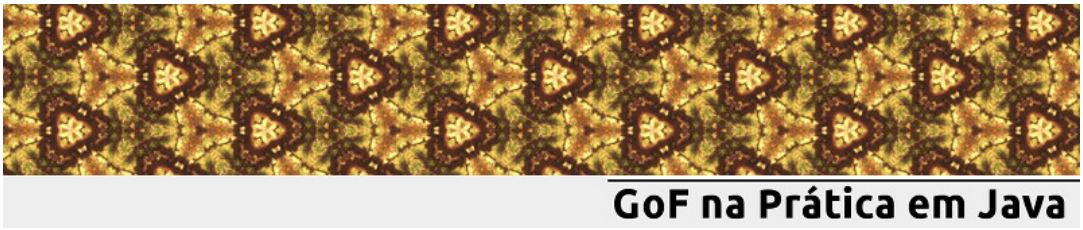
Abstract Factory

Fernando Anselmo
GoF na Prática em Java

Função deste Padrão

Prover uma interface para a criação de famílias de objetos relacionados ou independentes. Isolar a criação de objetos de seu uso e remover a dependência entre o cliente que usa o objetos e a classe dos objetos produzidos.



1 Ficha do Padrão

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

Conhecimentos : Interface, Classes Abstratas e Classes Concretas.

Consequências : Isolar as classes concretas, facilitar a troca de famílias de produtos e facilitar o suporte a novos tipos de produtos.

É usado quando : Um sistema deve ser independente de como os seus elementos são criados, compostos e representados; um sistema deve ser configurado para trabalhar com uma única família dentre múltiplas famílias de produtos; uma família de produtos relacionados é projetada para ser usada em conjunto e há necessidade de reforçar essa restrição; se quer criar uma biblioteca de classes de produtos, revelando apenas suas interfaces e não suas implementações.

2 Problema

O cliente, dono de um PetShop, precisa indicar para seus clientes as melhores raças de Cães e Gatos que são ideais para usar como modelo de Companhia ou Guarda sem precisar conhecer a estrutura de classes concretas criadas.

3 Prévia Estrutura de Classes

Classe abstrata para organizar a família dos Pets:

Listagem 1: *Classe Abstrata Pet*

```
1 abstract class Pet {
2   private String nome;
3   public Pet(String nome) {
4     this.nome = nome;
5   }
6   public String toString() {
7     return nome;
8   }
9 }
```

Classe abstrata para organizar a família dos Cachorros:

Listagem 2: *Classe Abstrata Cachorro*

```
abstract class Cachorro extends Pet {
  public Cachorro(String nome) {
    super(nome);
  }
}
```

Classe abstrata para organizar a família dos Gatos:

Listagem 3: *Classe Abstrata Gato*

```
abstract class Gato extends Pet {
  public Gato(String nome) {
    super(nome);
  }
}
```

Classe exemplo para um Cachorro de Companhia:

Listagem 4: *Classe Shitzu*

```
class Shitzu extends Cachorro {
  public Shitzu() {
    super("Shitzu");
  }
}
```

Classe exemplo para um Cachorro de Guarda:

Listagem 5: *Classe Pastor*

```
class Pastor extends Cachorro {
  public Pastor() {
    super("Pastor");
  }
}
```

Classe exemplo para um Gato de Companhia:

Listagem 6: *Classe Persa*

```
class Persa extends Gato {
  public Persa() {
    super("Persa");
  }
}
```

Classe exemplo para um Gato de Guarda:

Listagem 7: *Classe Ragdoll*

```
class Ragdoll extends Gato {
  public Ragdoll() {
    super("Ragdoll");
  }
}
```

4 Aplicação do Padrão

Interface com a estrutura para as Fábricas:

Listagem 8: *interface Modelo*

```
interface Modelo {
  Cachorro getCachorro();
  Gato getGato();
}
```

Classe da fábrica de construção dos Pets de Companhia:

Listagem 9: *Classe FabricaCompanhia*

```
class FabricaCompanhia implements Modelo {
  public Cachorro getCachorro() {
    return new Shitzu();
  }
  public Gato getGato() {
    return new Persa();
  }
}
```

Classe da fábrica de construção dos Pets de Guarda:

Listagem 10: *Classe FabricaGuarda*

```
class FabricaGuarda implements Modelo {
  public Cachorro getCachorro() {
    return new Pastor();
  }
  public Gato getGato() {
    return new Ragdoll();
  }
}
```

Classe com um exemplo de uso pelo cliente:

Listagem 11: *Classe PetShop*

```
import java.util.Scanner;

public class PetShop {
  private Scanner sc = new Scanner(System.in);
  public static void main(String [] args) {
    new PetShop().selecionar();
  }
  public void selecionar() {
    System.out.println("Informe 1-Companhia ou 2-Guarda");
    byte opc = sc.nextByte();
    Modelo modelo = null;
    switch (opc) {
      case 1: modelo = new FabricaCompanhia(); break;
      case 2: modelo = new FabricaGuarda(); break;
    }
    System.out.println("Cachorro: " + modelo.getCachorro());
    System.out.println("Gato: " + modelo.getGato());
  }
}
```

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.