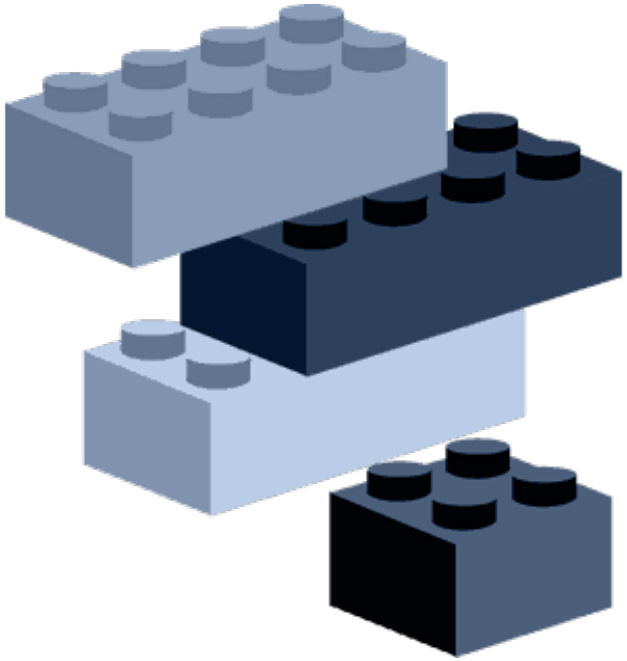
Design Patterns



INTRODUÇÃO

PRINCÍPIOS E DIAGRAMA DE CLASSES UML

UNIVAS

INTRODUÇÃO

Mais difícil do que projetar software orientado a objetos é projetar software reutilizável orientado a objetos.

UNIVAS

INTRODUÇÃO

Somam-se aos princípios dos quais nos inteiramos no encontro anterior outras recomendações e aprendizados que devem fazer parte da vida dos desenvolvedores.

UNIVAS

INTRODUÇÃO

Existem diferentes padrões de classes e comunicações com objetos que aparecem frequentemente em muitos projetos.

Eles resolvem problemas específicos e tornam o projeto mais flexível e reutilizável.

UNIVAS

INTRODUÇÃO

Quais os benefícios principais conferidos pelos padrões de projeto?

1. Não precisa reinventar a roda;
2. São universais;
3. Evita refatoração desnecessária;
4. Reutilização de código;
5. Abstrai e nomeia partes particulares do código;
6. Facilitam a criação de testes unitários.

Em outras palavras, ajudam o projetista a obter mais rapidamente um projeto adequado.

UNIVAS

DEFINIÇÃO

Efetivamente, portanto, padrões de projeto são descrições de objetos e classes comunicantes que precisam ser personalizadas para resolver um problema geral de projeto num contexto particular.

UNIVAS

O QUE ELES FAZEM?

1. Identifica classes e instâncias participantes, seus papeis, colaborações e a distribuição de responsabilidades;
2. Focaliza um problema ou tópico particular de projeto orientado a objetos;
3. Descreve em que situação pode ser aplicado

UNIVAS

PADRÕES DE PROJETO

Abstract Factory (95): Fornece uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.

Adapter (140): Converte a interface de uma classe em outra interface esperada pelos clientes. O Adapter permite que certas classes trabalhem em conjunto, pois de outra forma seria impossível por causa de suas interfaces incompatíveis.

Bridge (151): Separa uma abstração da sua implementação, de modo que as duas possam variar independentemente.

Builder (104): Separa a construção de um objeto complexo da sua representação, de modo que o mesmo processo de construção possa criar diferentes representações.

Chain of Responsibility (212): Evita o acoplamento do remetente de uma solicitação ao seu destinatário, dando a mais de um objeto a chance de tratar a solicitação. Encadeia os objetos receptores e passa a solicitação ao longo da cadeia até que um objeto a trate.

Command (222): Encapsula uma solicitação como um objeto, desta forma permitindo que você parametrize clientes com diferentes solicitações, enfileire ou registre (*log*) solicitações e suporte operações que podem ser desfeitas.

Composite (160): Compõe objetos em estrutura de árvore para representar hierarquias do tipo partes-todo. O Composite permite que os clientes tratem objetos individuais e composições de objetos de maneira uniforme.

Decorator (170): Atribui responsabilidades adicionais a um objeto dinamicamente. Os decorators fornecem uma alternativa flexível a subclasses para extensão da funcionalidade.

Façade (179): Fornece uma interface unificada para um conjunto de interfaces em um subsistema. O Façade define uma interface de nível mais alto que torna o subsistema mais fácil de usar.

Factory Method (112): Define uma interface para criar um objeto, mas deixa as subclasses decidirem qual classe a ser instanciada. O Factory Method permite a uma classe postergar (*defer*) a instanciação às subclasses.

Flyweight (187): Usa compartilhamento para suportar grandes quantidades de objetos, de granularidade fina, de maneira eficiente.

Interpreter (231): Dada uma linguagem, define uma representação para sua gramática juntamente com um interpretador que usa a representação para interpretar sentenças nessa linguagem.

Iterator (244): Fornece uma maneira de acessar seqüencialmente os elementos de uma agregação de objetos sem expor sua representação subjacente.

Mediator (257): Define um objeto que encapsula a forma como um conjunto de objetos interage. O Mediator promove o acoplamento fraco ao evitar que os objetos se refiram explicitamente uns aos outros, permitindo que você varie suas interações independentemente.

Memento (266): Sem violar o encapsulamento, captura e externaliza um estado interno de um objeto, de modo que o mesmo possa posteriormente ser restaurado para este estado.

Observer (274): Define uma dependência um-para-muitos entre objetos, de modo que, quando um objeto muda de estado, todos os seus dependentes são automaticamente notificados e atualizados.

Prototype (121): Especifica os tipos de objetos a serem criados usando uma instância prototípica e criar novos objetos copiando esse protótipo.

Proxy (198): Fornece um objeto representante (*surrogate*), ou um marcador de outro objeto, para controlar o acesso ao mesmo.

Singleton (130): Garante que uma classe tenha somente uma instância e fornece um ponto global de acesso para ela.

State (284): Permite que um objeto altere seu comportamento quando seu estado interno muda. O objeto parecerá ter mudado de classe.

Strategy (292): Define uma família de algoritmos, encapsula cada um deles e os torna intercambiáveis. O Strategy permite que o algoritmo varie independentemente dos clientes que o utilizam.

Template Method (301): Define o esqueleto de um algoritmo em uma operação, postergando a definição de alguns passos para subclasses. O Template Method permite que as subclasses redefinam certos passos de um algoritmo sem mudar sua estrutura.

Visitor (305): Representa uma operação a ser executada sobre os elementos da estrutura de um objeto. O Visitor permite que você defina uma nova operação sem mudar as classes dos elementos sobre os quais opera.

UNIVAS

ESCOPO

		Propósito		
		De criação	**Estrutural**	**Comportamental**
Escopo	**Classe**	Factory Method (112)	Adapter (class) (140)	Interpreter (231) Template Method (301)
	Objeto	Abstract Factory (95) Builder (104) Prototype (121) Singleton (130)	Adapter (object) (140) Bridge (151) Composite (160) Decorator (170) Façade (179) Flyweight (187) Proxy (198)	Chain of Responsibility (212) Command (222) Iterator (244) Mediator (257) Memento (266) Observer (274) State (284) Strategy (292) Visitor (305)

UNIVAS

PEQUENA REVISÃO SOBRE UML

Considerando o diagrama de classes, qual é a estrutura que representa uma classe?

NomeDaClasse
atributos
métodos

UNIVAS

PEQUENA REVISÃO SOBRE UML

Do ponto de vista da composição de um objeto, qual o papel do atributo?

E do método?

Como se define um atributo e um método no diagrama?

UNIVAS

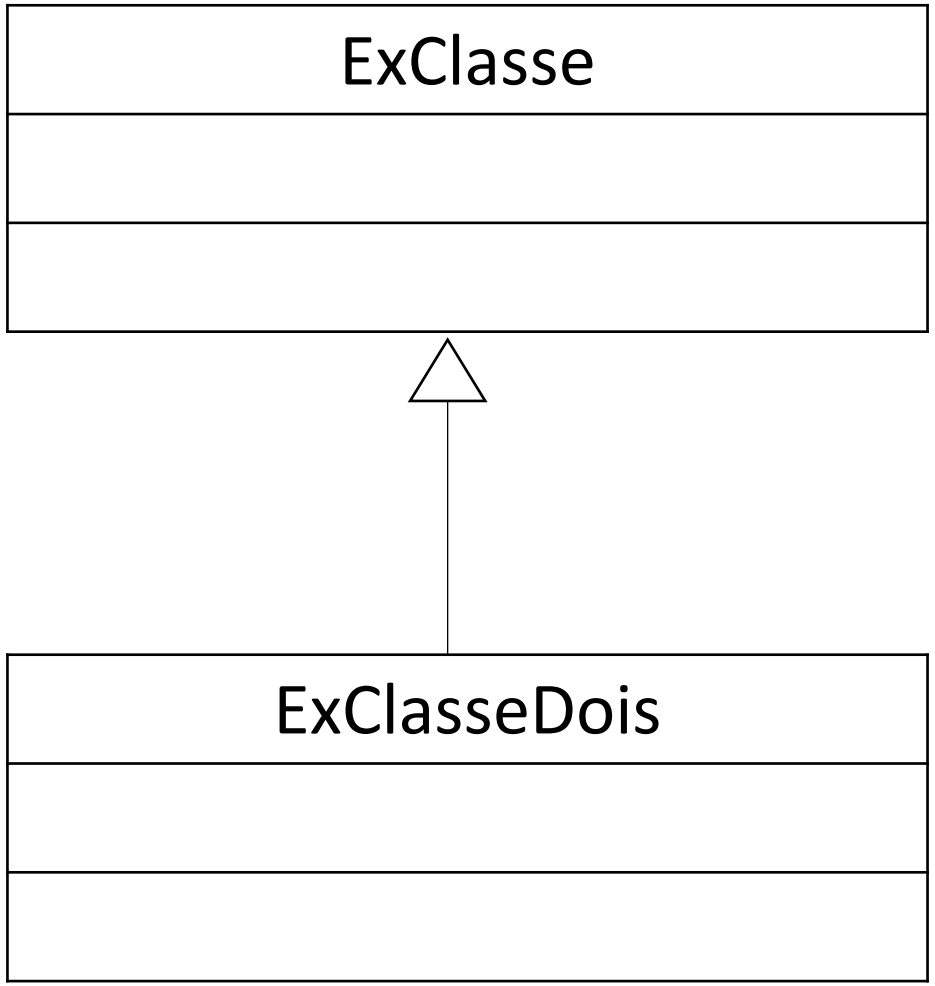
ExClasse
+ nome : string - cpf : inteiro # email: string
+ getNome() : string + getCpf() : inteiro + setNome(nome : string) + setCpf(cpf : inteiro)

UNIVAS

PEQUENA REVISÃO SOBRE UML

No contexto da programação, o que seria herança?

Como representa-la no diagrama de classes?

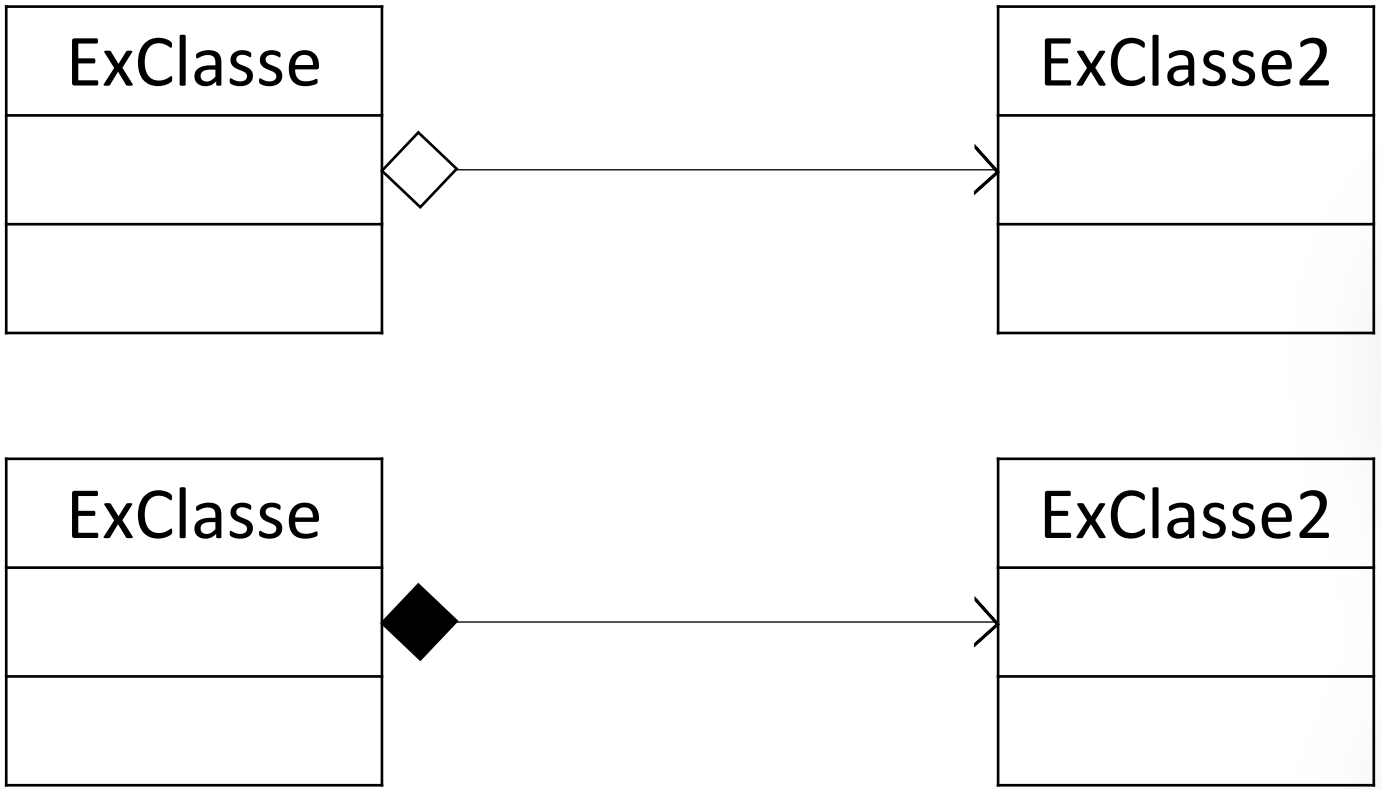

ExClasse
ExClasseDois

UNIVAS

PEQUENA REVISÃO SOBRE UML

No contexto da programação, o que seria agregação?

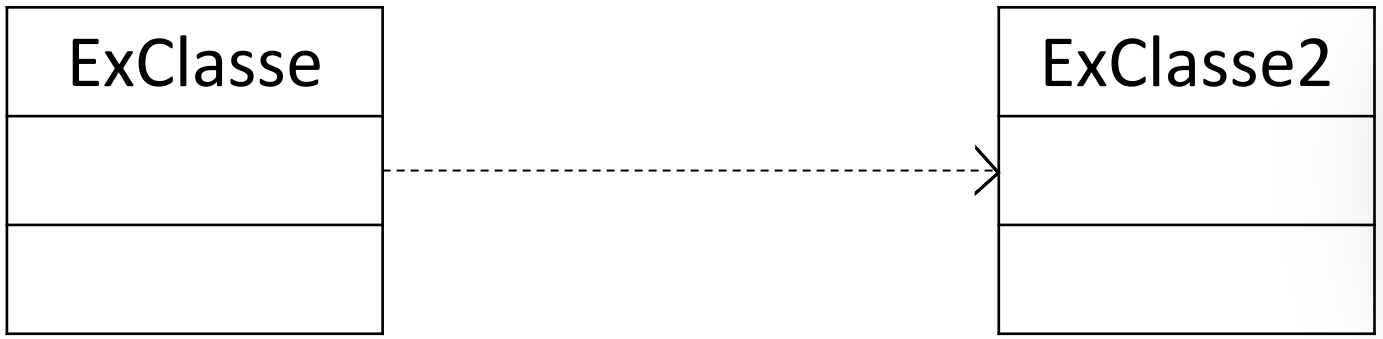
E composição?


ExClasse
ExClasse2
ExClasse
ExClasse2

UNIVAS

PEQUENA REVISÃO SOBRE UML

E a dependência?


ExClasse
ExClasse2

UNIVAS

Obrigado!

Professor Gustavo Dias
luizdias@univas.edu.br