Design Patterns

Hello!

I'm *Warteruzannan*

Departamento de Computação – UFSCar

Book, 1995 – Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides

Roteiro

- Introdução
- Padrões de projetos
- Conceitos necessários
- Exemplo de padrões
  - **Singleton**
  - **Template Method**

1

Design Patterns

be standardized, please

Introdução

- Os padrões de projetos surgiram em 1977 a partir das ideias de Christopher Alexander;
- Existiam padrões comuns de prédios quem eram eficazes e agradáveis;
- O padrão é uma descrição do problema e da essência de sua solução, de modo que a solução possa ser reusada em diferentes contextos.

Introdução

- Implementar um software reutilizável e flexível;

- Além de serem soluções já testadas para problemas comuns, tornaram-se um vocabulário para se falar sobre um projeto.

Na engenharia de software, um padrão de projeto é uma solução reutilizável para um problema que ocorre frequentemente dentro de um determinado contexto no design de software

Introdução

- Um padrão de projeto não é uma linguagem de programação;
- Ajudam a descobrir abstrações não tão óbvias;
- Objetos representando processos, algoritmos
  - State Pattern, Strategy, Observers, ...
- Melhoram a documentação, manutenção e comunicação;

Introdução

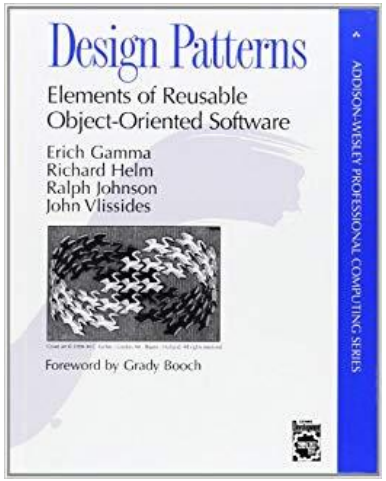
- Exemplo da comunicação SEM o uso de padrões:
  - **João pergunta a Pedro**: como você faz para atualizar a interface do usuário toda vez que o valor dos dados mudam?
  - **Pedro responde**: eu criei uma classe independente para manter os dados da aplicação. Toda vez que um dado é alterado, eu envio uma notificação para quem estiver interessado nesse dado. Esse componente, por sua vez, se atualiza com base no novo estado do sistema. (**E João: “Não entendi”**)

Introdução

- Exemplo da comunicação COM o uso de padrões:
  - **Pedro responde**: eu utilizei o padrão Observer;
  - **João responde**: Por que não disse isso antes?

Introdução

- Os padrões para projetos OO mais conhecidos foram originalmente descritos pela “Gangue dos Quatro” (*Gangue of Four – GoF*) em seu Livro: “Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos”



Introdução

- GoF descreve 23 padrões de projetos:
  - **Um nome** que servirá de referência ao padrão;
  - **Uma descrição** completa e o contexto de utilização desse padrão;
  - **A descrição dos elementos** que constituem o padrão, seus relacionamentos, responsabilidades e colaboradores;
  - **Uma descrição das consequências** da utilização do padrão.

Classificação

- A GoF propôs 23 padrões de projetos;
- Além disso, ela classificou esses padrões quanto a sua finalidade;
- Tal classificação ajuda a aprender os padrões mais rapidamente, bem como direcionar esforços na descoberta de novos padrões.

Classificação

- Podem ser classificados em três tipos:
  - **Criação:** voltado à criação de objetos;
  - **Estruturais**: se preocupam com a composição de classes ou objetos;
  - **Comportamental:** se preocupam com as interações e responsabilidades de objetos.

Classificação

		CRIAÇÃO	ESTRUTURAIS	COMPORTAMENTAL
ESCOPO	CLASSE	*Factory Method*	*Adapter*	*Interpreter* *Template Method*
	OBJETO	*Abstract Factory* *Builder* *Prototype* *Singleton*	*Bridge* *Composite* *Decorator* *Façade* *Flyweight* *proxy*	*Chain of Responsibility* *Command* *Iterator* *Mediator* *Memento* *Observer* *State* *Strategy* *Visitor*

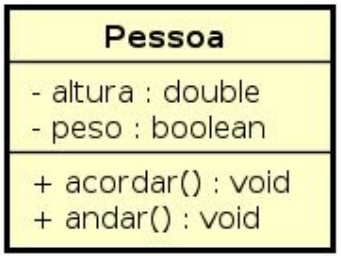
Objetos

How build?

Objetos

- Uma classe é uma representação de algo do mundo real e um objeto é uma instância dessa classe;

- Possui características (atributos) e comportamentos (operações);



Objetos

- Cada operação descreve uma ação com nome, parâmetros e tipo de retorno.
  - Conhecido como assinatura do método

```java
public int som(int numberA, int numberB){

  // Outras operações (caso existam)

     return numberA + numberB
}
```

Interfaces

How to communicate?

Interfaces

- Um conjunto de assinaturas que um objeto expõe;
- A interface de um objeto informa quais as requisições podem ser enviadas para ele;
- Um **Tipo** é um nome usado para denotar uma interface específica.

Interfaces

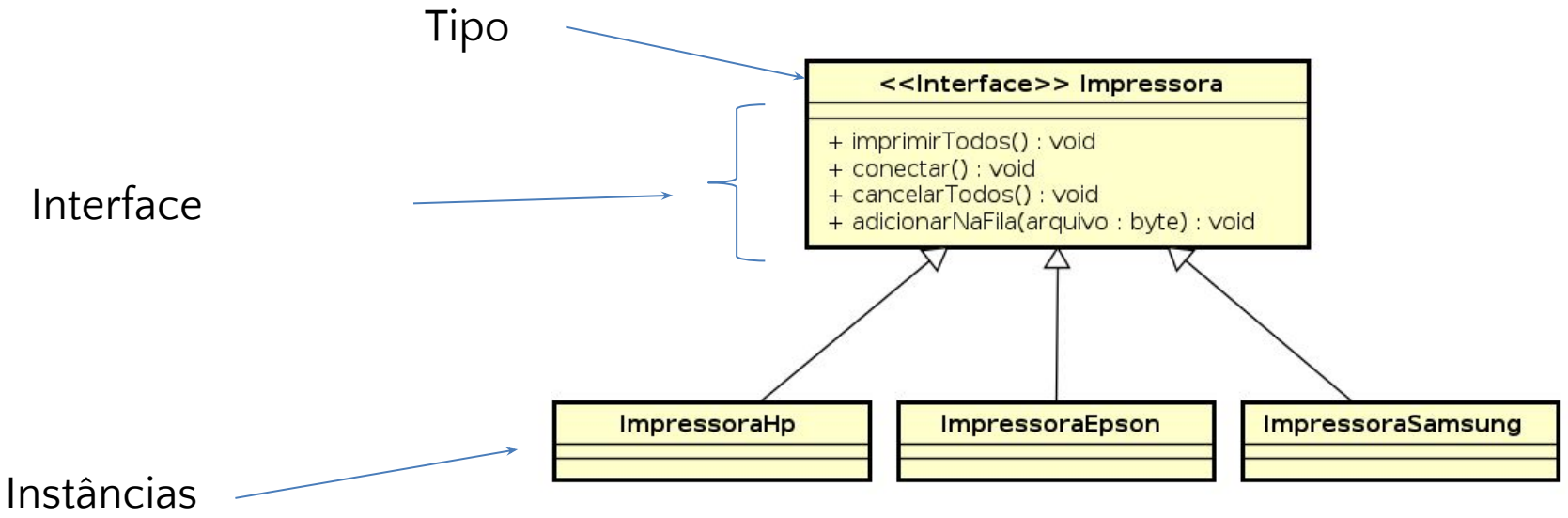
- Classes Abstratas e Interfaces, em Java, são abstrações cuja principal finalidade é definir uma interface comum a vários tipos de objetos;
- Interfaces não dizem nada sobre a implementação do objeto (ou seja, o interior dos métodos);
- Objetos com a mesma interface com implementações diferentes (polimorfismo).

Interfaces

- Para prover implementações para as operações abstratas das classes abstratas e das interfaces, uma classe concreta deve redefinir essas operações;

- Para isso, utiliza-se o mecanismo de herança e de realização de interface, respectivamente.

Interfaces


Tipo
Interface
Instâncias
<<Interface>> Impressora
+ imprimirTodos() : void
+ conectar() : void
+ cancelarTodos() : void
+ adicionarNaFila(arquivo : byte) : void
ImpressoraHp
ImpressoraEpson
ImpressoraSamsung

Requisito de aplicativo para impressão de documentos PDF.

O aplicativo deve utilizar “Impressora” como meio de imprimir os documentos.

Ligações (biding)

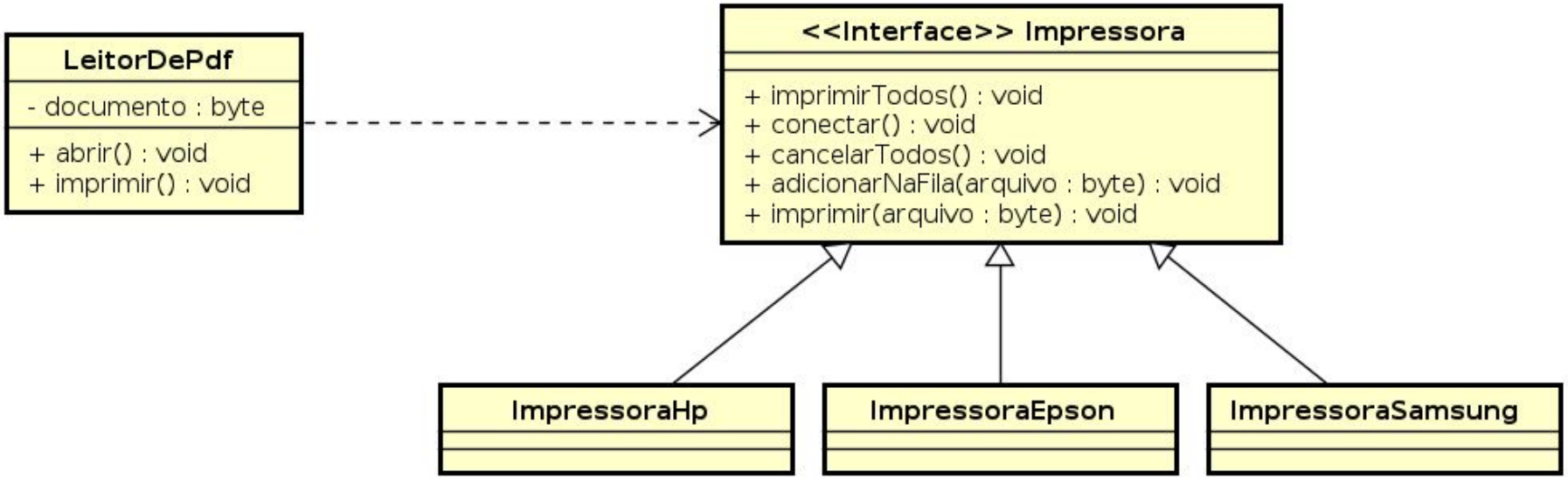
Dinâmica

A “ligação” entre objetos acontece em tempo de execução (Obrigado, polimorfismo)

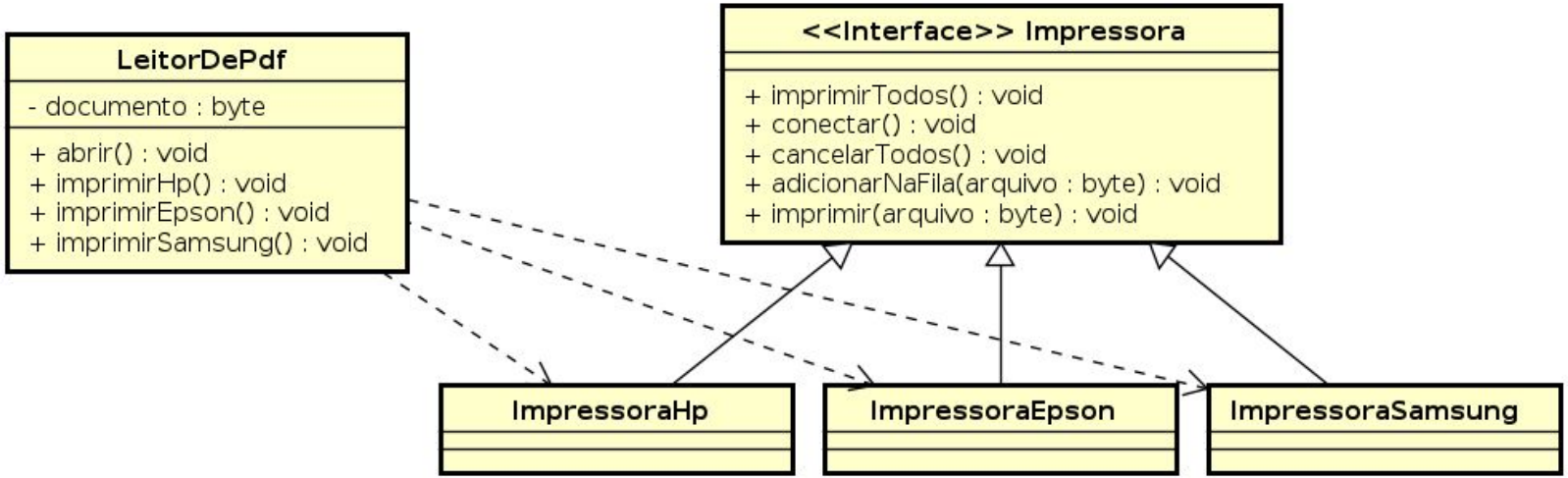
Estática

Acontece em tempo de compilação e é menos flexível.

Interfaces


LeitorDePdf
- documento : byte
+ abrir() : void
+ imprimir() : void
<<Interface>> Impressora
+ imprimirTodos() : void
+ conectar() : void
+ cancelarTodos() : void
+ adicionarNaFila(arquivo : byte) : void
+ imprimir(arquivo : byte) : void
ImpressoraHp
ImpressoraEpson
ImpressoraSamsung

Ligações (biding)




Let's go to the code!

Um pouco mais de conceitos

- Você deve programar para interfaces, não para implementações específicas;
- Utilizando herança, você pode definir a família de objetos compartilhando interfaces idênticas;
- Os clientes (código-fonte do cliente) desconhecem o tipo de objetos que eles usam.

Herança, composição e delegação

Is he your son?

Tem diferença? (lousa)

Herança

Indica o **é um** e é realizada em tempo de compilação. Indica que uma classe “herda” atributos, comportamentos e relacionamentos dos pais.

Composição

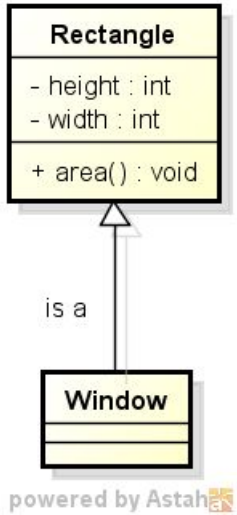
Indica o **tem um*** e é realizado em tempo de execução. Qualquer objeto pode ser trocado por outro em tempo de execução.

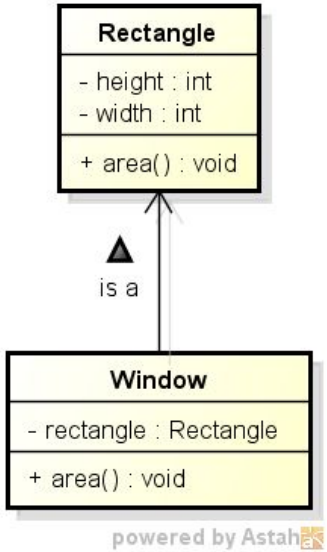
Delegação

Faz com que a composição seja tão poderosa quanto a herança. Dois objetos estão envolvidos a fim de manipular um requisição.

Delegação

- Perceba diferença






Let's go to the code!

2

Padrões criacionais

created!

Padrões criacionais

- Os padrões de criação (ou criacionais) abstraem o processo de instanciação de objetos em uma aplicação;

- Eles ajudam a tornar um sistema independente de como seus objetos são criados, compostos e representados;

Padrões criacionais

- Há duas características comuns nesses padrões:
  - Todos encapsulam conhecimento sobre quais classes concretas são usadas pelo sistema; e
  - Ocultam o modo como as instâncias destas classes sãos criadas e compostas;
- Tudo que o sistema sabe sobre esses objetos é que suas classes são subclasses de classes abstratas do sistema.

2

Singleton

To be unique!

Singleton (criação)

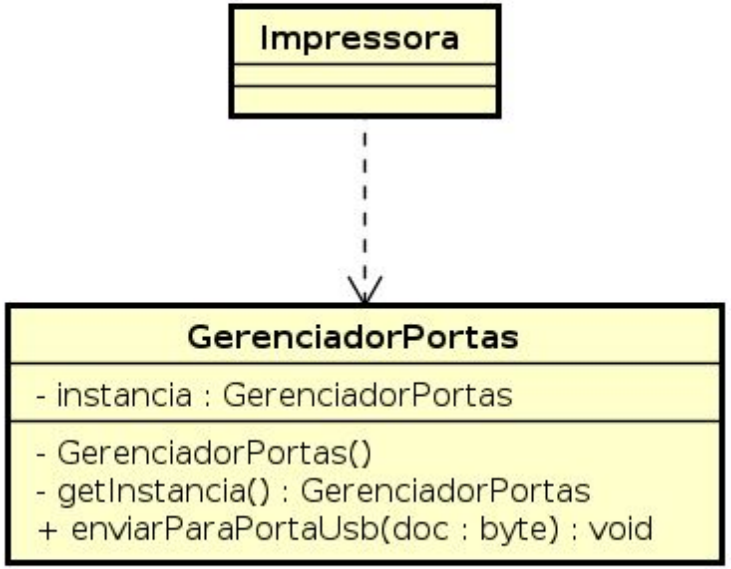
- Faz parte dos padrões criacionais e seu nome é oriundo do jogo de baralho Singleton, mais especificamente quando resta apenas uma carta na mão.

- Problema?
  - Imagine que você tenha que criar um software no qual deve existir apenas uma instância de uma classe;

Singleton

Intenção

Garantir que uma classe tenha apenas uma instância e forneça um ponto global de acesso a ela.




Let's go to the code!

Singleton

- **Cuidado:** Evite fazer cópias por todo código;
- Prefira injeção de dependências pois oferece mais flexibilidade;
- *Context* do **Android.**

2

Padrões comportamentais

Go go go go

Padrões comportamentais

- Se preocupam com algoritmos e a atribuição de responsabilidades entre objetos;

- Eles não descrevem apenas padrões para composição de objetos e classes, mas também padrões de comunicação entre objetos.

Padrões comportamentais

- A ideia é afastar o foco do fluxo de controle para permitir que você se concentre somente na maneira como os objetos são interconectados;

3

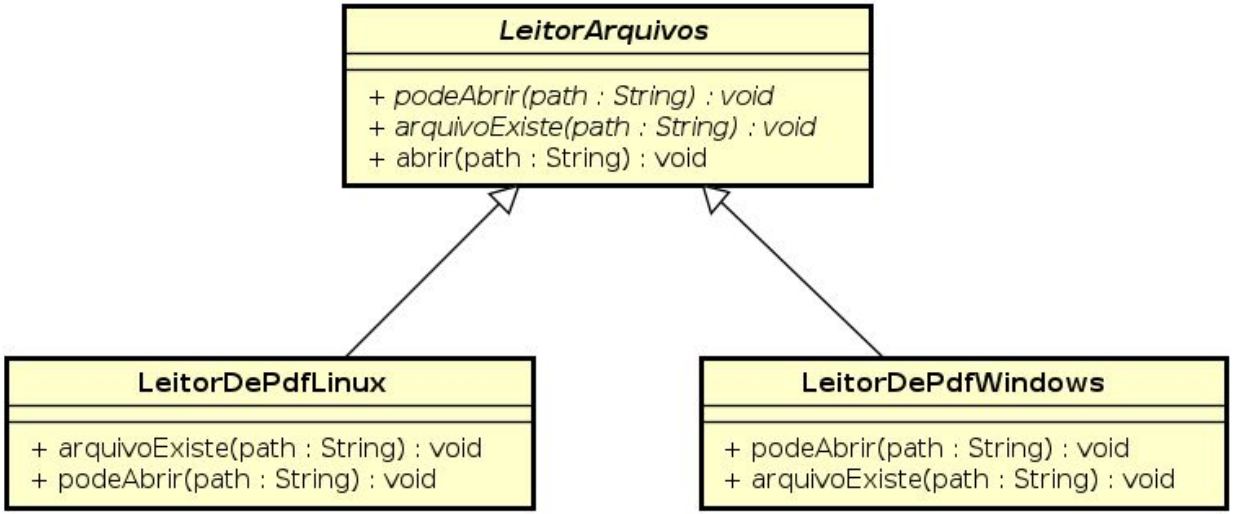
Template Method

Execute this!

Template Method (comportamental)

- **Intenção:** Definir um esqueleto de um algoritmo em uma operação, postergando alguns passos para as subclasses;
  - Define o esqueleto de um algoritmo dentro de um método;
  - Permite que as subclasses redefinam certos passos de um algoritmo sem alterar a estrutura do próprio algoritmo.

Template Method



Template Method

- Permite reutilizar código sem perder o controle do nossos algoritmos;
- Na classe abstrata temos métodos abstratos, concretos e finais;
- O Template Method é bastante utilizado inclusive na API Java (Java Swing);


Let's go to the code!

Thanks!

Any questions ?

You can find me at

- @warteruzannan
- wateruzannan.cunha@ufscar.br