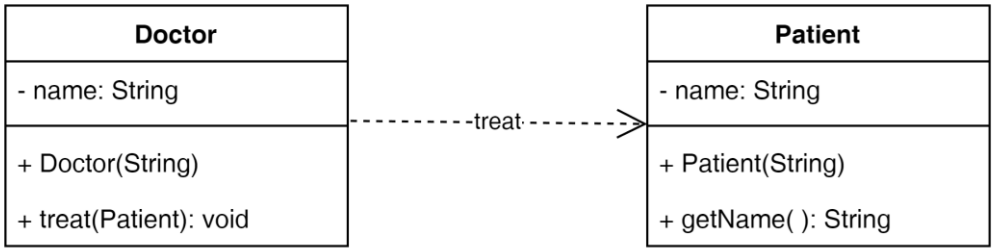
UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIAS DE COMPUTAÇÃO
SCC0504/SCC0204 – Programação Orientada a Objetos
Prof. José Fernando Rodrigues Júnior

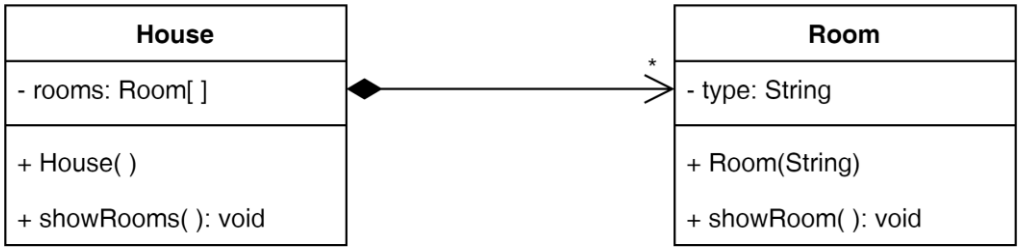
Exercícios UML

1.Considere duas classes: Medico e Paciente.
Implemente um relacionamento de **associação simples**, onde um Medico pode atender um Paciente, mas sem armazená-lo como atributo.
O Paciente também pode ser tratado por diferentes médicos, sem pertencer a nenhum deles.
Represente o relacionamento em UML e explique por que ele não se trata de agregação nem de composição.




```
class Patient {
    private String name;

    public Patient(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}

class Doctor {
    private String name;

    public Doctor(String name) {
        this.name = name;
    }

    public void treat(Patient patient) {
        System.out.println("Dr. " + name + " is treating patient " + patient.getName());
    }
}

public class UMLSimpleAssociationExample3 {
    public static void main(String[] args) {
        Doctor doctor = new Doctor("Smith");
        Patient patient = new Patient("John");

        doctor.treat(patient);
    }
}
```

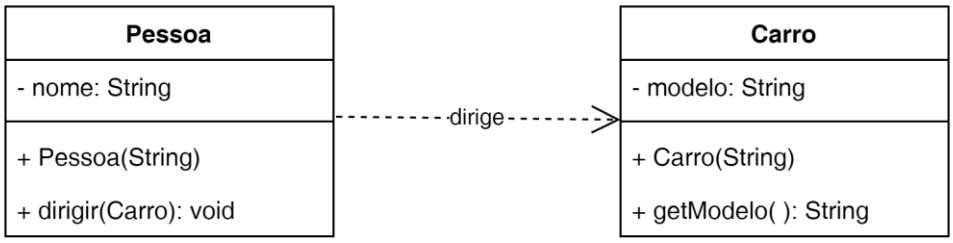
2.Considere duas classes: Casa e Quarto.
Implemente um relacionamento de **composição** onde uma Casa possui múltiplos Quartos. Cada Quarto deve ser criado dentro da Casa, e não pode existir sem ela.
Se a Casa for destruída, todos os Quartos também devem ser eliminados.
Represente o relacionamento em UML e explique por que se trata de uma composição.




```
class Room {
    private String type;

    public Room(String type) {
        this.type = type;
    }

    public void showRoom() {
        System.out.println("Room type: " + type);
    }
}

class House {
    private Room[] rooms; // Using an array instead of ArrayList

    public House() {
        rooms = new Room[3]; // Define a fixed-size array
        rooms[0] = new Room("Bedroom");
        rooms[1] = new Room("Kitchen");
        rooms[2] = new Room("Bathroom");
    }

    public void showRooms() {
        for (Room room : rooms) {
            room.showRoom();
        }
    }
}

public class UMLCompositionExample {
    public static void main(String[] args) {
        House house = new House();
        house.showRooms();
    }
}
```

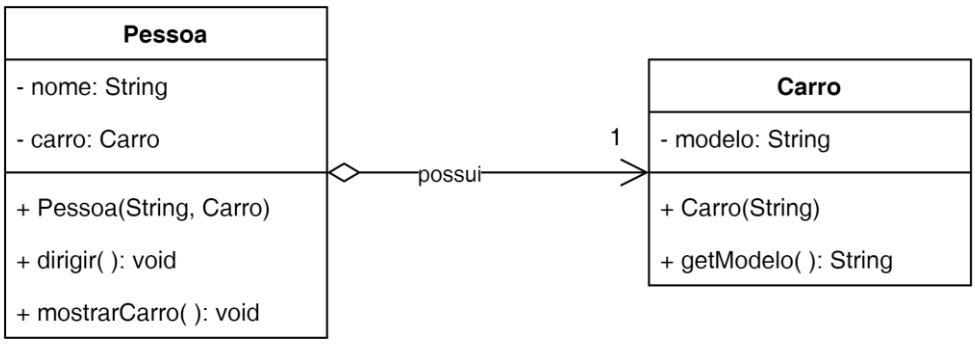
3.Associação x Agregação
Considere duas classes: Pessoa e Carro.

a) Simples Associação: Implemente um método onde uma Pessoa apenas dirige um Carro, sem armazená-lo como atributo.




```
class Carro {
    private String modelo;

    public Carro(String modelo) {
        this.modelo = modelo;
    }

    public String getModelo() {
        return modelo;
    }
}

class Pessoa {
    private String nome;

    public Pessoa(String nome) {
        this.nome = nome;
    }

    public void dirigir(Carro carro) { // Apenas interação, sem armazenar
        System.out.println(nome + " está dirigindo um " + carro.getModelo());
    }
}

public class SimplesAssociacao {
    public static void main(String[] args) {
        Pessoa pessoa = new Pessoa("Carlos");
        Carro carro = new Carro("Honda Civic");

        pessoa.dirigir(carro); // Simples associação: apenas interação momentânea
    }
}
```

b) Agregação: Modifique o código para que a Pessoa tenha um Carro, mantendo uma referência ao objeto Carro, mas sem criá-lo dentro da classe Pessoa.




```
class Carro {
    private String modelo;
```

```
    public Carro(String modelo) {
        this.modelo = modelo;
    }

    public String getModelo() {
        return modelo;
    }
}

class Pessoa {
    private String nome;
    private Carro carro; // Agregação: Pessoa tem um Carro, mas não o cria

    public Pessoa(String nome, Carro carro) {
        this.nome = nome;
        this.carro = carro;
    }

    public void dirigir() { // Apenas interação, sem armazenar
        System.out.println(nome + " está dirigindo um " + this.carro.getModelo());
    }

    public void mostrarCarro() {
        if (carro != null) {
            System.out.println(nome + " possui um " + carro.getModelo());
        } else {
            System.out.println(nome + " não possui um carro.");
        }
    }
}

public class Agregacao {
    public static void main(String[] args) {
        Carro carro = new Carro("Toyota Corolla");
        Pessoa pessoa = new Pessoa("Alice", carro); // O Carro é passado como parâmetro

        pessoa.mostrarCarro(); // Agregação: Pessoa tem um Carro
    }
}
```

Resumo:

Aspecto	Simples Associação (Pessoa-Carro)	Agregação (Pessoa-Carro)	Composição (Casa-Quarto)
Relação	Apenas **interação momentânea**	`Pessoa` **possui um** `Carro`, mas ele pode existir sem ela	`Casa` **possui** `Quartos`, que não existem sem a `Casa`
Armazena o objeto?	❌ Não	✅ Sim (como atributo)	✅ Sim (como atributo)
Ciclo de vida	Independente	Independente	`Quarto` **depende de** `Casa`
Dependência Semântica	Nenhuma posse	**Fraca posse**	**Forte posse (Quarto não existe sem Casa)**
Dependência de Compilação	✅ Sim (`Pessoa` usa `Carro` no método)	✅ Sim (`Pessoa` contém um atributo `Carro`)	✅ Sim (`Casa` contém um atributo `Quarto`)
Exemplo	`Pessoa.dirigir(Carro)` apenas usa o `Carro`	`Pessoa` tem um atributo `Carro` e recebe a referência	`Casa` cria e gerencia os `Quartos` internamente
Persistência do objeto	O `Carro` continua existindo após `Pessoa` ser removida	O `Carro` continua existindo após `Pessoa` ser removida	Se a `Casa` for removida, todos os `Quartos` também são destruídos

4) Implemente um sistema que modele uma Universidade, considerando três tipos de relacionamento:

Associação Simples:
Um Professor pode ministrar uma aula para um Aluno, mas sem armazená-lo como atributo.
O mesmo Aluno pode assistir aulas de vários Professores.
Implemente um método para representar essa interação.

Agregação:
Uma Universidade pode ter vários Professores cadastrados.
No entanto, um Professor pode existir independentemente da Universidade (ele pode trabalhar em outra instituição).
A Universidade deve armazenar uma lista de Professores, mas sem criá-los dentro da classe.

Composição:
Uma Universidade contém Departamentos.
Um Departamento não pode existir sem a Universidade.
Se a Universidade for removida, todos os seus Departamentos devem ser destruídos.

Modele em UML e implemente em Java.

```
class Aluno {
    private String nome;

    public Aluno(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }
}

class Professor {
    private String nome;

    public Professor(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }

    // Associação Simples: O professor ensina um aluno sem armazená-lo
    public void ensinar(Aluno aluno) {
        System.out.println("Professor " + nome + " está ensinando o aluno " +
aluno.getNome());
    }
}

class Departamento {
    private String nome;

    public Departamento(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }
}

class Universidade {
    private String nome;
    private Professor[] professores; // Agregação: Armazena professores sem criá-los
internamente
    private Departamento[] departamentos; // Composição: Departamentos são criados dentro da
Universidade
```

```
    private int contadorProfessores;

    public Universidade(String nome, int capacidadeProfessores, int quantidadeDepartamentos)
{
        this.nome = nome;
        this.professores = new Professor[capacidadeProfessores]; // Array fixo de professores
        this.departamentos = new Departamento[quantidadeDepartamentos]; // Array fixo de
departamentos
        this.contadorProfessores = 0;

        // Criando os departamentos internamente (Composição)
        departamentos[0] = new Departamento("Ciência da Computação");
        departamentos[1] = new Departamento("Engenharia Elétrica");
    }

    public void adicionarProfessor(Professor professor) {
        if (contadorProfessores < professores.length) {
            professores[contadorProfessores++] = professor;
        } else {
            System.out.println("Não é possível adicionar mais professores à " + nome);
        }
    }

    public void listarProfessores() {
        System.out.println("Professores da " + nome + ":");
        for (int i = 0; i < contadorProfessores; i++) {
            System.out.println("- " + professores[i].getNome());
        }
    }

    public void listarDepartamentos() {
        System.out.println("Departamentos da " + nome + ":");
        for (Departamento departamento : departamentos) {
            if (departamento != null) {
                System.out.println("- " + departamento.getNome());
            }
        }
    }
}

public class UniversidadeExemplo {
    public static void main(String[] args) {
        // Criando Universidade com capacidade para 2 professores e 2 departamentos
        Universidade universidade = new Universidade("Universidade XPTO", 2, 2);

        // Criando Professores (que podem existir sem a Universidade)
        Professor professor1 = new Professor("Dr. Silva");
        Professor professor2 = new Professor("Prof. Maria");

        // Criando Alunos
        Aluno aluno1 = new Aluno("Carlos");
        Aluno aluno2 = new Aluno("Ana");

        // Associação Simples: Professores ensinam alunos
        professor1.ensinar(aluno1);
        professor2.ensinar(aluno2);

        // Agregação: Adicionando professores à Universidade
        universidade.adicionarProfessor(professor1);
        universidade.adicionarProfessor(professor2);

        // Exibindo Professores
        universidade.listarProfessores();

        // Exibindo Departamentos (que foram criados dentro da Universidade)
        universidade.listarDepartamentos();
    }
}
```