**ABSTRAÇÃO**

Antes de falar sobre o padrão Factory Method, vamos começar com alguns [conceitos importantes](https://www.makerzine.com.br/educacao/pensamento-computacional-e-seus-4-pilares/) da computação, como a **abstração**.

Quando dizemos que a classe **Pessoa**tem um atributo **nome**, isso é uma abstração. Dessa forma, abstrair algo é isolar algumas características. Por exemplo, para dirigir um carro, não precisamos saber como o motor dele foi fabricado nem quais componentes ele possui, só precisamos ter a Carteira Nacional de Habilitação (CNH) e saber dirigir. Assim, todo o resto podemos abstrair.

**DECOMPOSIÇÃO**

Outro conceito muito importante na computação é a **decomposição**. Com ela, encontrar soluções se torna mais fácil quando dividimos algo complexo em partes simplificadas.

Sendo assim, ela torna mais fácil a solução de problemas complexos, pois a decomposição continua até o problema difícil ficar simples o suficiente para ser solucionado.

**NÍVEIS DE ABSTRAÇÃO E DECOMPOSIÇÃO**

Ao decompor, podemos perceber os níveis de abstração. Usando um exemplo da biologia, o nosso corpo humano possui vários órgãos e cada um é formado por vários tecidos, que são formadas por várias células, as quais possuem vários componentes.

Assim, podemos afirmar que os métodos e atributos estão em um nível de abstração abaixo do nível das classes, isso porque os métodos e atributos são os **detalhes** da implementação das classes.

Já as interfaces estão em um nível de abstração acima das classes.

**CONCRETO E ABSTRATO**

Existem classes e métodos concretos, classes e métodos abstratos e as interfaces.  Vamos relembrar as diferenças entre esses conceitos:

* método abstrato é a assinatura de um método;
* método concreto é a assinatura junto com a sua implementação;
* classes concretas possuem apenas métodos concretos e elas podem instanciar objetos.

**TIPOS ABSTRATOS E INTERFACE**

As [**classes abstratas**](https://www.devmedia.com.br/polimorfismo-classes-abstratas-e-interfaces-fundamentos-da-poo-em-java/26387)possuem pelo menos um método abstrato e não podem instanciar objetos, apenas herdar.

Já as interfaces possuem apenas métodos abstratos e não podem instanciar objetos.

**POLIMORFISMO E FLEXIBILIDADE PARA MUDAR**

Para definir uma interface de tipos abstratos para uma ou mais classes, podemos escolher a palavra reservada **interface**ou**abstract class**. Ambas formas funcionam e conseguem o polimorfismo.

Esse polimorfismo ocorre quando duas ou mais subclasses conseguem invocar métodos que contêm a mesma assinatura, mas que se comportam de forma diferente em cada subclasse. Vamos ver um exemplo a seguir.

**EXEMPLO DE POLIMORFISMO**

Observando a imagem, temos quatro seres vivos. Assim, teremos uma superclasse chamada de **SerVivo** e quatro subclasses: **Homem**, **Pato**, **Cachorro** e **Gato**.

Na superclasse, temos um método chamado **emitir\_som()**, o qual pode ser repetido em todas as subclasses. Portanto, temos a mesma assinatura com comportamentos diferentes, pois cada ser vivo emite um som diferente.

**ACOPLAMENTO**

O [**acoplamento**](https://www.devmedia.com.br/entendendo-coesao-e-acoplamento/18538)entre as classes é sobre como elas se conectam. Em geral, os padrões de projeto buscam evitá-lo. Pois, em um sistema, as classes podem ser intercaladas, mas é importante que uma dependa da outra o mínimo possível.

Diminuir o acoplamento torna o código mais flexível para futuras modificações.

**EXEMPLO DE ACOPLAMENTO**

Sistemas podem ser acoplados de maneira forte ou fraca. Um exemplo de sistema fracamente acoplado são as peças de LEGO, que podem ser utilizadas para construir diversas montagens.

Já um quebra-cabeça, por exemplo, se trata de algo fortemente acoplado, pois não podemos simplesmente utilizar peças diferentes em qualquer lugar.

Ou seja, um sistema fortemente acoplado dificulta as atualizações e manutenções futuras.

**FACTORY METHOD**

Após todos esses conceitos, vamos aprender sobre o padrão de projeto **Factory Method**.

Ele é um padrão de criação, ou seja, ele lida com a criação de objetos. Assim, podemos dizer que esse método é uma fábrica que cria objetos.

Vamos entender o funcionamento do Factory Method por meio do seu diagrama. O **quadro 3** é responsável pela criação de um Factory Method abstrato, que deve conter outros membros, como métodos e atributos.

No **quadro 4**, é criada uma classe que tem a responsabilidade de implementar o Factory Method, informando qual produto será retornado.

No **quadro 2**, temos os produtos concretos que serão retornados pelas classes que implementarem o Factory Method.

Já o **quadro 1** representa a interface de todos os produtos finais. Assim, a classe que elaborar o Factory Method deve retornar os produtos que implementam essa interface.

Vale destacar que o nome **produto** é utilizado porque o próprio padrão trata os objetos retornados pelo Factory Method como produtos.

Na programação de um Factory Method, precisamos ter uma interface para os objetos que pretendemos criar. Isso porque todos os objetos concretos criados precisam implementar a interface também criada.

Vamos observar um exemplo prático de como o Factory Method funciona.

**FACTORY METHODY – DESENVOLVIMENTO DA INTERFACE**

Utilizando a linguagem de programação Java, vamos criar um Factory Method polígono. Ele verificará a quantidade dos lados de diferentes polígonos e lançará uma descrição informando a quantidade de lados.

Inicialmente, a interface é criada para informar quais são as propriedades dos objetos criados. Na interface, temos um método que retorna uma string para mostrar o nome do objeto ao executarmos a programação.

**FACTORY METHODY – DESENVOLVIMENTO DAS CLASSES**

Com a interface já criada, podemos criar as classes. Aqui, temos duas classes que irão representar dois polígonos: um quadrado e um triângulo.

**CRIAÇÃO DO FACTORY METHODY**

Essa programação é a criação do Factory, que possui um método público estático.

Nas implementações do Factory, é analisada a quantidade de lados dos polígonos para se obter um retorno.

**FUNCINAMENTO DO FACTORY METHODY**

Essa classe testa o funcionamento do Factory. Assim, ao analisar a quantidade de lados dos polígonos, o nome do polígono correspondente será retornado.

Com o Factory Method, podemos criar diferentes objetos. Além desses, seria possível construir vários outros polígonos apenas complementando a programação.

Por fim, este é o resultado apresentado: uma descrição do polígono, contendo a quantidade de lados e seu respectivo nome.

**VAMOS PRATICAR**

Você foi contratado por uma empresa de entregas e precisa criar um sistema para lidar com diferentes tipos de envios, como pacotes pequenos, pacotes grandes, documentos, etc. Cada tipo de envio possui diferentes características e métodos de cálculo de preço. Você deve criar diferentes classes de envio que implementam uma interface comum, permitindo que o sistema possa lidar com diferentes tipos de envio de maneira uniforme e consistente. Qual conceito da programação você pode utilizar para realizar essa atividade?

A abstração é a melhor abordagem para resolver o problema de lidar com diferentes tipos de objetos com comportamentos e características diferentes. Ao utilizar o padrão Factory Method, estamos criando uma abstração da criação de objetos, permitindo que o sistema seja facilmente expandido para lidar com novos tipos de objetos sem afetar o código existente. E ai, acertou?

Portanto, o Factory Method define uma interface para criar objetos. Com esse padrão temos uma classe com um método, que cria um objeto.

Assim, com esse padrão, podemos:

* lidar com a criação de objetos;
* dar flexibilidade, permitindo a criação de novos Factories;
* determinar o tipo dos objetos por vir.

​

Com essas possibilidades, o Factory Method é um dos padrões de projetos mais utilizados.