Análisis y diseño orientado a objetos

Sobre la codificación:
Los atributos siempre se ponen arriba, al inicio de la clase
Si tengo un metodo que llama a otro método, entonces lo pongo debajo.
Se deben agrupar los métodos con cierta naturaleza de operaciones semejantes.
Si una variable no es un atributo de clase entonces ponerla dentro el método para reducir su tiempo de vida.
La variable debes estar inmediatamente antes de su utilización, si esta muy arriba, al momento de utilizarla no se entiende su contexto.
Se debe en lo posible escribir líneas cortas con un máximo de 80 caracteres.
Se debe dejar un espacio en blanco entre métodos.
No olvidar sangrar los bloques para facilitar la lectura.
No usar tabuladores entre las variables para alinearlas
Nunca romper la línea de sangrado por muy pequeña que sea la línea (tio bob).
Recordar siempre que una línea de código se escribe una vez y se lee 100 veces.
Hay que analizar el orden en el que ponemos los atributos para mejorar la lectura, es recomendable agruparlos si están fuertemente relacionados.
Si se trabaja en un equipo de trabajo, todos deben ponerse de acuerdo con un solo estilo de trabajo y luego todos deben usar ese estilo.
Se deben evitar los comentarios irrelevantes, textos largos, etc.

Las clases deben tener nombres sustantivos
Se debe pensar bien al poner el nombre de un atributo o método, debe ser auto explicativos y no necesitar comentarios.
Nada puede ser tan útil como un buen comentario en el lugar exacto (ejm explicacion de una expresión regular usada)
El código debe ser claro y expresivo para no necesitar comentarios.
No comentar código malo, mejor reescribelo.

Código mal nombrado.
Los nombres de variables y métodos deben revelar su intención
Nombres deben ser pronunciables de manera que puedan ser usados en una conversación.
Usar camelcase al nombrar las variables y métodos (no usar guión bajo.
Usar nombres del dominio del programa y de la solución(no inventar palabras nuevas)
Usar vocabulario de la solución y no palabras extrañas al problema
Cuando escogemos una palabra aférrate a ella, no usar sinónimos, no usar varios nombres para lo mismo. (ejm. insertar, añadir, ingresar, ...)
Usar las reglas de ámbito (si menciono algo en un bloque se busca eso mas arriba dentro del bloque)
Los nombres de los paquetes deben ser sustantivos.
Las clases deben ser sustantivos, deben iniciar con mayúsculas y debe dar una descripción de sí mismas en el nombre.
Los métodos deben ser verbos o una frase con un verbo y comenzar con minúsculas
Los métodos de acceso deben anteponer ger,set, put, is (is para booleano).
No poner caracteres especiales en los nombre !?* etc

No usar nombres abreviados "d" "mdp" "gr1" que no explican nada
Las variables nunca deben ser de una letra no usar l o O ya que se confunden con 1 y 0
La excepción son las i, j, k para los contadores en los bucles o para coordenadas vectoriales
Las excepciones en nombre son acrónimos que mundialmente serán reconocidos como http, jpg, sos, etc.
Todo el código en ingles, no debe haber una mezcla de español ingles.
Espacio entre cada atributo declarado
Los atributos no deben representar acciones si no cosas, eso es descomposición funcional
Si hay demasiadas static y esa es una mala señal

---- Descomposición funcional -------

La descomposición funcional es un anti patrón.
Síntomas
- Clases con nombres de funcional (que notan acción)
- Clases con un solo método
- Ausencia de principios orientado a objetos, como herencia y polimorfismo

* Imposible de comprender el software, de utilizar, de probar, ...
* Solución
  - Aplicar los patrones Generals para asignación de responsabilidades del software: GRASP

Categorización
Todas las entidades tienen una determinada propiedad o conjunto de propiedades en común forman una categoría. Estas propiedades son necesarias y suficientes para definir una categoría.

---------- PRINCIPIOS GRASP -------

Los principios generales de software para asignación de responsabilidades
- Ayudan al aprendizaje de entender el diseño objeto esencial, y aplicar el razonamiento del diseño de una manera metódica, racional, explicable.
- Este enfoque de comprensión y uso de los principios de diseño se basan en los patrones de asignación de responsabilidades.

Los principios son :
- Experto en la información
- Alta cohesión
- Bajo acoplamiento
- Inversión pura
- Controlador
- Creador
- Indirección
- Polimorfismo
- Variaciones protegidas

Experto en la información

¿cual es el principio general de la asignación de responsabilidades?
- Asignar responsabilidades al experto en la información. la clase que tiene la información necesaria para cumplir con la responsabilidad.
Cuando estés haciendo muchos get es peligroso.

hacer clases entre 200 y 500 líneas
3 a 5 atributos máximo
20 metodos como máximo

Hay que poner real atención a los requisitos. (la cosificación)

Métodos para identificar clases y objetos derivados de los requisitos del dominio (identificar clases en los requisitos)

-Cosas, objetos físicos o grupos de objetos que son tangibles
-conceptos , principios o ideas no tabibles
-cosas que pasan
-gente seres humanos

Al diseñar una clase no te centres tanto en los métodos, concentrate mas en el comportamiento de la clase y en las responsabilidades.
Las responsabilidades que un objeto mantiene y las acciones que puede realizar.
Las responsabilidades de un objeto son todos los servicios que presta a los contratos que apoya (contrato se refiere a que otra clase me pide cosas a través de mi interfaz)
Cuando hago un planteamiento tengo que cubrir ambos atributos y métodos no solo uno.
Los escenarios son cada uno de los distintos posibles caminos de un caso de uso
Cohesión es la alta relación entre los atributos de una clase para cumplir el objetivo
Evitar las clases que son una compilación de muchas utilidades (ejm clase útiles)

Estudiar
programación avanzada generics y closures
programación parametrizada

Cirugía a escopetazos (smel code)
No usar números mágicos, usar constantes literales (los números mágicos son variables que no explican su intención y no se sabe porque tienen el valor que tienen).
Romper los ciclos de dependencia (relaciones bidireccionales) exepto cuando romper el ciclo sea más complejo que dejarlo, cuando ya no te de ninguna ventaja.
Tratar de no usar métodos estáticos porque se convierten en funciones que puedo llamar desde donde me da la gana anteponiendo el nombre de la clase.

Alta cohesión y bajo acoplamiento
Problema: ¿Cómo mantener la complejidad manejable?, las clases tienen que ser cohesivas, todas las lineas del codigo de una clase tienen que estar estrechamente relacionadas con la responsabilidad que tiene esa clase.

- La cohesión, o más específicamente , la cohesión funcional, es una media de cómo de fuerte están relacionadas y enfocadas las responsabilidades que son de un elemento.
- Un elemento con responsabilidades muy relacionadas y que no hace una enorme cantidad de trabajo, tiene una alta cohesión. Estos elementos incluyen clases, subsistemas, y así sucesivamente.
- Una clase con baja cohesión, hace muchas cosas no relacionadas, o hace demasiado trabajo. Clases de baja cohesión a menudo representan un “grano muy grande” de abstracción, o han asumido responsabilidades que deberían ser delegadas a otros objetos.
- Estas clases son indeseables, adolecen de los siguientes problemas.
    - Difícil de comprender
    - Difícil de utilizar
    - Difícil de manejar
    - Difícil de mantener
    - Delicada estructura (constantemente afectada por cambios)

Les suelen llamar clases de utilidad (es un cajón de sastre donde meter todo lo que no me cabe)

Solución: Asignar una responsabilidad para que la cohesión siga siendo alta.
Ejm: si calculo en una clase el mes, es muy probable que necesite el año y el dia tambien, eso quiere decir que estos datos están estrechamente relacionados y la cohesión entre ellos es alta.
Discusión:

- Alta cohesión es un principio a tener en cuenta en todas las decisiones de diseño, es un objetivo fundamental para considerar continuamente. Es un principio valorativo que un diseñador aplica al evaluar todas las decisiones de diseño.
- El patrón de alta cohesión como muchas cosas en la tecnología de objetos tiene una analogía del mundo real. Es una observación común que si una persona toma demasiadas responsabilidades no relacionadas, en especial las que adecuadamente se deben delegar a otros, entonces la persona no es eficaz.

Discusión:

- Escenarios que ilustran diversos grados de cohesión funcional.
    - Muy baja cohesión: Una clase es la única responsable de muchas cosas en muy diferentes áreas funcionales.
    - Baja cohesión: Una clase tiene una responsabilidad exclusiva de una tarea de una tarea compleja en un área funcional.
    - Cohesión moderada: Una clase tiene responsabilidades ligeras y únicas en unas pocas áreas diferentes que están lógicamente relacionadas con el concepto de la clase, pero no entre sí.

    - Alta cohesión: Una clase tiene responsabilidades moderadas en un área funcional y colabora con otras clases para cumplir con las tareas. Como regla general, una clase con alta cohesión tiene un número relativamente pequeño de métodos, con una funcionalidad muy relacionadas, y no hace demasiado trabajo. Colabora con otros objetos para compartir el esfuerzo si la tarea es grande.

Contraindicaciones. Hay unos pocos casos en los que se justifica la aceptación de una cohesión menor

- Uno de los casos es la agrupación de responsabilidades o código en una clase o componente para simplificar el mantenimiento de una persona aunque se advierte que tal agrupación también puede hacer el mantenimiento peor: Clases de Utilidad
- Otro caso para los componentes con menor cohesión es con objetos de servidores distribuidos. Debido a las implicaciones generales y de rendimiento asociados con objetos remotos y la comunicación a distancia, a veces es deseable para crear menos y más grandes objetos servidores menos cohesivos que proporcionan una interfaz para muchas operaciones.

Beneficios:

- Se aumenta la claridad y facilidad de comprensión del diseño.
- Se simplifican mantenimiento y mejoras .
- Alta Cohesión es a menudo compatible con Bajo Acoplamiento
- El grano fino de la funcionalidad altamente relacionada soporta una mayor reutilización porque una clase cohesiva se puede utilizar para un propósito muy específico

Bajo acoplamiento

¿Que es el acoplamiento?

El acoplamiento es un grado de dependencia de otros módulos, de unas clases con otras clases. Necesito que exista esa clase para existir yo. Dependo de esa clase. De todas las clases que heredó, mencionó, uso , etc yo dependo entonces estoy acoplado a ellas.

Si son clases estáticas, las uso igual estoy acoplado, como a las clases de utilidad.

Problema. ¿Cómo apoyar el bajo acoplamiento, el bajo impacto del cambio y el aumento de la reutilización?

- El acoplamiento es una medida de la fuerza con un elemento está conectado a, tiene conocimiento de, o se basa en otros elementos. Un elemento con bajo (o débil) acoplamiento no es dependiente de muchos otros elementos; "demasiados" es dependiente del contexto, pero se examinará. Estos elementos incluyen clases, subsistemas, sistemas, y así sucesivamente.
- Estas clases pueden ser indeseables; algunas sufren los siguientes problemas:
    - Los cambios en las clases relacionadas fuerzan cambios locales.
    - Más difícil de entender de manera aislada.
    - Más difícil de reutilizar porque su uso requiere la presencia adicional de las clases de las que es dependiente.

Solución. Asignar responsabilidades de tal manera que el acoplamiento siga siendo bajo.

Discusión:

- Formas de acoplamiento directo de un Tipo X a un Tipo Y incluyen:

    - Tipo X tiene un atributo que hace referencia a una instancia Tipo Y.
    - Tipo X tiene un método que hace referencia a una instancia de Tipo Y, por cualquier medio. Estos suelen incluir un parámetro o variable local de tipo Tipo Y, o el objeto de retorno de un mensaje es una instancia de Tipo Y.
    - Un objeto Tipo X pide a los servicios de un objeto Tipo Y.
    - Tipo X es una clase derivada directa o indirecta de Tipo Y. Existe una tensión entre los conceptos de acoplamiento y herencia: las clases débilmente acopladas son deseables; y la herencia, la cual acopla parejas de superclases y sus subclases, nos ayuda a explotar el factor común entre las abstracciones.
- Forma de acoplamiento indirecto de un Tipo X a un Tipo Z será cuando el primero envía mensajes a objetos de Tipo Z que devuelvan los objetos de Tipo Y por un acoplamiento directo

Discusión:

- No hay una medida absoluta de acoplamiento cuando es demasiado alto. Lo que es importante es que un desarrollador puede medir el grado actual de acoplamiento, y evaluar si el aumento dará lugar a problemas. En general, las clases que son inherentemente de naturaleza muy genéricas, y con una alta probabilidad para su reutilización, deberían tener especialmente bajo acoplamiento.
- El Bajo Acoplamiento alienta la asignación de una responsabilidad de forma que su colocación no aumente el acoplamiento a un nivel tal que conduce a los resultados negativos que el alto acoplamiento puede producir.
- Un grado moderado de acoplamiento entre clases es normal y necesario para la creación de un sistema orientado a objetos en el que las tareas se cumplen por una colaboración entre los objetos conectados

Discusión:

- El Bajo Acoplamiento es un principio a tener en cuenta en todas las decisiones de diseño; es un objetivo fundamental para considerar continuamente.
- En la práctica, el nivel de acoplamiento por sí solo no puede considerarse en forma aislada de otros principios como el Experto y Alta Cohesión. Sin embargo, es un factor a considerar en la mejora de un diseño.
- El alto acoplamiento per se no es el problema; es el acoplamiento a elementos que son inestables en alguna dimensión, como su interfaz, su implementación, o su mera presencia. Podemos añadir flexibilidad, encapsular datos e implementaciones y diseñar en general con bajo acoplamiento en muchas áreas del sistema . Pero si ponemos esfuerzos en “posibles futuros" o bajando el acoplamiento en algún punto en el que, de hecho, no hay motivación realista, esto no será un tiempo bien empleado. Los diseñadores tienen que elegir sus batallas en la reducción de acoplamiento y centrarse en los puntos de alta inestabilidad o evolución realista.

Contraindicaciones:

- Alto Acoplamiento a elementos estables y generalizados rara vez es un problema. Por ejemplo, una aplicación Java J2EE de forma segura se puede acoplar a las

bibliotecas de Java (java.util, y así sucesivamente), porque son estables y generalizadas

Beneficios:

- No se ve afectado por los cambios en otros componentes
- Fácil de entender de manera aislada
- Conveniente para reutilizar

Lo verdaderamente peligroso no es el acoplamiento en sí, sino estar acoplado a un montón de clases inestables que no paran de cambiar y me hacen inestable a mi.
Ejm: acoplarse al string de java es irrelevante porque no va a cambiar en muchísimo tiempo.

Clases grandes

Errores

- Clases grandes

Mejora

- Estrategias de análisis
- alta cohesión
- bajo acoplamiento
- Invención Pura
- Librería incompleta

Lo interesante es lograr el menor acoplamiento.

Invención Pura

- **Problema**: Hay muchas situaciones en las que la asignación de responsabilidades solamente a las clases de software de la capa de dominio da lugar a problemas en cuanto a la escasa cohesión, elevado acoplamiento o de bajo potencial de reutilización
- Solución: Asignar un conjunto altamente cohesivo de responsabilidades a una clase artificial o de conveniencia que no representa un concepto (algo dominio del problema) , para apoyar alta cohesión, bajo acoplamiento, y reutilización
    - Ejemplo: “cadena de caracteres poética” con responsabilidades sobre las rimas que no se desean asignar a la “cadena de caracteres” general; “formateador de intervalos” para distintas presentaciones.

Si con lo que estoy haciendo mirando la capa del dominio, da lugar a problemas en cuanto a la escasa cohesión, o elevado acoplamiento o de bajo potencial de reutilización, cuando con lo que hemos visto antes no te vale y no se cumple lo que se tiene que cumplir, entonces inventate lo que te de la gana (Invención pura) inventate una nueva clase que te esta afectando en la cohesión o el acoplamiento. Si yo tengo una clase que se relacione con 20. Inventate una clase que se relacione ya no con veinte sino con meno y la clase principal se relacione con menos también