Desenvolvimento web

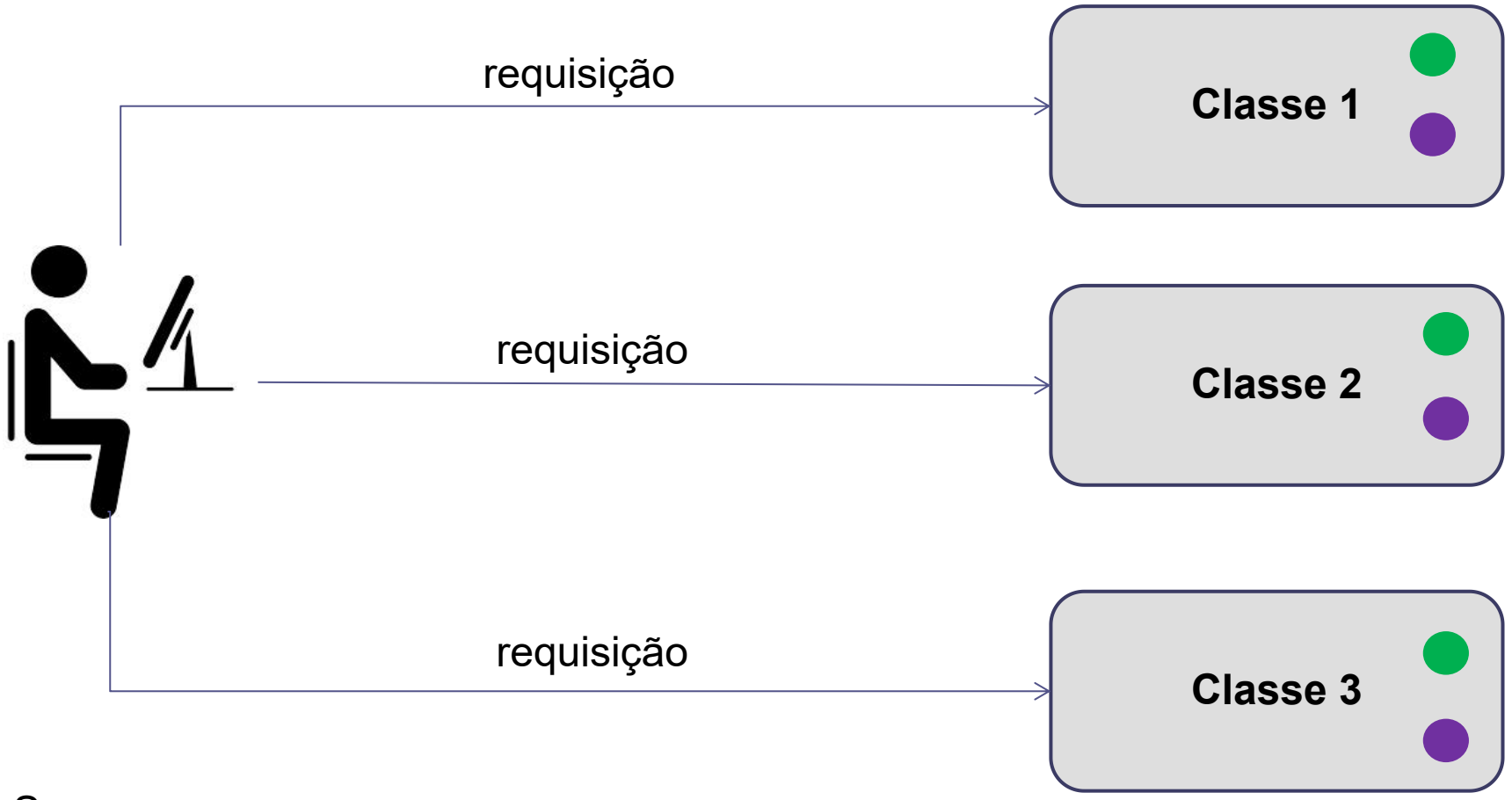
Back-end - Filtros

Profa. Dra. Luciana Zaina

E-mail: lzaina@ufscar.br

Introdução

- Existem requisitos que não são diretamente relacionados com a regra de negócio.
  - Exemplos: login, autorização de acesso, tratamento de erros.
- A primeira ideia seria colocar o código diretamente na classe que possui a lógica.
- Mas depois vemos que temos que acabar repetindo o código em várias classes.
- Ou então causamos um alto acoplamento entre a implementação e requisitos que muitas vezes não estão associados as regras de negócio.



Segurança

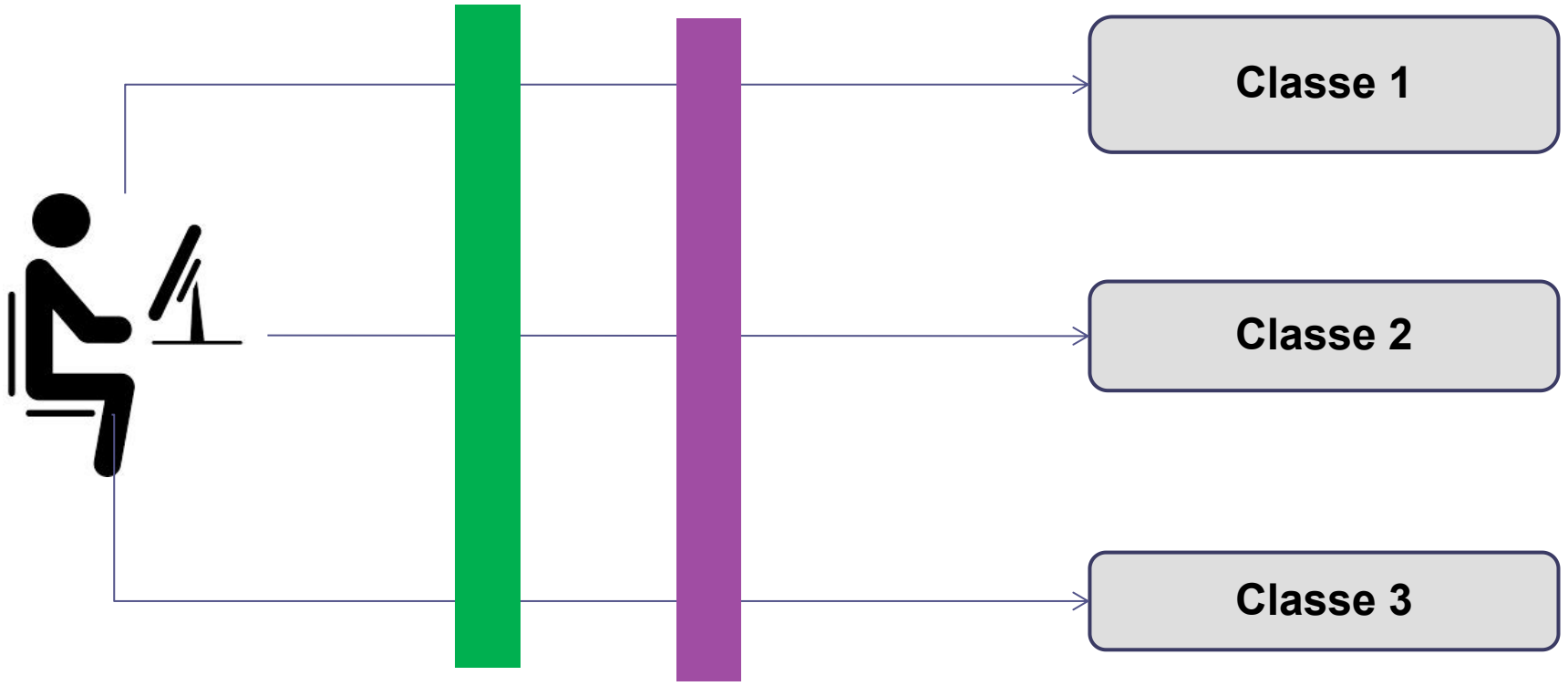
Tratamento de erros

Filtros

- São classes que permitem que executemos código **antes** da requisição e também **depois** que a resposta foi gerada.
- É um mecanismo que permite tirar o acoplamento e **isolar** comportamentos que não estão associados a lógica de negócio.
- Cada filtro encapsula apenas **uma responsabilidade**
  - Exemplo gerenciar login
- Um filtro pode **negar** a execução de um objeto que está na sequência.

Filtros

- Então é possível usar vários filtros em conjunto.



Segurança

Tratamento de erros

Design Pattern

- Os filtros que são utilizados em requisições ao servidor implementam o padrão de projeto **Chain of Responsability**.

Java Servlet

- Para cada filtro é necessário criar uma classe que implementa a interface servlet.Filter.
- E declarar o filtro no web.xml.
- Quando o container recebe uma requisição ele irá verificar o web.xml e se identificar um filtro direciona a requisição para o filtro.

web.xml

```
<filter>
   <filter-name>autentica</filter-name>
   <filter-class>controller.filtro.Autenticador</filter-class>
</filter>

<filter-mapping>
  <filter-name>autentica</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

indica as URLs ou servlet que o filtro será aplicado sempre que o servlet for chamado.

Métodos da inteface Filter

- método **init**(**FilterConfig** config):
  - inicia o ciclo de vida do filtro
  - o container invoca esse método apenas uma vez
  - é chamado assim que a aplicação web é iniciada
  - Interface FilterConfig: permite obter informações do filtro.

- método **destroy**( ):
  - permite tirar o filtro de serviço
  - chamado quando a aplicação web é finalizada.

Métodos da inteface Filter

- método **doFilter**(ServletRequest **request**, ServletResponse **response**, FilterChain **chain**):
  - é onde ocorre a filtragem
  - o container passa os objetos HttpServletRequest, HttpServletResponse, FilterChain
  - é este método que possibilita o desacoplamento do filtro com os servlets e o encadeamento de filtros
  - FilterChain será usado para encaminhar o handler para o próximo da cadeia (DP Chain of responsability).

Estrutura

```
public class Autenticar implements Filter {
        // implementação do init e destroy

        public void doFilter(ServletRequest request,
                ServletResponse response, FilterChain chain)
                        throws IOException, ServletException {
                        // todo o processamento vai aqui
        }
}
```

Função do Filtro

- Um filtro é implementado para:
  - **interceptar** vários requests semelhantes
  - executar algo
  - depois permitir que o processamento normal do request **prossiga** através das Servlets e JSPs normais.

- Qualquer código colocado **antes** da chamada **chain.doFilter** será executado na **ida**, qualquer código **depois** na **volta**.

- Exemplo: fazer um verificação de acesso antes da lógica, ou abrir um recurso (conexão ou transação) antes e na volta fechá-lo.


Vamos fazer um exemplo prático com filtros!

Back-end - Filtros

e-mail: lzaina@ufscar.br
Twitter: @lzaina
Linkedin: Luciana Zaina