Enumerações
Associação: Agregação e Composição.
Static, Final

Professor Isaac

Enumerações

Enumeração

- Embora a enumeração seja um recurso de programação comum que é encontrado em muitas outras linguagens de computador, ela não fazia parte da especificação Java original.

- Em sua forma mais simples, uma enumeração é uma lista de constantes nomeadas que define um novo tipo de dado.

- Do ponto de vista da programação, as enumerações são úteis sempre que precisamos definir um conjunto de valores que representam um grupo de itens.

Enumeração

- Uma enumeração é criada com o uso da palavra-chave **enum**. Por exemplo, aqui está uma enumeração simples que lista vários meios de transporte:

```
1   // Enumeração de meios de transporte.
2   enum Transport {
3    CAR, TRUCK, AIRPLANE, TRAIN, BOAT
4   }
```

- Os identificadores CAR, TRUCK, etc., são chamados de constantes de enumeração, ou constantes enum, na abreviação.

Enumeração

- Uma vez que você tiver declarado uma enumeração, poderá criar uma variável desse tipo.

```
Transport tp;
```

- Como tp é de tipo Transport, os únicos valores que ela pode receber são os definidos pela enumeração ou null.

  Por exemplo:

```
tp = Transport.AIRPLANE;
```

Enumeração

- Duas constantes de enumeração podem ser comparadas em busca de igualdade com o uso do operador relacional ==.

  Por exemplo:

```
if(tp == Transport.TRAIN){
  // TODO add your handling code here:
}
```

Enumeração

- O valor de uma enumeração também pode ser usado no controle de uma instrução switch. Obviamente, todas as instruções case devem usar constantes da mesma enum usada pela expressão switch.

  Por exemplo:

```
// Usa uma enum para controlar uma instrução switch.
switch(tp) {
  case CAR:
  // ...
  case TRUCK:
  // ...
```

Enumeração

- Quando uma constante de enumeração é exibida, como em uma instrução println( ), seu nome compõe a saída.

Por exemplo:

```
1   // Enumeração de meios de transporte.
2   enum Transport {
3     CAR, TRUCK, AIRPLANE, TRAIN, BOAT
4   }
5
6   class Main {
7     public static void main(String[] args) {
8       Transport tp;
9       tp = Transport.AIRPLANE;
10      // Exibe um valor da enum.
11      System.out.println("Valor do tp: " + tp);
12    }
13  }
```

```
Valor do tp: AIRPLANE
>
```

final

final

- Declara variáveis constantes de leitura.
- Exemplos:

```
public final double pi = 3.1415926; // Atributo com valor imutável (não pode ser modificado)

public final int x; // O conteúdo do atributo só pode e tem que ser atribuído no construtor

public final void metodo(); // O método não pode ser sobrescrito

public final class String{ } // classe não pode ser estendida (sem herança)

public final A ob = new A(); // Objeto não pode ser instanciado novamente
```

static

static - métodos

- São métodos que realizam tarefas que independem de objetos
- Estes métodos podem ser acessados pelo nome da classe, seguido do nome do método, sem nenhum objeto ter sido criado;

```
1  class Teste{
2      public static void metodo(){
3          System.out.println("método chamado sem objeto ");
4      }
5  }
6
7  class Main{
8      public static void main(String args[]){
9          Teste.metodo();
10     }
11 }
```

static - variável de instância

- São atributos compartilhados entre todos os objetos da mesma classe;
- Cada objeto não tem sua própria cópia deste atributo.

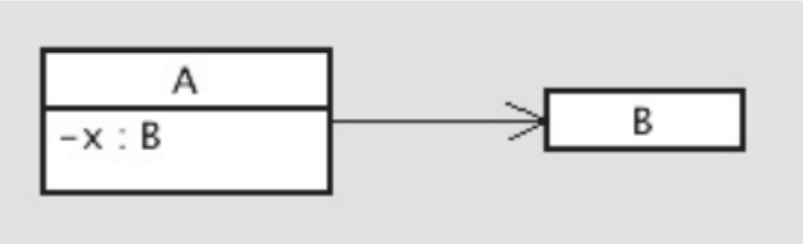
```
1  class Teste{
2      public static int i = 0;
3      public void metodo(){
4          System.out.println("método chamado pelo objeto " + i++);
5      }
6  }
7
8  class Main{
9      public static void main(String args[]){
10         Teste t = new Teste();
11         t.metodo();
12         Teste t1 = new Teste();
13         t1.metodo();
14         Teste t2 = new Teste();
15         t2.metodo();
16     }
17 }
```

```
método chamado pelo objeto 0
método chamado pelo objeto 1
método chamado pelo objeto 2
```

Associação: Agregação e Composição.

Associação

- Uma associação é um relacionamento estrutural entre duas classes através de seus objetos.
- Ocorre quando um objeto é construído usando outro(s) objeto(s).
- Na criação da classe um ou mais atributos são objetos de outras classes.



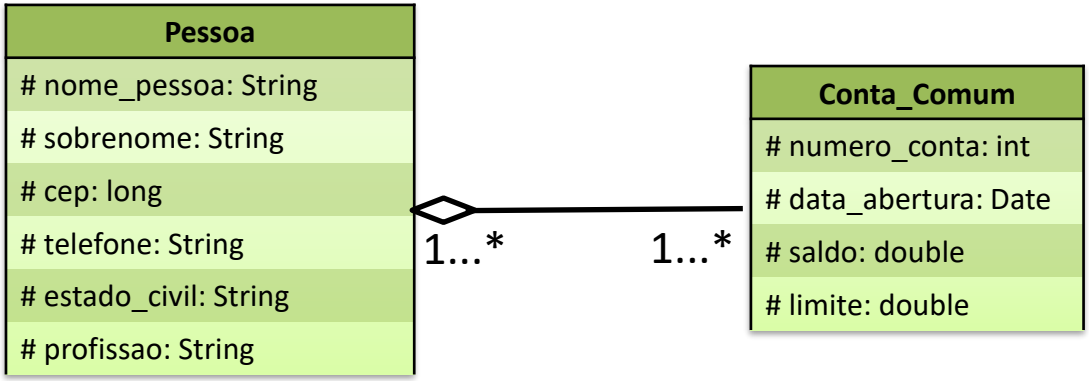
- Permite diminuir a complexidade da classe pela reutilização dos atributos e métodos já definidos pela(s) outra(s) classe

Agregação

- A agregação é uma forma especial de associação que é uma relação unidirecional entre classes.

- Na agregação as informações de um objeto (chamado objeto-todo) precisa ser complementada pelas informações contidas em um ou mais objetos de uma classe (chamados objeto-parte).

- No exemplo abaixo vemos que uma pessoa pode possuir uma ou mais contas, como a conta também pode possuir várias pessoas (conta conjunta).

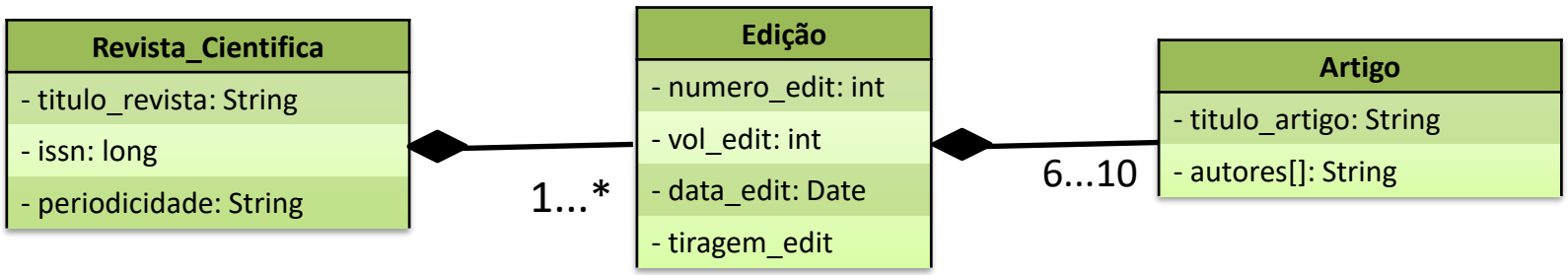


Exemplo do Livro: UML2 uma Abordagem prática.

Composição

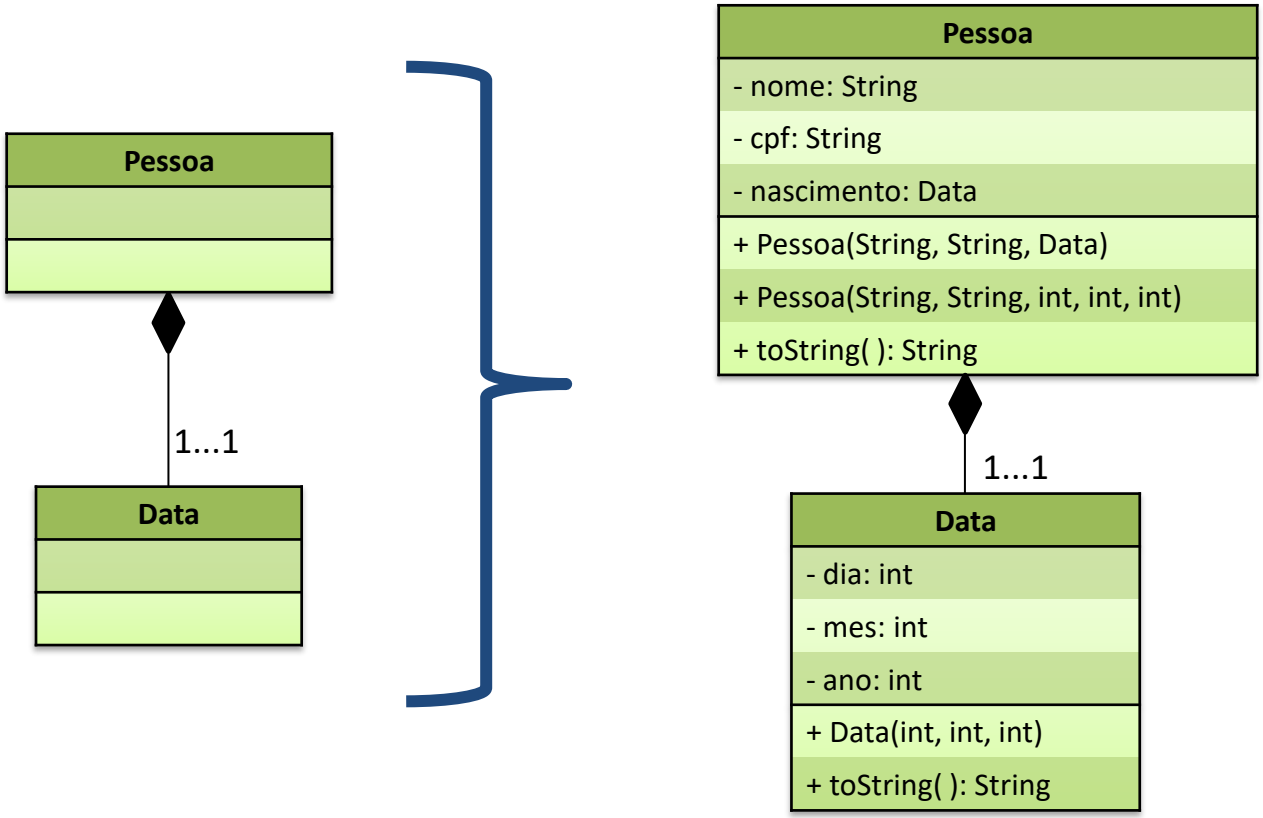
- A composição é uma forma restrita de agregação na qual duas classes são fortemente dependentes uma da outra, onde é apresentado um vínculo forte entre os objetos-todo e os objetos-parte.

- No exemplo abaixo vemos que uma Revista cientifica tem uma ou mais edições e essa edição está relacionada exclusivamente com a revista, e cada edição deverá conter de 6 a 10 artigos.

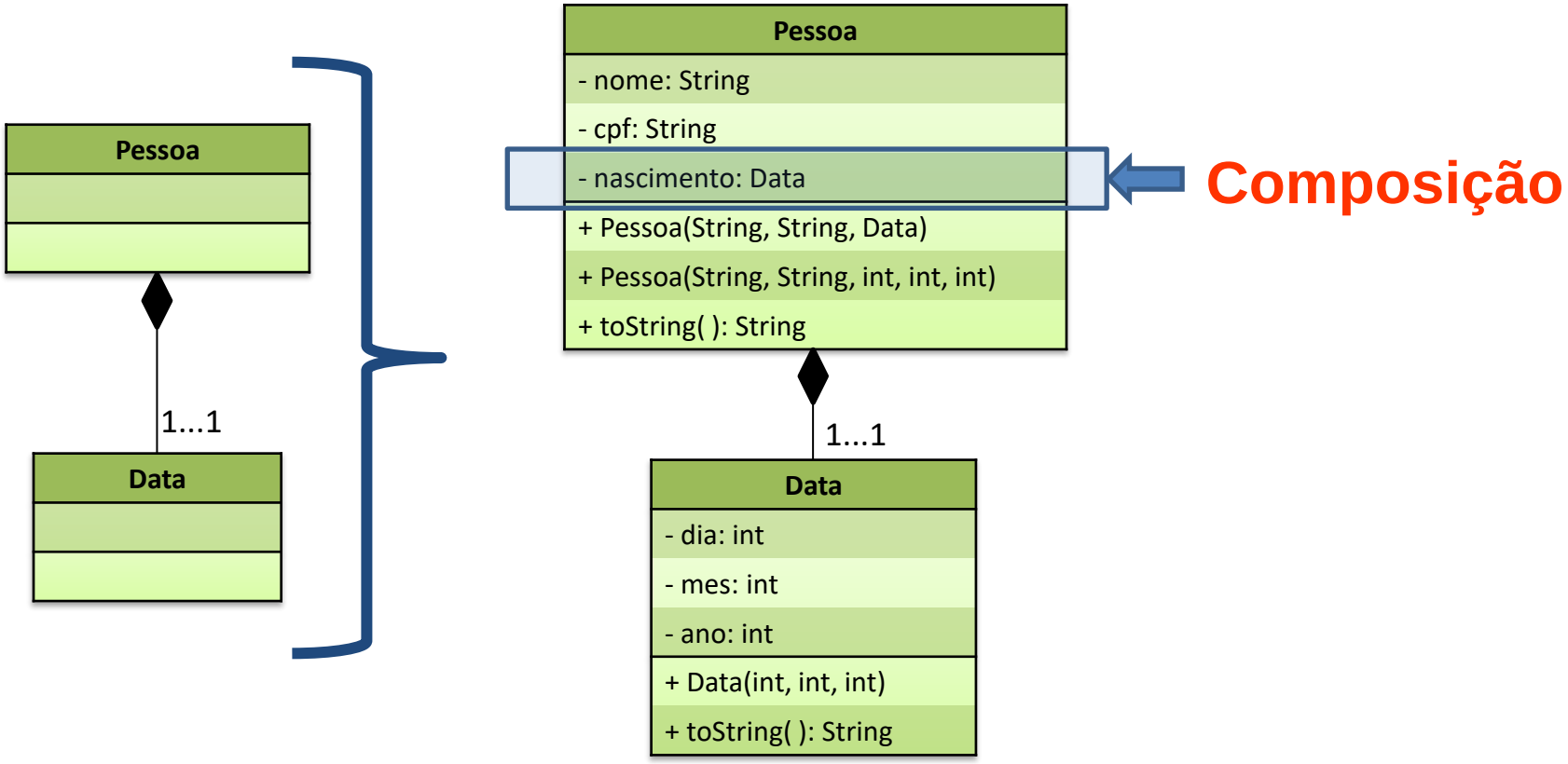


Exemplo do Livro: UML2 uma Abordagem prática.

Exemplo de Composição

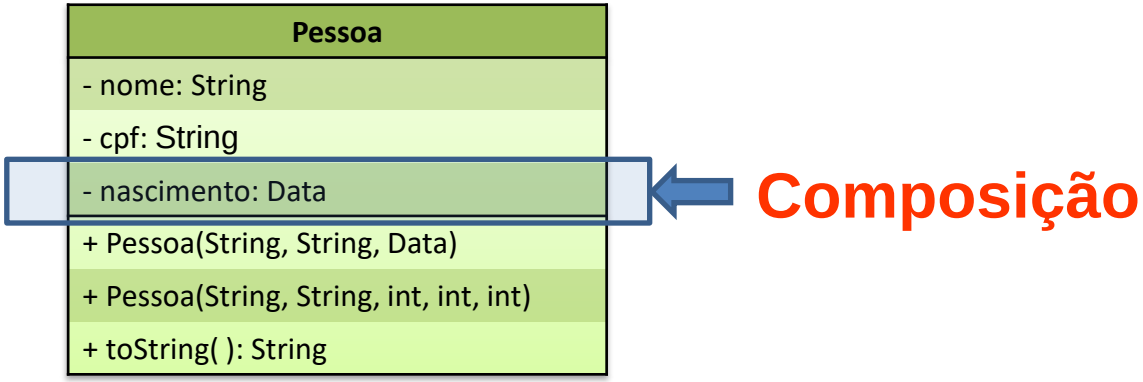


Exemplo de Composição



Composição

- A classe pode ter referências a objetos de outras classes como membros. Essa capacidade é chamada composição e é as vezes referida como um relacionamento **tem um.**
- Exemplo: A classe **Pessoa** precisa ter um objeto da classe **Data**.



Exemplo de Composição

```
public class Data {
        private int dia;
        private int mes;
        private int ano;
        // construtor da classe Data com três parâmetros
        public Data(int dia, int mês, int ano){
                this.dia = dia;
                this.mes = mês;
                this.ano = ano;
        }
```

Exemplo de Composição

```
        // retorna uma String com os atributos da classe Data
        public String toString() {
                return this.dia + "/" + this.mes + "/" + this.ano;
        }
} // fim da classe Data
```

Exemplo de Composição

```
public class Pessoa {
        private String nome;
        private String cpf;
        private Data nascimento;

        // construtor da classe Pessoa com três parâmetros usando um objeto Data
        public Pessoa(String nome, String cpf, Data nascimento){
                this.nome = nome;
                this.cpf = cpf;
                this.nascimento = nascimento;
        }
```

Exemplo de Composição

```
// construtor da classe Pessoa com cinco parâmetros
public Pessoa(String nome, long String, int dia, int mes, int ano){
        this.nome = nome;
        this.cpf = cpf;
        Data nascimento = new Data(dia, mes, ano);
        this.nascimento = nascimento;
}
```

Exemplo de Composição

```
        // retorna uma String com os atributos da classe Data
        public String toString() {
                return "Nome = " + this.nome + ", CPF = " + this.cpf +
                        ", Data de Nascimento = " + this.nascimento;
        }
} // fim da classe Pessoa
```

Exemplo de Composição

```
6   package composicao;
7   /**
8    * @author Isaac
9    */
10  public class Composicao {
11      /**
12       * @param args the command line arguments
13       */
14      public static void main(String[] args) {
15          // TODO code application logic here
16          Data nasc;
17          nasc = new Data(22,5,1994);
18          System.out.println("Data de  nascimento  =  "  +  nasc);
19          Pessoa  pess1 = new  Pessoa("Julia",  "12345678901", nasc);
20          System.out.println("Pessoa 1:  "  +  pess1);
21      }
22
23  }
```

```
run:
Data de  nascimento  =  22/5/1994
Pessoa 1:  Nome  =  Julia,  CPF  =  12345678901,  Data  de  Nascimento  =  22/5/1994
BUILD SUCCESSFUL (total time: 0 seconds)
```