MODELAGEM DE SOFTWARE

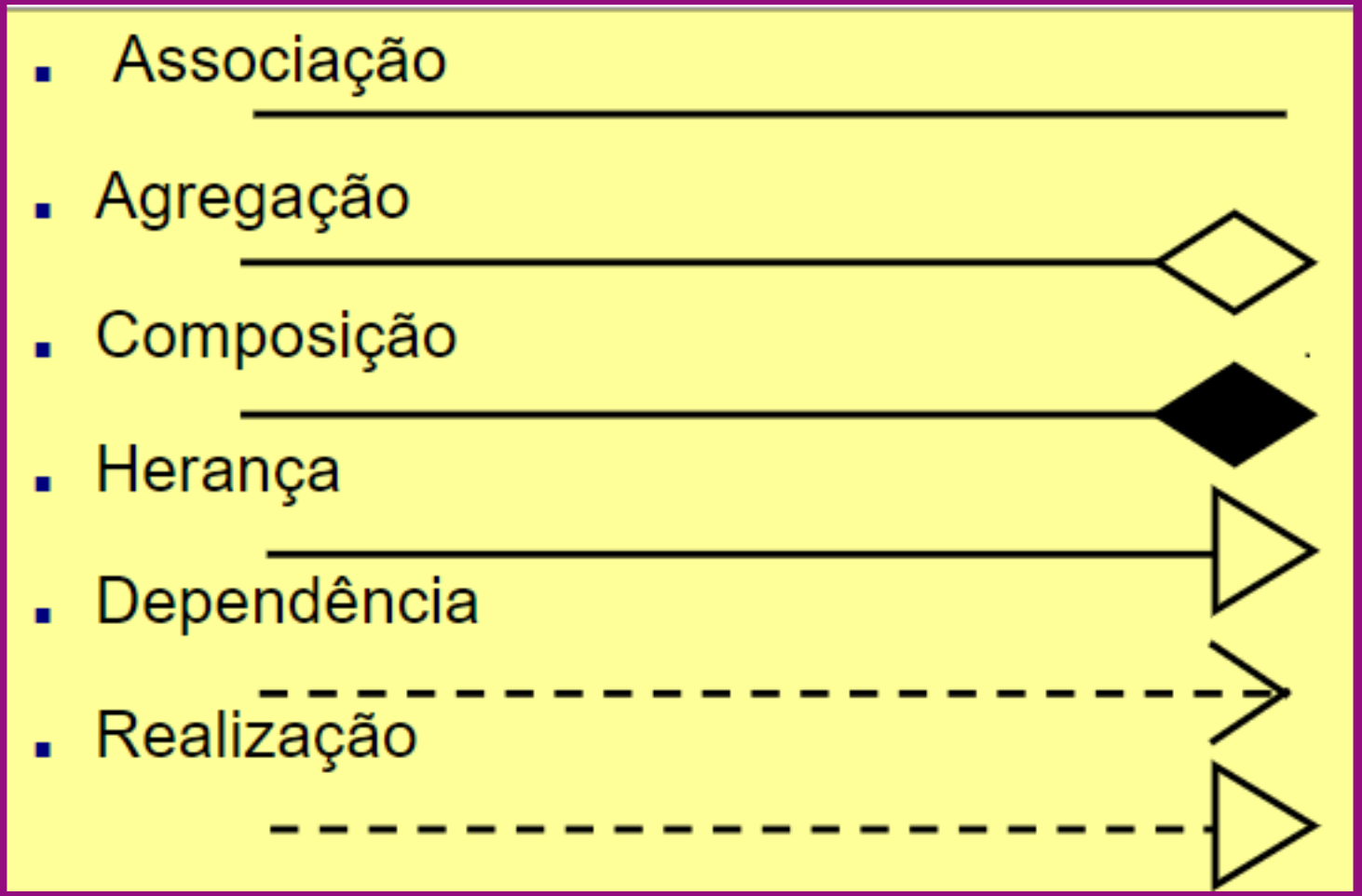
Prof. Richard Henrique de Souza
richard.souza@animaeducacao.com.br

Prof. Ricardo Ribeiro Assink
ricardo.assink@unisul.br

Prof. Rafael Lessa
rafael.lessa@unisul.br

Representação dos Relacionamentos

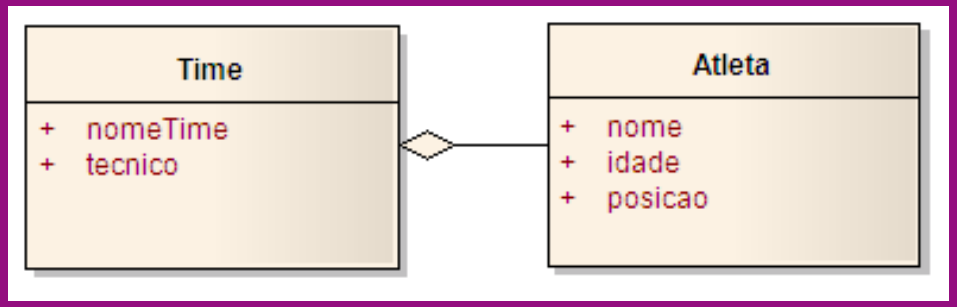
- Associação
- Agregação
- Composição
- Herança
- Dependência
- Realização



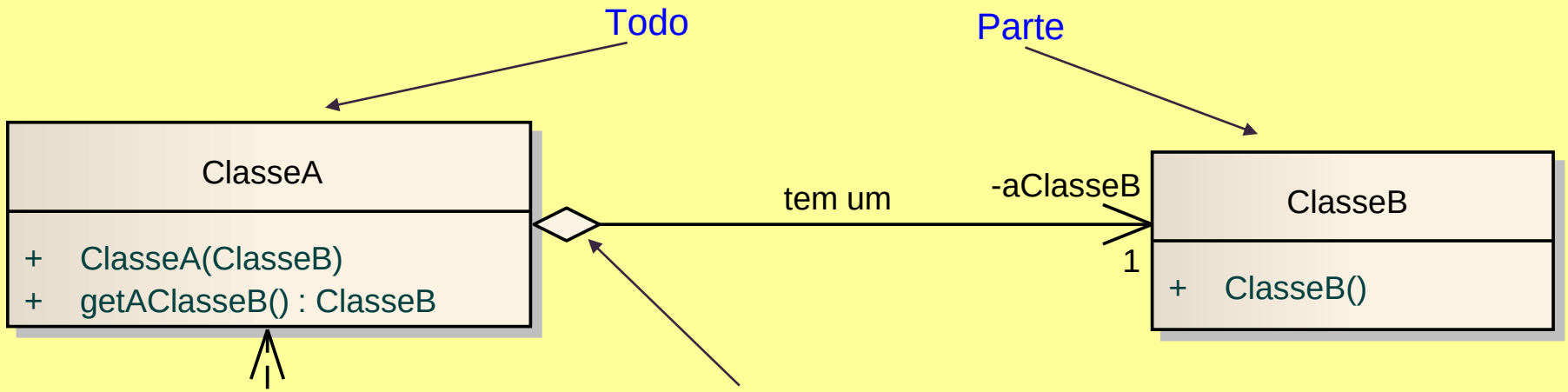
Conceitos da POO: agregação e composição

Agregação e Composição: um objeto todo é relacionado com suas partes (relacionamento “parte-de” ou “contém”)

Na **Agregação**, a existência do Objeto-Parte faz sentido, mesmo não existindo o Objeto-Todo



Agregação *tem um*



Relacionamento Agregação(Losango Aberto)

Observações:
-Sempre o relacionamento será unidirecional já que a Parte não pode estar em mais de um Todo;
-Pode-se omitir a seta pois está implícito;
-Sempre a multiplicidade do lado navegável/Parte será no mínimo 1.

Implementação Agregação *tem um*

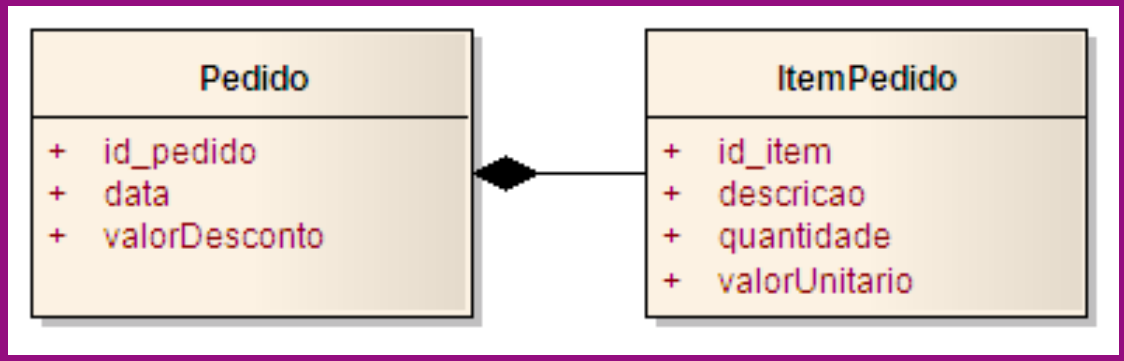
```
public class ClasseA {
   private ClasseB aClasseB; //Nome link

   //A Parte é recebida como parâmetro para se juntar ao
   //Todo
   //A Parte pode estar sendo utilizada por outro Todo
   public ClasseA(ClasseB b) {
      aClasseB = b;
   }
   //Permite recuperar a Parte para compartilhar com outro
   //Todo
   public ClasseB getAClasseB(){
     return aClasseB;
   }
}
```

```
public class ClasseB {

   public ClasseB(){

   }
}
```

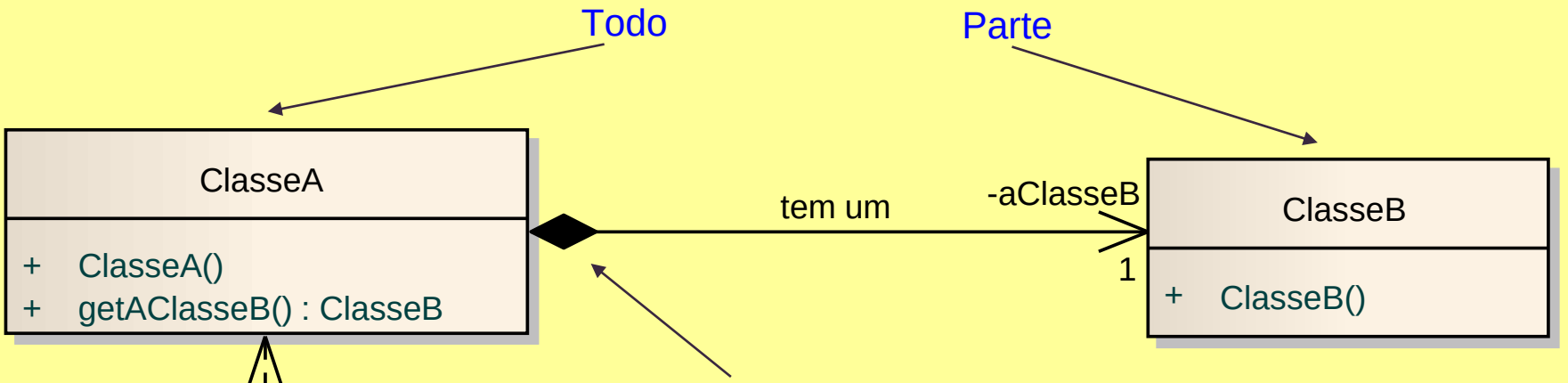
```
public class Principal {
   public static void main(String args[]){
      ClasseA a1 = new ClasseA(new ClasseB()); //Construtor com argumento criou a Parte
      ClasseA a2 = new ClasseA(a1.getAClasseB()); //Construtor com argumento recebeu a
                                                  // Parte de outro Todo
   }
}
```

Conceitos da POO: agregação e composição

Já a **Composição** é uma agregação mais forte; nela, a existência do Objeto-Parte NÃO faz sentido se o Objeto-Todo não existir.



Composição *tem um*



Relacionamento Composição(Losango Fechado)

Observações:
-Sempre o relacionamento será unidirecional já que a Parte não pode estar em mais de um Todo;
-Pode-se omitir a seta pois está implícito;
- Sempre a multiplicidade do lado navegável/Parte será no mínimo 1.

Implementação Composição *tem um*

```
public class ClasseA {

    private ClasseB aClasseB; //Nome do link

   //A Parte nasce junto com o Todo
   //A Parte não pode ser compartilhada com outro Todo
    public ClasseA() {
        aClasseB = new ClasseB();
    }
   //Pode recuperar a Parte antes da morte do Todo
    public ClasseB getAClasseB(){
      return aClasseB;
    }

}
```

```
public class ClasseB {

    public ClasseB(){

    }
}
```

```
public class Principal {
   public static void main(String args[]){
          ClasseA a1 = new ClasseA(); //Construtor sem argumento criou a Parte
   }
}
```

Diagrama de classes

O diagrama de classes demonstra a estrutura estática das classes de um sistema. É considerado estático já que a estrutura descrita é sempre válida em qualquer ponto do ciclo de vida do sistema.

Classes podem se relacionar com outras através de diversas maneiras: associação, dependência, especialização, ou em pacotes (classes agrupadas por características similares)

Modelos baseados em POO

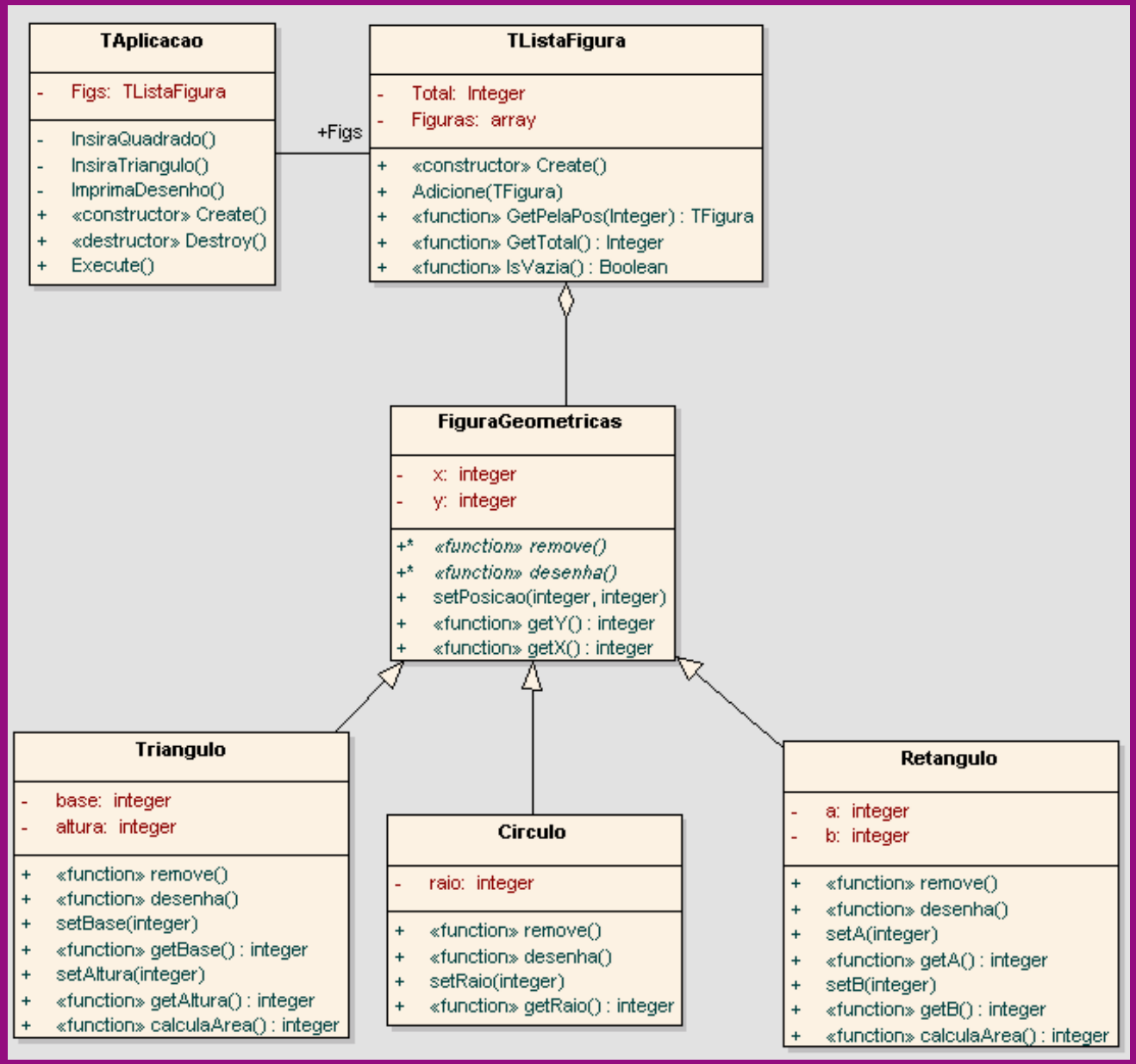


Diagrama de classes

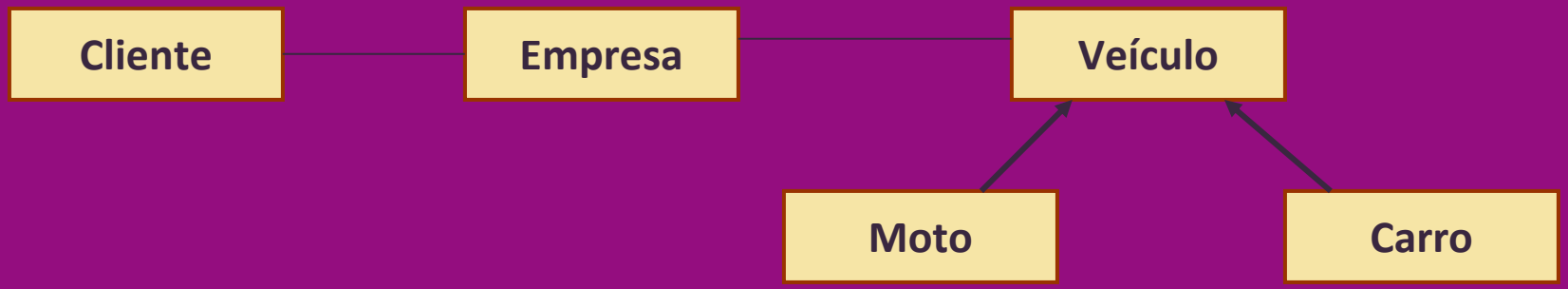


Diagrama de classes

Uma classe é a definição de atributos, operações e semântica de um conjunto de objetos. Todos os objetos desta classe correspondem a esta definição.

Classes são representadas por:

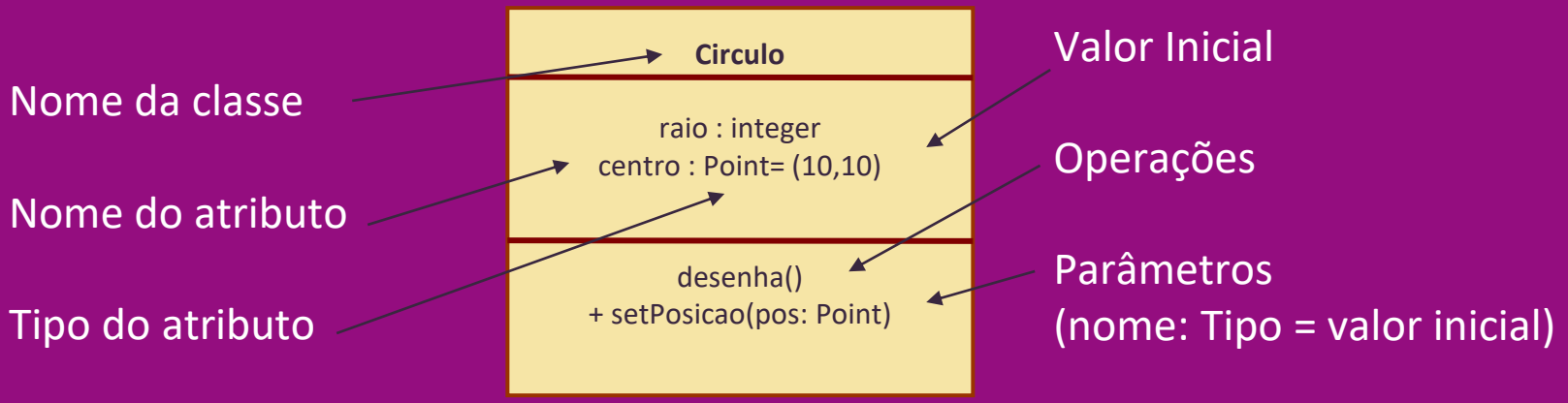


Diagrama de classes

Objetos são as instâncias de classe.

Atributos de classe são informações compartilhadas por todos os objetos da classe.

Atributos de objeto são informações específicas do objeto, usado para definir seu estado.

Operações são serviços que podem ser requeridos de um objeto Sendo descrito por sua assinatura (nome da operação)

Vamos trabalhar?

Exercício

- Vamos voltar no diagrama de Pessoa, Aluno e Professor, Disciplina e Turma.
- Vamos adicionar Titulação do Professor, Avaliação (nota) e endereço.
- Vamos mapear as associações e as cardinalidades.

Referência bibliográfica

FOWLER, Martin e SCOTT, Kendall. Uml Essencial. 2a. Edição. Bookman. Porto Alegre, 2000.

SCHNEIDER, Geri. Applying use case: a practical guide. Addison-Wesley, 1998.

OESTEREICH, Bernd. Developing Software with UML. Addison-Wesley, 1999.

CRÉDITOS

COORDENAÇÃO

Vera Rejane Niedersberg Schuhmacher

PROFESSORES

Rafael Lessa
Daniella Vieira