Modelagem UML

Prof. Dr. Rodrigo Piva

Modelagem UML

- UML (Unified Modeling Language) é uma linguagem de modelagem usada para representar visualmente sistemas de software. Os diagramas de classe UML são uma parte fundamental da UML e são usados para representar as classes, atributos, métodos e relacionamentos entre objetos em um sistema. Aqui estão alguns conceitos avançados em diagramas de classe UML:

Modelagem UML

- Herança: Herança é a capacidade de uma classe herdar propriedades e métodos de outra classe. Nos diagramas de classes UML, a herança é representada por uma seta apontando da subclasse para a superclasse, com um triângulo na extremidade da seta.
- Classes abstratas: classes abstratas são classes que não podem ser instanciadas por conta própria, mas podem ser usadas apenas como classes base para outras classes. Nos diagramas de classe UML, as classes abstratas são representadas por nomes de classe em itálico.
- Interfaces: as interfaces definem um conjunto de métodos que uma classe deve implementar. Nos diagramas de classe UML, as interfaces são representadas por um círculo com o nome da interface dentro.

Modelagem UML

- Multiplicidade: A multiplicidade define o número de instâncias de uma classe que podem ser associadas a instâncias de outra classe. Nos diagramas de classe UML, a multiplicidade é representada por um intervalo de valores (por exemplo, 0..1 ou 1..*) próximo à linha de associação.
- Agregação e composição: Agregação e composição são dois tipos de relacionamentos de associação entre classes. A agregação é um relacionamento "tem-um", em que uma classe contém instâncias de outra classe, mas as instâncias podem existir independentemente da classe que a contém. A composição é um relacionamento "parte de", em que uma classe contém instâncias de outra classe, mas as instâncias não podem existir independentemente da classe que a contém. Nos diagramas de classes UML, a agregação é representada por um losango no final da linha de associação apontando para a classe contida, enquanto a composição é representada por um losango preenchido.
- Dependência: A dependência é um relacionamento em que uma alteração em uma classe pode afetar outra classe, mas não há associação direta entre as duas classes.

Modelagem UML

- Nos diagramas de classe UML, a dependência é representada por uma seta tracejada apontando da classe dependente para a classe independente.
- Estes são apenas alguns dos conceitos avançados em diagramas de classe UML. Existem muitos mais, mas estes devem lhe dar um bom ponto de partida.
- Em UML (Unified Modeling Language), as interfaces são um tipo de construção usado para definir um conjunto de métodos que uma classe ou componente deve implementar. As interfaces são importantes porque permitem polimorfismo e abstração, o que pode tornar um sistema de software mais flexível e extensível.

Modelagem UML

- Existem vários tipos de interfaces em UML, incluindo:
- Interfaces de classe: Uma interface de classe é uma interface implementada por uma classe. Ele define um conjunto de métodos que a classe deve implementar.
- Interfaces de objeto: Uma interface de objeto é uma interface que é implementada por um objeto. Ele define um conjunto de métodos que o objeto deve implementar.

Modelagem UML

- Interfaces de componentes: Uma interface de componente é uma interface implementada por um componente de software. Ele define um conjunto de métodos que o componente deve implementar.
- Interfaces de função: Uma interface de função é uma interface usada para definir o comportamento de um objeto em uma função específica. Por exemplo, se um objeto está desempenhando o papel de "Cliente", ele pode implementar uma interface Cliente que define os métodos relevantes para esse papel.

Modelagem UML

- Interfaces de protocolo: Uma interface de protocolo é uma interface que define um conjunto de mensagens que podem ser enviadas entre objetos. Não define a implementação dessas mensagens, apenas suas assinaturas.
- Interfaces de retorno de chamada: Uma interface de retorno de chamada é uma interface usada para definir um método de retorno de chamada chamado por outro objeto ou componente.
- Esses são alguns dos tipos de interfaces que podem ser usados em UML. O tipo específico de interface usado dependerá dos requisitos do sistema de software que está sendo modelado.

Modelagem UML

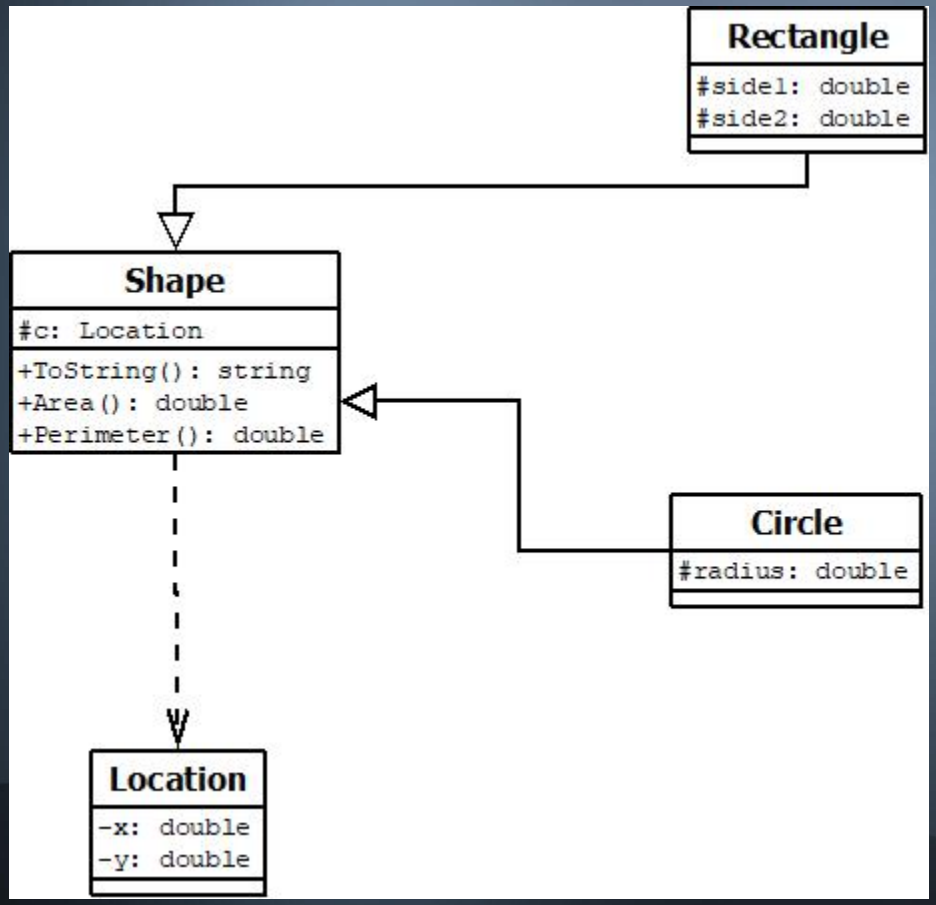
- Vamos dar uma olhada em um exemplo de como uma interface pode ser usada em UML.
- Suponha que estamos projetando um sistema de software que envolve várias formas diferentes, como círculos, retângulos e triângulos. Queremos criar um conjunto de classes que representem essas formas e queremos ser capazes de realizar certas operações sobre essas formas, como calcular sua área e perímetro.

Modelagem UML

- Para conseguir isso, podemos definir uma interface chamada "Shape" que define um conjunto de métodos que devem ser implementados por qualquer classe que represente uma forma. Por exemplo, a interface "Shape" pode definir métodos como "getArea()" e "getPerimeter()".
- Poderíamos então criar classes concretas que implementassem essa interface, como "Círculo“ e "Retângulo" . Cada uma dessas classes forneceria sua própria implementação dos métodos "getArea()" e "getPerimeter()", adaptada à forma específica que ela representa.
- Poderíamos então usar essas classes para criar objetos que representam formas específicas e poderíamos chamar os métodos "getArea()" e "getPerimeter()" nesses objetos para realizar cálculos nas formas. Como cada uma dessas classes implementa a interface "Shape", podemos tratá-las polimorficamente, o que significa que podemos usá-las de forma intercambiável em qualquer situação em que um objeto "Shape" seja necessário.

Modelagem UML

- Aqui está um exemplo de diagrama UML que mostra a interface "Shape" e algumas das classes que a implementam:



Modelagem UML

- Aqui está um exemplo de diagrama UML que mostra a interface "Shape" e algumas das classes que a implementam:

```
public class Shape
{
    protected Location c;

    public string ToString()
    {
        return string.Empty;
    }

    public double Area()
    {
        return 0.000;
    }

    public double Perimeter()
    {
        return 0.000;
    }
}

public class Location
{
    private double x, y;
}

public class Rectangle : Shape
{
    protected double side1,
side2;
}

public class Circle : Shape
{
    protected double radius;
}
```

Modelagem UML

- Neste exemplo, podemos ver que a interface "Shape" define dois métodos, "getArea()" e "getPerimeter()". As classes "Circle", "Rectangle" e "Triangle" implementam essa interface e fornecem suas próprias implementações desses métodos. Cada uma dessas classes tem seu próprio conjunto de atributos específicos para a forma que ela representa.
- Utilizando uma interface para definir os métodos que devem ser implementados por cada uma dessas classes, podemos garantir que cada classe de shape forneça um conjunto consistente de comportamentos, facilitando o trabalho com esses objetos de forma polimórfica.

Modelagem UML

- Em UML, um pacote lógico é um mecanismo de agrupamento usado para organizar elementos de modelo em uma estrutura hierárquica. Um pacote pode conter outros pacotes, classes, interfaces e outros tipos de elementos de modelo. Os pacotes são uma maneira de gerenciar a complexidade de um grande sistema, dividindo-o em unidades menores e mais gerenciáveis.
- Pacotes lógicos em UML são normalmente usados para agrupar classes relacionadas e outros elementos de modelo com base em sua funcionalidade ou propósito. Por exemplo, você pode ter um pacote para todas as classes relacionadas à autenticação do usuário, outro pacote para todas as classes relacionadas ao armazenamento de dados e assim por diante. Ao organizar suas classes e outros elementos em pacotes lógicos, você pode facilitar a navegação e a compreensão do seu modelo.

Modelagem UML

Aqui estão alguns dos principais recursos e benefícios do uso de pacotes lógicos em UML:

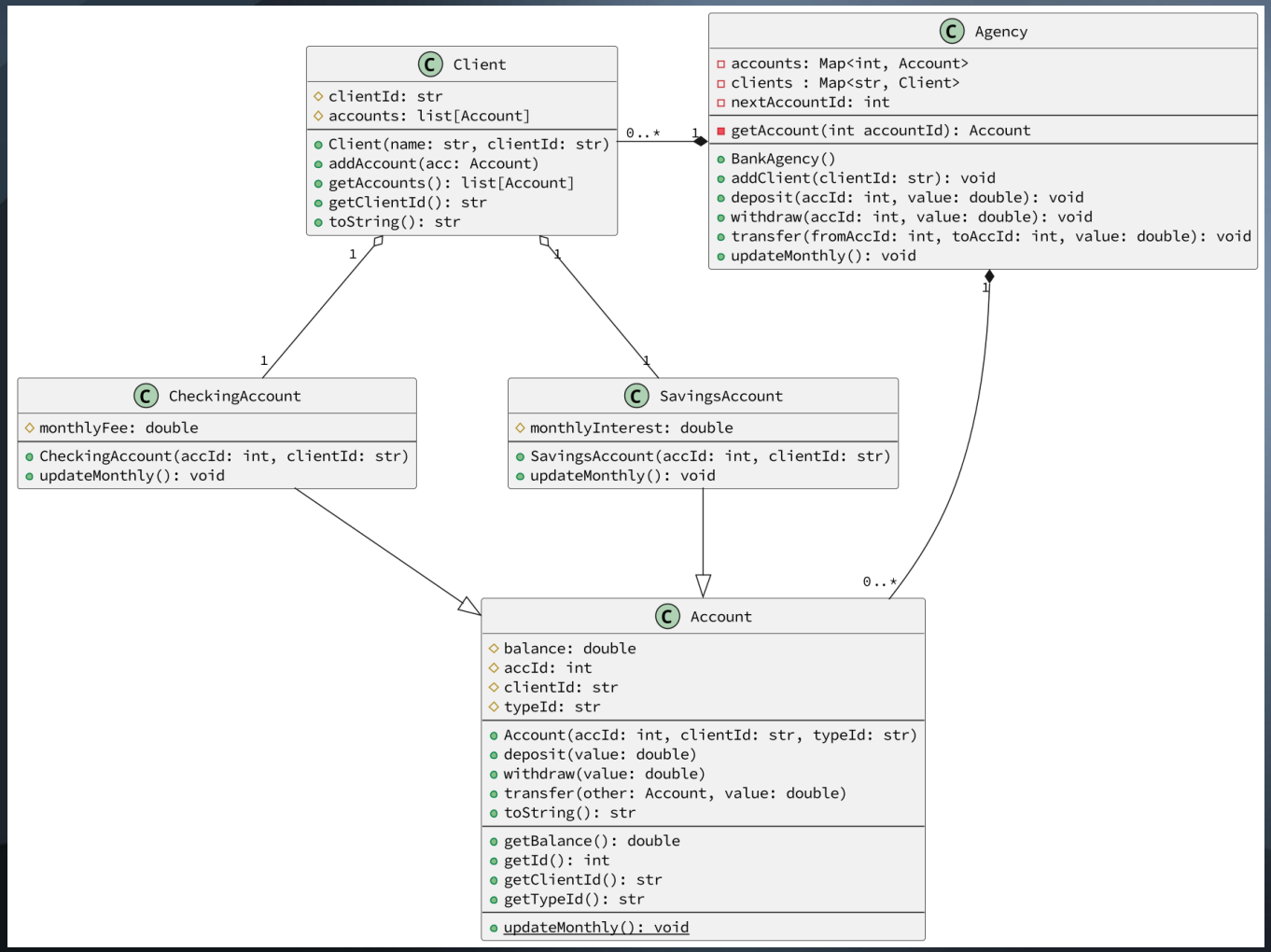
- Estrutura hierárquica: os pacotes lógicos permitem organizar os elementos do modelo em uma estrutura hierárquica, facilitando a navegação e a compreensão do modelo.
- Encapsulamento: os pacotes fornecem uma maneira de encapsular elementos de modelo relacionados e disponibilizá-los para outras partes do sistema somente quando necessário.
- Visibilidade: os pacotes fornecem uma maneira de controlar a visibilidade dos elementos do modelo. Você pode especificar quais elementos do modelo são visíveis para outras partes do sistema e quais não são.

Modelagem UML

- Modularização: Ao dividir seu sistema em pacotes lógicos, você pode criar um sistema mais modular que seja mais fácil de testar, manter e estender.
- Reutilização: os pacotes fornecem uma maneira de agrupar elementos de modelo relacionados, facilitando a reutilização desses elementos em outras partes do sistema.

Modelagem UML

- Aqui está um exemplo de um diagrama de objetos para um sistema bancário simples:



Neste exemplo, temos quatro classes: "Account", "CheckingAccount", "SavingsAccount" e "Clien". As classes "CheckingAccount" e "SavingsAccount" são subclasses da classe "Account".

Modelagem UML

- Aqui está um exemplo de código Java que corresponde ao diagrama de objeto:

```
public class Account
{
    private int accountNumber;
    private double balance;

    public Account(int accountNumber, double
balance)
    {
        this.accountNumber = accountNumber;
        this.balance = balance;
    }

    public int getAccountNumber()
    {
        return accountNumber;
    }

    public void setAccountNumber(int accountNumber)
    {
        this.accountNumber = accountNumber;
    }

    public double getBalance()
    {
        return balance;
    }

    public void setBalance(double balance)
    {
        this.balance = balance;
    }
}
```

Modelagem UML

- Aqui está um exemplo de código Java que corresponde ao diagrama de objeto:

```
public class CheckingAccount extends Account
{
    private double overdraftLimit;

public CheckingAccount(int accountNumber, double
balance, double overdraftLimit)
{
    super(accountNumber, balance);
    this.overdraftLimit = overdraftLimit;
}

public double getOverdraftLimit()
{
    return overdraftLimit;
}

public void setOverdraftLimit(double
overdraftLimit)
{
    this.overdraftLimit = overdraftLimit;
}
}
```

Modelagem UML

- Aqui está um exemplo de código Java que corresponde ao diagrama de objeto:

```
public class SavingsAccount extends Account
{
    public SavingsAccount(int accountNumber, double balance)
{
    super(accountNumber, balance);
}
}
```

Modelagem UML

- Aqui está um exemplo de código Java que corresponde ao diagrama de objeto:

```
public class Customer
{
    private String name;
    private String address;
    private Account account;

    public Customer(String name, String address, Account account)
    {
        this.name = name;
        this.address = address;
        this.account = account;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public String getAddress()
    {
        return address;
    }

    public void setAddress(String address)
    {
        this.address = address;
    }

    public Account getAccount()
    {
        return account;
    }

    public void setAccount(Account account)
    {
        this.account = account;
    }
}
```

Modelagem UML

```
public class BankingSystem

{

    public static void main(String[]
args)

    {

        // Create account objects

        Account account1 = new
Account(1001, 5000);

        CheckingAccount checkingAcct1 =
new CheckingAccount(2001, 1000, 500);

        SavingsAccount savingsAcct1 = new
SavingsAccount(3001, 5000);



        // Create customer object and
associate with account object

        Customer customer1 = new
Customer("John Smith", "123 Main St.",
account1);


customer1.getAccount().setAccountNumber(1
002); // Change account number



        // Print out customer and account
information

        System.out.println("Customer
Name: " + customer1.getName());

        System.out.println("Customer
Address: " + customer1.getAddress());

        System.out.println("Account
Number: " +
customer1.getAccount().getAccountNumber()
);

        System.out.println("Account
Balance: " +
customer1.getAccount().getBalance());


        // Associate checking and savings
accounts with customer


customer1.setAccount(checkingAcct1);


customer1.setAccount(savingsAcct1);


        // Print out updated customer and
account information

        System.out.println("Customer
Name: " + customer1.getName());

        System.out.println("Customer
Address: " + customer1.getAddress());

        System.out.println("Checking
Account Number: " +
((CheckingAccount)customer1.getAccount())
.getAccountNumber());

        System.out.println("Checking
Account Balance: " +
((CheckingAccount)customer1.getAccount())
.getBalance());

        System.out.println("Checking
Account Overdraft Limit: " +
((CheckingAccount)customer1.getAccount())
.getOverdraftLimit());

        System.out.println("Savings
Account Number: " +
((SavingsAccount)customer1.getAccount()).
getAccountNumber());

        System.out.println("Savings
Account Balance: " +
((SavingsAccount)customer1.getAccount()).
getAccountBalance());
```

Modelagem UML

- Esta saída representa o estado do sistema após a execução do código. As primeiras linhas exibem o estado inicial dos objetos Cliente e Conta, enquanto as linhas posteriores mostram o estado atualizado depois que os objetos CheckingAccount e SavingsAccount foram associados ao objeto Customer.

```
Customer Name: John Smith
Customer Address: 123 Main St.
Account Number: 1002
Account Balance: 5000.0
Customer Name: John Smith
Customer Address: 123 Main St.
Checking Account Number: 3001
Checking Account Balance: 5000.0
Checking Account Overdraft Limit: 500.0
Savings Account Number: 3001
Savings Account Balance: 5000.0
```

Dúvidas???