Web 3

Lesson 7: Configuration

EXAM QUESTIONS…

- ☑ Where do you store properties in a web application?
- ☑ What is the ServletContext?
- ☑ …

PROBLEM?

properties hardcoded

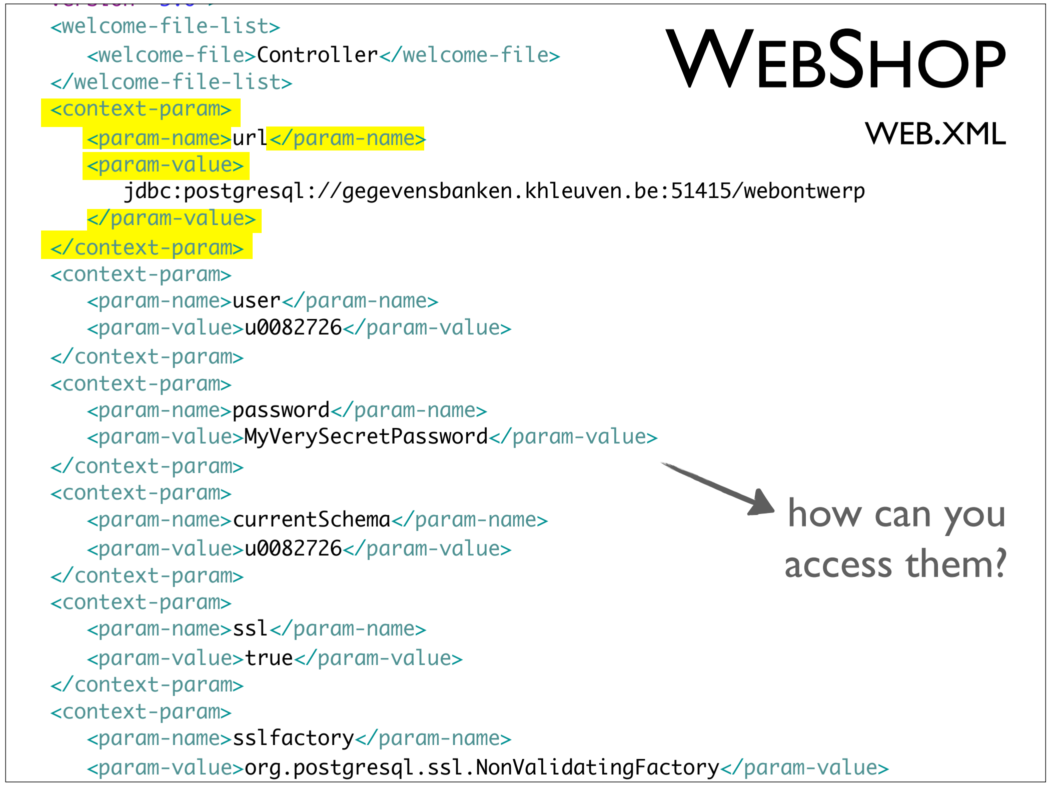
```
public class PersonDbSql implements PersonDb {
```

```
public class ProductDbSql implements ProductDb {
    private Properties properties = new Properties();
    private String url =
         "jdbc:postgresql://gegevensbanken.khleuven.be:51617/lector?currentSchema=u008272

    public ProductDbSql() {
        properties.setProperty("user", "XXX");
        properties.setProperty("password", "XXX");
        properties.setProperty("ssl", "true");
        properties.setProperty("sslfactory", "org.postgresql.ssl.NonValidatingFactory");

        try {
            Class.forName("org.postgresql.Driver");
        } catch (ClassNotFoundException e) {
            throw new DbException(e.getMessage(), e);
        }
    }

    @Override
    public Product get(int productId) {
        Product product = null;
        String sql = "SELECT * FROM product WHERE productId = ? ";
```

DRY !

SOLUTION

- Store properties
  - configuration file $\searrow$ web.xml
- Read properties
  - when application starts



WEBSHOP

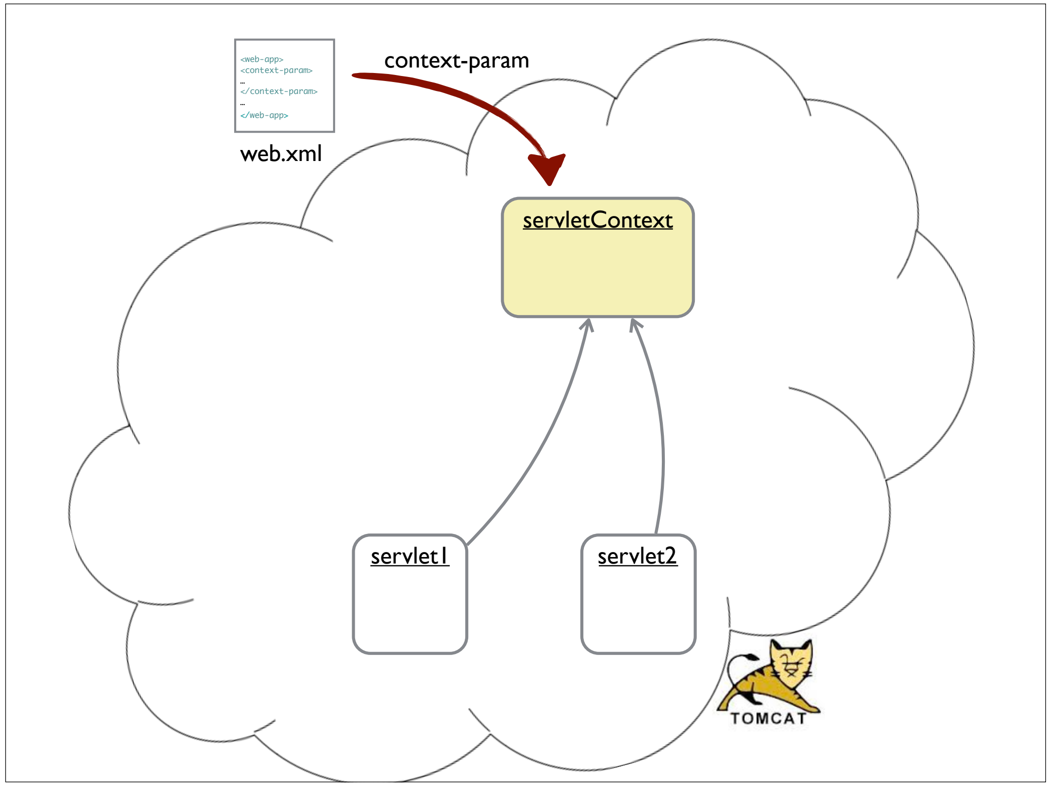
WEB.XML

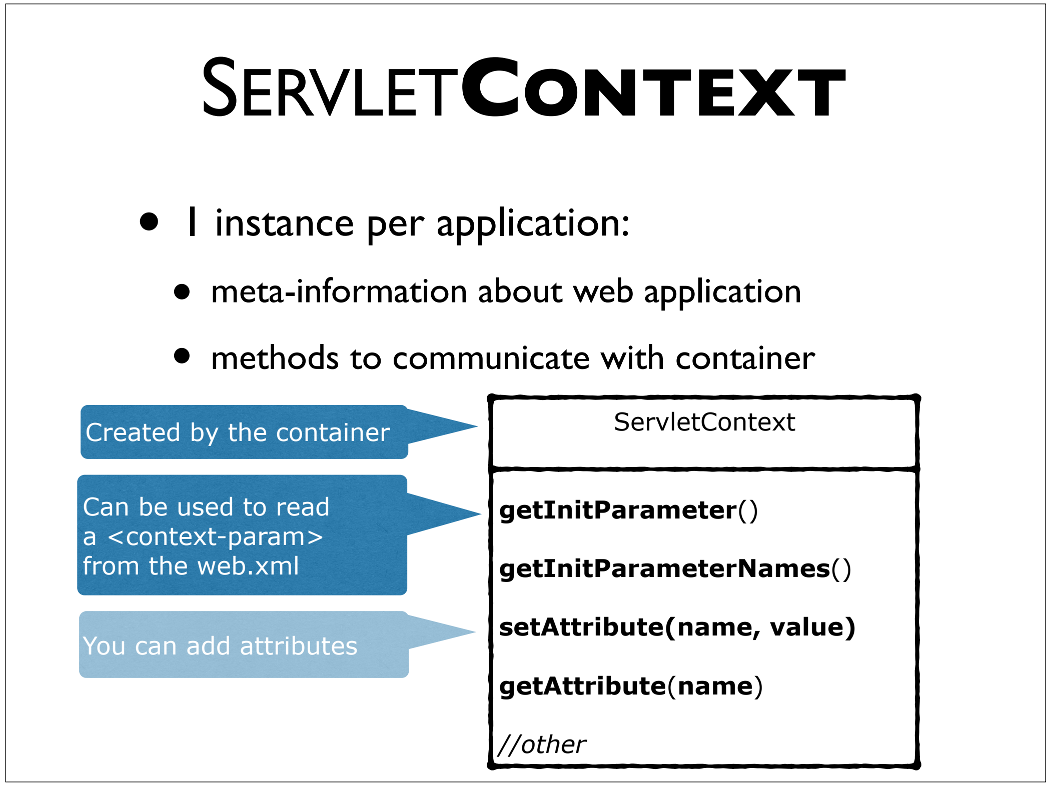
```xml
<welcome-file-list>
   <welcome-file>Controller</welcome-file>
</welcome-file-list>
<context-param>
   <param-name>url</param-name>
   <param-value>
      jdbc:postgresql://gegevensbanken.khleuven.be:51415/webontwerp
   </param-value>
</context-param>
<context-param>
   <param-name>user</param-name>
   <param-value>u0082726</param-value>
</context-param>
<context-param>
   <param-name>password</param-name>
   <param-value>MyVerySecretPassword</param-value>
</context-param>
<context-param>
   <param-name>currentSchema</param-name>
   <param-value>u0082726</param-value>
</context-param>
<context-param>
   <param-name>ssl</param-name>
   <param-value>true</param-value>
</context-param>
<context-param>
   <param-name>sslfactory</param-name>
   <param-value>org.postgresql.ssl.NonValidatingFactory</param-value>
```

how can you access them?

In the web.xml you can add properties using the tag <context-param>.

How can we access those context parameters?


```
<web-app>
<context-param>
…
</context-param>
…
</web-app>
```
web.xml
context-param
servletContext
servlet1
servlet2
TOMCAT



SERVLETCONTEXT

- I instance per application:
  - meta-information about web application
  - methods to communicate with container

Created by the container

Can be used to read
a <context-param>
from the web.xml

You can add attributes

ServletContext

getInitParameter()

getInitParameterNames()

setAttribute(name, value)

getAttribute(**name**)

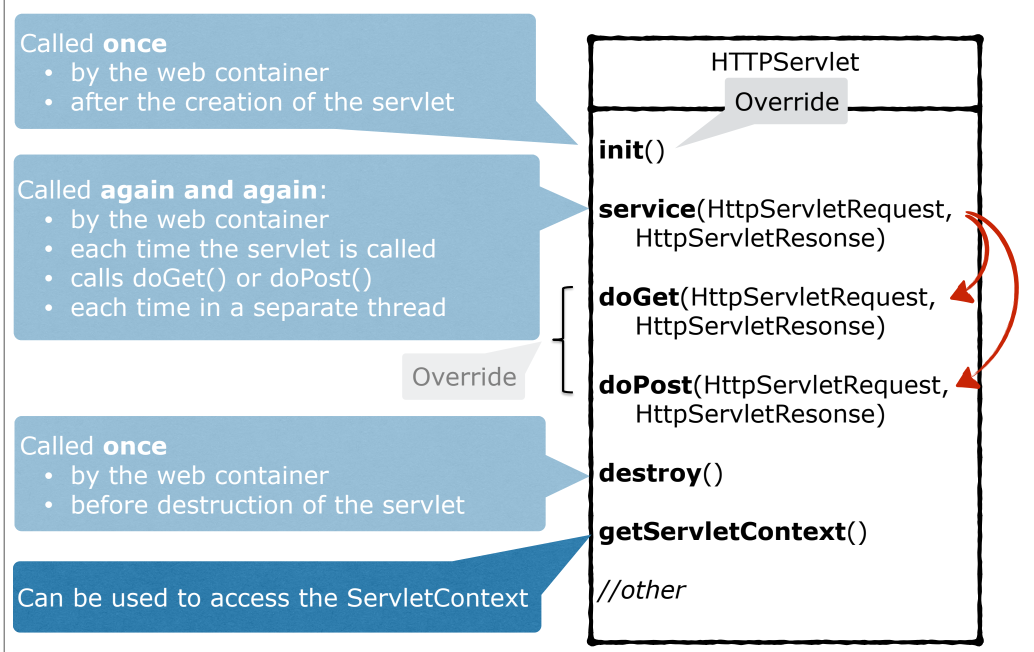
//other

When an application is deployed, the container creates 1 ServletContext object to store meta-information about the application.
The context parameters of the web.xml are automatically stored in this object.

When will we read these context parameters?

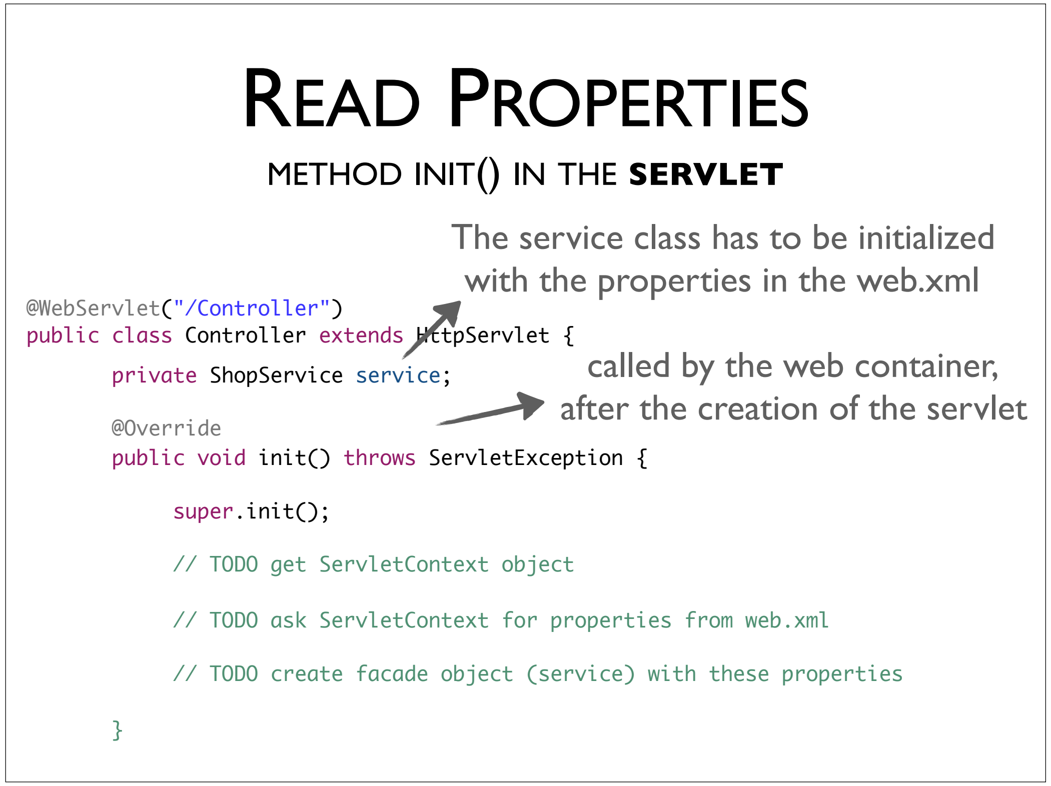
We can use the method getInitParameter to read the a property

HTTPServlet


Called **once**
• by the web container
• after the creation of the servlet
Called **again and again**:
• by the web container
• each time the servlet is called
• calls doGet() or doPost()
• each time in a separate thread
Override
Called **once**
• by the web container
• before destruction of the servlet
Can be used to access the ServletContext
HTTPServlet
Override
init()
service(HttpServletRequest, HttpServletResonse)
doGet(HttpServletRequest, HttpServletResonse)
doPost(HttpServletRequest, HttpServletResonse)
destroy()
getServletContext()
//other

We discussed the HTTPServlet class in Web 2.
This class also has a method getServletContext() which we can use to get hold of the ServletContext object.

We saw that the HTTPServlet class has a method init(), which is called immediately after the creation of the servlet.
We can override this method to get the context parameters from the ServletContext and use them.



READ PROPERTIES

METHOD INIT() IN THE **SERVLET**

The service class has to be initialized with the properties in the web.xml

called by the web container, after the creation of the servlet

```
@WebServlet("/Controller")
public class Controller extends HttpServlet {

       private ShopService service;

       @Override
       public void init() throws ServletException {

            super.init();

            // TODO get ServletContext object

            // TODO ask ServletContext for properties from web.xml

            // TODO create facade object (service) with these properties

       }
```

Dictionary is the service class in this example application. For User Management, this would be the UserSystem class

ServletContext

METHOD INIT() IN THE **SERVLET**

```java
private ShopService service;

@Override
public void init() throws ServletException {

     super.init();

     ServletContext context = getServletContext();
```

ask for the ServletContext

```java
     // TODO ask ServletContext for properties from web.xml

     // TODO create facade object (service) with these properties

}
```

CONTEXT.GETINITPARAMETER()

METHOD INIT() IN THE **SERVLET**

```
private ShopService service;

@Override
public void init() throws ServletException {

     super.init();

     ServletContext context = getServletContext();

     Properties properties = new Properties();
     properties.setProperty("user", context.getInitParameter("user"));
     properties.setProperty("password", context.getInitParameter("password"));
     properties.setProperty("ssl", context.getInitParameter("ssl"));
     properties.setProperty("sslfactory", context.getInitParameter("sslfactory"));
     properties.setProperty("url", context.getInitParameter(“url"));

     // TODO create facade object (service) with these properties

}
```

read parameters from web.xml

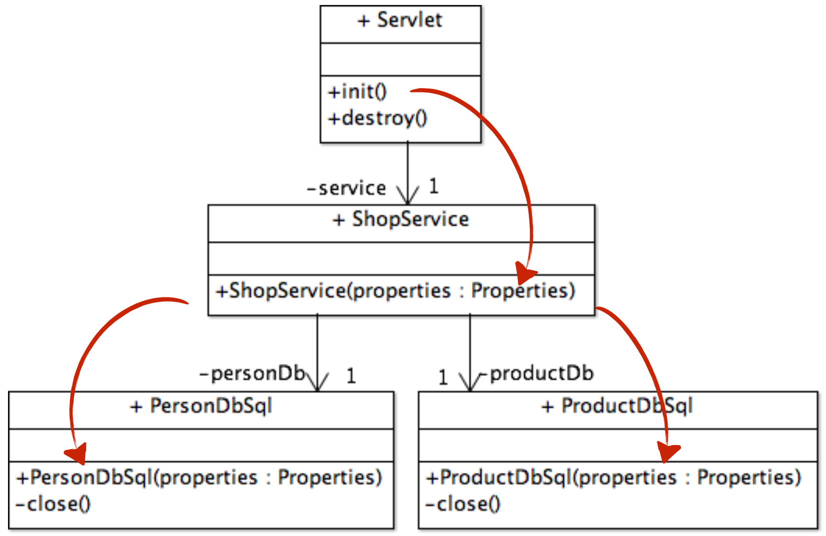
Create Model: Facade

METHOD INIT() IN THE **SERVLET**

```
private ShopService service;

@Override
public void init() throws ServletException {

     super.init();

     ServletContext context = getServletContext();

     Properties properties = new Properties();
     properties.setProperty("user", context.getInitParameter("user"));
     properties.setProperty("password", context.getInitParameter("password"));
     properties.setProperty("ssl", context.getInitParameter("ssl"));
     properties.setProperty("sslfactory", context.getInitParameter("sslfactory"));
     properties.setProperty("url", context.getInitParameter(“url"));

     service = new ShopService(properties);

}
```

create facade object
with properties

CLASS DIAGRAM



+ Servlet
+init()
+destroy()
-service
1
+ ShopService
+ShopService(properties : Properties)
-personDb
1
1
-productDb
+ PersonDbSql
+PersonDbSql(properties : Properties)
-close()
+ ProductDbSql
+ProductDbSql(properties : Properties)
-close()

Refactor Facade

```java
public class ShopService {
     private PersonDb personDb;
     private ProductDb productDb;

     public ShopService(Properties properties) {
          personDb = new PersonDbSQL(properties);
          productDb = new ProductDbSQL(properties);
     }

     ...
}
```

REFACTOR DB-CLASSES

```
public class ProductDbSql {
   private Properties properties;
   private String url;

   public ProductDbSql(Properties properties) {
      try {
         Class.forName("org.postgresql.Driver");
         this.properties = properties;
         this.url = properties.getProperty("url");
      } catch (Exception e) {
         throw new DbException(e.getMessage(), e);
      }
   }

   ...
}
```

Remark

method init() - Better

```
private ShopService service;

@Override
public void init() throws ServletException {

    super.init();

    ServletContext context = getServletContext();

    Properties properties = new Properties();
    Enumeration<String> parameterNames = context.getInitParameterNames();
    while (parameterNames.hasMoreElements()){
       String propertyName = parameterNames.nextElement();
       properties.setProperty(propertyName, context.getInitParameter(propertyName));
    }

    service = new ShopService (properties);

}
```

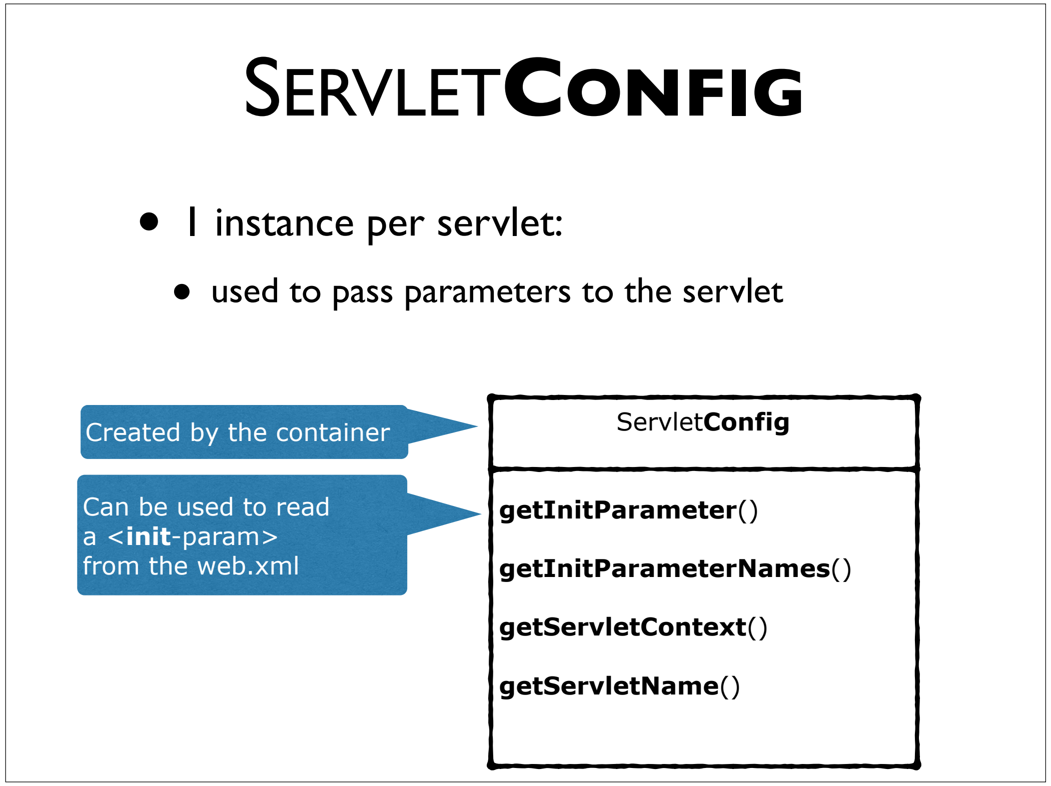
↘ more flexible

WHY IN INIT() METHOD AND NOT IN CONSTRUCTOR?

Because we have to wait for the **ServletContext**-object:

1. Container creates servlet
2. Container creates:
   - ServletConfig
   - ServletContext
3. Container calls init()

?



ServletConfig

- 1 instance per servlet:
  - used to pass parameters to the servlet

Created by the container

Can be used to read a <**init**-param> from the web.xml

ServletConfig

getInitParameter()

getInitParameterNames()

getServletContext()

getServletName()

ServletConfig object is created by web container for each servlet to pass information to a servlet during initialization. This object can be used to get configuration information from web.xml file.

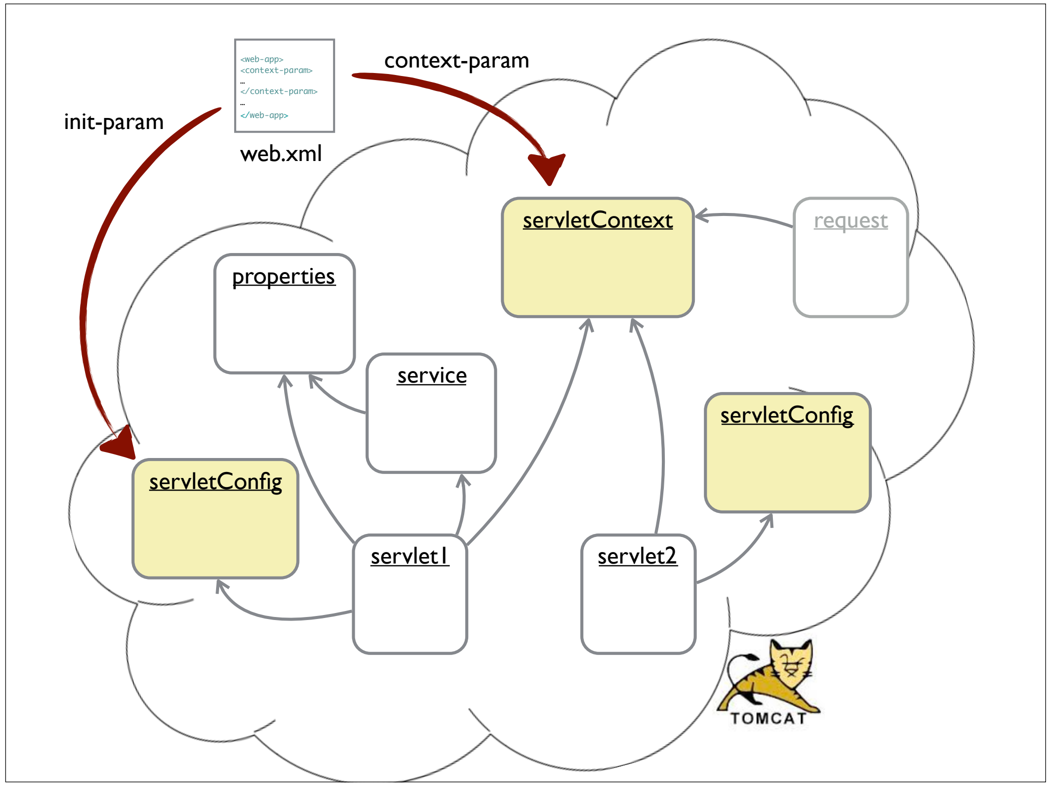
We can use the method getInitParameter to read the a property

Example: Tomcat

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://java.sun.com/xml/ns/javaee" xmlns:jsp="http://java.sun.com/xml/ns/
javaee/jsp"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3_0.xsd"
   version="3.0">
   <servlet>
      <servlet-name>default</servlet-name>
      <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
      <init-param>
         <param-name>debug</param-name>
         <param-value>0</param-value>
      </init-param>
      <init-param>
         <param-name>listings</param-name>
         <param-value>false</param-value>
      </init-param>
      <load-on-startup>1</load-on-startup>
   </servlet>
```

QUESTION

- How does the Tomcat memory look like, after calling the init() method?



```
<web-app>
<context-param>
…
</context-param>
…
</web-app>
```

context-param
init-param
web.xml
servletContext
request
properties
service
servletConfig
servletConfig
servlet1
servlet2
TOMCAT