

Projeto de Programas

Paulo Torrens

paulotorrens@gnu.org

Departamento de Ciência da Computação
Centro de Ciências e Tecnologias
Universidade do Estado de Santa Catarina

2020/1

Modelo Entidade-Relacionamento

- Ao projetar um software, é necessária consideração sobre o armazenamento das informações necessárias para a execução
- Em muitos projetos, especialmente os de maior porte, as informações são salvas através de um software externo, o **banco de dados**
- Embora existam outras variações, bancos de dados costumam funcionar através de **relacionamento** entre **entidades**
 - Entidades são representadas através de **tabelas**
 - Bancos que não usam o modelo relacional são tradicionalmente chamados de bancos NoSQL
 - A linguagem **SQL** (*Structured Query Language*) é uma linguagem padronizada para manipulação de dados relacionais

Modelo Entidade-Relacionamento

- Ao projetar um software, é necessária consideração sobre o armazenamento das informações necessárias para a execução
- Em muitos projetos, especialmente os de maior porte, as informações são salvas através de um software externo, o **banco de dados**
- Embora existam outras variações, bancos de dados costumam funcionar através de **relacionamento** entre **entidades**
 - Entidades são representadas através de **tabelas**
 - Bancos que não usam o modelo relacional são tradicionalmente chamados de bancos NoSQL
 - A linguagem **SQL** (*Structured Query Language*) é uma linguagem padronizada para manipulação de dados relacionais

Modelo Entidade-Relacionamento

- Ao projetar um software, é necessária consideração sobre o armazenamento das informações necessárias para a execução
- Em muitos projetos, especialmente os de maior porte, as informações são salvas através de um software externo, o **banco de dados**
- Embora existam outras variações, bancos de dados costumam funcionar através de **relacionamento** entre **entidades**
 - Entidades são representadas através de **tabelas**
 - Bancos que não usam o modelo relacional são tradicionalmente chamados de bancos NoSQL
 - A linguagem **SQL** (*Structured Query Language*) é uma linguagem padronizada para manipulação de dados relacionais

Modelo Entidade-Relacionamento

- Ao projetar um software, é necessária consideração sobre o armazenamento das informações necessárias para a execução
- Em muitos projetos, especialmente os de maior porte, as informações são salvas através de um software externo, o **banco de dados**
- Embora existam outras variações, bancos de dados costumam funcionar através de **relacionamento** entre **entidades**
 - Entidades são representadas através de **tabelas**
 - Bancos que não usam o modelo relacional são tradicionalmente chamados de bancos NoSQL
 - A linguagem **SQL** (*Structured Query Language*) é uma linguagem padronizada para manipulação de dados relacionais

Modelo Entidade-Relacionamento

- Ao projetar um software, é necessária consideração sobre o armazenamento das informações necessárias para a execução
- Em muitos projetos, especialmente os de maior porte, as informações são salvas através de um software externo, o **banco de dados**
- Embora existam outras variações, bancos de dados costumam funcionar através de **relacionamento** entre **entidades**
 - Entidades são representadas através de **tabelas**
 - Bancos que não usam o modelo relacional são tradicionalmente chamados de bancos NoSQL
 - A linguagem **SQL** (*Structured Query Language*) é uma linguagem padronizada para manipulação de dados relacionais

Modelo Entidade-Relacionamento

- Ao projetar um software, é necessária consideração sobre o armazenamento das informações necessárias para a execução
- Em muitos projetos, especialmente os de maior porte, as informações são salvas através de um software externo, o **banco de dados**
- Embora existam outras variações, bancos de dados costumam funcionar através de **relacionamento** entre **entidades**
 - Entidades são representadas através de **tabelas**
 - Bancos que não usam o modelo relacional são tradicionalmente chamados de bancos NoSQL
 - A linguagem **SQL** (*Structured Query Language*) é uma linguagem padronizada para manipulação de dados relacionais

Modelo Entidade-Relacionamento

- Uma forma de representar modelos relacionais é através do **diagrama ER**
- Conceitualmente, há três tipos de informação necessárias para a representação de um modelo:
 - **Entidades**, representando objetos de interesse para o negócio, como Cliente, Livro, Empréstimo, etc
 - **Atributos**, que são campos de uma entidade, salvando uma informação relevante a ela; por exemplo, um Cliente deve ter um nome, e-mail, endereço, etc
 - **Relacionamentos**, que são formas de se “navegar” dentro dos dados, e representam relações entre as entidades existentes; um relacionamento geralmente é representado por um verbo: um Cliente efetua um Empréstimo, um Empréstimo contém um Livro, etc

Modelo Entidade-Relacionamento

- Uma forma de representar modelos relacionais é através do **diagrama ER**
- Conceitualmente, há três tipos de informação necessárias para a representação de um modelo:
 - **Entidades**, representando objetos de interesse para o negócio, como Cliente, Livro, Empréstimo, etc
 - **Atributos**, que são campos de uma entidade, salvando uma informação relevante a ela; por exemplo, um Cliente deve ter um nome, e-mail, endereço, etc
 - **Relacionamentos**, que são formas de se “navegar” dentro dos dados, e representam relações entre as entidades existentes; um relacionamento geralmente é representado por um verbo: um Cliente efetua um Empréstimo, um Empréstimo contém um Livro, etc

Modelo Entidade-Relacionamento

- Uma forma de representar modelos relacionais é através do **diagrama ER**
- Conceitualmente, há três tipos de informação necessárias para a representação de um modelo:
 - **Entidades**, representando objetos de interesse para o negócio, como Cliente, Livro, Empréstimo, etc
 - **Atributos**, que são campos de uma entidade, salvando uma informação relevante a ela; por exemplo, um Cliente deve ter um nome, e-mail, endereço, etc
 - **Relacionamentos**, que são formas de se “navegar” dentro dos dados, e representam relações entre as entidades existentes; um relacionamento geralmente é representado por um verbo: um Cliente efetua um Empréstimo, um Empréstimo contém um Livro, etc

Modelo Entidade-Relacionamento

- Uma forma de representar modelos relacionais é através do **diagrama ER**
- Conceitualmente, há três tipos de informação necessárias para a representação de um modelo:
 - **Entidades**, representando objetos de interesse para o negócio, como Cliente, Livro, Empréstimo, etc
 - **Atributos**, que são campos de uma entidade, salvando uma informação relevante a ela; por exemplo, um Cliente deve ter um nome, e-mail, endereço, etc
 - **Relacionamentos**, que são formas de se “navegar” dentro dos dados, e representam relações entre as entidades existentes; um relacionamento geralmente é representado por um verbo: um Cliente efetua um Empréstimo, um Empréstimo contém um Livro, etc

Modelo Entidade-Relacionamento

- Uma forma de representar modelos relacionais é através do **diagrama ER**
- Conceitualmente, há três tipos de informação necessárias para a representação de um modelo:
 - **Entidades**, representando objetos de interesse para o negócio, como Cliente, Livro, Empréstimo, etc
 - **Atributos**, que são campos de uma entidade, salvando uma informação relevante a ela; por exemplo, um Cliente deve ter um nome, e-mail, endereço, etc
 - **Relacionamentos**, que são formas de se “navegar” dentro dos dados, e representam relações entre as entidades existentes; um relacionamento geralmente é representado por um verbo: um Cliente efetua um Empréstimo, um Empréstimo contém um Livro, etc

Modelo Entidade-Relacionamento

- O intuito do diagrama ER é representar, **com níveis variados de precisão**, as necessidades das informações de um sistema
 - Muitas vezes, cabe ao programador ou DBA (*Database Administrator*) implementar o banco de dados baseando-se no diagrama
- Diagramas ER não apresentam um padrão efetivo, porém possuem algumas convenções que facilitam o entendimento
 - Entidades são representadas através de retângulos
 - Atributos são ou representados através de círculos conectados à entidade, ou apresentados dentro dela em forma similar a um diagrama UML
 - Relacionamentos são representados através de losangos conectados às entidades que relacionam

Modelo Entidade-Relacionamento

- O intuito do diagrama ER é representar, **com níveis variados de precisão**, as necessidades das informações de um sistema
 - Muitas vezes, cabe ao programador ou DBA (*Database Administrator*) implementar o banco de dados baseando-se no diagrama
- Diagramas ER não apresentam um padrão efetivo, porém possuem algumas convenções que facilitam o entendimento
 - Entidades são representadas através de retângulos
 - Atributos são ou representados através de círculos conectados à entidade, ou apresentados dentro dela em forma similar a um diagrama UML
 - Relacionamentos são representados através de losangos conectados às entidades que relacionam

Modelo Entidade-Relacionamento

- O intuito do diagrama ER é representar, **com níveis variados de precisão**, as necessidades das informações de um sistema
 - Muitas vezes, cabe ao programador ou DBA (*Database Administrator*) implementar o banco de dados baseando-se no diagrama
- Diagramas ER não apresentam um padrão efetivo, porém possuem algumas convenções que facilitam o entendimento
 - Entidades são representadas através de retângulos
 - Atributos são ou representados através de círculos conectados à entidade, ou apresentados dentro dela em forma similar a um diagrama UML
 - Relacionamentos são representados através de losangos conectados às entidades que relacionam

Modelo Entidade-Relacionamento

- O intuito do diagrama ER é representar, **com níveis variados de precisão**, as necessidades das informações de um sistema
 - Muitas vezes, cabe ao programador ou DBA (*Database Administrator*) implementar o banco de dados baseando-se no diagrama
- Diagramas ER não apresentam um padrão efetivo, porém possuem algumas convenções que facilitam o entendimento
 - Entidades são representadas através de retângulos
 - Atributos são ou representados através de círculos conectados à entidade, ou apresentados dentro dela em forma similar a um diagrama UML
 - Relacionamentos são representados através de losangos conectados às entidades que relacionam

Modelo Entidade-Relacionamento

- O intuito do diagrama ER é representar, **com níveis variados de precisão**, as necessidades das informações de um sistema
 - Muitas vezes, cabe ao programador ou DBA (*Database Administrator*) implementar o banco de dados baseando-se no diagrama
- Diagramas ER não apresentam um padrão efetivo, porém possuem algumas convenções que facilitam o entendimento
 - Entidades são representadas através de retângulos
 - Atributos são ou representados através de círculos conectados à entidade, ou apresentados dentro dela em forma similar a um diagrama UML
 - Relacionamentos são representados através de losangos conectados às entidades que relacionam

Modelo Entidade-Relacionamento

- O intuito do diagrama ER é representar, **com níveis variados de precisão**, as necessidades das informações de um sistema
 - Muitas vezes, cabe ao programador ou DBA (*Database Administrator*) implementar o banco de dados baseando-se no diagrama
- Diagramas ER não apresentam um padrão efetivo, porém possuem algumas convenções que facilitam o entendimento
 - Entidades são representadas através de retângulos
 - Atributos são ou representados através de círculos conectados à entidade, ou apresentados dentro dela em forma similar a um diagrama UML
 - Relacionamentos são representados através de losangos conectados às entidades que relacionam

Modelo Entidade-Relacionamento

- Alguns atributos especiais podem ser classificados como **chaves primárias**, geralmente sublinhados no diagrama
 - A ideia de uma chave primária é **identificar** um elemento daquela entidade, sendo um valor único; por exemplo, um Cliente pode ser identificado por um UUID ou por seu CPF
- Relacionamentos possuem uma **cardinalidade**
 - Ao se criar uma relação entre entidades A e B, é necessário informar quantos A se relacionam a B, e quantos B se relacionam a A
 - Cada lado de uma relação possui uma cardinalidade, podendo ser **opcional** (zero ou um), **única** (apenas um), **múltipla** (zero ou mais), etc
 - Um para muitos, muitos para muitos, etc
 - Por exemplo, tendo **Cliente 1** \longleftrightarrow **N Empréstimo** como uma relação, um cliente pode efetuar múltiplos empréstimos, mas cada empréstimo registrado é atrelado a apenas um cliente

Modelo Entidade-Relacionamento

- Alguns atributos especiais podem ser classificados como **chaves primárias**, geralmente sublinhados no diagrama
 - A ideia de uma chave primária é **identificar** um elemento daquela entidade, sendo um valor único; por exemplo, um Cliente pode ser identificado por um UUID ou por seu CPF
- Relacionamentos possuem uma **cardinalidade**
 - Ao se criar uma relação entre entidades A e B, é necessário informar quantos A se relacionam a B, e quantos B se relacionam a A
 - Cada lado de uma relação possui uma cardinalidade, podendo ser **opcional** (zero ou um), **única** (apenas um), **múltipla** (zero ou mais), etc
 - Um para muitos, muitos para muitos, etc
 - Por exemplo, tendo **Cliente 1** \longleftrightarrow **N Empréstimo** como uma relação, um cliente pode efetuar múltiplos empréstimos, mas cada empréstimo registrado é atrelado a apenas um cliente

Modelo Entidade-Relacionamento

- Alguns atributos especiais podem ser classificados como **chaves primárias**, geralmente sublinhados no diagrama
 - A ideia de uma chave primária é **identificar** um elemento daquela entidade, sendo um valor único; por exemplo, um Cliente pode ser identificado por um UUID ou por seu CPF
- Relacionamentos possuem uma **cardinalidade**
 - Ao se criar uma relação entre entidades A e B, é necessário informar quantos A se relacionam a B, e quantos B se relacionam a A
 - Cada lado de uma relação possui uma cardinalidade, podendo ser **opcional** (zero ou um), **única** (apenas um), **múltipla** (zero ou mais), etc
 - Um para muitos, muitos para muitos, etc
 - Por exemplo, tendo **Cliente 1** \longleftrightarrow **N Empréstimo** como uma relação, um cliente pode efetuar múltiplos empréstimos, mas cada empréstimo registrado é atrelado a apenas um cliente

Modelo Entidade-Relacionamento

- Alguns atributos especiais podem ser classificados como **chaves primárias**, geralmente sublinhados no diagrama
 - A ideia de uma chave primária é **identificar** um elemento daquela entidade, sendo um valor único; por exemplo, um Cliente pode ser identificado por um UUID ou por seu CPF
- Relacionamentos possuem uma **cardinalidade**
 - Ao se criar uma relação entre entidades A e B, é necessário informar quantos A se relacionam a B, e quantos B se relacionam a A
 - Cada lado de uma relação possui uma cardinalidade, podendo ser **opcional** (zero ou um), **única** (apenas um), **múltipla** (zero ou mais), etc
 - Um para muitos, muitos para muitos, etc
 - Por exemplo, tendo **Cliente 1** \longleftrightarrow **N Empréstimo** como uma relação, um cliente pode efetuar múltiplos empréstimos, mas cada empréstimo registrado é atrelado a apenas um cliente

Modelo Entidade-Relacionamento

- Alguns atributos especiais podem ser classificados como **chaves primárias**, geralmente sublinhados no diagrama
 - A ideia de uma chave primária é **identificar** um elemento daquela entidade, sendo um valor único; por exemplo, um Cliente pode ser identificado por um UUID ou por seu CPF
- Relacionamentos possuem uma **cardinalidade**
 - Ao se criar uma relação entre entidades A e B, é necessário informar quantos A se relacionam a B, e quantos B se relacionam a A
 - Cada lado de uma relação possui uma cardinalidade, podendo ser **opcional** (zero ou um), **única** (apenas um), **múltipla** (zero ou mais), etc
 - Um para muitos, muitos para muitos, etc
 - Por exemplo, tendo **Cliente 1** \longleftrightarrow **N Empréstimo** como uma relação, um cliente pode efetuar múltiplos empréstimos, mas cada empréstimo registrado é atrelado a apenas um cliente

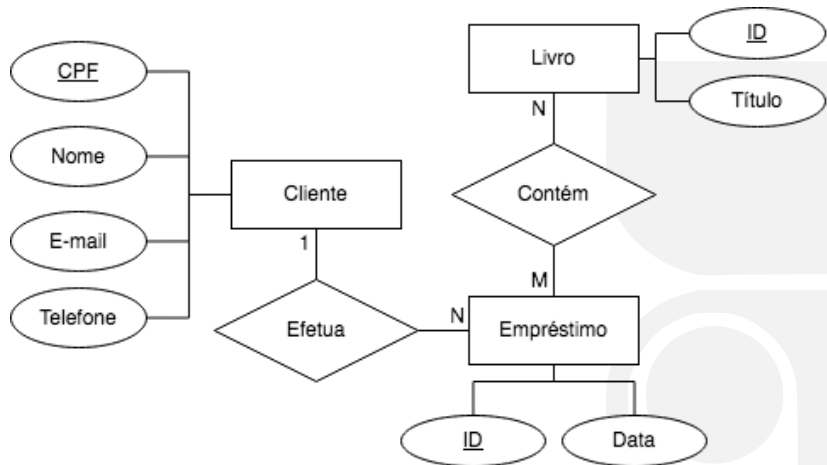
Modelo Entidade-Relacionamento

- Alguns atributos especiais podem ser classificados como **chaves primárias**, geralmente sublinhados no diagrama
 - A ideia de uma chave primária é **identificar** um elemento daquela entidade, sendo um valor único; por exemplo, um Cliente pode ser identificado por um UUID ou por seu CPF
- Relacionamentos possuem uma **cardinalidade**
 - Ao se criar uma relação entre entidades A e B, é necessário informar quantos A se relacionam a B, e quantos B se relacionam a A
 - Cada lado de uma relação possui uma cardinalidade, podendo ser **opcional** (zero ou um), **única** (apenas um), **múltipla** (zero ou mais), etc
 - Um para muitos, muitos para muitos, etc
 - Por exemplo, tendo **Cliente 1** \longleftrightarrow **N Empréstimo** como uma relação, um cliente pode efetuar múltiplos empréstimos, mas cada empréstimo registrado é atrelado a apenas um cliente

Modelo Entidade-Relacionamento

- Alguns atributos especiais podem ser classificados como **chaves primárias**, geralmente sublinhados no diagrama
 - A ideia de uma chave primária é **identificar** um elemento daquela entidade, sendo um valor único; por exemplo, um Cliente pode ser identificado por um UUID ou por seu CPF
- Relacionamentos possuem uma **cardinalidade**
 - Ao se criar uma relação entre entidades A e B, é necessário informar quantos A se relacionam a B, e quantos B se relacionam a A
 - Cada lado de uma relação possui uma cardinalidade, podendo ser **opcional** (zero ou um), **única** (apenas um), **múltipla** (zero ou mais), etc
 - Um para muitos, muitos para muitos, etc
 - Por exemplo, tendo **Cliente 1** \longleftrightarrow **N Empréstimo** como uma relação, um cliente pode efetuar múltiplos empréstimos, mas cada empréstimo registrado é atrelado a apenas um cliente

Modelo Entidade-Relacionamento (notação de Chen)



Notação *crow's foot*

- O tipo de notação demonstrada anteriormente se chama notação de Chen
- Outra alternativa para representar modelos ER é através da notação *crow's foot*
- Entidades são representadas através de retângulos com duas seções, similar a um diagrama de classes UML
 - Atributos são anotados diretamente dentro das entidades
- Relacionamentos são representados diretamente por linhas conectando entidades, onde as pontas das setas representam a cardinalidade
 - Similar à notação de Chen, a cardinalidade de uma entidade em um relacionamento é anotada diretamente perto da entidade
 - Um traço representa uma cardinalidade obrigatória
 - Um círculo representa uma cardinalidade opcional
 - O “pé de corvo” representa uma cardinalidade múltipla

Notação *crow's foot*

- O tipo de notação demonstrada anteriormente se chama notação de Chen
- Outra alternativa para representar modelos ER é através da notação *crow's foot*
- Entidades são representadas através de retângulos com duas seções, similar a um diagrama de classes UML
 - Atributos são anotados diretamente dentro das entidades
- Relacionamentos são representados diretamente por linhas conectando entidades, onde as pontas das setas representam a cardinalidade
 - Similar à notação de Chen, a cardinalidade de uma entidade em um relacionamento é anotada diretamente perto da entidade
 - Um traço representa uma cardinalidade obrigatória
 - Um círculo representa uma cardinalidade opcional
 - O “pé de corvo” representa uma cardinalidade múltipla

Notação *crow's foot*

- O tipo de notação demonstrada anteriormente se chama notação de Chen
- Outra alternativa para representar modelos ER é através da notação *crow's foot*
- Entidades são representadas através de retângulos com duas seções, similar a um diagrama de classes UML
 - Atributos são anotados diretamente dentro das entidades
- Relacionamentos são representados diretamente por linhas conectando entidades, onde as pontas das setas representam a cardinalidade
 - Similar à notação de Chen, a cardinalidade de uma entidade em um relacionamento é anotada diretamente perto da entidade
 - Um traço representa uma cardinalidade obrigatória
 - Um círculo representa uma cardinalidade opcional
 - O “pé de corvo” representa uma cardinalidade múltipla

Notação *crow's foot*

- O tipo de notação demonstrada anteriormente se chama notação de Chen
- Outra alternativa para representar modelos ER é através da notação *crow's foot*
- Entidades são representadas através de retângulos com duas seções, similar a um diagrama de classes UML
 - Atributos são anotados diretamente dentro das entidades
- Relacionamentos são representados diretamente por linhas conectando entidades, onde as pontas das setas representam a cardinalidade
 - Similar à notação de Chen, a cardinalidade de uma entidade em um relacionamento é anotada diretamente perto da entidade
 - Um traço representa uma cardinalidade obrigatória
 - Um círculo representa uma cardinalidade opcional
 - O “pé de corvo” representa uma cardinalidade múltipla

Notação *crow's foot*

- O tipo de notação demonstrada anteriormente se chama notação de Chen
- Outra alternativa para representar modelos ER é através da notação *crow's foot*
- Entidades são representadas através de retângulos com duas seções, similar a um diagrama de classes UML
 - Atributos são anotados diretamente dentro das entidades
- Relacionamentos são representados diretamente por linhas conectando entidades, onde as pontas das setas representam a cardinalidade
 - Similar à notação de Chen, a cardinalidade de uma entidade em um relacionamento é anotada diretamente perto da entidade
 - Um traço representa uma cardinalidade obrigatória
 - Um círculo representa uma cardinalidade opcional
 - O “pé de corvo” representa uma cardinalidade múltipla

Notação *crow's foot*

- O tipo de notação demonstrada anteriormente se chama notação de Chen
- Outra alternativa para representar modelos ER é através da notação *crow's foot*
- Entidades são representadas através de retângulos com duas seções, similar a um diagrama de classes UML
 - Atributos são anotados diretamente dentro das entidades
- Relacionamentos são representados diretamente por linhas conectando entidades, onde as pontas das setas representam a cardinalidade
 - Similar à notação de Chen, a cardinalidade de uma entidade em um relacionamento é anotada diretamente perto da entidade
 - Um traço representa uma cardinalidade obrigatória
 - Um círculo representa uma cardinalidade opcional
 - O “pé de corvo” representa uma cardinalidade múltipla

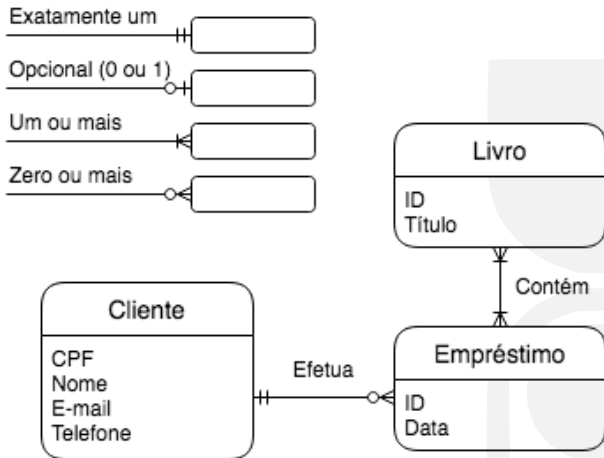
Notação *crow's foot*

- O tipo de notação demonstrada anteriormente se chama notação de Chen
- Outra alternativa para representar modelos ER é através da notação *crow's foot*
- Entidades são representadas através de retângulos com duas seções, similar a um diagrama de classes UML
 - Atributos são anotados diretamente dentro das entidades
- Relacionamentos são representados diretamente por linhas conectando entidades, onde as pontas das setas representam a cardinalidade
 - Similar à notação de Chen, a cardinalidade de uma entidade em um relacionamento é anotada diretamente perto da entidade
 - Um traço representa uma cardinalidade obrigatória
 - Um círculo representa uma cardinalidade opcional
 - O “pé de corvo” representa uma cardinalidade múltipla

- O tipo de notação demonstrada anteriormente se chama notação de Chen
- Outra alternativa para representar modelos ER é através da notação *crow's foot*
- Entidades são representadas através de retângulos com duas seções, similar a um diagrama de classes UML
 - Atributos são anotados diretamente dentro das entidades
- Relacionamentos são representados diretamente por linhas conectando entidades, onde as pontas das setas representam a cardinalidade
 - Similar à notação de Chen, a cardinalidade de uma entidade em um relacionamento é anotada diretamente perto da entidade
 - Um traço representa uma cardinalidade obrigatória
 - Um círculo representa uma cardinalidade opcional
 - O “pé de corvo” representa uma cardinalidade múltipla

- O tipo de notação demonstrada anteriormente se chama notação de Chen
- Outra alternativa para representar modelos ER é através da notação *crow's foot*
- Entidades são representadas através de retângulos com duas seções, similar a um diagrama de classes UML
 - Atributos são anotados diretamente dentro das entidades
- Relacionamentos são representados diretamente por linhas conectando entidades, onde as pontas das setas representam a cardinalidade
 - Similar à notação de Chen, a cardinalidade de uma entidade em um relacionamento é anotada diretamente perto da entidade
 - Um traço representa uma cardinalidade obrigatória
 - Um círculo representa uma cardinalidade opcional
 - O “pé de corvo” representa uma cardinalidade múltipla

Modelo Entidade-Relacionamento (notação *crow's foot*)



- Historicamente, várias notações gráficas para modelagem de sistemas foram propostas, sem uma forma prevalente
- O padrão UML (do inglês, *Unified Modeling Language*) foi proposto em 1994 como uma forma padronizada de se modelar sistemas **orientados a objetos**, e se tornou popular para o projeto de *software*
- O UML é descrito por um **metamodelo**: ele é modelado através de diagramas que descrevem diagramas
- Atualmente na versão 2.5.1, o UML contém vários tipos de diagramas distintos, se dividindo em duas categorias
 - Diagramas estruturais, cujo objetivo é representar partes estáticas de um projeto, como o diagramas de classe
 - Diagramas comportamentais, cujo objetivo é representar iterações e fluxos dinâmicos em um sistema, como por exemplo os diagramas de atividade e sequência

- Historicamente, várias notações gráficas para modelagem de sistemas foram propostas, sem uma forma prevalente
- O padrão UML (do inglês, *Unified Modeling Language*) foi proposto em 1994 como uma forma padronizada de se modelar sistemas **orientados a objetos**, e se tornou popular para o projeto de *software*
- O UML é descrito por um **metamodelo**: ele é modelado através de diagramas que descrevem diagramas
- Atualmente na versão 2.5.1, o UML contém vários tipos de diagramas distintos, se dividindo em duas categorias
 - Diagramas estruturais, cujo objetivo é representar partes estáticas de um projeto, como o diagramas de classe
 - Diagramas comportamentais, cujo objetivo é representar iterações e fluxos dinâmicos em um sistema, como por exemplo os diagramas de atividade e sequência

- Historicamente, várias notações gráficas para modelagem de sistemas foram propostas, sem uma forma prevalente
- O padrão UML (do inglês, *Unified Modeling Language*) foi proposto em 1994 como uma forma padronizada de se modelar sistemas **orientados a objetos**, e se tornou popular para o projeto de *software*
- O UML é descrito por um **metamodelo**: ele é modelado através de diagramas que descrevem diagramas
- Atualmente na versão 2.5.1, o UML contém vários tipos de diagramas distintos, se dividindo em duas categorias
 - Diagramas estruturais, cujo objetivo é representar partes estáticas de um projeto, como o diagramas de classe
 - Diagramas comportamentais, cujo objetivo é representar iterações e fluxos dinâmicos em um sistema, como por exemplo os diagramas de atividade e sequência

- Historicamente, várias notações gráficas para modelagem de sistemas foram propostas, sem uma forma prevalente
- O padrão UML (do inglês, *Unified Modeling Language*) foi proposto em 1994 como uma forma padronizada de se modelar sistemas **orientados a objetos**, e se tornou popular para o projeto de *software*
- O UML é descrito por um **metamodelo**: ele é modelado através de diagramas que descrevem diagramas
- Atualmente na versão 2.5.1, o UML contém vários tipos de diagramas distintos, se dividindo em duas categorias
 - Diagramas estruturais, cujo objetivo é representar partes estáticas de um projeto, como o diagramas de classe
 - Diagramas comportamentais, cujo objetivo é representar iterações e fluxos dinâmicos em um sistema, como por exemplo os diagramas de atividade e sequência

- Historicamente, várias notações gráficas para modelagem de sistemas foram propostas, sem uma forma prevalente
- O padrão UML (do inglês, *Unified Modeling Language*) foi proposto em 1994 como uma forma padronizada de se modelar sistemas **orientados a objetos**, e se tornou popular para o projeto de *software*
- O UML é descrito por um **metamodelo**: ele é modelado através de diagramas que descrevem diagramas
- Atualmente na versão 2.5.1, o UML contém vários tipos de diagramas distintos, se dividindo em duas categorias
 - Diagramas estruturais, cujo objetivo é representar partes estáticas de um projeto, como o diagramas de classe
 - Diagramas comportamentais, cujo objetivo é representar iterações e fluxos dinâmicos em um sistema, como por exemplo os diagramas de atividade e sequência

- Historicamente, várias notações gráficas para modelagem de sistemas foram propostas, sem uma forma prevalente
- O padrão UML (do inglês, *Unified Modeling Language*) foi proposto em 1994 como uma forma padronizada de se modelar sistemas **orientados a objetos**, e se tornou popular para o projeto de *software*
- O UML é descrito por um **metamodelo**: ele é modelado através de diagramas que descrevem diagramas
- Atualmente na versão 2.5.1, o UML contém vários tipos de diagramas distintos, se dividindo em duas categorias
 - Diagramas estruturais, cujo objetivo é representar partes estáticas de um projeto, como o diagramas de classe
 - Diagramas comportamentais, cujo objetivo é representar iterações e fluxos dinâmicos em um sistema, como por exemplo os diagramas de atividade e sequência

Diagrama de classes

- Através do **diagrama de classes** é possível representar a estrutura planejada para as classes em um sistema orientado a objetos
- Detalhes: www.uml-diagrams.org/class.html
- O diagrama é representado através de um retângulo com **três compartimentos** distintos
 - O primeiro compartimento contém o **nome da classe**, centralizado e em negrito, dentre outras possíveis informações
 - O segundo compartimento contém **atributos**, que são os campos de uma classe, e pode ser omitido
 - O terceiro compartimento contém **operações**, que são os métodos de uma classe, e pode ser omitido
 - Chamamos atributos e operações coletivamente de *features*
- As *features* podem pertencer à própria classe (estáticas), sendo sublinhadas, ou à instâncias delas, além de possuírem uma visibilidade (pública, privada, protegida, ou do pacote)

Diagrama de classes

- Através do **diagrama de classes** é possível representar a estrutura planejada para as classes em um sistema orientado a objetos
- Detalhes: www.uml-diagrams.org/class.html
- O diagrama é representado através de um retângulo com **três compartimentos** distintos
 - O primeiro compartimento contém o **nome da classe**, centralizado e em negrito, dentre outras possíveis informações
 - O segundo compartimento contém **atributos**, que são os campos de uma classe, e pode ser omitido
 - O terceiro compartimento contém **operações**, que são os métodos de uma classe, e pode ser omitido
 - Chamamos atributos e operações coletivamente de *features*
- As *features* podem pertencer à própria classe (estáticas), sendo sublinhadas, ou à instâncias delas, além de possuírem uma visibilidade (pública, privada, protegida, ou do pacote)

Diagrama de classes

- Através do **diagrama de classes** é possível representar a estrutura planejada para as classes em um sistema orientado a objetos
- Detalhes: www.uml-diagrams.org/class.html
- O diagrama é representado através de um retângulo com **três compartimentos** distintos
 - O primeiro compartimento contém o **nome da classe**, centralizado e em negrito, dentre outras possíveis informações
 - O segundo compartimento contém **atributos**, que são os campos de uma classe, e pode ser omitido
 - O terceiro compartimento contém **operações**, que são os métodos de uma classe, e pode ser omitido
 - Chamamos atributos e operações coletivamente de *features*
- As *features* podem pertencer à própria classe (estáticas), sendo sublinhadas, ou à instâncias delas, além de possuírem uma visibilidade (pública, privada, protegida, ou do pacote)

Diagrama de classes

- Através do **diagrama de classes** é possível representar a estrutura planejada para as classes em um sistema orientado a objetos
- Detalhes: www.uml-diagrams.org/class.html
- O diagrama é representado através de um retângulo com **três compartimentos** distintos
 - O primeiro compartimento contém o **nome da classe**, centralizado e em negrito, dentre outras possíveis informações
 - O segundo compartimento contém **atributos**, que são os campos de uma classe, e pode ser omitido
 - O terceiro compartimento contém **operações**, que são os métodos de uma classe, e pode ser omitido
 - Chamamos atributos e operações coletivamente de *features*
- As *features* podem pertencer à própria classe (estáticas), sendo sublinhadas, ou à instâncias delas, além de possuírem uma visibilidade (pública, privada, protegida, ou do pacote)

Diagrama de classes

- Através do **diagrama de classes** é possível representar a estrutura planejada para as classes em um sistema orientado a objetos
- Detalhes: www.uml-diagrams.org/class.html
- O diagrama é representado através de um retângulo com **três compartimentos** distintos
 - O primeiro compartimento contém o **nome da classe**, centralizado e em negrito, dentre outras possíveis informações
 - O segundo compartimento contém **atributos**, que são os campos de uma classe, e pode ser omitido
 - O terceiro compartimento contém **operações**, que são os métodos de uma classe, e pode ser omitido
 - Chamamos atributos e operações coletivamente de *features*
- As *features* podem pertencer à própria classe (estáticas), sendo sublinhadas, ou à instâncias delas, além de possuírem uma visibilidade (pública, privada, protegida, ou do pacote)

Diagrama de classes

- Através do **diagrama de classes** é possível representar a estrutura planejada para as classes em um sistema orientado a objetos
- Detalhes: www.uml-diagrams.org/class.html
- O diagrama é representado através de um retângulo com **três compartimentos** distintos
 - O primeiro compartimento contém o **nome da classe**, centralizado e em negrito, dentre outras possíveis informações
 - O segundo compartimento contém **atributos**, que são os campos de uma classe, e pode ser omitido
 - O terceiro compartimento contém **operações**, que são os métodos de uma classe, e pode ser omitido
 - Chamamos atributos e operações coletivamente de *features*
- As *features* podem pertencer à própria classe (estáticas), sendo sublinhadas, ou à instâncias delas, além de possuírem uma visibilidade (pública, privada, protegida, ou do pacote)

Diagrama de classes

- Através do **diagrama de classes** é possível representar a estrutura planejada para as classes em um sistema orientado a objetos
- Detalhes: www.uml-diagrams.org/class.html
- O diagrama é representado através de um retângulo com **três compartimentos** distintos
 - O primeiro compartimento contém o **nome da classe**, centralizado e em negrito, dentre outras possíveis informações
 - O segundo compartimento contém **atributos**, que são os campos de uma classe, e pode ser omitido
 - O terceiro compartimento contém **operações**, que são os métodos de uma classe, e pode ser omitido
 - Chamamos atributos e operações coletivamente de *features*
- As *features* podem pertencer à própria classe (estáticas), sendo sublinhadas, ou à instâncias delas, além de possuírem uma visibilidade (pública, privada, protegida, ou do pacote)

Diagrama de classes

- Através do **diagrama de classes** é possível representar a estrutura planejada para as classes em um sistema orientado a objetos
- Detalhes: www.uml-diagrams.org/class.html
- O diagrama é representado através de um retângulo com **três compartimentos** distintos
 - O primeiro compartimento contém o **nome da classe**, centralizado e em negrito, dentre outras possíveis informações
 - O segundo compartimento contém **atributos**, que são os campos de uma classe, e pode ser omitido
 - O terceiro compartimento contém **operações**, que são os métodos de uma classe, e pode ser omitido
 - Chamamos atributos e operações coletivamente de *features*
- As *features* podem pertencer à própria classe (estáticas), sendo sublinhadas, ou à instâncias delas, além de possuírem uma visibilidade (pública, privada, protegida, ou do pacote)

Diagrama de classes

- É possível demonstrar relações entre classes UML através de **relacionamentos**, representados através de linhas conectando duas classes com pontas distintas
 - **Generalização** representa a ideia que uma classe herda *features* de uma classe pai, representando a ideia de herança em linguagens orientadas a objetos, como, por exemplo, as classes Cachorro e Mamífero (oposto: **especialização**)
 - **Associação** representa que duas classes são, semanticamente, associadas em um sistema, e trocam mensagens entre si, como, por exemplo, as classes Vôo e Avião
 - **Composição** representa que uma classe compõe outra classe que não existe independentemente, como, por exemplo, as classes Casa e Cômodo (um cômodo não existe sem uma casa)
 - **Agregação** representa que uma classe agrega outra classe, a qual pode existir independentemente, como, por exemplo, as classes Matéria e Aluno (se a matéria for cancelada, os alunos continuam existindo)

Diagrama de classes

- É possível demonstrar relações entre classes UML através de **relacionamentos**, representados através de linhas conectando duas classes com pontas distintas
 - **Generalização** representa a ideia que uma classe herda *features* de uma classe pai, representando a ideia de herança em linguagens orientadas a objetos, como, por exemplo, as classes Cachorro e Mamífero (oposto: **especialização**)
 - **Associação** representa que duas classes são, semanticamente, associadas em um sistema, e trocam mensagens entre si, como, por exemplo, as classes Vôo e Avião
 - **Composição** representa que uma classe compõe outra classe que não existe independentemente, como, por exemplo, as classes Casa e Cômodo (um cômodo não existe sem uma casa)
 - **Agregação** representa que uma classe agrega outra classe, a qual pode existir independentemente, como, por exemplo, as classes Matéria e Aluno (se a matéria for cancelada, os alunos continuam existindo)

Diagrama de classes

- É possível demonstrar relações entre classes UML através de **relacionamentos**, representados através de linhas conectando duas classes com pontas distintas
 - **Generalização** representa a ideia que uma classe herda *features* de uma classe pai, representando a ideia de herança em linguagens orientadas a objetos, como, por exemplo, as classes Cachorro e Mamífero (oposto: **especialização**)
 - **Associação** representa que duas classes são, semanticamente, associadas em um sistema, e trocam mensagens entre si, como, por exemplo, as classes Vôo e Avião
 - **Composição** representa que uma classe compõe outra classe que não existe independentemente, como, por exemplo, as classes Casa e Cômodo (um cômodo não existe sem uma casa)
 - **Agregação** representa que uma classe agrega outra classe, a qual pode existir independentemente, como, por exemplo, as classes Matéria e Aluno (se a matéria for cancelada, os alunos continuam existindo)

Diagrama de classes

- É possível demonstrar relações entre classes UML através de **relacionamentos**, representados através de linhas conectando duas classes com pontas distintas
 - **Generalização** representa a ideia que uma classe herda *features* de uma classe pai, representando a ideia de herança em linguagens orientadas a objetos, como, por exemplo, as classes Cachorro e Mamífero (oposto: **especialização**)
 - **Associação** representa que duas classes são, semanticamente, associadas em um sistema, e trocam mensagens entre si, como, por exemplo, as classes Vôo e Avião
 - **Composição** representa que uma classe compõe outra classe que não existe independentemente, como, por exemplo, as classes Casa e Cômodo (um cômodo não existe sem uma casa)
 - **Agregação** representa que uma classe agrega outra classe, a qual pode existir independentemente, como, por exemplo, as classes Matéria e Aluno (se a matéria for cancelada, os alunos continuam existindo)

Diagrama de classes

- É possível demonstrar relações entre classes UML através de **relacionamentos**, representados através de linhas conectando duas classes com pontas distintas
 - **Generalização** representa a ideia que uma classe herda *features* de uma classe pai, representando a ideia de herança em linguagens orientadas a objetos, como, por exemplo, as classes Cachorro e Mamífero (oposto: **especialização**)
 - **Associação** representa que duas classes são, semanticamente, associadas em um sistema, e trocam mensagens entre si, como, por exemplo, as classes Vôo e Avião
 - **Composição** representa que uma classe compõe outra classe que não existe independentemente, como, por exemplo, as classes Casa e Cômodo (um cômodo não existe sem uma casa)
 - **Agregação** representa que uma classe agrega outra classe, a qual pode existir independentemente, como, por exemplo, as classes Matéria e Aluno (se a matéria for cancelada, os alunos continuam existindo)

Diagrama de classes (geral)

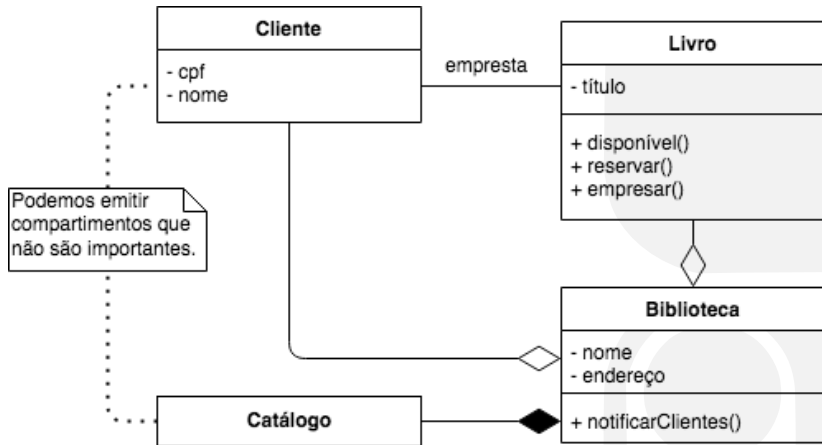


Diagrama de classes (generalização)

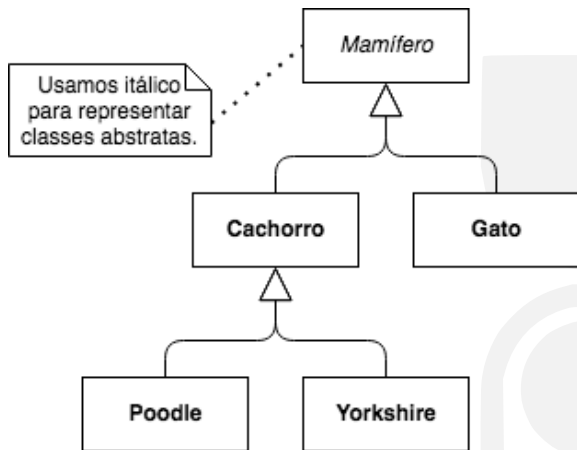


Diagrama de classes (composição)

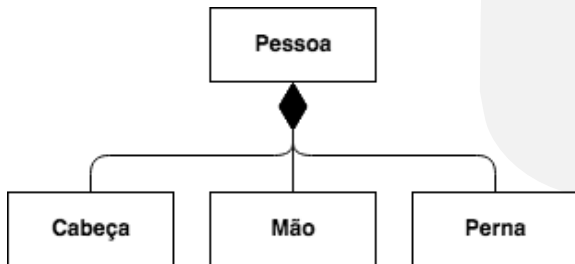


Diagrama de classes (agregação)

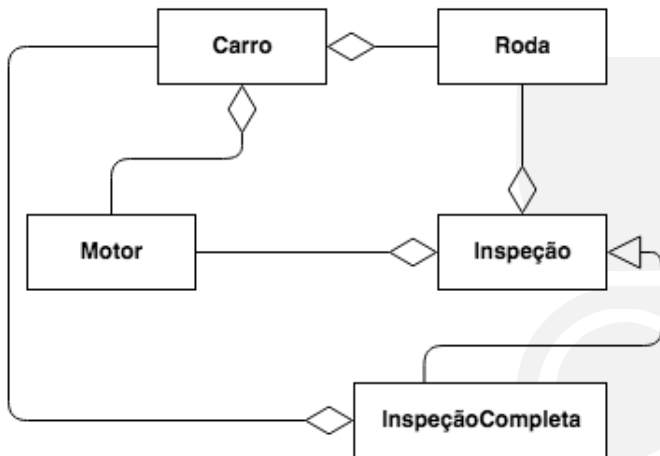


Diagrama de atividade

- O **diagrama de atividades** representa uma sequência (ou melhor dizendo, um fluxo) de **ações** necessárias para se completar uma tarefa durante a execução de um sistema
- www.uml-diagrams.org/activity-diagrams.html
- Diagramas de atividade representam uma das principais formas de se representar **comportamento** dentro de diagramas UML, junto ao diagrama de máquina de estado e de sequência
- Atividades são representadas dentro de um retângulo com pontas arredondadas, com o nome da atividade em **negrito** no campo superior esquerdo (o “nome da função”)
- Atividades são essencialmente **assíncronas**, possuindo um **estado inicial**, representado por um círculo preenchido
- Ações **fluem** de umas para as outras até o **estado final**, que é representado por um círculo preenchido dentro de outro círculo

Diagrama de atividade

- O **diagrama de atividades** representa uma sequência (ou melhor dizendo, um fluxo) de **ações** necessárias para se completar uma tarefa durante a execução de um sistema
- www.uml-diagrams.org/activity-diagrams.html
- Diagramas de atividade representam uma das principais formas de se representar **comportamento** dentro de diagramas UML, junto ao diagrama de máquina de estado e de sequência
- Atividades são representadas dentro de um retângulo com pontas arredondadas, com o nome da atividade em **negrito** no campo superior esquerdo (o “nome da função”)
- Atividades são essencialmente **assíncronas**, possuindo um **estado inicial**, representado por um círculo preenchido
- Ações **fluem** de umas para as outras até o **estado final**, que é representado por um círculo preenchido dentro de outro círculo

Diagrama de atividade

- O **diagrama de atividades** representa uma sequência (ou melhor dizendo, um fluxo) de **ações** necessárias para se completar uma tarefa durante a execução de um sistema
- www.uml-diagrams.org/activity-diagrams.html
- Diagramas de atividade representam uma das principais formas de se representar **comportamento** dentro de diagramas UML, junto ao diagrama de máquina de estado e de sequência
- Atividades são representadas dentro de um retângulo com pontas arredondadas, com o nome da atividade em **negrito** no campo superior esquerdo (o “nome da função”)
- Atividades são essencialmente **assíncronas**, possuindo um **estado inicial**, representado por um círculo preenchido
- Ações **fluem** de umas para as outras até o **estado final**, que é representado por um círculo preenchido dentro de outro círculo

Diagrama de atividade

- O **diagrama de atividades** representa uma sequência (ou melhor dizendo, um fluxo) de **ações** necessárias para se completar uma tarefa durante a execução de um sistema
- www.uml-diagrams.org/activity-diagrams.html
- Diagramas de atividade representam uma das principais formas de se representar **comportamento** dentro de diagramas UML, junto ao diagrama de máquina de estado e de sequência
- Atividades são representadas dentro de um retângulo com pontas arredondadas, com o nome da atividade em negrito no campo superior esquerdo (o “nome da função”)
- Atividades são essencialmente **assíncronas**, possuindo um **estado inicial**, representado por um círculo preenchido
- Ações **fluem** de umas para as outras até o **estado final**, que é representado por um círculo preenchido dentro de outro círculo

Diagrama de atividade

- O **diagrama de atividades** representa uma sequência (ou melhor dizendo, um fluxo) de **ações** necessárias para se completar uma tarefa durante a execução de um sistema
- www.uml-diagrams.org/activity-diagrams.html
- Diagramas de atividade representam uma das principais formas de se representar **comportamento** dentro de diagramas UML, junto ao diagrama de máquina de estado e de sequência
- Atividades são representadas dentro de um retângulo com pontas arredondadas, com o nome da atividade em negrito no campo superior esquerdo (o “nome da função”)
- Atividades são essencialmente **assíncronas**, possuindo um **estado inicial**, representado por um círculo preenchido
- Ações **fluem** de umas para as outras até o **estado final**, que é representado por um círculo preenchido dentro de outro círculo

Diagrama de atividade

- O **diagrama de atividades** representa uma sequência (ou melhor dizendo, um fluxo) de **ações** necessárias para se completar uma tarefa durante a execução de um sistema
- www.uml-diagrams.org/activity-diagrams.html
- Diagramas de atividade representam uma das principais formas de se representar **comportamento** dentro de diagramas UML, junto ao diagrama de máquina de estado e de sequência
- Atividades são representadas dentro de um retângulo com pontas arredondadas, com o nome da atividade em negrito no campo superior esquerdo (o “nome da função”)
- Atividades são essencialmente **assíncronas**, possuindo um **estado inicial**, representado por um círculo preenchido
- Ações **fluem** de umas para as outras até o **estado final**, que é representado por um círculo preenchido dentro de outro círculo

Diagrama de atividade

- Estados intermediários, representados em retângulos de bordas arredondadas, são usados para invocar ações dentro do fluxo
- Como a ideia é que informações fluam entre estados, usamos losangos para representar pontos de **decisão** (*decision*) e **união** (*merge*) de informações
 - As setas partindo de um ponto de decisão apresentam **condições** (*guards*), representados entre colchetes sobre elas
- Com o fim de se representar diagramas de forma genérica, é possível representar pontos de **divisão** (*fork*) e de **junção** (*join*), utilizando linhas fortes, marcando, respectivamente, o início e o término de fluxos concorrentes

Diagrama de atividade

- Estados intermediários, representados em retângulos de bordas arredondadas, são usados para invocar ações dentro do fluxo
- Como a ideia é que informações fluam entre estados, usamos losangos para representar pontos de **decisão** (*decision*) e **união** (*merge*) de informações
 - As setas partindo de um ponto de decisão apresentam **condições** (*guards*), representados entre colchetes sobre elas
- Com o fim de se representar diagramas de forma genérica, é possível representar pontos de **divisão** (*fork*) e de **junção** (*join*), utilizando linhas fortes, marcando, respectivamente, o início e o término de fluxos concorrentes

Diagrama de atividade

- Estados intermediários, representados em retângulos de bordas arredondadas, são usados para invocar ações dentro do fluxo
- Como a ideia é que informações fluam entre estados, usamos losangos para representar pontos de **decisão** (*decision*) e **união** (*merge*) de informações
 - As setas partindo de um ponto de decisão apresentam **condições** (*guards*), representados entre colchetes sobre elas
- Com o fim de se representar diagramas de forma genérica, é possível representar pontos de **divisão** (*fork*) e de **junção** (*join*), utilizando linhas fortes, marcando, respectivamente, o início e o término de fluxos concorrentes

Diagrama de atividade

- Estados intermediários, representados em retângulos de bordas arredondadas, são usados para invocar ações dentro do fluxo
- Como a ideia é que informações fluam entre estados, usamos losangos para representar pontos de **decisão** (*decision*) e **união** (*merge*) de informações
 - As setas partindo de um ponto de decisão apresentam **condições** (*guards*), representados entre colchetes sobre elas
- Com o fim de se representar diagramas de forma genérica, é possível representar pontos de **divisão** (*fork*) e de **junção** (*join*), utilizando linhas fortes, marcando, respectivamente, o início e o término de fluxos concorrentes