

# Arquiteturas de Aplicações Web



# Aplicações Convencionais vs. Web

- Aplicações convencionais
  - Escritas usando uma **linguagem de programação** (ex.: Java)
- Sites de conteúdo estático
  - Escritos usando uma **linguagem de marcação** (ex.: HTML)
- Aplicações Web
  - Escritas usando uma mistura de **linguagem de marcação** (ex.: HTML) com **linguagem de programação** (ex.: Java)
  - Conteúdo dinâmico

# Estratégias para gerar conteúdo dinâmico



# Exemplo de “marcação” na “programação” (Servlet)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class AloMundo extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws IOException {
        PrintWriter out = response.getWriter();
        response.setContentType("text/html");
        out.println("<HTML><BODY>");
        out.println("<P>Servlet Alo Mundo...</P>");
        for (int i = 0; i < 10; i++) {
            out.println(i + "<BR>");
        }
        out.println("</BODY></HTML>");
    }
}
```

# Exemplo de “programação” na “marcação” (JSP)

```

<HTML>
<BODY>
<P>JSP Alo Mundo...</P>
<% for (int i = 0; i < 10; i++) { %>
    <%= i %><BR>
<% } %>
</BODY>
</HTML>

```

# Servlet x JSP

- Servlet:
  - Java é a linguagem principal
  - Indicado para implementar **regras de negócio e manipulação de dados**
- JSP:
  - HTML é a linguagem principal
  - Indicado para **interface com o usuário**

# Dificuldade

- Conciliar os dois mundos
  - Ferramentas distintas (IDE vs. Editor)
  - Habilidades distintas (Programador vs. Web Designer)
- IDE Java
  - Boa para escrever código Java
  - Ruim para escrever HTML
- Editor HTML
  - Bom para escrever HTML
  - Ruim para escrever código Java

# Necessidade

- Organizar a forma de trabalho com essas tecnologias
- Especial relevância
  - para **sistemas grandes e complexos**
  - com **equipes multidisciplinares**



# Aplicações Web

- Grande variedade de tipos de aplicação e domínios
  - Blog pessoal
  - Site de um curso
  - Ferramenta de busca
  - Rede social
  - Home-banking
  - E-commerce
- Grande variedade de requisitos não-funcionais
  - Desempenho
  - Escalabilidade
  - Robustez
  - Segurança
  - Disponibilidade
  - Portabilidade

# Arquiteturas Web

- Visam separação de responsabilidades entre os componentes da aplicação Web
  - Atendem os **requisitos não-funcionais** esperados pela aplicação
  - A um custo do seu **aumento de complexidade**
- Qual a arquitetura que melhor resolve o problema **no curto, médio e longo prazo?**
  - Curto prazo: desenvolvimento
  - Médio prazo: produção
  - Longo prazo: manutenção

# Arquiteturas Web

- As principais arquiteturas separam as responsabilidades em camadas
- Cada camada pode estar em uma ou mais máquinas diferentes
- O número de camadas (contado do lado servidor) varia em função da complexidade a ser lidada

# Arquiteturas Web

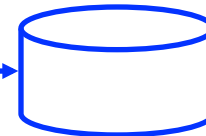
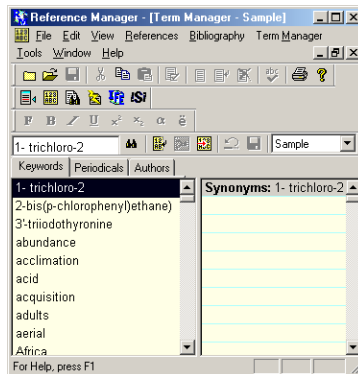
- Sistemas cliente-servidor tradicionais têm somente 1 camada no lado servidor
  - Armazenamento
- Arquiteturas em 2 camadas têm separação das entidades em
  - Apresentação
  - Armazenamento
- Arquiteturas em 3 camadas têm separação das entidades em
  - Apresentação
  - Aplicação
  - Armazenamento

# Arquitetura em 1 camada

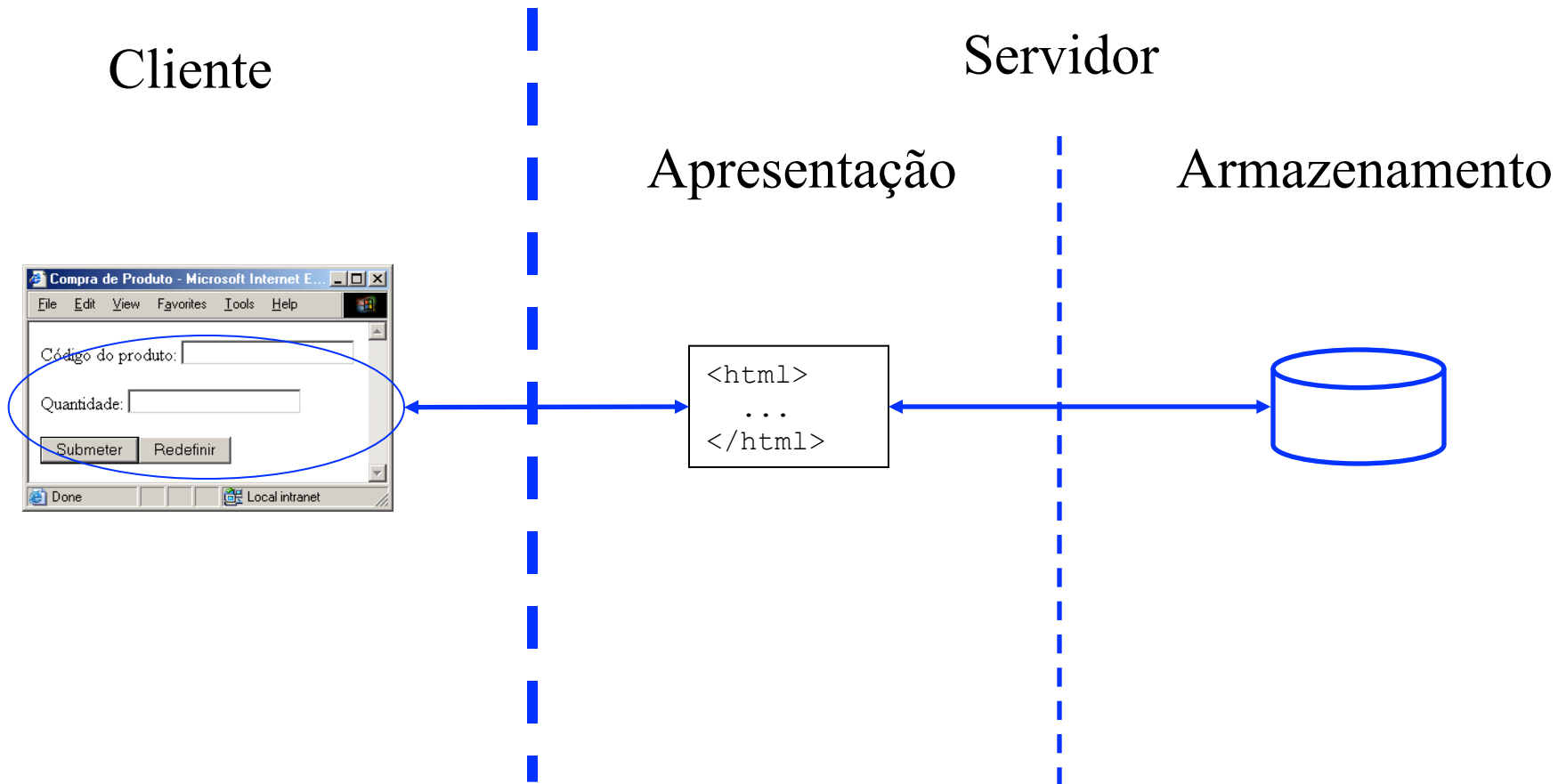
Cliente

Servidor

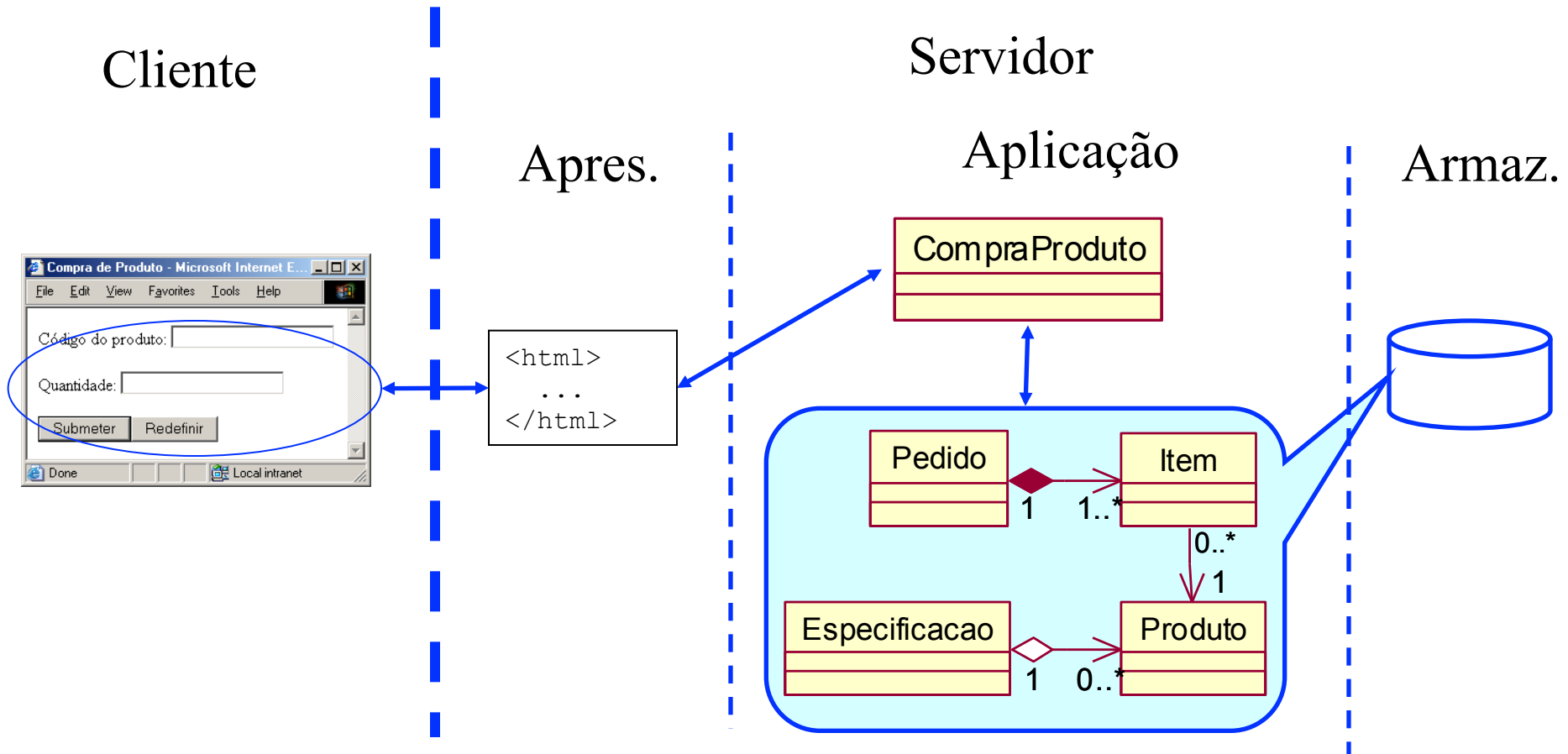
Armazenamento



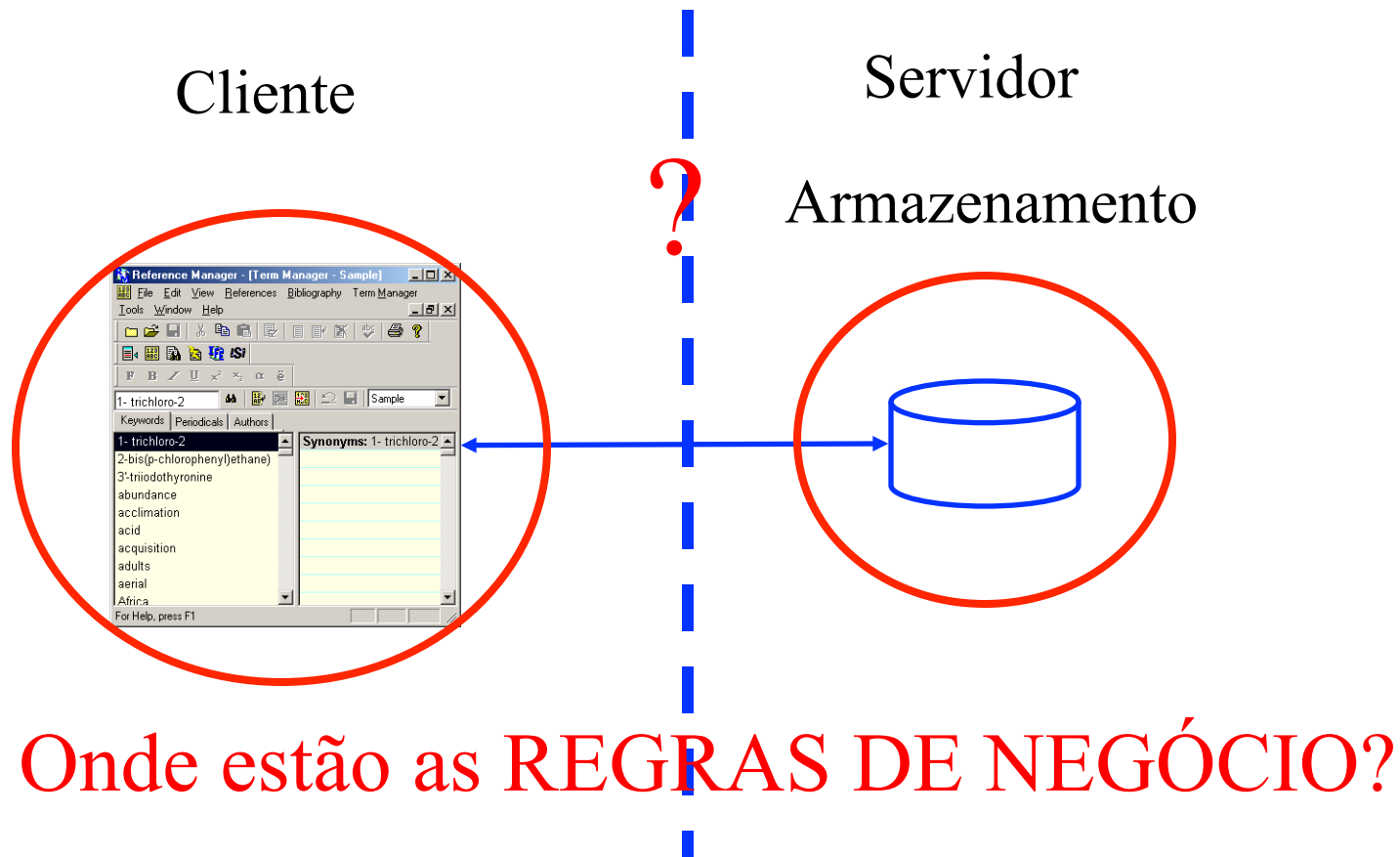
# Arquitetura em 2 camadas



# Arquitetura em 3 camadas



# Regras de negócio





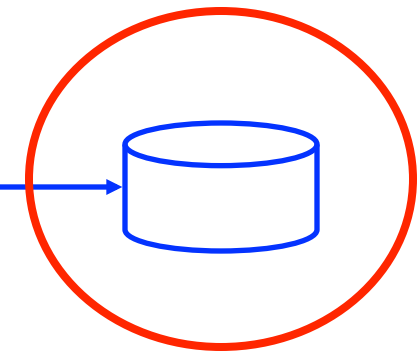
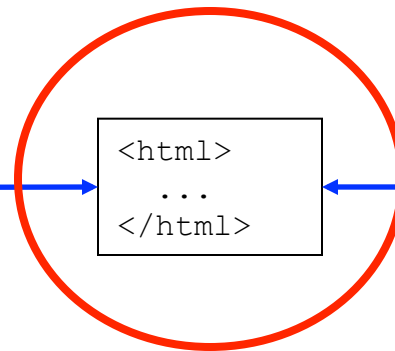
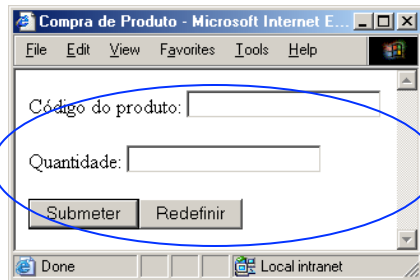
# Regras de negócio

Cliente

Servidor

Apresentação

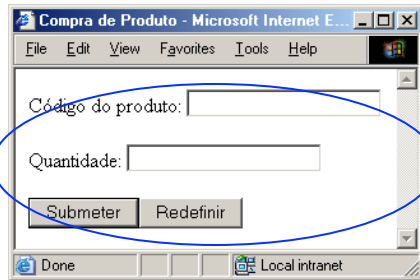
Armazenamento



Onde estão as REGRAS DE NEGÓCIO?

# Regras de negócio

Cliente



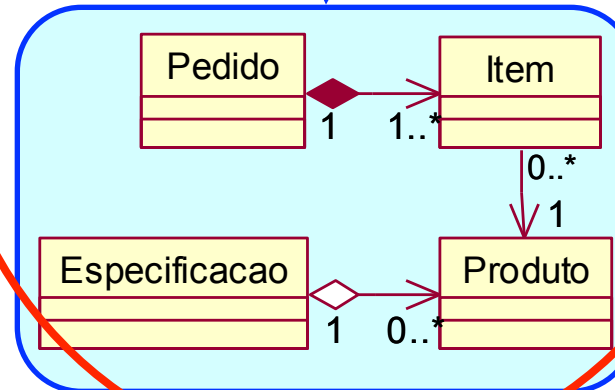
Apres.

```
<html>
...
</html>
```

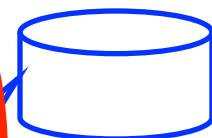
Servidor

Aplicação

CompraProduto



Armaz.



**REGRAS DE NEGÓCIO**

# Por que Java?

Java Community Process (JCP)  
Especificações (JSR)

Implementações

Ferramentas

Implementação  
de referência

Demais  
implementações

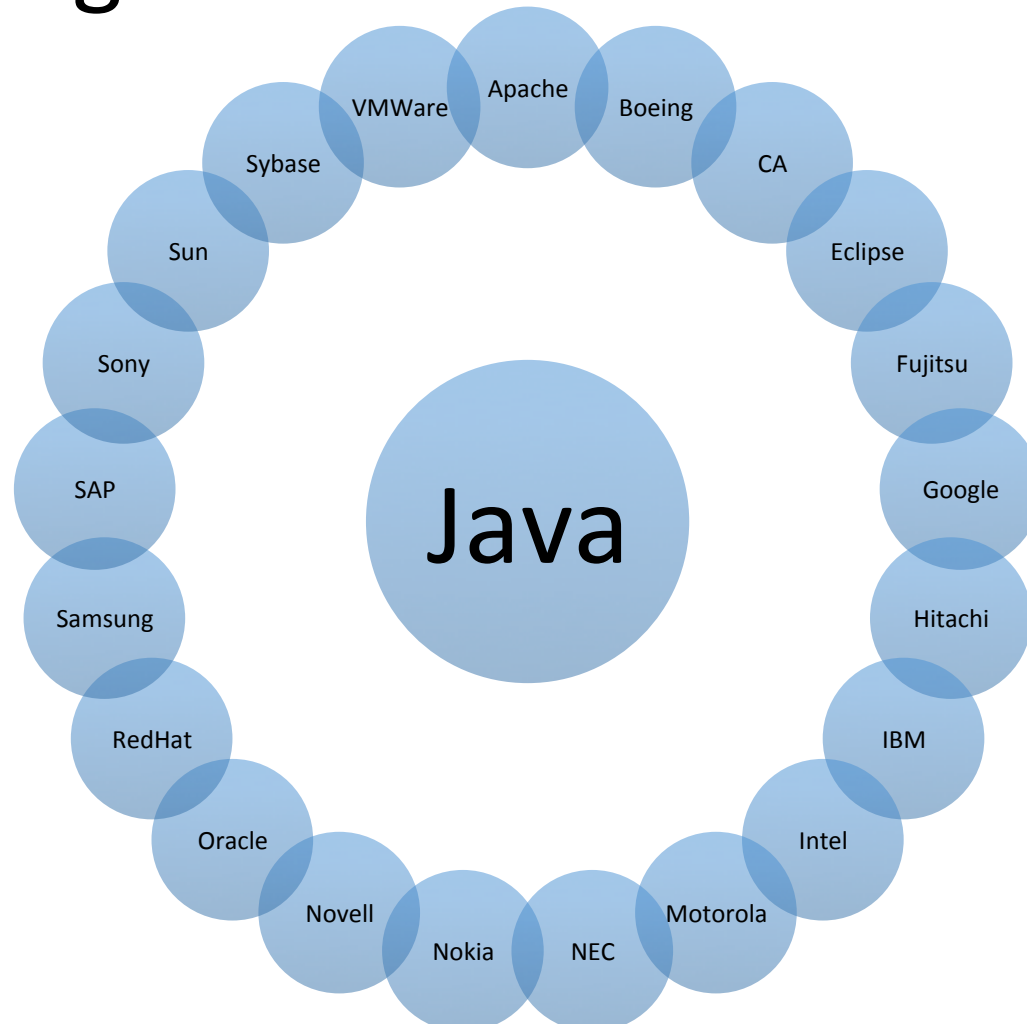
Teste de  
compatibilidade

IDEs

Ferramentas de  
apoio

# Por que Java?

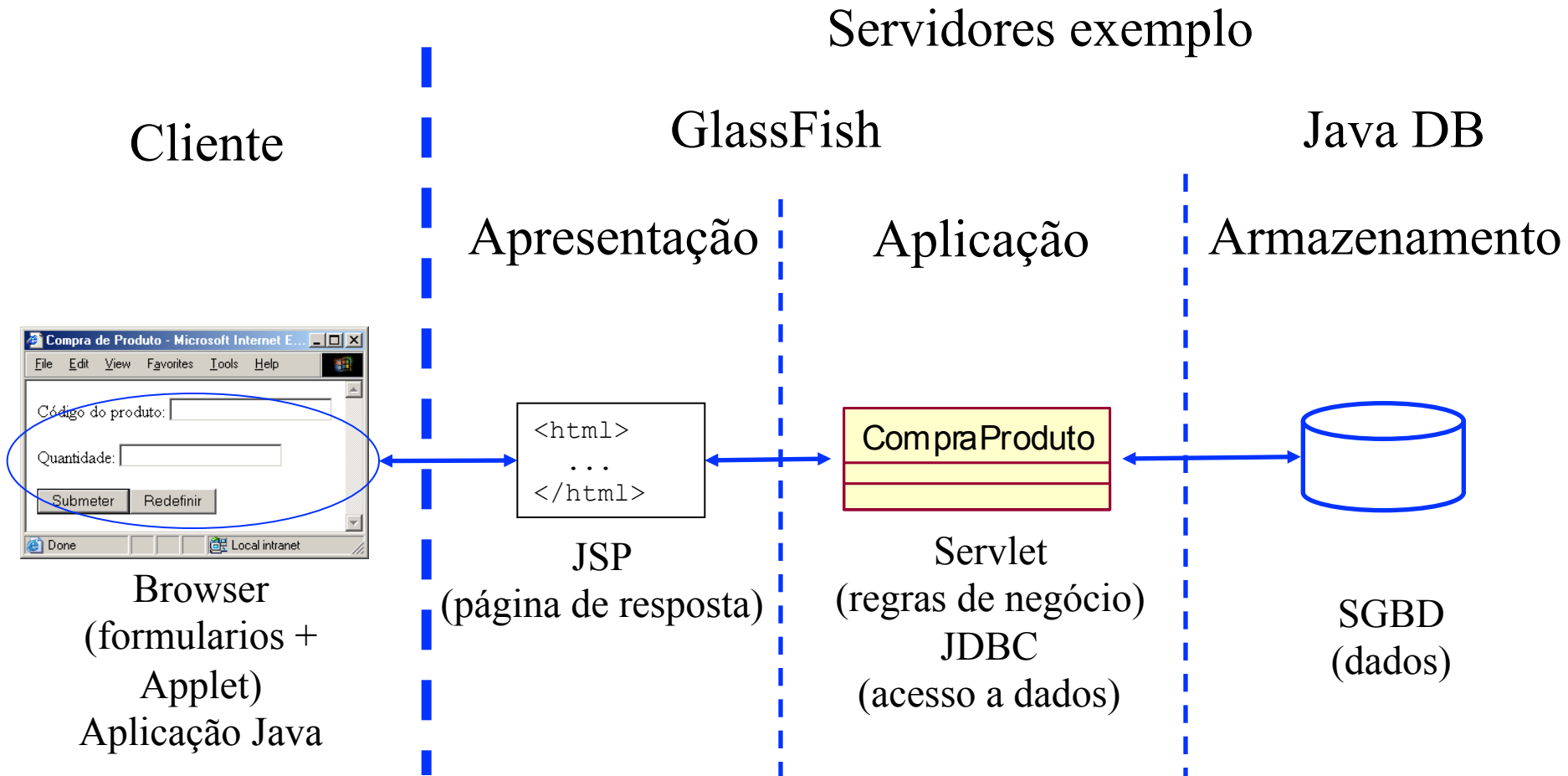
(alguns membros do JCP)



# Arquiteturas Web em Java

- Java permite a adoção de diferentes arquiteturas web
- Dentre as mais famosas estão
  - Model2 (mais simples)
  - Java EE (mais complexa)

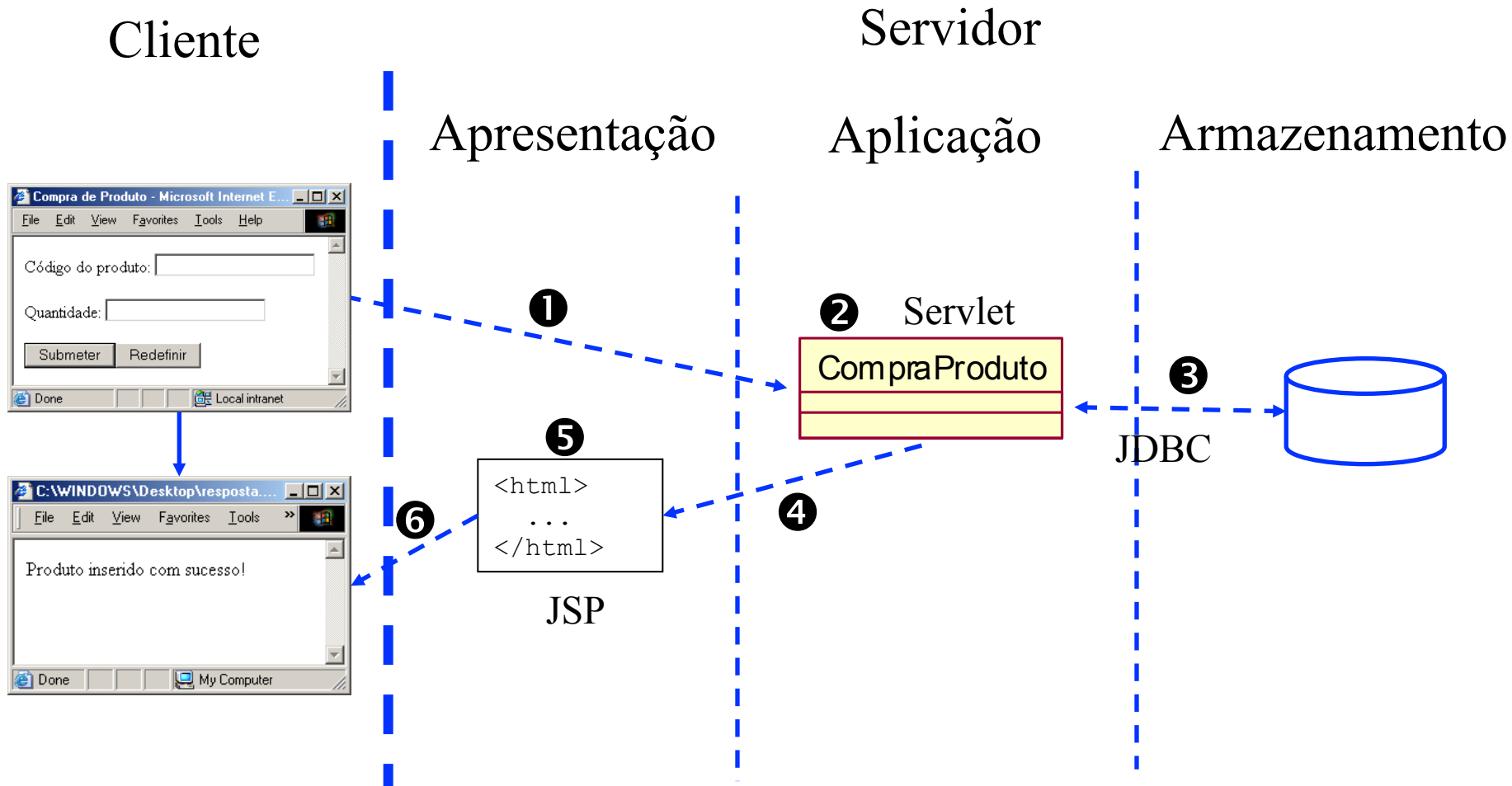
# Arquitetura Model2 em Java



# Arquitetura Model2 em Java (elementos fundamentais)

- JSP
  - Páginas HTML com código Java embutido
- Servlet
  - Classes Java que rodam em servidores
- JDBC
  - API de acesso a banco de dados em Java

# Arquitetura Model2 em Java (cenário típico)



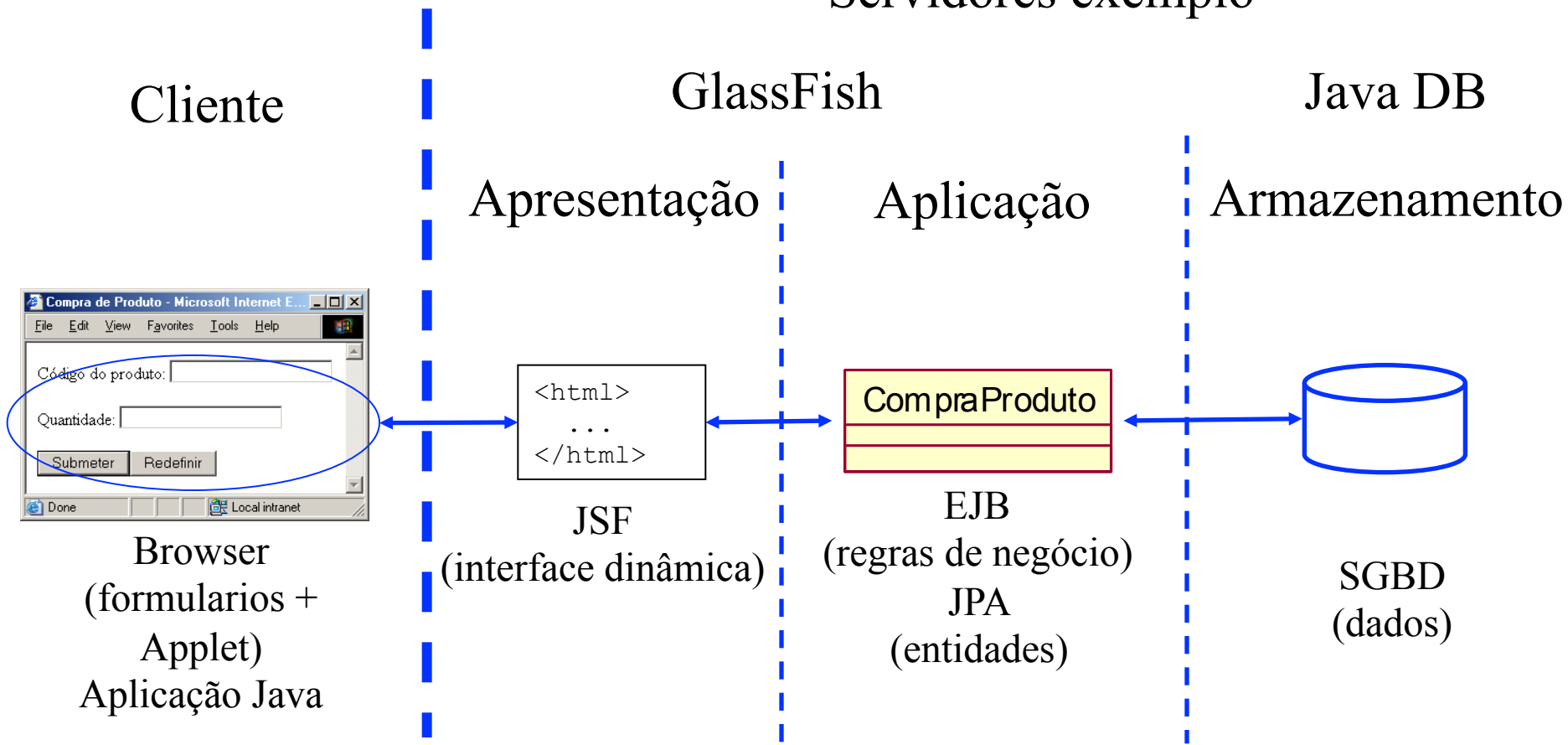


# Arquitetura Model2 em Java (cenário típico)

1. Cliente solicita um Servlet usualmente após o preenchimento de um formulário HTML
2. Servidor interpreta o Servlet na camada de aplicação
3. Se necessário, a camada de aplicação se comunica com a camada de armazenamento através de JDBC
4. Camada de aplicação redireciona o fluxo para a camada de apresentação
5. Servidor constrói uma página de resposta usando JSP
6. Servidor retorna a página de resposta.

# Arquitetura Java EE

Servidores exemplo



# Arquitetura Java EE (elementos fundamentais)

- JSF
  - Framework de apresentação que faz uso disciplinado de Servlet e JSP
- EJB
  - Componentes de negócio
- JPA
  - Entidades persistentes

# Arquitetura Java EE (cenário típico)

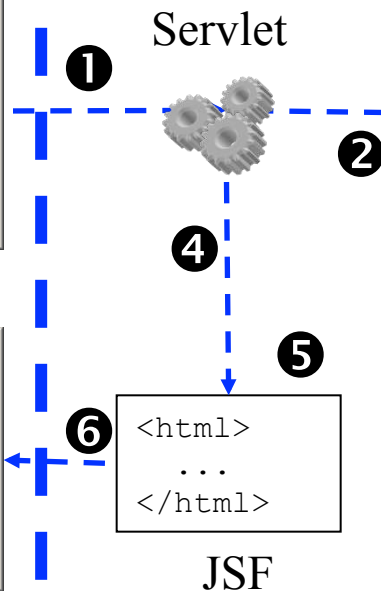
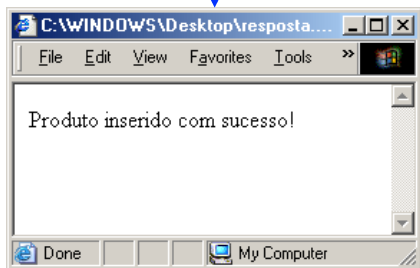
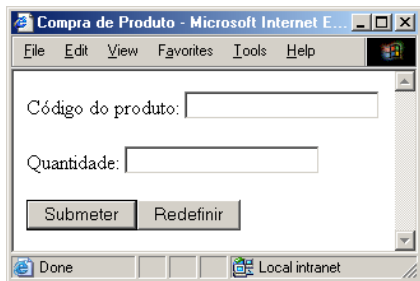
Cliente

Apresentação

Servidor

Aplicação

Armazenamento



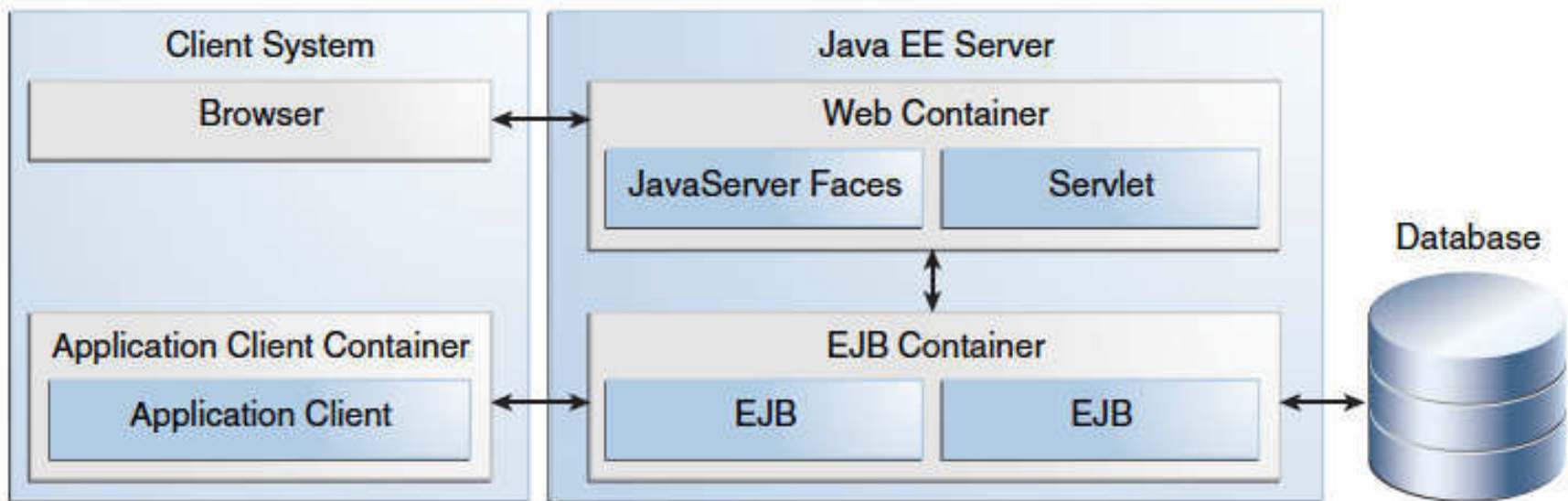
# Arquitetura Java EE (cenário típico)

1. Cliente solicita um Servlet usualmente após o preenchimento de um formulário HTML
2. Servidor interpreta o Servlet e redireciona o fluxo para um EJB na camada de aplicação
3. Se necessário, a camada de aplicação faz uso de outros EJBs ou se comunica com a camada de armazenamento através de entidades JPA
4. Camada de aplicação redireciona o fluxo para a camada de apresentação
5. Servidor constrói uma página de resposta usando JSF
6. Servidor retorna a página de resposta

# Containers

- Infraestrutura capaz de oferecer serviços básicos para códigos Java
  - Segurança
  - Transação
  - Lookup e injeção de dependências e recursos
  - Conectividade remota
  - Gestão do ciclo de vida

# Containers



Fonte: livro Java EE 7 Tutorial

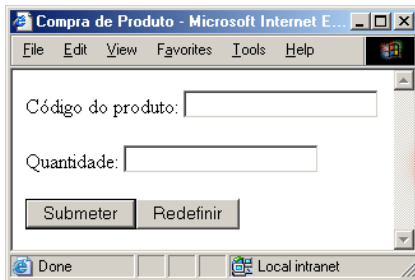
# Containers

- Container Web
  - Interface entre componentes web (Servlets, JSP ou JSF) e o servidor Java EE
- Container EJB
  - Interface entre componentes EJB e o servidor Java EE
- Container Cliente
  - Interface entre aplicações Java e o servidor Java EE



# Comunicação cliente-servidor

Cliente



Servidor



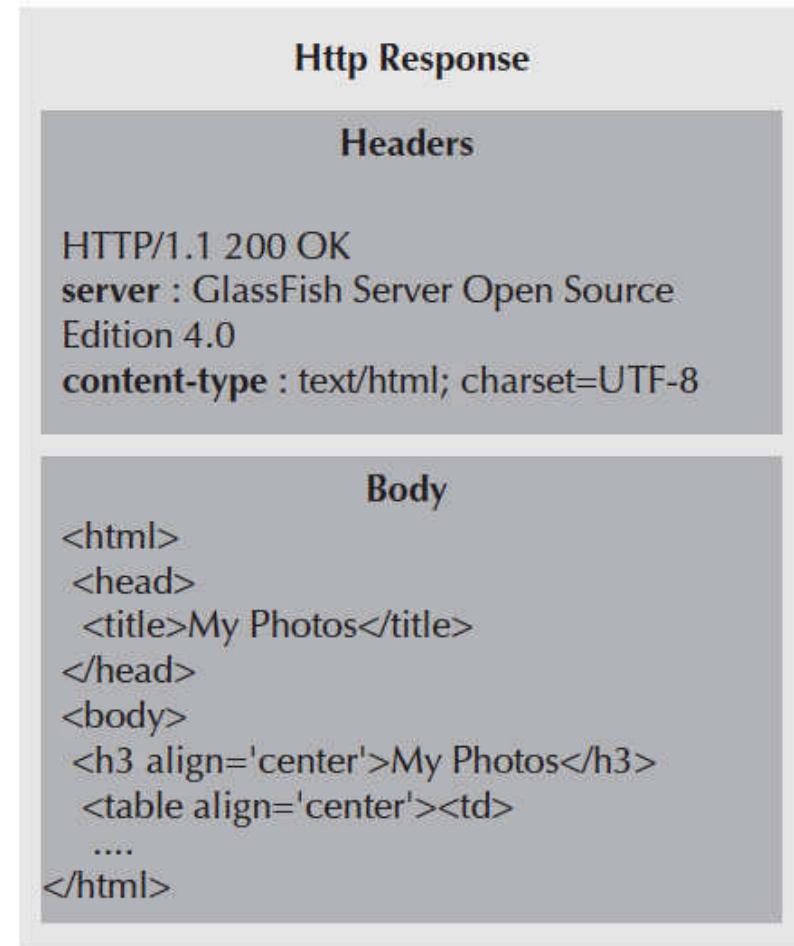
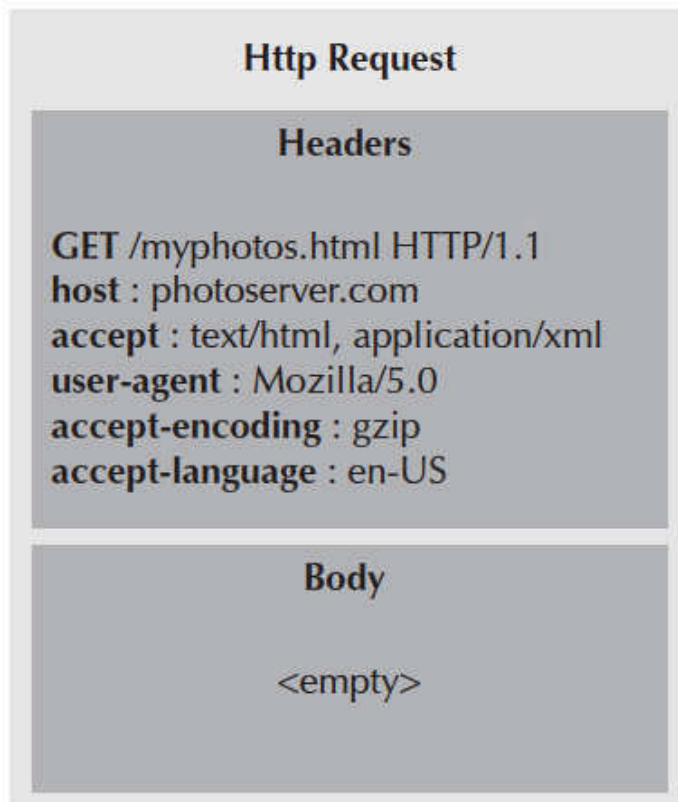
Protocolo HTTP



# Protocolo HTTP

- O protocolo HTTP fornece um mecanismo simples de requisição-resposta
  - Sem manutenção de estado
  - Independente do tipo do conteúdo
- Partes de uma mensagem HTTP
  - Requisição (método + URL)/Resposta (código)
  - Cabeçalho (pares chave-valor)
  - Corpo da mensagem

# Protocolo HTTP



Fonte: livro Java EE 7: The Big Picture

# Protocolo HTTP

## (Alguns métodos de requisição)

- GET
  - Consulta dados do servidor
  - Permite passagem de parâmetros, que aparecem na URL
  - Não deve alterar o estado do servidor
- POST
  - Insere dados no servidor, sem limite de tamanho
  - Útil para enviar dados sensíveis, pois não ficam visíveis na URL
  - Pode alterar o estado do servidor
- PUT
  - Atualiza dados no servidor
- DELETE
  - Remove dados do servidor

# Protocolo HTTP

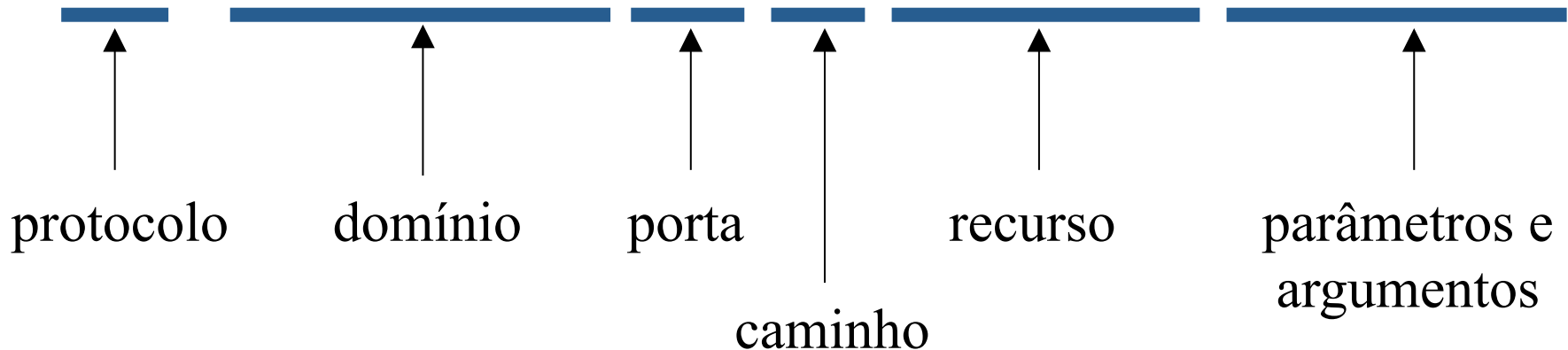
## (Alguns códigos de resposta)

- 200 OK
- 401 Not Authorized
- 403 Forbidden
- 404 Not Found
- 408 Request Timeout
- 429 Too Many Requests
- 500 Internal Server Error
- 503 Service Unavailable

# Protocolo HTTP (URL)

- Uma URL é um conjunto de informações de identificação de recurso:

http://www.xpto.com:8080/pub/exemplo.jsp?p1=v1&p2=v2



# Arquiteturas de Aplicações Web

