ELE2 – LPG3

Eletiva 2 – Linguagem de Programação 3 (Java)
Prof. Davi Reis

Aula 05 – Aplicações WEB - Servlets

JavaEE (Java Enterprise Edition)

- Conhecido no passado como J2EE
- Especificação para construção de servidores WEB de aplicações Java
- API:
  - Servlets, são utilizados para o desenvolvimento de aplicações Web com conteúdo dinâmico.
    - Contém uma API que abstrai e disponibiliza os recursos do servidor Web de maneira simplificada para o programador;
    - Classe Java que responde solicitações HTTP e que aceita codificação HTML em seu conteúdo.

JavaEE (Java Enterprise Edition)

- API:
  - JSP (Java Server Pages), uma especialização do servlet que permite que conteúdo dinâmico seja facilmente desenvolvido.
    - Basicamente é uma página HTML onde podem ser inseridos códigos JAVA.

JavaEE (Java Enterprise Edition)

- API:
  - EJBs (Enterprise Java Beans), utilizados no desenvolvimento de componentes de *software*.
    - Permitem que o programador se concentre nas necessidades do negócio do cliente, enquanto questões de infra-estrutura, segurança, disponibilidade e escalabilidade são responsabilidade do servidor de aplicações.

Nota: não confundir com JavaBeans, que são objetos mutáveis usados apenas para transportar valores entre diferentes camadas da aplicação (uma espécie de DTO).

Servidores JavaEE

- No mercado, existem diversos servidores Java, muitos gratuitos e "*open source*".
- Os mais conhecidos são:
  - Tomcat, *container* Java do Apache
  - GlassFish, da Sun
  - WebSphere, da IBM
  - WebLogic, da Oracle
  - Jetty, da Eclipse
  - JBoss
  - WildFly, da RedHat
  - ...entre outros

Servlet - Definição

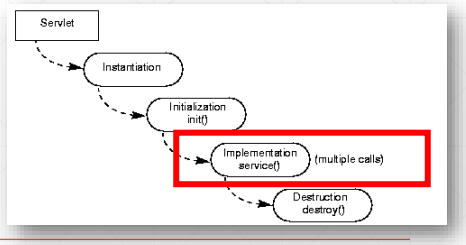
- É uma classe JAVA
  - Serve para estender a capacidade de um servidor de aplicações a uma classe comum
  - Oferece mecanismos para tratamento de requisição e resposta
- Comumente utilizada para programar uma resposta a uma requisição HTTP

API

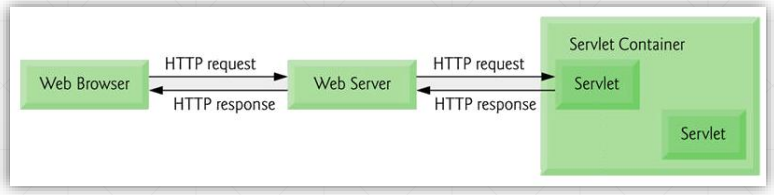
- Package javax.servlet
- Package javax.servlet.http
- Suporte para:
  - Gerenciamento de **ciclo de vida**
  - Acesso ao contexto
  - Utilidades
  - Classes de suporte específico ao HTTP

Ciclo de Vida

- Servlets são **executados no servidor**
- O servidor de aplicações é responsável por inicializar, invocar e destruir a instância do Servlet
- Os métodos herdados pela classe são:
  - init()
  - service()
  - destroy()



Ciclo de Requisição



Funcionamento padrão

- Numa arquitetura MVC, por exemplo, Servlets geralmente se encaixam como *controllers* (em aplicações Java web)
- Nessa linha, os Servlets:
  - são frequentemente usados como *controllers* (na camada de controle) do MVC.
  - recebem solicitações HTTP dos clientes, atuando como ponto de entrada para a lógica de controle da aplicação.
  - processam solicitações, interagem com a camada *Model* para obter ou atualizar dados e decidem qual *View* deve ser apresentada.
  - após o processamento, os Servlets frequentemente encaminham a solicitação para algum elemento/tecnologia da camada de apresentação (*View*) para gerar a resposta final.

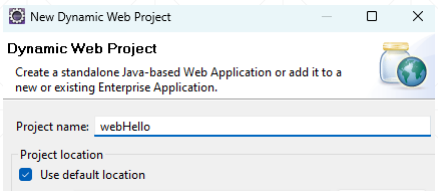
Exemplos de uso

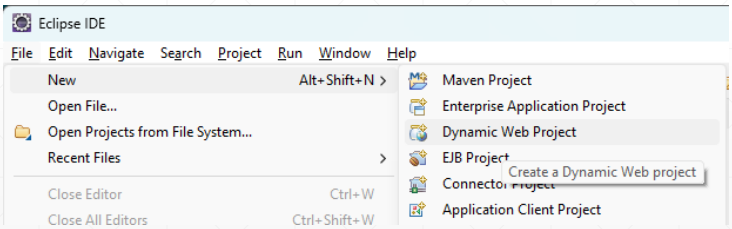
- Os Servlets são muito versáteis em Java, podendo ser empregados para:
  - Model-View-Controller (MVC): como Controller, pode atuar entre a Model e a View, recebendo solicitações do cliente, executando a lógica de controle e atualizando a Model e/ou a View.
  - API RESTful: pode ser usados na criação de serviços web RESTful (popular abordagem para construção de APIs em arquiteturas web), podendo processar solicitações HTTP (GET, POST, PUT, DELETE, etc.) e fornecer respostas em formatos como JSON ou XML.
  - Integração com frameworks MVC: como ocorre no próprio Spring MVC, pode receber requisições e delegar o processamento adicional a componentes específicos do framework.
  - Segurança de aplicações: Servlets são parte integrante de mecanismos de segurança em Java EE, podendo ser configurados para autenticação, autorização e demais práticas de segurança.
  - Aplicações diversas: processamento de formulários do usuário, controle de sessão do usuário, rastreamento do estado do usuário entre requisições, processamento de arquivos etc.

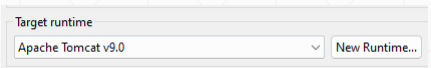
Criando um Servlet no...

Criando um Servlet

- No Eclipse, crie um novo projeto (File ➔ New)
- Escolha a categoria “Dynamic Web Project”
- Chame o projeto de “webHello”
- Escolha qualquer servidor (“*Target runtime*”) já previamente configurado e finalize o wizard.

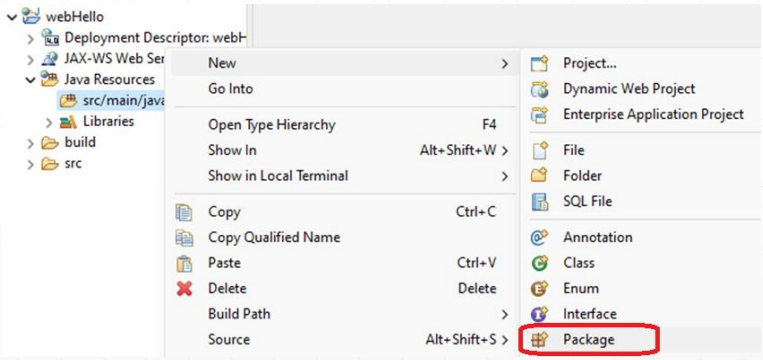






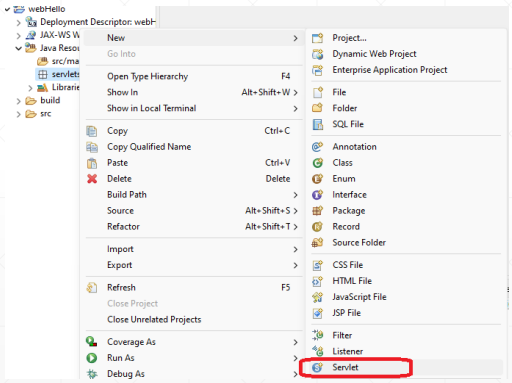
Criando o Pacote

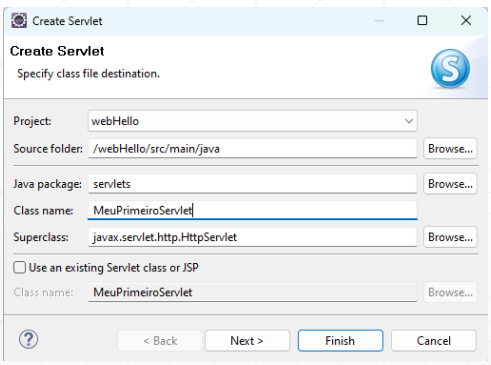
- Em “Java Resources”, crie um pacote (package) chamado “servlets”



Criando o Servlet

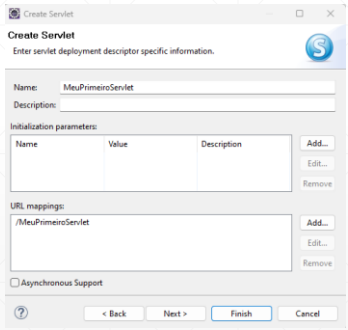
- Crie um servlet chamado “MeuPrimeiroServlet” e clique em Next

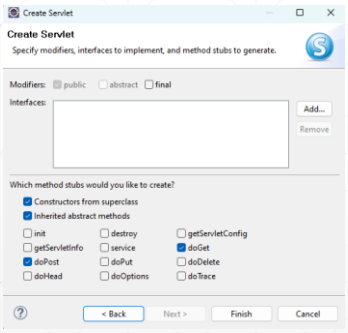




Criando o Servlet

- Na tela seguinte, mantenha o nome lógico e o URL mappings como está e avance.
- Por fim, confira se os métodos gerados estão como você deseja e finalize.





Analisando o modelo

```java
package servlets;

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

/**
 * Servlet implementation class MeuPrimeiroServlet
 */
@WebServlet("/MeuPrimeiroServlet")
public class MeuPrimeiroServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public MeuPrimeiroServlet() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        response.getWriter().append("Served at: ").append(request.getContextPath());
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }
}
```

Pode-se criar uma saída HTML no método doGet ()

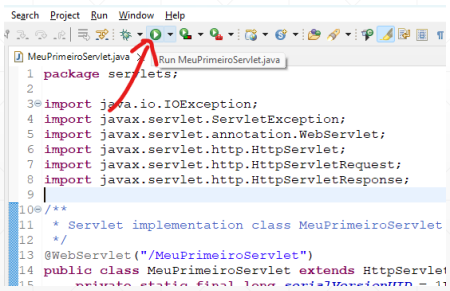
```
/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // response.getWriter().append("Served at: ").append(request.getContextPath());

    // configura tipo de retorno do conteúdo que está sendo devolvido, bem
    // como o padrão de caracteres (para permitir acentuação)
    response.setContentType("text/html; charset=UTF-8");

    try(PrintWriter out = response.getWriter()){
        out.println("<!DOCTYPE HTML>");
        out.println("<html>");
        out.println("<head><title>Meu primeiro servlet</title></head>");
        out.println("<body>");
        out.println("<h2>Meu primeiro servlet</h2>");
        out.println("<h3>Sendo executado em: " + request.getContextPath() + "</h3>");
        out.println("<h3>Fatec-RL - ADS </h3>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

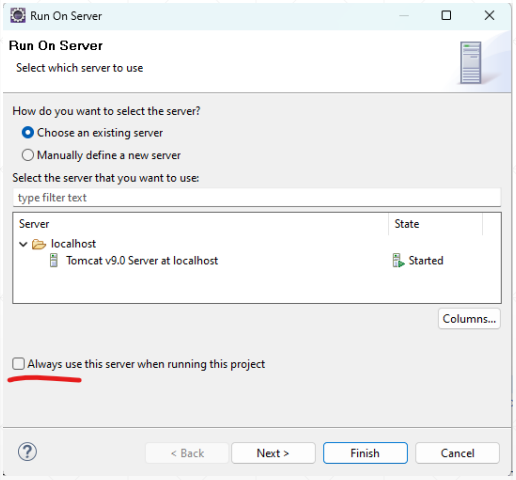

Executando o Servlet

- Pode ser feito de 04 formas:
  - Executando o projeto (conforme a imagem abaixo)
  - Pelas teclas de atalho <CTRL> + <F11>
  - Pelo menu “Run”, opção “Run”
  - Executando diretamente o arquivo no menu de contexto (no “Project Explorer”)



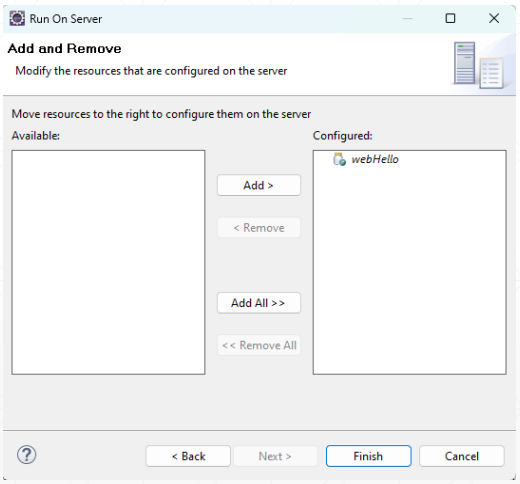
Executando o Servlet

- Em seguida, deve-se selecionar o servidor de aplicação correspondente e avançar (se não quiser que essa pergunta ocorra sempre, pode-se selecionar o checkbox na parte debaixo da tela)



Executando o Servlet

- Por fim, se o projeto estiver na lista de disponíveis, movê-lo/adicioná-lo para a lista de projetos já configurados.



Contador de requisições

- Podemos utilizar um atributo de instância do Servlet para controlar a quantidade de visitas à página

```
19  public class HomeServlet extends HttpServlet {
20
21      int contador = 0;
22
23      /**...*/
30      protected void processRequest(HttpServletRequest request, HttpServletResponse response)
31      throws ServletException, IOException {
32          response.setContentType("text/html;charset=UTF-8");
33          PrintWriter out = response.getWriter();
34          try {
35              out.println("<html>");
36              out.println("<head>");
37              out.println("<title>Meu primeiro servlet</title>");
38              out.println("</head>");
39              out.println("<body>");
40              out.println("<h1>Quantidade de requisições (visitas): " + contador++ + "</h1>");
41              out.println("</body>");
42              out.println("</html>");
43          } finally {
44              out.close();
45          }
46      }
```


Parâmetros

- Uma requisição web pode conter parâmetros oriundos de HTML forms ou da própria URL
- Podemos capturar esses parâmetros através do objeto request, parâmetro do método processRequest()

```
29      protected void processRequest(HttpServletRequest request,
30      throws ServletException, IOException {
31          response.setContentType("text/html;charset=UTF-8");
32          PrintWriter out = response.getWriter();
33          try {
34              out.println("<html>");
35              out.println("<head>");
36              out.println("<title>Meu primeiro servlet</title>"
37              out.println("</head>");
38              out.println("<body>");
39              request.getParameter
40              out.p
41              out.p
42              out.p
43          } finally
```

```
getParameter(String name)          String
getParameterMap()                  Map
getParameterNames()                Enumeration
getParameterValues(String name)    String[]
```

Capturando parâmetros da query

- São parâmetros enviados pela URL da página acrescida de '?' Com os parâmetros separados por '&'

```
29      protected void processRequest(HttpServletRequest request, HttpServletResponse response)
30      throws ServletException, IOException {
31          response.setContentType("text/html;charset=UTF-8");
32          PrintWriter out = response.getWriter();
33          try {
34              out.println("<html>");
35              out.println("<head>");
36              out.println("<title>Meu primeiro servlet</title>");
37              out.println("</head>");
38              out.println("<body>");
39              String par1 = request.getParameter("parametro1");
40              String par2 = request.getParameter("parametro2");
41              out.println("<h1>Recebido os parâmetros: " + par1 + " e " + par2 + "</h1>");
42              out.println("</body>");
43              out.println("</html>");
44          } finally {
45              out.close();
46          }
47      }
```


Redirecionando o tratamento de uma requisição

- Para dar continuidade a uma requisição do cliente, pode ser necessário redirecionar o tratamento para outro recurso ou URL. Isso pode ser feito, basicamente, com o uso de um dos comandos abaixo:
  - **request.getRequestDispatcher**: usado para obter um objeto RequestDispatcher, que é uma interface para encaminhar a solicitação do cliente para outro recurso no servidor, como um Servlet, JSP, outro recurso Java ou até mesmo um recurso estático.
    - Ex.: RequestDispatcher dispatcher = request.getRequestDispatcher("/OutroServlet");
      dispatcher.forward(request, response);
  - **response.sendRedirect**: usado para redirecionar o navegador do cliente para outra página ou recurso, instruindo o navegador a fazer uma nova solicitação para a URL especificada.
    - Ex.: response.sendRedirect("https://fatecrl.edu.br");

Redirecionando o tratamento de uma requisição

- Se necessário enviar um valor ou objeto adicional para ser tratado com o restante do processamento por um próximo recurso, basta criar um novo atributo:

```
request.setAttribute("objetoCliente", cliente);
RequestDispatcher dispatcher = request.getRequestDispatcher("ServletSaldos");
dispatcher.forward(request, response);
```

  - Para recuperar esse elemento, basta usar o método “getAttribute” e o nome do objeto (no caso, “objeto cliente”).
  - Nesse exemplo, se houver outra requisição (outro “request”) feito pelo usuário, o objeto acima é descartado.

- Para enviar criar um elemento que permaneça disponível mesmo após a ocorrência de outras requisições do usuário, deve-se recorrer a criação de elementos na “sessão” entre o cliente e o servidor:

```
request.getSession().setAttribute("objetoCliente", cliente);
```


Obrigado!