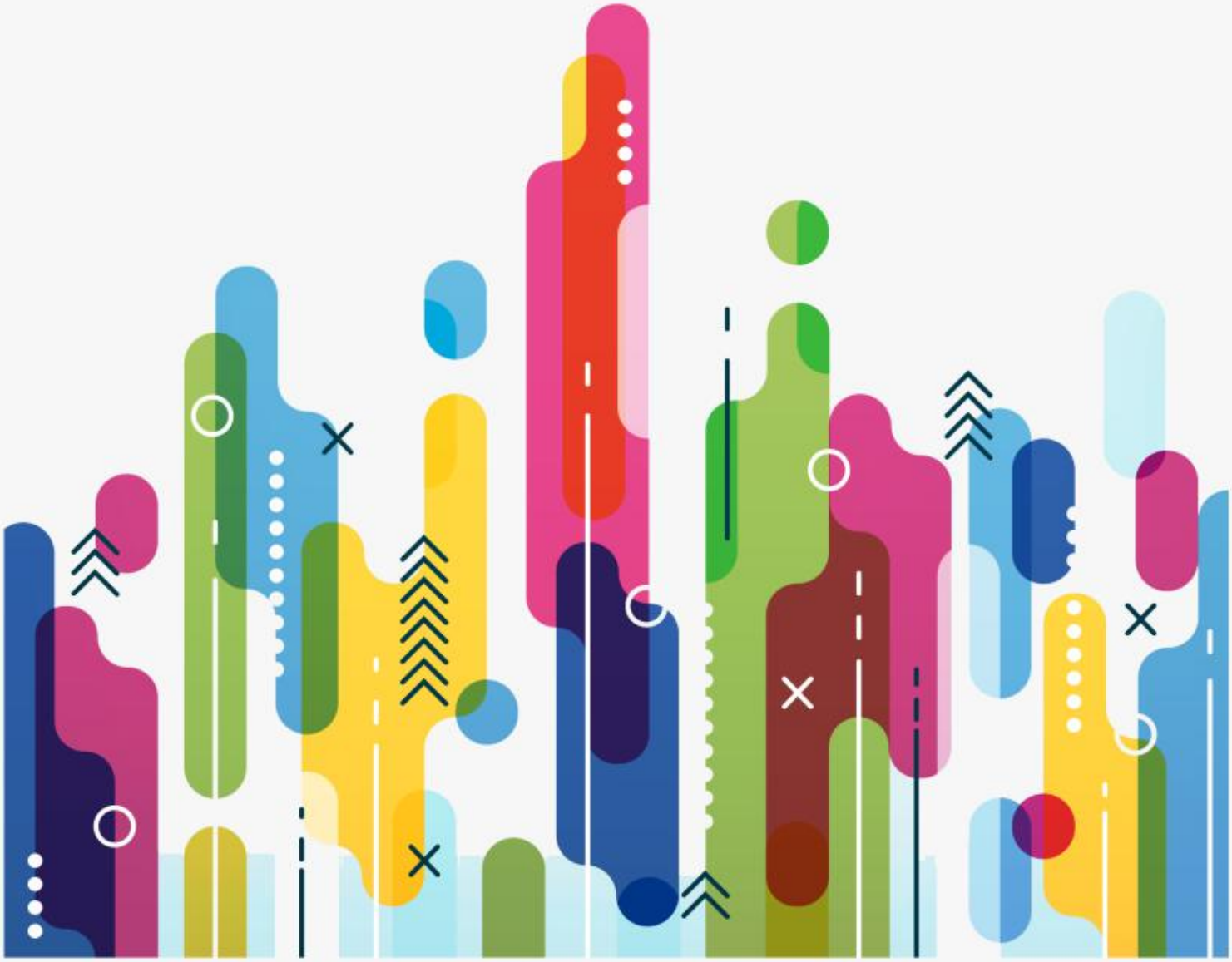
CRIANDO UMA APLICAÇÃO JAVA WEB COM SERVLETS 2



INSTITUTO FEDERAL
Paraná

Programação Web III
Professor Jefferson Chaves

CRIANDO UMA APLICAÇÃO JAVA WEB COM SERVLET

Programar **aplicações web** para atualmente é um grande desafio. Existe uma pilha de tecnologias e protocolos, conceitos e outros detalhes (HTTP, cookies, stateless, assincronicidade, etc) cujo entendimento é fundamental para a construções dessas aplicações. De forma simplificada, é possível separar as exigências de uma aplicação web em duas catagorias: equisitos funcionais e requisitos não funcionais.

Os requisitos **não funcionais**, também chamados de **requisitos não funcionais tecnológicos**, são requisitos que não tem relação com o problema em si, mas fazem parte da solução tecnológica. São exemplos de requisitos não funcionais:

- **Gerenciamento de conexões HTTP;**
- **Gerenciamento de threads;**
- **Cache de objetos;**
- **Gerenciamento da sessão web;**
- **Gerenciamento de transação com banco de dados;**
- **Entre outros.**

Tais requisitos, por serem comuns a maioria das aplicações web, passaram a **ser responsabilidade do servidor**, cabendo eventualmente, a configuração desses recursos do servidor: o programador usa esses recursos mas não é obrigado a conhecer detalhes de como estão implementados.

Além da questão tecnologica, aplicações atuais possuem **requisitos funcionais** e regras de negócio bastante complicadas. Codificar essas regras implicam em um grande trabalho.

Para dar suporte ao processo de desenvolvimento, a maioria das linguagens de programação (Python, PHP, Java, Ruby, etc) disponibilizam recursos, tecnologias e frameworks que **possibilitam abstrair a complexidade de uma aplicação web e focar apenas nas regras de negócio**.

Em aplicações web usando Java, usamos o Jakarta EE, que tem como objetivo especificar uma série de componentes para que o desenvolvedor possa concentrar-se apenas em implementar regras de negócio da aplicação, apoiado pelos recursos para gestão dos requisitos não funcionais.

O principal componente dessa arquitetura é a **API de Servlets**. Essa API tem, de forma simplificada, o objetivo de receber chamadas HTTP, processá-las e devolver uma resposta ao cliente;

Cada **Servlet** é, portanto, um objeto que recebe requisições **(request)** e produz algo **(response)**, como uma página HTML dinamicamente gerada, uma imagem ou apenas dados em formato .json. Um **Servlet** permite:

- Fazer com que uma classe seja acessível via navegador;
- Receber e converter parâmetros enviados;
- Distinguir requisições de métodos HTTP diferentes;
- Executar suas lógicas e regras de negócio;
- etc.

A **Servlet** possui um ciclo de vida bem definido além de possuir métodos específicos associados a cada método HTTP (aqui, não podemos confundir métodos HTTP, tais como post, get, put, delete, com os métodos de uma classe Java). Cada método da classe que representa a Servlet começa com o prefixo **“do”**: doGet(), doPost() e etc.

Cada método recebe como parâmetro, dois objetos. HttpServletRequest e HttpServletResponse.

O objeto **HttpServletRequest** é o responsável por representar uma requisição HTTP realizada por um cliente. O servidor recebe a requisição, "converte" a requisição em um objeto dessa classe e chama o método adequado.

Uma requisição pode conter **parâmetros** enviados pelo cliente. Os parâmetros pode ser enviados por formulários, **cujos inputs sejam nomeados**, ou por urls que contenham "query strings" tais como https://www.facebook.com/search/top/**?q=ifpr,** em que "q" é o parâmetro enviado e "ifpr" é o valor informado.

Objetos da classe **HttpServletRequest** possuem dois métodos úteis para obtenção de parâmetros:

- **String getParameter(String parameter)** → Obtém os parâmetros enviados por clientes por uma query string ou por formulário;

- **String [] getParameterValues(String parameter)** → Obtém os parâmetros enviados por clientes por meio de um formulário com campos multivalorados.
  Exemplo: um pessoa ao preencher um formulário para visitar Fernando de Noronha precisou marcar quais seus interes na ilha: mergulho, passeios, gastronomia, trilhas e eventos. A pessoa pode assinalar nenhum ou todos os itens. Nesse caso será recebida pela requisição não apenas uma String, mas sim uma array de Strings com as opções marcadas. Note que, eventualmente esse array pode estar vazio.

Se for implementado um algoritmo para resposta, o servidor obtém o objeto da classe **HttpServletResponse**, traduz para uma reposta HTTP e devolve ao cliente.

Outro recurso importante do Servlet são os recuros de "**delegação de encaminhamento**" e "**redicionamento**".

O **redirecionamento** é um tipo de resposta, em que o servlet pede para que o cliente cuide de redirecionar a página.

Já a **delegação de encaminhamento** ocorre completamente no lado do servidor e o resultado da ação de encaminhamento é transparente para o cliente.

RequestDispatcher	**SendRedirect ou Status code 3XX**
Encaminha somente para outra servlet ou para um jsp (mesmo domínio);	Encaminha para um URL (outros domínios);
Permite anexar atributos na requisição;	Atributos ou dados são inseridos por **query strings;**
Arquivo requisitado e o destino usarão a mesma requisição.	Arquivo requisitado e o destino NÃO usarão a mesma requisição.

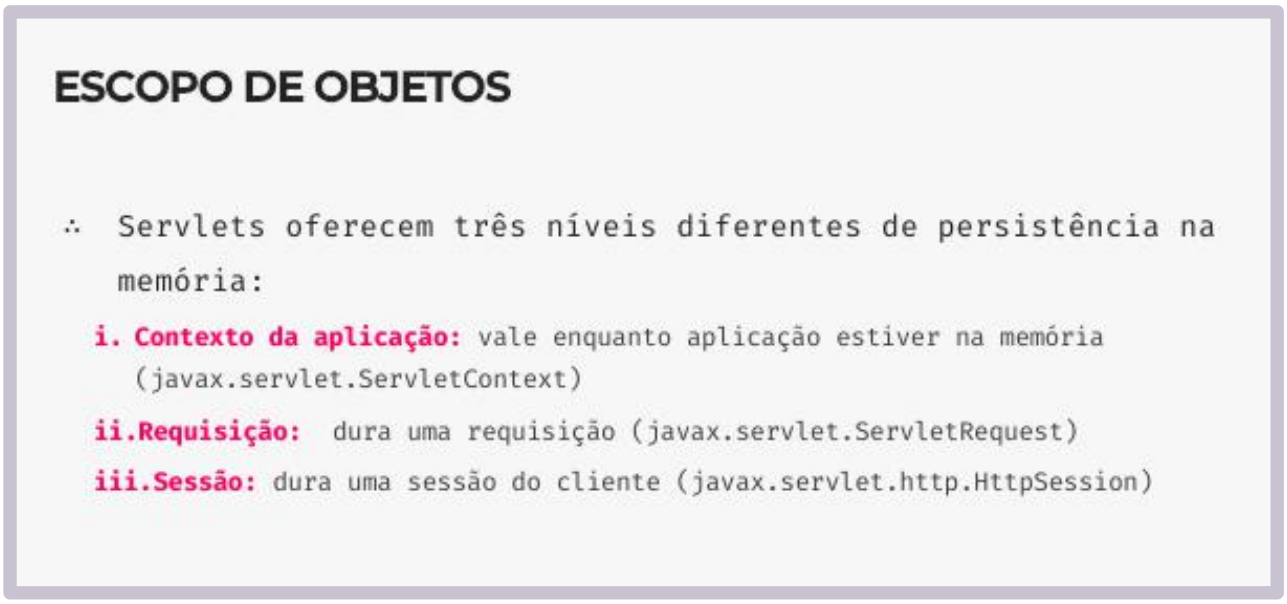
veja mais detalhes aqui.

Outro conceito importante é o chamado Escopo de Aplicação. O escopo está relacionado a acessibilidade de informações entre os componentes da aplicação.

Em aplicações web usando Java, quando se deseja que uma informação esteja disponível para todos os clientes de uma

aplicação, essa informação deve ser adicionada ao escopo chamado "Contexto da Aplicação".

Caso uma informação esteja relacionada a um único cliente, e não possa estar visível a outros clientes, então essa informação deve ser inserida no chamado "escopo da requisição".



Exemplo: inserir informações no contexto da aplicação:

```
//Obter o contexto da aplicação
ServletContext contexto = getServletContext();

//adicionar atributos ao contexto da aplicação
contexto.setAttribute("chave", "valor");

//obter atributos
String atributo = (String) contexto.getAttribute("chave");

//remover atributos
contexto.removeAttribute("chave");
```

Atividades

- A atividade deve estar em um repositório GIT;
- Apenas o link do repositório deve ser entregue como resposta para atividade

1. Implemente uma aplicação web que contenha um formulário para concorrer a uma vaga de emprego no Canadá. O formulário deve ter os seguintes campos:

   a) Nome;
   b) Data Nascimento;
   c) Idioma nativo com as opções: inglês, espanhol e português (apenas um idioma pode ser escolhido - utilize um input do tipo select ou radio buttons);
   d) Seleção de habilidades técnicas com as opções: Java, JavaScript, HTML e CSS (deve ser usado um checkbox - mais de uma opção pode ser marcada);

   Programe uma Servlet que receba os dados desse formulário e mostre as informações devidamente formatadas para o usuário. Campos em branco ou não preenchidos não devem ser aceitos.

2. Implemente uma aplicação web que contenha os seguintes arquivos:

a) index.jsp: deve apresentar um formulário de login, com os campos usuário e senha;

b) LoginServlet

i. Deve verificar se um login tem como o usuario==admin e senha==admin;
ii. Se estiver correto, armazene um atributo no contexto da aplicação indicando que está logado;
iii. Caso contrário, redirecione o usuário para o arquivo index.jsp com uma mensagem de erro;

c) OutraServlet e pagina.jsp

i. Verificar se o usuário está logado;
ii. Se estiver continua;
iii. Caso contrário, redirecione ou dispache o usuário para o arquivo index.jsp com uma mensagem de erro;

d) LogoutServlet

i. Se houver atributos do usuário no contexto da aplicação, eles devem ser removidos.