Modelagem UML

Prof. Dr. Rodrigo Piva

Modelagem UML

- A Unified Modeling Language (UML) é uma linguagem visual padronizada usada para modelar sistemas de software. A modelagem conceitual é um aspecto fundamental da UML e envolve a criação de uma visão de alto nível do sistema que está sendo modelado. O objetivo da modelagem conceitual é fornecer uma visão geral clara e concisa do sistema, sem se prender aos detalhes da implementação.
- A modelagem conceitual em UML normalmente é feita usando diagramas de classes, que representam as classes, atributos e relacionamentos no sistema. Uma classe é um modelo ou projeto para criar objetos e contém atributos (dados) e métodos (comportamento). Os relacionamentos entre as classes podem ser expressos usando vários tipos de associações, como generalização (herança), agregação e composição.

Modelagem UML

- A UML também inclui outros tipos de diagramas que podem ser usados para modelagem conceitual, como diagramas de casos de uso, diagramas de atividades e diagramas de sequência. Os diagramas de caso de uso mostram as interações entre os atores (usuários ou outros sistemas) e o sistema que está sendo modelado e ajudam a identificar os requisitos do sistema. Os diagramas de atividades mostram o fluxo de atividades dentro do sistema e podem ser usados para modelar processos de negócios ou fluxos de trabalho do sistema. Os diagramas de sequência mostram as interações entre objetos no sistema e ajudam a identificar a sequência de mensagens e ações que ocorrem.
- Em resumo, a modelagem conceitual em UML é um aspecto fundamental do desenvolvimento de software e envolve a criação de visualizações de alto nível do sistema que está sendo modelado usando diagramas de classes, diagramas de casos de uso, diagramas de atividades e diagramas de sequência. Ao criar modelos claros e concisos do sistema, a UML pode ajudar a garantir que o sistema seja bem projetado, atenda aos requisitos do usuário e seja facilmente mantido ao longo do tempo.

Modelagem UML

- A visão geral da UML é fornecer uma linguagem visual padronizada para modelagem de sistemas de software que pode ser usada em diferentes domínios, indústrias e tecnologias. A UML foi desenvolvida com o objetivo de unificar as diferentes abordagens de modelagem de software existentes na época e criar uma linguagem comum que pudesse ser usada por desenvolvedores, analistas, designers e partes interessadas durante todo o ciclo de vida do desenvolvimento de software.
- A visão da UML é baseada em vários princípios-chave, incluindo:

Modelagem UML

1. Padronização: UML fornece uma notação padrão para modelagem de sistemas de software que é amplamente aceita na indústria de software. Isso ajuda a garantir consistência e interoperabilidade entre diferentes ferramentas e plataformas.
2. Abstração: a UML permite que os desenvolvedores criem modelos de sistemas de software em diferentes níveis de abstração, desde modelos conceituais de alto nível até modelos de implementação de baixo nível. Isso ajuda a gerenciar a complexidade e permite que os desenvolvedores se concentrem nos aspectos mais importantes do sistema.

Modelagem UML

1. Flexibilidade: UML é uma linguagem flexível que pode ser adaptada a diferentes domínios, indústrias e tecnologias. Ele fornece um rico conjunto de construções de modelagem que podem ser usadas para representar uma ampla variedade de sistemas de software, desde aplicativos de desktop simples até sistemas distribuídos complexos.
2. Comunicação: UML é uma linguagem visual que pode ser facilmente compreendida pelas partes interessadas que podem não ter formação técnica. Isso o torna uma ferramenta poderosa para comunicar o design e os requisitos de software para as partes interessadas não técnicas, como gerentes, clientes e usuários.

Modelagem UML

- No geral, a visão da UML é fornecer uma linguagem poderosa e flexível para modelagem de sistemas de software que possa ser usada por todas as partes interessadas envolvidas no ciclo de vida do desenvolvimento de software. Ao fornecer uma notação padronizada para modelagem de software, a UML ajuda a melhorar a comunicação, gerenciar a complexidade e garantir a qualidade dos sistemas de software.

Modelagem UML

- O conceito arquitetônico da UML é baseado na ideia de que os sistemas de software podem ser decompostos em componentes menores e mais gerenciáveis, que podem ser organizados em uma estrutura ou arquitetura lógica. A UML fornece vários diagramas e construções arquiteturais que podem ser usados para modelar e descrever a estrutura e o comportamento de sistemas de software.
- Um dos principais conceitos arquitetônicos da UML é o uso da arquitetura baseada em componentes. Nesta abordagem, os sistemas de software são decompostos em um conjunto de componentes independentes e reutilizáveis que podem ser combinados para formar sistemas maiores. Os componentes em UML podem ser modelados usando diagramas de componentes, que mostram os relacionamentos entre os componentes e as interfaces que eles fornecem.

Modelagem UML

- Outro importante conceito de arquitetura em UML é o uso de visualizações de modelagem. Uma visão é um subconjunto do modelo geral que se concentra em um aspecto ou preocupação particular do sistema que está sendo modelado. A UML fornece várias visualizações, como a visualização lógica, a visualização do processo e a visualização de implantação, que podem ser usadas para modelar diferentes aspectos do sistema.
- A UML também inclui vários padrões arquiteturais, que são soluções reutilizáveis para problemas arquitetônicos comuns. Esses padrões podem ser usados para projetar e implementar sistemas de software de maneira consistente e eficiente.

Modelagem UML

- No geral, o conceito arquitetônico da UML é focado na criação de um modelo claro e conciso da estrutura e comportamento dos sistemas de software, usando uma combinação de arquitetura baseada em componentes, visualizações de modelagem e padrões de arquitetura. Ao fornecer uma linguagem padronizada para modelagem de arquitetura, a UML ajuda a garantir que os sistemas de software sejam bem projetados, sustentáveis e escaláveis.

Modelagem UML

- Na UML, uma classe é um bloco de construção fundamental usado para modelar sistemas de software orientados a objetos. Uma classe é um modelo ou modelo para a criação de objetos e define os atributos (dados) e métodos (comportamento) dos objetos que são criados a partir dela. As classes são normalmente usadas para representar entidades ou conceitos do mundo real, como um cliente, um pedido ou um produto.
- Uma classe UML é representada usando um diagrama de classe, que é uma representação gráfica da classe e seus relacionamentos com outras classes no sistema. O diagrama de classes consiste em uma caixa retangular dividida em três seções: o nome da classe, os atributos (dados) da classe e os métodos (comportamento) da classe.

Modelagem UML

- O nome da classe é normalmente escrito em negrito e colocado no topo do retângulo. Os atributos da classe são listados na seção do meio do retângulo e normalmente são representados como pares nome-valor. Os atributos representam os dados ou o estado dos objetos criados a partir da classe. Por exemplo, uma classe de cliente pode ter atributos como nome do cliente, endereço e número de telefone.
- Os métodos da classe são listados na seção inferior do retângulo e representam o comportamento ou as ações que podem ser executadas nos objetos criados a partir da classe. Os métodos geralmente têm um nome e um conjunto de parâmetros e podem retornar um valor. Por exemplo, uma classe de cliente pode ter métodos como placeOrder(), cancelOrder() e updateProfile().

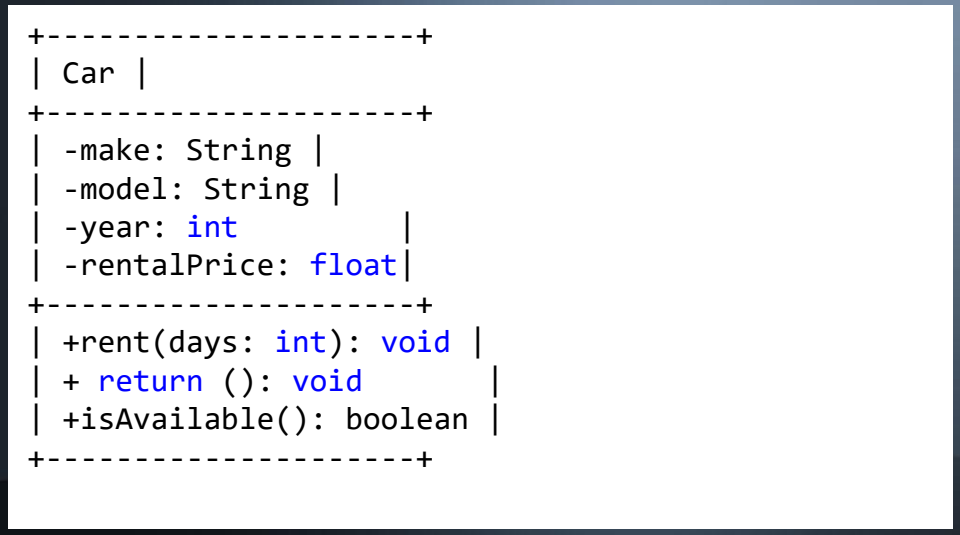
Modelagem UML

- Além dos atributos e métodos, as classes UML também podem ter relacionamentos com outras classes do sistema. Esses relacionamentos são representados usando vários tipos de associações, como generalização (herança), agregação e composição.
- No geral, uma classe UML é um elemento-chave do projeto de software orientado a objetos e fornece uma ferramenta poderosa para representar a estrutura e o comportamento dos sistemas de software. Ao usar classes UML para modelar entidades e conceitos em um sistema de software, os desenvolvedores podem criar designs claros e concisos que são facilmente compreendidos e mantidos ao longo do tempo.

Modelagem UML

- vamos considerar um exemplo simples de um diagrama de classes UML para um sistema de aluguel de carros. Neste exemplo, definiremos uma classe chamada "Car" que representa os atributos básicos e o comportamento de um carro alugado.

- Veja como o diagrama de classe UML para a classe "Car" pode parecer:



Modelagem UML

Neste diagrama de classe:

- O nome da classe é "Carro"
- Os atributos da classe são "make", "model", "year" e "rentalPrice"
- Os métodos da classe são "rent", "return" e "isAvailable"
- O "-" antes de cada atributo indica que ele é privado (ou seja, acessível apenas dentro da classe)
- O "+" antes de cada método indica que ele é público (ou seja, acessível de fora da classe)

A classe "Carro" tem quatro atributos:

- "marca" e "modelo" são strings que representam a marca e o modelo do carro, respectivamente.
- "ano" é um número inteiro que representa o ano em que o carro foi fabricado.
- "rentalPrice" é um número de ponto flutuante que representa o preço por dia para alugar o carro.

Modelagem UML

A classe "Car" também possui três métodos:

- "rent" recebe um parâmetro inteiro que representa o número de dias em que o carro será alugado. Este método define a disponibilidade do carro como "falso" e calcula o preço do aluguel com base no número de dias e no preço do aluguel diário.
- "retorno" define a disponibilidade do carro como "verdadeiro".
- "isAvailable" retorna um valor booleano que indica se o carro está disponível para aluguel ou não.

Este diagrama de classe UML fornece uma representação clara e concisa da classe "Car" e seus atributos e comportamento. Usando diagramas de classe UML para modelar as entidades e conceitos em um sistema de software, os desenvolvedores podem criar designs fáceis de entender e modificar ao longo do tempo.

Modelagem UML

- Aqui está uma breve visão geral de cada tipo de relacionamento e como ele pode ser implementado em UML:

- Associações: Uma associação é um relacionamento básico entre duas classes, onde uma classe está associada à outra de alguma forma. Por exemplo, uma classe "cliente" pode ser associada a uma classe "pedido", pois um cliente pode fazer vários pedidos. As associações são representadas em UML usando uma linha entre as duas classes, com multiplicidade opcional e nomes de papéis para indicar a natureza do relacionamento.

- Agregações: Uma agregação é uma forma especializada de associação em que uma classe representa uma coleção de objetos e a outra classe representa os objetos contidos na coleção. Por exemplo, uma classe "agência de aluguel de carros" pode agregar vários objetos "carros". As agregações são representadas em UML usando uma forma de losango na classe que contém a coleção, com uma linha conectando-a à classe que representa os objetos da coleção.

Modelagem UML

- Composições: Uma composição é semelhante a uma agregação, mas representa um relacionamento mais forte em que os objetos da coleção são considerados parte do objeto que a contém. Por exemplo, um objeto "carro" pode conter vários objetos "pneu". As composições são representadas em UML usando uma forma de diamante preenchida na classe que contém a coleção.

- Generalizações: Uma generalização é um relacionamento entre uma classe geral (isto é, uma superclasse) e uma classe especializada (isto é, uma subclasse), onde a subclasse herda os atributos e o comportamento da superclasse. Por exemplo, uma classe "sedan" pode herdar de uma classe "carro" mais geral. As generalizações são representadas em UML usando uma seta sólida apontando da subclasse para a superclasse.

Modelagem UML

- Dependências: Uma dependência é um relacionamento entre duas classes onde uma classe depende da outra de alguma forma. Por exemplo, uma classe de "aluguer de carros" pode depender de uma classe de "cartão de crédito" para pagamento. As dependências são representadas em UML usando uma seta tracejada apontando da classe dependente para a classe independente.

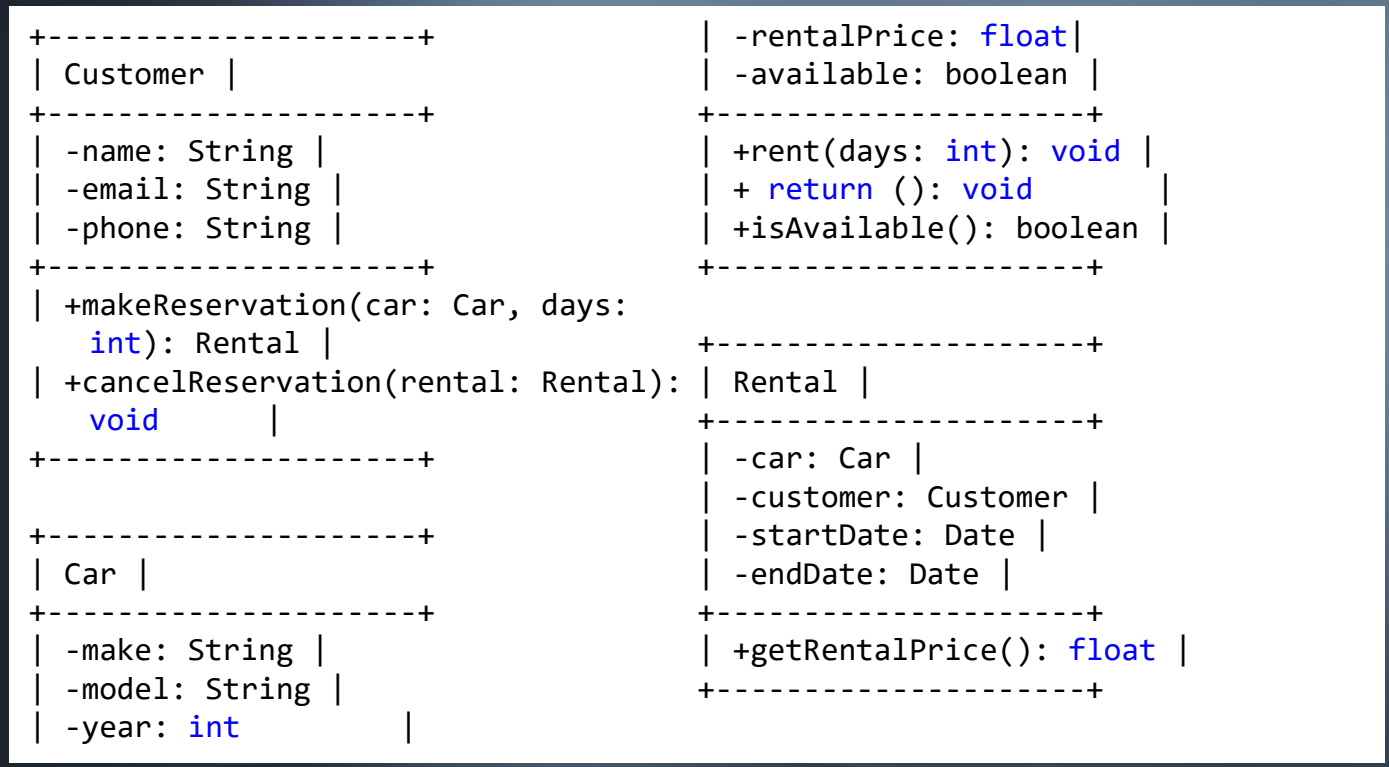
- Para implementar esses relacionamentos em UML, você normalmente os desenharia em um diagrama de classe UML usando os símbolos e anotações apropriados. Além disso, você precisaria definir os detalhes específicos de cada relacionamento, como multiplicidade, nomes de função e atributos e métodos herdados por subclasses. Ao usar esses relacionamentos para modelar as conexões e interações entre classes em um sistema de software, você pode criar designs fáceis de entender, modificar e manter ao longo do tempo.

Modelagem UML

- Composições: Uma composição é semelhante a uma agregação, mas representa um relacionamento mais forte em que os objetos da coleção são considerados parte do objeto que a contém. Por exemplo, um objeto "carro" pode conter vários objetos "pneu". As composições são representadas em UML usando uma forma de diamante preenchida na classe que contém a coleção.

- Generalizações: Uma generalização é um relacionamento entre uma classe geral (isto é, uma superclasse) e uma classe especializada (isto é, uma subclasse), onde a subclasse herda os atributos e o comportamento da superclasse. Por exemplo, uma classe "sedan" pode herdar de uma classe "carro" mais geral. As generalizações são representadas em UML usando uma seta sólida apontando da subclasse para a superclasse.

Modelagem UML



Modelagem UML

Neste diagrama de classe:

A classe "Carro" representa os carros alugados no sistema, com atributos e métodos para rastrear disponibilidade, preço de aluguel e histórico de aluguel.

A classe "Cliente" representa os clientes que podem alugar carros, com atributos para nome, e-mail e número de telefone, bem como métodos para reserva e cancelamento de aluguéis.

A classe "Aluguel" representa uma transação de aluguel específica entre um cliente e um carro, com atributos para o carro, cliente, datas de aluguel e preço do aluguel.

Existem vários relacionamentos definidos entre essas classes:

Modelagem UML

Neste diagrama de classe:

A classe "Carro" representa os carros alugados no sistema, com atributos e métodos para rastrear disponibilidade, preço de aluguel e histórico de aluguel.

A classe "Cliente" representa os clientes que podem alugar carros, com atributos para nome, e-mail e número de telefone, bem como métodos para reserva e cancelamento de aluguéis.

A classe "Aluguel" representa uma transação de aluguel específica entre um cliente e um carro, com atributos para o carro, cliente, datas de aluguel e preço do aluguel.

Existem vários relacionamentos definidos entre essas classes:

Modelagem UML

- A classe "Customer" possui um método "makeReservation" que recebe um objeto "Car" e um número de dias como parâmetros e retorna um novo objeto "Rental" representando a reserva. Este é um exemplo de associação entre as classes "Cliente" e "Carro".
- A classe "Customer" também possui um método "cancelReservation" que toma como parâmetro um objeto "Rental" e cancela a reserva correspondente. Este é outro exemplo de associação entre as classes "Cliente" e "Aluguer".
- A classe "Car" possui um método "rent" que define a disponibilidade do carro como "false" e calcula o preço do aluguel com base no número de dias e no preço diário do aluguel. Este método cria um novo objeto "Aluguel" para representar a transação de aluguel. Este é um exemplo de agregação entre as classes "Carro" e "Aluguel".

Modelagem UML

- A classe "Aluguel" possui atributos para os objetos carro e cliente, representando a associação entre o aluguel e o cliente e o carro. Este é um exemplo de agregação entre as classes "Aluguel" e "Cliente" e "Carro".
- A classe "Rental" também possui um método chamado "getRentalPrice" que calcula o preço total do aluguel com base nas datas de aluguel e no preço do aluguel do carro. Este método usa informações das classes "Car" e "Rental", representando uma relação de dependência entre as duas.
- Usando esses relacionamentos para definir as interações entre classes no sistema, podemos criar um modelo UML abrangente e bem estruturado que representa com precisão o sistema de software que estamos construindo.

Modelagem UML

Aqui está um exemplo de implementação de relacionamentos UML entre classes no sistema de aluguel de carros que definimos anteriormente.

Primeiro, vamos implementar a classe "Car" em Python:

```
class Car :
    def __init__(self, make, model, year,
rental_price, available=True):
        self.make = make
        self.model = model
        self.year = year
        self.rental_price = rental_price
        self.available = available

    def rent(self, days):
        self.available = False
        return Rental(self, days)

    def return_car(self):
        self.available = True

    def is_available(self):
        return self.available
```

Modelagem UML

Em seguida, vamos implementar a classe "Cliente":

```
class Customer :
    def __init__(self, name, email, phone):
        self.name = name
        self.email = email
        self.phone = phone

    def make_reservation(self, car, days):
        return Rental(car, self, days)

    def cancel_reservation(self, rental):
        rental.cancel()
```

Modelagem UML

E por fim, vamos implementar a classe "Aluguel":

```
class Rental :
    def __init__(self, car, customer, days):
        self.car = car
        self.customer = customer
        self.start_date = datetime.datetime.now()
        self.end_date = self.start_date + datetime.timedelta(days
= days)

    def get_rental_price(self):
        days_rented = (self.end_date - self.start_date).days
        return days_rented * self.car.rental_price

    def cancel(self):
        self.car.return_car()
```

Modelagem UML

E por fim, vamos implementar a classe "Aluguel":

```
class Rental :
    def __init__(self, car, customer, days):
        self.car = car
        self.customer = customer
        self.start_date = datetime.datetime.now()
        self.end_date = self.start_date + datetime.timedelta(days
= days)

    def get_rental_price(self):
        days_rented = (self.end_date - self.start_date).days
        return days_rented * self.car.rental_price

    def cancel(self):
        self.car.return_car()
```

Modelagem UML

Nesta implementação, podemos ver como os relacionamentos entre as classes são implementados:

- A classe "Car" possui um método "rent" que retorna um novo objeto "Rental" que representa a transação de aluguel. Este é um exemplo de agregação entre as classes "Carro" e "Aluguer".
- A classe "Customer" possui um método "make_reservation" que retorna um novo objeto "Rental" representando a reserva. Este é um exemplo de associação entre as classes "Cliente" e "Carro" e "Aluguer".
- A classe "Customer" também possui um método "cancel_reservation" que cancela a reserva correspondente chamando o método "cancel" do objeto "Rental". Este é outro exemplo de associação entre as classes "Cliente" e "Aluguer".

Modelagem UML

- A classe "Aluguel" possui atributos para os objetos carro e cliente, representando a associação entre o aluguel e o cliente e o carro. Este é um exemplo de agregação entre as classes "Aluguel" e "Cliente" e "Carro".
- A classe "Rental" também possui um método chamado "get_rental_price" que calcula o preço total do aluguel com base nas datas de aluguel e no preço do aluguel do carro. Este método usa informações das classes "Car" e "Rental", representando uma relação de dependência entre as duas.
- Ao implementar os relacionamentos UML em nosso código, podemos garantir que nosso sistema de software reflita com precisão a estrutura e o comportamento definidos no modelo UML.

Modelagem UML

- A classe "Aluguel" possui atributos para os objetos carro e cliente, representando a associação entre o aluguel e o cliente e o carro. Este é um exemplo de agregação entre as classes "Aluguel" e "Cliente" e "Carro".
- A classe "Rental" também possui um método chamado "get_rental_price" que calcula o preço total do aluguel com base nas datas de aluguel e no preço do aluguel do carro. Este método usa informações das classes "Car" e "Rental", representando uma relação de dependência entre as duas.
- Ao implementar os relacionamentos UML em nosso código, podemos garantir que nosso sistema de software reflita com precisão a estrutura e o comportamento definidos no modelo UML.

Dúvidas???