Filtros

- Onde implementar funcionalidades não relacionadas com a lógica de negócios? Exemplos de funcionalidades deste tipo: logging no sistema, tratamento de erro, autorização, etc;
- **Filtros:** classes que são executadas antes e/ou depois de uma requisição;
- **Exemplo.** Criamos um pacote `filtro` no projeto e dentro dele criamos uma classe `ExemploDeFiltro` que implementa a classe `javax.servlet.Filter`:

```
@WebFilter("/meufiltro")
public class ExemploDeFiltro implements Filter {

	@Override
	public void destroy() {

	}

	@Override
	public void doFilter(ServletRequest request, ServletResponse response,
		FilterChain chain) throws IOException, ServletException {
		// colocar código a ser executado antes da requisição
		chain.doFilter(request, response);
		// colocar código a ser executado depois da requisição
	}

	@Override
	public void init(FilterConfig arg0) throws ServletException {

	}
}
```

Filter tem três métodos que devem ser implementados: `init()`, `destroy()` e `doFilter()`. É no método `doFilter()` que vamos colocar o código que queremos executar antes e/ou depois da requisição ser atendida pela nossa aplicação. Você pode deixar os outros dois métodos vazios, mas a estrutura deles deve estar no código. A anotação `@WebFilter("/meufiltro")` indica a url de requisição que deverá passar pelo filtro. A opção `urlPatterns = {"/*"}` indica que o filtro deve ser chamado antes de todas as requisições da aplicação. **Exemplos:**

```
@WebFilter("/mvc") // apenas requisições para "mvc" passarão por esse filtro
@WebFilter(urlPatterns = {"/*"}) // todas as requisições passarão por esse filtro
@WebFilter(urlPatterns = {"/mvc", "/mvc2"}) // apenas requisições para "mvc" e "mvc2" passarão pelo filtro
```

- **Exercício 1**. Baixe o projeto "aula5" no site da disciplina e o modifique, criando um filtro para medir o tempo de execução da aplicação:

```
public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {

      long tempoInicial = System.currentTimeMillis();

      chain.doFilter(request, response);

      long tempoFinal = System.currentTimeMillis();
      String uri = ((HttpServletRequest) request).getRequestURI();
      System.out.println("Tempo da requisição em " + uri + " foi de: "
                  + (tempoFinal - tempoInicial));
}
```

Não esqueça de acrescentar o `import`:

```
import javax.servlet.http.HttpServletRequest;
```

E de acrescentar a anotação: `@WebFilter("/mvc")`

Organizando as Conexões com o Banco de Dados

- **Exercício 2.** No mesmo projeto, crie um filtro chamado "`FiltroConexao`" que será responsável por gerenciar as conexões do banco de dados:

```
public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
      try {
            Connection conn = new ConnectionFactory().getConnection();
            // fornece uma conexão para a requisição
            request.setAttribute("conexao", conn);

            chain.doFilter(request, response);

            // fecha a conexão cedida para a requisição
            conn.close();
      } catch (SQLException e){
            throw new ServletException("Erro na conexão com o banco",e);
      }
}
```

Modifique o construtor de `AlunoDAO` como segue:

```
public AlunoDAO(Connection conn) {
      this.connection = conn;
}
```

Modifique as classes com as operações de inserção, alteração e remoção para pegar a conexão fornecida na própria requisição pelo filtro da seguinte forma:

```
Connection conn = (Connection) request.getAttribute("conexao");
AlunoDAO dao = new AlunoDAO(conn);
```