What is the Singleton pattern, and what problem does it solve in software design?	Describe the structure of the Singleton pattern. What are the main components involved?	What are the benefits of using the Singleton pattern in software development? Provide some practical examples.
The Singleton pattern solves two problems at the same time, violating the Single Responsibility Principle. Ensure that a class has just a single instance. Provide a global access point to that instance.	All implementations of the Singleton have these two steps in common: Make the default constructor private, to prevent other objects from using the new operator with the Singleton class. Create a static creation method that acts as a constructor. Under the hood, this method calls the private constructor to create an object and saves it in a static field. All following calls to this method return the cached object.	The government is an excellent example of the Singleton pattern. A country can have only one official government. Regardless of the personal identities of the individuals who form governments, the title, “The Government of X”, is a global point of access that identifies the group of people in charge.
Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.	The Singleton class declares a static method that returns the same instance of its own class.The Singleton’s constructor should be hidden from the client code. Calling the method should be the only way of getting the Singleton object.	- You can be sure that a class has only a single instance. - You gain a global access point to that instance. - The singleton object is initialized only when it’s requested for the first time. The Singleton pattern disables all other means of creating objects of a class except for the special creation method. This method either creates a new object or returns an existing one if it has already been created.
O padrão consegue garantir que um objeto tem apenas uma instância e apenas um ponto global de acesso. Impede que vários objetos iguais sejam criados.	Simplesmente uma class Singleton com um metodo getInstance()	O metodo singleton só é inicializado a primeira vez depois retorna sempre o mesmo objeto e garante que não há varios pontos de entrada para o mesmo. A class Runtime do java.
Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance. Solving two problems: Ensure that a class has just a single instance and Provide a global access point to that instance. The singleton pattern states that each object should only have one instance.	The Singleton class declares the static method getInstance that returns the same instance of its own class. The Singleton’s constructor should be hidden from the client code. Calling the getInstance method should be the only way of getting the Singleton object. .	You can be sure that a class has only a single instance. You gain a global access point to that instance. The singleton object is initialized only when it’s requested for the first time. Example: The government is an excellent example of the Singleton pattern. A country can have only one official government. Regardless of the personal identities of the individuals who form governments, the title, “The Government of X”, is a global point of access that identifies the group of people in charge. It helps access and control instances of objects
The Singleton pattern is a creational design pattern that ensures a class has only one instance and provides a global point of access to that instance. So it restricts the instantiation of a class to a single object. This pattern solves multiple issues like: Global Access, Resource Management and more.	The singleton pattern is structured like this: It has a Singleton Class (which itself has a private constructor), a Static Instance Variable and a Static Method for Accessing Instance (This is to allow other classes to acess the single instance of this one).	Some benefits are: 1.Encapsulation of object creation. 2.Centralized control over object creation logic. 3.Flexibility and extensibility for accommodating changes or additions to object creation. 4.Decoupling client code from concrete implementations, promoting programming against interfaces. 5.Facilitation of other design patterns. Now, for some practical examples, these include: Database Connection Factory, GUI Component Factory, Document Converter Factory, Logger Factory, and Vehicle Factory. In each case, the Factory pattern provides a centralized mechanism for creating objects, promoting encapsulation, flexibility, and decoupling from concrete implementations.
The Singleton pattern is a software design pattern that restricts the instantiation of a class to a single instance, ensuring that only one instance of the class exists in the Java Virtual Machine (JVM). This pattern is useful when exactly one object is needed to coordinate actions across a system. It provides a global point of access to this instance and controls the instantiation of the class, often by making the constructor private The Singleton pattern solves the problem of ensuring that a class has only one instance and provides a global point of access to it. This is particularly useful in scenarios where a single point of control is needed to avoid naming conflicts or collisions, manage resources efficiently, and ensure thread safety	The structure of the Singleton pattern involves a single class with a private constructor to prevent direct instantiation. The class contains a static method that returns the single instance of the class. If the instance does not exist, it is created; otherwise, the existing instance is returned. This ensures that only one instance of the class is created and used throughout the application. The main components of the Singleton pattern are: Singleton Class: The class that is designed to have only one instance. Private Constructor: To prevent other classes from instantiating it. Static Instance Variable: Holds the single instance of the class. Public Static Method: Provides a global point of access to the single instance.	The Singleton pattern is beneficial for managing shared resources or state across an application, reducing complexity and coupling in code, and providing a global access point to a single instance. It is particularly useful in scenarios like configuration settings management, database connection pooling, logging, device communication, and application state management. Some examples in more detail: Configuration Settings: Managing application-wide configuration settings to ensure consistency and reduce the risk of errors. Database Connection Pooling: Efficiently reusing database connections to improve performance and reduce overhead. Logging: Ensuring all log messages are written to the same log file or database for easier tracking and analysis.
O Singleton é um padrão de projeto de software que garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto. Deste modo, o padrão Singleton oferece várias vantagens em relação ao uso de variáveis globais, como garantir que haja apenas uma fonte de verdade e consistência para a classe, ainda reduz o uso de memória e o tempo de inicialização, pois a instância é criada apenas quando necessária.	O Singleton pattern está dividido em três componentes, sendo estas Classe Singleton, Instância Única, Método de Acesso Estático. Singleton:Esta é a classe que implementa o padrão Singleton. Geralmente, ela possui um construtor privado para evitar a criação de instâncias fora da classe e um método estático para acessar a única instância da classe. Instância Única:É a única instância da classe Singleton que é compartilhada por todos os clientes que desejam acessá-la. Método de Acesso Estático:É um método estático na classe Singleton que fornece um meio para os clientes obterem a instância única da classe. Geralmente, esse método é chamado de getInstance().	Encapsulamento da Lógica de Criação de Objetos: O padrão Factory encapsula a lógica de criação de objetos dentro de um único componente, tornando mais fácil de gerenciar e manter. Isso ajuda a reduzir a duplicação de código e garante que o código de criação de objetos esteja centralizado, melhorando a legibilidade e manutenibilidade do código. Flexibilidade e Escalabilidade: O padrão Factory fornece um mecanismo flexível para a criação de objetos, permitindo uma fácil extensão e escalabilidade do sistema. Novas classes concretas podem ser adicionadas ao sistema sem modificar o código do cliente existente, desde que sigam a interface da fábrica. Isto pode ser usado por exemplo em: Frameworks de GUI: Em frameworks de GUI, as fábricas são frequentemente usadas para criar diferentes tipos de componentes de interface do usuário, como botões, campos de texto e painéis. Por exemplo, uma ButtonFactory pode criar instâncias de Button com base no estilo ou tema desejado.
The singleton pattern allows the ensurance that an object will only have one instance, with easy access and control over the constructor.	Private Constructor and a static method that returns the reference to that instance	It minimizes pollution of the namespace and allow for lazy allocation and initializatio. A car can only have one key, therefore, in code, using a singleton to create the key is a best practice, because it guarantees there will only be one

A class has only one instance and provides a global access point to its object. Almost sacrifices all transparency for convenience.

Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.
The Singleton pattern solves two problems :
Ensure that a class has just a single instance.The most common reason for this is to control access to some shared resource—for example, a database or a file.
Provide a global access point to that instance. Just like a global variable, the Singleton pattern lets you access some object from anywhere in the program. However, it also protects that instance from being overwritten by other code.
How to implement:
Make the default constructor private, to prevent other objects from using the new operator with the Singleton class.
Create a static creation method that acts as a constructor. Under the hood, this method calls the private constructor to create an object and saves it in a static field. All following calls to this method return the cached object.

Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.

The Singleton pattern solves two problems at the same time, violating the Single Responsibility Principle:

1 - Ensure that a class has just a single instance. Why would anyone want to control how many instances a class has? The most common reason for this is to control access to some shared resource—for example, a database or a file. Here's how it works: imagine that you created an object, but after a while decided to create a new one. Instead of receiving a fresh object, you'll get the one you already created.

2 - Provide a global access point to that instance. Remember those global variables that you used to store some essential objects? While they're very handy, they're also very unsafe since any code can potentially overwrite the contents of those variables and crash the app.Just like a global variable, the Singleton pattern lets you access some object from anywhere in the program. However, it also protects that instance from being overwritten by other code.

There's another side to this problem: you don't want the code that solves problem #1 to be scattered all over your program. It's much better to have it within one class, especially if the rest of your code already depends on it.

É um padrão de desenho Creational que garante que uma classe tenha apenas uma instância e ao mesmo tempo fornece um ponto de acesso global a essa mesma instância. Resolve o problema de certificar-se de que uma classe tenha apenas uma única instância e fornece um ponto de acesso global a essa mesma instância

The Singleton design pattern is a creational pattern that ensures that a class has a single instance, all the while providing a global access point to said instance. It solves two problems, violating the single responsibility principle. The first being one where when you want to create a new object after another, instead of getting the new object you get the previous one. The second problem it solves is the lack of safety of global variables that can cause overwritting the contents of said variables and possibly crash the app.
The singleton pattern states that an object should only have one instance

Singleton Class: This is the class for which only one instance is desired throughout the application's lifecycle

Static instance: This is a static member variable within the Singleton class that holds the reference to the single instance of the class

GetInstance() Method(or similar): This is a static method (or property) typically named GetInstance, getInstance, or similar, which provides a way to access the single instance of the Singleton class

Private Constructor: The constructor of the Singleton class is made private to prevent external classes from instantiating the class directly

The main components of the Singleton pattern structure are the single class, its private constructor, a public static method that provides access to this instance (and that creates it if it does not exist yet) and a private static member variable that stores the single instance of the class.

The Singleton class declares the static method getInstance that returns the same instance of its own.
In the implementation of a Singleton class, one crucial aspect is to hide the constructor from external access. By making the constructor private, we prevent other classes from instantiating the Singleton directly. Instead, we provide a static method, usually named getInstance, that serves as the entry point for accessing the Singleton instance.

A classe Singleton declara o método estático getInstance que retorna a mesma instância da sua própria classe.
O construtor do Singleton deve estar oculto do código do cliente.
Método getInstance deve ser a única maneira de obter o objeto Singleton.

The singleton pattern structure possesses only the Singleton class, where it declares the getInstace() static method, which returns the same instance of its own class.

.

Encapsulation of Object Creation Logic, flexibility and extensibility, centralized configuration and management, decoupling of client and concrete classes and enhanced testing.

Pratical example:

Game Development: In game development, factories are commonly used for creating various game objects such as characters, weapons, and enemies. Each type of game object may have its own factory responsible for creating instances of that object.

Using the singleton ensures only one entity of a class exists and ensures, for example, that only one object has access to a certain resource (for example a database). It also as a global access point, with no need to create and transfer data between objects.

A practical use for this would be creating a program that simulates a factory, and there must be only one boss of the factory (that can manage the factory, by firing / hiring employees, etc). Only one boss must exist so instead of allowing the client to create innumerous instances of Boss, the boss itself has a method to create itself once and refer to itself if it already exists.
The Singleton pattern disables all other means of creating objects of a class except for the special creation method. This method either creates a new object or returns an existing one if it has already been created.
Unlike global variables, the Singleton pattern guarantees that there's just one instance of a class. Nothing, except for the Singleton class itself, can replace the cached instance.

Here are some other benefits of using the Factory pattern in software development:

- Encapsulation: The Factory pattern encapsulates the object creation process within a separate class or method, hiding the implementation details from the client code. This promotes information hiding and helps maintain a clean and modular design.

- Flexibility and Extensibility: By using factories, you can easily introduce new types of objects or variations without modifying existing client code. This makes the system more adaptable to changes and allows for easier extension in the future.

- Decoupling: The Factory pattern decouples the client code from the concrete classes being instantiated. Clients only need to interact with the factory interface, rather than directly instantiating concrete classes, which reduces dependencies and improves maintainability.

- Centralized Configuration: Factories provide a centralized location for configuring object creation logic. This allows you to easily manage and update instantiation rules without scattering the logic throughout the codebase.

- Consistency: Factories ensure that objects are created in a

O uso do padrão de Singleton tem vários benifícios:
-> garante que uma classe tem apenas uma instância, o que permite o controlo do acesso aos recursos partilhados;
-> fornece um ponto de acesso global à instância, permitindo que outras partes do código acessem facilmente sem serem precisas referências, e melhor a legibilidade do código;
-> a instância é criada apenas quando necessária, evitando o gasto de recursos não necessários;
-> controlo centralizado do ciclo de vida da instância através do seu encapsulamento.

O governo é um excelente exemplo de um padrão Singleton. Um país pode ter apenas um governo oficial. Independentemente das identidades pessoais dos indivíduos que formam governos, o título, "O Governo de X", é um ponto de acesso global que identifica o grupo de pessoas no command.

You can be sure that a class has only a single instance, gain a global access point to that instance, and the singleton object is initialized only when it's requested for the first time.
Accessing and controling instances of objects is easier, since there is only one per object

The Singleton pattern is a design pattern that makes sure only a single instance of a class exists throughout an app and provides access to that instance from anywhere in the code.

Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.

Este design pattern tem como objetivo restringir a criação de instâncias de uma classe a 1 objeto. Desta forma, é assegurada a existência de apenas 1 instância da class, sendo possível atribuir-lhe um ponto de acesso global e único.

The Singleton pattern is a creational design pattern that ensures that a class has only one instance, while giving a global access point to that instance. It solves the problem of controlling access to some shared resource, and also, it lets you access some object from anywhere in the program, protecting it from being overwritten by other code.

The Singleton pattern is a design pattern that ensures a class only has a single instance throughout a program. This is useful when you need exactly one object, like a configuration manager. It solves the problem of accidentally creating multiple instances and provides a central access point for the single one that exists. However, it can also make code tightly coupled and harder to test, so it's not always the ideal solution.

Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.
Problems that it solves: (related with Single Responsibility Principle)
- Ensure that a class has just a single instance
- Provide a global access point to that instance

O padrão Singleton é um padrão de design criacional usado na programação de software para garantir que uma classe tenha apenas uma única instância, ao mesmo tempo em que fornece um ponto de acesso global a essa instância.

O problema que o Singleton visa resolver está relacionado com a necessidade de garantir que uma classe só possa ser instanciada uma vez, e de fornecer um acesso fácil a essa instância única de qualquer ponto do programa. Isso é particularmente útil quando múltiplas partes do código necessitam de acessar a um recurso compartilhado e é importante que todas essas partes acessem a mesma instância desse recurso, mantendo a consistência dos dados e evitando possíveis conflitos devido a múltiplas instâncias.

Portanto, o padrão Singleton resolve dois problemas principais em design de software: a necessidade de controlar o número de instâncias de uma classe e a necessidade de fornecer um acesso global e consistente a essa única instância.

Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.
Make the default constructor private, to prevent other objects from using the new operator with the Singleton class.
Create a static creation method that acts as a constructor. Under the hood, this method calls the private constructor to create an object and saves it in a static field. All following calls to this method return the cached object.

The structure of the Singleton pattern is based on two main components, a private constructor (hidden from the client) that prevents instantiation of the class from outside, ensuring that it cannot be made using the "new" keyword, and a static method that returns the same instance of the class every time it is called. This instance is usually stored in a private static variable within the class.

Client and singleton are the main components.

The Singleton class declares the static method getInstance that returns the same instance of its own class.

The Singleton's constructor should be hidden from the client code. Calling the getInstance method should be the only way of getting the Singleton object.

The structural components of the Singleton pattern are the following:
Private Constructor: The Singleton class employs a private constructor to ensure that no other class can instantiate it.
Static Instance: A static variable holds the Singleton instance, ensuring that it's tied to the class itself rather than any object instance.
Public Static Method: This method provides access to the Singleton instance. It either returns the existing instance or creates one if none exists.

This pattern declares a static method to always return the same instance of its own class. Calling this method should be the only way of getting the desired object. It is important to refer that this method also created the object if it hasn't been initialized.

The structure of a Singleton class typically includes:
A private static variable of the class type to hold the single instance.
A private constructor to prevent instantiation from outside the class.
A public static method (usually named getInstance) that returns the single instance of the class. This method checks if the instance is already created; if not, it creates one and returns it.

A classe respetiva tem um getInstance() que irá retorna um new no caso da instancia nunca ter sido criada.

A estrutura do padrão Singleton envolve principalmente três componentes chave para a sua implementação e funcionamento:

Instância Única: O componente central do padrão Singleton é a própria instância única da classe. Este padrão garante que apenas uma instância da classe seja criada e exista durante o tempo de vida da aplicação. Essa instância é mantida privada dentro da classe e é acessada através de um método estático.

Construtor Privado: Para evitar que outras classes criem instâncias da classe Singleton, o construtor é feito privado. Isso impede a criação direta de objetos usando o operador new fora da classe. O acesso ao construtor privado é controlado internamente pela classe Singleton.

Método de Acesso Estático: Este método fornece um ponto de acesso global à instância única. Ele verifica se a instância já existe; se não, a instância é criada internamente. Se já existe, o método retorna a instância existente. Esse método estático é geralmente nomeado como getInstance() ou algo similar, proporcionando um meio fácil e global de acessar a instância única sem expor o construtor da classe.

The Singleton class declares the static method getInstance that returns the same instance of its own class.
The Singleton's constructor should be hidden from the client code. Calling the getInstance method should be the only way of getting the Singleton object.

The benefits of using this pattern are, for example, being sure that each class has a single instance, having global access to it and granting that the singleton object is only initialized when its requested for the first time, promoting consistency and a controlled access. A pratical example of this pattern's use is logging, because, typically, you want to ensure that all log messages are written to a single log file or stream, regardless of where in the code they are generated. Using the Singleton pattern for a logging system ensures that there is only one instance of the logger throughout the application, providing a centralized point for logging activities.

You can be sure that a class has only a single instance.
You gain a global access point to that instance.
The singleton object is initialized only when it's requested for the first time.

Os benefícios do uso do Singleton Pattern refletem-se na reduçao do coupling e na melhora da coesão através da separação do processo de criação e do resto do código, permitindo que cada classe tenha apenas um único propósito e responsabilidade. Alguns exemplos disso incluem: melhor resolução na colisão de nomes, lazy initialization, redução da memory footprint, etc.

It assures the class has only a single instance, it provides a global access point to that instance, and it is initialized only when it's requested for the first time.

- You can be sure that a class only has one instance
- You gain a global access point to that instance.
- The singleton object is initialized only when it's requested for the first time.

A common use case for the Singleton pattern is managing a database connection. This ensures that only one connection is active at any given time, preventing resource leaks and improving performance.
For managing application configurations, a Singleton can ensure that all parts of the application use the same configuration instance, avoiding inconsistencies.

Conseguimos garantir que a classe só tem um único objeto em simultâneo. Por exemplo, se tiveremos uma classe que define uma base de dados, apenas queremos criar uma nova base de dados se não existir nenhuma instancia da mesma, e caso exista utilizamos a que existe.

A utilização do padrão Singleton no desenvolvimento de software oferece várias vantagens, especialmente em cenários onde é crucial garantir que uma classe tenha apenas uma instância durante o ciclo de vida da aplicação

O padrão Singleton pode ser útil quando várias partes do código necessitam de acessar um recurso compartilhado. Ao garantir que apenas uma instância do objeto é criada, o Singleton ajuda a evitar conflitos de acesso e garante a consistência dos dados

Dado que criar múltiplas instâncias de objetos que realizam as mesmas operações ou contêm os mesmos dados pode ser um desperdício de recursos, como memória e processamento, este padrão ajuda a economizar recursos ao restringir a criação de objetos desnecessários.

O Singleton também facilita o acesso,pois fornece um ponto de acesso global à sua instância, o que pode simplificar o acesso ao objeto em diferentes partes da aplicação sem necessidade de passar o objeto explicitamente entre os componentes.

Exemplo: Uma aplicação com uma única conexão à base de dados compartilhada por toda a aplicação para evitar a abertura de múltiplas conexões.

Exemplo: Um sistema de configuração global acessível de qualquer parte da aplicação sem a necessidade de passar um objeto de configuração entre diferentes componentes.

Exemplo: Para integrar uma aplicação com serviços externos, um Singleton pode ser utilizado para gerenciar a conexão e as

imagine that you created an object, but after a while decided to create a new one. Instead of receiving a fresh object, you'll get the one you already created.

Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.

It solves two problems:

1. Ensure that a class has just a single instance, for example, to control access to some shared resource—for example, a database or a file.

2. Provide a global access point to that instance

O padrão Singleton é um padrão creacional que assegura que uma classe apenas tem uma instância e promove um ponto de acesso global para essa instância. Este padrão ajuda a evitar a poluição do namespace encapsulando a instância dentro da própria classe.

Este padrão permite garantir que uma classe tem uma única instância e fornecer um ponto de acesso global à mesma, com inicialização na primeira utilização (just-in-time ou lazy).
O problema deste é que é impossível de implementar com um construtor, uma vez que este cria uma nova instância do objeto por defeito. A criação de um ponto de acesso global também é um desafio, tal como a lazy initialization.
Assim, definir o construtor como privado, a par de um atributo estático para uma instância
de si próprio e um método também estático para lhe aceder.

É um creational pattern que garante que uma classe seja instanciada apenas uma vez. Os problemas que esta tenta resolver são a necessidade de garantir que uma classe apenas é instanciada uma única vez e que existe apenas um único ponto de acesso a essa classe.

O singleton pattern assegura que uma classe apenas tem uma instancia, dando um ponto de acesso global, resolvendo o problema de ter varias instancias da mesma classe. Assim quando se cria um objeto e resolve-se voltar a criar outro, em vez de ser criado um novo, é devolvido o anterior.

O padrão Singleton garante que apenas existe uma instância de uma classe e permite acesso global à mesma.

O padrão Singleton é uma abordagem de design de software que assegura a existência de apenas uma instância de uma determinada classe e oferece um ponto único de acesso global a essa instância. Ele é útil para resolver desafios como compartilhamento de recursos, controle de acesso, gerenciamento de estado e otimização de uso de memória.
Essencialmente, o padrão Singleton simplifica a interação e o controle de recursos centralizados dentro de um sistema.

Um singleton é uma classe da qual só pode existir uma instância, por exemplo, se tivermos um site de compras, o singleton pattern ajuda a ter a certeza que só vai ser instanciado um carrinho de compras.

The singleton method ensures there is only one instance of each class in order to control access to some resources.

The singleton design pattern dictates that only a single instance of a may exist in the codebase. This helps in situations where a signle class may access a certain resource, such as a database for example.

The main element of the Singleton Pattern is:
- the Singleton class declares the static method getInstance that returns the same instance of its own class.

Thus, Singleton's constructor should be hidden from the client code. Calling the getInstance method should be the only way of getting the Singleton object.

Singleton em si, tem um construtor privado que nunca é utilizado e, também, tem um método getInstance() que só é utilizado para aceder ao ponto de entrada para a própria classe.

A classe Singleton declara o método estático getInstance que retorna a mesma instância de sua própria classe.
O construtor do Singleton deve estar oculto do código do cliente. Chamar o método getInstance deve ser a única maneira de obter o objeto Singleton.

A classe que será instanciada deve ter um construtor privado.
O Singleton deve ter um método de criação estático que atua como construtor. Este método chama o construtor da classe a ser instanciada e guarda o objeto num campo estático.

O singleton patter tem um objeto da classe Singleton que tem um metodo estatico getInstance que retorna esse mesmo objeto, assegurando que é igual para todos.

O singleton consiste em criar uma variável privada que contém a instância da classe e um método estático (normalmente getInstance()) que cria esta instância, caso já exista apenas retorna a mesma, sendo este a única forma de aceder ao objeto. Para que não possam ser criadas mais instâncias o construtor tem de ser privado

A estrutura do padrão Singleton envolve geralmente os seguintes componentes principais:

Classe Singleton: Esta é a classe para a qual se deseja apenas uma instância. Geralmente contém alguma funcionalidade útil que precisa ser acessada globalmente. A classe normalmente possui um construtor privado para evitar a instanciação direta e um método estático para fornecer acesso à única instância.

Variável de Instância: A classe contém uma variável estática que mantém a única instância da classe. Essa variável geralmente é nomeada como "instância" e geralmente é privada para garantir que só possa ser acessada através do método estático.

Método Estático getInstance: Este método é responsável por retornar a única instância da classe. Geralmente verifica se uma instância da classe já foi criada. Se uma instância existir, retorna essa instância; caso contrário, cria uma nova
instância e a retorna

getInstance para retornar o próprio objeto, caso este já exista, ou criar um novo.

The constructor is private and a static creation method is implemented.

In the singleton design pattern there must exist a private static variable "instance" that controls wether or not a singleton instance was created. There must also be a method that checks this assertion.

Benefits:

- A class has only a single instance.

- You gain a global access point to that instance.

- The singleton object is initialized only when it's requested for the first time.

Gestão eficiente de recursos (evita o consumo desnecessário de recursos);
Consistência de dados (como só há uma instância ela facilita a manutenção da consistência dos dados);
Facilidade de acesso (reduz a necessidade de passar objetos de um componente para outro);

O padrão Singleton oferece vários benefícios, incluindo:
Controlo de acesso único: Garante que uma classe tenha apenas uma instância, útil para controlar o acesso a recursos partilhados como uma base de dados ou arquivo de configuração, simplificando o acesso a serviços comuns.
Economia de recursos: Evita a criação desnecessária de objetos, o que pode ser importante para recursos com alto custo de inicialização ou limitados.

Exemplos práticos:
Gestor de configurações: Um objeto Singleton que carrega as configurações do sistema no início e as disponibiliza globalmente.
Conexão à base de dados: Utilizar um Singleton para gerir a conexão com a base de dados garante que não existam múltiplas instâncias de conexões abertas, o que poderia sobrecarregar o servidor de dados.

São ocultados detalhes relativos à construção do objeto.
É garantido que é apenas instanciado uma única vez.
Existe um acesso global ao Singleton e ao objeto que nele é instanciado.
O objeto só é criado quando é utilizado.
Múltiplas threads podem utilizar o objeto de forma segura, evitando conflitos.

Assegura-mos que a classe apenas tem uma instancia, ganhamos um ponto de acesso global a essa instancia, e assim o objeto apenas é iniciado quando é chamado pela primeira vez.

- É possivel garantir que uma classe tem apenas uma instância;
- Obtém se um ponto de acesso global a esse objeto;
- O objeto apenas é inicializado após o primeiro pedido de acesso à instância.

Os benefícios do uso do padrão Factory no desenvolvimento de software são a abstração da criação de objetos, porque permite encapsular a lógica de criação de objetos em uma classe separada. O desacoplamento porque ao utilizar o padrão Factory, o código que requer objetos não precisa conhecer as classes concretas que estão sendo instanciadas. Por fim a facilidade de manutenção e extensibilidade, porque ao concentrar a criação de objetos em uma única classe, torna-se mais simples e seguro fazer modificações ou adicionar novos tipos de objetos.

Exemplo prático:
No contexto de um aplicativo de desenho, podemos usar uma fábrica de objetos gráficos para criar uma variedade de formas geométricas, como círculos, retângulos, dependendo das preferências do usuário. Isso simplifica a extensão do sistema de desenho, permitindo a adição de novos tipos de formas geométricas sem a necessidade de modificar diretamente o código existente.

Tem-se a certeza que a classe tem apenas uma instância;
Ganha-se acesso global a essa instância;
Só pode ser inicializado a primeira vez que é chamado;

It encapsulates object creation logic in a separate class, making the code more maintainable and flexible.

The singleton pattern ensures that a class has a single instantiation (which can be useful when some resources require access to them one at a time). This pattern is thread safe, which means that when multiple threads are involved it is assured that resources that can't be accessed by multiple elements at once aren't.

The Singleton pattern ensures that only one instance of a class exists throughout an application.
It provides global access to this single instance from any part of the codebase.
The singleton instance acts as a central point of control or coordination.
Use Cases:
Factories and Builders: Singleton ensures consistent behavior across the application.
Program State Management: Objects holding program state (e.g., configuration settings, user preferences) benefit from the Singleton pattern.
Resource Managers: Efficiently manage shared resources (e.g., database connections, thread pools).
In summary, the Singleton pattern promotes efficiency, consistency, and global accessibility in software design.

The Singleton pattern employs a static member within the class. This static member ensures that memory is allocated only once, preserving the single instance of the Singleton class.
The Singleton pattern incorporates a private constructor. This constructor serves as a barricade against external attempts to create instances of the Singleton class.
A crucial aspect of the Singleton pattern is the presence of a static factory method, when someone requests an instance, this method either creates a new instance (if none exists) or returns the existing instance to the caller.

Single Instance:
The Singleton Pattern guarantees that a class has only one instance throughout the application.
This is useful in scenarios where multiple instances must be avoided, such as when managing shared resources or configurations1.
Global Access:
Because a Singleton instance is globally accessible, you can access its methods and properties from anywhere in the application.
This eliminates the need to pass object instances or manage complex dependencies1.
Lazy Initialization:
The Singleton pattern supports lazy initialization, meaning that the instance is created only when it is accessed for the first time.
This can improve performance by deferring instantiation until necessary.

Some practical examples of Singleton pattern usage:

Caches:
Caches often benefit from Singleton behavior. A single cache instance can store frequently accessed data, improving performance by avoiding redundant computations or database queries.
Example: An in-memory cache for storing frequently accessed user profiles or product data.
Thread Pools:
Managing thread pools efficiently requires a single point of control.
Singleton ensures that only one thread pool instance exists, preventing resource wastage.
Example: A web server handling multiple client requests, where each request is processed by a thread from the singleton thread pool.
Configuration Managers:
Singleton can manage application-wide configuration settings,

The Singleton pattern ensures only one instance of a given class exists at any given time and that it's accessible by other classes globally. This way, other classes have shared access to variables, attributes and resources (such as, for example, a database or any other part of shared memory).

The main components in the singleton pattern are, a private static attribute on the class where the unique instance is stored, a private constructor to forbid instantiation of the class and a public static method to get the unique instance stored in the static attribute.

The benefits of using a singleton are that it removes the need to control multiple instances of a given class making it easier to manage. Whenever you have global variables that are shared across many instances of a class, that is dangerous since any part of the code can accidentally (or not) change those variables breaking the rest of the program.
The way to solve it is to have a singleton since it can make all those global variables private and unchanging.

The Singleton pattern insures that a class has only one instance and provides a global access point to it

Static member
Private constructor
Static Factory method

The class contains only a single instance, you get a global access point to that instance and the singleton object is initialized only when it is requested for the first time.

Singleton is a creational design pattern that ensures a class has only one instance, by providing a global access point to it. It is implemented by the default constructor private, to prevent other objects from using the new operator with the Singleton class. It solves the problem of using unsafe global variables since any code can potentially overwrite the contents of those variables

Has Client and Singleton itself who is hidden from the client code. Singleton has a private constructor to no other object use that constructor. Uses a getInstance() method to get Client instance.

- Ensures that a class has only a single instance
- There is a global access point to that instance.
- The singleton object is initialized only when it's requested for the first time

O padrão singleton é um padrão que restringe a instanciação de uma classe a uma única instância, ele resolve o problema de garantir que apenas uma instância de uma classe exista em toda a aplicação e fornece um ponto de acesso global a essa instância.

O padrao singleton esta dividido em quatro componentes que são Classe Singleton, Instância Privada e Estática, Método de Acesso Estático, Funcionalidades Adicionais Opcionais, e a componente principal e a classe single que e projetada para ter apenas uma instancia em toda aplicaçao.

O beneficio de usar factory Pattery e que facilita a memoria e simplifica muito mais o programa como na aula passada que tínhamos de fazer o programa de POO antigo e usar o método factory pattern.

The Singleton pattern is a design pattern that restricts the instantiation of a class to a single instance. This is useful when exactly one object is needed to coordinate actions across the system. It allows a way to ensure that a class has only one instance, and to provide a global point of access to it. This can be useful for things like logging, driver objects, caching, thread pools, and database connections.

All constructors of the class are private. Has a static instance of itself. Has a static method that returns a reference to the instance.

The benefit of the singleton pattern is that it can help manage shared resources or state throughout the application.

For example, by using a singleton to store configuration settings, cache data, or log messages.

Because of the singleton, we can avoid creating multiple objects that consume memory and resources and ensure consistency and coordination between them.

Os beneficos do Padrão Factory são exemplialmete Desacoplamento em que o padrão desacopla o código que cria objetos do código que usa os objetos, Flexibilidade que permite a introdução de novas classes concretas sem alterar o código, Reusabilidade que promove a reutilização do reutilização de código ao concentrar a criação de objetos e o Controle sobre a criação, isto é, o padrão decide dicnamicamente quais são as classes que deve instanciar com base em critérios específicos.

Um exemplo é um sistema de notifiações:

- Este notifica diferentes tipos de notificações exmeplo MSM, mail entre outros dependendo das preferências do usuário ou de condições específicas do sistema.

Onde tem ima interface que "Notificacao" que tens os metedos necessarios para cada notificação como por exemplo metedo enviar.

As classe concretas implementamos classes concretas para cada tipo de notificação exmeplo EmailNotificacao

E o Factory e por fim criamos uma classe NotificacaoFactory com um método criarNotificacao que recebe um parâmetro indicando o tipo de notificação desejado. Na qual o metedo ira instanciar com base nesse parâmetro e retornaria uma instância da interface Notificacao.

The Singleton pattern is a creational design pattern that ensures a class has only one instance, providing a global point of access to it. This pattern is useful when you need a single object to coordinate actions across the system.
The Singleton pattern solves the problem of controlling the number of instances of a particular class. This is useful when having multiple instances can lead to negative consequences, such as performance issues, thread safety problems, or conflicts in a shared resource scenario. By enforcing a single instance, the Singleton pattern helps maintain a consistent state and avoids these issues.

When it comes toi the Singleton Pattern there is only one main component, the Singleton Class, this class return the static method getInstance(), that returns the same instance of it's own, this is the only way to get the Singleton Objet. The creation of a new object only occurs when no instance have been created.

Vantaguens: facilmente adaptar-se para enviar diferentes tipos de notificações em diferentes contextos sem depender diretamente das classes concretas.Assim não alterando o restante do sistema.

Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.

It ensures that a class has just a single instance and provides a global access point to that instance.

A private static field in the class for storing the singleton instance.
A public static creation method for getting the singleton instance.
A “lazy initialization” inside the static method.
The constructor of the class is private.

A class has only a single instance.
Global access point to that instance.
The singleton object is initialized only when it’s requested for the first time.

É um padrão de design que garante que a classe tenha apenas uma instancia enquanto permite um ponto de acesso global para essa instancia.
Resolve o problema do controle de acesso a objetos.

A classe singleton declara o método estático getinstance() que retorna a mesma instancia da própria classe. O construtor do Singleton deve estar escondido do código do cliente, e chamando o método getisntance() é a unica maneira de aceder ao objeto.

- O objeto é apenas inicializado quando pedido pela primeira vez.
- Você ganha um ponto de acesso global a essa instancia.
- Teremos certeza que a class tem apenas uma instancia.

O governo é um bom exemplo de Singleton Pattern. Um país pode apenas ter um governo oficial. Independente de quem está no governo , o título, "O governo de X"é um ponto de acesso global que identifica o grupo de pessoas no poder.

Singleton é um padrão que garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto. Os problemas que pretende resolver são: certificar que uma classe tem apenas uma instância; E fornecer uym ponto de acesso global para essa instância.

Declarar um método estático getInstance que retorna a mesma instância de sua própria classe.

Ter a certeza que a classe tem apenas uma instância.

The singleton pattern ensures a class only has one instance, and provides a global point of access to it.
Having only one instance provides a way to control access to some shared resource
The Singleton pattern lets you access some object from anywhere in the program. However, it also protects that instance from being overwritten by other code.

Add a private static field to the class for storing the singleton instance.
Declare a public static creation method for getting the singleton instance.
Implement “lazy initialization” inside the static method. It should create a new object on its first call and put it into the static field. The method should always return that instance on all subsequent calls.
Make the constructor of the class private. The static method of the class will still be able to call the constructor, but not the other objects.
Go over the client code and replace all direct calls to the singleton’s constructor with calls to its static creation method.

You can be sure that a class has only a single instance.
You gain a global access point to that instance.
The singleton object is initialized only when it’s requested for the first time.

O padrão Singleton é um padrão criacional que garante que uma classe tenha apenas uma instância e forneça um ponto de acesso global a ela. Responde ao problema de quando é necessário que uma classe tenha apenas uma instância disponível para todos os clientes.

As componentes de uma estrutura Singleton sao: Construtor Privado, para impedir que outras classes instancie a classe Singleton; Variável Estática Privada, para manter a única instância da classe Singleton; Método Público Estático, getInstance(), que retorna a única instância da classe Singleton.

Beneficios: Desacoplamento; Reutilização de Código; Simplificação; Flexibilidade; Controle sobre a Criação de Objetos;
Exemplo prático: Processamento de Pagamentos.

O padrão Singleton tem como objetivo garantir que uma classe tenha apenas uma única instância de um objeto e fornecer um ponto de acesso global a essa instância.

Este padrao resolve dois problemas:
- Certifica-se de que uma classe tenha apenas uma instância;
- Fornece um ponto de acesso global para essa instância.

Os principais componentes envolvidos no padrão Singleton são os seguintes:

1 - Classe Singleton: Esta é a classe para a qual apenas uma única instância precisa ser criada. A classe é projetada de forma que seja responsável por criar sua própria instância e fornecer um ponto global de acesso a essa instância. A classe normalmente define um método ou propriedade estática para acessar a instância singleton.

2 - Construtor privado: a classe singleton normalmente possui um construtor privado para evitar a instanciação direta da classe de fora. Ao tornar o construtor privado, outras classes não podem criar novas instâncias da classe singleton.

3 - Instância estática: a classe singleton contém uma variável de membro estático que contém a única instância da classe. Essa variável geralmente é privada e estática para garantir que apenas uma instância seja criada e acessada em todo o aplicativo.

4 - Método de acesso: a classe singleton fornece um método ou propriedade estática que permite que outras partes do programa acessem a instância singleton. Este método é responsável por criar a instância caso ela ainda não exista e retornar a instância existente caso contrário

- Evitamos um acoplamento forte entre o criador e os produtos de concreto.
- Princípio da Responsabilidade Única - podemos mover o código de criação do produto para um local no programa, facilitando, assim, o suporte do código.
- Princípio Aberto/Fechado - podemos introduzir novos tipos de produtos no programa sem desviar o código do cliente existente.

It is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.
They allow for developers to use pre-defined and proven templates to structure a logical solution to a type of problem.
It ensures that a class has just a single instance and provides a global access point to that instance. It also helps to keep resource utilization under control.

It prevents other objects from using the new operator, by making the default constructor private and it creats a static creation method that acts as a constructor.

The benefits of using the Singleton pattern are that you can be sure that a class has only a single instance, you gain a global access point to that instance and the object is only initialized when it’s requested for the first time

Este padrão assegura a que a classe tenha apenas uma instância, e a mesma tem um ponto de acesso global

Os componentes principais deste padrão são um construtor privado, prevenindo o acesso a outras classes. Outra peça importante é a criação de um método estático que funcionará como construtor ou para apanhar a instância do objeto já criado

Alguns usos do padrão Singleton são: Gerir acesso a um objeto "Database" e assegura que é apenas feita uma conexão em todo o sistema a este. O Singleton pode também ser utilizado para gerir o estado de um jogo tendo a informação de resultados, informação do jogador, etc. pondo toda essa informação de fácil acesso a partir da classe.

O padrão Singleton serve para assegurar que uma classe tem apenas uma instancia. Quando um cliente quer criar mais uma instancia do objeto, vai receber o mesmo objeto já criado. Neste padrão devemos dar vários pontos de acesso a essa instancia.

A estrutura principal do padrão é ter dentro de uma classe um construtor private e um método static que devolve uma instancia dessa classe. No método static devemos ter alguma forma de devolver uma nova instancia, se ainda não existe, ou a instancia que já existe, por exemplo, ter um atributo na classe que guarde um null inicialmente e quando o método é chamado, guarda (e devolve) a instancia do objeto. Quando o método é chamado outra vez, terá um "if (instancia != null) return instancia;" que irá devolver a instancia existente.

Dois dos benificios são: ter a garantia de que só existe uma instancia dessa classe; existe um acesso global a essa instancia.
Um exemplo prático é uma base de dados ou outras formas de memória partilhada.

It is a pattern that ensures that a class has only one instance, and provides a global access point to that instance.

It's main components are a "Private Constructor" (ensures no external code can create an instance with "new"; constructor should be hidden from the client code) and a "Static getInstance()" method (way of getting an instance of the Singleton class).

The benefits are promoting a cleaner and more flexible code and the advantages would be Loose Coupling, Improved Maintainability, Flexibility and Extensibility and Encapsulating Object Creation Complexity.
A practical example would be a Shape Creator, a drawing app that allows the user to create different shapes where the code would request the desired shape from the factory.

Tem benefícios como garantir que uma classe tenha uma única instância, útil para coordenar ações por toda a aplicação, fornece um ponto de acesso global à instância, o que pode substituir variáveis globais, a instância é criada somente quando necessário, não no carregamento da classe.

O padrão Singleton ajuda a garantir que apenas uma instância de uma classe seja criada e fornece um ponto de acesso global a essa instância.
Este padrão restringe o acesso ao construtor da classe, impedindo que outros objetos usem o operador new para criar várias instâncias da classe permitindo que exista apenas uma instância da classe.

O padrão Singleton tem um construtor privado (que esconde o construtor), uma instância privada, e um método publico estático para aceder a instância criada,

Exemplo pratico:
O governo é um padrão Singleton. Um país pode ter apenas um governo oficial. Independentemente das identidades pessoais dos indivíduos que formam governos, o título, "O Governo de X", é um ponto de acesso global que identifica o grupo de pessoas no command.

It reduces coupling and improves cohesion, promotes reusability and extensibility by providing the same interface or abstract class for different implementations or variations of the object and increases flexibility and scalability by making it possible to change the behavior or appearance of the object by changing the parameters or input of the factory method or class. Example: A game has different classes. Everytime a new class is added without the use of the Factory pattern, the main class will have to be changed. If the Factory pattern is being used, it is only needed to create a new class without modifying the code.

The Singleton Pattern is a design paradigm that restricts the instantiation of a class to a single object, and provides a global point of access to it.

Static member, private constructor, static factory method.

É um padrão de creação que consiste um uma classe cujo constructor é privado, impedindo de ser instanciado, mas possui uma variável privada "static" que guarda a instancia desta classe (inicialmente a null), e um método público "static" que instancia um objeto se ainda não tiver sido instanciado ou devolve a instância desta classe.
Com esta implementação apenas uma instância desta classe pode ser criada e qualquer futura chamada ou método que substitui o construtor devolve a instância criada na primeira vez.
Logo é útil para garantir apenas uma instância de uma classe e uma forma fácil dela ser acessada globalmente.

Os principais components são: uma classe "singleton", que possui uma variável privada static "instance" e um método público static "getInstance".

É usado quando uma classe deve ter apenas uma instância desta. Como exemplo uma classe Logger, pois é essencial que haja uma instância de um Logger para guardar as informações de login e gerencia-las

A implementação do padrão Singleton envolve:
- um construtor privado, que previne a instanciação da classe fora da sua implementação;
- uma variável private static _instance que pode conter dois valores distintos: null, se a classe ainda não foi instanciada; ou a própria instância da classe caso esta já se encontre instanciada.

Por vezes, pode ser necessário garantir que uma classe é instanciada apenas uma vez de modo a que exista um único ponto de controlo. Isto pode ser necessário, por exemplo, com classes que gerem o acesso a uma base de dado. Este problema é resolvido recorrendo ao padrão de desenvolvimento Singleton.

Pode-se aceder ao valor da variável _instance através de um getter GetInstance() que, quando chamado, verifica se a variável comtém null ou não e devolve a mesma instância sempre que chamado. Caso contenha null, instacia a classe e atribui-a a ela própria, devolvendo-a.

Reduz a ligação e melhora a coesão ao desassociar o processo de criação do resto do código. É útil quando a criação de objetos é complexa e requer uma lógica específica. Ao encapsular a criação de objetos em fábricas, facilita-se a manutenção do código, uma vez que as alterações na criação de objetos estão centralizadas.

Os benefícios são:
- Temos a certeza que uma classe tem apenas uma instância
- Obtém-se um ponto de acesso global para a instância criada
- O objeto Singleton é iniciado apenas quando é preciso.

Exemplo Prático:
- O governo é um ótimo exemplo do padrão Singleton, um país pode ter apenas um só governo oficial, e para além disso dizer "O governo de X país" é um ponto de acesso global para identificar as pessoas à frente deste, independentemente da identificação pessoal dos indivíduos que formam o governo.

O Singleton Pattern serve para restringir uma classe a ter apenas uma instância, o que pode ser útil para aceder a uma base de dados ou um ficheiro e protege a classe de ser sobre escrita por outras.

A estrutura envolve um cliente e um Singleton:
- O cliente -> É responsável por solicitar uma instância da classe Singleton
- Singleton -> Implementa o padrão.

Garante que uma classe só tem uma única instância;
Tem um ponto de acesso global a essa instância;
O objeto é instânciado apenas da primeira vez que é requisitado através do getInstance().
Uma Universidade tem apenas uma Administração apesar de ser composta por um conjunto de indivíduos. "A Administração X", é o ponto de acesso global que identifica as pessoas que fazem parte e são responsáveis.

O que é: É um padrão criacional que garante que só existe uma instância de um objeto no ciclo de vida do programa e providencia um ponto de acesso global à instância, todavía viola o princípio da Single Responsability.

A classe Singleton tem um método estático (getInstance()) que devolve uma instância única da própria classe. O construtor deve estar escondido do client. Pelo que o método do getInstance deve ser a única forma de obter a instância desta classe.

Tem como obejtivo de certificar uma classe tenha apenas uma instância e forneça uma instância global em que de aceda a ele.
Muitas vezes utilizado quando se precisa que um objeto coordene açoes em todo o sistema.
Alguns exemplos de componentes envolvidas sao: Um construtor privado, um metodo estático e uma variável de instancia privada estática.

Em um sistema que precisa de um gerenciador de configurações globais, o Singleton pode ser usado para garantir que apenas uma instância desse gerenciador exista e forneça acesso global às configurações.

O Singleton permite garantir que uma classe tenha apenas uma instância, enquanto prove um ponto de acesso global para essa instância.

O padrão Singleton é um padrão de design que permite garantir que uma classe tenha apenas uma instância, ao mesmo tempo que fornece um ponto de acesso global para essa instância. Este padrão resolve dois problemas, sendo estes:

1- Verificar se a classe só tem uma única instância
2- Garante um acesso geral a essa instância

O padrão Singleton é um padrão de design criacional que garante que uma classe tenha apenas uma instância e fornece um ponto de acesso global a essa instância. Sua estrutura geralmente envolve os seguintes componentes:

1-Classe Singleton: Esta é a classe para a qual apenas uma instância deve existir ao longo do programa. Geralmente, ela contém métodos úteis e dados relevantes para seu propósito.

2-Construtor Privado: A classe Singleton geralmente tem um construtor que é definido como privado, impedindo que outras classes instanciem objetos dessa classe diretamente.

3-Membro de Instância Estático: Dentro da classe Singleton, há uma variável membro estática que mantém a única instância da classe. Esta variável é frequentemente nomeada algo como instancia.

4-Método Estático para Acesso: Existe um método estático, frequentemente chamado de getInstance(), que fornece o ponto de acesso global à única instância da classe. Este método é responsável por criar a instância se ela ainda não existir, ou retornar a instância existente se já existir.

Benefícios do Padrão Factory:

1-Encapsulação de lógica de criação: Isola a criação de objetos do código do cliente.

2-Flexibilidade e extensibilidade: Facilita a adição de novos tipos de objetos sem modificar o código do cliente.

3-Desacoplamento: Permite que o código do cliente interaja com uma interface de fábrica em vez de classes concretas, promovendo um código mais limpo e manutenível.

4-Configuração centralizada: Fornece um local único para gerenciar a criação de objetos.

5-Suporte para injeção de dependência: Pode ser usado em conjunto com frameworks de injeção de dependência para gerenciar a criação e injeção de objetos.

Exemplos Práticos:

1-Frameworks de GUI: Criação de componentes de interface do usuário.

2-Acesso a Banco de Dados: Criação de objetos de acesso a dados com base em configurações.

3-Bibliotecas de Log: Criação de diferentes tipos de registradores de log.

4-Desenvolvimento de Jogos: Criação de objetos de jogo como personagens e inimigos.

5-Contêineres de Injeção de Dependência: Gerenciamento de