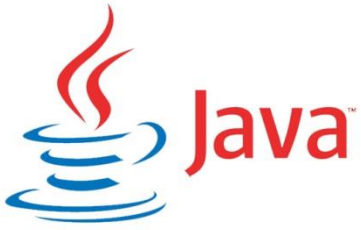
Lógica de Programação

Unidade 16 – Associação entre Classes de Objetos



QI ESCOLAS E FACULDADES
Curso Técnico em Informática

SUMÁRIO

INTRODUÇÃO

Nesta unidade trataremos sobre as relações que podem existir entre as classes do sistema, bem como sua representação na UML e a codificação em Java.

DEPENDÊNCIA

A forma mais comum de relacionamento entre classes de um sistema é a de **dependência**. Você já deve ter observado no *BlueJ*, quando criamos a classe *Main* e nela instanciamos um objeto de outra classe, imediatamente ele representa a relação utilizando uma linha tracejada com uma seta aberta. Observe na figura 1 a relação de dependência entre a classe Data e a classe *Main*.

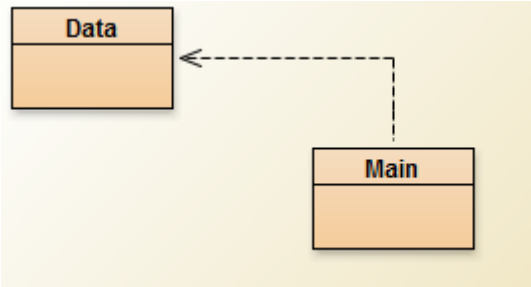


Figura 1 – Dependência

Observe que a seta sai da classe *Main* e aponta para a classe Data, indicando que a primeira **utiliza** a segunda, ou seja, sempre que utilizamos uma classe dentro de outra (no caso Data dentro da Main) estamos estabelecendo uma dependência, onde mudanças na implementação na Data podem afetar o funcionamento da *Main*. **O excesso de dependência entre as classes pode tornar o sistema complexo de atualizar.**

ASSOCIAÇÃO

Uma relação de associação é uma relação de dependência mais forte, com significado. Se pensarmos nas classes Aluno e Disciplina, podemos ter alunos e disciplinas no sistema sem que um aluno esteja obrigatoriamente matriculado em uma disciplina e vice-versa. Quando a relação não é de uso (no exemplo da figura 1, a classe *Main* **usa** a classe Data) ela não é classificada simplesmente como dependência, dependendo da situação ela pode ser classificada como **associação**. No exemplo do aluno e da disciplina, temos:

Um aluno cursa uma ou mais disciplinas.

Aqui podemos observar que temos dois objetos e que eles possuem uma relação. Esta relação é representada no diagrama por uma linha sólida ligando as classes (figura 2).

Figura 2

A linha sólida sem setas indica que a relação é binária. Interpretamos que um aluno cursa disciplina e que a disciplina é cursada pelo aluno. Assim as duas classes se conhecem.

Caso o objetivo seja direcionar a relação, utiliza-se uma seta aberta, indicando qual classe utiliza qual. Observe a figura 3.

Figura 3

No exemplo acima indicamos que uma Turma está associada à classe Aluno, então podemos navegar na classe Aluno através da classe Turma, mas o contrário não é verdadeiro.

Multiplicidade

Ainda no diagrama podemos representar quantos objetos podem se relacionar. De acordo com Larman (2007), o valor da multiplicidade informa quantas instâncias podem ser associadas com outra instância num determinado momento, mas não ao longo do tempo.

Por exemplo, podemos determinar que uma turma pode conter nenhum ou no máximo 40 alunos. Observe o exemplo da figura 4.

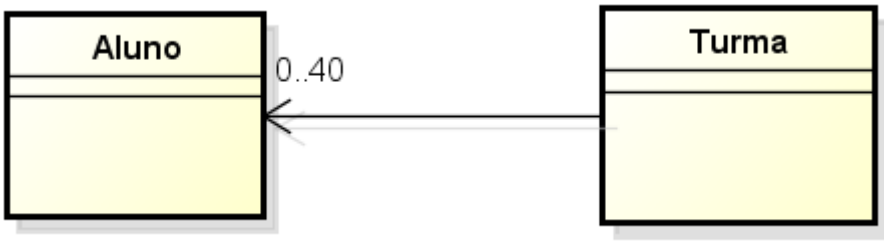


Figura 4

Exemplos de representação de multiplicidade

1 .....apenas um

*.....muitos

0..*.....nenhum ou muitos

1..*.....no mínimo 1

2 .....exatamente 2

1..5 .....mínimo 1 e máximo 5

Assim, na figura 5 representamos uma relação entre as classes A e B.

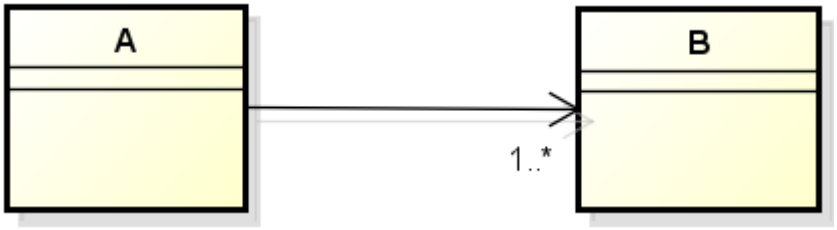


Figura 5

Lê-se: para cada instância de A, temos no mínimo 1 ou várias instâncias de B associadas a ela.

Estudo de caso I

Vamos considerar um CD de músicas. Um CD tem ano de lançamento, título e um conjunto de faixas. Cada faixa tem um nome, um ou mais artistas (que tem um nome e um gênero musical) e uma duração, composta de minutos e segundos. Observando que

cada música tem um tempo, somando os tempos de todas as músicas sabemos a duração total do CD.

Primeiro passo: mapear os objetos

Quais os objetos que conseguimos identificar? CD, Faixa, Tempo, Artista.

Segundo passo: identificar as relações

Um CD tem faixas;

Uma faixa tem um tempo de duração;

Uma faixa tem artistas;

Terceiro passo: montar o diagrama com os objetos e atributos

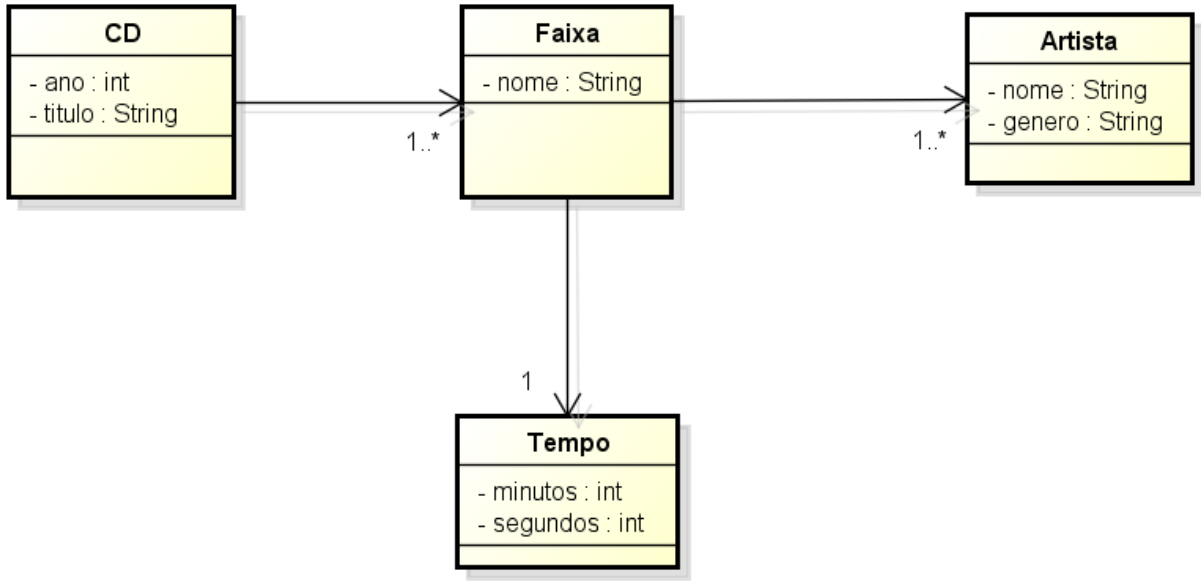


Figura 6

Vamos analisar o diagrama: A classe CD tem uma ligação com a classe Faixa, onde a multiplicidade indica que um CD pode conter apenas uma ou muitas faixas (o número não é determinado). Isto quer dizer que, dentro da classe CD teremos uma coleção de objetos da classe Faixa. A classe Faixa, por sua vez, tem uma ligação com a classe Tempo, onde a multiplicidade indica que cada objeto faixa contém apenas um tempo. A classe Faixa também possui uma ligação com a classe Artista, indicando que uma faixa pode conter um ou mais artistas.

Estudo de caso II

Considere um *set* de uma partida de vôlei. Participam da partida dois times, cada time com um nome, um técnico e a quantidade de pontos acumulados. Na partida os times precisam atingir 25 pontos. A partida encerra quando um time atingir 25 pontos (com dois pontos de diferença) ou quando um deles tiver dois 2 pontos a mais que outro (acima dos 25 pontos).

Mapeamento dos objetos

Time, Set

Identificando as relações

Um set é disputado por dois times

Montando o diagrama de classe com atributos

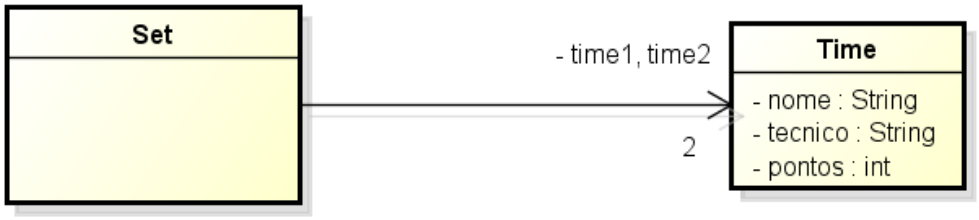


Figura 7

Observamos que a relação de associação indica que um set está associado a dois times. Portanto, dentro da classe Set teremos dois atributos que serão objetos da classe Time.

Montando o diagrama de classe com os métodos

Vamos pensar nas operações: os times pontuam e precisamos controlar o andamento da partida. Logo, podemos acrescentar um método pontuar na classe Time e também um método continuarSet na classe Set que indique *true* se a partida deve continuar ou *false* se ela deve encerrar. Na classe Set podemos ter um método que nos retorne o time campeão.

Vamos observar o diagrama de classe com os métodos:

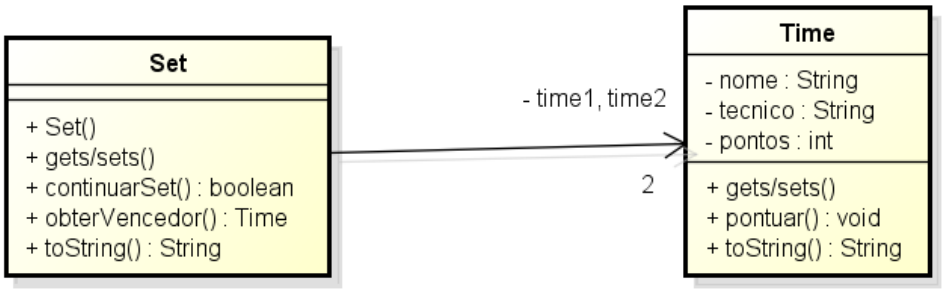


Figura 8

Codificando

Iniciamos pelas classes que recebem as setas, no caso a classe Time.

Classe Time

```
public class Time {
    private String nome;
    private String tecnico;
    private int pontos;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getTecnico() {
        return tecnico;
    }

    public void setTecnico(String tecnico) {
        this.tecnico = tecnico;
    }

    public int getPontos() {
        return pontos;
    }

    public void setPontos(int pontos) {
        this.pontos = pontos;
    }

    public void pontuar(){
        this.pontos++;
    }

    public String toString(){
        return this.nome + " - " + this.pontos + " ponto(s)";
    }
}
```

Classe Set

```
public class Set {
    private Time time1;
    private Time time2;

    public Set(){
        this.time1 = new Time();
        this.time2 = new Time();
    }

    public Time getTime1() {
        return time1;
    }

    public void setTime1(Time time1) {
        this.time1 = time1;
    }

    public Time getTime2() {
        return time2;
    }

    public void setTime2(Time time2) {
        this.time2 = time2;
    }

    public boolean continuarSet(){
        int pt1 = this.time1.getPontos();
        int pt2 = this.time2.getPontos();
        if(pt1 > pt2 && pt1 - pt2 >=2 && pt1 >=25
                || pt2 > pt1 && pt2 - pt1 >= 2 && pt2 >= 25){
            return false;
        }else{
            return true;
        }

    }

    public Time obterVencedor(){
        int pt1 = this.time1.getPontos();
        int pt2 = this.time2.getPontos();
        if(pt1 > pt2){
            return this.time1;
        }else{
            return this.time2;
        }
    }
    public String toString(){
        return "Time 01: " + this.time1
                + "\nTime 02: " + this.time2;
    }
}
```

Na classe Set, temos o construtor, que instancia os dois objetos da classe Time. Depois dos *gets* e *sets*, temos o método *continuarSet*.

Este método primeiramente obtém a quantidade de pontos de cada time e armazena nas variáveis:

```
int pt1 = this.time1.getPontos();
int pt2 = this.time2.getPontos();
```

Após isto, temos um *if* que testa se algum dos times atingiu a pontuação para vencer.

```
if(pt1 > pt2 && pt1 - pt2 >=2 && pt1 >=25
                || pt2 > pt1 && pt2 - pt1 >= 2 && pt2 >= 25){
```

Lê-se: Se os pontos do time1 forem maiores que os pontos do time2 **e** se a diferença de pontos por maior ou igual a 2 **e** se o total de pontos do time1 for maior ou igual a 25 **ou** os pontos do time2 forem maiores que os pontos do time1 **e** se a diferença de pontos por maior ou igual a 2 **e** se o total de pontos do time2 for maior ou igual a 25. Se esta condição for verdadeira, a partida não deve continuar, portanto o método retorna *false*. Caso contrário, retorna *true*.

Em seguida temos o método *obterVencedor* que retorna o time com mais pontos.

Classe Main

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner ler = new Scanner(System.in);
        Set set1 = new Set();
        byte escolha;

        /* Definição dos times */
        Time t1 = new Time();
        System.out.println("**** TIME 1 ****");
        System.out.print("Nome do time 1: ");
        t1.setNome(ler.next());
        System.out.print("Técnico do time 1: ");
        t1.setTecnico(ler.next());

        Time t2 = new Time();
        System.out.println("**** TIME 2 ****");
        System.out.print("Nome do time 2: ");
        t2.setNome(ler.next());
        System.out.print("Técnico do time 2: ");
        t2.setTecnico(ler.next());

        set1.setTime1(t1);
        set1.setTime2(t2);

        do{ /*início do controle da partida */
            System.out.println(set1 + "\n");
            System.out.println("1 - Marcar ponto para: " + t1.getNome());
```

```
            System.out.println("2 - Marcar ponto para: " + t2.getNome());
            escolha = ler.nextByte();
            switch(escolha){
                case 1:
                    t1.pontuar();
                    System.out.println("Ponto para " + t1.getNome());
                    break;
                case 2:
                    t2.pontuar();
                    System.out.println("Ponto para " + t2.getNome());
                    break;
            }
        }while(set1.continuarSet());

        System.out.println("\nPartida encerrada!!");
        System.out.println(set1);
        System.out.println("VENCEDOR: " + set1.obterVencedor());
    }
}
```

Na *Main*, criamos o objeto set1 e em seguida criamos os times, definindo seus dados; após isto, os times são enviados para o objeto set1 via método *set*. Em seguida, o sistema apresenta as opções: pontuar para o time 1 ou pontuar para o time 2. O sistema continua executando enquanto o método continuarSet retornar verdadeiro (enquanto algum dos times não atingir a pontuação para vencer). Por último apresentamos os resultados.

REFERÊNCIAS

LARMAN, Craig. **Utilizando UML e Padrões: *Uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo***. Porto Alegre: Bookman, 3ª ed., 2007.