Modelagem UML

Prof. Dr. Rodrigo Piva

Modelagem UML

- A modelagem orientada a objetos (OOM) é um desenvolvimento relativamente recente em engenharia de software. Surgiu na década de 1980 como uma resposta às limitações da programação processual, que era o paradigma de programação dominante na época.
- As origens da OOM podem ser rastreadas até o trabalho de vários pesquisadores no campo da ciência da computação. Um dos primeiros pioneiros da OOM foi Kristen Nygard, um cientista da computação norueguês que co-inventou a linguagem de programação da Simula na década de 1960. Simula foi a primeira linguagem de programação para suportar conceitos de programação orientados a objetos, como herança, encapsulamento e polimorfismo.

Modelagem UML

Na década de 1980, foram desenvolvidas várias novas linguagens de programação orientadas a objetos, incluindo Smalltalk, C ++ e Objective-C. Esses idiomas ajudaram a popularizar o uso do OOM no desenvolvimento de software e levaram ao desenvolvimento de muitas novas metodologias e técnicas de OOM.

Uma das metodologias de OOM mais influentes é a linguagem de modelagem unificada (UML), desenvolvida na década de 1990 por um grupo de cientistas da computação, incluindo Grady Booch, James Rumbaugh e Ivar Jacobson. A UML é uma notação gráfica padronizada para modelar sistemas orientados a objetos e se tornou o padrão de fato da OOM em muitas organizações de desenvolvimento de software.

Modelagem UML

Hoje, a OOM é uma abordagem amplamente usada para o desenvolvimento de software e é suportada por uma ampla gama de linguagens, ferramentas e estruturas de programação. Ele continua a evoluir à medida que surgem novas tecnologias e paradigmas de programação e continua sendo uma área importante de pesquisa e desenvolvimento na ciência da computação.

Modelagem UML

A programação orientada a objetos (OOP) é um paradigma de programação baseado na idéia de objetos, que pode conter dados e código para manipular esses dados. A modelagem orientada a objetos (OOM) é o processo de projetar os objetos e seus relacionamentos em um sistema orientado a objetos. Existem vários princípios no OOM que ajudam a orientar o processo de modelagem. Aqui estão alguns dos principais:

Modelagem UML

- Abstração: refere-se ao processo de simplificar conceitos complexos do mundo real em representações mais simples e gerenciáveis no modelo de software. A abstração ajuda a reduzir a complexidade de um sistema, facilitando a compreensão, a manutenção e a extensão.
- Encapsulamento: essa é a idéia de agrupar dados e comportamento em um objeto e restringir o acesso a esses dados por meio de interfaces bem definidas. O encapsulamento ajuda a garantir que os objetos permaneçam internamente consistentes e protejam seu estado interno de corromper.
- Herança: Esta é a idéia de criar uma nova classe que herda as propriedades e métodos de uma classe existente e depois estendê -las ou modificá -las conforme necessário. A herança ajuda a promover a reutilização do código e permite que os desenvolvedores criem classes mais especializadas a partir de mais gerais.
- Polimorfismo: essa é a idéia de fornecer uma única interface que pode ser implementada por várias classes de maneiras diferentes. O polimorfismo ajuda a tornar o código mais flexível e extensível e permite maior reutilização de código.
- Composição: refere -se ao processo de combinar objetos mais simples para criar os mais complexos. A composição é frequentemente usada quando um objeto precisa usar a funcionalidade de outro objeto, mas não precisa herdar todas as suas propriedades e métodos.
- Seguindo esses princípios, os desenvolvedores podem criar sistemas de software mais fáceis de entender, manter e estender com o tempo.

Modelagem UML

A modelagem orientada a objetos (OOM) é o processo de projetar sistemas de software usando princípios de programação orientados a objetos. Envolve a identificação dos objetos e seus relacionamentos dentro de um sistema e, em seguida, definindo as propriedades, métodos e comportamentos desses objetos de uma maneira que reflita o domínio do mundo real que está sendo modelado. A OOM é frequentemente usada no desenvolvimento de software para criar sistemas de software modulares e flexíveis que são fáceis de entender, manter e estender com o tempo.

Modelagem UML

Durante o processo OOM, os desenvolvedores normalmente usam diagramas e outras técnicas de modelagem para representar os objetos e seus relacionamentos dentro de um sistema. Esses diagramas podem ajudar a identificar os principais objetos no sistema, suas propriedades e comportamentos e como eles interagem entre si. Depois que os objetos e seus relacionamentos forem identificados, os desenvolvedores podem começar a escrever código para implementar esses objetos e seus comportamentos no sistema de software real.

Modelagem UML

O objetivo da OOM é criar um sistema de software que modela o domínio do mundo real que está sendo representado de uma maneira precisa, fácil de entender e flexível o suficiente para acomodar mudanças ao longo do tempo. Ao usar os princípios e técnicas de programação orientados a objetos, os desenvolvedores podem criar sistemas de software mais fáceis de manter, mais reutilizáveis e mais extensíveis do que as abordagens tradicionais de programação processual.

Modelagem UML

A implementação da modelagem orientada a objetos (OOM) envolve várias etapas. Aqui está uma visão geral de alto nível do processo:

Identifique os objetos: a primeira etapa da OOM é identificar os objetos que serão usados no sistema de software que está sendo desenvolvido. Os objetos podem ser qualquer coisa, desde objetos físicos no mundo real até conceitos abstratos, como eventos, transações ou relacionamentos.

Defina as propriedades do objeto: Depois que os objetos forem identificados, a próxima etapa é definir suas propriedades. Propriedades do objeto são os atributos que descrevem o objeto, como seu tamanho, cor, forma ou valor.

Modelagem UML

Defina os métodos de objeto: Além das propriedades, os objetos também têm comportamentos ou métodos que descrevem como eles interagem com outros objetos. Os métodos podem ser considerados como as ações que um objeto pode executar, como mover, calcular ou exibir informações.

Defina as relações de objetos: os objetos raramente existem isoladamente e geralmente têm relacionamentos com outros objetos no sistema. Esses relacionamentos podem ser individuais, um para muitos ou muitos para muitos, e podem ser modelados usando várias técnicas como agregação, composição ou herança.

Modelagem UML

Crie diagramas de classe: Uma vez que os objetos, propriedades, métodos e relacionamentos fossem identificados, os desenvolvedores podem criar diagramas de classe para representar os objetos e seus relacionamentos graficamente. Os diagramas de classe são uma maneira padronizada de representar a estrutura de um sistema orientado a objetos e podem ser usados para comunicar o design do sistema a outros desenvolvedores.

Implementar os objetos e as classes: Depois que os diagramas de classe foram criados, os desenvolvedores podem começar a implementar os objetos e classes no sistema de software. Isso normalmente envolve a gravação de código em uma linguagem de programação orientada a objetos, como Java, Python ou C ++.

Modelagem UML

Teste e refina: À medida que o sistema está sendo desenvolvido, é importante testá -lo para garantir que ele esteja funcionando corretamente. Os desenvolvedores podem refinar o design do sistema, conforme necessário, com base no feedback dos testes e feedback do usuário.

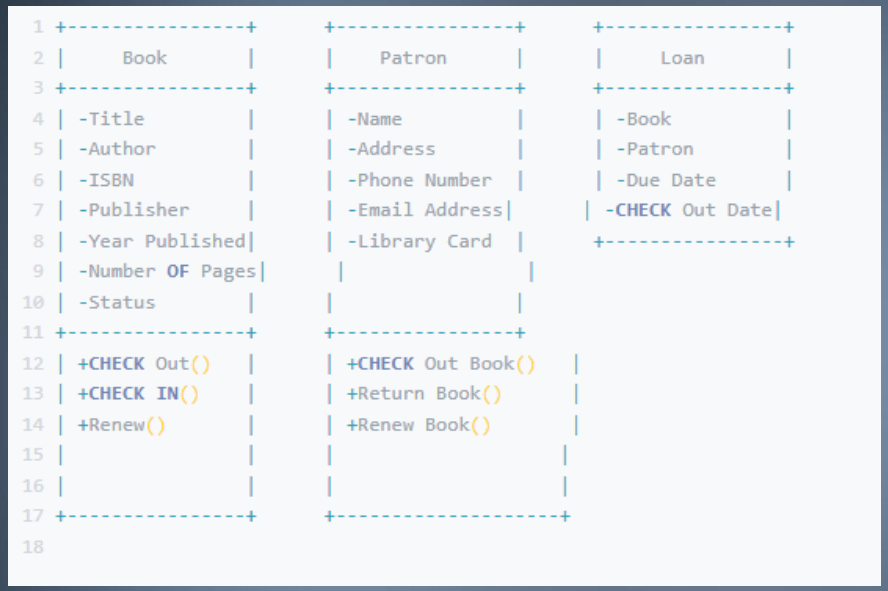
Seguindo essas etapas, os desenvolvedores podem criar sistemas de software mais modulares, flexíveis e reutilizáveis do que as abordagens tradicionais de programação processual. A OOM ajuda a simplificar o design e o desenvolvimento de sistemas complexos, dividindo -os em objetos menores e mais gerenciáveis que podem ser facilmente entendidos e mantidos ao longo do tempo.

Modelagem UML

- Digamos que queremos modelar um sistema de biblioteca simples usando os princípios de programação orientados a objetos. Podemos começar identificando os principais objetos no sistema:
- Livro: Um livro físico ou eletrônico na coleção da biblioteca.
- Patrono: um patrono da biblioteca que pode emprestar livros.
- Empréstimo: um registro de um livro que está sendo verificado por um patrono.
- Em seguida, definiríamos as propriedades e métodos para cada um desses objetos:
- Livro:
- Propriedades: Título, autor, ISBN, editor, ano publicado, número de páginas, status (check -out ou disponível)
- Métodos: confira, check -in, renovar
- Patrono:
- Propriedades: nome, endereço, número de telefone, endereço de e -mail, número do cartão da biblioteca
- Métodos: Confira o livro, o livro de retorno, o livro de renovação
- Empréstimo:
- Propriedades: livro, patrono, data de vencimento, data de conferência
- Métodos: Nenhum
- Em seguida, definiríamos as relações entre esses objetos:
- Um livro pode ser verificado por zero ou mais clientes.
- Um consumidor pode conferir zero ou mais livros.
- Um empréstimo registra a relação entre um livro e um patrono quando um livro é verificado.

Modelagem UML

Por fim, criaríamos diagramas de classe para representar esses objetos e seus relacionamentos graficamente. Aqui está um exemplo de como o diagrama de classes pode ser:



Este diagrama de classes mostra as relações entre os objetos de livro, patrono e empréstimo. Podemos ver que o objeto do livro possui propriedades como título, autor, ISBN e status, além de métodos como check -out, check -in e renovação. Da mesma forma, o objeto Patrono possui propriedades como nome, endereço e número do cartão da biblioteca, além de métodos como check -out Book, Return Book e Renow Book. Finalmente, o objeto de empréstimo registra o relacionamento entre um livro e um patrono quando um livro é verificado.

Modelagem UML

O exemplo do sistema da biblioteca que forneci pode ser modelado em várias linguagens de programação que suportam princípios de programação orientados a objetos, como Java, Python ou C ++. Aqui está um exemplo de implementação da aula de livros em Python:

main.py

```python
class Book:
    def __init__(self, title, author, isbn, publisher, year_published, num_pages):
        self.title = title
        self.author = author
        self.isbn = isbn
        self.publisher = publisher
        self.year_published = year_published
        self.num_pages = num_pages
        self.status = "available"

    def check_out(self):
        if self.status == "available":
            self.status = "checked out"
            return True
        else:
            return False

    def check_in(self):
        if self.status == "checked out":
            self.status = "available"
            return True
        else:
            return False

    def renew(self):
        if self.status == "checked out":
            return True
        else:
            return False
```

Modelagem UML

Esta implementação define uma aula de livros com propriedades como título, autor e ISBN, além de métodos como check_out, check_in e renovação. O método check_out altera o status do livro de "disponível" para "check -out", enquanto o método check_in altera o status de "check -out" para "disponível". O método de renovação permite que o livro seja verificado para um período de empréstimo adicional sem precisar ser devolvido à biblioteca.

Implementações semelhantes podem ser criadas para as classes de patrono e empréstimo. Depois que todas as classes forem implementadas, o sistema pode ser testado e refinado para garantir que ele esteja funcionando corretamente.

Modelagem UML

Digamos que queremos modelar um sistema bancário simples usando os princípios de programação orientados a objetos. Podemos começar identificando os principais objetos no sistema:

Conta: uma conta bancária pertencente a um cliente.

Cliente: um cliente bancário que possui uma ou mais contas.

Transação: um registro de uma transação envolvendo uma conta.

Em seguida, definiríamos as propriedades e métodos para cada um desses objetos:

Conta:

Propriedades: número da conta, cliente, saldo, taxa de juros

Métodos: depositar, retirar, calcular juros

Cliente:

Propriedades: nome, endereço, número de telefone, endereço de e -mail

Métodos: Abertura da conta, fechar conta, depósito, retirar

Transação:

Propriedades: conta, valor, tipo (depósito ou retirada), data

Métodos: Nenhum

Em seguida, definiríamos as relações entre esses objetos:

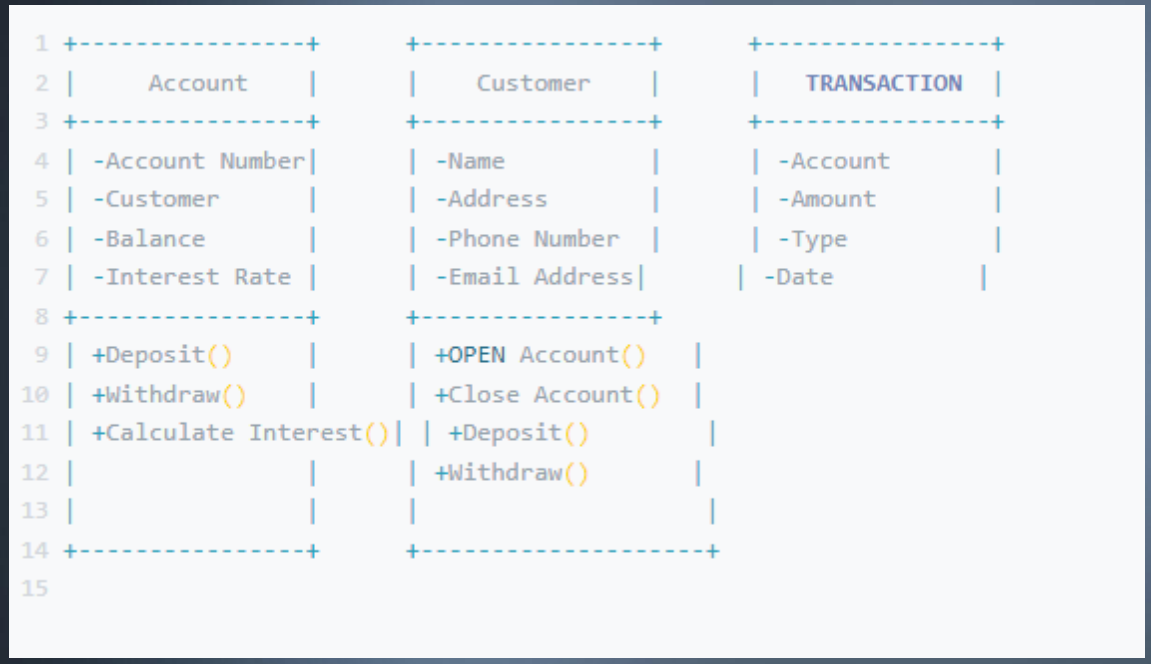
Uma conta é de propriedade de um cliente.

Um cliente pode possuir zero ou mais contas.

Uma transação registra a relação entre uma conta e uma quantia de dinheiro sendo depositada ou retirada.

Modelagem UML

Por fim, criaríamos diagramas de classe para representar esses objetos e seus relacionamentos graficamente. Aqui está um exemplo de como o diagrama de classes pode ser:



Este diagrama de classes mostra as relações entre os objetos de conta, cliente e transação. Podemos ver que o objeto da conta possui propriedades como número da conta, cliente e saldo, além de métodos como depósito, retirada e calcular juros. Da mesma forma, o objeto do cliente possui propriedades como nome, endereço e endereço de email, além de métodos como conta aberta, conta fechada, depósito e retirada. Finalmente, o objeto de transação registra a relação entre uma conta e uma quantia de dinheiro sendo depositada ou retirada.

Modelagem UML

Aqui está um exemplo de implementação da classe de conta em Python:

```python
class Account:
    def __init__(self, account_number, customer, balance=0, interest_rate=0.02):
        self.account_number = account_number
        self.customer = customer
        self.balance = balance
        self.interest_rate = interest_rate

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            return True
        else:
            return False

    def calculate_interest(self):
        interest = self.balance * self.interest_rate
        self.balance += interest
        return interest
```

Esta implementação define uma classe de conta com propriedades como account_number, cliente, saldo e interest_rate, além de métodos como depósito, retirada e calculate_interest. O método de depósito adiciona uma quantia especificada ao saldo da conta, enquanto o método de retirada subtraia uma quantia especificada do saldo da conta, desde que haja fundos suficientes na conta. O método calculate_intest calcula a quantidade de juros devidos ao saldo da conta e o adiciona ao saldo.

Da mesma forma, as classes de clientes e transações podem ser implementadas com suas próprias propriedades e métodos. Depois que todas as classes forem implementadas, o sistema pode ser testado e refinado para garantir que ele esteja funcionando corretamente.

Modelagem UML

Em conclusão, a modelagem orientada a objetos é uma abordagem poderosa para o design de software que ajuda os desenvolvedores a criar código mais modular, reutilizável e sustentável. Envolve dividir um sistema em um conjunto de objetos, definir as propriedades e métodos de cada objeto e especificar as relações entre eles. Com a modelagem orientada a objetos, os desenvolvedores podem criar sistemas de software mais fáceis de entender, testar e modificar com o tempo.

EXERCICIO 1

Rosicleide não tem mais tempo de fazer as compras pessoalmente.

Precisa detalhar o produto, de forma a permitir delegar essa tarefa a outra pessoa. Além disso, não quer que paguem um valor absurdo por algum produto.

Sendo assim, incluiu em sua planilha as colunas "preço máximo já comprado" e "preço máximo a pagar" no mês corrente, onde esta última coluna é calculada a partir da coluna anterior acrescida de 10%.

O "preço máximo já comprado" é inserido na planilha, a partir das compras efetivamente realizadas.

Quais são os atributos e/ou métodos que precisam ser incluídos nas classes do exercicio para refletir esse novo cenário.

EXERCICIO 2

Eriovaldo tem uma coleção grande de DVD´s e gostaria de cadastrar no seu Smartphone a lista desses DVD's, pois às vezes nem sabe o que tem. Ele pensou em cadastrar o nome do cantor{a) ou conjunto, o título do CD e o ano de lançamento.

Identifique as classes, atributos e métodos desse cenário. Represente os relacionamentos como atributos derivados.

Dúvidas???