Filters

- Where should we implement features that are not related to the business logic? For example: logging to the system, error handling, authorization, etc;
- **Filters:** classes that are not executed before and/or after a request;
- **Example.** We create a package `filter` in the project and inside it we create a class `ExampleOfFilter` that implements the class `javax.servlet.Filter`:

```
@WebFilter("/myfilter")
public class ExampleOfFilter implements Filter {

	@Override
	public void destroy() {

	}

	@Override
	public void doFilter(ServletRequest request, ServletResponse response,
		FilterChain chain) throws IOException, ServletException {
		// add code to be executed before the request
		chain.doFilter(request, response);
		// add code to be executed after the request
	}

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

	}
}
```

`Filter` has three methods that must be implemented: `init()`, `destroy()` and `doFilter()`. It is in the method `doFilter()` that we will add the code to be executed before and/or after a request being answered by our application. You can leave the other two methods empty, but their structure must be in the code. The annotation `@WebFilter("/myfilter")` indicates the `url` of request that must be send by the filter. The option `urlPatterns = {"/*"}` indicates that the filter must be called before all application requests. **Examples:**

```
@WebFilter("/mvc") // only requests to "mvc" will go by this filter
@WebFilter(urlPatterns = {"/*"}) // all requests will go by this filter
@WebFilter(urlPatterns = {"/mvc", "/mvc2"}) // only requests to "mvc" and "mvc2" will
go by this filter
```

Exercise 1. Download the project "class5" and modify it by creating a filter to measure the execution time of the application:

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

      long initialTime = System.currentTimeMillis();

      chain.doFilter(request, response);

      long finalTime = System.currentTimeMillis();
      String uri = ((HttpServletRequest) request).getRequestURI();
      System.out.println("Time of the request in " + uri + " was: "
                  + (finalTime - initialTime));
}
```

Don't forget to add the `import`:

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

And to add the annotation: `@WebFilter("/mvc")`

Organizing the Connections with the Database

Exercise 2. In the same project, create a filter named "FilterConnection" that will be responsible for managing the connections with the database:

```
public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
      try {
            Connection conn = new ConnectionFactory().getConnection();
            // it provides a connection to the request
            request.setAttribute("connection", conn);

            chain.doFilter(request, response);

            // it closes the connection given for the request
            conn.close();
      } catch (SQLException e){
            throw new ServletException("Database connection error",e);
      }
}
```

Modify the constructor in StudentDAO as follows:

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

Modify the classes containing the operations to insert, update, and remove so that it takes the connection provided in the request by the filter as follows:

```
Connection conn = (Connection) request.getAttribute("connection");
StudentDAO dao = new StudentDAO(conn);
```