VIANNSTITUTO JUNIOR



Análise OO

Modelo Orientado a Objetos

Professor: Camillo Falcão



Modelo orientado a objetos

- Para um modelo orientado a objetos, o framework conceitual é o modelo de objetos. Existem quatro elementos principais desse modelo:
 - Abstração;
 - Encapsulamento;
 - Modularidade;
 - Hierarquia.



Modelo orientado a objetos

- Para o modelo de objetos, existem três elementos secundários:
 - Tipificação;
 - Concorrência;
 - Persistência.
- Esses elementos são partes úteis, mas não essenciais, do modelo de objetos.



Abstração

"Abstração surge de um reconhecimento de similaridades entre certos objetos, situações ou processos do mundo real, e a decisão de concentrar-se sobre essas similaridades e ignorar as diferenças"

Dahl, Dijkstra e Hoare



Abstração

 Uma boa abstração enfatiza detalhes que possuem significado para o usuário e suprime outros detalhes.

 Uma abstração foca na visão de fora do objeto e serve para separar o comportamento essencial do objeto de sua implementação.



Abstração





Abstração - Contratos

- A visão de fora de cada objeto define um contrato no qual outros objetos podem depender.
 - Esse contrato deve ser cumprido pela visão interna do próprio objeto (frequentemente em colaboração com outros objetos).
 - Este contrato estabelece todos os pressupostos que um objeto cliente pode fazer sobre o comportamento de um objeto servidor. Em outras palavras, este contrato abrange as responsabilidades de um objeto, ou seja, o comportamento para o qual foi responsabilizado.

(Objeto cliente é um objeto que utiliza recursos de outro objeto conhecido como servidor)



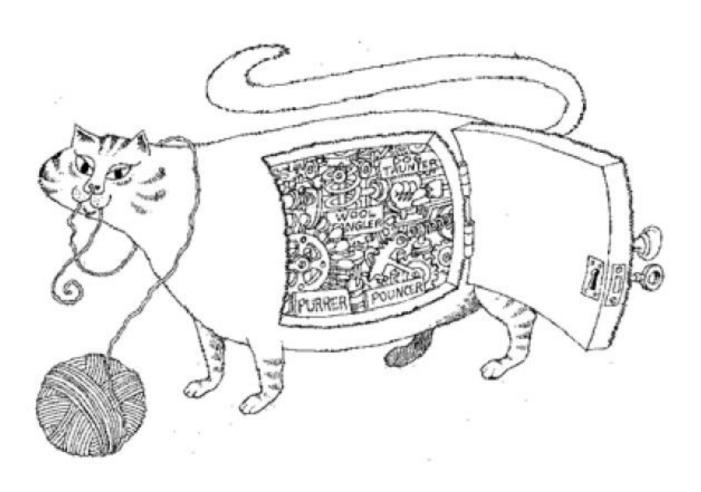
Encapsulamento

 A abstração de um objeto deve preceder a sua implementação.

 Uma vez que uma implementação é selecionada, ela deve ser tratada como um segredo da abstração e deve ser escondida da maioria dos objetos clientes.



Encapsulamento





Encapsulamento

- Conceitos complementares:
 - Abstração foca no comportamento observável de um objeto.
 - Encapsulamento foca na implementação que causa esse comportamento.



Modularidade

- O ato de particionamento de um programa em componentes individuais pode reduzir a sua complexidade em algum grau.
- O particionamento de um programa cria um número de fronteiras bem definidas e documentadas dentro do programa.
 - Essas fronteiras ou interfaces são de valor inestimável para a compreensão do programa.



Modularidade





Modularidade

- Módulos tem a função de contêineres físicos onde são declaradas as classes do projeto lógico.
 - Classes correlacionadas são agrupadas em um mesmo módulo, que expõe somente aqueles elementos que outros módulos absolutamente precisam acessar.
 - Problema: se o número de módulos for muito grande, o usuário pode ter dificuldade em encontrar as classes que ele precisa.
- Considerando que a solução pode não ser conhecida quando é iniciada a etapa de projeto, a decomposição em módulos menores pode ser bastante difícil.



Hierarquia

- Exceto para as aplicações mais triviais, nós podemos encontrar muito mais abstrações do que podemos compreender em um dado momento.
- O encapsulamento nos auxilia a gerenciar essa complexidade ao ocultar a visão interna de nossas abstrações.
- A modularidade nos permite agrupar logicamente abstrações relacionadas.
- Mas isso não é suficiente!



Hierarquia

- Um conjunto de abstrações frequentemente formam uma hierarquia.
- Identificar essas hierarquias durante o projeto pode simplificar o entendimento do problema.

- Definição de hierarquia:
 - "Hierarquia é um ranking de ordenação de abstrações"





- Exemplos de hierarquias:
 - Herança Simples;
 - Herança Múltipla;
 - Agregação.





 Herança define um relacionamento entre classes onde uma classe compartilha a estrutura ou comportamento definida em uma ou mais classes.

- A herança representa uma hierarquia de abstrações na qual uma subclasse herda de uma ou mais superclasses.
- Geralmente uma subclasse aumenta ou redefine a estrutura e comportamento de suas superclasses.





- Uma herança denota um relacionamento do tipo "é um". Exemplos:
 - Um urso "é um" tipo de mamífero;
 - Uma casa "é um" tipo de ativo tangível;
 - Um quicksort "é um" tipo particular de algoritmos de ordenação.
- Herança então implica em uma hierarquia generalização/especialização, onde uma subclasse especializa a estrutura ou comportamento mais geral de sua superclasse.



- Teste para herança:
 - Se B não é uma espécie de A, então B não pode herdar de A.



Herança simples

 Herança simples é quando uma classe compartilha a estrutura ou comportamento definida em apenas uma classe.

 A herança simples é um elemento essencial aos sistemas orientados a objetos.



Herança múltipla

 Herança múltipla é quando uma classe compartilha a estrutura ou comportamento definida em mais de uma classe.

 Uma herança múltipla pode ser reduzida a para uma herança simples com agregação de outras classes pela subclasse.



Agregação

 Enquanto hierarquias do tipo "é um" denota relacionamentos de generalização/especialização, hierarquias do tipo "parte de" descrevem relacionamentos de agregação.





• Exemplo:

- Um jardim consiste em uma coleção de plantas e um plano de cultivo.
- Em outras palavras: plantas são "parte de" um jardim e plano de cultivo é "parte de" um jardim.
- Os relacionamentos descritos no exemplo acima são conhecidos como agregações.





 A agregação permite o agrupamento físico de estruturas logicamente relacionadas.

 A herança permite a esses agrupamentos comuns serem facilmente reutilizados entre diferentes abstrações.



Agregações

- Questões de propriedade:
 - A abstração do jardim permite que diferentes plantas sejam utilizadas durante a existência do jardim.
 - Alterar uma planta no jardim não muda a identidade do jardim como um todo.
 - Remover o jardim também não necessariamente destrói todas as suas plantas (as mesmas podem ser transplantadas).
 - Existe uma independência entre a vida útil do jardim e de suas plantas.
 - O objeto do plano de cultivo é intrinsecamente relacionado com o objeto jardim e não existe independentemente.
 - Ao se criar uma instância de jardim, cria-se automaticamente uma instância do plano de cultivo.
 - Quando uma instância do objeto jardim é destruída, a instância de seu objeto plano de cultivo também é destruída.





"Um tipo é uma caracterização precisa das propriedades estruturais ou comportamentais que uma coleção de entidades compartilham."

Deutsch





- Em nosso curso, utilizaremos os termos tipo e classe como sendo sinônimos, porém um tipo e uma classe não são exatamente a mesma coisa.
 - Separar os conceitos de tipos e classes pode ser confuso para um curso introdutório e adiciona pouco valor.

Uma classe implementa um tipo.



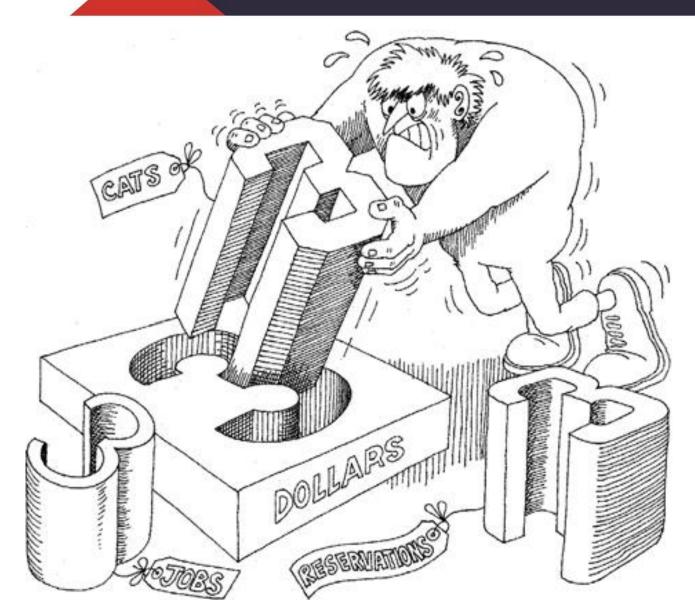


 Objetos de diferentes tipos não podem ser trocados, exceto em algumas formas muito restritas.

- Linguagens de programação orientadas a objetos podem ser:
 - Fortemente tipificadas;
 - Fracamente tipificadas;
 - Não tipificadas.



Tipificação forte impede a mistura de abstrações







- Vantagens da tipificação forte:
 - Sem verificação de tipo, um programa pode 'quebrar'
 de forma misteriosa durante a sua execução.
 - Na maioria dos sistemas, o ciclo de ediçãocompilação-depuração é tedioso, o que torna indispensável a detecção antecipada de erros.
 - As declarações de tipo ajudam a documentar programas.
 - A maioria dos compiladores pode gerar código de máquina mais eficiente se os tipos são declarados.



Tipificação

- Tipificação estática e tipificação dinâmica referem-se ao momento em que os nomes são ligados a seus tipos.
- Em inglês, tipificação estática é conhecida como:
 - Static typing;
 - Static binding;
 - Early binding.
- Em inglês, tipificação dinâmica é conhecida como:
 - Dynamic typing;
 - Late binding.



Tipificação

- Tipificação estática: os tipos de todas as variáveis e expressões são fixados em tempo de compilação.
- Tipificação dinâmica: os tipos de todas as variáveis e expressões não são conhecidos até a execução do programa.
- Exemplos de linguagens OO:
 - Fortemente e estaticamente tipificadas: Ada;
 - Fortemente tipificada com suporte de tipificação dinâmica: C++, C#, Java.
 - Não tipificada e com suporte de tipificação dinâmica:
 Smalltalk.





Polimorfismo:

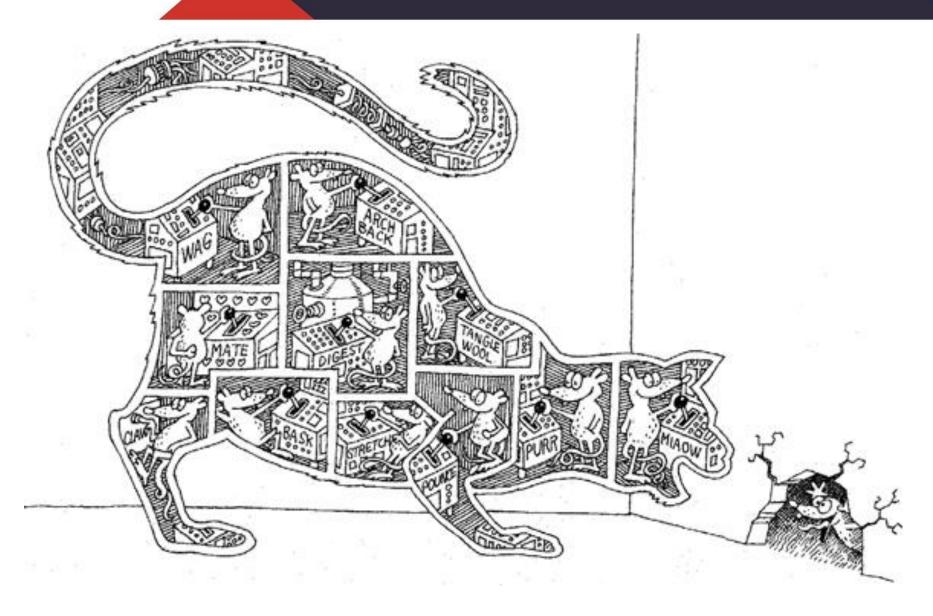
- Condição que existe quando as características de tipificação dinâmica e herança interagem.
- Representa um conceito no qual um nome simples, como uma declaração de variável, pode denotar objetos de diferentes classes que são interrelacionadas por uma superclasse comum.
- Qualquer objeto de qualquer subclasse pode ser utilizado no lugar de sua superclasse. Logo, pode-se existir diferentes subclasses que herdam de uma mesma superclasse. Como qualquer uma dessas subclasses pode ser utilizada no lugar de sua superclasse, o programa pode apresentar comportamentos diferentes, dependendo da subclasse utilizada.



 Para certos tipos de problemas, sistemas automatizados podem ter que lidar com eventos diferentes simultaneamente.

 Outros tipos de problemas podem envolver tantos cálculos que estes excedem a capacidade de um único processador.







 Enquanto a programação orientada a objetos foca em abstração de dados, encapsulamento e herança, a concorrência foca em abstrações de processos e sincronização.

 O objeto é o conceito que unifica esses dois diferentes pontos de visão: cada objeto (delineado à partir de uma abstração de um mundo real) pode representar uma thread de controle separada (uma abstração de processo). Esses objetos são chamados objetos ativos.



- Em um sistema baseado em um projeto orientado a objetos, pode-se conceituar o mundo como consistindo em um conjunto de objetos cooperativos, sendo que alguns desses objetos são ativos, servindo como centro de atividade independente.
- Assim, concorrência pode ser definida como:
 - Concorrência é a propriedade que distingue um objeto ativo de um que não é ativo.



Persistência

- Persistência é a propriedade de um objeto na qual sua existência transcende:
 - O tempo, ou seja, o objeto continua a existir mesmo após o seu criador deixar de existir.
 - O espaço, ou seja, a localização do objeto se move à partir do espaço de endereço em que foi criado.



Persistência

- Principais formas de persistência de objetos:
 - Bases de dados orientadas a objetos.
 - Representação em bases de dados relacionais.
- Em bases de dados relacionais, o mapeamento objeto relacional pode ser:
 - Customizado por um desenvolvedor;
 - Facilitado por um framework objeto-relacional.