Skip to navigation

Marcos Brizeno

Desenvolvimento de Software #showmethecode

Template Method

setembro 18, 2011

Mão na massa: Template Method

Vamos mostrar agora o padrão Template Method!

Problema

Suponha um player de música que oferece várias maneiras de reproduzir as músicas de uma playlist. Para exemplificar suponha que podemos reproduzir a lista de músicas da seguinte maneira:

- Ordenado por nome da música
- Ordenado por nome do Autor
- Ordenado por ano
- Ordenado por estrela (preferência do usuário)

Uma ideia seria utilizar o padrão Strategy e implementar uma classe que define o método de reprodução para cada tipo de reprodução da playlist. Esta seria uma solução viável, pois manteríamos a flexibilidade para implementar novos modos de reprodução de maneira bem simples.

No entanto, observe que, o algoritmo para reprodução de uma playlist é o mesmo, independente de qual modo esta sendo utilizado. A única diferença é a criação da playlist, que leva em consideração um dos atributos da música.

Para suprir esta dificuldade vamos ver o padrão Template Method!

Template Method

Como de costume, vejamos a intenção do padrão Template Method:

"Definir o esqueleto de um algoritmo em uma operação, postergando alguns passos para as subclasses. Template Method permite que subclasses redefinam certos passo de um algoritmo sem mudar a estrutura do mesmo." [1]

Perfeito para o nosso problema! Precisamos definir o método de ordenação da Playlist mas só saberemos qual atributo utilizar em tempo de execução. Vamos definir então a estrutura de dados que define uma música:

```
1  public class MusicaMP3 {
2      String nome;
3      String autor;
4      String ano;
5      int estrelas;
6
7      public MusicaMP3(String nome, String autor, String ano, int estrela) {
8          this.nome = nome;
9          this.autor = autor;
10         this.ano = ano;
11         this.estrelas = estrela;
12     }
13 }
```

Para escolher como a playlist deve ser ordenada vamos criar uma pequena enumeração:

```
1  public enum ModoDeReproducao {
2      porNome, porAutor, porAno, porEstrela
3  }
```

Agora vamos escrever a nossa classe que implementa o método template para ordenação da lista:

```
public abstract class OrdenadorTemplate {
    public abstract boolean isPrimeiro(MusicaMP3 musica1, MusicaMP3 musica2

    public ArrayList<MusicaMP3> ordenarMusica(ArrayList<MusicaMP3> lista) {
        ArrayList<MusicaMP3> novaLista = new ArrayList<MusicaMP3>();
        for (MusicaMP3 musicaMP3 : lista) {
            novaLista.add(musicaMP3);
        }

        for (int i = 0; i < novaLista.size(); i++) {
            for (int j = i; j < novaLista.size(); j++) {
                if (!isPrimeiro(novaLista.get(i), novaLista.get(j))) {
                    MusicaMP3 temp = novaLista.get(j);
                    novaLista.set(j, novaLista.get(i));
                    novaLista.set(i, temp);
                }
            }
        }

        return novaLista;
    }
}
```

Basicamente definimos um método de ordenação, no caso o método Bolha, e deixamos a comparação dos atributos para as subclasses. Essa é a ideia de utilizar o método template, definir o esqueleto e permitir a personalização dele nas subclasses. Veja o exemplo da classe a seguir que ordena as músicas por nome:

```
public class OrdenadorPorNome extends OrdenadorTemplate {

    @Override
    public boolean isPrimeiro(MusicaMP3 musica1, MusicaMP3 musica2) {
        if (musica1.nome.compareToIgnoreCase(musica2.nome) <= 0) {
            return true;
        }
        return false;
    }

}
```

Para implementar as outras formas de reprodução da lista basta definir, na subclasse, o método que compara uma música com outra e diz se é necessário trocar.

O exemplo a seguir define a ordenação baseado no nível de preferência do usuário (aquelas estrelinhas dos players de música)

```
public class OrdenadorPorEstrela extends OrdenadorTemplate {

    @Override
    public boolean isPrimeiro(MusicaMP3 musica1, MusicaMP3 musica2) {
        if (musica1.estrelas > musica2.estrelas) {
            return true;
        }
        return false;
    }

}
```

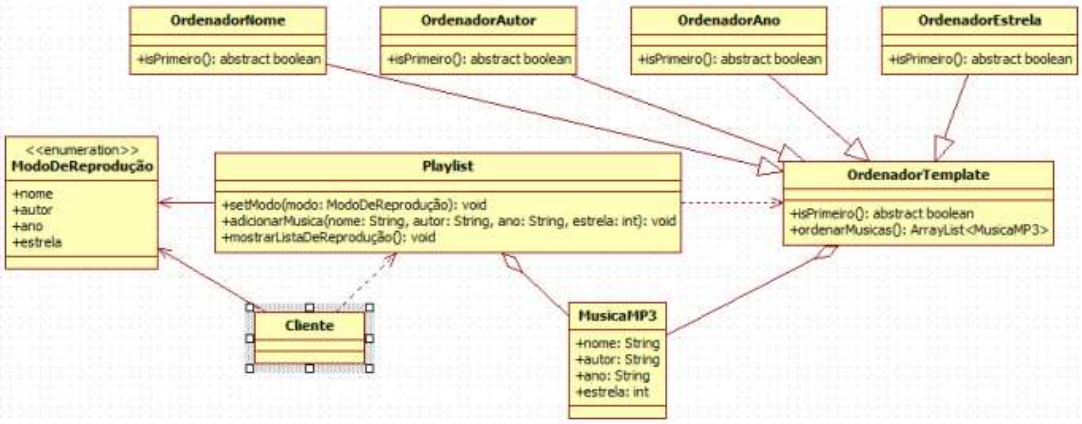
Pronto, definimos o algoritmo padrão e suas variações. Agora vamos ver a classe que manipula a playlist:

```
public class PlayList {
    protected ArrayList<MusicaMP3> musicas;
    protected OrdenadorTemplate ordenador;

    public PlayList(ModoDeReproducao modo) {
        musicas = new ArrayList<MusicaMP3>();
        switch (modo) {
        case porAno:
            ordenador = new OrdenadorPorAno();
            break;
        case porAutor:
            ordenador = new OrdenadorPorAutor();
            break;
        case porEstrela:
            ordenador = new OrdenadorPorEstrela();
            break;
        case porNome:
            ordenador = new OrdenadorPorNome();
            break;
        default:
            break;
        }
    }

    public void setModoDeReproducao(ModoDeReproducao modo) {
        ordenador = null;
        switch (modo) {
        case porAno:
            ordenador = new OrdenadorPorAno();
            break;
        case porAutor:
            ordenador = new OrdenadorPorAutor();
            break;
        case porEstrela:
            ordenador = new OrdenadorPorEstrela();
            break;
        case porNome:
            ordenador = new OrdenadorPorNome();
            break;
        default:
            break;
        }
    }

    public void adicionarMusica(String nome, String autor, String ano,
            int estrela) {
        musicas.add(new MusicaMP3(nome, autor, ano, estrela));
    }

    public void mostrarListaDeReproducao() {
        ArrayList<MusicaMP3> novaLista = new ArrayList<MusicaMP3>();
        novaLista = ordenador.ordenarMusica(musicas);

        for (MusicaMP3 musica : novaLista) {
            System.out.println(musica.nome + " - " + musica.autor + "\n Ano
                    + musica.ano + "\n Estrelas: " + musica.estrelas);
        }
    }
}
```

Definimos métodos para inserir músicas e exibir a playlist e, de acordo com o parâmetro passado, criamos uma playlist. O código cliente ficaria da seguinte maneira:

```
1  public static void main(String[] args) {
2
3      PlayList minhaPlayList = new PlayList(ModoDeReproducao.porNome);
4      minhaPlayList.adicionarMusica("Everlong", "Foo Fighters", "1997", 5);
5      minhaPlayList.adicionarMusica("Song 2", "Blur", "1997", 4);
6      minhaPlayList.adicionarMusica("American Jesus", "Bad Religion", "1993",
7              3);
8      minhaPlayList.adicionarMusica("No Cigar", "Milencollin", "2001", 2);
9      minhaPlayList.adicionarMusica("Ten", "Pearl Jam", "1991", 1);
10
11     System.out.println("=== Lista por Nome de Musica ===");
12     minhaPlayList.mostrarListaDeReproducao();
13
14     System.out.println("\n=== Lista por Autor ===");
15     minhaPlayList.setModoDeReproducao(ModoDeReproducao.porAutor);
16     minhaPlayList.mostrarListaDeReproducao();
17
18     System.out.println("\n=== Lista por Ano ===");
19     minhaPlayList.setModoDeReproducao(ModoDeReproducao.porAno);
20     minhaPlayList.mostrarListaDeReproducao();
21
22     System.out.println("\n=== Lista por Estrela ===");
23     minhaPlayList.setModoDeReproducao(ModoDeReproducao.porEstrela);
24     minhaPlayList.mostrarListaDeReproducao();
25 }
```

Veja o quão simples foi alterar o modo como a lista é construída. No final, essa é a estrutura do projeto utilizando o Templte Method:



Um pouco de teoria:

Já exemplificamos a principal vantagem do padrão Template Method, a facilidade de alteração do algoritmo principal. No entanto, deve-se tomar cuidado ao utilizar o padrão pois, se for preciso definir muitas operações nas subclasses, talvez seja necessário refatorar o código ou repensar o design.

Outro problema é que, ao definir o método que executa o algoritmo genérico, não é possível proteger este método das subclasses. Ou seja, o cliente do código precisa saber exatamente quais operações substituir para alcançar o efeito desejado. Por exemplo, caso o programador da subclasse OrdenadorPorEstrela redefinisse o método de ordenação para realizar qualquer outra operação, poderíamos ter problemas.

Por isso é tão importante definir os métodos que devem ser sobrescritos como abstratos (abstract em java, ou virtual puro em C++). Dessa maneira garante-se o princípio Open/Closed [2], que diz que uma classe deve ser aberta para extensões (fácil criar novas maneiras de reproduzir músicas) e fechada para alterações.

Código fonte completo

O código completo pode ser baixado no seguinte repositório Git: https://github.com/MarcosX/Padroes-de-Projeto.

Os arquivos estão como um projeto do eclipse, então basta colocar no seu Workspace e fazer o import.

Se gostou do post compartilhe com seus amigos e colegas, senão, comente o que pode ser melhorado. Encontrou algum erro no código? Comente também. Possui alguma outra opinião ou alguma informação adicional? Comenta ai! 😀

Referências:

[1] GAMMA, Erich et al. Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos.
[2] WIKIPEDIA. SOLID. Disponível em: http://en.wikipedia.org/wiki/SOLID_(object-oriented_design). Acesso em: 15 set. 2011.

Padrões de Projeto, Template Method Java, Padrões, Projeto, Template Method 5 Comentários

__PRESENT