**O que Aprendemos:**

* Apache Tomcat ou apenas Tomcat é um servidor web em Java
* Tomcat entende o protocolo HTTP e roda por padrão no http://localhost:8080
* O projeto Java faz parte da URL, no nosso caso: http://localhost:8080/gerenciador
* Uma aplicação web Java pode ter páginas HTML
* Uma servlet é um objeto Java que podemos chamar a partir de uma requisição HTTP
* Para mapear a URL para uma servlet usamos a anotação @WebServlet
* Uma servlet deve estender a classe HttpServlet e sobrescrever um determinado método (por exemplo service)
* escrever uma servlet que atende apenas GET ou POST
* ler parâmetros da requisição dentro da servlet
* criar um formulário HTML usando as tags form e input
* enviando os dados pelo formulário através do POST
* diferença entre GET e POST
* JSP significa Java Server Pages
* JSP é uma página automaticamente processada pelo Tomcat
* Para gerar HTML dinamicamente no JSP usamos Scriptlets
* Um scriptlet <% %> é um código Java dentro do HTML
* Um scriptlet só funciona em uma página JSP
* Usamos o RequestDispatcher para chamar um JSP a partir da servlet
* Obtemos o RequestDispatcher a partir do HttpServletRequest
* Usamos a requisição para colocar ou pegar um atributo (setAttribute(.., ..) ou getAttribute(..))
* Expression Language (EL) é uma linguagem simples e limitada para imprimir o resultado de uma expressão
* EL usa a sintaxe de ${ .. }
* JSTL é a biblioteca padrão de tags
* JSTL não vem com Tomcat e precisamos copiar um JAR
* JSTL define 4 taglibs, as mais importantes são core e fmt
* a taglib core serve para controle de fluxo, fmt para formatação
* é preciso importar as taglib, core e fmt separadamente
* Expression Language (EL) é uma linguagem simples e limitada para imprimir o resultado de uma expressão
* EL usa a sintaxe de ${ .. }
* JSTL é a biblioteca padrão de tags
* JSTL não vem com Tomcat e precisamos copiar um JAR
* JSTL define 4 taglibs, as mais importantes são core e fmt
* a taglib core serve para controle de fluxo, fmt para formatação
* é preciso importar as taglib, core e fmt separadamente
* o problema de reenviar uma requisição
* a diferença entre redirecionamento pelo cliente e servidor
* para redirecionar pelo navegador usamos o método response.sendRedirect("endereço")
* o código de resposta para redirecionamento HTTP é 30X (301 ou 302)
* Como implementar a remoção de empresas
* Como implementar a edição de empresas
* Qual atributo correto devemos utilizar para identificação dos elementos (id)
* Como definir um input escondido
* O que significa CRUD
* A importância do web.xml
* O que é inversão de controle
* Como o Tomcat trabalha por debaixo dos panos
* Como inicializar o Tomcat na linha de comando
* Como gerar o WAR (Web ARchive)
* Como fazemos o deploy da nossa aplicação

Parte 2:

* MVC significa Model-View-Controller
* MVC divide a aplicação em 3 camadas lógicas
* Cada camada tem a sua responsabilidade
* O controlador central e as ações fazem parte da camada Controller, que define o fluxo da aplicação
* Os JSPs fazem parte da camada View, que define a interface
* As classes do modelo fazem parte da camada Model, que encapsula as regras de negócio
* MVC facilita a manutenção e evolução da aplicação
* Os JSPs devem ficar "escondidos" na pasta WEB-INF, pois dependem da ação
* A representar o usuário através de uma classe Usuario
* A criar um formulário de login
* A criar a ação para chamar o formulário
* A criar a ação verificar o login e a senha
* Por padrão, o navegador não envia nenhuma identificação sobre o usuário
* Quando o Tomcat recebe uma nova requisição (sem identificação), gerará um ID
* O ID fica salvo no cookie de nome JSessionID
* O ID é um hash (número aleatório)
* O cookie é anexado à resposta HTTP
* O navegador reenvia o cookie automaticamente nas próximas requisições
* O Tomcat gera, além do ID, um objeto chamado HttpSession
* A vida do objeto HttpSession fica atrelado ao ID
* Para ter acesso à HttpSession, basta chamar request.getSession()
* Usamos a HttpSession para guardar dados sobre o usuário (login, permissões, carrinho de compra)
* A HttpSession tem um ciclo de vida e será automaticamente invalidada
* Um Filter e Servlet são bem parecidos
* Comparado com Servlet, o Filter tem o poder de parar o fluxo
* Para escrever um filtro, devemos implementar a interface javax.servlet.Filter
* Para mapear o filtro, usamos a anotação @WebFilter ou o web.xml
* Vários filtros podem funcionar numa cadeia (um chama o próximo, mas todos são independentes)
* Para definir a ordem de execução, devemos mapear os filtros no web.xml
* Um filtro recebe como parâmetro, do método doFilter, um ServletRequest e um ServletResponse
* Ambos, ServletRequest e ServletResponse, são interfaces mais genéricas do que HttpServletRequeest e HttpServletResponse
* Para chamar o próximo filtro na cadeia, usamos o objeto FilterChain