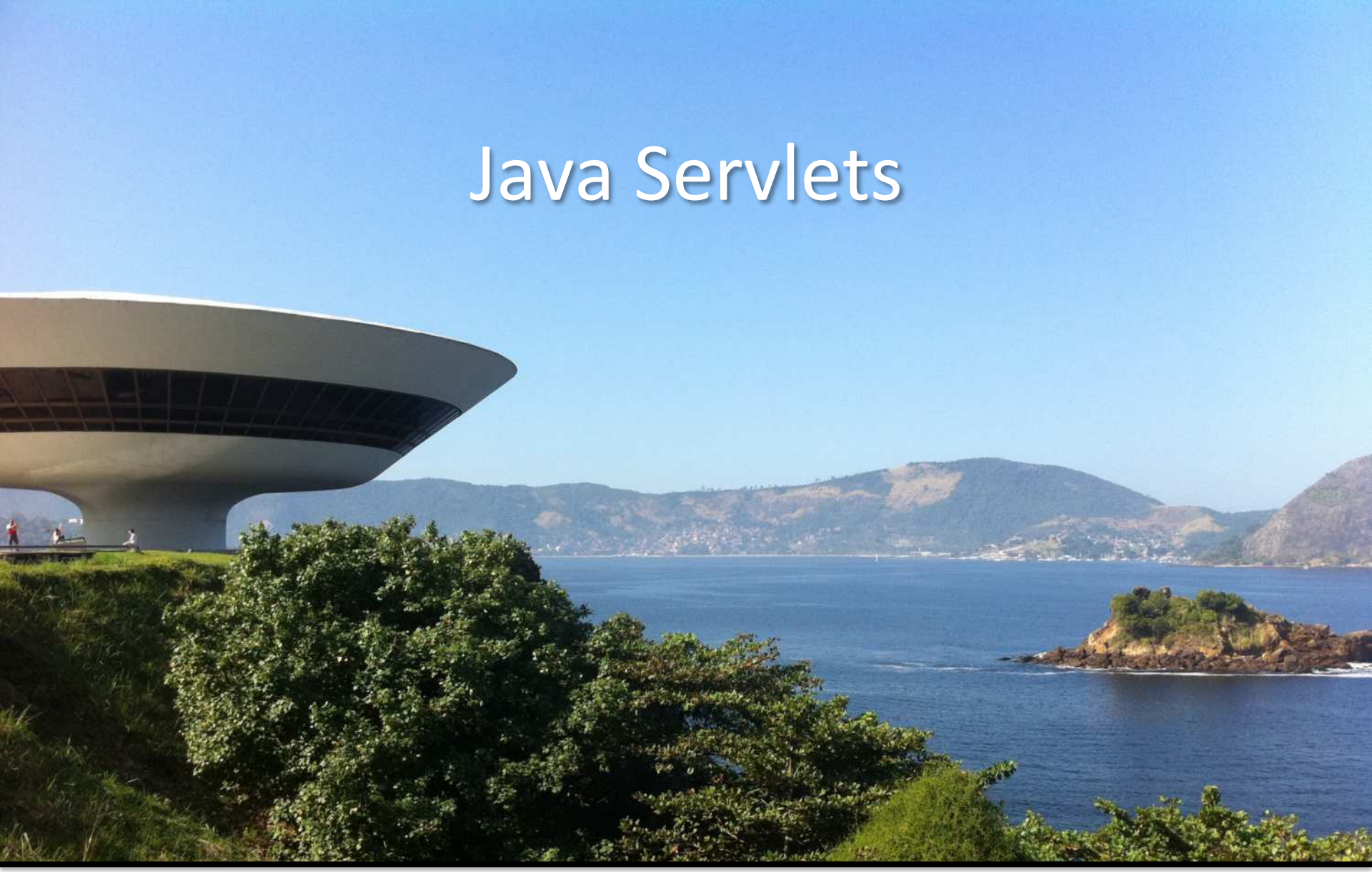
Java Servlets



Instituto de Computação

Leonardo Gresta Paulino Murta

leomurta@ic.uff.br

Especificação/IDE/Implementação

- Esse curso foi preparado em 03/2015 usando a seguinte especificação, IDE e implementação
- Especificação
  - Java Servlet 3.1 (05/2013, JEE 7)
  - Java Servlet 3.0 (12/2009, JEE 6)
- IDE
  - JDK 8u40
  - NetBeans 8.0.2 na distribuição Java EE
- Implementação
  - GlassFish 4.1 (vem no NetBeans)

Agenda

- O que são Servlets?
- Arquitetura
- Hierarquia
- Ciclo de vida
- Empacotamento
- Passagem de Parâmetros
- Concorrência
- Armazenamento de dados
  - Sessões
  - Cookies
- Uso de recursos externos

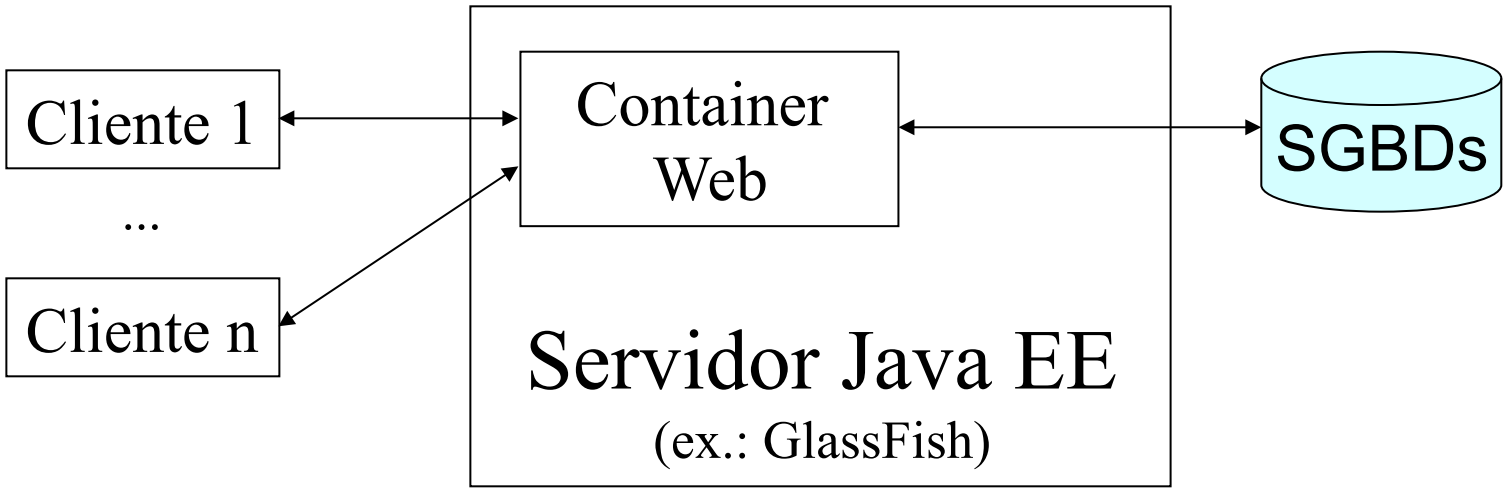
O que são Servlets ?

- Classes Java que estendem funcionalidades de servidores web para geração de conteúdo dinâmico
- Adotam o modelo de programação requisição-resposta
- Programação “no lado” do servidor
- “Substituem” scripts CGI, oferecendo
  - Escalabilidade
  - Portabilidade
  - Facilidade de desenvolvimento

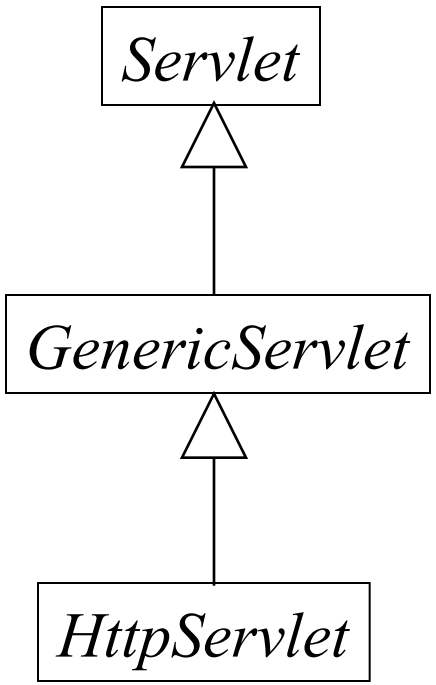
Arquitetura

- Baseado em arquitetura cliente-servidor
- Os clientes
  - Requisitam uma página ao servidor, passando informações
  - Recebem a resposta
- O servidor
  - Trata as conexões (necessita-se de controle de concorrência)
  - Processa as informações passadas
  - Gera uma página de retorno.

Arquitetura



Hierarquia de *Servlets*



- **Interface *Servlet*:** Fornece um nível primitivo para tratamento de conexões, independentemente de protocolo
- **Classe *GenericServlet*:** Implementa a interface *Servlet* para facilitar futuras heranças
- **Classe *HttpServlet*:** Estende a classe *GenericServlet* para o tratamento de conexões HTTP

Ciclo de vida (métodos básicos)

- A **interface Servlet** define três métodos fundamentais
- init
  - Será chamado na primeira requisição ao servlet
  - Deve executar a inicialização do servlet
- service
  - Será chamado em todas as requisições ao servlet
  - Deve executar o serviço para o qual o servlet foi projetado
- destroy
  - Será chamado quando o servlet for destruído (pelo administrador ou por um shutdown do servidor web)
  - Deve liberar os recursos alocados pelo servlet

Ciclo de vida (métodos de tratamento de serviços)

- A **classe HttpServlet** implementa o método service
  - Verifica qual serviço HTTP está sendo requisitado
  - Repassa a chamada para o método específico, que deve ser implementado pelo desenvolvedor
- doGet
  - Trata as conexões HTTP GET
  - Passagem de parâmetros pela URL (visível ao usuário)
- doPost
  - Trata as conexões HTTP POST
  - Passagem de parâmetros pelo cabeçalho HTTP (invisível ao usuário)
- doPut
  - Trata as conexões HTTP PUT
- doDelete
  - Trata as conexões HTTP DELETE

Ciclo de vida (argumentos)

- Todos os métodos de tratamento de serviços recebem como argumento objetos que representam a requisição e a resposta
- HTTPServletRequest:
  - Encapsula a comunicação de chamada, do cliente para o servidor
- HTTPServletResponse
  - Encapsula a comunicação de retorno, do servidor para o cliente.

Ciclo de vida (HttpServletRequest)

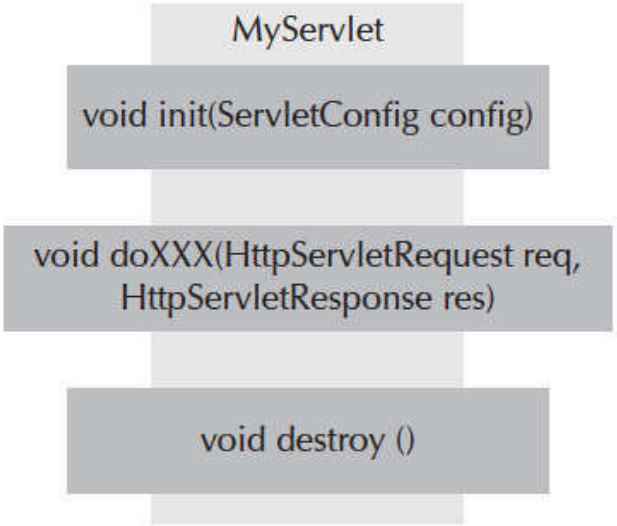
- Métodos de suporte à coleta de parâmetros do formulário
  - getParameterNames
  - getParameter
  - getParameterValues
- Método de acesso direto à stream de entrada
  - getInputStream

Ciclo de vida (HttpServletResponse)

- Método de suporte à escrita da página
  - getWriter
- Método de acesso direto à stream de saúda
  - getOutputStream

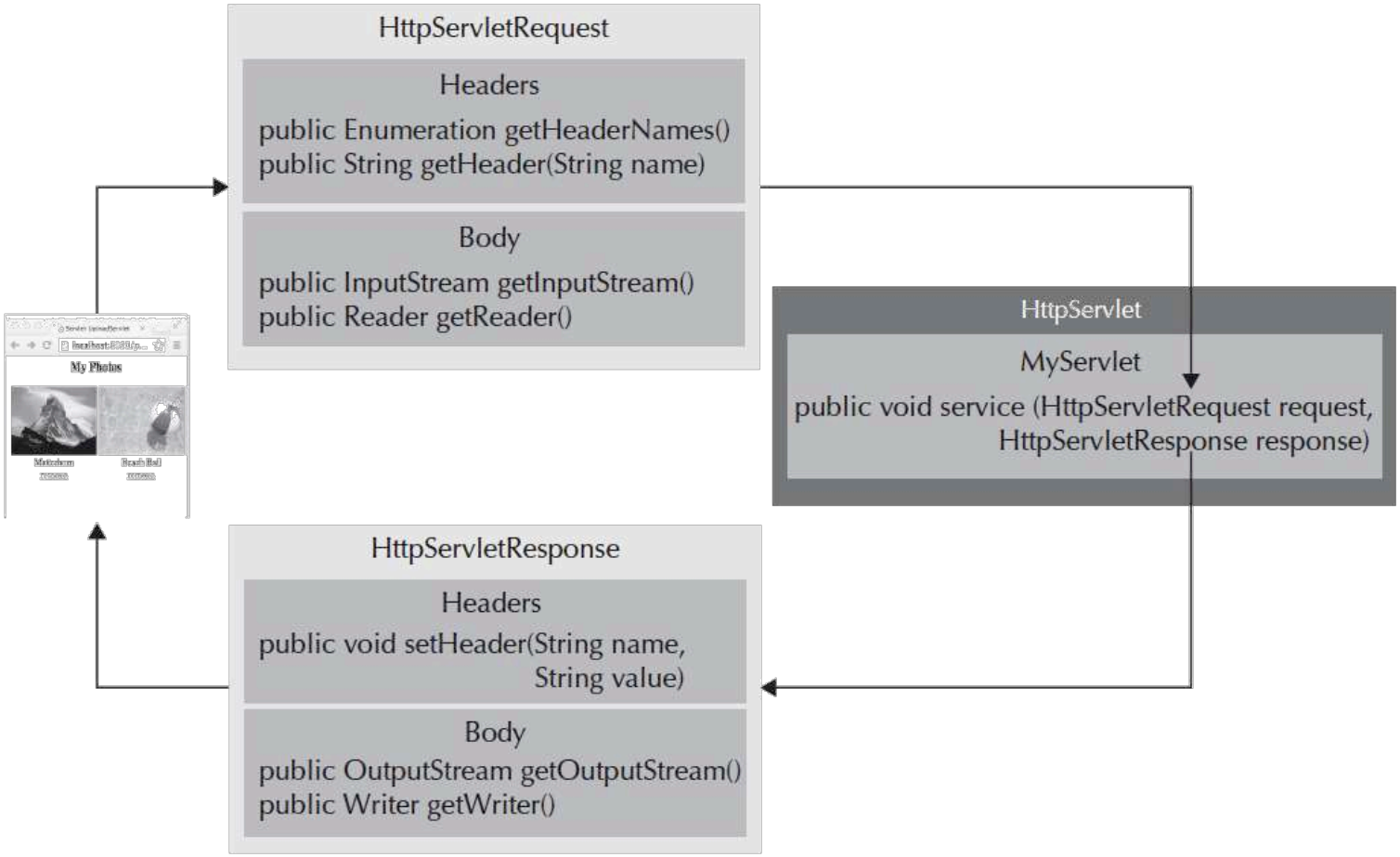
Ciclo de Vida

1. Container instancia o Servlet
2. Container chama o método init() do Servlet
3. Container coloca o Servlet disponível na URL indicada
4. Para cada requisição da URL:
   - Container chama o método do...() do Servlet
5. Container chama o método destroy() do Servlet



Fonte: Livro Java EE 7: The Big Picture

Ciclo de Vida



Fonte: livro Java EE 7: The Big Picture

Implementando doGet e doPost

- Passos
  1. Ler os dados da requisição
  2. Escrever o header da resposta
  3. Obter o PrintWriter para escrita da resposta
  4. Escrever a resposta
- Importante
  - Incluir sempre o tipo de conteúdo no header da resposta
  - Sempre definir o tipo de conteúdo antes de começar a escrever a saída

Exemplo: Alo Mundo

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class AloMundoServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response) throws IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.println("<HTML><BODY>");
    out.println("<P>Alo Mundo!</P>");
    out.println("</BODY></HTML>");
    out.close();
  }
}
```

Descritor de Implantação

- Via XML
  - Arquivo /WEB-INF/web.xml
  - Vantagem de separação de responsabilidades
- Via anotação
  - Marcações no próprio código fonte
  - Vantagem de facilidade de manutenção

Descritor de Implantação (via XML)

- Dois elementos mais usados neste descritor : `<servlet>` e `<servlet-mapping>` :
- `<servlet>` : associa um nome de *servlet* a seu nome "completo" (*fully-qualified name*)

```
<servlet>
  <servlet-name>AloMundo</servlet-name>
  <servlet-class>AloMundoServlet</servlet-class>
</servlet>
```

- `<servlet-mapping>` : associa um caminho a um determinado *servlet*

```
<servlet-mapping>
  <servlet-name>AloMundo</servlet-name>
  <url-pattern>/alomundo</url-pattern>
</servlet-mapping>
```

Descritor de Implantação (via Anotação)

- Usando a anotação @WebServlet na declaração da classe do Servlet
  - Caminho relativo completo (ex: “/MeuServlet”)
  - Caminho relativo com wildcard (ex: “/MeuServlet/*”)
  - Extensão (ex.: “*.jpg”)
- Resolução de conflitos se dá nessa ordem

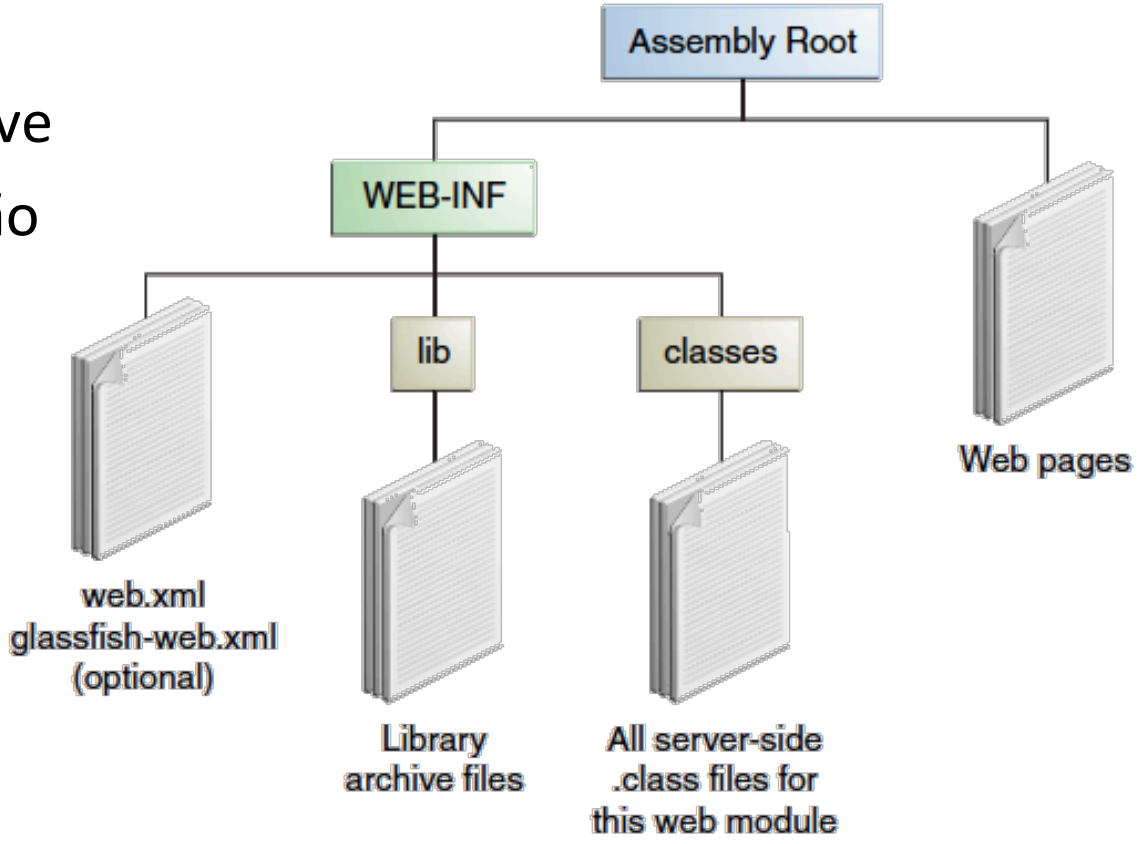
Descritor de Implantação (via Anotação)

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/alomundo")
public class AloMundoServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response) throws IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.println("<HTML><BODY>");
    out.println("<P>Alo Mundo!</P>");
    out.println("</BODY></HTML>");
    out.close();
  }
}
```

Empacotamento

- Arquivo WAR
  - WAR = Web Archive
  - Contém a aplicação web
  - Zip com extensão .war
  - Deve seguir uma estrutura de diretórios predefinida



Fonte: livro Java EE 7 Tutorial

Empacotamento

- <APLICACAO>/
  - Contém páginas HTML estáticas, jsp, etc
- <APLICACAO>/WEB-INF/web.xml:
  - Descritor de implantação da aplicação
  - Descreve servlets e outros componentes que constituem a aplicação
- <APLICACAO>/WEB-INF/classes/:
  - Contém as classes compiladas JAVA da aplicação (servlets e as demais)
  - Se a aplicação está organizada em pacotes, a estrutura de diretórios abaixo deste deve ser respeitada
- <APLICACAO>/WEB-INF/lib/:
  - Contém as bibliotecas (.jar) utilizadas na aplicação

Empacotamento

```
\exemplo
    \WEB-INF
        \classes
            \meupacote1
                TesteServlet.class
            \meupacote2
                \subpacote1
                    ClasseApoio.class
            Teste2Servlet.class
        \lib
            biblioteca.jar
        web.xml
    index.html
    teste.jsp
```

Empacotamento (ciclo de trabalho)

Exercício

- Criar servlet Alo mundo, listando números de 0 a 99
  - Criar web.xml em WEB-INF, associando o servlet ao endereço http://localhost:8080/exercicio/alomundo
  - Depois de testar, remover o web.xml e usar a anotação @WebServlet para gerar o mesmo efeito

Exercício

- Fazer um Servlet que informa todas as informações contidas no header
  - Usar request.getHeaderNames() para pegar todos os nomes
  - Usar request.getHeader(String) para pegar o valor associado a um nome específico

Passagem de parâmetros

- Para acessar os parâmetros em HttpServletRequest
  - Enumeration getParameterNames()
    - Fornece o nome de todos os parâmetros
  - String getParameter(String name)*
    - Fornece o valor de um dado parâmetro
    - Caso existam mais de um valor, o primeiro é retornado
  - String[] getParameterValues(String name)*
    - Fornece todos os valores de um dado parâmetro

- * null é retornado caso o parâmetro não exista.

Exercício

- Evolua o exercício anterior permitindo que o usuário informe o valor inicial e o valor final da contagem
  - Via GET, sem formulário http://localhost:8080/exemplo/alomundo?inicio=10&fim=30
- Em seguida, faça o mesmo utilizando POST
  - Nesse caso, criar formulário e usar o método doPost(...)
- Dica: para transformar String em int, use
  - int Integer.parseInt(String);

Concorrência

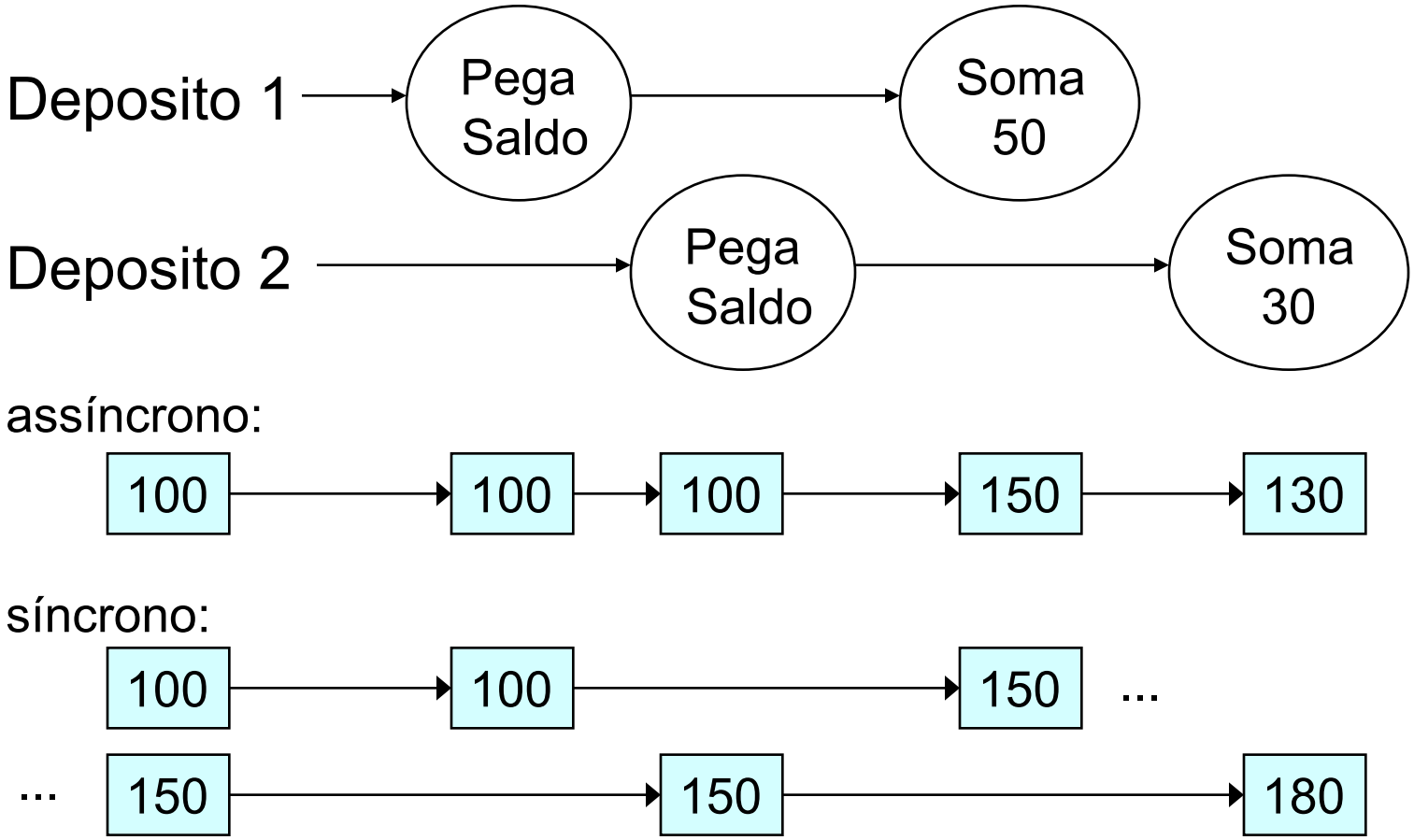
- O Container decide quando instanciar um Servlet
  - Pool: múltiplas instâncias ativas ao mesmo tempo (pool)
  - Economia de recursos: nenhuma instância ativa, com instanciação por demanda
- O desenvolvedor deve se preocupar
  - Com acesso concorrente nos métodos do...()
  - Em nunca guardar dados em atributos do Servlet

Concorrência

- Java possibilita o uso de threads
  - Múltiplas linhas de execução de um mesmo trecho de código
  - Para cada chamada a um determinado servlet, é criado uma thread
- É necessário garantir que regiões críticas do código serão respeitadas
  - Somente uma thread deve estar em uma região crítica a cada momento
  - A região crítica deve ser demarcada para ser sincronizada

Concorrência

Exemplo: Deposito em uma conta bancária:



Concorrência

- Para garantir o acesso síncrono a uma região crítica:
  - Definia que um determinado método é uma região crítica, através do uso do **modificador synchronized** no cabeçalho deste método
  - Esse método deve conter o mínimo possível de código, para evitar gargalos
  - assíncrono: public int deposita(Conta conta, Valor valor);
  - síncrono: public synchronized int deposita(Conta conta, Valor valor);

Armazenamento de Dados

- É comum precisar armazenar dados para processamento futuro
- Campo escondido
  - Dado presente na página de retorno do usuário
- Sessão
  - Entre diferentes interações do usuário
  - Mesma execução do browser
- Cookie
  - Entre diferentes sessões do usuário
  - Diferentes execuções do browser
- Aplicação
  - Entre diferentes usuários
  - Mesma execução do servidor de aplicação
- Banco de dados
  - Entre diferentes execuções do servidor de aplicação

Campos escondidos

- Mecanismo alternativo de gerenciamento de sessão
  - Cada formulário contém campos *hidden* para transferir as informações de sessão em conjunto com seus controles:

```
<input type=hidden name=total value="15">
```

  - O gerenciamento de sessão funciona mesmo sem *cookies*!

- Problemas:
  - O usuário pode alterar o conteúdo dos campos *hidden* alterando o código HTML das páginas
  - Uso não recomendado

Exercício

- Fazer um somatório com monitor de acessos
  - Transformar a aplicação anterior em somatório
  - Mostrar o resultado do somatório
  - Ao final, listar quantas vezes o usuário utilizou o serviço

Exercício

- Chamada

http://localhost:8080/exemplo/somatorio?inicio=1&fim=4

- Resultado

O somatório de 1 a 4 é 10.

Você usou o serviço 3 vezes.

Controle de Sessões

- Exemplo
  - Em um site de comercio eletrônico é necessário permitir que o cliente escolha quais produtos deseja comprar
  - Ao termino, tem que ser possível determinar quais produtos foram escolhidos
- O controle de sessões pode ser efetuado através de três mecanismos
  - Cookie
  - Parâmetro (URL)
  - Protocolo SSL
- Decisão do container, transparente para o desenvolvedor

Controle de Sessões

- Método **getSession** existente no objeto (recebido como parâmetro) da classe **HttpServletRequest**.
- A chamada ao método getSession deve ser efetuada antes de qualquer chamada ao método getWriter da classe HttpServletResponse
- O método getSession retorna um objeto da classe HttpSession, onde é possível
  - Ler todos os atributos armazenados com o método **getAttributeNames()**
  - Armazenar valores, através do método **setAttribute(nome, valor)**
  - Recuperar valores, através do método **getAttribute(nome)**
- O método **setMaxInactiveInterval** da classe HttpSession permite a configuração do tempo máximo de atividade de uma seção
- O método **invalidate** da classe HttpSession permite a finalização da seção

Controle de Sessões

- Exemplo de concatenador

```
...
HttpSession session = request.getSession();
String texto = request.getParameter(”texto");
if (texto != null) {
  String textoAntigo =
   (String)session.getAttribute(“texto”);
  String textoNovo = textoAntigo + texto;
  session.setAttribute(“texto”, textoNovo);
}
...
```

Exercício

- Repita o exercício anterior usando sessão no lugar de campo escondido

Cookies

- Cookies servem para armazenar por tempo determinado alguma informação no browser do cliente
- Usos mais comuns são para
  - Deixar o cliente acessar o sistema sem pedir senha
  - Memorizar quantas vezes aquele browser já acessou o site
  - Personalizar propagandas
- Os cookies existentes são acessados através do método **getCookies** existente no objeto da classe **HttpServletRequest**
- O método getCookies retorna um array de objetos da classe **Cookie**
- Para cada opjeto da classe Cookie, é possível
  - Armazenar valores, através do método **setValue(String)**
  - Recuperar valores, através do método **getValue()**

Cookies

- Adição de Cookie no browser do usuário
  - Chamar o método **addCookie** da classe **HttpServletResponse**, passando como parâmetro o novo cookie
  - A chamada ao método addCookie deve ser efetuada antes de qualquer chamada ao método getWriter da classe HttpServletResponse
  - O método **setMaxAge** determina por quanto tempo, em segundos, o cookie é válido

Cookies

- Exemplo de **Cookies** (concatenador):

```
...
Cookie[] cookies = request.getCookies();
String textoAntigo = “”;
for (int i = 0; i < cookies.length; i++) {
  Cookie cookie = cookies[i];
  if (cookie.getName().equals(“texto”))
    textoAntigo = cookie.getValue();
}
String texto = request.getParameter(”texto");
String textoNovo = textoAntigo + texto;
Cookie cookie = new Cookie(“texto”, textoNovo);
response.addCookie(cookie);
...
```

Exercício

- Repita o exercício anterior informando também o número total de vezes que o usuário utilizou o serviço
- Resultado

O somatório de 1 a 4 é 10.

Você usou o serviço 3 vezes nessa sessão.
Você usou o serviço 5 vezes nesse browser.

Sessões x Cookies

- Sessões podem fazer uso de cookies
  - API de alto nível
- Sessões "morrem" quando o browser é fechado
- Cookies persistem nas máquinas cliente

Aplicação

- Permite armazenar informações de forma que qualquer thread de qualquer servlet possa acessar
- Objeto do tipo **ServletContext** Pode ser obtido de **request.getServletContext()**
  - Representa o container
  - Único para todos os Servlets da aplicação
- Guarda um Map de "atributos" onde podem ser escritos/lidos dados temporários dos Servlets
  - Ler todos os atributos armazenados com o método **getAttributeNames()**
  - Armazenar valores, através do método **setAttribute(nome, valor)**
  - Recuperar valores, através do método **getAttribute(nome)**
- Mesma API do objeto que representa a sessão

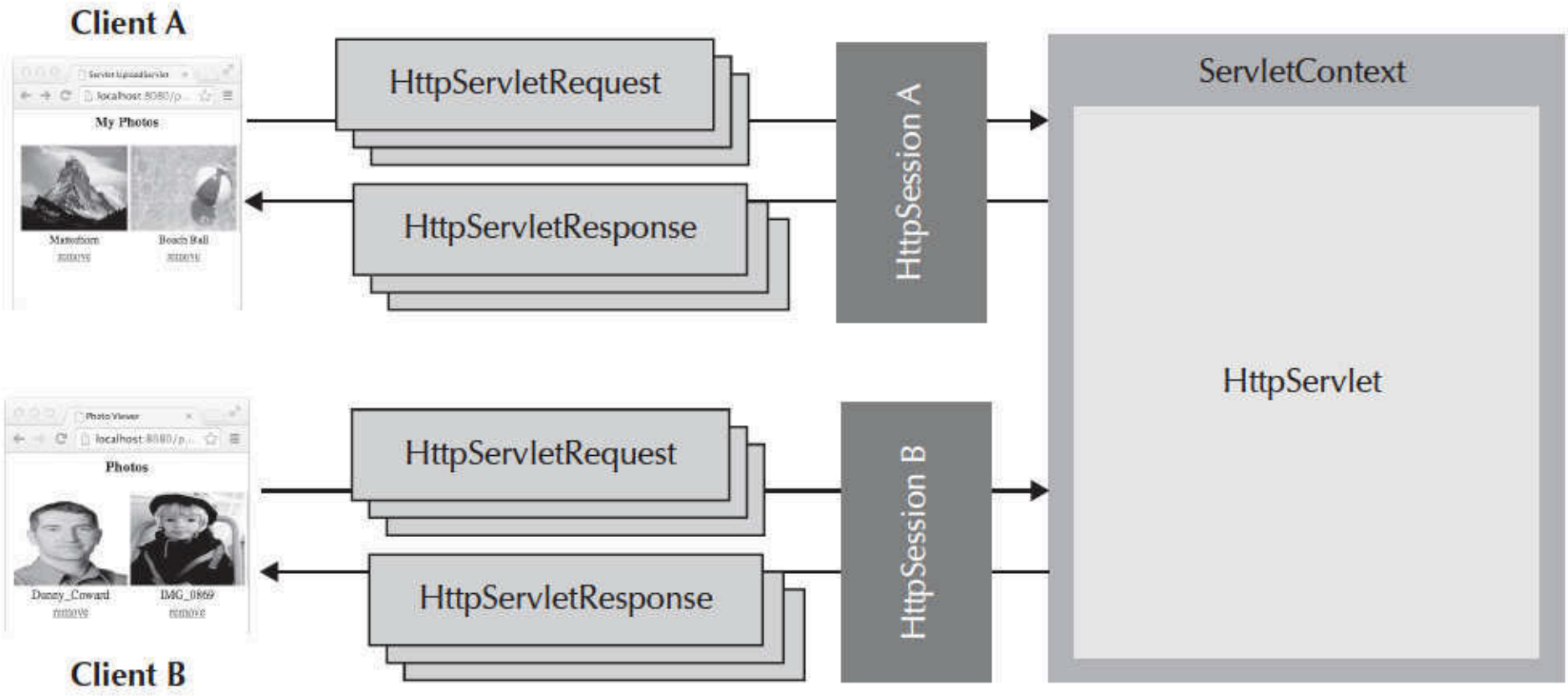
Exercício

- Repita o exercício anterior informando também o número total de vezes que qualquer usuário usou o serviço desde que o servidor foi iniciado
- Resultado

O somatório de 1 a 4 é 10.

Você usou o serviço 3 vezes nessa sessão.
Você usou o serviço 5 vezes nesse browser.
Esse serviço foi usado 14 vezes desde que o servidor foi iniciado.

ServletContext vs. HttpSession



Fonte: Livro Java EE 7: The Big Picture

Exercício

- Discutir inicialmente no grupo quando dados serão armazenados em cookies, sessões, aplicação ou banco de dados no contexto do trabalho final
- Apresentar para os outros grupos as situações que esse elementos serão utilizados

Redirecionamento

- É possível, em um *servlet*, acessar paginas HTML, outros servlets, JSP, etc.
- Via método **sendRedirect(String)** do objeto **HttpServletResponse**
  - Passa nova URL ao browser, que acessa novamente o servidor
  - A URL muda
- Via **RequestDispatcher** do objeto **ServletContext**.
  - forward(): Encaminha a solicitação internamente, sem que o cliente saiba
  - include(): Inclui a saída de outro Servlet e continua o processamento do Servlet atual

Redirecionamento (inclusão)

```
public class AloMundoRodapeServlet extends HttpServlet {
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
                    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML><BODY>");
    out.print("<P>Alo Mundo! </P>");
    RequestDispatcher dispatcher =
      getServletContext()
        .getRequestDispatcher("/rodape.html");
    if (dispatcher != null)
        dispatcher.include(request, response);
    out.println("</BODY></HTML>");
    out.close();
  }
}
```

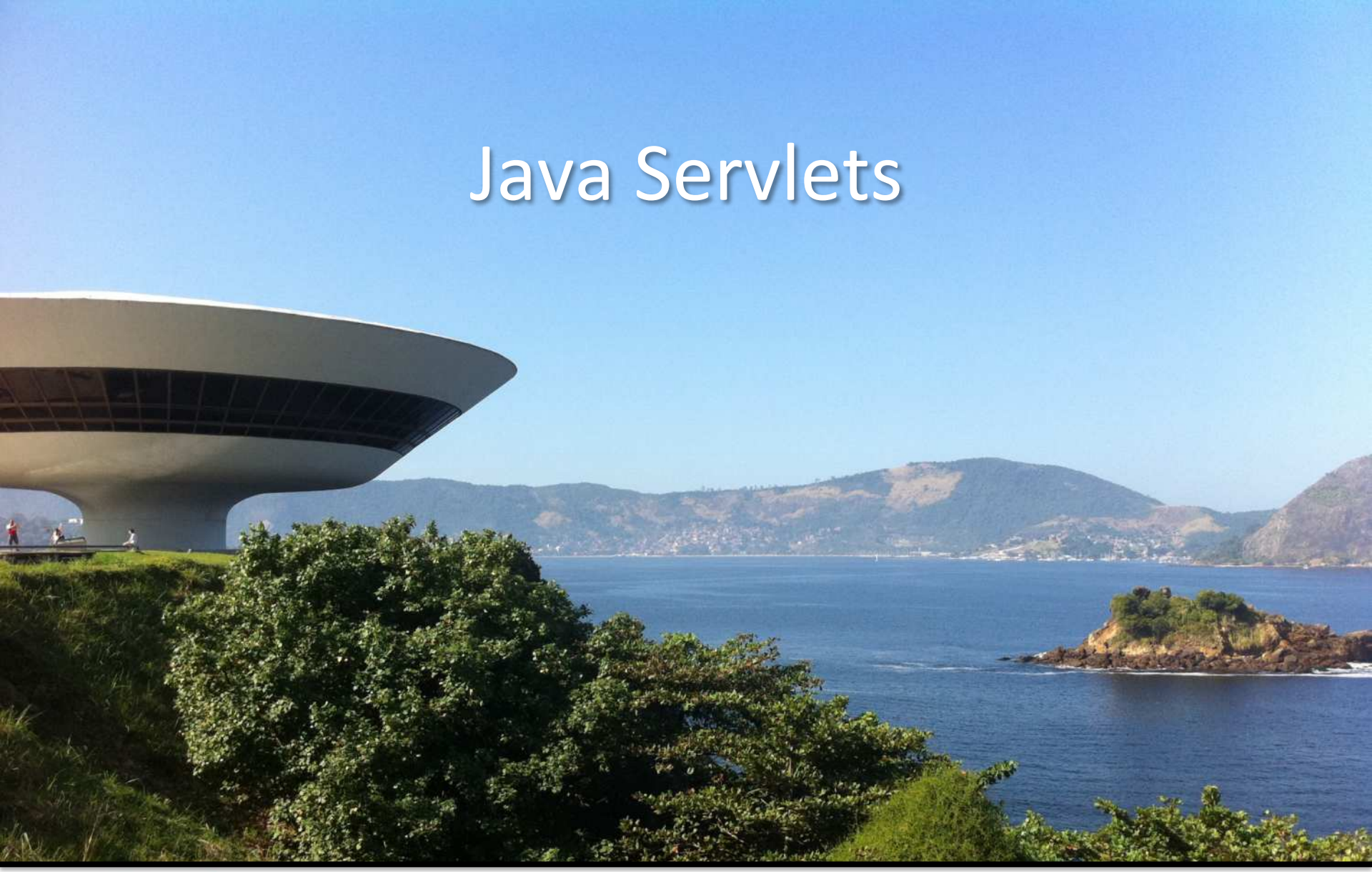
Redirecionamento (encaminhamento)

```
public class EncaminhadorServlet
        extends HttpServlet {
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response){
        RequestDispatcher encaminhador =
          request
            .getRequestDispatcher("/encaminhado.htm");
        if(encaminhador!=null)
         encaminhador.forward(request, response);
   }
}
```

Exercício

- Incluir as páginas cabecalho.html e rodape.html no servlet de somatório
- Encaminhar para uma página padrão de erro (erro.html) caso os parâmetros não tenha sido preenchidos

Java Servlets



Instituto de Computação

Leonardo Gresta Paulino Murta

leomurta@ic.uff.br