**Design Patterns**

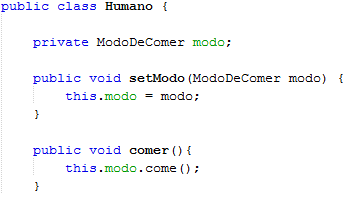
**Strategy**

Alterar o comportamento em tempo de execução

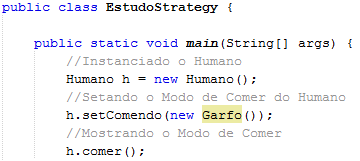
Exemplo: Modo de Comer de um Humano. Um Humano pode comer de garfo ou hashi.

Solução: Criar a **Interface** (Modo de Comer) onde você cria um **método abstrato** (come ()) que será implementado pelas classes filhas. Criar as **classes filhas** que irão **implementar** (implements) a interface assim herdando o método abstrato e tendo que implementa-lo.

Feito isso criar a **classe** (Humano) que será **composta pela interface** (**private** Modo de Comer) sendo assim terá que ser **criado os métodos** **SET** (ALT + INSERT) e o método de apresentação (comer ()).



Na **Main instanciar o humano** (humano h = new humano ()), depois **instanciar um modo de comer** (hashi ou garfo) do tipo Modo de Comer. Depois **setar o modo de comer do humano e dar get no modo de comer**.

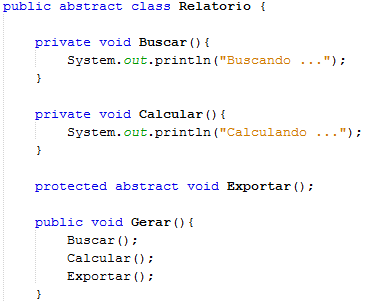


**Template Method**

Altera o comportamento de um método em questão. Realizar uma sequência de passos o qual um dos passos pode ser modificado.

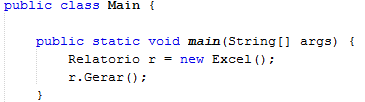
Exemplo: Geração de Relatórios onde você tem que buscar calcular e exportar para PDF ou EXCEL o relatório.

No Template você **cria uma classe abstrata** (Relatório), pois ela contem alguns métodos concretos. Sendo esses métodos o **Buscar do tipo privado, Calcular do tipo Privado e o Exportar do abstrato e protected**, pois tem dois tipos de exportar. Por fim o método **Gerar** que ira definir a **ordem de execução** dos métodos já criados.



Depois **criar as classes filhas** (PDF e EXCEL) que irão **estender** (extends) a classe pai, sendo assim **implementar o método abstrato** herdado.

Logo após **criar um pacote novo** onde ira **conter a main** e assim **instanciar um PDF do tipo Relatório** e **chamar o método de sequencia** (quando for chamar o método o único que poderá aparecer e o de sequencia).



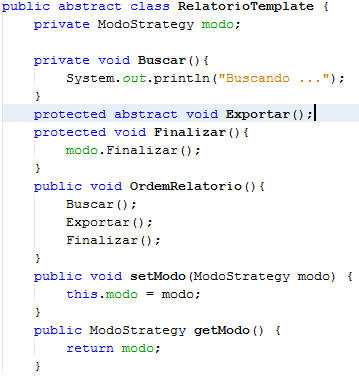
**Bridge**

Utilizado quando é desejável que uma interface possa variar independentemente das suas implementações.

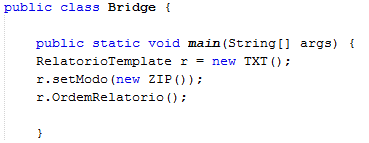
Exemplo: Um Relatório Exportado para TXT ou PDF com Zip ou Criptografia.

Começando pelo **Strategy cria sua interface e os filhos** onde ele irem implementar a interface pai.

Depois **cria se a classe abstrata do Template** onde a Interface do **Strategy ira compor ela** sendo assim precisando de um **método set**. A classe abstrata irar usar mais **quatro métodos** um **padrão** (Buscar ()) uma **abstrata** (Exportar ()) que será implementado pelas classes filhas do Template, e uma **método que ira chamar a Strategy** (Finalizar ()). E por ultima o **Método Gerar** que terá a ordem de execução.



Na main **instancia o RelatorioTemplate** e o seu **tipo** (TXT ou PDF) e depois **seta o modo** (zip ou cript) e por ultimo **executa o método Gerar().**



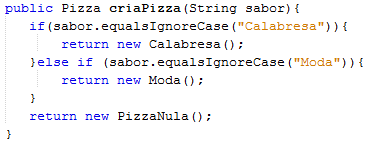
***Factory Method***

Delegar a responsabilidade de instanciar objetos à uma fábrica, diminuindo o acoplamento.

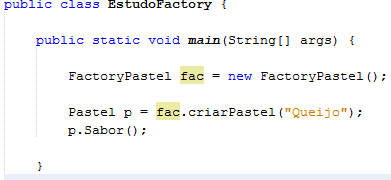
Exemplo: Pizzaria onde uma pizza tem um sabor e um modo para monta-la.

Primeiro **cria uma interface** (Pizza) onde tem um **metado abstrato** (sabor), depois as **classes filhas** (Calabresa, Moda e Nula) que irão implementar a classe pai (Pizza).

Depois **criar a classe Factory** onde está possuirá o **método do tipo do objeto que ira criar onde recebera uma variável do tipo String**. No seu corpo **tem um if** onde ira comparar duas strings sendo assim instanciar os objetos correspondentes.



Na **Main instancia-se a Factory** e **instancia uma pizza chamando o método da factory passando uma string.**



***State***

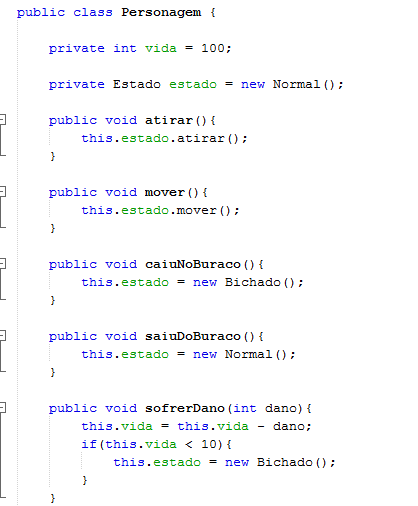
Exemplo: Personagem onde ele tem vários estados normal, morto.

Implementação parecida com o Strategy.

Primeiro **cria uma interface** (Estado) onde tem um **dois metados abstratos** (atira) e (corre), depois as **classes filhas** (Normal, Morrendo e Invencivel) que irão implementar a classe pai (Estado).

Depois **criar a classe Personagem** onde está contem a classe Estado já recebendo o estado Normal vai criar os métodos de troca de estado.

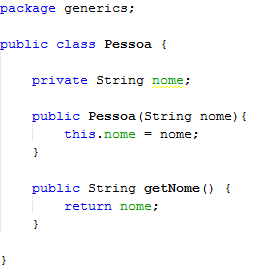
Exemplo caiu no buraco = morrendo ; saiu do buraco = normal

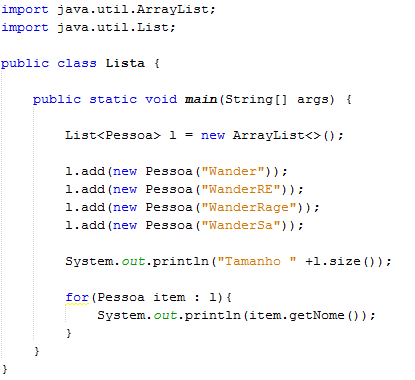
******

***List (Não e um pattern)***

Apenas pra revisar sobre list e array list.

O List é uma interface e o ArrayList é a classe que a implementa



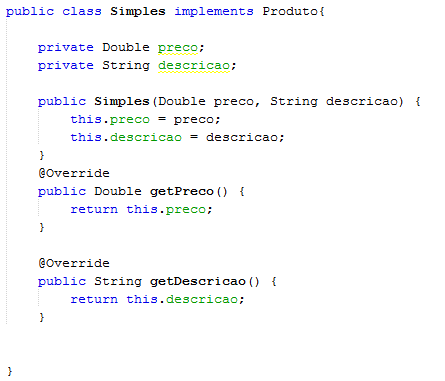


***Composite***

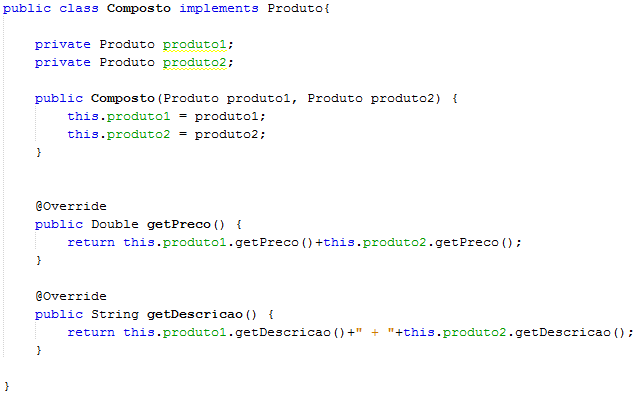
Exemplo: Cupom Fiscal de um Produto onde herda para Simples e Composto onde o composto tem mais de um produto.

Primeiro **cria uma interface** (Produto) onde tem um **dois metados abstratos** não void (getPreço) e (getDescrição), depois as **classes filhas** (Simples e Compostos) que irão implementar a classe pai (Produto).

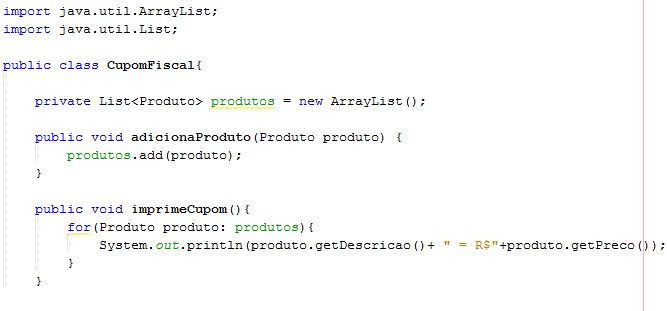
Na Classe filha Simples tem um construtor recebendo duas variáveis preço e descrição.



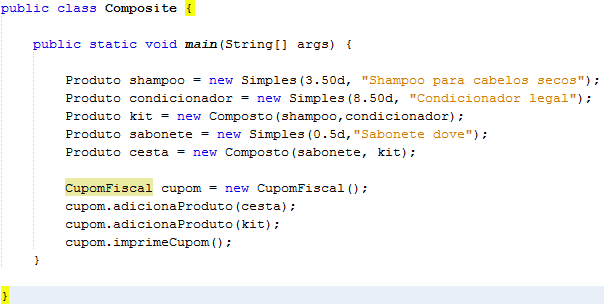
Na Classe filha Composto ela tem uma composição de 2 produtos e passa esses dois produtos no método Construtor



Depois **criar a classe Cupom Fiscal** onde está contem uma Lista de Produtos é métodos de adicionar produto e imprimir o produto através de um FOR.



Na Main você instancia um produto simples ou composto. Depois instancia um cupom fiscal e adiciona no Cupom Fiscal e manda Imprimir.

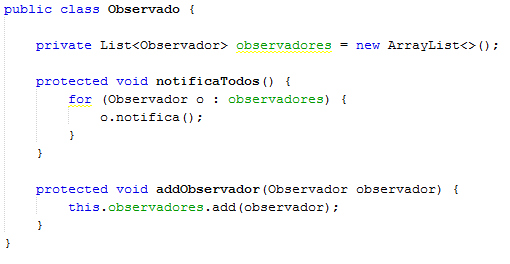


***Observer***

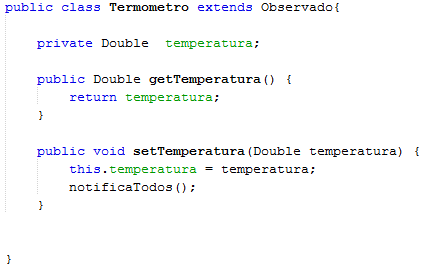
Exemplo: notificar a temperatura de um termômetro.

Primeiro **cria uma interface** (Observador) onde tem **um método abstrato** (notifica) depois as **classes filhas** (Display e App) que irão implementar a classe pai (Observador).

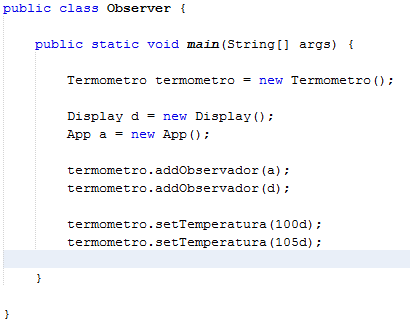
Depois criar a Classe Observada que recebe uma Lista de Observadores onde tem os métodos de adicionar Observador e de notificar através de um FOR.



Prosseguindo cria-se uma classe Termômetro onde estende da classe Observada. Com uma variável onde se cria um get e um set onde no set chama a classe notifica todos.



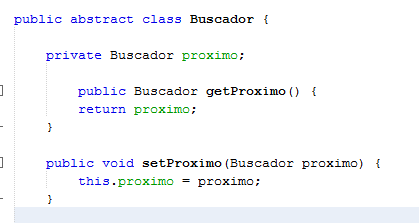
Na Main você instancia o Termômetro, o Display e o App. Depois adiciona o display e o app no Termômetro. Pronto agora e só setar a temperatura do termômetro.



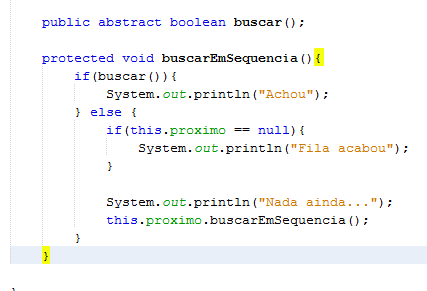
***Chain of Responsibility***

Exemplo: Buscar um arquivo em um local (Nuvem,Local,Servidor).

Primeiro cria uma classe abstrata (Buscador) contendo uma variável do tipo buscador de nome próximo e seu get e set.

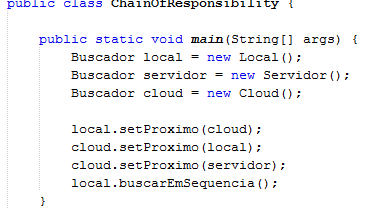


Depois um método abstrato do tipo boolean (Buscar ()) e depois um Método void (BuscarEmSequencia()) onde SE chamar o método Buscar retornar um texto (Achou) OU Se this.proximo == null retornar um texto(Fila Acabou) SE NÃO retornar um texto (Nada Ainda) e setar o próximo para chamar o método novamente.



Criar 3 classes que irão extender e implementar a classes Buscador onde deve retornar true ou falso e um texto.

Na Main instanciar as 3 classes como Buscador e chamar o setProximo para casa uma passando um parâmetro.



***Proxy/Decorator***

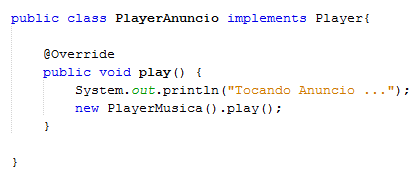
Exemplo: Spotify a onde Player onde toca o Player Normal e outro com Anuncio.

Cria uma interface Player com o método void play ().

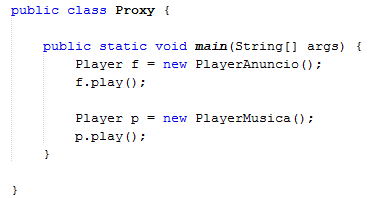
Cria 2 classes (Player Anuncio e Musica) onde essa duas implementa Player.

Na PlayerMusica faz o método play com um Texto qualquer.

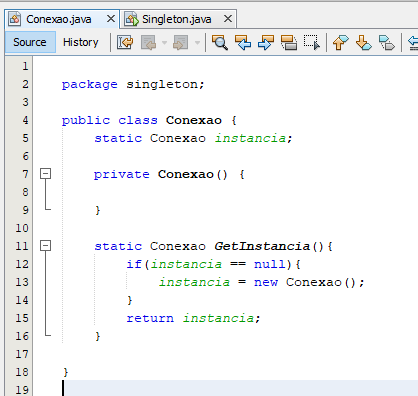
Na PlayerAnuncio faz o método play com um texto qualquer instanciando o PlayerMusica e chamando seu método.

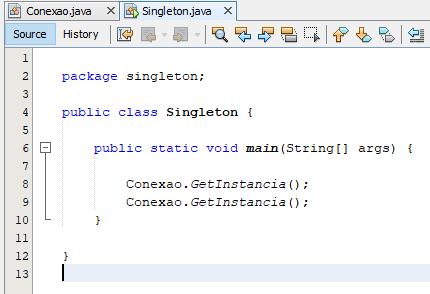
******

Na Main

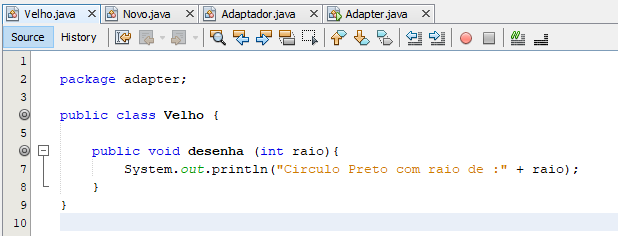
******

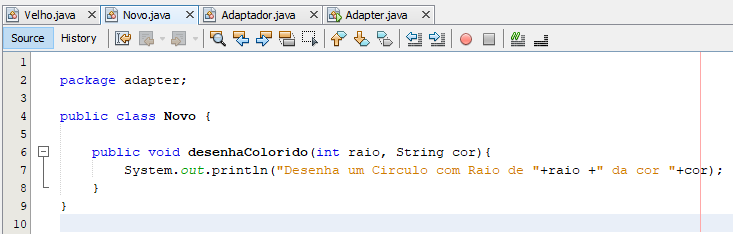
***Singleton***

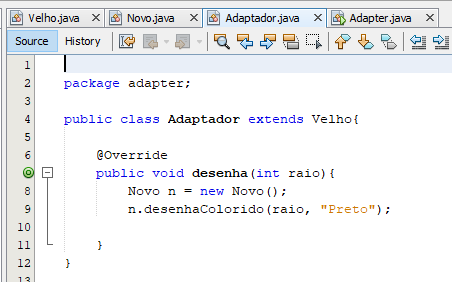
******

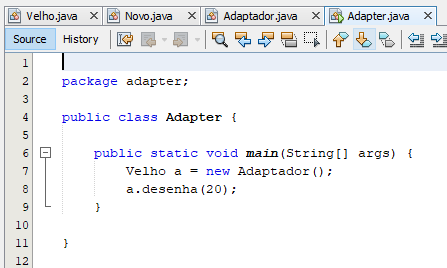
******

***Adapter***

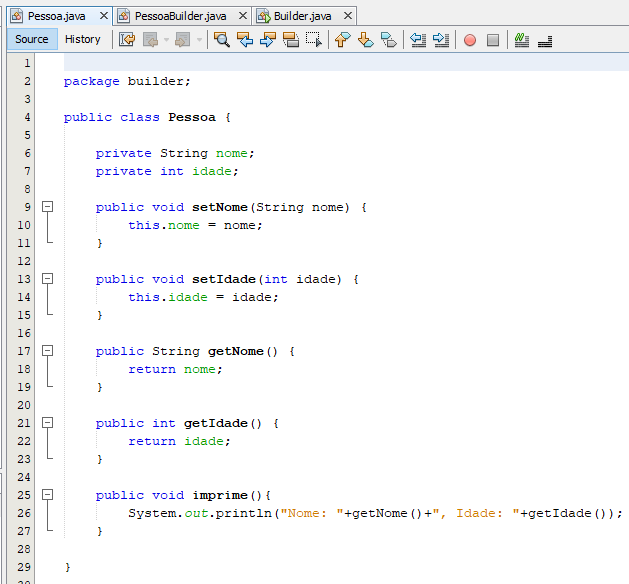
******

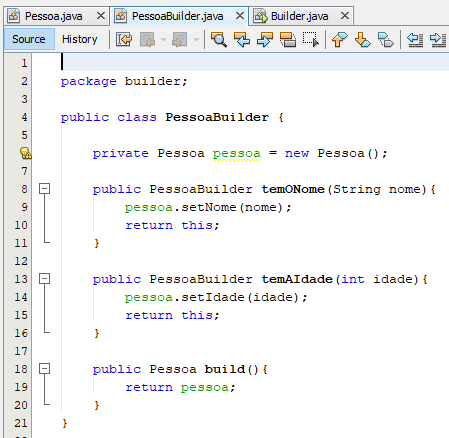
******

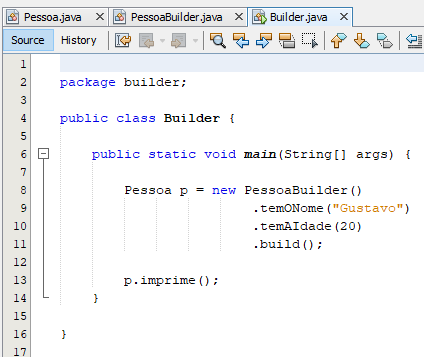
******

******

***Builder***

******

******

******