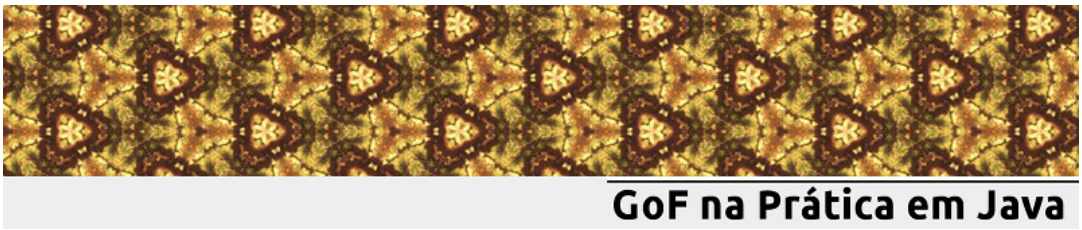
Factory Method

Fernando Anselmo
GoF na Prática em Java

Função deste Padrão

Define uma interface para a criação de um objeto, mas deixa as sub-classes definirem qual classe deve ser instanciada e permite a uma classe delegar a instanciação das sub-classes.



1 Ficha do Padrão

Tipo : Criacional, diz respeito ao processo de criação dos objetos.

Conhecimentos : Classes Abstratas e Classes Concretas.

Consequências : Provê ganchos para subclasses, e conecta a hierarquia de classes paralelas quando existe uma delegação.

É usado quando : Uma classe não pode antecipar a classe de objeto que deve ser criada. Uma classe permite que suas sub-classes especifiquem os objetos que serão criados. Classes delegam responsabilidades para uma dentre várias sub-classes auxiliares, e se deseja localizar o conhecimento de qual sub-classe auxiliar implementa essa delegação.

2 Problema

É necessário criar objetos de várias sub-classes de veículos, sem que o cliente necessáriamente saiba qual objeto está sendo criado.

3 Prévia Estrutura de Classes

Classe abstrata para organizar a família dos veículos:

Listagem 1: *Classe Abstrata Veiculo*

```
1  abstract class Veiculo {
2    private String modelo;
3
4    public String getModelo() {
5      return modelo;
6    }
7    public void setModelo(String modelo) {
8      this.modelo = modelo;
9    }
10 }
```

Classe exemplo de um veículo tipo Opala:

Listagem 2: *Classe Opala*

```
class Opala extends Veiculo {
  public Opala() {
    setModelo("Opala");
  }
}
```

Classe exemplo de um veículo tipo Vectra:

Listagem 3: *Classe Vectra*

```
class Vectra extends Veiculo {
  public Vectra() {
    setModelo("Vectra");
  }
}
```

4 Aplicação do Padrão

Classe abstrata para estrutura do criador dos tipos de veículo:

Listagem 4: *Classe Abstrata GMCriador*

```
abstract class GMCriador {
  public abstract Opala factoryOpala();
  public abstract Vectra factoryVectra();
}
```

Classe concreta com as fábricas de criação:

Listagem 5: *Classe GMCriadorConcreto*

```
class GMCriadorConcreto extends GMCriador {
  public Opala factoryOpala() {
    return new Opala();
  }
  public Vectra factoryVectra() {
    return new Vectra();
  }
}
```

Classe com um exemplo de uso pelo cliente:

Listagem 6: *Classe Cliente*

```
public class Cliente {
  public static void main(String[] args) {
    new Cliente().executar();
  }
  public void executar() {
    GMCriador gm = new GMCriadorConcreto();
    Veiculo vectra = gm.factoryVectra();
    Veiculo opala = gm.factoryOpala();
    System.out.println(vectra.getModelo());
    System.out.println(opala.getModelo());
  }
}
```

Referências

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides *Design Patterns. Elements of Reusable Object-Oriented Software 1 ed*. Estados Unidos, Addison-Wesley, 1995, ISBN 0-201-63361-2.