**Padrão Abstract Factory:**

Fábrica Abstrata Pode ser uma classe abstrata ou uma interface, mas a classe abstrata é utilizada com maior frequência. Seu objetivo é declarar métodos de criação de objetos do tipo

ProdutoAbstrato, que são implementados por uma classe do tipo FabricaConcreta, que estende ou implementa a FabricaAbstrata. Produto Abstrato Pode ser uma classe abstrata ou uma interface, mas a classe abstrata é utilizada com maior frequência. Produto abstrato declara os métodos que são implementados por classes do tipo ProdutoConcreto.

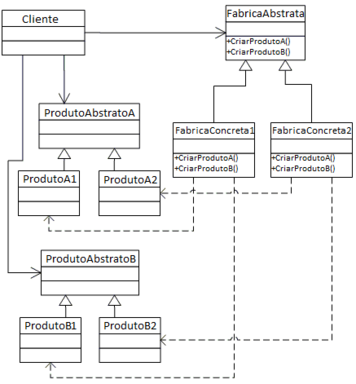
FabricaConcreta cria internamente um objeto do tipo ProdutoConcreto, mas esse objeto é retornado como um ProdutoAbstrato. O Abstract Factory não sabe qual ProdutoConcreto está sendo criado, mas sabe quais métodos do produto ele pode utilizar. Fábrica Concreta Estende ou implementa a FabricaAbstrata. O objetivo dessa classe é implementar os métodos declarados em FabricaAbstrata, criando um objeto do tipo ProdutoConcreto e retornando-o como um ProdutoAbstrato. Isso é polimorfismo. É comum existir mais de uma classe do tipo ProdutoConcreto assim como ocorre com FabricaConcreta. A quantidade de classes do tipo FabricaConcreta está diretamente ligada com a quantidade de classes do tipo ProdutoConcreto.

Produto Concreto(ProdutoA1, ProdutoA2, etc..) Estende ou implementa a classe ProdutoAbstrato. Nessa classe são implementados os métodos declarados em ProdutoAbstrato. Essa é a classe que faz uma instância concreta ser criada. Para cada FabricaConcreta, há pelo menos um ProdutoConcreto.

Utilização: A fábrica determina o tipo concreto do objeto a ser criado, e é nela que o objeto é realmente criado. No entanto, a fábrica só retorna um ponteiro abstrato para o objeto concreto criado.

O código do cliente não tem conhecimento algum do tipo concreto. Objetos concretos são, de fato criados pela fábrica, mas o código do cliente acessa tais objetos só através da sua interface abstrata.

A adição de novos tipos concretos é feita modificando o código do cliente para usar uma fábrica diferente, uma modificação que é tipicamente uma linha em um arquivo. A nova fábrica, em seguida, cria objetos de um tipo de concreto diferente, mas ainda retorna um ponteiro do mesmo tipo abstrato como antes. Isto é significativamente mais fácil do que modificar o código de cliente para instanciar um novo tipo. Se todos os objetos de fábrica são armazenados globalmente em um objeto Singleton, e todo o código do cliente passa pelo Singleton para acessar a fábrica adequada para a criação do objeto, então alterar as fábricas se torna tão fácil como mudar o objeto Singleton.



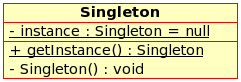
**Padrão Singleton**

Singleton é um padrão de projeto de software. Esse padrão garante a existência de apenas uma instância de uma classe preservando o ponto de acesso global do objeto.

Nota linguística: O termo vem do conjunto de significados em inglês. (objeto matemático) tendo apenas um elemento.

Alguns projetos exigem que certas classes tenham apenas uma instância. Por exemplo, em aplicativos que requerem uma infraestrutura de registro. Você pode usar classes no formato singleton. Desta forma, apenas um objeto é responsável pelo login em todas as aplicações acessíveis através da classe singleton.

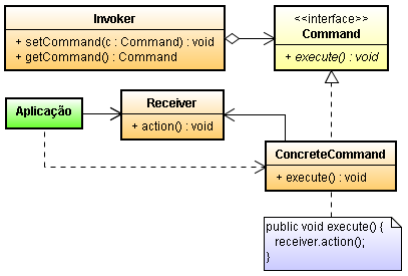
O padrão singleton é considerado por muitos desenvolvedores como um antipadrão, portanto, tenha cuidado ao usá-lo. Quando você precisa de apenas uma instância da classe, por exemplo, a conexão com o banco de dados, é garantido que nessa execução a classe será instanciada apenas uma vez.



**Padrão Command**

Command é um dos 11 padrões comportamentais dentre os 23 padrões de projeto de software do GOF. Na programação orientada a objeto, o command é um padrão no qual um objeto é usado para encapsular toda informação necessária para executar uma ação ou acionar um evento em um momento posterior**.**

**Objetivo**: O Padrão Command tem como definição encapsular uma solicitação como um objeto, o que lhe permite parametrizar outros objetos com diferentes solicitações, enfileirar ou registrar solicitações e implementar recursos de cancelamento de operações. Isso inclui informações como o nome do método, o objeto que o método pertence e os valores dos parâmetros do método.



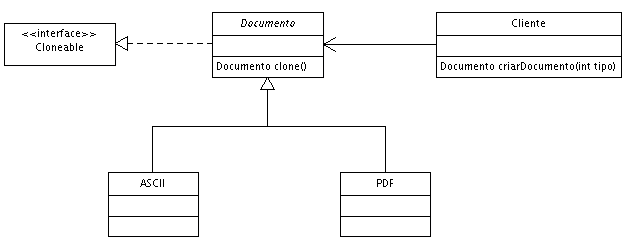
**Aplicabilidade:** Os objetos de comando facilitam a criação de componentes gerais que precisam delegar, sequenciar ou executar chamadas de método no momento de sua escolha. A chave para esse padrão é uma classe Command abstrata, que declara uma interface para executar operações. O uso de um objeto invocador permite que a execução de comandos seja realizada de forma conveniente, bem como a implementação de diferentes modos de comandos.

**Utilizar quando:**

* Parametrizar objetos por uma ação a ser executada. Você pode expressar tal parametrização numa linguagem procedural através de uma função callback, ou seja, uma função que é registrada em algum lugar para ser chamada em um momento mais adiante. Os Commands são uma substituição orientada a objetos para callbacks;
* Especificar, enfileirar e executar solicitações em tempos diferentes. Um objeto Command pode ter um tempo de vida independente da solicitação original. Se o receptor de uma solicitação pode ser representado de uma maneira independente do espaço de endereçamento, então você pode transferir um objeto Command para a solicitação para um processo diferente e lá atender a solicitação;
* Suportar desfazer operações. A operação Execute, de Command, pode armazenar estados para reverter seus efeitos no próprio comando. A interface do Command deve ter acrescentada uma operação Unexecute, que o reverte.efeitos de uma chamada anterior de Execute. Os comandos executados são armazenados em uma lista histórica. O nível ilimitado de desfazer e refazer operações é obtido percorrendo esta lista para trás e para frente, chamando operações Unexecute e Execute, respectivamente.

**Padrão Prototype**

**Estrutura:** O padrão Prototype exige a implementação de uma operação de clonagem em cada uma das classes concretas do protótipo. Esta tarefa pode ser inconveniente, no caso do reaproveitamento de classes preexistentes que não possuem tal operação, ou mesmo complexa, se for considerada a possibilidade de existirem referências circulares nos atributos de um objeto (um objeto possui um atributo que referencia um objeto que, por sua vez, referencia o objeto original).

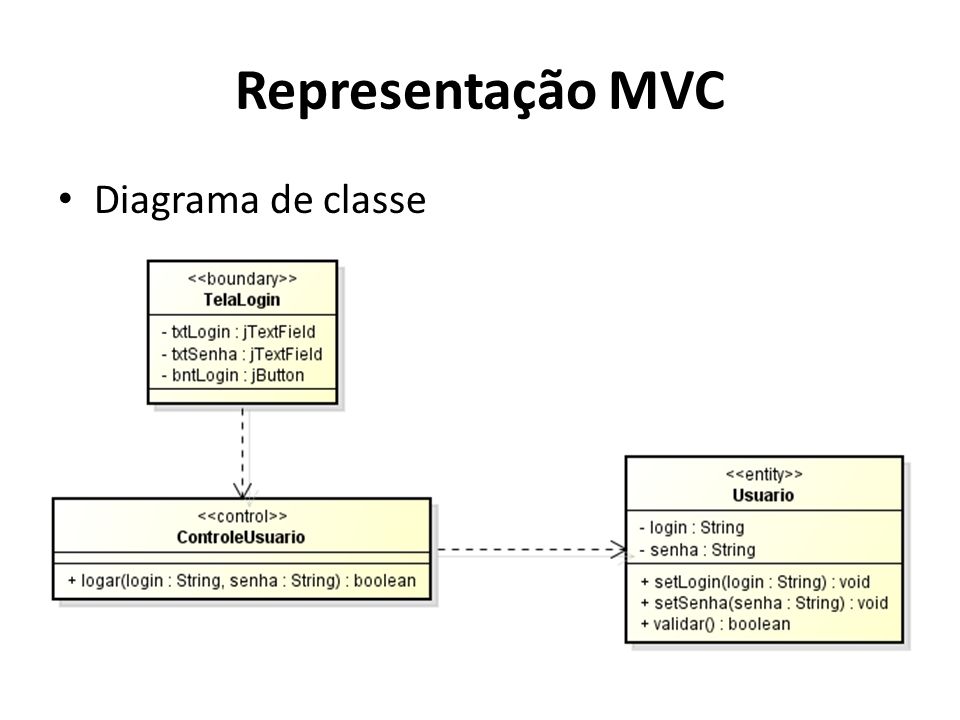


Utilização: O padrão Prototype é aplicado quando há a necessidade de literalmente clonar um objeto. Esse padrão pode ser utilizado em sistemas que precisam ser independentes da forma como seus componentes são criados, compostos e representados. Pode ser útil em sistemas com as seguintes características: sistemas que utilizam classes definidas em tempo de execução, sistemas que utilizam o padrão Abstract Factory para criação de objetos.

**Padrão MVC**

**Explicando:**

1. Primeiramente o controlador (Controller), que interpreta as entradas do mouse ou do teclado enviadas pelo usuário e mapeia essas ações do usuário em comandos que são enviados para o modelo (Model) e/ou para a janela de visualização (View) para efetuar a alteração apropriada;
2. Por sua vez, o modelo (Model) gerencia um ou mais elementos de dados, responde a perguntas sobre o seu estado e responde a instruções para mudar de estado. O modelo sabe o que o aplicativo quer fazer e é a principal estrutura computacional da arquitetura, pois é ele quem modela o problema a ser resolvido;
3. Por fim, a visão (View) gerencia a área retangular do display e é responsável por apresentar as informações para o usuário através de uma combinação de gráficos e textos. A visão não sabe nada sobre o que a aplicação está atualmente fazendo, pois tudo que ela realmente faz é receber instruções do controle e informações do modelo e então exibi-las. A visão também se comunica de volta com o modelo e com o controlador para reportar o seu estado.



Por que usar MVC?

Muitos bootcamps de programação ensinam esse padrão de arquitetura de software por alguns benefícios que justificam o MVC como uma das mais escolhidas no processo de desenvolvimento. Esses benefícios são: 

* **Segurança**: O controller funciona como uma espécie de filtro capaz de impedir que qualquer dado incorreto chegue até a camada modelo.
* **Organização**: Esse método de programação permite que um novo desenvolvedor tenha muito mais facilidade em entender o que foi construído, assim como os erros se tornam mais fácil de serem encontrados e corrigidos.
* **Eficiência**: Como a arquitetura de software é dividida em 3 componentes , sua aplicação fica muito mais leve, permitindo que vários desenvolvedores trabalhem no projeto de forma independente.
* **Tempo**: Com a dinâmica facilitada pela colaboração entre os profissionais de desenvolvimento, o projeto pode ser concluído com muito mais rapidez, tornando o projeto escalável.
* **Transformação**: As mudanças que forem necessárias também são mais fluidas, já que não será essencial trabalhar nas regras de negócio e correção de bugs.