Do Projeto à Codificação

Reginaldo Ré
reginaldo@utfpr.edu.br

Universidade Tecnológica Federal do Paraná

Análise e Projeto Orientados a Objetos
2018/2

Agenda

1. Diagrama de Classes de Projeto

Agenda

1. Diagrama de Classes de Projeto

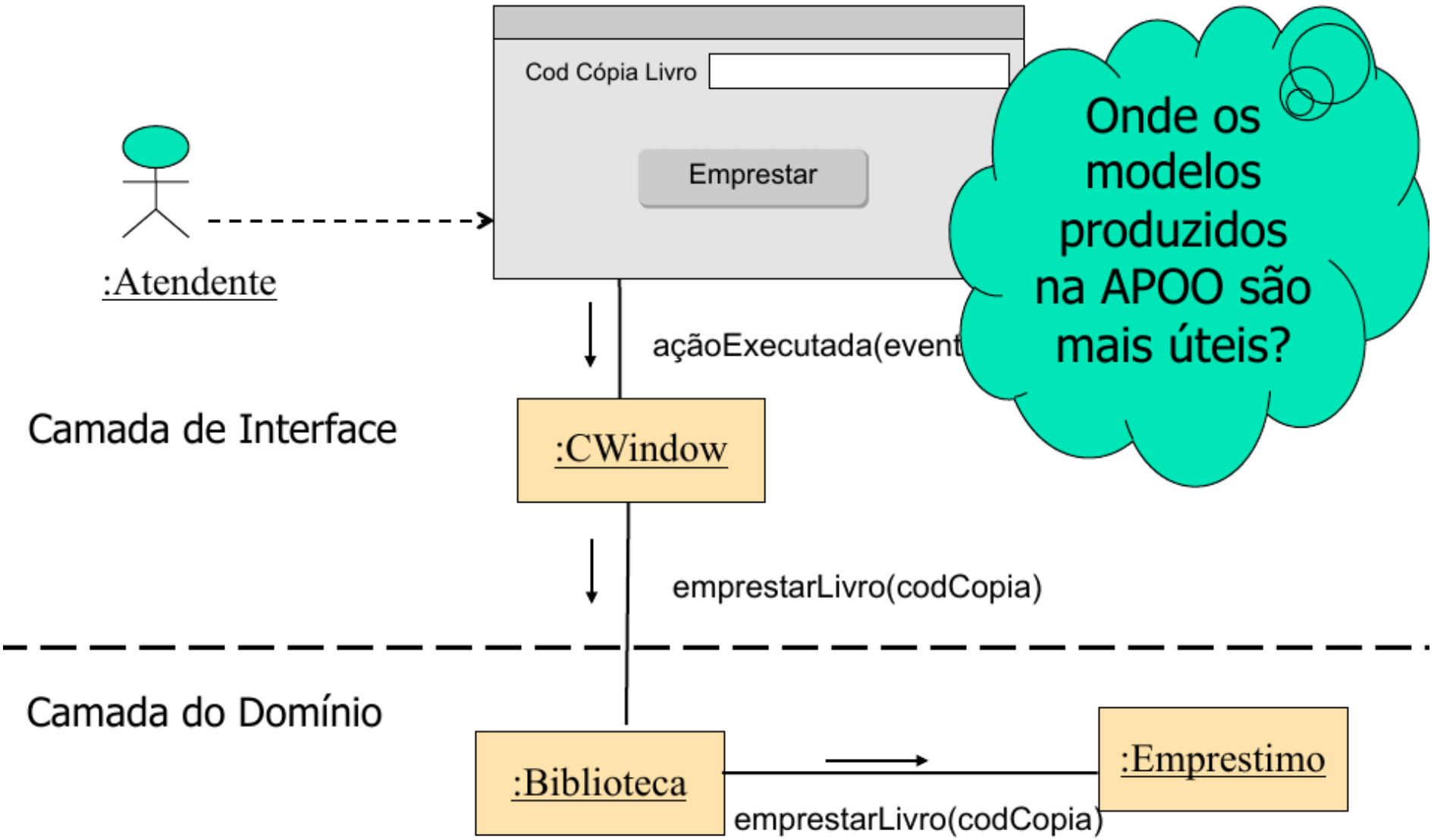
O que já vimos até agora

- Diagramas de Caso de Uso
- Casos de Uso Completo Abstrato
- Modelo Conceitual
  - Casos de Uso com substantivos e verbos sublinhados
- Diagrama de Sequencia do Sistema
  - Modelo Conceitual + Casos de Uso
  - Um para cada caso de uso
- Contratos de Operações do Sistema
  - Um contrato para cada operação do DSS
- Diagramas de Comunicação
  - Um diagrama de comunicação para cada operação do sistema
- Diagrama de Classes de Projeto
  - Diagramas de Comutação + Modelo Conceitual

Como mapear o projeto para a Implementação (1/2)

- Resultados obtidos no projeto são o ponto de partida, mas muito trabalho ainda tem que ser feito
- Muitas alterações podem ocorrer e problemas podem surgir e precisam ser solucionados
- Prepare-se: pode haver mudanças e desvios de projeto!!!
- Protótipos e código exploratório pode ter sido produzido na fase de projeto
- Ferramentas CASE (geração semi-automática do código) podem ajudar
- Código a ser escrito
  - Classes + Métodos
  - Interfaces
- Linguagem a ser usada como exemplo: JAVA

Como mapear o projeto para a Implementação (2/2)

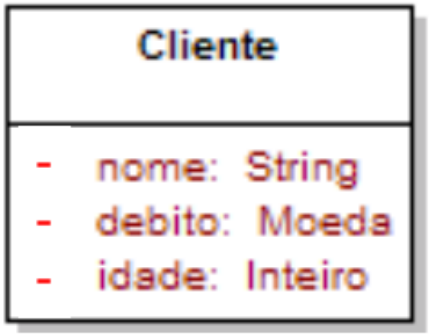


Resposta: Camada de domínio!!!

Definição de Classes

- Uma classe de programa deve ser criada para cada classe do Diagrama de Classes de Projeto
- Método Criar $\rightarrow$ gera construtores em Java, por exemplo
- Tipos de atributos $\rightarrow$ podem ser adotados tipos nativos da linguagem ou serem criados tipos a partir dos tipos nativos
- Definição e assinaturas dos métodos

Definição de Classes




```
class Cliente {
        private String nome;
        private Float debito;
        private Integer idade;
}
```

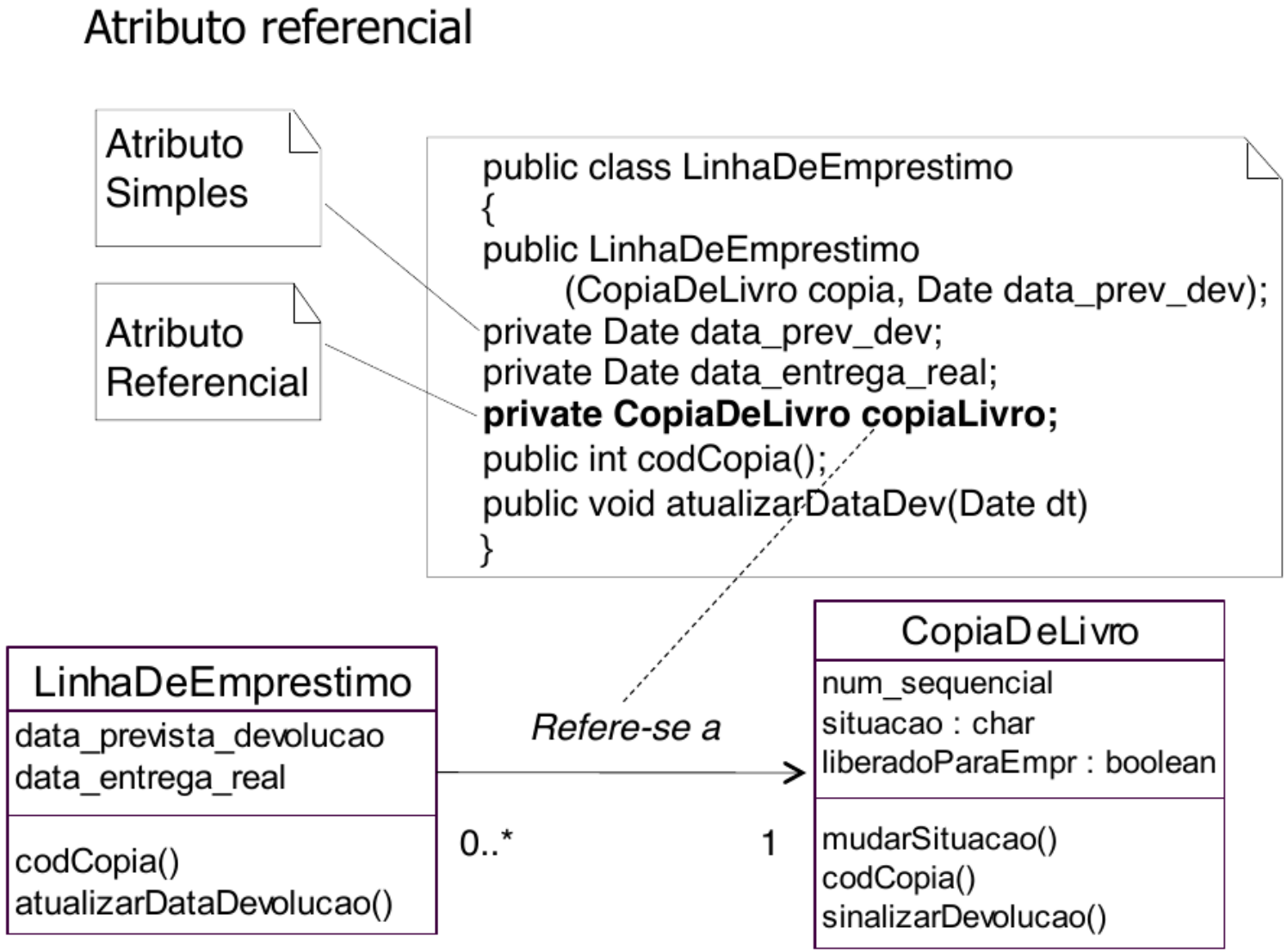
```
class Cliente {
    private String nome;
    private Float debito;
    private Integer idade;

    public void setNome(String nome) { this.nome = nome; }
    public void setDebito(Float debito) { this.debito = debito; }
    public void setIdade(Integer idade) { this.idade = idade; }
    public String getNome() { return nome; }
    public Float getDebito() { return debito; }
    public Integer getIdade() { return idade; }
}
```

Atributos Referenciais (1/2)

- São atributos que referenciam um outro objeto complexo e não um tipo primitivo
- São sugeridos pelas associações e pela navegabilidade no diagrama de classes de projeto
- No diagrama de classes, os atributos referenciais estão normalmente implícitos (ao invés de explícitos como os demais atributos)

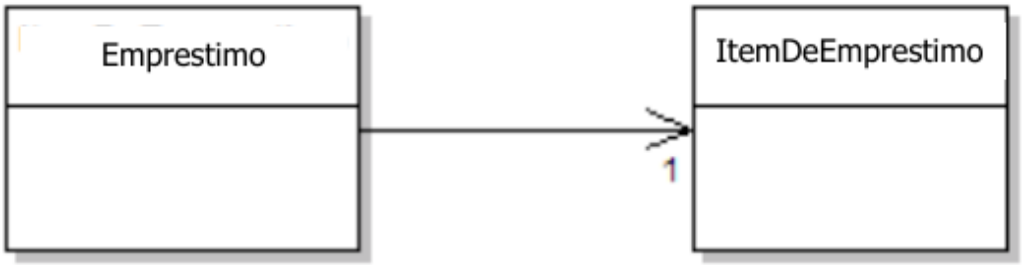
Atributos Referenciais (2/2)



Associação para 1 (1/2)

- Como associação é estritamente para 1
  - não é possível destruir a associação
  - método para destruir a associação não deve ser implementado
- Como a associação para 1 é obrigatória para o objeto na origem
  - método criador da classe deve ter como parâmetro o elemento a ser associado.

Associação para 1 (2/2)




```
class Emprestimo {
        private ItemDeEmprestimo item;

        public Emprestimo(ItemDeEmprestimo item) {
                this.associaItem(item)
        }
        public void associaItem(ItemDeEmprestimo item) {
                this.item = item;
        }
        public ItemDeEmprestimo getItem() {
                return item;
        }
}
```

Associação para 0..1 (1/2)

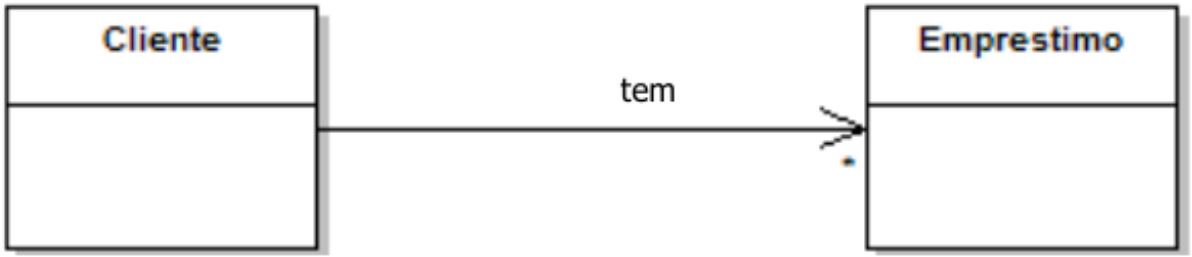
- É possível destruir a associação
  - deve ser implementado o método correspondente
- Não é necessário passar um objeto como parâmetro para o método criador, pois associação para 0..1 não é obrigatória

Associação para 0..1 (2/2)


```
class Venda {
        private Pagamento pagamento;

        public Venda() { }
        public void associaPagamento(Pagamento pagamento) {
                this.pagamento = pagamento;
        }
        public void desassociaPagamento() {
                this.pagamento = null;
        }
        public Pagamento getPagamento() {
                return pagamento;
        }
}
```

Associação para *

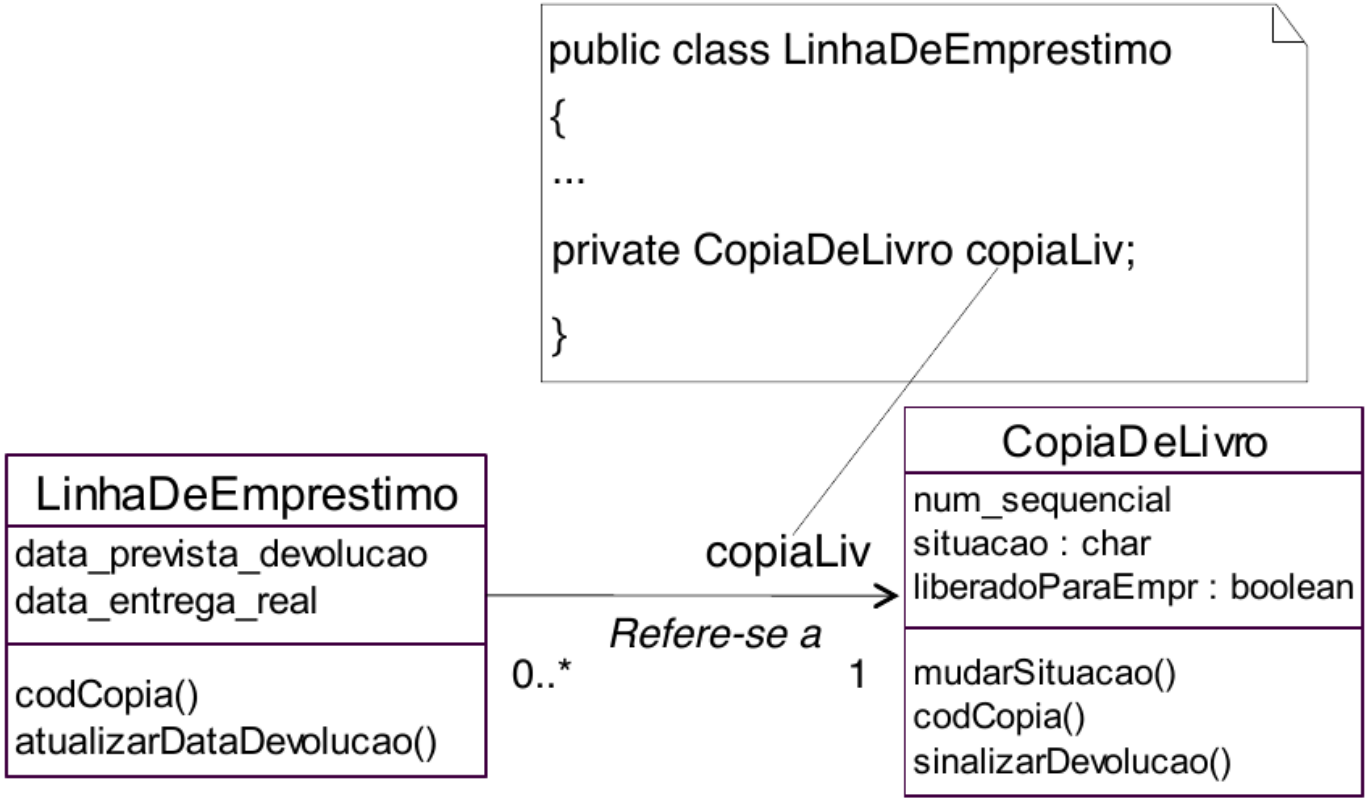



```
class Cliente {
        private Set emprestimos = new HashSet();

        public Cliente () { }
        public void adicionaEmprestimo(Emprestimo emprestimo) {
                this.emprestimos.add(emprestimo);
        }
        public void removeEmprestimo(Emprestimo emprestimo) {
                this.emprestimos.remove(emprestimo);
        }
        public Set getEmprestimos () {
                return Collections.unmodifiableSet(emprestimos);
        }
}
```

Atributos Referenciais e Nomes de Papéis

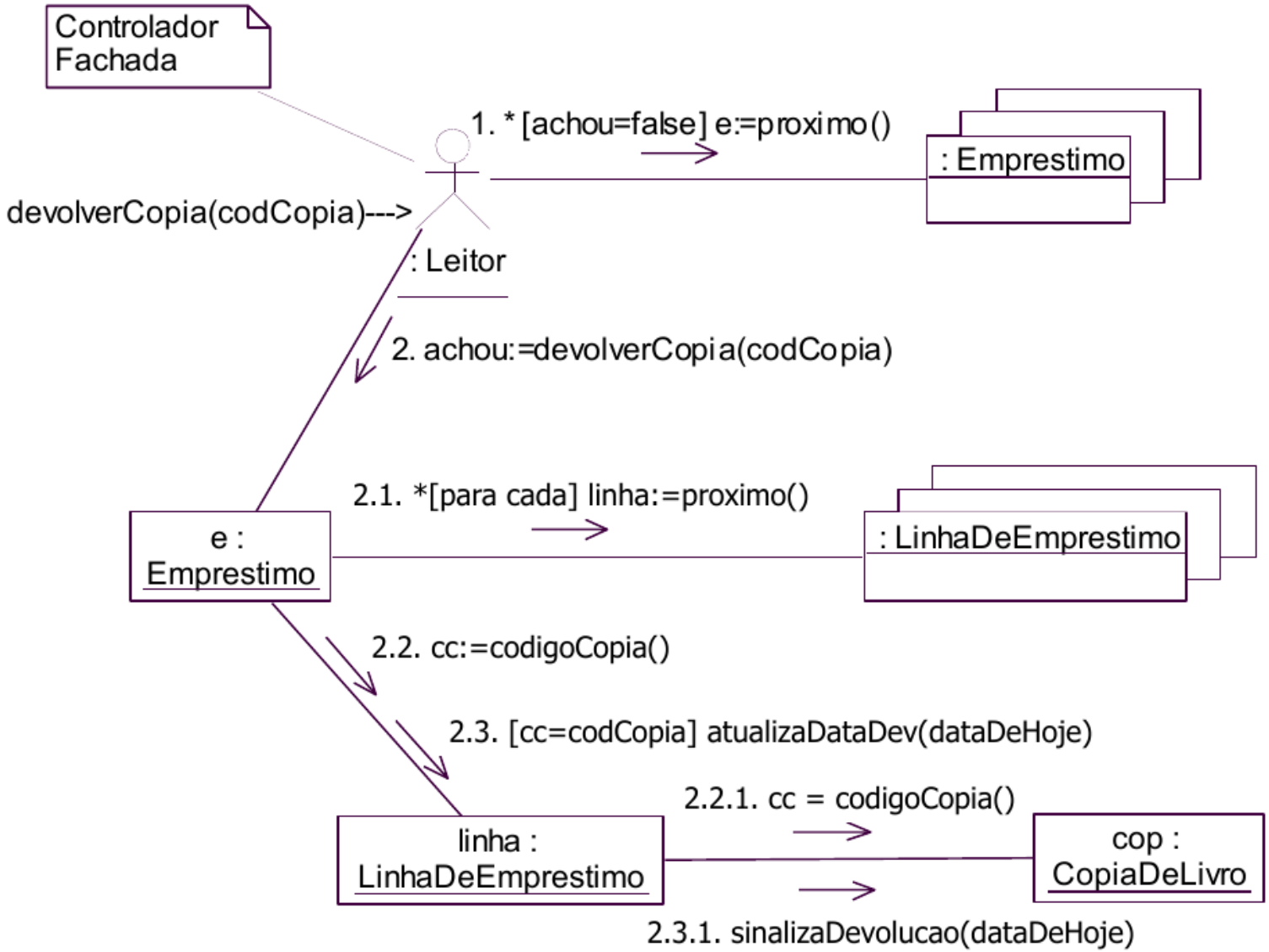
- O nome de papel identifica o papel da classe na associação e fornece frequentemente algum contexto semântico sobre a sua natureza.
- Se houver um nome de papel no diagrama de classes de projeto, utilize-o como base para o nome do atributo referencial durante a geração de código.



Criar Métodos a Partir de Diagramas de Comunicação (1/2)

- Sequência de mensagens de um diagrama de comunicação é traduzida para uma série de comandos de programação na definição do método.
- Métodos de acesso (set e get) não serão ilustrados, por simplicidade, mas devem ser criados.
- Exemplo:
  - A classe Emprestimo no diagrama de comunicação do método (operação) devolverCopia.

Criar Métodos a Partir de Diagramas de Comunicação (2/2)



Criar Métodos a Partir de Diagramas de Comunicação

Classe Leitor

```
public class Leitor {
   private String nome;
   private Char[] tipo;
   private Boolean achou=false;
   private List emprestimos = new ArrayList();
   public void devolverCopia(int codCopia) {
      Iterator i = emprestimos.iterator();
      while (i.hasNext()) && (!achou) {
         Emprestimo e = (Emprestimo) i.next();
         achou=e.devolverCopia(codCopia)}
      }
   }
}
```

Criar Métodos a Partir de Diagramas de Comunicação

Classe Emprestimo

```
public class Emprestimo {
   private Date data_de_emprestimo;
   private Char[] situacao;
   private int cc=0;
   private List linhas = new ArrayList();
   public Boolean devolverCopia(int codCopia) {
      private Boolean ach = false;
      Iterator i = linhas.iterator();
      Date dataDeHoje = new Date();
      while (i.hasNext()) && (!ach) {
         LinhaDeEmprestimo linha = (LinhaDeEmprestimo) i.next();
         cc=linha.codigoCopia();
         if (cc==codCopia) {
            linha.atualizaDataDev(dataDeHoje);
            ach := true;
         }
      }
      return ach;
}
```

Exceções e Tratamento de Erros

- Até aqui...
  - tratamento de erros foi ignorado (intencionalmente)
  - mas deve ser considerado na fase de projeto em sistemas reais
- Exemplo:
  - os contratos podem ser anotados com observações sobre situações típicas de erros e o plano geral de tratamento desses erros.
- A UML não tem uma notação especial para ilustrar exceções

Ordem de Implementação

- Podem ser implementadas e testadas na seguinte ordem:
  - Das classes com acoplamento mais baixo para as classes com acoplamento mais alto
- Exemplo:
  - Começar por Livro ou Leitor. Em seguida, as classes que dependem de implementações prévias: Emprestimo ou LinhaDeEmprestimo, e assim por diante.