# Capítulo 7

# Projeto e Implementação

# Tópicos cobertos

engenharia de SOFTWARE

- Projeto orientado a objetos com UML
- Padrões de projeto
- Questões de implementação
- Desenvolvimento *open source*

# Projeto e implementação

- O projeto e implementação de software é o estágio no processo de engenharia de software no qual um sistema de software executável é desenvolvido.

- As atividades de projeto e implementação de software são invariavelmente intercaladas.

  - ✓ O projeto de software é uma atividade criativa na qual você identifica os componentes de software e seus relacionamentos baseando-se nos requisitos do cliente.

  - ✓ A implementação é o processo de realização do projeto em um programa.

## Construir ou comprar

- Nas mais variadas áreas, já é possível comprar sistemas de prateleira (COTS – Commercial Off-The-Shelf) que podem ser adaptados aos requisitos dos usuários.

  - ✓ Por exemplo, se você quer implementar um sistema de prontuário médico, você pode comprar um pacote que já é usado em hospitais. Pode ser mais barato e mais rápido usar essa abordagem ao invés de desenvolver um sistema em uma linguagem de programa convencional.

- Quando você desenvolve uma aplicação dessa forma, o processo de projeto passa a se preocupar em como usar os recursos de configuração deste sistema para cumprir os requisitos desse.

# O processo de projeto orientado a objetos



- Os processos de projeto orientados a objetos envolvem o desenvolvimento de vários modelos diferentes de sistema.

- Eles precisam de muito esforço no desenvolvimento e na manutenção desses modelos e, para sistemas pequenos, talvez não tenham um bom custo-benefício.

- No entanto, para sistemas de grande porte, desenvolvidos por diferentes grupos, os modelos de projeto são um importante mecanismo de comunicação.

## Estágios do processo

- Existe uma grande variedade de diferentes processos de projeto orientados a objetos, essa escolha dependerá da organização que está usando o processo.

- As atividades comuns nesses processos incluem:

  - ✓ A definição do contexto e interações do sistema;
  - ✓ O projeto de arquitetura do sistema;
  - ✓ A identificação dos principais objetos de classe do sistema;
  - ✓ O desenvolvimento dos modelos de projeto;
  - ✓ As especificações de interface de objetos.

- O processo ilustrado, usa um projeto para uma estação meteorológica no deserto.

# Contexto e interações de sistema

- Entender os relacionamentos entre o software que está sendo desenvolvido e seu ambiente externo é essencial na decisão de como prover a funcionalidade requerida para o sistema e como estruturar o sistema para se comunicar com seu ambiente.

- Entender o contexto também lhe permite estabelecer os limites do sistema.

- Estabelecer os limites do sistema ajuda a decidir quais recursos serão implementados no sistema que está sendo desenvolvido e quais serão implementados em outros sistemas associados.

# Modelos de contexto e de interação

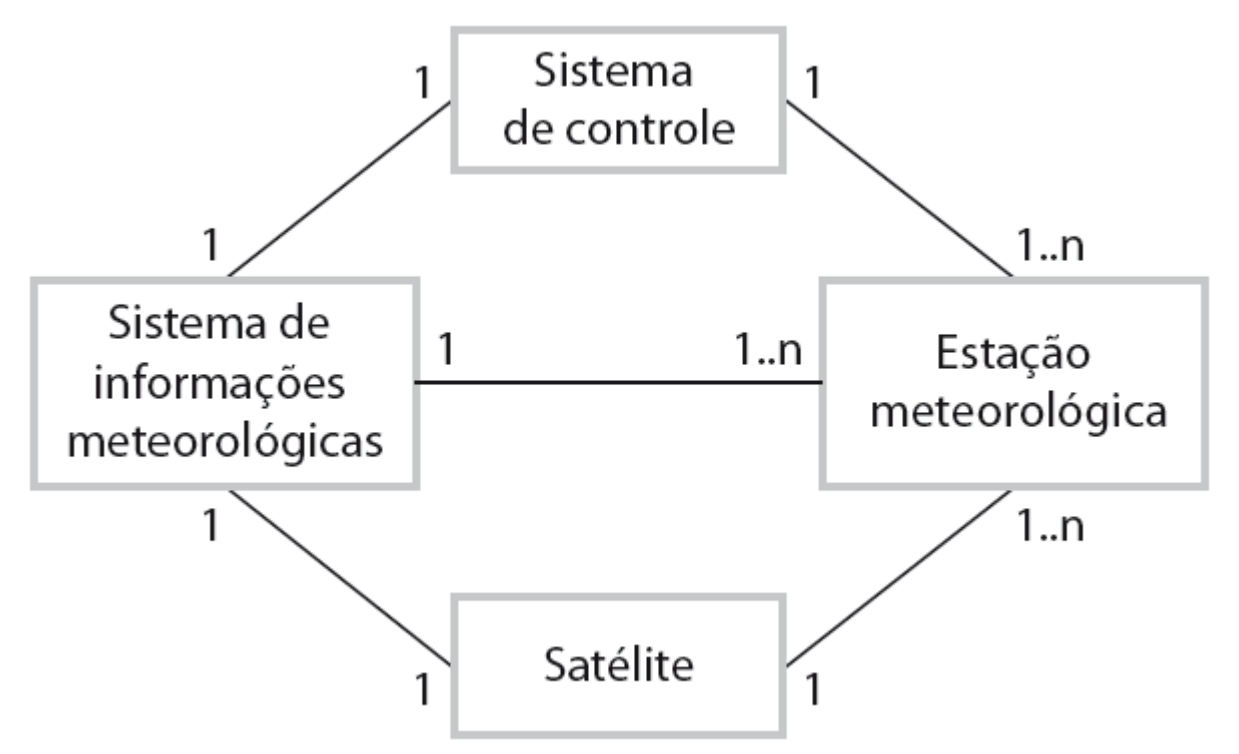
engenharia de SOFTWARE

- Um modelo de contexto de sistema é um modelo estrutural que mostra outros sistemas no ambiente do sistema que está sendo desenvolvido.

- Um modelo de interação é um modelo dinâmico que mostra como o sistema interage com seu ambiente, durante o seu uso.

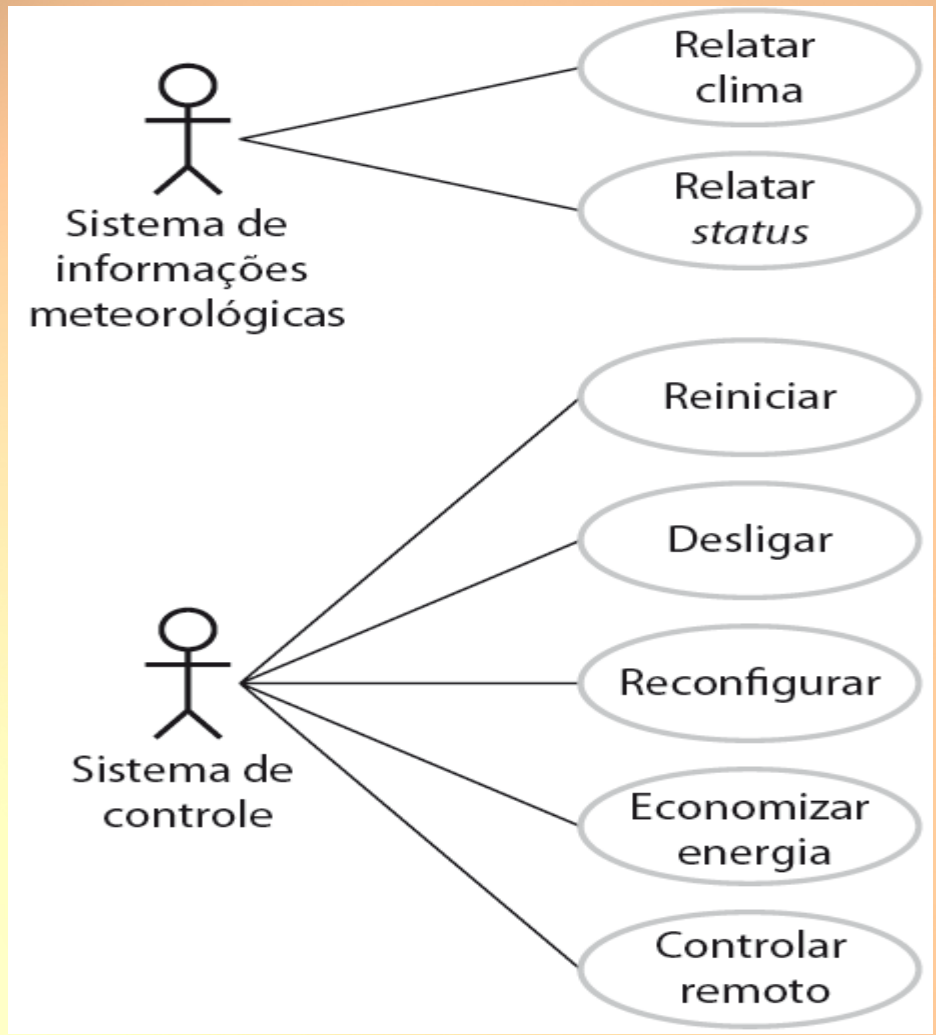
# Contexto de sistema para a estação meteorológica





# Os casos de uso da estação meteorológica





# Descrição de caso de uso – Relatar clima

engenharia de SOFTWARE

Caso de uso: Relatar clima

Atores: Sistema de informações meteorológicas, estação meteorológica

Dados: A estação meteorológica envia um resumo dos dados meteorológicos coletados a partir dos instrumentos, no período de coleta, para o sistema de informações meteorológicas. Os dados enviados são o máximo, mínimo e médio das temperaturas de solo e de ar; a máxima, mínima e média da pressão do ar; a velocidade máxima, mínima e média do vento; a precipitação de chuva total e a direção do vento, amostrados a cada cinco minutos.

Estímulo: O sistema de informações meteorológicas estabelece um link de comunicação via satélite com a estação e solicita a transmissão dos dados.

Resposta: Os dados resumidos são enviados para o sistema de informações meteorológicas.

Comentários: Geralmente, solicita-se que as estações meteorológicas enviem relatórios a cada hora, mas essa frequência pode diferir de uma estação para a outra e pode ser modificada no futuro.

# Projeto de arquitetura

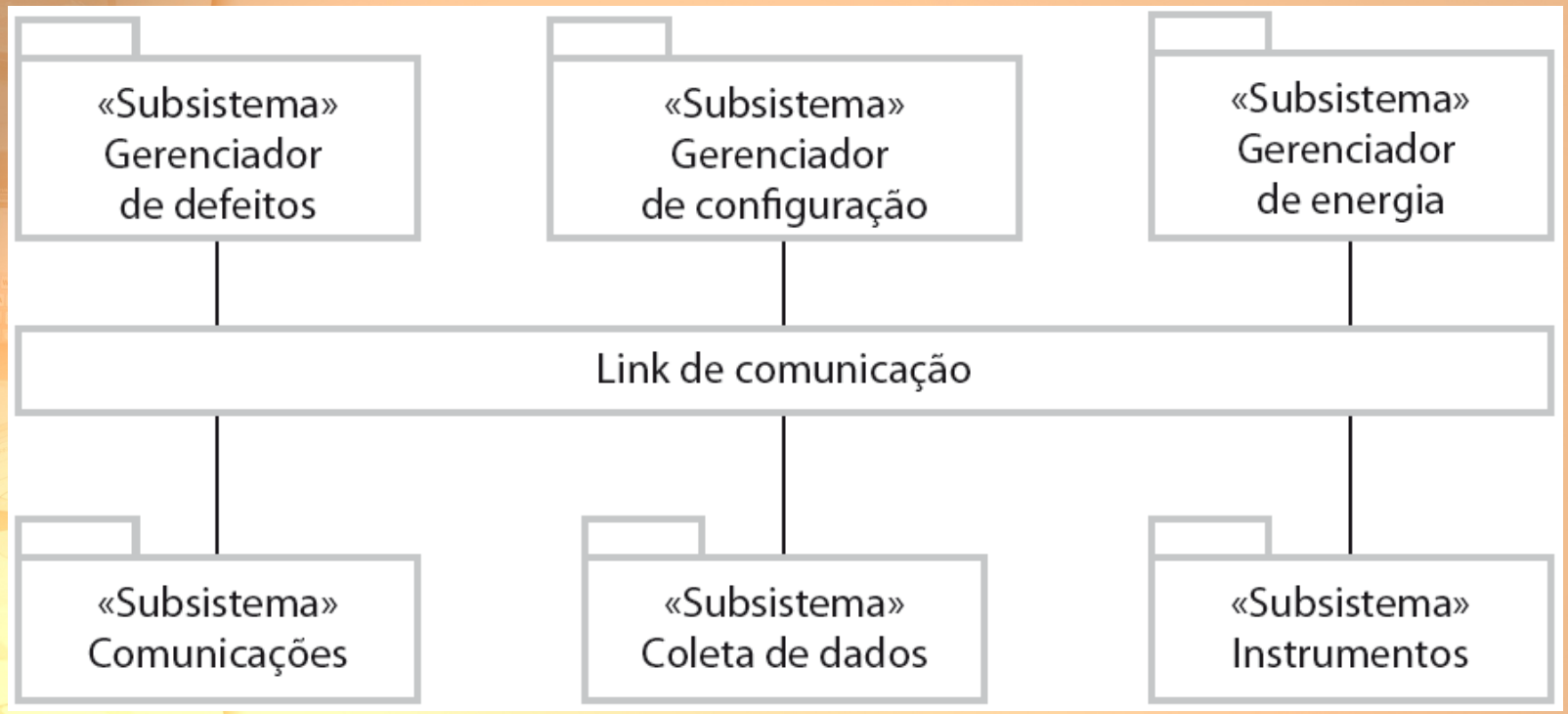
- Assim que as interações entre o sistema e seu ambiente forem entendidas, usa-se essa informação no projeto de arquitetura do sistema.

- Identificam-se os componentes principais que compõem o sistema e suas interações, e então pode-se organizar os componentes usando um padrão de arquitetura como por exemplo, um modelo em camadas ou cliente-servidor.

- A estação meteorológica é composta de subsistemas independentes que se comunicam através de transmissão de mensagens em um infraestrutura comum.

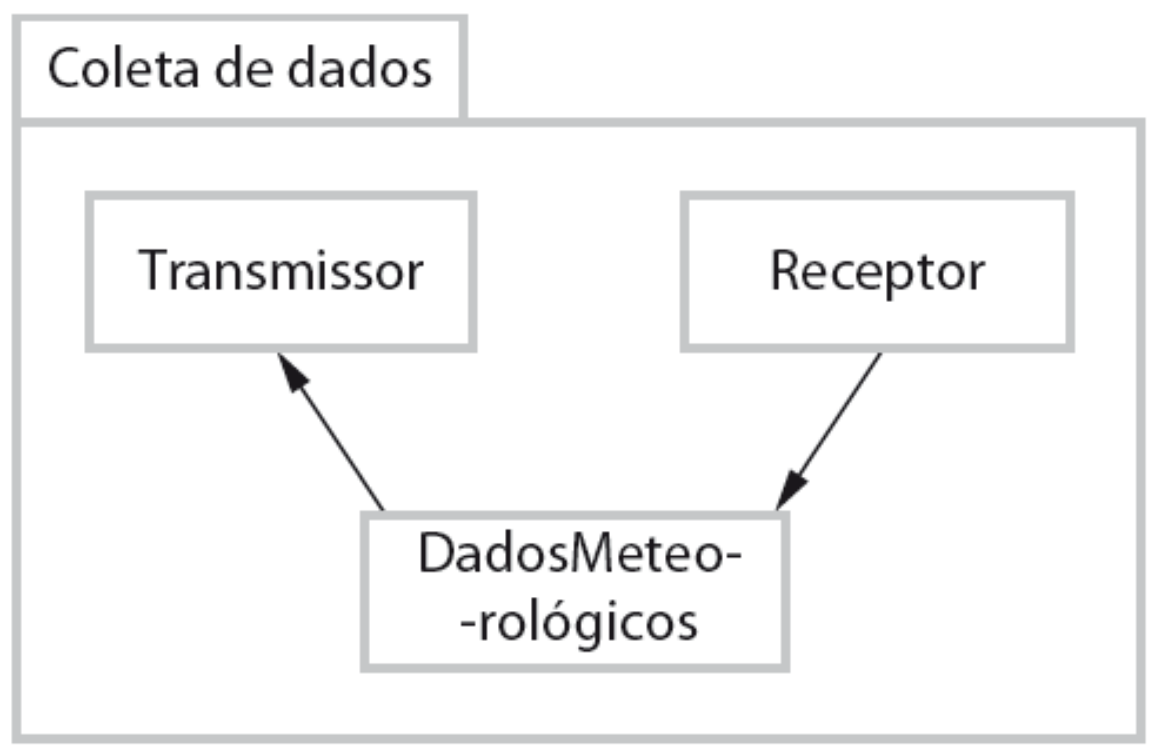
# Arquitetura de alto nível da estação meteorológica





# Arquitetura do sistema de coleta de dados





# Identificação das classes de objetos

- Geralmente, a identificação das classes de objetos é uma parte difícil do projeto orientado a objetos.
- Não existe uma 'fórmula mágica' para a identificação dos objetos. Isso depende do conhecimento, da experiência e habilidade dos projetistas de sistema.
- A identificação de objetos é um processo iterativo. Você provavelmente não conseguirá acertá-lo na primeira tentativa.

## Abordagens de identificação

- Use uma abordagem gramatical baseada em uma descrição em linguagem natural do sistema (usada no método Hood de projeto orientado a objetos, Hood OOD).
- Baseie a identificação em entidades tangíveis no domínio da aplicação.
- Use uma abordagem comportamental e identifique os objetos baseando-se em o quê participa de qual comportamento.
- Use uma análise baseada em cenários.
- Os objetos, atributos e métodos em cada cenário são identificados.

# Descrição da estação meteorológica

- Uma estação meteorológica é um pacote de instrumentos controlados por software que coletam dados, executam algum processamento de dados e transmitem esse dados para processamento posterior.

- Os instrumentos incluem termômetros de chão e de ar, um anemômetro, um cata-vento, um barômetro e um medidor de chuva. Os dados são coletados periodicamente.

- Quando um comando é dado para transmitir os dados meteorológicos, a estação meteorológica processa e resume os dados coletados.

- Os dados resumidos são transmitidos para o computador de mapeamento quando um pedido é recebido.

# Classes de objeto da estação meteorológica



- A identificação das classes de objeto no sistema de estação meteorológica pode se basear no hardware e em dados tangíveis do sistema:

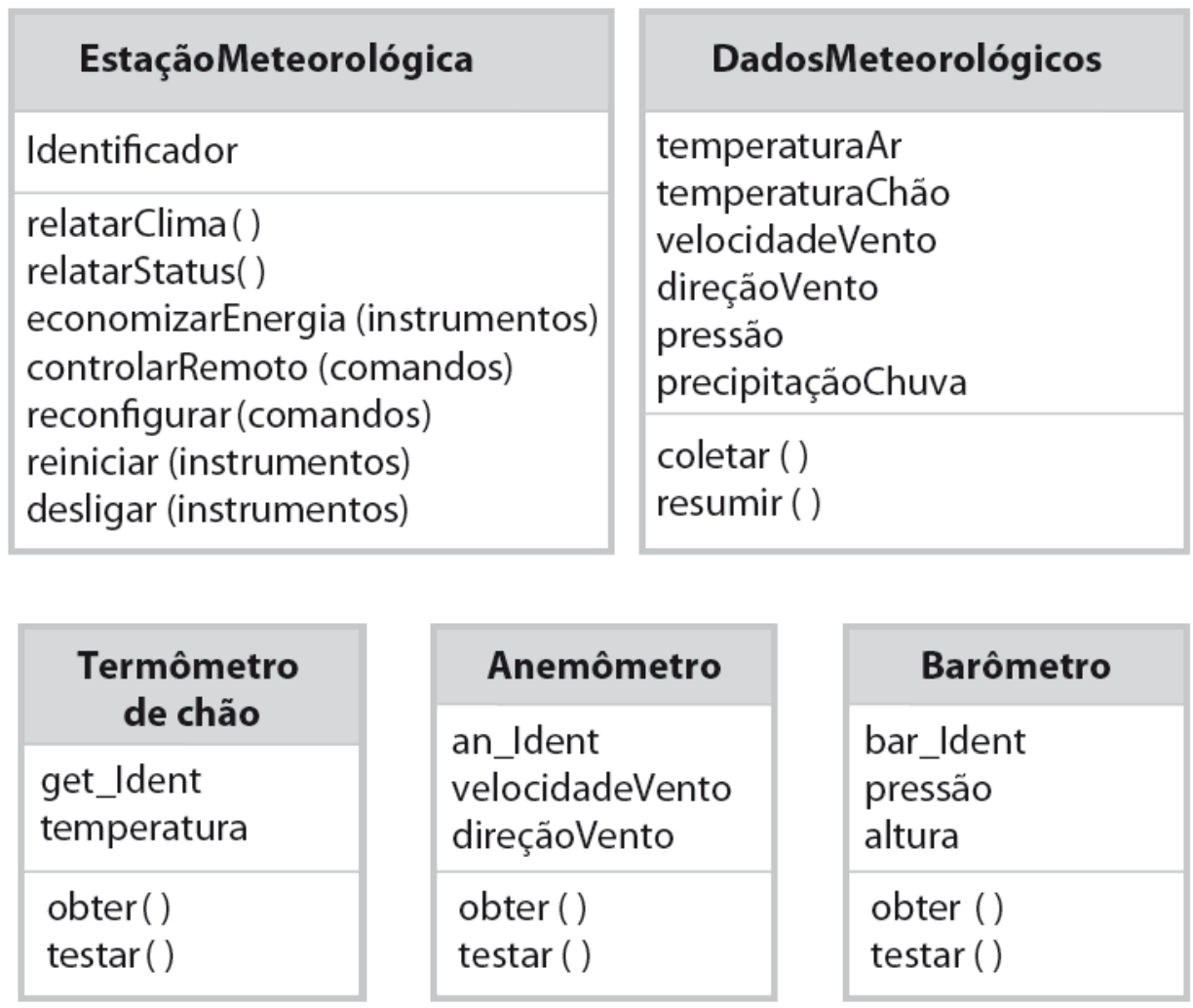
  - ✓ Termômetro de chão, Anemômetro, Barômetro
  - ➢ Objetos do domínio da aplicação que são objetos de 'hardware' relacionados aos instrumentos do sistema.

  - ✓ Estação meteorológica
  - ➢ A interface básica da estação meteorológica que a liga a seu ambiente. Reflete as interações identificadas no modelo de caso de uso.

  - ✓ Dados meteorológicos
  - ➢ Encapsula os dados resumidos dos instrumentos.

# Classes de objeto da estação meteorológica

engenharia de SOFTWARE



# Modelos de projeto

- Os modelos de projeto mostram os objetos e classes de objeto, e os relacionamentos entre essas entidades.
  - ✓ Os modelos estáticos descrevem a estrutura estática do sistema nos termos das classes de objetos e suas relações.
  - ✓ Os modelos dinâmicos descrevem as interações dinâmicas entre os objetos.

# Exemplos de modelos de projeto

- Os modelos de subsistemas que mostram os agrupamentos lógicos de objetos em subsistemas coerentes.
- Os modelos de sequência que mostram a sequência das interações dos objetos.
- Os modelos de máquina de estados que mostram como objetos individuais mudam de estado em resposta a eventos.
- Outros modelos incluem modelos de casos de uso, modelos de agregação, modelos de generalização, etc.

# Modelo de subsistema

engenharia de SOFTWARE

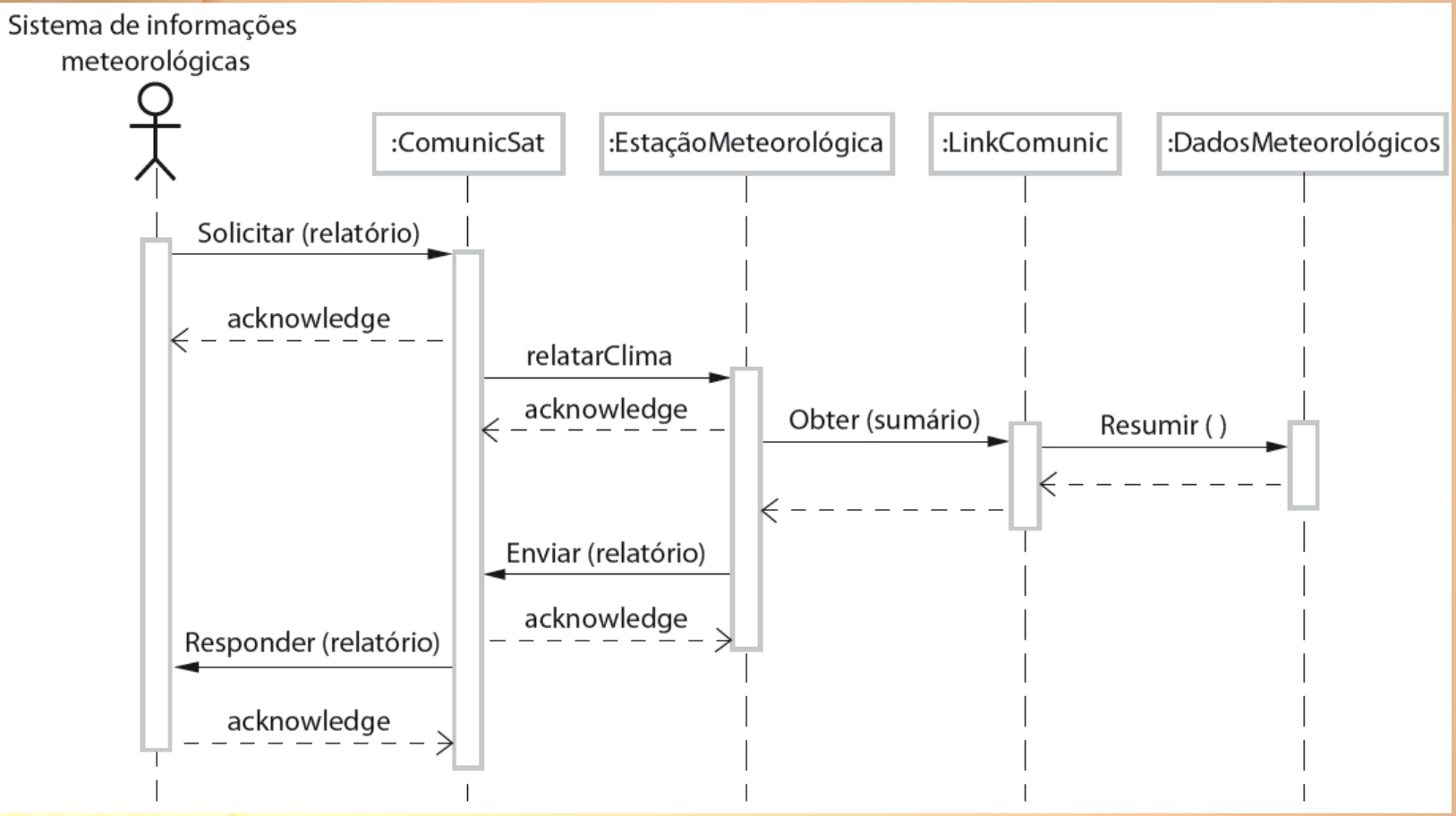
- Mostra como o projeto está organizado em grupos de objetos relacionados logicamente.
- **Em UML, esses são apresentados usando pacotes – um construto encapsulador. Esse é um modelo lógico. (não encontrei isso no livro)**
- A organização real dos objetos no sistema pode ser diferente.

# Modelos de sequência

- Os modelos de sequência mostram a sequência das interações dos objetos que ocorrem.
  - ✓ Os objetos são organizados horizontalmente no topo;
  - ✓ O tempo é representado verticalmente para que os modelos sejam lidos de cima para baixo;
  - ✓ As interações são representadas por setas com rótulos. Diferentes estilos de setas representam diferentes tipos de interação;
  - ✓ Um retângulo fino na linha de vida de um objeto representa o momento em que o objeto é o objeto controlador do sistema.

# Diagrama de sequência descrevendo a coleta de dados



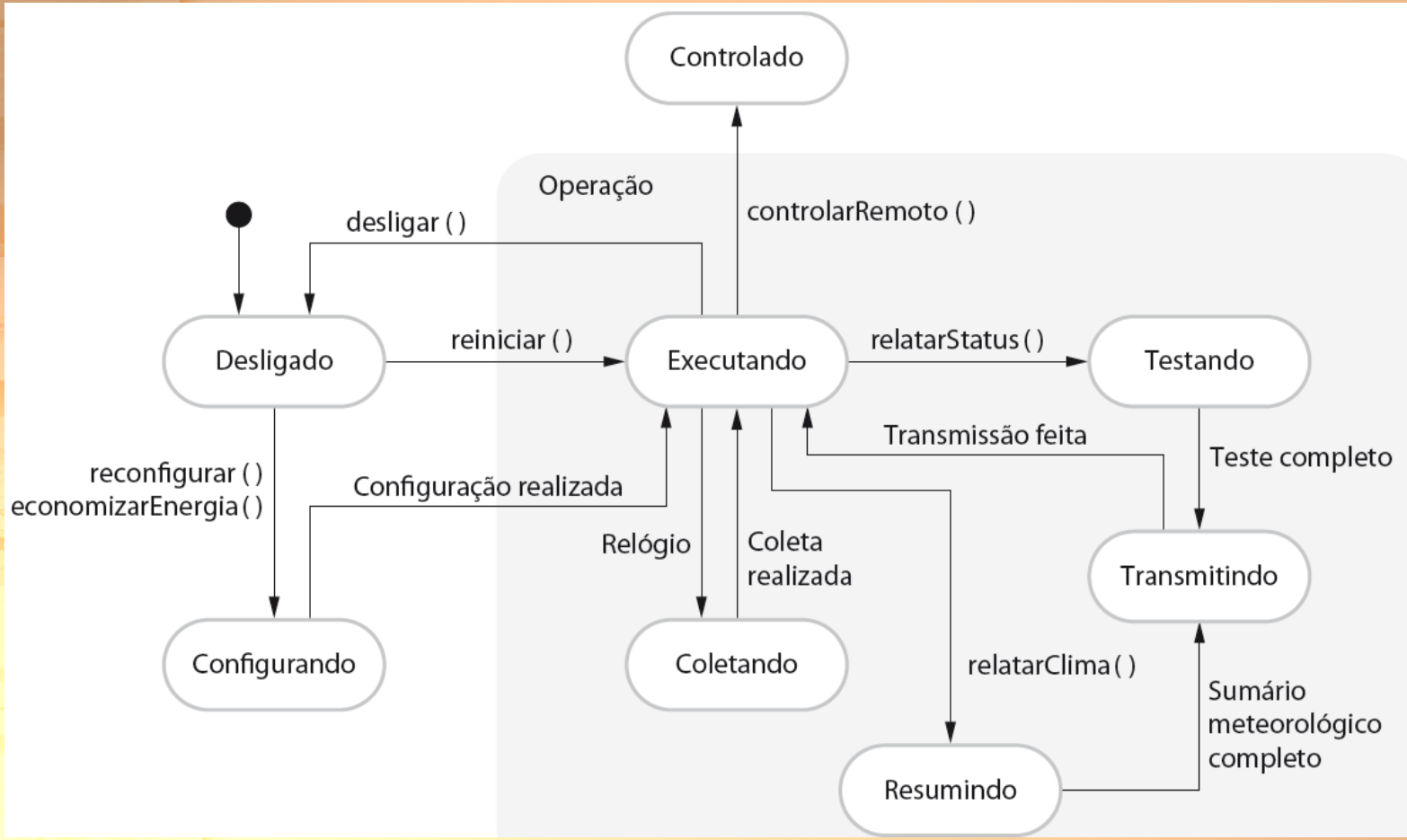


## Diagrama de estado

- O diagrama de estado é usado para mostrar como os objetos respondem a diferentes pedidos de serviços e as transições de estado acionadas por esses pedidos.
- Os diagramas de estado são modelos úteis, de alto nível, de um sistema, ou do comportamento de um objeto em execução.
- Geralmente não é necessário um diagrama de estado para todos os objetos do sistema.
- Vários dos objetos em um sistema são relativamente simples e um modelo de estado adiciona detalhes desnecessários ao projeto.

# Diagrama de estado da estação meteorológica



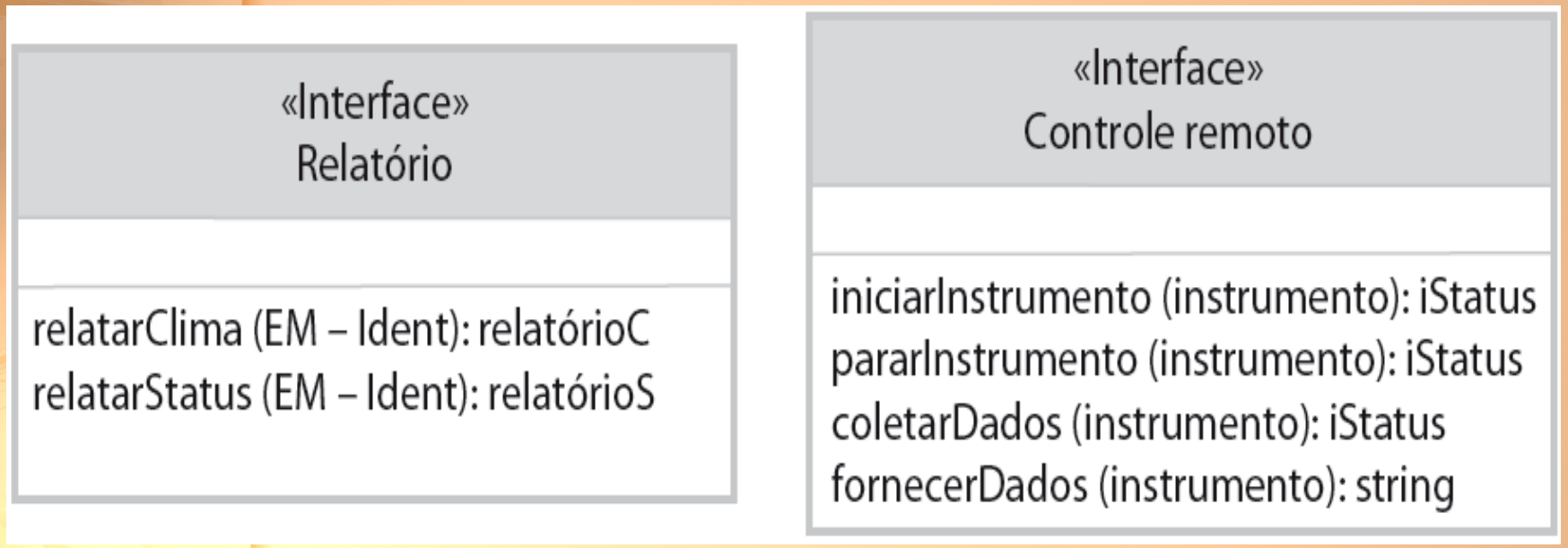


## Especificações de interface

- As interfaces de objeto precisam ser especificadas para que se possa fazer paralelamente o projeto dos objetos e de outros componentes.
- Os projetistas devem evitar projetar a representação da interface, devem escondê-la no próprio objeto.
- Os objetos podem ter várias interfaces, as quais são pontos de vista dos métodos oferecidos.
- A UML usa diagramas de classes para a especificação da interface, mas a linguagem Java também pode ser usada.

# Interfaces da estação meteorológica





## Pontos Importantes

- O projeto e a implementação de softwares são atividades intercaladas.
- O nível de detalhamento no projeto depende do tipo de sistema, e se está sendo usada uma abordagem orientada a planos ou ágil.
- O processo de projeto orientado a objetos inclui atividades para projetar da arquitetura de sistema, identificar os objetos no sistema, descrever o projeto usando diferentes modelos de objetos e documentar as interfaces de componentes.

## Pontos Importantes

- Uma série de diferentes modelos pode ser produzida durante um processo de projeto orientado a objetos.
- Inclusive os modelos estáticos (modelos de classes, modelos de generalização, modelos de associação) e os modelos dinâmicos (modelos de sequência, modelos de máquina de estados).
- As interfaces de componentes devem ser definidas com exatidão para que os outros objetos possam usá-las.
- Um estereótipo de interface da UML pode ser usado para definir as interfaces.

## Padrões de projeto

- Um padrão de projeto é uma forma de reusar conhecimento abstrato sobre um problema e sua solução.
- Um padrão é uma descrição do problema e a essência da sua solução.
- Esse precisa ser abstrato o suficiente para ser reusado em configurações diferentes.
- Geralmente, as descrições do padrão fazem uso de características orientadas a objetos, como herança e polimorfismo.

## Elementos do padrão

- Nome
  - ✓ Um identificador significativo do padrão.
- Descrição do problema
- Descrição da solução
  - ✓ Não um projeto concreto mas um template, uma solução de projeto que pode ser instanciada de diferentes maneiras.
- Consequências
  - ✓ Os resultados e os compromissos da aplicação desse padrão.

# O padrão Observer

- Nome
  - ✓ Observer.
- Descrição
  - ✓ Separa o estado do display do estado do objeto do próprio objeto.
- Descrição do problema
  - ✓ Usada quando vários displays de estado são necessárias.
- Descrição da solução
  - ✓ Ver slide com descrição em UML.
- Consequências
  - ✓ Otimizações para melhorias do desempenho do display são impraticáveis.

# O padrão Observer (1)



| | |
|---|---|
| **Nome do padrão:** | Observer |
| **Descrição:** | Separa o *display* do estado de um objeto a partir do objeto em si e permite que sejam fornecidos *displays* alternativos. Quando o estado do objeto muda, todos os *displays* são automaticamente notificados e atualizados para refletir a mudança. |
| **Descrição do problema:** | Em muitas situações, você precisa fornecer vários *displays* de informações do estado, como um *display* gráfico e em tabela. Nem todos eles podem ser conhecidos quando a informação é especificada. Todas as apresentações alternativas devem apoiar a interação e, quando o estado é alterado, todos os *displays* devem ser atualizados.<br>Esse padrão pode ser usado em todas as situações em que mais de um formato de *display* de informações de estado é necessário, e em que saber sobre os formatos de *display* específicos usados não é necessário para o objeto que mantém as informações do estado. |

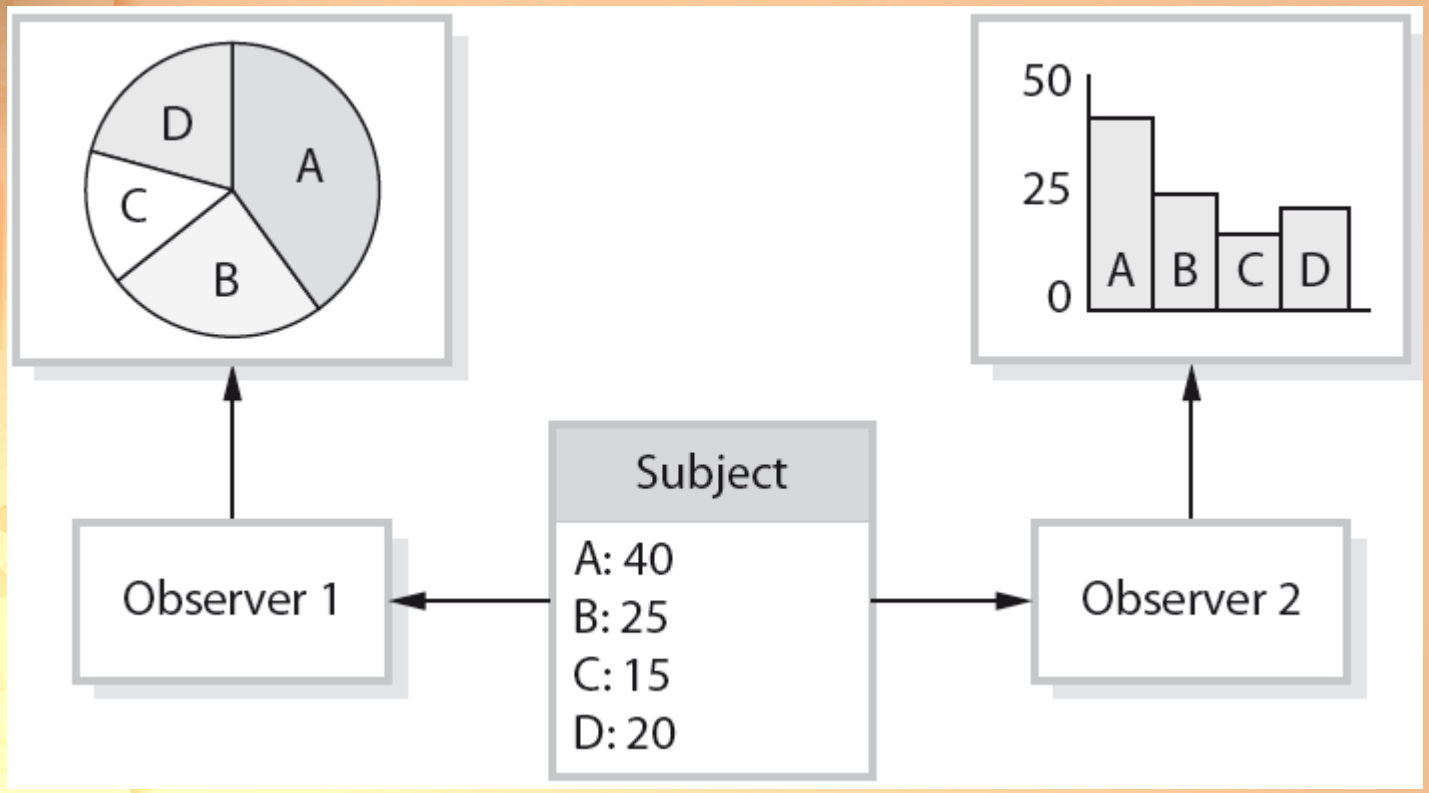
# O padrão Observer (2)



| | |
|---|---|
| **Descrição da solução:** | Trata-se de dois objetos abstratos, Subject e Observer, e dois objetos concretos, ConcreteSubject e ConcreteObject, que herdam os atributos dos objetos abstratos relacionados. Os objetos abstratos incluem as operações gerais aplicáveis em todas as situações. O estado a ser exibido é mantido no ConcreteSubject, que herda as operações de Subject permitindo adicionar ou remover Observers (cada Observer corresponde a um *display*) e emitir uma notificação quando o estado mudar.<br><br>O ConcreteObserver mantém uma cópia do estado do ConcreteSubject e implementa a interface atualizar ( ) do Observer, que permite que essas cópias sejam mantidas nessa etapa. Automaticamente, o ConcreteObserver exibe o estado e reflete as mudanças sempre que o estado é atualizado. O modelo UML do padrão é mostrado na Figura 7.10. |
| **Consequências:** | O Subject só conhece o Observer abstrato e não sabe detalhes da classe concreta. Portanto, há um acoplamento mínimo entre esses objetos. Devido a essa falta de conhecimento, as otimizações que melhoram o desempenho do *display* são impraticáveis. As alterações no Subject podem causar uma série de atualizações ligadas aos Observers relacionados para serem geradas, algumas das quais podem não ser necessárias. |

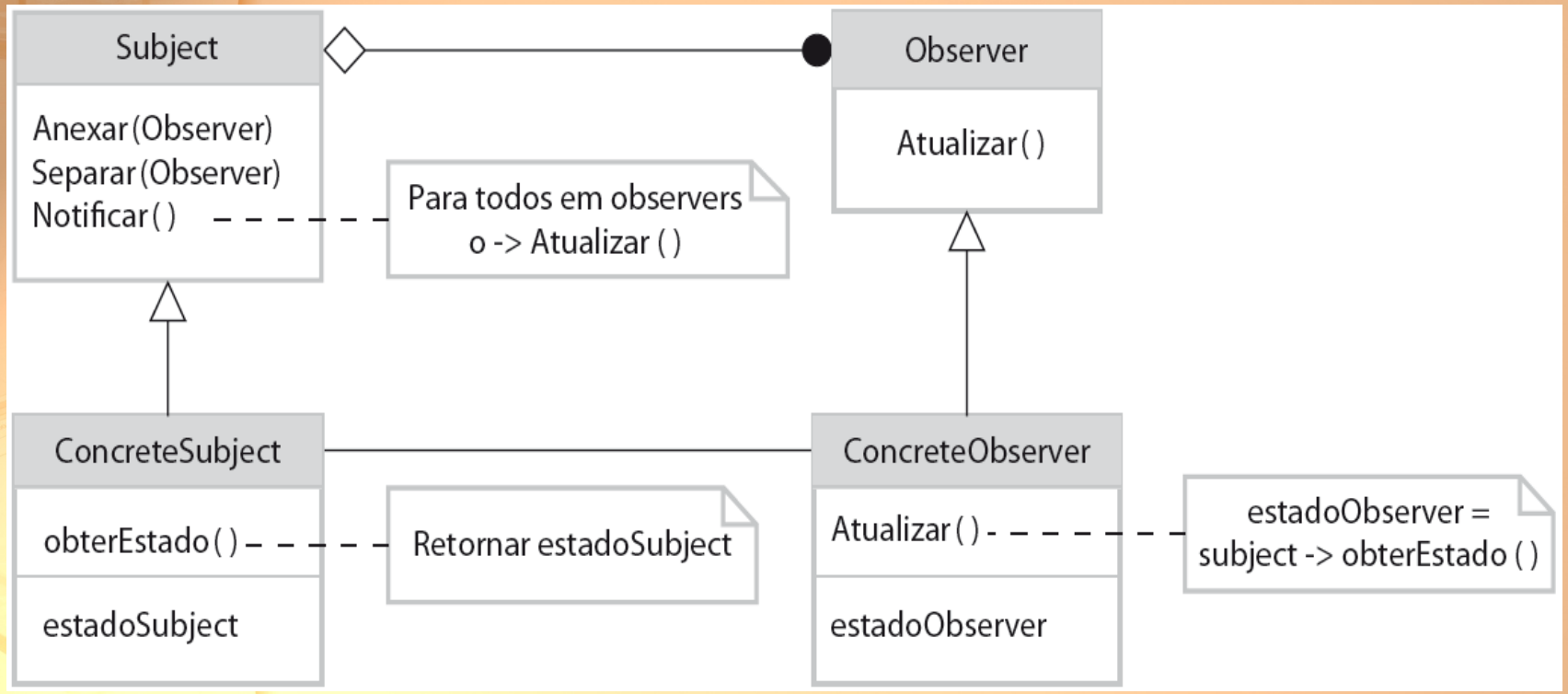
# Múltiplos displays usando o padrão Observer





# Um modelo UML do padrão Observer





## Problemas de projeto

- Para usar padrões no seu projeto, você precisa reconhecer que qualquer problema de projeto que você está encarando pode ter um padrão associado, o qual pode ser aplicado.

  - ✓ Informar a vários objetos que o estado de algum outro objeto mudou (padrão Observer).
  - ✓ Ligar as interfaces a um número de objetos relacionados, os quais, geralmente, foram desenvolvidos incrementalmente (padrão Façade).
  - ✓ Prover uma forma padrão de acesso aos elementos em uma coleção, sem se ater em como essa coleção é implementada. (padrão Iterador).
  - ✓ Permitir a possibilidade de extensão da funcionalidade de uma classe existente em tempo de execução (padrão Decorador).

## Questões de implementação

- O foco aqui não é na programação, apesar de, obviamente, essa ser importante, mas em outras questões de implementação que geralmente não são cobertas em textos sobre programação:

  - ✓ Reúso – A maioria dos softwares modernos são construídos pelo reuso de componentes e sistemas existentes. Quando se está desenvolvendo softwares, deve-se fazer o maior uso possível de códigos existentes.
  - ✓ Gerenciamento de configuração – Durante o processo de desenvolvimento, deve-se manter registro, em um sistema de gerenciamento de configuração, das várias versões diferentes de cada componente de software.
  - ✓ Desenvolvimento host-target – Geralmente, o software de produção não é executado no mesmo computador que o ambiente de desenvolvimento de software. Ao invés disso, desenvolve-se o software em um computador (o sistema host) e se executa em um computador separado (o sistema target).

## Reúso

- De 1960 a 1990, a maioria dos novos softwares foram desenvolvidos a partir do zero, escrevendo-se todo código em uma linguagem de alto nível.
  - ✓ O único reúso de software era o reúso das funções e objetos em bibliotecas de linguagem de programação.
- As pressões por redução de custos e prazo tornaram essa abordagem cada vez mais inviável, especialmente para sistemas comerciais baseados na Internet.
- Uma abordagem de desenvolvimento baseada em reúso de softwares existentes emergiu, e atualmente, essa geralmente é usada para softwares científicos e de negócios.

## Níveis de reúso

- O nível de abstração
  - ✓ Nesse nível, não se reusa o software diretamente, mas usa-se o conhecimento de abstrações bem sucedidas no projeto do seu software.
- O nível de objeto
  - ✓ Nesse nível, reusa-se diretamente os objetos de uma biblioteca, ao invés de se escrever o código.
- O nível de componentes
  - ✓ Os componentes são coleções de objetos e as classes de objeto que você reusa nos sistemas de aplicação.
- O nível de sistema
  - ✓ Nesse nível, reusa-se sistemas de aplicação inteiros.

## Custos de reúso

- Os custos de tempo gasto na busca por softwares para reúso e a avaliação de, se esses atendem ou não às suas necessidades.
- Quando se aplicam  os custos de aquisição do software reusável. Para grandes sistemas de prateleira, esses custos podem ser bem altos.
- Os custos de adaptação e configuração dos componentes do software reusável ou sistemas para refletir os requisitos do sistema que você está desenvolvendo.
- Os custos de integração entre os elementos de software reusável (caso esteja usando software de fontes diferentes) e o novo código que você desenvolveu.

# Gerenciamento de configuração



- Gerenciamento de configuração é o nome dado para o processo geral de gerenciamento de um sistema de software em mudança.

- O objetivo do gerenciamento de configuração é dar suporte ao processo de integração do sistema para que todos os desenvolvedores possam acessar o código do projeto e os documentos de uma maneira controlada, descobrir quais mudanças foram feitas e compilar e ligar os componentes para criar um sistema.

# Atividades do gerenciamento de configuração



- Gerenciamento de versões, em que é dado o suporte para manter registro das diferentes versões dos componentes de software. Sistemas de gerenciamento de versões incluem recursos para coordenar o desenvolvimento de diversos programadores.

- Integração de sistemas, em que é dado o suporte para auxiliar os desenvolvedores a definir quais versões dos componentes serão usadas para criar cada versão do sistema. Em seguida, essa descrição é usada para construir o sistema automaticamente, pela compilação e ligação dos componentes necessários.

- Rastreamento de problemas, em que é dado suporte aos usuários para reportarem bugs e outros problemas, e para permitir a todos os desenvolvedores que vejam quem está trabalhando nesses problemas e quando esses serão resolvidos.

# Desenvolvimento host-target

- A maioria dos softwares é desenvolvida em um computador (o host), mas é executado em uma máquina separada (o target).

- Mais genericamente, podemos falar de uma plataforma de desenvolvimento e uma plataforma de execução.

  - ✓ Uma plataforma é mais do que apenas o hardware.
  - ✓ Inclui o sistema operacional instalado além de outros softwares de suporte como um sistema de gerenciamento de banco de dados ou, para plataformas de desenvolvimento, um ambiente de desenvolvimento interativo.

- Geralmente, a plataforma de desenvolvimento tem diferentes softwares instalados da plataforma de execução; essas plataformas podem ter arquiteturas diferentes .

## Ferramentas de plataforma de desenvolvimento

engenharia de SOFTWARE

- Um compilador integrado é um sistema de edição orientado a sintaxe, que permita a criação, edição e compilação de códigos.
- Um sistema de depuração de linguagem.
- Ferramentas de edição gráfica, como por exemplo ferramentas para edição de modelos da UML.
- Ferramentas de teste, como o Junit, que podem executar, automaticamente, um conjunto de testes em uma nova versão de um programa.
- Ferramentas de apoio a projetos que podem auxiliar na organização do código para diferentes projetos de desenvolvimento.

# Ambiente de Desenvolvimento Integrados (IDE)



- Geralmente, as ferramentas de desenvolvimento de software são agrupadas para criarem um ambiente de desenvolvimento integrado (IDE – Integrated Development Environment).

- Um IDE é um conjunto de ferramentas de software que dá apoio a diferentes aspectos do desenvolvimento de software em um framework comum e em uma interface de usuário.

- Os IDEs são criados para dar apoio ao desenvolvimento em linguagens de programação específicas como o Java.

- O IDE da linguagem pode ser desenvolvido especialmente, ou pode ser uma instância de um IDE de uso geral com ferramentas de apoio a uma linguagem específica.

## Fatores de implantação de componente/ sistema



- Se um componente for destinado a uma arquitetura de hardware específica ou se depender de outros sistemas de software, obviamente, esse precisa ser implantado em uma plataforma que forneça suporte ao hardware e software necessários.

- Os sistemas de alta disponibilidade podem necessitar que os componentes sejam implantados em mais de uma plataforma. O que significa que, no evento de falha de uma plataforma, uma implementação alternativa do componente estará disponível.

- No caso de alto nível de tráfego de comunicação entre os componentes, geralmente, faz sentido implantá-los na mesma plataforma ou em plataformas fisicamente próximas. O que reduz o atraso entre o envio de uma mensagem por um componente e a recepção pelo outro.

# Desenvolvimento *open source*

- O desenvolvimento *open source* é uma abordagem de desenvolvimento de software na qual o código fonte de um sistema de software é publicado e os voluntários são convidados a participar no processo de desenvolvimento.

- Suas raízes estão no Free Software Foundation (www.fsf.org), que advoga que o código fonte não deveria ser proprietário, mas, deveria estar sempre disponível para que os usuários possam examiná-lo e modificá-lo como quiserem.

- O software *open source* estendeu essa ideia usando-se a Internet para recrutar uma população muito maior de desenvolvedores voluntários. Vários deles também são usuários do código.

## Sistemas *open source*

- O produto *open source* mais conhecido é, sem dúvida, o sistema operacional Linux, o qual é amplamente usado como sistema servidor e cada vez mais, como um ambiente desktop.

- Outros importantes produtos *open source* são o Java, o Apache web server e o sistema de gerenciamento de banco de dados mySQL.

# Questões de *open source*

engenharia de SOFTWARE

- O produto que está sendo desenvolvido deve usar componentes *open source*?

- Deve ser usada uma abordagem *open source* para o desenvolvimento de software?

## Negócios de *open source*

- Cada vez mais, empresas estão usando uma abordagem *open source* para o desenvolvimento.
- Seu modelo de negócios não é bom na venda de um produto de software, mas são bons no fornecimento de suporte para esse produto.
- Acreditam que envolver a comunidade *open source* permitirá o desenvolvimento mais barato e mais rápido de softwares, e criará uma comunidade de usuários para o software.

## Licenciamento de *open source*

- Um princípio fundamental do desenvolvimento *open source* é que o código fonte deve ser disponibilizado gratuitamente, o que não significa que qualquer um possa fazer o que quiser com o código.

  - ✓ Legalmente, o desenvolvedor do código (tanto uma empresa, quanto um indivíduo) ainda é o proprietário do código. Ele pode colocar restrições em como esse deve ser usado, incluindo condições vinculadas legalmente, em uma licença de software *open source*.
  - ✓ Alguns desenvolvedores *open source* acreditam que se um componente *open source* é usado para desenvolver um novo sistema, esse sistema também deve ser *open source*.
  - ✓ Outros querem permitir que seu código seja usado sem essa restrição. Os sistemas desenvolvidos podem ser proprietários e vendidos como sistemas de código fechado.

## Modelos de licença

- A GNU  General Public Licence (GPL). Essa é a chamada licença ‘recíproca’, o que significa que se você usa um software *open source* licenciado sob a GPL, você precisa tornar esse software *open source*.

- A GNU Lesser General Public License (LGPL). Essa é uma variação da licença GPL na qual você pode escrever componentes que se ligam a códigos *open source*, sem precisar publicar o código desses componentes.

- A Berkley Standard Distribution (BSD). Essa é uma licença não recíproca, o que significa que não é necessário republicar quaisquer mudanças ou modificações feitas no código *open source*. E que é possível incluir o código em sistemas proprietários comercializados.

## Gerenciamento de licenças

- Estabelecer um sistema para manter informações sobre os componentes *open source* baixados e usados.
- Estar ciente dos diferentes tipos de licenças e compreender como um componente é licenciado antes de usá-lo.
- Estar ciente dos caminhos de evolução dos componentes.
- Educar as pessoas sobre o *open source*.
- Ter sistemas de auditoria em vigor.
- Participar da comunidade *open source*.

## Pontos Importantes

- Ao desenvolver um software, é importante sempre considerar a possibilidade de reúso dos softwares existentes, tanto na forma dos componentes, quanto nos serviços ou sistemas completos.

- O gerenciamento de configuração é o processo de gerenciar as mudanças em um sistema de software em evolução. Esse é essencial quando uma equipe de pessoas está cooperando para desenvolver um software.

- A maior parte do desenvolvimento de um software é desenvolvimento host target. Usa-se um IDE em uma máquina host para desenvolver o software, o qual é transferido para uma máquina target, para a execução.

- O desenvolvimento *open source* envolve tornar o código fonte de um sistema disponível publicamente. O que significa que várias pessoas podem propor mudanças e melhorias no software.