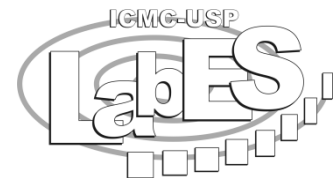




Design Patterns

Lucas Bueno Ruas de Oliveira
Profa. Elisa Yumi Nakagawa

SSC 122 – Engenharia de Software II
1. Semestre 2010



Classificação

- Gamma et al. (1995) definiram 23 padrões que possuem aplicação em diversas áreas.
- Os padrões são classificados em três grupos
 - Padrões de Estrutura
 - Padrões de Criação
 - Padrões de Comportamento

Padrões de Criação

- Isolar do cliente os detalhes do processo de instanciação de um objeto
- Permite que o sistema se torne menos dependente de como os objetos são criados.

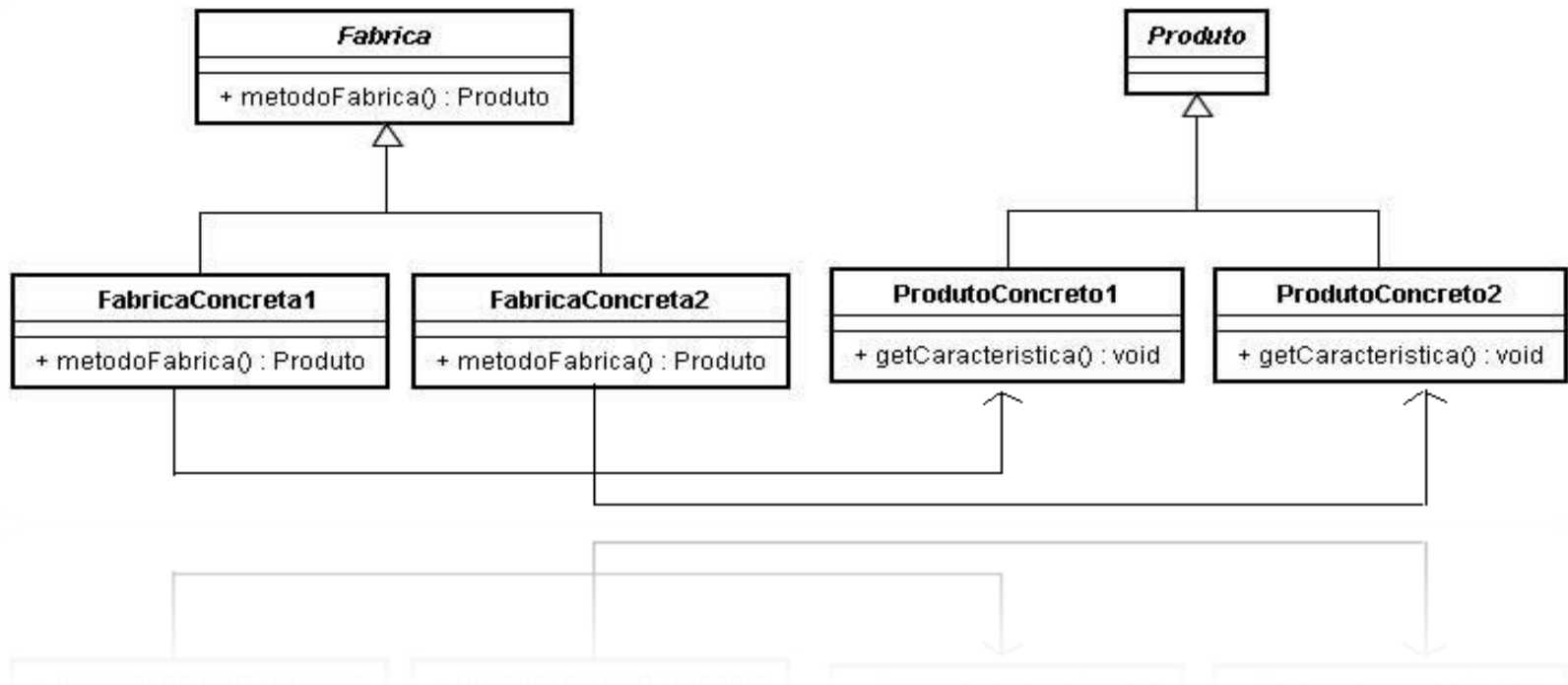
Padrões de Criação

- *Factory Method* (Método Fábrica)
- *Abstract Factory* (Fábrica Abstrata)
- *Builder* (Construtor)
- *Prototype* (Protótipo)
- *Singleton* (Objeto Único)

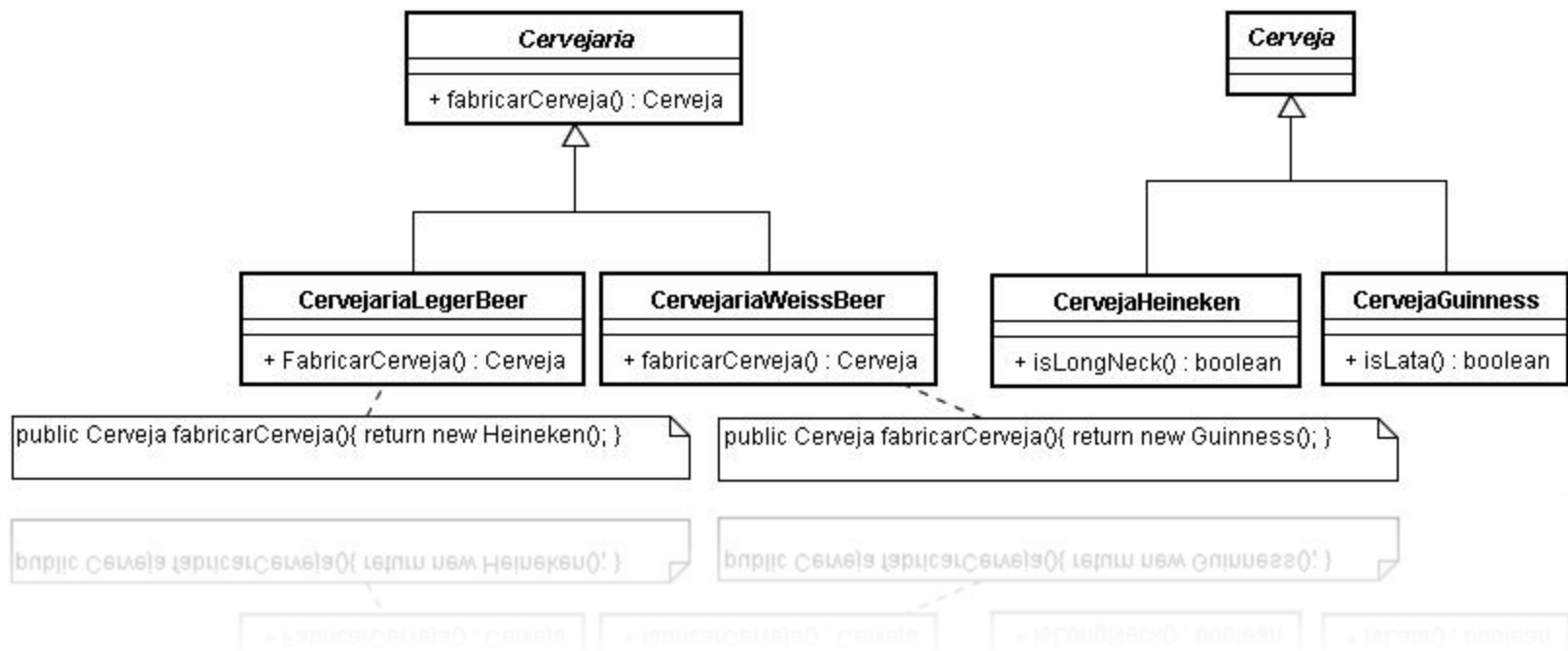
Factory Method - Problema

- Quando não existe a intenção de se especificar detalhes sobre a criação de um determinado objeto.
- Transferência da responsabilidade de criação de uma classe para outra classe.

Factory Method - Estrutura



Factory Method - Exemplo



Factory Method - Exemplo

```
public abstract class Cervejaria {  
  
    abstract public Cerveja fabricarCerveja();  
  
}
```

```
public class CervejariaWeissBeer extends Cervejaria{  
  
    public Cerveja fabricarCerveja(){  
        return new CervejaGuinness();  
    }  
  
}
```

```
public class CervejariaLagerBeer extends Cervejaria{  
  
    public Cerveja fabricarCerveja(){  
        return new CervejaHeineken();  
    }  
  
}
```


Factory Method - Exemplo

```
public class Cerveja {  
  
    private double teorAlcoolico;  
    private int qtdCerveja;  
    protected double preco;  
  
    public int getQtdCerveja(){return qtdCerveja;}  
    public void setQtdCerveja(int ml){qtdCerveja = ml;}  
  
    public double getTeorAlcoolico(){return teorAlcoolico;}  
    public void setTeorAlcoolico(double ta) {teorAlcoolico = ta;}  
  
    public double getPreco(){return preco;}  
    public void setPreco(double p) {preco = p;}  
}
```

Factory Method - Exemplo

```
public class CervejaGuinness extends Cerveja{  
  
    private boolean lata;  
  
    public CervejaGuinness(){}  
  
    public boolean isLata() {return lata;}  
    public void setLata(boolean lt) {lata = lt;}  
  
    public double getPreco() {return preco;}  
    public void setPreco(double p) {preco = p;}  
}
```

Factory Method - Exemplo

```
public class CervejaHeineken extends Cerveja{

    private boolean longNeck;

    public CervejaHeineken() {}

    public boolean isLongNeck() {return longNeck;}
    public void setLongNeck(boolean ln) {longNeck = ln;}

    public double getPreco(){return preco;}
    public void setPreco(double p) {preco = p;}

}
```

```

public class App {
    public static void main(String args[]){
        //Constroi as cervejarias
        CervejariaWeissBeer fabricaGui = new CervejariaWeissBeer();
        CervejariaLagerBeer fabricaHei = new CervejariaLagerBeer();
        //Criando um "fardinho" de latas, fardos de latas vem com 12
        List<CervejaGuinness> fardoGui = new ArrayList<CervejaGuinness>();
        CervejaGuinness guinness;
        //Fabricando Cervejas
        for(int i = 0; i < 12; ++i){
            guinness = (CervejaGuinness) fabricaGui.fabricarCerveja();
            guinness.setLata(true);
            guinness.setPreco(9.90);
            guinness.setTeorAlcoolico(6.5);
            fardoGui.add(guinness);
        }
        //Criando um "fardinho" de long necks, fardos de long neck vem com 6
        List<CervejaHeineken> fardoHei = new ArrayList<CervejaHeineken>();
        CervejaHeineken heineken;
        for(int i = 0; i < 6; ++i){
            heineken = (CervejaHeineken) fabricaHei.fabricarCerveja();
            heineken.setLongNeck(true);
            heineken.setPreco(1.79);
            heineken.setTeorAlcoolico(4.5);
            fardoHei.add(heineken);
        }
        double precoHei=0, precoGui=0, teorGui=0, teorHei=0;
        for(CervejaGuinness g: fardoGui){
            precoGui += g.getPreco();teorGui+=g.getTeorAlcoolico()*500/100;}
        for(CervejaHeineken h: fardoHei){
            precoHei += h.getPreco();teorHei+=h.getTeorAlcoolico()*355/100;}
        System.out.println("O fardo de Guinness custou: "+precoGui+ " e você consumiu "+teorGui+" ml de alcool." );
        System.out.println("O fardo de Heineken custou: "+precoHei+ " e você consumiu "+teorHei+" ml de alcool." );
        System.out.println("Se beber, não dirija!");
    }
}

```

Factory Method - Consequências

- É retirado de um objeto a responsabilidade de sua criação.
- Fábricas concretas podem se tornar criadoras de diversos objetos que possuem uma característica em comum.

Singleton (Objeto Único)- Problema

- Como garantir que em uma aplicação só exista uma única instância de uma determinada classe?
- Como evitar que objetos com alto custo de criação possam prejudicar a sua aplicação.

Singleton (Objeto Único)- Solução

Singleton
- <u>instance : Singleton</u>
- Singleton() : void <u>+ getInstance() : Singleton</u>

1. Construtor Privado
2. Variável estática do tipo do *Singleton* dentro da própria classe.
3. Método de acesso estático getInstance().

Singleton (Objeto Único)- Solução

```
3 public class Singleton {
4
5     private static Singleton SINGLETON;
6
7     //Construtor PRIVADO, ninguém instancia a classe fora dela.
8     private Singleton() {}
9
10    static public Singleton getInstance() {
11        //cria o singleton apenas uma vez
12        if(SINGLETON == null) {
13            SINGLETON = new Singleton();
14        }
15        return SINGLETON;
16    }
17
18 }
```


Singleton (Objeto Único)- Solução

```
3 public class App {  
4  
5     public static void main(String args[]){  
6         //ESTA ABORDAGEM NÃO EXISTE  
7         //Singleton sing = new Singleton();  
8         //AGORA ESTA CORRETO  
9         Singleton singA = Singleton.getInstance();  
10        System.out.println("Eu sou um singleton e minha instância é: "+singA);  
11        //Imprime:  
12        //Eu sou um singleton e minha instância é: br.usp.icmc.pattern.singleton.Singleton@19821f  
13        Singleton singB = Singleton.getInstance();  
14        System.out.println("Eu sou um singleton e minha instância é: "+singB);  
15        //Imprime:  
16        //Eu sou um singleton e minha instância é: br.usp.icmc.pattern.singleton.Singleton@19821f  
17        System.out.println("Por que os toString() de singA e singB possuem o mesmo retorno?");  
18    }  
19 }
```

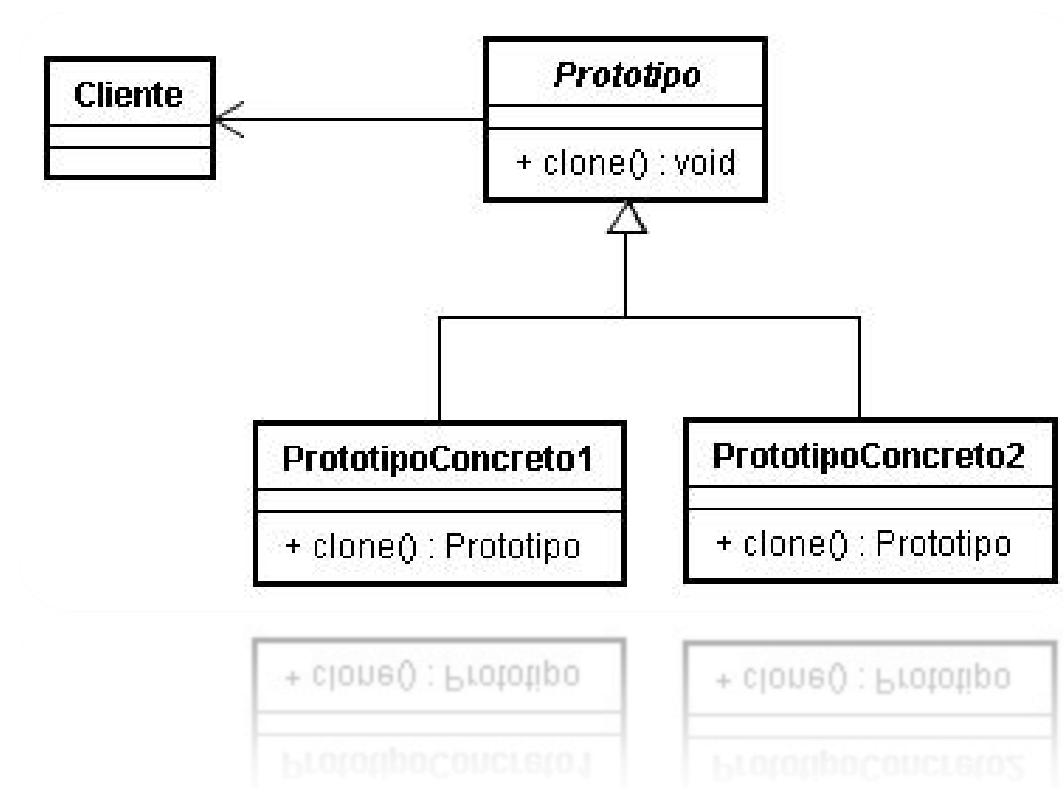
Singleton - Consequências

- O objeto é acessível em qualquer lugar da aplicação.
- Garantia de que não sejam criadas mais de uma instância de uma determinada classe.
- Possibilita que métodos pertencentes a objetos com alto custo de criação possam ser utilizados sem prejudicar o desempenho da aplicação.

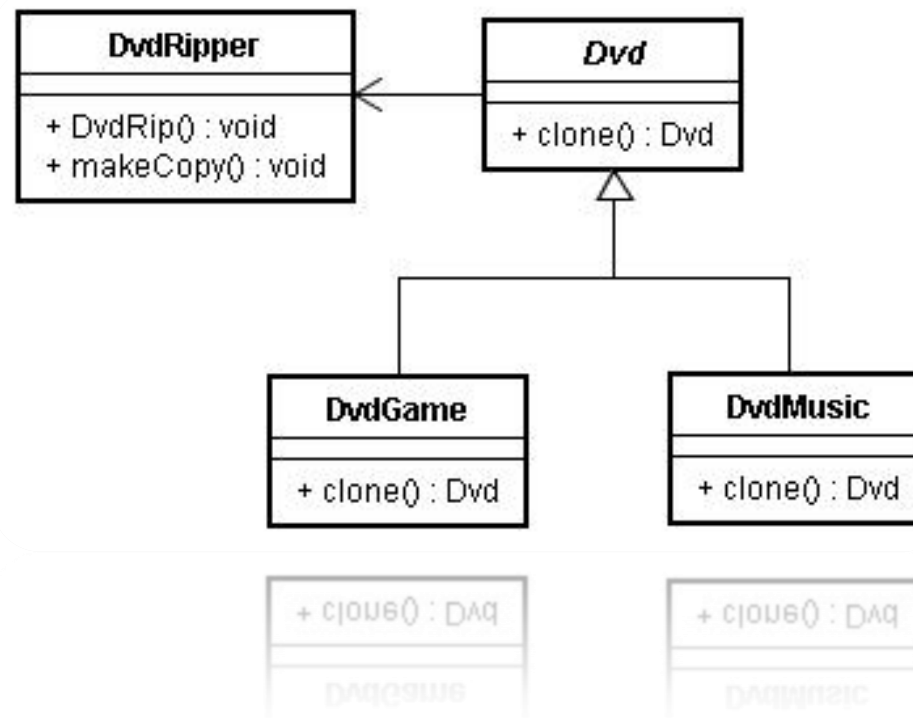
Prototype (Protótipo) - Problema

- Necessidade de se criar de maneira organizada diversos objetos com informações idênticas.
- Garantia de segurança na manutenção da informação.

Prototype (Protótipo) - Solução



Prototype (Protótipo) - Exemplo



Prototype (Protótipo) - Exemplo

```
public class DvdRipper {  
  
    public DvdRipper() {}  
  
    public Dvd makeCopy(Dvd dvd) {  
        return dvd.clone();  
    }  
}
```

```
public abstract class Dvd implements Cloneable{  
  
    @Override  
    abstract protected Dvd clone();  
}
```

Prototype (Protótipo) - Exemplo

```
public class DvdGame extends Dvd{

    private String nome;
    private String console;
    private int indicacaoIdade;

    public DvdGame() {}

    public String getConsole(){return console;}
    public void setConsole(String csl){console = csl;}

    public int getIndicacaoIdade(){return indicacaoIdade;}
    public void setIndicacaoIdade(int indIda){indicacaoIdade = indIda;}

    public String getNome(){return nome;}
    public void setNome(String n){nome = n;}
    @Override
    protected Dvd clone() {
        DvdGame clone = new DvdGame();
        clone.setConsole(console);
        clone.setIndicacaoIdade(indicacaoIdade);
        clone.setNome(nome);
        return clone;
    }
    //Verifique se os objetos sao os mesmos ...
    public String toString(){
        return "Nome: "+nome+" Indicação de Idade: "+ indicacaoIdade+ " Console: "
        + console+"\nInstância: " + super.toString()+"\n";
    }
}
```

Prototype (Protótipo) - Exemplo

```
public class DvdMusic extends Dvd{

    private String titulo;
    private String autor;
    private int numFaixas;

    public DvdMusic(){}

    public String getAutor() {return autor;}
    public void setAutor(String g){autor = g;}

    public int getNumFaixas(){return numFaixas;}
    public void setNumFaixas(int numF) {numFaixas = numF;}

    public String getTitulo(){return titulo;}
    public void setTitulo(String ttl){titulo = ttl;}
    @Override
    protected Dvd clone() {
        DvdMusic clone = new DvdMusic();
        clone.setAutor(autor);
        clone.setNumFaixas(numFaixas);
        clone.setTitulo(titulo);
        return clone;
    }
    public String toString(){
        return "Titulo: "+titulo+" Num Faixas: " + numFaixas+ " Gravadora: "
            + autor+"\nInstância: " + super.toString()+"\n";
    }
}
```


Prototype (Protótipo) - Exemplo

```
public class App {  
  
    public static void main(String[] args) {  
        //cria a copiadora de dvds  
        DvdRipper ripper = new DvdRipper();  
        //constroi um DVD do The Doors  
        DvdMusic dvdMusic = new DvdMusic();  
        dvdMusic.setAutor("The Doors");  
        dvdMusic.setNumFaixas(12);  
        dvdMusic.setTitulo("Elektra");  
        //Constroi um DVD do Biohazard 5  
        DvdGame dvdGame = new DvdGame();  
        dvdGame.setConsole("PlayStation 3");  
        dvdGame.setIndicacaoIdade(16);  
        dvdGame.setNome("Biohazard 5");  
        List<Dvd> copiasList = new ArrayList<Dvd>();  
        //faz 15 cópias de cada  
        for(int i = 0; i<15; ++i){  
            copiasList.add(ripper.makeCopy(dvdGame));  
            copiasList.add(ripper.makeCopy(dvdMusic));  
        }  
        for(Dvd dvd : copiasList)  
            System.out.println(dvd);  
    }  
}
```

Prototype (Protótipo) - Consequências

- Criação simplificada de diversos objetos diferentes com o mesmo valor.
- Dados importantes podem ser manipulados, garantindo-se a segurança da informação por meio de uma cópia.

Pesquisar

- *Abstract Factory* (Fábrica Abstrata)
- *Builder* (Construtor)

Exercício

- Você é um engenheiro de software que vai implementar um sistema sobre um banco de dados legado (já existente).
- Em sua aplicação, por questões estratégicas, você vai precisar utilizar um outro banco de dados, diferente do primeiro.
- Suponha que o custo de criação de um objeto da classe Connection seja alto.

Exercício

- Utilize um padrão de projeto para construir classes que retornem objetos do tipo Connection (note que temos objetos tipo Connection de dois diferentes bancos de dados).
- Implemente a filosofia de Singleton para garantir a criação de apenas uma Connection para cada banco.

Exercício

- Modele a solução desse problema explicitando (utilizando-se *notes* no diagrama UML) a diferença entre os métodos *getConnection* das duas classes criadoras de conexão.

Bibliografia

- ALEXANDER, Christopher. *A Pattern Language: Towns, Buildings, Construction*. Ed. USA: Oxford University Press, 1977.
- GAMMA, Erich et al, *A Design Patterns: Elements of Reusable Object-Oriented Software*, Construction. 36. Ed. Portland: Addison-Wesley, 1995.
- SERAPHIM, Enzo. *Tópicos Especiais de Programação*. Notas de aula, Universidade Federal de Itajubá-UNIFEI, Itajubá-MG, Brasil, 2008.