Programação Orientada a Objetos

Padrões de Criação

Cristiano Lehrer, M.Sc.

ybadoo

Objetivos

- Apresentar cada um dos 23 padrões clássicos descrevendo:
  - O problema que solucionam.
  - A solução.
  - Diagramas UML (*Unified Modeling Language*).
  - Exemplos em Java.
  - Aplicações típicas.

ybadoo

O que é um padrão?

“Um padrão descreve um problema que ocorre repetidas vezes em nosso ambiente, e então descreve o núcleo da solução para aquele problema, de tal maneira que você pode usar essa solução milhões de vezes sem nunca fazê-lo da mesma forma duas vezes.”

(contexto, problema, solução)

ybadoo

Padrões clássicos ou padrões GoF

- O livro *Design Patterns* (1994) de Erich Gamma, John Vlissides, Ralph Jonhson e Richard Helm, descreve 23 padrões de projeto:
  - São soluções genéricas para os problemas mais comuns do desenvolvimento de software orientado a objetos.
  - O livro tornou-se um clássico na literatura orientada a objeto e continua atual.
  - Não são invenções. São documentação de soluções obtidas através da experiência. Foram coletados de experiências de sucesso na indústria de software, principalmente de projetos em C++ e SmallTalk.
  - Os quatro autores, são conhecidos como *The Gang of Four*, ou *GoF*.

ybadoo

Benefícios

- Capturam *expertise* e tornam esse conhecimento acessível a *non-experts*, usando formato padrão.
- Facilitam comunicação entre desenvolvedores:
  - Linguagem comum.
- Facilitam reutilização de projetos bem-sucedidos.
- Facilitam compreensão de projeto.
- Facilitam modificação de projeto.
- Promovem documentação de projeto.

ybadoo

Elementos de um padrão

- Nome.
- Problema:
  - Quando aplicar o padrão, em que condições?
- Solução:
  - Descrição abstrata de um problema e como usar os elementos disponíveis (classes e objetos) para solucioná-lo.
- Consequências:
  - Custos e benefícios de se aplicar o padrão.
  - Impacto na flexibilidade, extensibilidade, portabilidade e eficiência do sistema.

ybadoo

Classificação dos Padrões de Projeto

- Propósito – o que o padrão faz:
  - Padrões de criação: abstraem o processo de criação de objetos a partir da instanciação de classes.
  - Padrões estruturais: tratam da forma como classes e objetos estão organizados para a formação de estruturas maiores.
  - Padrões comportamentais: preocupam-se com algoritmos e a atribuição de responsabilidades entre objetos.
- Escopo – em que o padrão de projeto é aplicado:
  - Padrões de classes: em geral estáticos, definidos em tempo de compilação.
  - Padrões de objetos: em geral dinâmicos, definidos em tempo de execução.

Classificação dos padrões do GoF

		Propósito		
		1. Criação	*2. Estrutura*	*3. Comportamento*
Escopo	Classe	*Factory Method*	*Class Adapter*	*Interpreter* *Template Method*
	Objeto	*Abstract Factory* *Builder* *Prototype* *Singleton*	*Object Adapter* *Bridge* *Composite* *Decorator* *Facade* *Flyweight* *Proxy*	*Chain of Responsibility* *Command* *Iterator* *Mediator* *Memento* *Observer* *State* *Strategy* *Visitor*

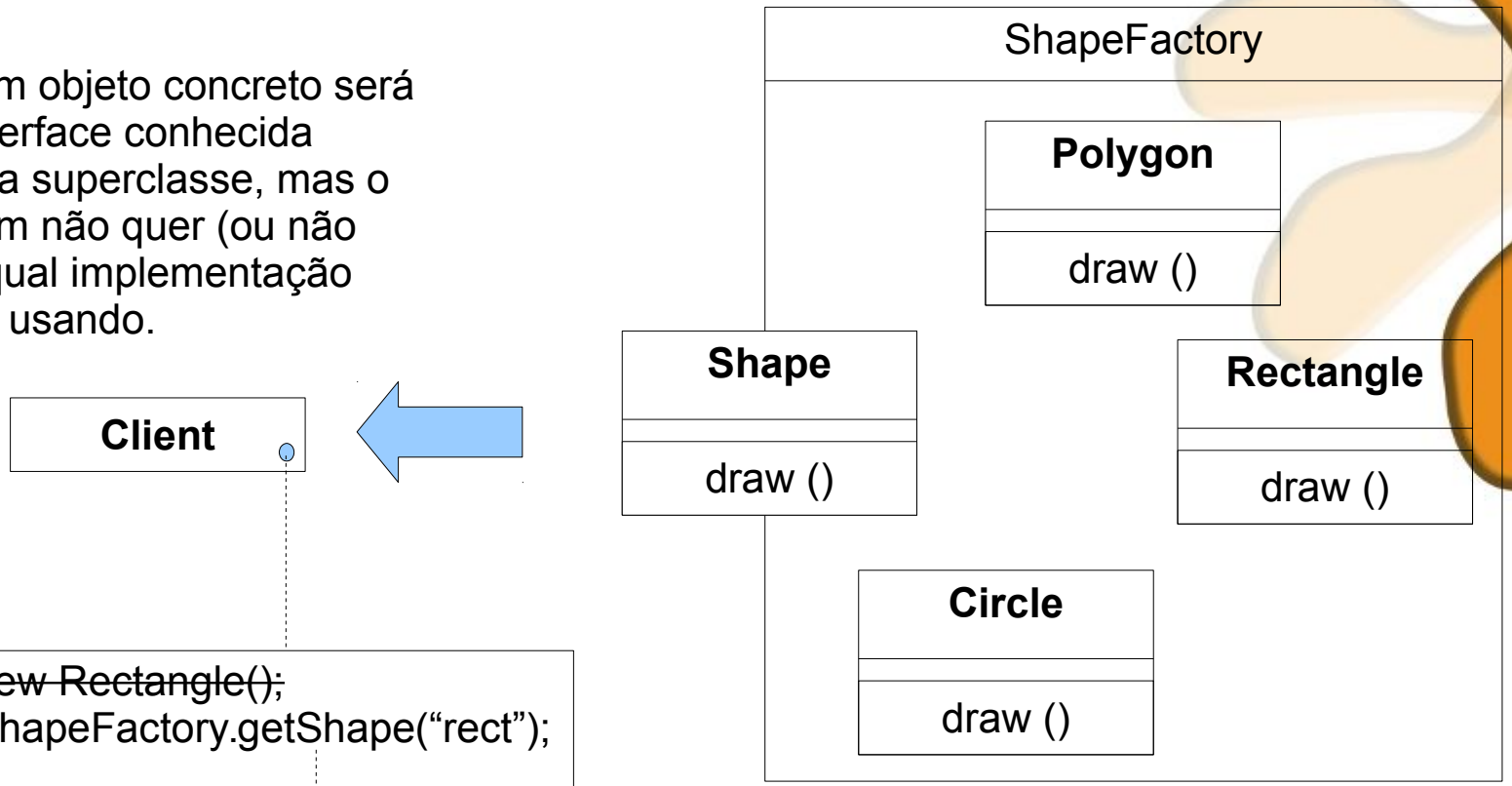
Factory Method

“Definir uma interface para criar um objeto, mas deixar que subclasses decidirem que classe instanciar. O Factory Method permite que uma classe delegue a responsabilidade de instanciamento às subclasses.”

ybadoo

Problema

O acesso a um objeto concreto será através da interface conhecida através de sua superclasse, mas o cliente também não quer (ou não pode) saber qual implementação concreta está usando.




```
~~Shape shape = new Rectangle();~~
Shape shape = ShapeFactory.getShape("rect");
shape.draw();
```

```
public static Shape getShape(String type)  {
        ShapeFactory factory = (ShapeFactory) typeMap.get(type);
        return factory.getShape();  // non-static Factory Method
}
```

Solução

- Propósito:
  - Definir uma interface para criação (*factory*) de objetos.
  - Subclasses decidem qual classe instanciar.
- Também conhecido como *Virtual Constructor*.
- Motivação:
  - *Framework*: relações entre objetos com classes abstratas.
- Aplicação:
  - Quando a classe não deve antecipar a classe de objetos que deve ser instanciada.

ybadoo

Como implementar

- É possível criar um objeto sem ter conhecimento algum de sua classe concreta?
  - Esse conhecimento deve estar em alguma parte do sistema, mas não precisa estar no cliente.
  - *Factory Method* define uma interface comum para criar objetos.
  - O objetivo específico é determinado nas diferentes implementações dessa interface.
  - O cliente do *Factory Method* precisa saber sobre implementações concretas do objeto criador do produto desejado.

ybadoo

Estrutura

- *Product*:
  - Define a interface de objetos que o método *factory* cria.
- *ConcreteProduct*:
  - Implementa a interface de *Product*.
- *Creator*:
  - Declara o método *factory* o qual retorna um objeto do tipo *Product*.
  - Pode chamar o método *factory* para criar um objeto *Product*.
- *ConcreteCreator*:
  - Redefine o método *factory* para retornar uma instância de um *ConcreteProduct*.

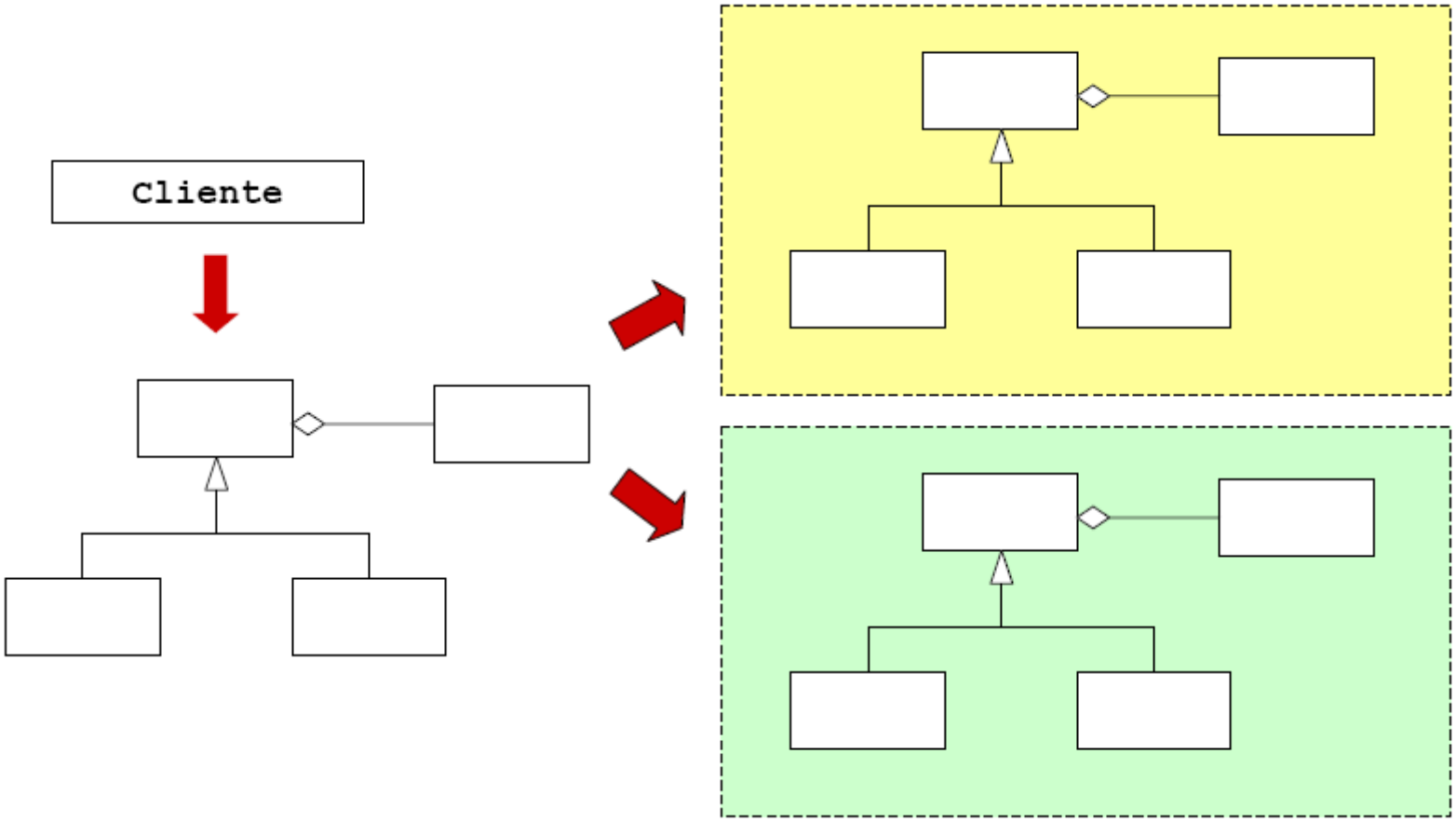
ybadoo

Abstract Factory

“Fornecer uma interface para criar famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.”

Problema

- Criar uma família de objetos relacionados sem conhecer suas classes concretas.



Solução

- Propósito:
  - Prover uma interface para geração de famílias de objetos relacionados (e/ou dependentes) sem especificar suas classes concretas.
- Também conhecido como *Kit*.
- Motivação:
  - *Toolkit* de *user interface* com múltiplas aparências.
- Aplicação:
  - Um sistema deve ser configurado de acordo com uma dentre várias famílias de produtos.

Estrutura

- *AbstractFactory*:
  - Declara uma interface para operações que criam objetos-produto abstratos.
- *ConcreteFactory*:
  - Implementa as operações que criam objetos-produto concretos.
- *AbstractProduct*:
  - Declara uma interface para um tipo de objeto-produto.
- *ConcreteProduct*:
  - Define um objeto-produto a ser criado pela correspondente fábrica concreta. Implementa a interface de *AbstractProduct*.

ybadoo

Consequências

- Isola *ConcreteClasses*.
- Cliente só utiliza interfaces para criar os produtos da família.
- Encapsula cada família de produtos e permite que o cliente troque uma por outra quando for conveniente.
- Como o *AbstractFactory* fixa o conjunto de produtos que podem ser criados, o suporte a novos produtos fica prejudicado.

ybadoo

Builder

“Separar a construção de um objeto complexo de sua representação para que o mesmo processo de construção possa criar representações diferentes.”

ybadoo

Problema

- Um programa que lê arquivos no formato RTF (*Rich Text Format*) deve ser capaz de converter um documento RTF para outros formatos distintos.
- Como modificar o programa facilmente de modo que ele converta um documento RTF para outros formatos diferentes?

ybadoo

Aplicabilidade

- Um algoritmo para a criação de um objeto deve ser independente de como suas partes são montadas.
- O processo de construção deve permitir diferentes representações para o objeto construído.

ybadoo

Estrutura (1/2)

- *Builder*:
  - Especifica uma interface abstrata para criação de partes de um objeto-produto.
- *ConcreteBuilder*:
  - Constrói e monta partes do produto pela implementação da interface de *Builder*.
  - Define e mantém a representação que cria.
  - Fornece uma interface para recuperação do produto.
- *Director*:
  - Constrói um objeto usando a interface *Builder*.

Estrutura (2/2)

- *Product*:
  - Representa o objeto complexo em construção. *ConcreteBuilder* constrói a representação interna do produto e define o processo pelo qual ele é montado.
  - Inclui classes que definem as partes constituintes, inclusive as interfaces para a montagem das partes no resultado final.

Quando usar?

- *Builder* permite que uma classe se preocupe com apenas uma parte da construção de um objeto. É útil em algoritmos de construção complexos:
  - Use-o quando o algoritmo para criar um objeto complexo precisar ser independente das partes que compõem o objeto e da forma como o objeto é construído.
- *Builder* também suporta substituição dos construtores, permitindo que a mesma interface seja usada para construir representações diferentes dos mesmos dados:
  - Use quando o processo de construção precisar suportar representações diferentes do objeto que está sendo construído.

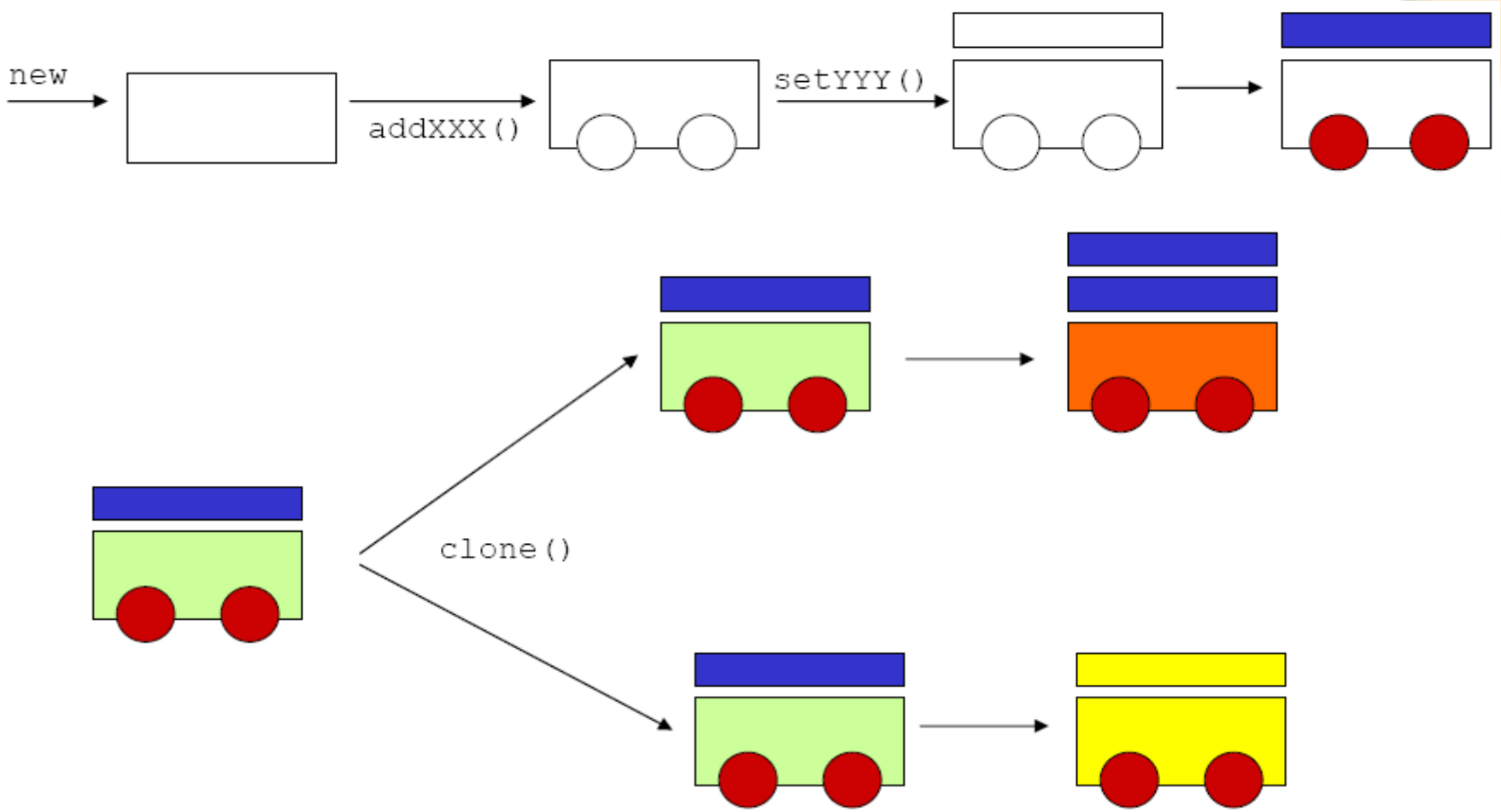
Prototype

"Especificar os tipos de objetos a serem criados usando uma instância como protótipo e criar novos objetos ao copiar este protótipo."

ybadoo

Problema

- Criar um objeto novo, mas aproveitar o estado previamente existente em outro objeto.



Estrutura

- *Prototype*:
  - Declara uma interface para clonar a si próprio.
- *ConcretePrototype*:
  - Implementa uma operação para clonar a si próprio.
- *Client*:
  - Cria um novo objeto solicitando a um protótipo que clone a si próprio.

Consequências

- O padrão *Prototype* permite que um cliente crie novos objetos ao copiar objetos existentes.
- Uma vantagem de criar objetos deste modo é poder aproveitar o estado existente de um objeto.
- Cliente chama um protótipo, requisitando em Clone dele. Em seguida usa o objeto clonado e/ou inicia objetos com ele.

Singleton

“Garantir que uma classe só tenha uma única instância, e prover um ponto de acesso global a ela.”

ybadoo

Problema

- Garantir que apenas um objeto exista, independente de número de requisições que receber para criá-lo.
- Aplicações:
  - Um único banco de dados.
  - Um único acesso ao arquivo de *log*.
  - Um único objeto que representa um vídeo.
- Objetivo: garantir que uma classe só tenha uma instância.

ybadoo

Estrutura

- Singleton:
  - Define uma operação *Instance* que permite aos clientes acessarem sua única instância. *Instance* é uma operação de classe (ou seja, em Java é um método de classe).
  - Pode ser responsável pela criação da sua própria instância única.

Prós e contras

- Vantagens:
  - Acesso central e extensível a recursos e objetos.
- Desvantagens:
  - Qualidade da implementação depende da linguagem.
  - Uso (abuso) como substituto para variáveis globais.
  - Difícil ou impossível de implementar em ambiente distribuído (é preciso garantir que cópias serializadas refiram-se ao mesmo objeto).

ybadoo

Resumo (1/2)

- *Factory Method*:
  - Para isolar a classe concreta do produto criado da interface usada pelo cliente.
- *Abstract Factory*:
  - Para criar famílias inteiras de objetos que têm algo em comum sem especificar suas interfaces.
- *Builder*:
  - Para construir objetos complexos em várias etapas e/ou que possuam representações diferentes.

ybadoo

Resumo (2/2)

- *Prototype*:
  - Para criar objetos usando outro como base.
- *Singleton*:
  - Quando apenas uma instância for permitida.

ybadoo