Controle - reduzindo o número de servlets ainda mais

- **Problema de administração:** uma servlet para cada lógica diferente de cada modelo é inviável de manter em projetos reais;
- **Opção que tentamos:** reduzir para uma servlet para cada modelo agrupando todas as operações do modelo dentro de uma mesma servlet.
- **Ainda melhor:** reduzir para **UMA** servlet para **TODA** a aplicação! A princípio, poderíamos implementar uma única servlet para a aplicação enviando todas as lógicas e suas operações para a mesma servlet e lá dentro indicando as ações apropriadas com o uso de um "if". **Desvantagem**: a servlet se tornaria enorme.
- Vamos usar polimorfismo. Vamos modificar o projeto `aula5` e torná-lo completamente MVC.
- **Passo 1.** No pacote `"br.edu.ufabc.prograd.servlet"`, modifique a classe `ControllerServlet` da seguinte forma:

```
	@WebServlet("/mvc")
	protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		String objetivo = request.getParameter("objetivo");
		String nomeDaClasse = "br.edu.ufabc.prograd.logica." + objetivo;

		try {
			Class classe = Class.forName(nomeDaClasse);
			Logica logica = (Logica) classe.newInstance();
			logica.executa(request, response);
		} catch (Exception e) {
			throw new ServletException("Erro na lógica de negócios",e);
		}
	}
```

- **Passo 2.** Crie um pacote `"br.edu.ufabc.prograd.logica"` e, dentro dele, crie uma interface chamada `Logica` com o seguinte conteúdo:

```
	public interface Logica {
		void executa(HttpServletRequest request, HttpServletResponse response)
			throws Exception;
	}
```

- **Passo 4.** Ainda no pacote `"br.edu.ufabc.prograd.logica"`, crie uma classe `InsereAluno` para inserir os alunos no banco de dados:

```
public class InsereAluno implements Logica {
	@Override
	public void executa(HttpServletRequest request,HttpServletResponse response)
			throws Exception {
		// buscando parâmetros do request
		String nome = request.getParameter("nome");
		String email = request.getParameter("email");
		String endereco = request.getParameter("endereco");
```

```
            // instancia objeto Aluno e insere no BD
            Aluno aluno = new Aluno();
            aluno.setNome(nome);
            aluno.setEmail(email);
            aluno.setEndereco(endereco);
            AlunoDAO dao = new AlunoDAO();
            dao.insere(aluno);

            // reencaminha a informação para um jsp
            response.sendRedirect("/aula5/listataglib2.jsp");
        }
}
```

- **Passo 5.** No arquivo `.jsp`, atenção aos detalhes do formulário cuja ação deve ser igual ao `@WebServlet` mapeado no servlet:

```
<form name="cadastro" id="formulario" action="mvc" method="POST">
```

- **Passo 6.** Ainda no arquivo `.jsp`, atenção ao `input hidden`. O valor passado à servlet através desse campo deve ser exatamente igual ao nome da classe que será instanciada dentro da servlet:

```
<input type="submit" value="Inserir" onclick="setar('InsereAluno')"/>
```

- **Exercício 1**. Crie as classes de "*AlteraAluno*" e "*RemoveAluno*" para alterar e remover um aluno, respectivamente. Coloque as três operações (inserção, alteração e remoção) junto com a listagem dos registros onde cada linha apresenta um `checkbox` para facilitar a escolha do registro (como feito na aula anterior).
- **Exercício 2.** Ao invés de implementar cada lógica para redirecionar para um arquivo jsp, faça o método `executa` retornar uma string com a página para onde redirecionar e faça a servlet pegar essa string e executar o redirecionamento.
- **Exercício 3.** As lógicas InsereAluno e AlteraAluno são muito parecidas. Tente criar uma versão que faça as duas.