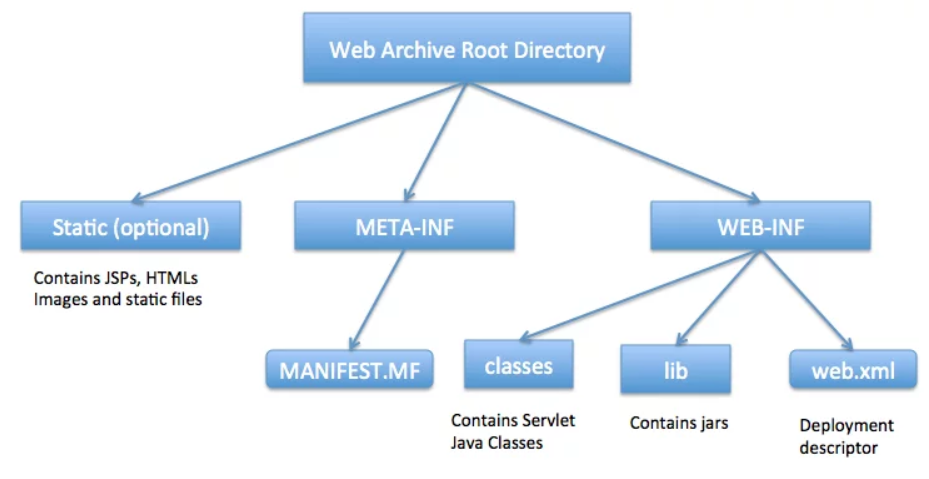
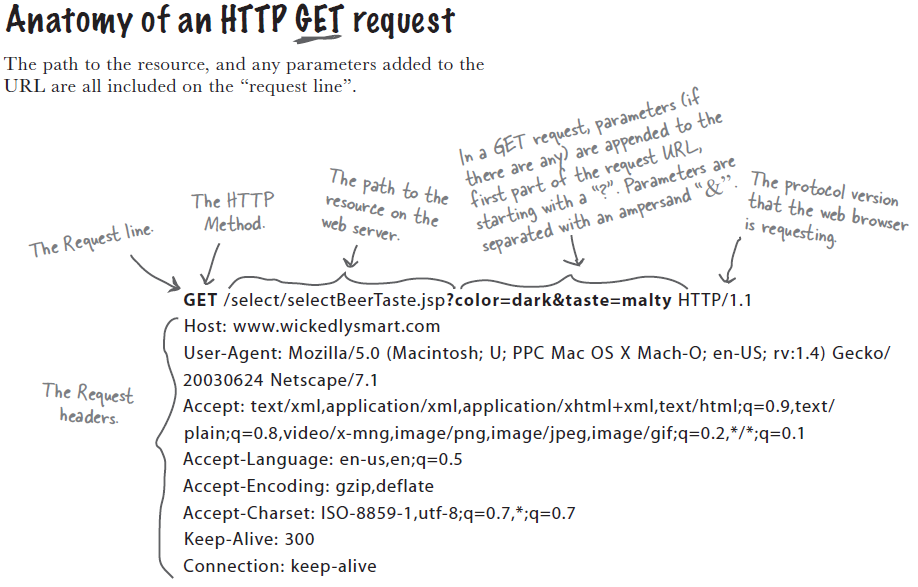
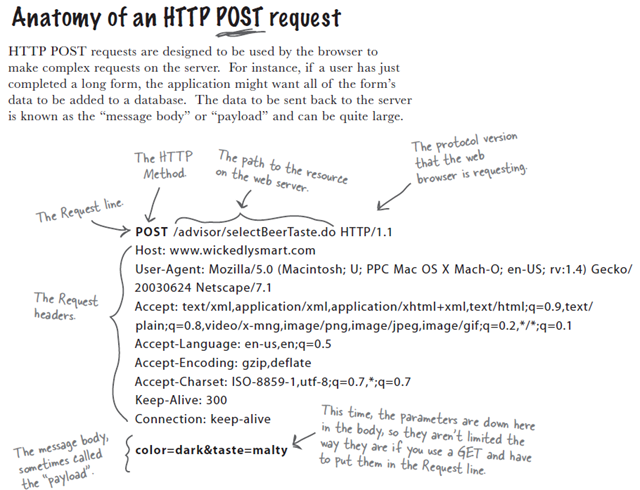
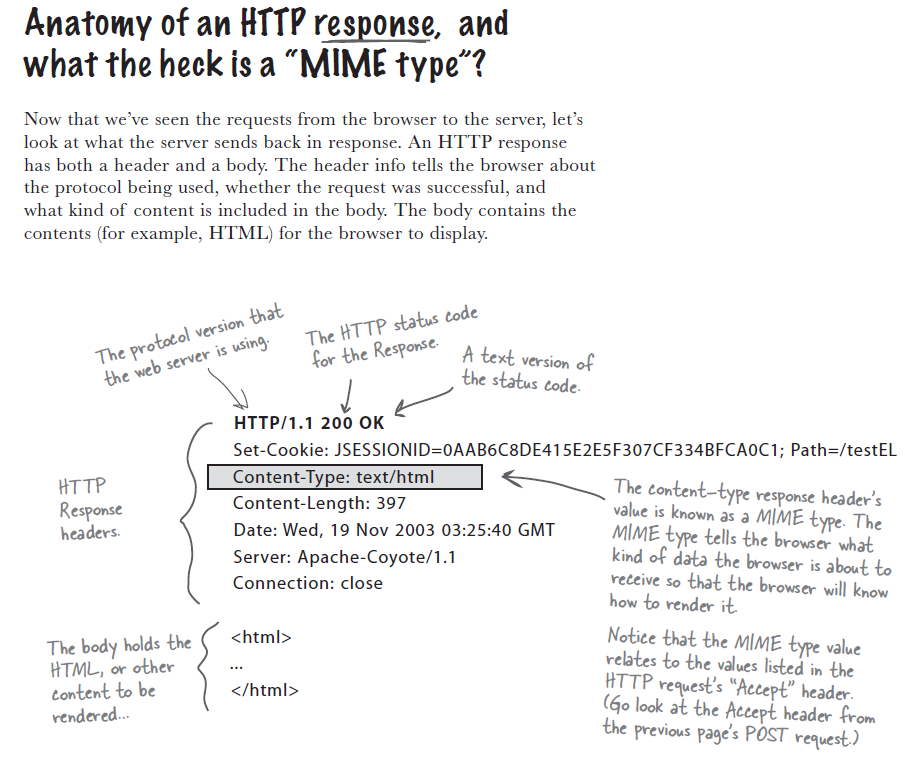
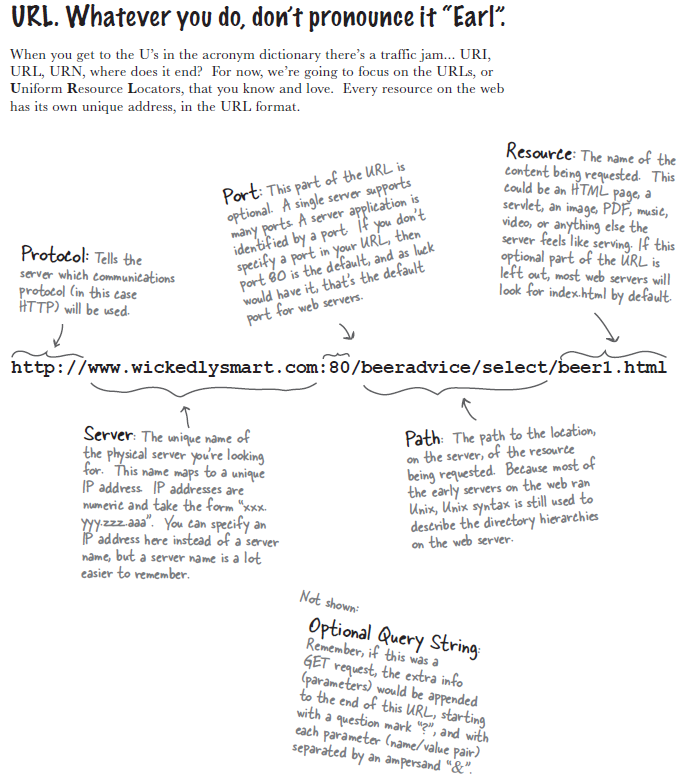
# Web app structure





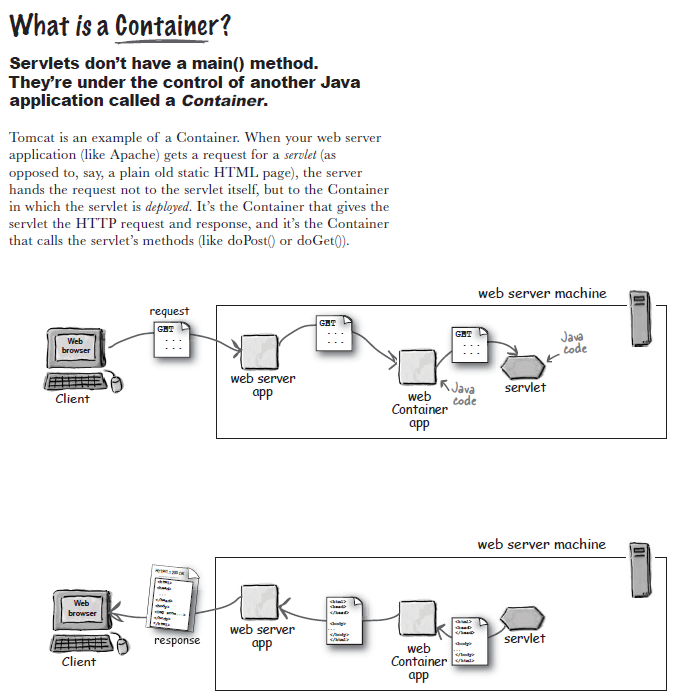


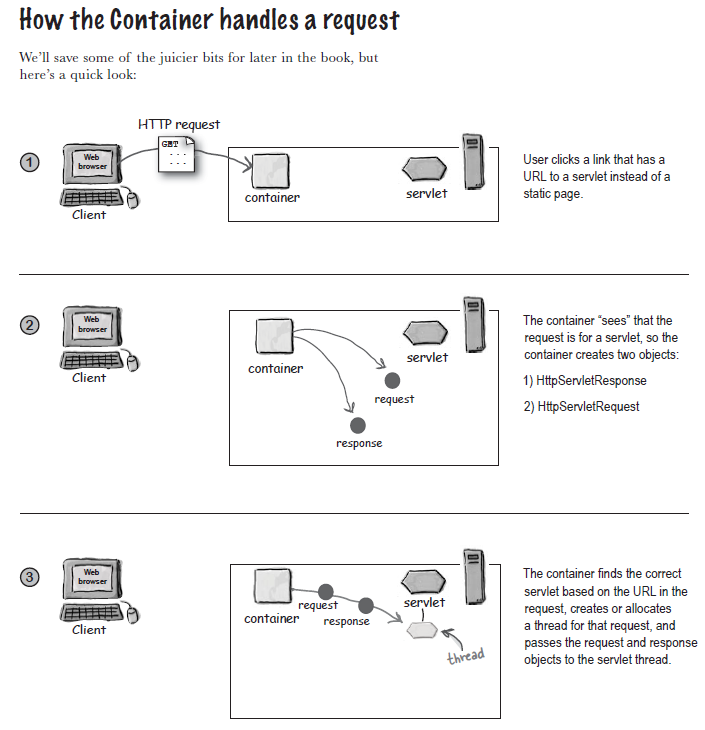
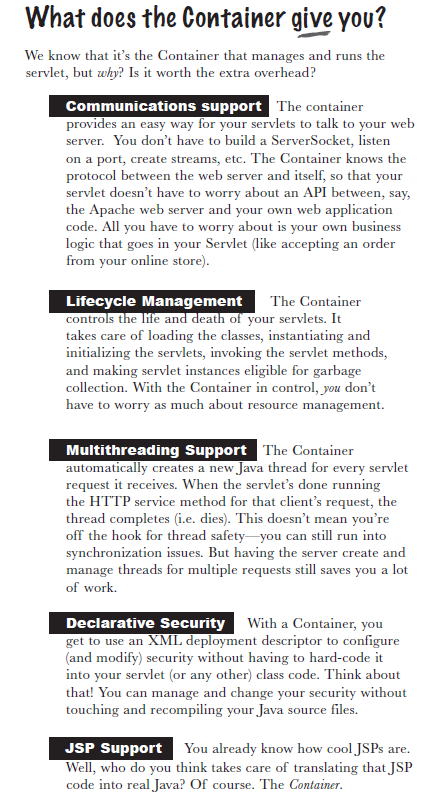


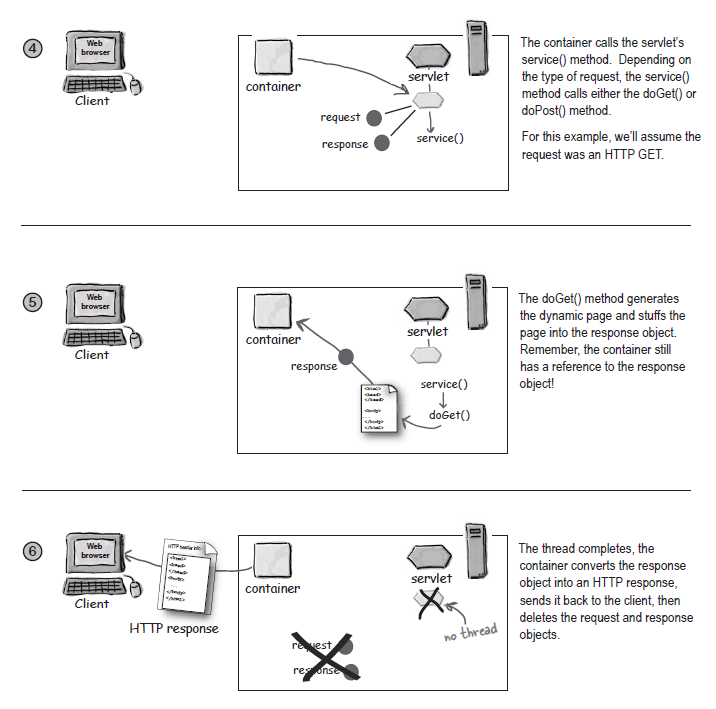


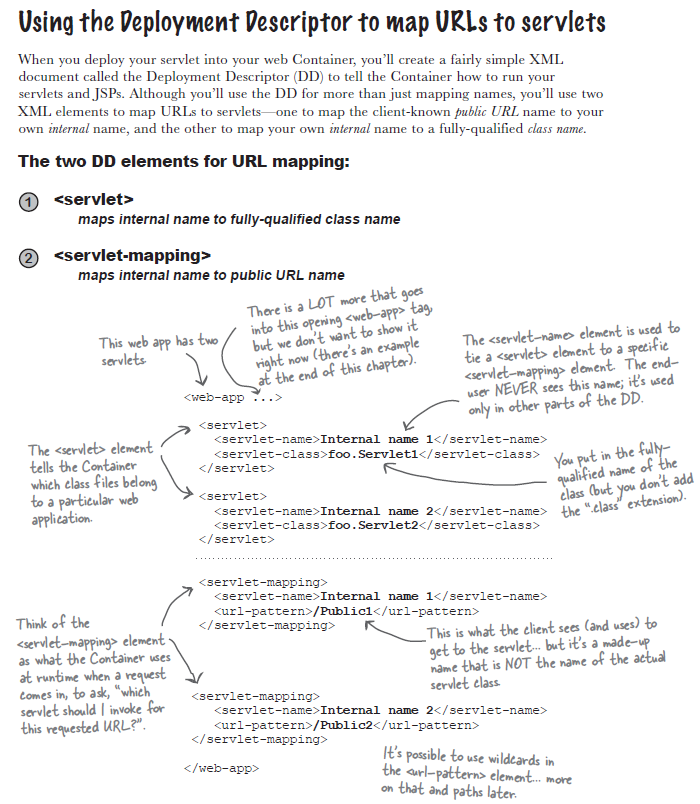
Web сервер работает только со статическими данными (?), для получения динамических страниц или сохранения данных в БД сервер использует вспомогательные приложения (например, CGI или Servlets).

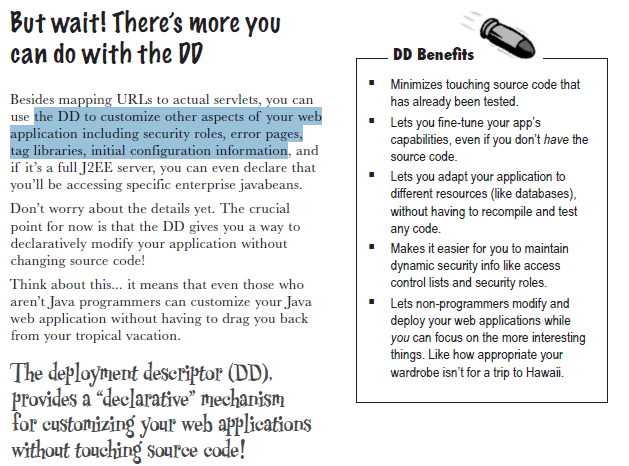
Веб сервер общается с servlet-ами не напрямую, а через контейнер сервлетов (например, tomcat). Контейнер управляет жизненным циклом сервлета, в зависимости от запроса вызывает необходимые методы, … Для каждого запрос создается отдельный java Thread.







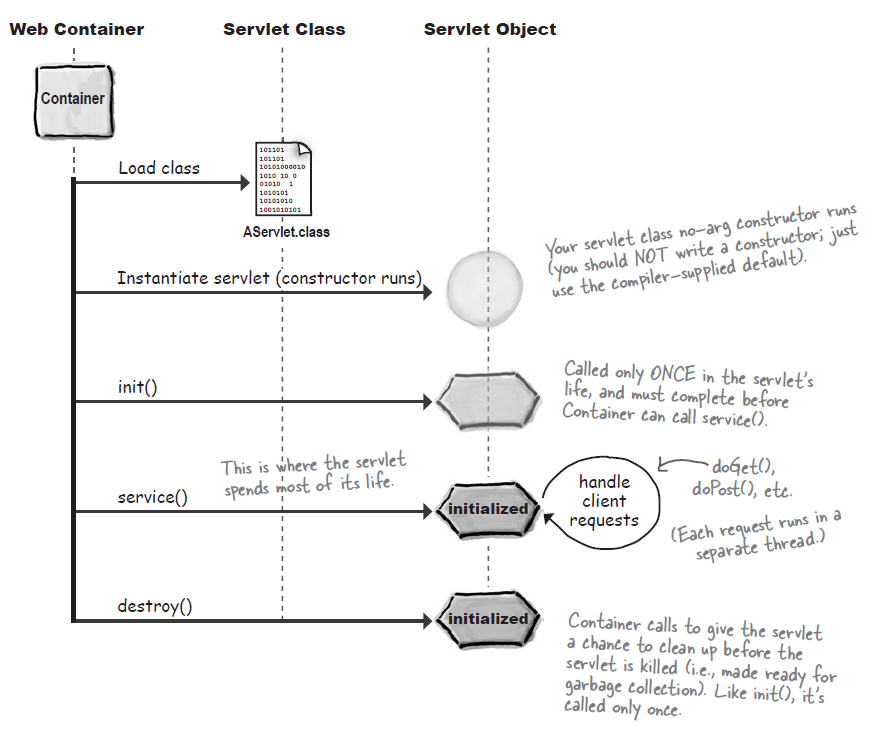


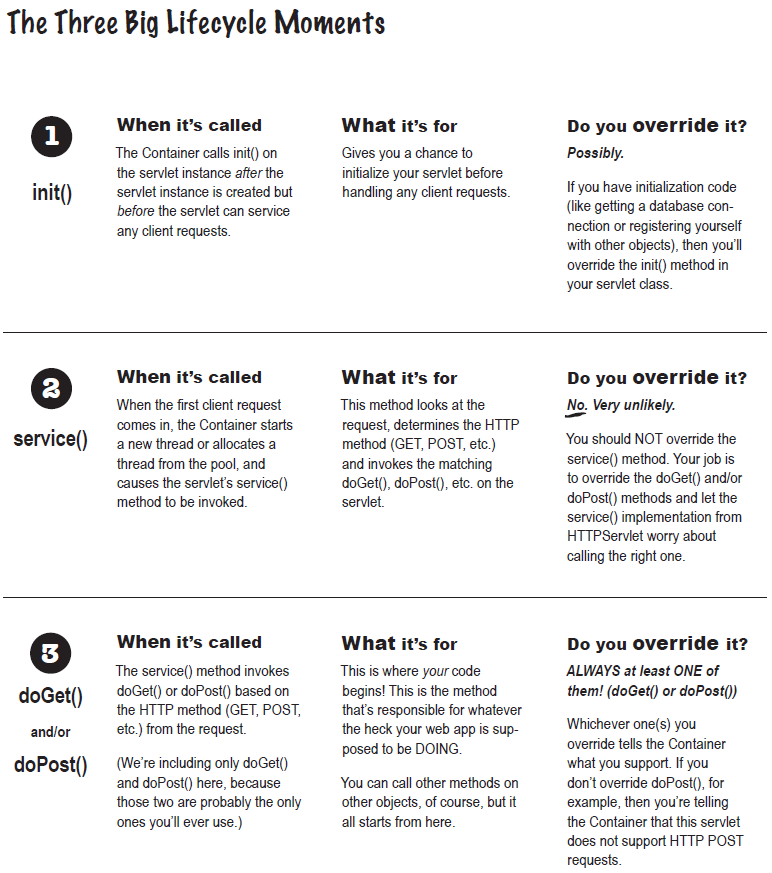


Роль контейнера в жизненном цикле сервлетов:

1. Создание request и response объектов.
2. Создание или выделение потока для сервлета.
3. Вызов метода service() сервлета, с передачей объектов request и response.

Жизненный цикл сервлета:





Сервлет и многопоточность:

A Java servlet container / web server is typically multithreaded. That means, that multiple requests to the same servlet may be executed at the same time. Therefore, you need to take concurrency into consideration when you implement your servlet.

I am not going to describe Java concurrency in great detail here. If you are interested in this topic, read my tutorial on [Java Concurrency](http://tutorials.jenkov.com/java-concurrency/index.html).

To make sure that a servlet is thread safe, there are a few basic rules of thumb you must follow:

1. Your servlet service() method should not access any member variables, unless these member variables are thread safe themselves.
2. Your servlet service() should not reassign member variables, as this may affect other threads executing inside the service() method. If you really, really need to reassign a member variable, make sure this is done inside a synchronized block.
3. Rule 1 and 2 also counts for static variables.
4. Local variables are always thread safe. Keep in mind though, that the object a local variable points to, may not be so. If the object was instantiated inside the method, and never escapes, there will be no problem. On the other hand, a local variable pointing to some shared object, may still cause problems. Just because you assign a shared object to a local reference, does not mean that object automatically becomes thread safe.

The request and response objects are of course thread safe to use. A new instance of these are created for every request into your servlet, and thus for every thread executing in your servlet.

Here is a diagram which illustrates the servlet concurrency rules / issues mentioned above. The red boxes represent state (variables) that your servlet's service() method should be careful about accessing.

|  |
| --- |
| Java Servlet Concurrency Rules |
| **Java Servlet Concurrency Rules.** |

## Other Shared Resources

Of course it is not only the member variables and static variables inside the servlet class itself, that you need to be careful about accessing. Static variables in any other class which are accessed by your servlet, must also be thread safe. The same is true for member variables of any shread objects accessed by your servlet.

## Code Example

Here is a code example that shows you some of the rules I have been talking about in this text.

public class SimpleHttpServlet extends HttpServlet {

// Not thread safe, static.

**protected static List list = new ArrayList();**

// Not thread safe

**protected Map map = new HashMap();**

// Thread safe to access object, not thread safe to reassign variable.

**protected Map map = new ConcurrentHashMap();**

// Thread safe to access object (immutable), not thread safe to reassign variable.

**protected String aString = "a string value";**

protected void doGet( HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

// Not thread safe, unless the singleton is 100% thread safe.

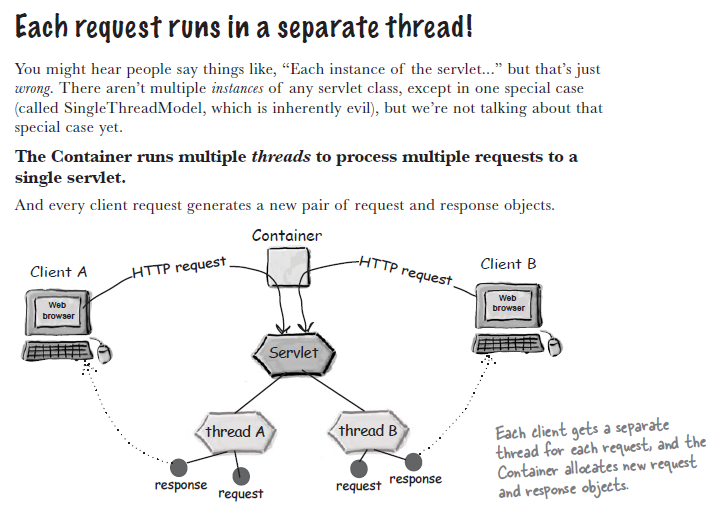
**SomeClass.getSomeStaticSingleton();**

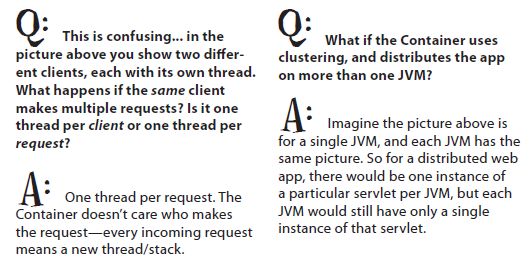
// Thread safe, locally instantiated, and never escapes method.

**Set set = new HashSet();**

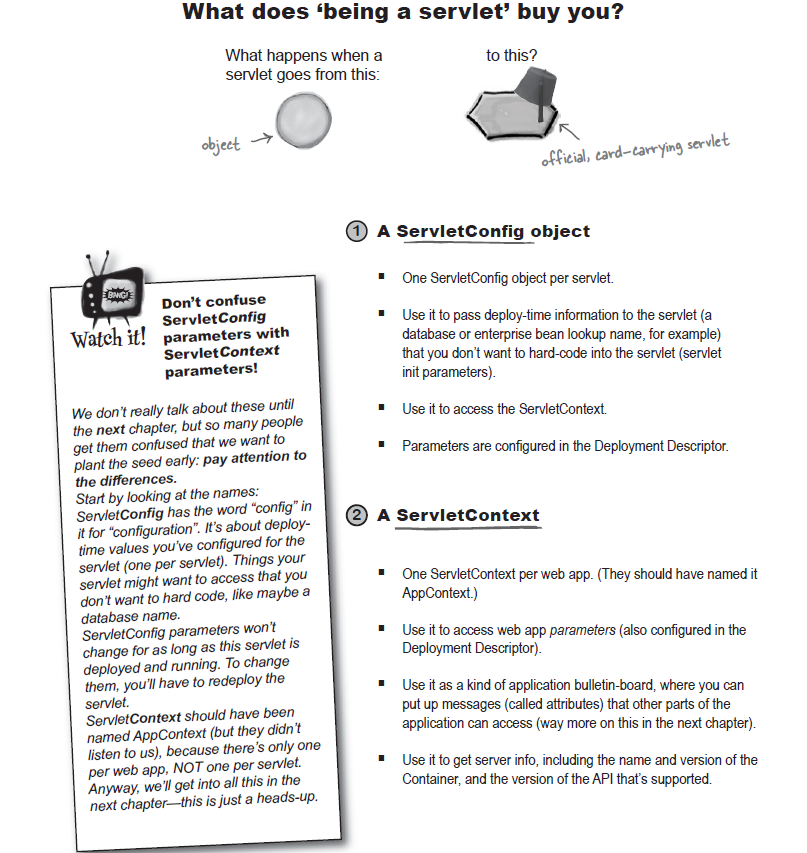
}

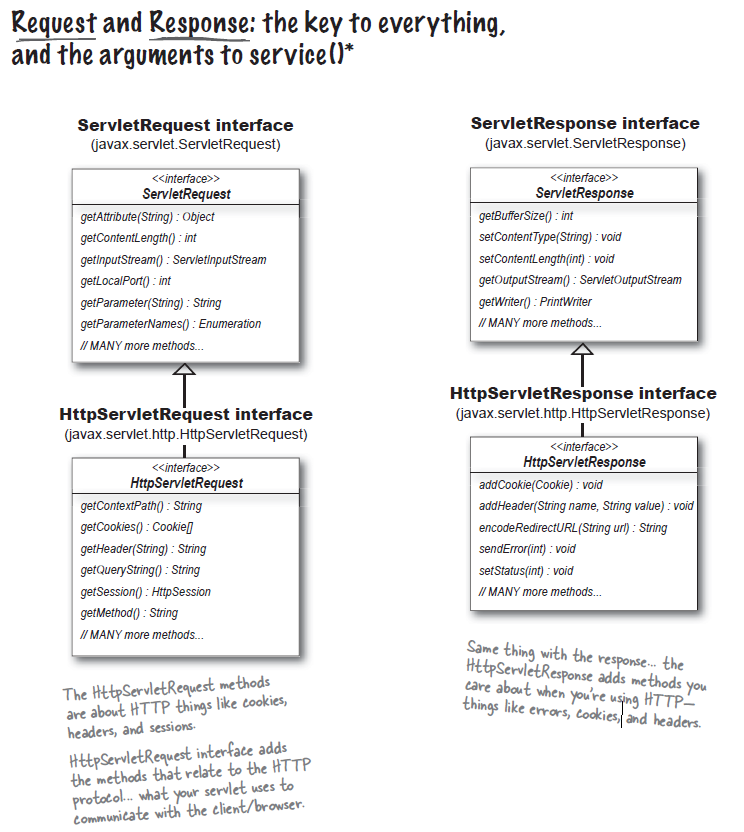
}

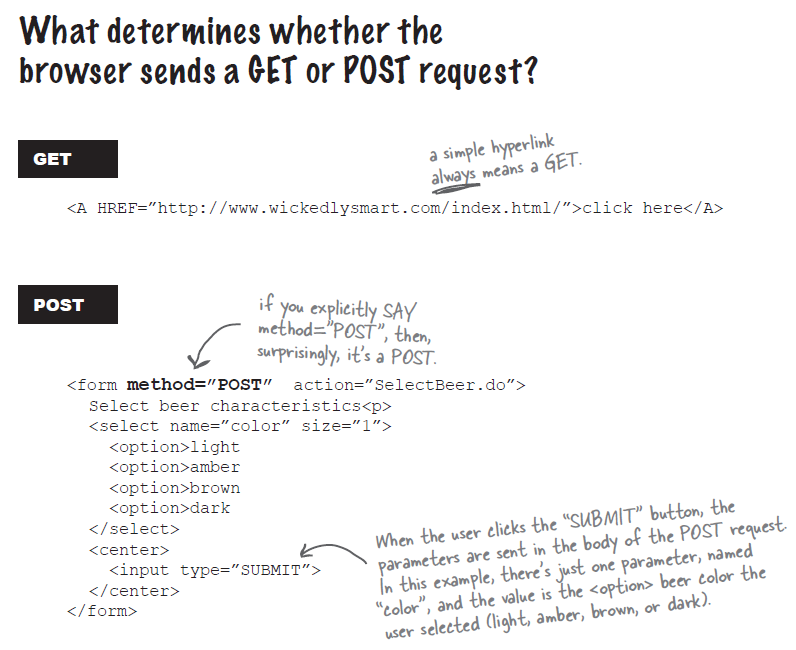




Method init() always completes before the first call to service().



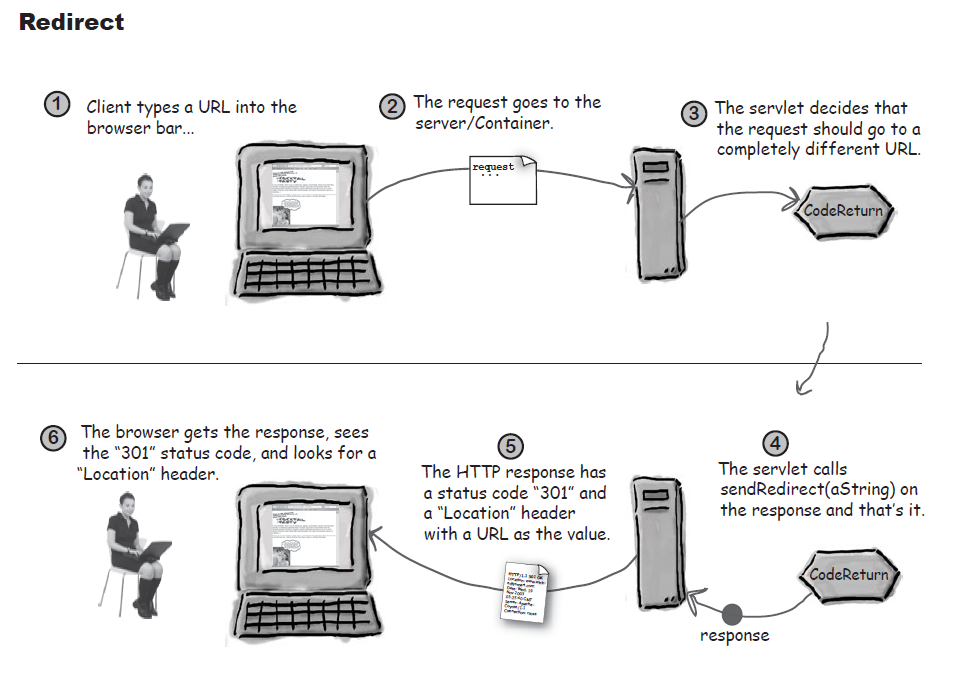


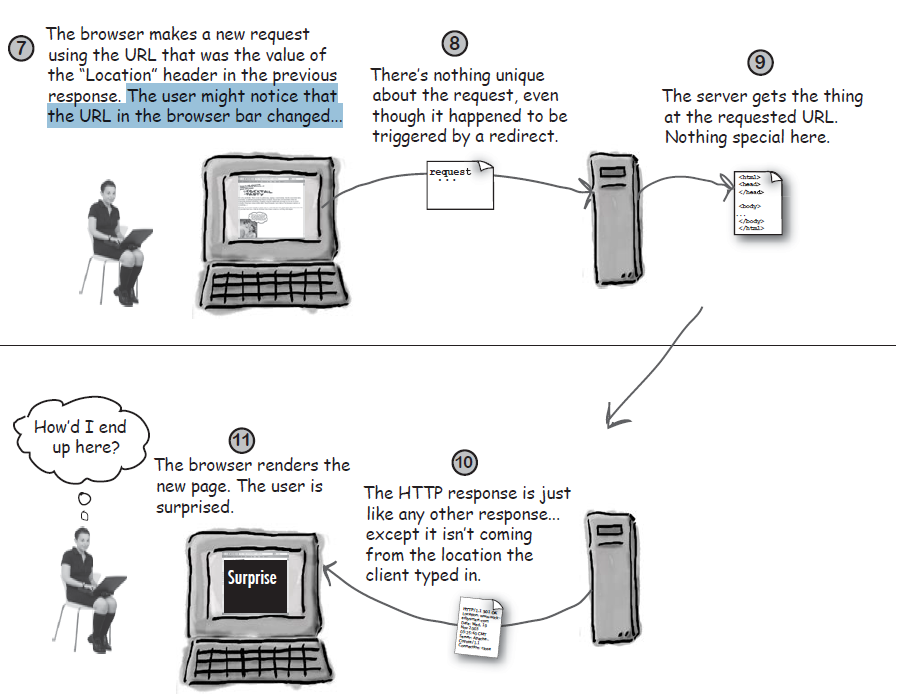


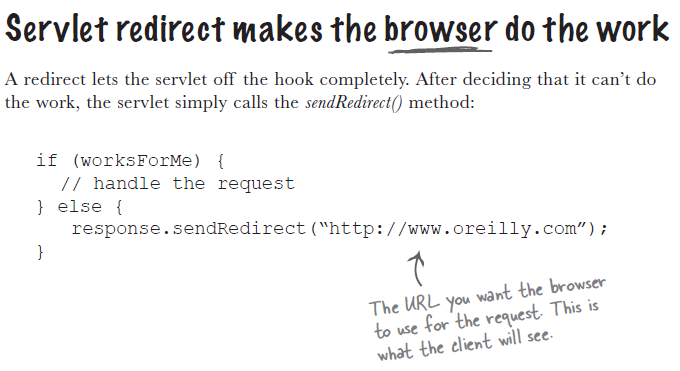
**POST** is not a default method for the form tag!

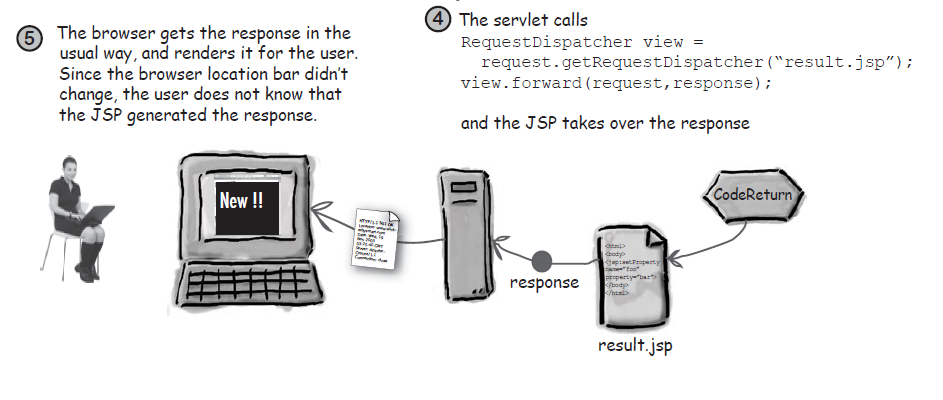
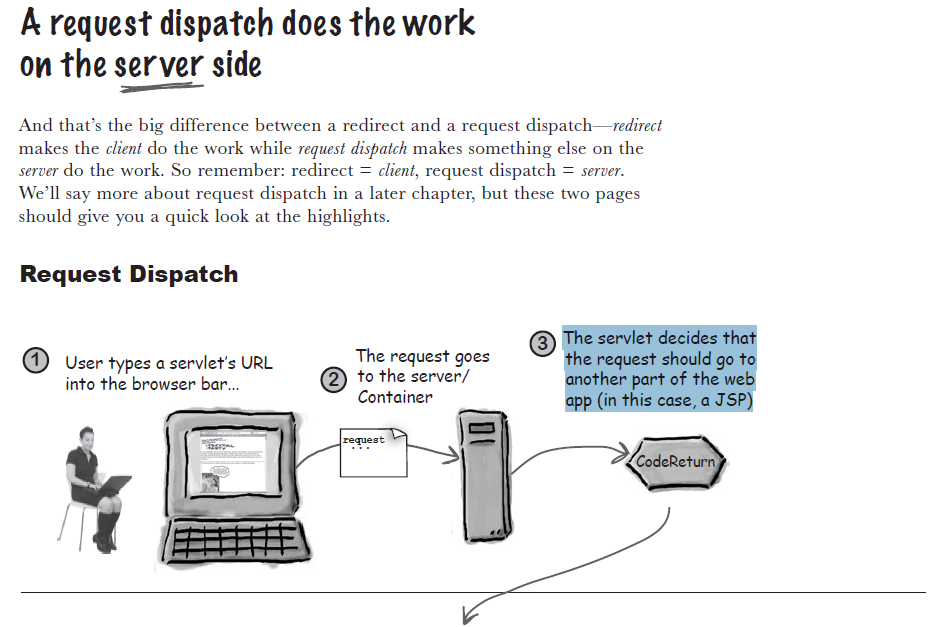
Для объекта Response можно устанавливать значения существующих Headers и создавать новые.



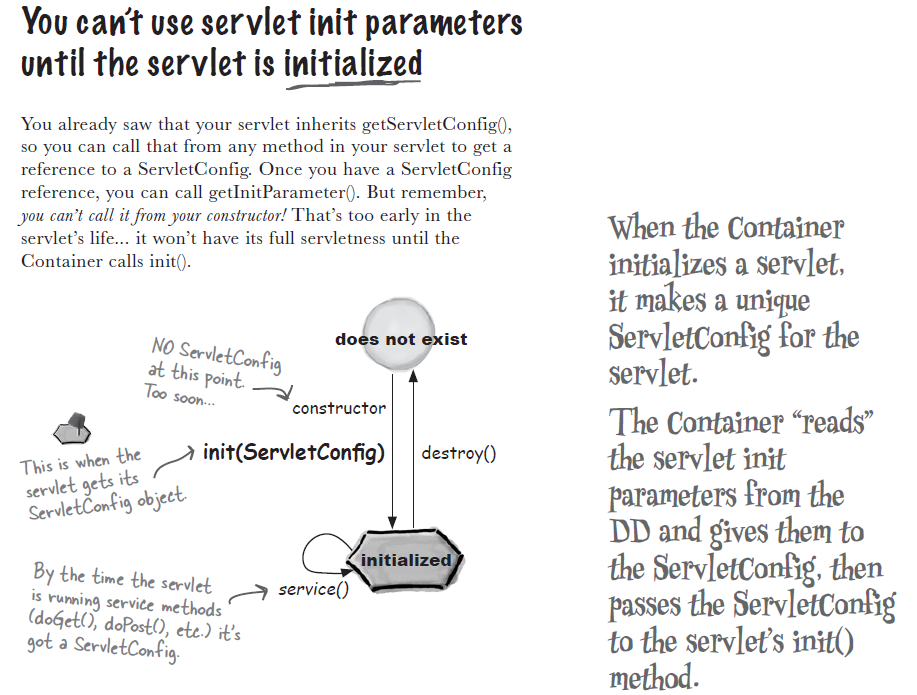


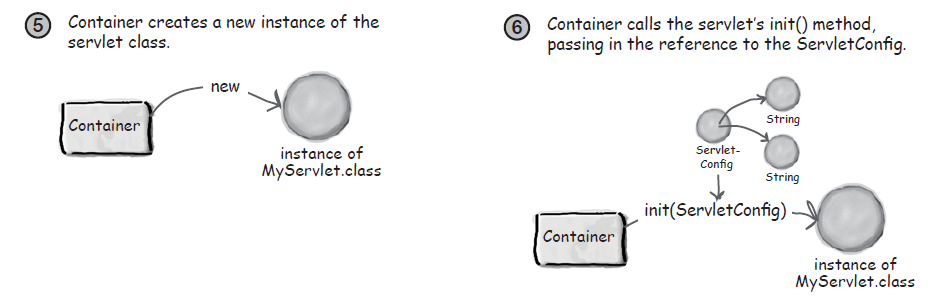
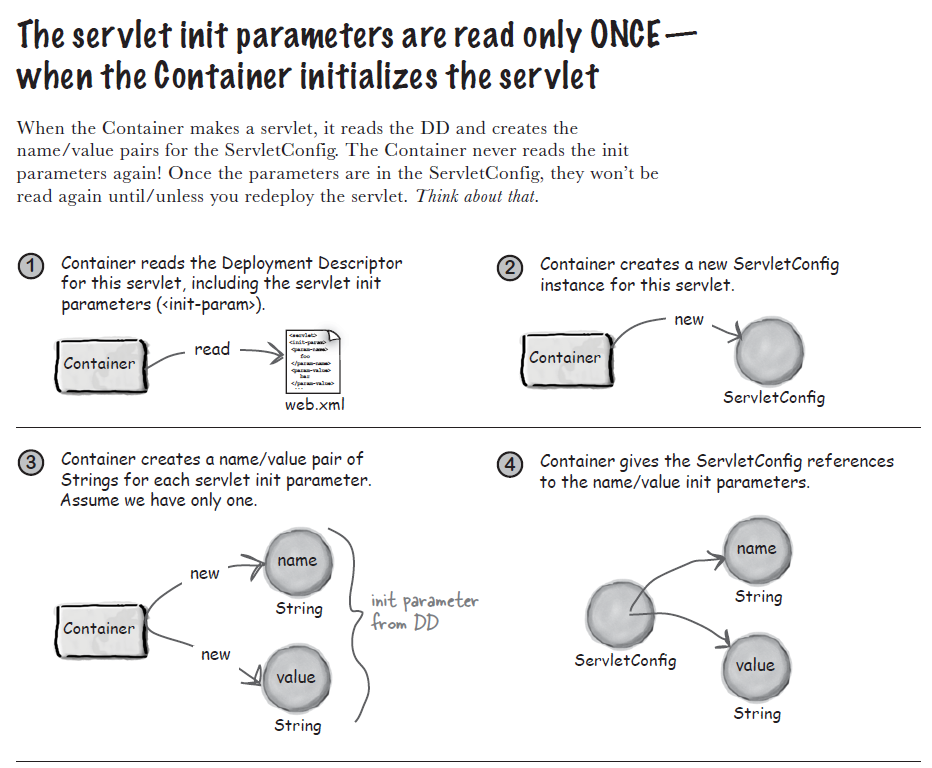


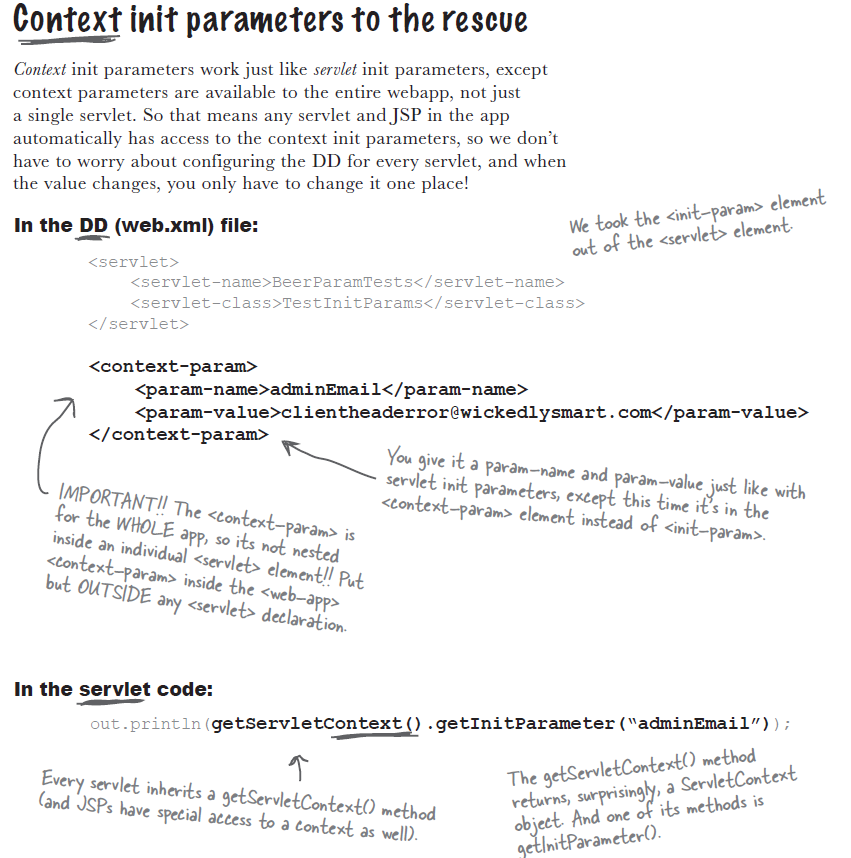


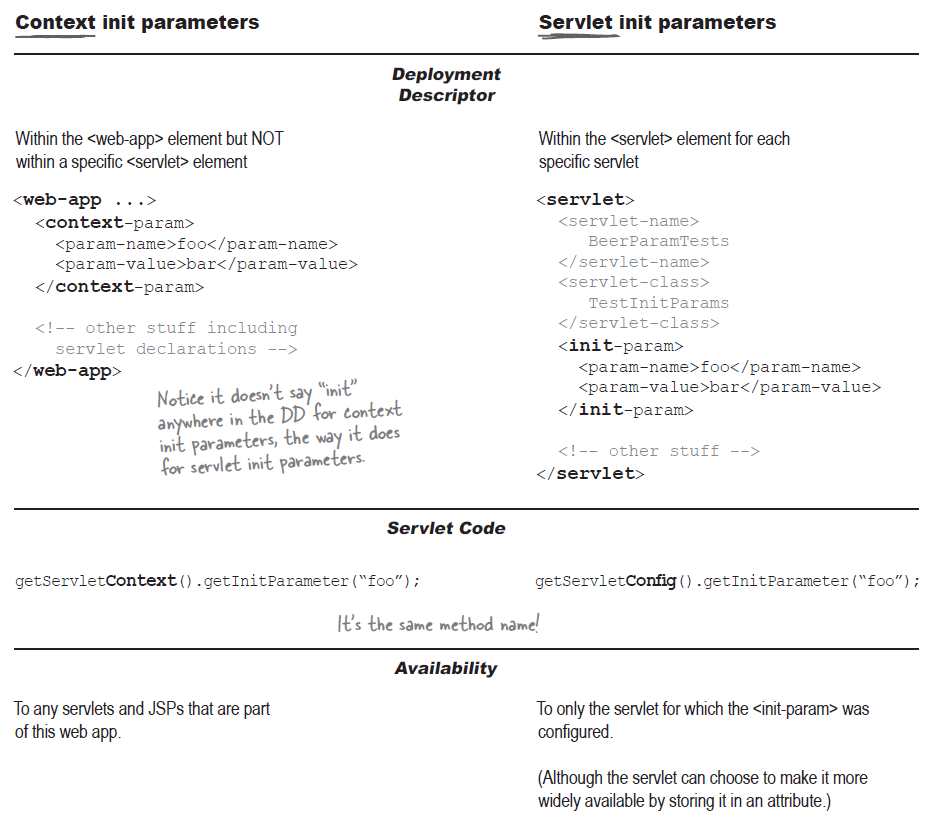


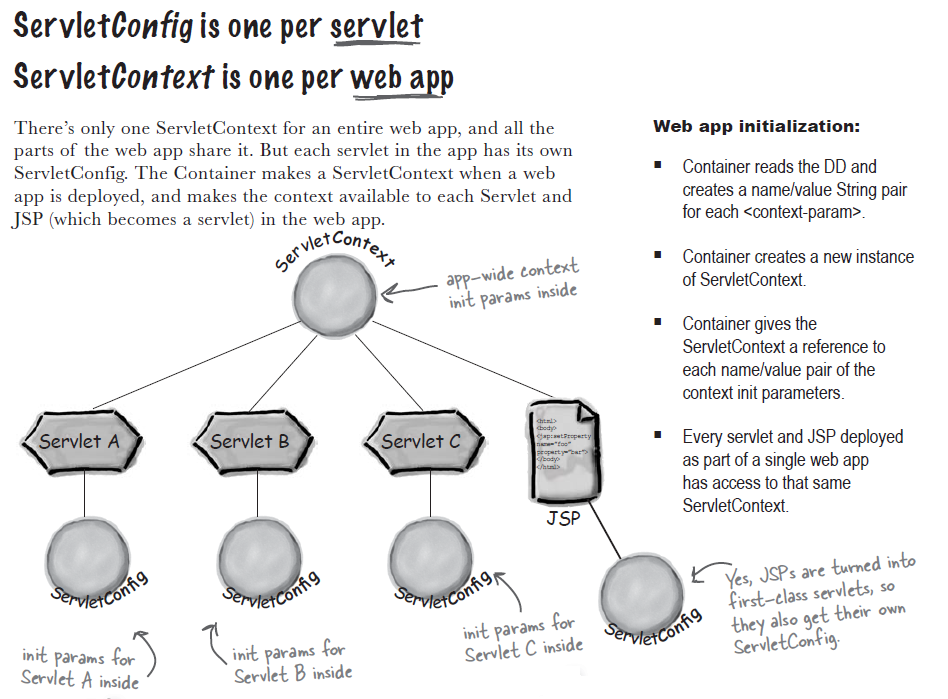












Параметры для ServletConfig и ServletContext читаются из web.xml один раз и не изменяются (нет setXXX() методов).

Стартовая точка для веб-приложения – ContextListener (?)

