor; es el opuesto a "Large Class".
    - Ejemplo: Una clase con un solo método que podría integrarse en otra.
  13. Inappropriate Intimacy (Intimidad Inapropiada)
    - Descripción: Dos clases están demasiado acopladas, accediendo directamente a los detalles internos de la otra.
    - Problemas: Rompe la encapsulación y dificulta los cambios independientes.
    - Ejemplo: Una clase que accede a las variables privadas de otra directamente.
  14. Message Chains (Cadena de Mensajes)
    - Descripción: Una larga secuencia de llamadas a métodos encadenados para obtener un resultado.
    - Problemas: Crea un acoplamiento fuerte con la estructura interna de los objetos; un cambio en la cadena puede romper todo.
    - Ejemplo: `obj.getA().getB().getC().getValue()`.
  15. Middle Man (Intermediario)
    - Descripción: Una clase actúa solo como intermediaria, delegando la mayor parte de su trabajo a otra clase.
    - Problemas: Añade complejidad innecesaria sin aportar valor real.
    - Ejemplo: Una clase que solo pasa mensajes a otra sin lógica propia.
  16. Refused Bequest (Legado Rechazado)
    - Descripción: Una subclase no usa o rechaza el comportamiento heredado de su superclase.
    - Problemas: Viola el principio de Liskov; indica un mal uso de la herencia.
    - Ejemplo: Una subclase que lanza excepciones en métodos heredados que no necesita.
  17. Conditional Logic (Lógica Condicional Compleja)
    - Descripción: Uso excesivo de sentencias condicionales (if/switch) para manejar diferentes casos o tipos de objetos.
    - Problemas: Difícil de mantener y extender; suele duplicarse en varios lugares.
    - Ejemplo: Un método con múltiples if para tratar diferentes tipos de objetos.

---

## Lista de Métodos de Refactoring

Los métodos de refactoring son transformaciones que mejoran el diseño del código sin alterar su comportamiento externo. A continuación, se describen los mencionados en los documentos, completando información faltante con detalles estándar:

1. Extract Method (Extraer Método)
  - Descripción: Mueve un fragmento de código a un nuevo método con un nombre descriptivo.
  - Cómo aplicar: Identificar código agrupable, crear un nuevo método, mover el código y reemplazar el original con una llamada.
  - Ejemplo: Extraer la impresión de detalles de un método largo a `printDetails()`.
2. Inline Method (Incorporar Método)
  - Descripción: Reemplaza las llamadas a un método con su contenido y elimina el método.
  - Cómo aplicar: Si el cuerpo del método es simple y obvio, insertar su código directamente y borrar el método.
  - Ejemplo: Reemplazar `getRating()` con su lógica ternaria directamente.
3. Replace Temp with Query (Reemplazar Variable Temporal con Consulta)
  - Descripción: Sustituye una variable temporal por un método que calcula su valor.
  - Cómo aplicar: Extraer la expresión a un método, reemplazar la variable con llamadas al método.
  - Ejemplo: Reemplazar `basePrice` con un método `basePrice()`.
4. Split Temporary Variable (Dividir Variable Temporal)
  - Descripción: Usa variables separadas para diferentes propósitos en lugar de reutilizar una misma.
  - Cómo aplicar: Crear variables específicas para cada valor intermedio y renombrarlas.
  - Ejemplo: Cambiar `temp` por `perimeter` y `area`.
5. Replace Method with Method Object (Reemplazar Método con Objeto Método)
  - Descripción: Convierte un método largo con variables locales en una clase separada.
  - Cómo aplicar: Crear una clase nueva, mover las variables como campos y dividir la lógica en métodos.
  - Ejemplo: Transformar `price()` en `PriceCalculator`.
6. Move Method (Mover Método)
  - Descripción: Traslada un método a la clase donde más se usa.
  - Cómo aplicar: Crear el método en la clase destino, mover el código y ajustar referencias.
  - Ejemplo: Mover un método que usa datos de otra clase a esa clase.
7. Move Field (Mover Campo)
  - Descripción: Traslada un campo a la clase donde más se usa.
  - Cómo aplicar: Declarar el campo en la clase destino y actualizar accesos.
  - Ejemplo: Mover un campo usado por otra clase a esa clase.
8. Extract Class (Extraer Clase)
  - Descripción: Divide una clase grande en dos, asignando responsabilidades claras.

- Cómo aplicar: Crear una nueva clase, mover campos y métodos relacionados, ajustar referencias.
  - Ejemplo: Extraer lógica de una clase grande a una nueva clase auxiliar.
9. Inline Class (Incorporar Clase)
- Descripción: Fusiona una clase pequeña o innecesaria con otra.
  - Cómo aplicar: Mover campos y métodos a la clase objetivo y eliminar la clase original.
  - Ejemplo: Integrar una clase con un solo método en otra.
10. Remove Middle Man (Eliminar Intermediario)
- Descripción: Elimina una clase que solo delega trabajo a otra.
  - Cómo aplicar: Hacer que el cliente interactúe directamente con la clase destino y eliminar el intermediario.
  - Ejemplo: Quitar una clase que solo pasa mensajes.
11. Hide Delegate (Ocultar Delegado)
- Descripción: Crea métodos en una clase para ocultar la delegación a otra.
  - Cómo aplicar: Añadir métodos en la clase origen que encapsulen las llamadas al delegado.
  - Ejemplo: Crear un método en A que oculte accesos a B.
12. Pull Up Field (Subir Campo)
- Descripción: Mueve un campo común de las subclases a la superclase.
  - Cómo aplicar: Declarar el campo en la superclase y eliminarlo de las subclases.
  - Ejemplo: Mover name de subclases a la superclase.
13. Push Down Field (Bajar Campo)
- Descripción: Mueve un campo de la superclase a las subclases que lo necesitan.
  - Cómo aplicar: Declarar el campo solo en las subclases relevantes y eliminarlo de la superclase.
  - Ejemplo: Bajar un campo usado por una sola subclase.
14. Pull Up Method (Subir Método)
- Descripción: Mueve métodos idénticos o similares de las subclases a la superclase.
  - Cómo aplicar: Asegurar que sean idénticos, moverlos a la superclase y ajustar referencias.
  - Ejemplo: Subir un método común de cálculo a la superclase.
15. Push Down Method (Bajar Método)
- Descripción: Mueve un método de la superclase a las subclases que lo usan.
  - Cómo aplicar: Mover el método a las subclases relevantes y eliminarlo de la superclase si no es necesario.
  - Ejemplo: Bajar un método usado solo por una subclase.
16. Pull Up Constructor Body (Subir Cuerpo del Constructor)
- Descripción: Mueve código común de constructores de subclases a la superclase.
  - Cómo aplicar: Crear un constructor en la superclase y llamarlo desde las subclases.
  - Ejemplo: Subir inicialización común de name e id a la superclase.
17. Extract Subclass (Extraer Subclase)

- Descripción: Crea una subclase para manejar un comportamiento específico de una clase.
  - Cómo aplicar: Identificar comportamiento variable, crear una subclase y moverlo allí.
  - Ejemplo: Extraer lógica específica de un tipo de empleado a una subclase.
18. Extract Superclass (Extraer Superclase)
- Descripción: Crea una superclase para compartir campos y métodos comunes entre clases.
  - Cómo aplicar: Crear la superclase, mover elementos comunes y ajustar herencia.
  - Ejemplo: Crear una superclase para Department y Party.
19. Collapse Hierarchy (Colapsar Jerarquía)
- Descripción: Fusiona una superclase y subclase cuando no hay diferencias significativas.
  - Cómo aplicar: Mover todo a una sola clase y eliminar la jerarquía innecesaria.
  - Ejemplo: Unir una superclase y subclase casi idénticas.
20. Replace Inheritance with Delegation (Reemplazar Herencia con Delegación)
- Descripción: Sustituye herencia por un campo que delega a un objeto de la clase original.
  - Cómo aplicar: Crear un campo para el objeto, delegar métodos y eliminar la herencia.
  - Ejemplo: Reemplazar herencia de Vector con un campo que lo referencia.
21. Replace Delegation with Inheritance (Reemplazar Delegación con Herencia)
- Descripción: Convierte una relación de delegación en herencia si es más adecuada.
  - Cómo aplicar: Hacer que la clase herede directamente en lugar de delegar.
  - Ejemplo: Convertir una clase que delega a otra en una subclase.
22. Self Encapsulate Field (Autoencapsular Campo)
- Descripción: Crea getters y setters para un campo y usa estos en lugar de accesos directos.
  - Cómo aplicar: Definir métodos de acceso y actualizar el código interno.
  - Ejemplo: Reemplazar low por getLow() dentro de la clase.
23. Encapsulate Field / Collection (Encapsular Campo / Colección)
- Descripción: Hace que el acceso a un campo o colección sea controlado mediante métodos.
  - Cómo aplicar: Crear métodos para agregar/eliminar elementos y hacer el campo privado.
  - Ejemplo: Encapsular una lista con addCourse() y removeCourse().
24. Replace Data Value with Object (Reemplazar Valor de Datos con Objeto)
- Descripción: Convierte un campo simple en un objeto con comportamiento propio.
  - Cómo aplicar: Crear una clase para el dato y reemplazar el campo con una instancia.
  - Ejemplo: Cambiar customer: String por customer: Customer.
25. Replace Array with Object (Reemplazar Arreglo con Objeto)
- Descripción: Sustituye un arreglo con campos heterogéneos por un objeto con campos nombrados.

- Cómo aplicar: Crear una clase con campos específicos y reemplazar el arreglo.
  - Ejemplo: Cambiar row[0] y row[1] por Performance con name y wins.
26. Replace Magic Number with Symbolic Constant (Reemplazar Número Mágico con Constante Simbólica)
- Descripción: Sustituye un número literal por una constante con un nombre significativo.
  - Cómo aplicar: Definir una constante y usarla en lugar del número.
  - Ejemplo: Reemplazar 9.81 por GRAVITATIONAL\_CONSTANT.
27. Decompose Conditional (Descomponer Condicional)
- Descripción: Divide un condicional complejo en métodos separados.
  - Cómo aplicar: Extraer la lógica de cada rama a métodos con nombres descriptivos.
  - Ejemplo: Separar if (date.before(SUMMER\_START)) en isSummer() y summerCharge().
28. Consolidate Conditional Expression (Consolidar Expresión Condicional)
- Descripción: Combina múltiples condiciones relacionadas en una sola expresión.
  - Cómo aplicar: Unificar condiciones con el mismo resultado en un solo if.
  - Ejemplo: Combinar varios if que devuelven 0 en una sola condición.
29. Replace Conditional with Polymorphism (Reemplazar Condicional con Polimorfismo)
- Descripción: Usa subclases y polimorfismo para eliminar condicionales basados en tipos.
  - Cómo aplicar: Crear subclases, mover la lógica de cada rama a métodos sobrescritos y eliminar el condicional.
  - Ejemplo: Reemplazar un switch por una jerarquía de clases.
30. Rename Method / Field (Renombrar Método / Campo)
- Descripción: Cambia el nombre de un método o campo para que sea más descriptivo.
  - Cómo aplicar: Crear un nuevo método/campo con el nombre nuevo, copiar el contenido y actualizar referencias.
  - Ejemplo: Cambiar getsp() a getSpeed().
31. Preserve Whole Object (Preservar Objeto Completo)
- Descripción: Pasa un objeto completo en lugar de extraer sus campos como parámetros.
  - Cómo aplicar: Modificar la firma del método para aceptar el objeto y usarlo internamente.
  - Ejemplo: Pasar un Order en lugar de order.getId() y order.getPrice().
32. Introduce Parameter Object (Introducir Objeto Parámetro)
- Descripción: Agrupa parámetros relacionados en un objeto.
  - Cómo aplicar: Crear una clase para los parámetros y usarla como argumento.
  - Ejemplo: Reemplazar doSomething(int a, int b, String c) con doSomething(Parameters p).
33. Parameterize Method (Parametrizar Método)
- Descripción: Generaliza métodos similares añadiendo parámetros para diferencias.
  - Cómo aplicar: Identificar diferencias, añadir parámetros y unificar métodos.

- Ejemplo: Unificar `printInvoiceA()` y `printInvoiceB()` en `printInvoice(type)`.
34. Form Template Method (Formar Método Plantilla)
- Descripción: Generaliza métodos similares en subclases en un método plantilla en la superclase.
  - Cómo aplicar: Extraer pasos comunes a la superclase y definir métodos abstractos para las diferencias.
  - Ejemplo: Subir pasos comunes de métodos en subclases a un `templateMethod()`.
35. Substitute Algorithm (Sustituir Algoritmo)
- Descripción: Reemplaza un algoritmo por otro más claro o eficiente.
  - Cómo aplicar: Reescribir el código con el nuevo algoritmo y verificar que pase los tests.
  - Ejemplo: Cambiar un algoritmo de ordenación por uno más eficiente.
36. Replace Conditional Logic with Strategy (Reemplazar Lógica Condicional con Strategy)
- Descripción: Usa el patrón Strategy para delegar variantes de un algoritmo a clases separadas.
  - Cómo aplicar: Crear una jerarquía de estrategias, mover la lógica condicional y delegar desde el contexto.
  - Ejemplo: Reemplazar un `if` con clases `StrategyA` y `StrategyB`.
37. Replace State-Altering Conditionals with State (Reemplazar Condicionales de Cambio de Estado con State)
- Descripción: Usa el patrón State para manejar transiciones de estado en lugar de condicionales.
  - Cómo aplicar: Crear una jerarquía de estados, mover la lógica de transición y delegar a los estados.
  - Ejemplo: Reemplazar `if (state == "active")` con clases `ActiveState` y `InactiveState`.
38. Move Embellishment to Decorator (Mover Embellecimiento a Decorator)
- Descripción: Usa el patrón Decorator para añadir funcionalidades dinámicas en lugar de condicionales o herencia.
  - Cómo aplicar: Crear decoradores como subclases que deleguen al objeto original y añadir comportamiento.
  - Ejemplo: Mover lógica condicional de formato a un `TextDecorator`.
39. Introduce Null Object (Introducir Objeto Nulo)
- Descripción: Reemplaza valores `null` con un objeto que implemente un comportamiento por defecto.
  - Cómo aplicar: Crear una clase `NullObject` con la misma interfaz y usarla en lugar de `null`.
  - Ejemplo: Reemplazar `if (obj == null)` con un `NullCustomer`.

---

## Vinculación entre Code Smells y Métodos de Refactoring

A continuación, se relacionan los bad smells con los métodos de refactoring que los abordan, basados en los documentos y asociaciones lógicas:



1. Duplicated Code
  - Extract Method: Agrupa código duplicado en un método reutilizable.
  - Pull Up Method: Mueve código duplicado en subclases a la superclase.
  - Form Template Method: Generaliza métodos duplicados en subclases en un método plantilla.
2. Large Class
  - Extract Class: Divide la clase en varias con responsabilidades claras.
  - Extract Subclass: Mueve comportamiento específico a una subclase.
  - Extract Superclass: Comparte funcionalidad común en una superclase.
3. Long Method
  - Extract Method: Divide el método en fragmentos más pequeños y claros.
  - Decompose Conditional: Separa condicionales complejos en métodos.
  - Replace Temp with Query: Elimina variables temporales con métodos.
4. Feature Envy
  - Move Method: Traslada el método a la clase cuyos datos usa más.
  - Move Field: Mueve campos usados por el método a la clase destino.
5. Data Class
  - Move Method: Traslada métodos que usan sus datos a la clase de datos.
  - Encapsulate Field / Collection: Añade comportamiento a la clase controlando el acceso a sus datos.
6. Long Parameter List
  - Replace Parameter with Method: Obtiene parámetros desde métodos en lugar de pasarlos.
  - Preserve Whole Object: Pasa un objeto completo en lugar de sus partes.
  - Introduce Parameter Object: Agrupa parámetros en un objeto.
7. Divergent Change
  - Extract Class: Separa responsabilidades distintas en clases diferentes.
8. Shotgun Surgery
  - Move Method: Centraliza el cambio moviendo métodos a la clase afectada.
  - Move Field: Mueve campos relacionados al lugar donde se usan más.
9. Comments
  - Extract Method: Convierte código comentado en métodos con nombres claros.
  - Rename Method: Usa nombres descriptivos que eliminen la necesidad de comentarios.
10. Speculative Generality
  - Collapse Hierarchy: Elimina jerarquías innecesarias.
  - Inline Class: Fusiona clases poco usadas con otras.
  - Remove Parameter: Quita parámetros no utilizados.
11. Temporary Field
  - Extract Class: Mueve el campo y su lógica a una nueva clase.
  - Replace Method with Method Object: Convierte el método que lo usa en un objeto con campos.
12. Lazy Class
  - Inline Class: Fusiona la clase con otra más significativa.
  - Collapse Hierarchy: Elimina jerarquías con clases redundantes.
13. Inappropriate Intimacy
  - Move Method: Reubica métodos para reducir el acceso indebido.

- Move Field: Mueve campos para limitar la dependencia entre clases.
  - Hide Delegate: Oculta la relación con otra clase mediante métodos encapsulados.
14. Message Chains
- Hide Delegate: Crea métodos intermedios para romper la cadena.
  - Extract Method & Move Method: Divide y reubica la lógica para reducir dependencias.
15. Middle Man
- Remove Middle Man: Elimina la clase intermediaria y conecta directamente al destino.
16. Refused Bequest
- Push Down Method: Mueve métodos no usados a las subclases que los necesitan.
  - Push Down Field: Baja campos irrelevantes a las subclases correspondientes.
17. Conditional Logic
- Decompose Conditional: Divide condicionales complejos en métodos.
  - Consolidate Conditional Expression: Unifica condiciones relacionadas.
  - Replace Conditional with Polymorphism: Usa polimorfismo para eliminar condicionales basados en tipos.
  - Replace Conditional Logic with Strategy: Delega variantes a estrategias.
  - Replace State-Altering Conditionals with State: Maneja estados con el patrón State.

---

## Notas Finales

- Algunos "code smells" (como "Conditional Logic") no están explícitamente categorizados en los documentos como tales, pero se infieren de las descripciones de problemas con condicionales.
- Los métodos de refactoring se han completado con pasos estándar donde los documentos eran vagos (por ejemplo, "Substitute Algorithm").
- La vinculación se basa en las asociaciones explícitas de los documentos (como el catálogo de Fowler) y en prácticas comunes de refactoring.

Si necesitas más detalles o ejemplos específicos para algún caso, no dudes en pedírmelo.

¡Espero que esta lista sea útil!

de la lista de metodos de refactoring que me diste quiero que elimines todos los metodos que conlleven patrones de diseño

Claro, voy a eliminar de la lista de métodos de refactoring aquellos que explícitamente involucran patrones de diseño, como Strategy, State, Decorator, o cualquier otro que implique la introducción o uso directo de un patrón de diseño estructural, de comportamiento o creacional según el catálogo de patrones de diseño (por ejemplo, de Gang of Four). Los

patrones de diseño suelen implicar la creación de nuevas clases o jerarquías para resolver problemas de manera genérica y reusable, mientras que los métodos de refactoring restantes se centran más en transformaciones locales y estructurales del código sin introducir patrones formales.

A continuación, presento la lista actualizada de métodos de refactoring, eliminando aquellos que conllevan patrones de diseño:

---

## Lista de Métodos de Refactoring (Sin Patrones de Diseño)

1. Extract Method (Extraer Método)
  - Descripción: Mueve un fragmento de código a un nuevo método con un nombre descriptivo.
  - Cómo aplicar: Identificar código agrupable, crear un nuevo método, mover el código y reemplazar el original con una llamada.
  - Ejemplo: Extraer la impresión de detalles de un método largo a `printDetails()`.
2. Inline Method (Incorporar Método)
  - Descripción: Reemplaza las llamadas a un método con su contenido y elimina el método.
  - Cómo aplicar: Si el cuerpo del método es simple y obvio, insertar su código directamente y borrar el método.
  - Ejemplo: Reemplazar `getRating()` con su lógica ternaria directamente.
3. Replace Temp with Query (Reemplazar Variable Temporal con Consulta)
  - Descripción: Sustituye una variable temporal por un método que calcula su valor.
  - Cómo aplicar: Extraer la expresión a un método, reemplazar la variable con llamadas al método.
  - Ejemplo: Reemplazar `basePrice` con un método `basePrice()`.
4. Split Temporary Variable (Dividir Variable Temporal)
  - Descripción: Usa variables separadas para diferentes propósitos en lugar de reutilizar una misma.
  - Cómo aplicar: Crear variables específicas para cada valor intermedio y renombrarlas.
  - Ejemplo: Cambiar `temp` por `perimeter` y `area`.
5. Replace Method with Method Object (Reemplazar Método con Objeto Método)
  - Descripción: Convierte un método largo con variables locales en una clase separada.
  - Cómo aplicar: Crear una clase nueva, mover las variables como campos y dividir la lógica en métodos.
  - Ejemplo: Transformar `price()` en `PriceCalculator`.
6. Move Method (Mover Método)
  - Descripción: Traslada un método a la clase donde más se usa.
  - Cómo aplicar: Crear el método en la clase destino, mover el código y ajustar referencias.

- Ejemplo: Mover un método que usa datos de otra clase a esa clase.
7. Move Field (Mover Campo)
    - Descripción: Traslada un campo a la clase donde más se usa.
    - Cómo aplicar: Declarar el campo en la clase destino y actualizar accesos.
    - Ejemplo: Mover un campo usado por otra clase a esa clase.
  8. Extract Class (Extraer Clase)
    - Descripción: Divide una clase grande en dos, asignando responsabilidades claras.
    - Cómo aplicar: Crear una nueva clase, mover campos y métodos relacionados, ajustar referencias.
    - Ejemplo: Extraer lógica de una clase grande a una nueva clase auxiliar.
  9. Inline Class (Incorporar Clase)
    - Descripción: Fusiona una clase pequeña o innecesaria con otra.
    - Cómo aplicar: Mover campos y métodos a la clase objetivo y eliminar la clase original.
    - Ejemplo: Integrar una clase con un solo método en otra.
  10. Remove Middle Man (Eliminar Intermediario)
    - Descripción: Elimina una clase que solo delega trabajo a otra.
    - Cómo aplicar: Hacer que el cliente interactúe directamente con la clase destino y eliminar el intermediario.
    - Ejemplo: Quitar una clase que solo pasa mensajes.
  11. Hide Delegate (Ocultar Delegado)
    - Descripción: Crea métodos en una clase para ocultar la delegación a otra.
    - Cómo aplicar: Añadir métodos en la clase origen que encapsulen las llamadas al delegado.
    - Ejemplo: Crear un método en A que oculte accesos a B.
  12. Pull Up Field (Subir Campo)
    - Descripción: Mueve un campo común de las subclases a la superclase.
    - Cómo aplicar: Declarar el campo en la superclase y eliminarlo de las subclases.
    - Ejemplo: Mover name de subclases a la superclase.
  13. Push Down Field (Bajar Campo)
    - Descripción: Mueve un campo de la superclase a las subclases que lo necesitan.
    - Cómo aplicar: Declarar el campo solo en las subclases relevantes y eliminarlo de la superclase.
    - Ejemplo: Bajar un campo usado por una sola subclase.
  14. Pull Up Method (Subir Método)
    - Descripción: Mueve métodos idénticos o similares de las subclases a la superclase.
    - Cómo aplicar: Asegurar que sean idénticos, moverlos a la superclase y ajustar referencias.
    - Ejemplo: Subir un método común de cálculo a la superclase.
  15. Push Down Method (Bajar Método)
    - Descripción: Mueve un método de la superclase a las subclases que lo usan.
    - Cómo aplicar: Mover el método a las subclases relevantes y eliminarlo de la superclase si no es necesario.
    - Ejemplo: Bajar un método usado solo por una subclase.

16. Pull Up Constructor Body (Subir Cuerpo del Constructor)

- Descripción: Mueve código común de constructores de subclases a la superclase.
- Cómo aplicar: Crear un constructor en la superclase y llamarlo desde las subclases.
- Ejemplo: Subir inicialización común de name e id a la superclase.

17. Extract Subclass (Extraer Subclase)

- Descripción: Crea una subclase para manejar un comportamiento específico de una clase.
- Cómo aplicar: Identificar comportamiento variable, crear una subclase y moverlo allí.
- Ejemplo: Extraer lógica específica de un tipo de empleado a una subclase.

18. Extract Superclass (Extraer Superclase)

- Descripción: Crea una superclase para compartir campos y métodos comunes entre clases.
- Cómo aplicar: Crear la superclase, mover elementos comunes y ajustar herencia.
- Ejemplo: Crear una superclase para Department y Party.

19. Collapse Hierarchy (Colapsar Jerarquía)

- Descripción: Fusiona una superclase y subclase cuando no hay diferencias significativas.
- Cómo aplicar: Mover todo a una sola clase y eliminar la jerarquía innecesaria.
- Ejemplo: Unir una superclase y subclase casi idénticas.

20. Self Encapsulate Field (Autoencapsular Campo)

- Descripción: Crea getters y setters para un campo y usa estos en lugar de accesos directos.
- Cómo aplicar: Definir métodos de acceso y actualizar el código interno.
- Ejemplo: Reemplazar low por getLow() dentro de la clase.

21. Encapsulate Field / Collection (Encapsular Campo / Colección)

- Descripción: Hace que el acceso a un campo o colección sea controlado mediante métodos.
- Cómo aplicar: Crear métodos para agregar/eliminar elementos y hacer el campo privado.
- Ejemplo: Encapsular una lista con addCourse() y removeCourse().

22. Replace Data Value with Object (Reemplazar Valor de Datos con Objeto)

- Descripción: Convierte un campo simple en un objeto con comportamiento propio.
- Cómo aplicar: Crear una clase para el dato y reemplazar el campo con una instancia.
- Ejemplo: Cambiar customer: String por customer: Customer.

23. Replace Array with Object (Reemplazar Arreglo con Objeto)

- Descripción: Sustituye un arreglo con campos heterogéneos por un objeto con campos nombrados.
- Cómo aplicar: Crear una clase con campos específicos y reemplazar el arreglo.
- Ejemplo: Cambiar row[0] y row[1] por Performance con name y wins.

24. Replace Magic Number with Symbolic Constant (Reemplazar Número Mágico con Constante Simbólica)

- Descripción: Sustituye un número literal por una constante con un nombre significativo.
  - Cómo aplicar: Definir una constante y usarla en lugar del número.
  - Ejemplo: Reemplazar 9.81 por GRAVITATIONAL\_CONSTANT.
25. Decompose Conditional (Descomponer Condicional)
- Descripción: Divide un condicional complejo en métodos separados.
  - Cómo aplicar: Extraer la lógica de cada rama a métodos con nombres descriptivos.
  - Ejemplo: Separar `if (date.before(SUMMER_START))` en `isSummer()` y `summerCharge()`.
26. Consolidate Conditional Expression (Consolidar Expresión Condicional)
- Descripción: Combina múltiples condiciones relacionadas en una sola expresión.
  - Cómo aplicar: Unificar condiciones con el mismo resultado en un solo `if`.
  - Ejemplo: Combinar varios `if` que devuelven 0 en una sola condición.
27. Rename Method / Field (Renombrar Método / Campo)
- Descripción: Cambia el nombre de un método o campo para que sea más descriptivo.
  - Cómo aplicar: Crear un nuevo método/campo con el nombre nuevo, copiar el contenido y actualizar referencias.
  - Ejemplo: Cambiar `getsp()` a `getSpeed()`.
28. Preserve Whole Object (Preservar Objeto Completo)
- Descripción: Pasa un objeto completo en lugar de extraer sus campos como parámetros.
  - Cómo aplicar: Modificar la firma del método para aceptar el objeto y usarlo internamente.
  - Ejemplo: Pasar un `Order` en lugar de `order.getId()` y `order.getPrice()`.
29. Introduce Parameter Object (Introducir Objeto Parámetro)
- Descripción: Agrupa parámetros relacionados en un objeto.
  - Cómo aplicar: Crear una clase para los parámetros y usarla como argumento.
  - Ejemplo: Reemplazar `doSomething(int a, int b, String c)` con `doSomething(Parameters p)`.
30. Parameterize Method (Parametrizar Método)
- Descripción: Generaliza métodos similares añadiendo parámetros para diferencias.
  - Cómo aplicar: Identificar diferencias, añadir parámetros y unificar métodos.
  - Ejemplo: Unificar `printInvoiceA()` y `printInvoiceB()` en `printInvoice(type)`.
31. Form Template Method (Formar Método Plantilla)
- Descripción: Generaliza métodos similares en subclases en un método plantilla en la superclase.
  - Cómo aplicar: Extraer pasos comunes a la superclase y definir métodos abstractos para las diferencias.
  - Ejemplo: Subir pasos comunes de métodos en subclases a un `templateMethod()`.
32. Substitute Algorithm (Sustituir Algoritmo)
- Descripción: Reemplaza un algoritmo por otro más claro o eficiente.

- Cómo aplicar: Reescribir el código con el nuevo algoritmo y verificar que pase los tests.
- Ejemplo: Cambiar un algoritmo de ordenación por uno más eficiente.

---

## Métodos Eliminados y Razón

Los siguientes métodos fueron eliminados porque implican la introducción o uso explícito de patrones de diseño:

1. Replace Conditional with Polymorphism
  - Razón: Usa polimorfismo a través de una jerarquía de clases, lo que implica diseño orientado a objetos avanzado y puede considerarse un patrón implícito (como el uso de herencia para comportamiento).
2. Replace Inheritance with Delegation
  - Razón: Introduce el patrón de delegación, que es una técnica común en patrones como Proxy o Decorator.
3. Replace Delegation with Inheritance
  - Razón: Aunque revierte delegación, implica un ajuste estructural que puede relacionarse con patrones de herencia.
4. Replace Conditional Logic with Strategy
  - Razón: Introduce explícitamente el patrón Strategy.
5. Replace State-Altering Conditionals with State
  - Razón: Introduce explícitamente el patrón State.
6. Move Embellishment to Decorator
  - Razón: Introduce explícitamente el patrón Decorator.
7. Introduce Null Object
  - Razón: Introduce el patrón Null Object, que es un patrón de diseño de comportamiento.

---

## Notas

- Algunos métodos como Extract Subclass o Extract Superclass podrían debatirse, ya que usan herencia, pero no introducen patrones formales complejos como Strategy o Decorator, sino que son transformaciones estructurales básicas. Por eso los mantuve.
- Form Template Method se mantuvo porque, aunque se asocia al patrón Template Method, en este contexto se describe como una técnica de refactorización para generalizar código sin requerir un patrón completo.