# Preguntas POO

### ¿Qué es un Objeto?

Es una entidad con datos almacenados y tareas realizables durante un tiempo de ejecución. Un objeto puede ser creado instanciando una clase o mediante escritura directa de código y la replicación de otros objetos.

### ¿Qué es una Clase?

Es una plantilla que tiene como finalidad u objetivo la creación de objetos. Se utilizan para representar entidades o conceptos.

### ¿Qué es una Instancia?

Es un miembro de una clase con atributos en lugar de variables.

### ¿Qué es un Atributo?

Es una especificación que define la propiedad de un objeto, elemento o archivo.

### ¿Qué es un Método/función?

Es una subrutina cuyo código es definido en una clase. Si pertenece a una clase se le denomina métodos estáticos, si pertenece a un objeto se le denomina métodos de instancia.

### ¿Qué es una Subrutina?

Es un subalgoritmo del algoritmo principal que permite resolver una tarea específica.

### ¿Qué es un Subalgoritmo?

Es un método de programación que sirve para dividir las partes de un algoritmo y resolver cada una de ellas particularmente.

### ¿Qué es un Diagrama de Clases?

Es un tipo de diagrama con estructura estática que describe la estructura de un sistema mostrando sus clases, atributos, funciones, y las relaciones entre los objetos.

### ¿Qué es un namespace?

Es “un contenedor abstracto” en el que un grupo de uno o más identificadores únicos pueden existir.

### ¿Qué son las funciones inline?

Es aquella función que, debido a su brevedad, puede ser desarrollada en el propio archivo .h. Así, el compilador hace una sustitución de la llamada por el código de dicha función, ahorrando tiempo de ejecución y memoria.

inline int getResultado(int total){

return total;

}

### ¿Qué es el Constructor?

Es una subrutina cuya misión es inicializar un objeto de una clase.

### ¿Qué es el Destructor?

Es una función cuyos principales cometidos son:

* Liberar los recursos computacionales que el objeto de dicha clase haya adquirido en su tiempo de ejecución, una vez ha terminado.
* Quitar los vínculos que pudiesen tener los recursos u objetos usados.

### ¿Qué es una Referencia?

Es un indicativo hacia un objeto, es decir, es una variable cualitativa que contiene la dirección de otra variable cualitativa.

### Iniciadores de Miembros

Si los objetos de la clase necesitan ser inicializados, ésto debe hacerse con iniciadores de miembros.

### Constructor de Copia

MiClase obj1;

Podemos declarar otro objeto (obj2) haciendo una copia de obj1 de la siguiente manera.

MiClase obj2(obj1);

O bien podemos declarar otro objeto (obj3) asignándole el objeto obj1 de la siguiente forma.

MiClase obj3 = obj1;

### Pair

Es una plantilla de clase que proporciona una forma de almacenar dos objetos distintos como una sola unidad.

### Tuple

Es una colección de tamaño fijo de valores distintos.

### Variables Miembro Estáticas o Static Members

Cuando se usan, solo una copia del miembro estático es compartida por todos los objetos de una clase en un programa.

### Funciones miembro static

Al declarar un miembro como estático lo haces independiente de cualquier objeto de la clase. A una función miembro estática se le puede llamar incluso si no existen objetos de la clase.

Ej: nombre\_clase::nombre\_funcion()

Uso:

* Agrupar funciones independientes en un mismo módulo.
* Poder usarlas sin tener que declarar objetos de su clase.

### Polimorfismo

Proceso mediante el cual se puede acceder a diferentes implementaciones de una función utilizando el mismo nombre.

### Funciones Virtuales

Las funciones virtuales se usan cuando algunas de las funciones que forman parte de una interfaz compartida no pueden concretar las clases derivadas que no están totalmente definidas.

* Definen la interfaz genérica.
* No se pueden definir objetos de estas clases.
* Son usadas para derivar de ellas y para el polimorfismo.

### Tratamiento de señales (Excepciones)

Dentro de un bloque try se pretende evaluar una o más expresiones. Si dentro de dicho bloque se produce algo que inesperado, se lanza por medio de throw una excepción, la misma que será capturada por un catch específico.

* Puede haber más de un catch y throw.
* Si hay un throw sin su catch correspondiente, el programa se interrumpirá abruptamente después de haber pasado por todos los catchs que se hayan definido y de no haber encontrado el adecuado.
* Tipos de excepciones: int, float, char, etc.

### Punteros Inteligentes o Smart Pointers

Es un tipo abstracto de dato que simula el comportamiento de un puntero. Gestiona automáticamente la memoria mediante un registro y comprueba los límites para reducir errores causados por el mal uso de punteros, manteniendo la eficiencia.

### Map

Permite asociar etiquetas a valores (cada etiqueta tiene un único valor asociado).

map<v1,v2> n;

### Herencia, iniciadores base

Si el constructor de la clase base tiene un parámetro obligatorio, éste debe pasarse en un iniciador base de su clase original.

### Puntero this

Es un parámetro implícito a todas las funciones miembro de una clase, es decir, todas las funciones miembro de una clase lo reciben automáticamente a excepción, de las funciones estáticas.

### ¿Qué es una función Friend en C++?

### Es aquella que, no siendo función miembro de la clase, puede acceder a su parte privada.

### Este mecanismo, que puede parecer una contradicción a la filosofía del encapsulamiento, se permite para que dos clases puedan compartir funciones de utilidad para ambas, que acceden a sus datos privados en orden a mejorar la eficiencia al programar.

### Sección protected

* El desarrollador/a de la clase derivada puede acceder a la parte protected de la clase base.
* El desarrollador/a de la clase base permite que las clases derivadas accedan a la sección protected.

### Algorithm Library

Define funciones para una serie de finalidades (la ordenación, la búsqueda o el recuento) que operan con rangos de elementos.

### Ficheros (#include <fstream>)

* ifstream, derivada de istream, para los ficheros de entrada.
* ofstream, derivada de ostream, para los ficheros de salida.
* fstream, derivada de iostream, para los ficheros de entrada y salida.

### Describe el patrón de diseño Builder

El patrón de diseño Builder ayuda a las construcciones de objetos complejos (con muchos atributos internos o en constructores con muchos parámetros) definiendo un objeto intermedio que ayude con la definición de objeto compuesto.

### Describe el patrón de diseño Factory

### Patrón de diseño creacional. Facilita la creación de instancias de diferentes clases pertenecientes a una misma familia. La diferencia con Builder es que los subproductos son objetos distintos. Sirve para crear objetos diferentes de una misma familia en una sola llamada.

### Existen dos implementaciones básicas:

### Mediante una utility class (Abstract Factory), que define la interfaz de cada sub-factory.

### También mediante una utility class pero con un método static (“Factory Method”), que recibe una descripción del objeto deseado y devuelve un puntero de la clase base que apunta al objeto deseado.

### Facilita la creación y manipulación de objetos de la misma familia y la ampliación a más clases de la familia.

### Define el criterio para evaluar la descomposición Comprensibilidad modular

### Un método favorece la Comprensibilidad Modular si ayuda a producir software en el cual un lector humano puede entender cada módulo sin tener que conocer los otros, o, en el peor caso, teniendo que examinar sólo unos pocos de los restantes módulos.

### Afecta mucho a la comprensión y al mantenimiento. Se suele asumir que si nos resulta difícil expresar qué hace o para qué se ha creado un módulo, ese módulo no está bien diseñado. Los módulos demasiado grandes tampoco ayudan, además, son poco reutilizables. Define el contexto de invariante de clase y escribe dos ejemplos ¿Cuándo es correcta una clase?

Es una propiedad que hace que un objeto esté bien definido, caracterizado por su utilidad y simpleza. Es una verificación práctica del estado de un objeto. class fecha fecha correcta class Ruleta fecha banca>0 Una clase es correcta si su implementación es consistente con las precondiciones, postcondiciones y las invariantes.

### Beneficios de la reutilización

### Rapidez de desarrollo, fiabilidad, eficiencia, menor mantenimiento, mayor consistencia en los desarrollos, mejora costes y es una inversión rentable y se espera la mejoría de todos los factores de calidad.

### Codifica una función en C++ que intercambie el valor de dos variables int pasadas por referencia

int intercambio(int &x, int &y){

int aux;

aux = x;

x = y;

y = aux;

}

### Describe los factores de calidad del software integridad y accesibilidad. Indica también si son factores internos o externos. Pon un ejemplo (puedes inventar un ejemplo o bien utilizar el caso de algún software conocido).

Integridad (integrity): el producto no debe corromperse ni comprometerse ni ser vulnerable en ninguna de sus partes, ni siquiera por el hecho de su utilización masiva o por una gran acumulación de datos. Un sistema que no es seguro pierde su integridad con facilidad. Es un factor externo.

Ejemplo: el sistema que controla los movimientos bancarios.

Accesibilidad informática: consiste en el acceso a la información sin limitación alguna. Por tanto, un programa accesible es aquel que permita el acceso a la información sin limitación alguna con el fin de poder elaborar, reproducir, manipular, etc. dicha información, así como el acceso sin ningún tipo de limitación a las herramientas y opciones de dicho programa.

Ejemplo: el jefe de una empresa accede al sistema para ver y/o modificar al personal que tiene contratado.

### ¿Qué es un patrón de diseño?

Un patrón de diseño es una descripción de las clases y objetos que lo componen, y de sus relaciones entre sí. Cada patrón está especializado en resolver un problema de diseño general en un determinado contexto. Las ventajas que implica su uso son diseños más sencillos, menos costosos y de más calidad.

### ¿Qué significa encapsulado y ocultación de la información en TADs? ¿Qué utilidad tiene?

Significa que un TAD tiene una vista exterior, en la que se ocultan detalles, estructuras de datos y los secretos de su comportamiento interno y una vista interior, en la que se puede ver cómo está· hecha por dentro y cómo se comporta internamente cada operación, las estructuras de datos utilizadas, el código, etc.

La separación clara de estas dos vistas es muy importante, ya que en el diseño de un TAD no puede intervenir la vista interna. Igual ocurre con el cliente del TAD, para el que no debe ser necesario conocer la vista interna de un TAD para poder usarlo.

Esta separación vista interna/externa es como si la parte interna del TAD estuviera oculta o encapsulada con respecto a la vista externa. El encapsulado es un pilar de la programación con TADs.

Sirve para facilitar el diseño y hacerlo de más calidad, además de hacer la labor del cliente más sencilla y potente. También evita que nadie modifique el estado de forma inadecuada, protege de operaciones no permitidas sobre los objetos, simplifica la comprensión del objeto (no ser· necesario conocer sus datos internos) y, si en el futuro se modifican los datos internos de un objeto, no afectará a ningún programa que use dicho objeto, ya que nunca se accede a los datos internos.

### Define los factores de calidad del software robustez e integridad.

Robustez (reliability): es la capacidad del producto de comportarse con exactitud, de manera satisfactoria y precisa en todas las situaciones. Por tanto, manejar correctamente situaciones imprevistas, de error, o fuera de lo normal. Esta característica es imprescindible en infraestructuras críticas y sistemas de gran responsabilidad, sistemas en tiempo real, etc. Es un factor externo.

Integridad (integrity): el producto no debe corromperse ni comprometerse ni ser vulnerable en ninguna de sus partes, ni siquiera por el hecho de su utilización masiva o por una gran acumulación de datos. Un sistema que no es seguro pierde su integridad con facilidad. Es un factor externo.

### ¿Qué es MVC? ¿Qué componentes tiene y cuál es su utilidad?

MVC es una triada de patrones. Sus componentes son: un modelo (objeto), una vista (su representación) y un controlador (modo en el que reacciona el usuario). Usa las propiedades de varios patrones de diseño que cooperan: observer, composite, strategy, etc.

Se emplea siempre en el desarrollo de aplicaciones web. Está presente en casi todos los frameworks de desarrollo modernos. Se puede aplicar a cualquier aplicación.

Provoca una simplificación del desarrollo, separa desarrollos y proporciona varias presentaciones para un mismo modelo.

### Diferencia entre herencia pública y privada.

La herencia consiste en definir una clase a partir de otra previamente definida: es el proceso por el que una clase hereda las propiedades y funcionalidades de otras clases.

En la herencia pública, los elementos public de la clase base siguen siendo public y los protected siguen siendo protected en la clase derivada, es decir, pueden ser accedidos desde la clase derivada y se heredan. Los usuarios de la clase derivada podrán acceder únicamente a la parte pública de la clase base y la derivada. Este tipo de herencia es el más habitual.

En cambio, en la herencia privada, los elementos públicos y protected de la clase base pasan a ser privados en la clase derivada y pueden ser accedidos desde la clase derivada, pero no se heredan a otras clases ni pueden ser accedidos por los usuarios de la clase derivada. Es el modo por defecto.

### Describe eficiencia. Indica también si es un factor interno o externo. Pon un ejemplo

Eficiencia (efficiency): se trata de conseguir que el programa, además de realizar correctamente aquello para lo que ha sido creado, lo realice de la mejor forma posible. Algunos factores que afectan a la eficiencia son: el espacio en memoria utilizado por el programa y el tiempo de ejecución (los dos más importantes en la mayoría de los casos), el espacio en disco, la memoria temporal, número de accesos a determinado hardware (de comunicaciones, por ejemplo), etc. Es un factor externo e interno.

Por ejemplo, si tenemos dos programas que calculan la suma de una serie de números, uno lo hace con un bucle y otro de forma manual, ser· mejor el del bucle, ya que tardar· menos tiempo.

### Describe los principios para una buena de composición modular: Acceso uniforme y Principio abierto-cerrado.

Acceso uniforme: todos los servicios ofrecidos por un módulo deben estar disponibles a través de una notación uniforme, sin importar si están implementados a través de almacenamiento o de un cálculo. La uniformidad del sistema es importante- Ya que el usuario hace un esfuerzo por entender el módulo, no le cambiemos la nomenclatura, las reglas básicas, etc. en el otro módulo.

Principio Abierto-cerrado: puede parecer contradictorio, pero son elementos de naturaleza diferente.

Los módulos deben ser a la vez abiertos y cerrados.

* Abierto: esforzarse en diseñar el módulo de manera que quede abierto a facilitar la posterior modificación, ampliación, etc. (considerar que el 70% del coste del software se dedica al mantenimiento). Para ello se diseñan arquitecturas abiertas, se siguen estándares y convenciones, etc.
* Cerrado: el módulo estar· cerrado si está· disponible para ser usado e integrado en el sistema. Su diseño no impedir· que el módulo quede completado y listo para ser usado en la aplicación para la que se ha creado

### Comenta al menos 2 usos del calificador “const” en C++

Se usa, por ejemplo, para definir un valor constante al tamaño de un vector o para definir que un objeto tenga invariable los atributos durante el tiempo de ejecución.

### Describe el patrón de diseño iterator y observer

Iterator: es un patrón de diseño de comportamiento. Proporciona el acceso secuencial a los distintos elementos de un agregado, sin exponer su representación interna (independencia de la representación interna). Por ejemplo, para recorrer una lista. Es sencillo de usar y proporciona un acceso uniforme a todas las colecciones.

Ejemplos: C++: STL iterators. Phyton: for...

Observer: es un patrón de diseño de comportamiento. Permite la suscripción siempre actualizada a los datos. Define una dependencia del tipo uno a muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes.

Tiene una estructura definida por el par: sujeto-observador. El sujeto tiene los datos que se publican y los observadores son los objetos que se nutren de esos datos.

Aplicaciones: infinidad de aplicaciones tienen este esquema. Está presente en el patrón de diseño MVC.

Por ejemplo, el sujeto puede ser una base de datos y un observador puede ser una estadística, una gráfica, una hoja de cálculo…

### ¿Qué son las referencias en C++?

Las referencias en C++ pueden verse como alias o nombres alternativos que pueden darse a una variable u objeto. Al definirse siempre debe indicarse a qué variable hace referencia, no pueden definirse sin hacer referencia a una variable o un objeto. Esto las hace muy seguras, ya que evitamos que una referencia apunte a una zona errónea.

Se usan sobre todo en paso de parámetros a funciones: sirve para que una función cree una referencia al objeto (y no una copia) y si se modifica la referencia se modificar· el objeto original. Así, se evita el proceso de copia del objeto y el consecuente gasto en memoria y tiempo de ejecución.

### Describe los factores de calidad del software robustez y portabilidad. Indica tambiÈn si son factores internos o externos. Pon un ejemplo (puedes inventar un ejemplo o bien utilizar el caso de algún software conocido).

Portabilidad (portability): es la capacidad o facilidad del producto de ejecutarse en otro hardware diferente o en otro sistema operativo diferente. Aquí es muy importante que el programa no haga uso de características de bajo nivel del hardware o que aísle la parte dependiente del hardware para que al portarlo solo sea necesario modificar dicha parte. Es un factor externo e interno.

Por ejemplo, una aplicación que pueda usarse tanto en Windows como en Linux o Mac OS (Spotify…).

### Describe los factores de calidad del software fiabilidad y compatibilidad. Indica también si son factores internos o externos. Añade a cada explicación un ejemplo (puedes inventar un ejemplo o bien utilizar el caso de algún software conocido)

Fiabilidad (o eficiencia): se trata de conseguir que el programa, además de realizar correctamente aquello para lo que ha sido creado, lo realice de la mejor forma posible. Algunos factores que afectan a la eficiencia son: el espacio en memoria utilizado por el programa y el tiempo de ejecución (los dos más importantes en la mayoría de los casos), el espacio en disco, la memoria temporal, número de accesos a determinado hardware (de comunicaciones, por ejemplo), etc. Es un factor externo e interno.

Por ejemplo, si tenemos dos programas que calculan la suma de una serie de números, uno lo hace con un bucle y otro de forma manual, ser· mejor el del bucle, ya que tardar· menos tiempo.

Compatibilidad (interoperability): es la facilidad que tienen los programas de combinarse entre sí. Es la posibilidad de utilizar los datos de salida de un programa como entrada de otro programa. La clave es la estandarización y el consenso entre organismos, instituciones y empresas, en primera instancia, y entre programadores en ˙última instancia. Es un factor externo.

Por ejemplo, si un programa le indica una orden al sistema y este la interpreta y la ejecuta sin problemas.

### ¿Qué es la STL de C++?

Es la Standard Template Library de C++. En ella existen una serie de clases y plantillas definidas que se usan fácilmente en los programas. A muchos de estos tipos de les da el nombre de “contenedores”, porque de forma genérica pueden “contener” internamente cualquier tipo base.

Están los contenedores clásicos: vector, list y deque. Y también existen otros contenedores como los asociativos: multiset, set, map y multimap.

### Define continuidad modular y protección modular.

Un método satisface la Continuidad Modular si en el sistema resultante desarrollado con

ese método, un pequeño cambio en la especificación o en los requisitos provoca sólo

cambios en un único módulo o en un número reducido de ellos.

Un método satisface la Protección Modular si produce arquitecturas en las cuales el

efecto de una situación anormal que se produzca dentro de un módulo durante la

ejecución queda confinado a dicho módulo, o en el peor caso se propaga sólo a unos

pocos módulos vecinos.

Un error dentro de un módulo debe ser tratado en dicho módulo y solucionado allí si es

posible. Si un error dentro de un módulo no es detectado por el módulo y siguen

ejecutándose otros módulos arrastrando el error, éste será difícil de corregir y de

impredecibles consecuencias.

### Selecciona verdadero o falso.

* La librería STL de C++ significan Standard Technology Library.

Falso: Standard Template Library.

* En C++ ya no hace falta usar guardas de inclusión.

Falso: sí hay que usarlas en todos los ficheros de declaración para que al ser incluidos posteriormente por otros programas no haya problemas de redefinición.

* En C++ para compilar hacemos uso de GCC, GNU Compiler Collection.

Verdadero.

* Los 4 pilares de la POO son Abstracción, Herencia, Encapsulamiento y Polimorfismo.

Verdadero.

* Si heredamos de forma privada una clase A, en una clase B, un objeto B solo podrá acceder a los elementos públicos de B.

Falso: en la herencia privada, los elementos públicos y protected de la clase base pasan a ser privados en la clase derivada y pueden ser accedidos desde la clase derivada, pero no se heredan a otras clases ni pueden ser accedidos por los usuarios de la clase derivada.

### Define qué es un patrón de diseño. Describe los patrones de diseño Template Method y Parameterized Types. Comenta las ventajas del uso de cada uno de ellos

Es un concepto utilizado en la etapa de diseño del software. Es una solución reutilizable a un problema de diseño de software: soluciones de diseño elegantes y de calidad a problemas complejos. Son elementos en el diseño de software que se repiten una y otra vez.

Está compuesto por objetos, clases y las relaciones entre ellos.

Cada patrón está especializado en resolver un problema de diseño concreto en un determinado contexto.

Las ventajas que implica su uso son: diseños futuros más sencillos, menos costosos y de más calidad.

Template Method: es un patrón de diseño estructural. Separa las cosas que cambian de las que no cambian. Un método describe un comportamiento que se concreta en las clases derivadas.

* Aplicaciones: un mismo proceso se concreta de forma diferente en distintas clases.
* Ventaja: modificabilidad.

Parameterized Types: también llamado “generics”. Es un patrón de diseño estructural. Se define un nuevo tipo sin especificar los tipos de todos sus componentes.

* Aplicaciones: genericidad. Templates en C++ (STL, plantillas de función y de clase).
* Ventaja: genericidad.

### ¿Qué significa la palabara re-escribir (override) que se utiliza en algunas funciones? ¿De qué tipo de funciones estamos hablando?

Es un modificador que extiende el método de clase base virtual. De hecho, desde C++11, en la clase derivada es preferible el uso de override, dejando virtual para la clase base. Estamos hablando de las funciones virtuales.

### Software de calidad

Podemos clasificar las cualidades del software en dos tipos: calidad funcional (grado de acercamiento del funcionamiento del software a lo que se determinó en sus especificaciones y requisitos funcionales) y calidad estructural (cumplimiento de características internas no-funcionales del software que dan soporte a otras características internas del software también deseables como la mantenibilidad, facilidad de extensión, etc.)

Los factores de calidad se clasifican en tres bloques (McCall en 1977):

* Operación del producto, transición del producto y revisión del producto.
* Operación del producto: corrección, robustez, eficiencia, integridad y facilidad de uso.
* Transición del producto: portabilidad, reutilidad/reusabilidad y compatibilidad.
* Revisión del producto: mantenibilidad, extensibilidad y testable. Otros factores importantes: seguridad, accesibilidad, oportunidad y economía.

### Corrección (correctness)

Es la capacidad del producto para realizar de forma adecuada aquello para lo que fue creado, tal como se definió en los documentos de especificación y requerimientos. Es un factor externo.

### Corrección (correctness)

Es la capacidad del producto para realizar de forma adecuada aquello para lo que fue creado, tal como se definió en los documentos de especificación y requerimientos. Es un factor externo.

### Reutilidad/reusabilidad (reusability)

Es la capacidad del producto de ser reutilizado en su totalidad o en parte por otros productos, con el objetivo de ahorrar tiempo en soluciones redundantes ya hechas con anterioridad. De esta forma, un programa debe agrupar en módulos aislados los aspectos dependientes de la aplicación particular, mientras que las utilidades que puedan generalizarse deben mantenerse preparadas para ser utilizadas por otros productos y así ahorrar costes de producción. Diseño por capas de diferente nivel de abstracción. Es un factor interno.

### Mantenibilidad (maintainability)

Capacidad de encontrar y corregir un defecto en el software. Es un factor interno.

### Extensibilidad (flexibility)

Facilidad de adaptar el producto a cambios en la especificación de requisitos. Cuanto mayor es el programa suele costar más esfuerzo, y si no se aplica un buen diseño, y sobre todo simple, este factor puede verse muy perjudicado. Es un factor interno.

### Testable (testability)

Habilidad de poder validarse el software. Es un factor interno.

### Seguridad

Es la capacidad del producto de proteger sus componentes de usos no autorizados y de situaciones excepcionales de pérdida de información. Para ello debe prever mecanismos de control de acceso, encriptación, clave de acceso y también, desde el otro punto de vista, copias de seguridad, procesos rutinarios de comprobación, etc. Es un factor externo.

### Composite

Patrón de diseño estructural. Se crean jerarquías de manera que se tratan igual a objetos individuales que a los compuestos.

Clases abstractas abiertas a incorporación de nuevos componentes que se tratarán igual.

Los objetos se tratan de manera uniforme sean primitivas o grupos.

Aplicaciones: allí donde se quiera simplificar la interfaz, sean primitivas, sean grupos complejos de objetos.

Consecuencias: simplificación, interfaz sencilla, acceso uniforme.

### Strategy

Patrón de diseño de comportamiento. Familia de algoritmos intercambiables. Se prepara una descripción genética del algoritmo para que posteriormente se instancie con el que convenga. Se prepara la “llamada” al algoritmo que ser· el m·s conveniente en cada caso. Define un conjunto de algoritmos encapsulados que pueden ser intercambiados para llevar a cabo un comportamiento específico.

Ejemplo: intercambiar el método de ordenación para un conjunto de datos. Otras estrategias: para buscar, clasificar, filtrar, etc.

Aplicaciones: cuando preveamos distintos comportamientos en un futuro, podemos habilitarlo. Estructuras de datos complejas que podrán implementarse en un futuro de otras formas.

Consecuencias: posibilidad de mejorar eficiencias y rendimientos en el futuro. Permitir otras estrategias de solución distintas a la propuesta inicialmente. Facilitar ampliaciones.

Es parecido a Template Method. Template Method define el esqueleto de una operación que puede ser diferente seg˙n cada subcase concrete los placeholders. Se resuelve en tiempo de compilación.

Strategy es en tiempo de ejecución. El objeto recibe la estrategia como parámetro y en tiempo de ejecución se cambia. Cambia seg˙n el contexto.

### Factory vs Builder

Builder facilita crear y manipular objetos de la misma clase pero con diferente configuración

Factory facilita crear y manipular objetos de diferente clase pero de la misma familia con una sencilla llamada a una función.

### Singleton

Es un patrón de diseño creacional. Nos asegura que solo exista una única instancia/objeto de una clase. Es útil cuando se necesita exactamente un objeto para coordinar acciones a través del sistema.

Encapsula un recurso del que solo hay una instancia y lo pone a disposición de toda la aplicación. Puede ser hardware, un servicio, un dato global, etc.

Variables globales se administran mejor en una clase con una única instancia.