Web Frameworks

and

Struts 2

Problem area

- Mixing application logic and markup is bad practise
  - Harder to change and maintain
  - Error prone
  - Harder to re-use

```
public void doGet( HttpServletRequest request, HttpServletResponse response )
{
    PrintWriter out = response.getWriter();

    out.println( ”<html>\n<body>” );

    if ( request.getParameter( ”foo” ).equals( ”bar” ) )
        out.println( ”<p>Foo is bar!</p>” );
    else
        out.println( ”<p>Foo is not bar!</p>” );

    out.println( ”</body>\n</html>” );
}
```

Advantages

- Separation of application logic and web design through the *MVC pattern*
- Integration with template languages
- Some provides built-in components for
  - Form validation
  - Error handling
  - Internationalization
  - IDE integration

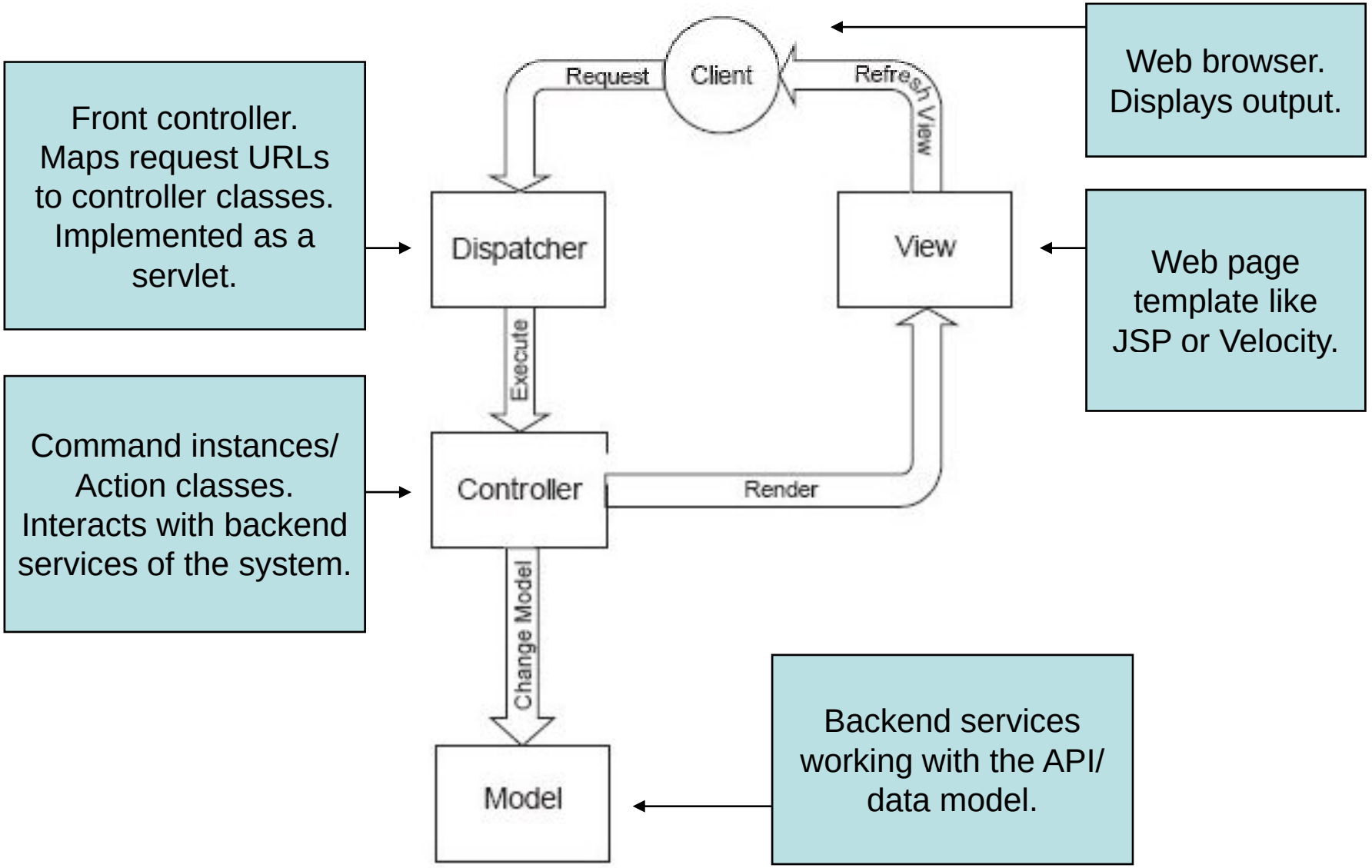
The MVC pattern

- Breaks an application into three parts:
  - Model: The domain object model / service layer
  - View: Template code / markup
  - Controller: Presentation logic / action classes
- Defines interaction between components to promote loose coupling and re-use
  - Each file has one responsibility
  - Enables division of labour between programmers and designers

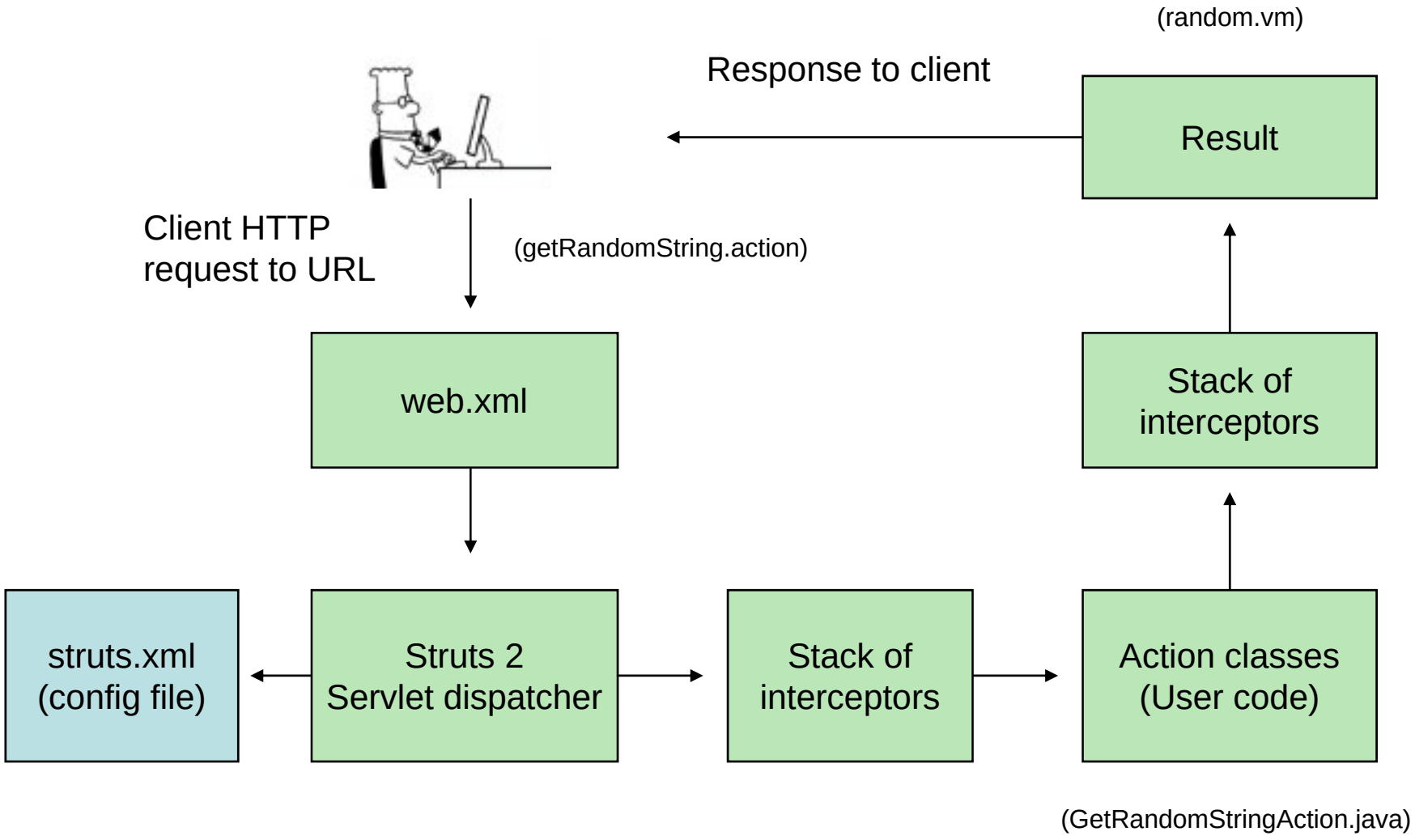
Struts 2

- Based on *WebWork*
- Built on top of *XWork* – a command pattern framework
- Integrated with the *Spring IoC container*
- Provides a clean implementation of the *MVC pattern*

MVC with Front Controller



Action Flow


(random.vm)
Response to client
Result
Client HTTP
request to URL
(getRandomString.action)
web.xml
Stack of
interceptors
struts.xml
(config file)
Struts 2
Servlet dispatcher
Stack of
interceptors
Action classes
(User code)
(GetRandomStringAction.java)

web.xml

- Maps URL patterns to the Struts dispatcher
- Most typical pattern is **.action*
- Located in WEB-INF/ folder
- Can redirect to the *Filter-* or *ServletDispatcher*

```
<filter>
    <filter-name>struts</filter-name>
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
</filter>

<filter-mapping>
    <filter-name>struts</filter-name>
    <url-pattern>*.action</url-pattern>
</filter-mapping>
```

struts.xml

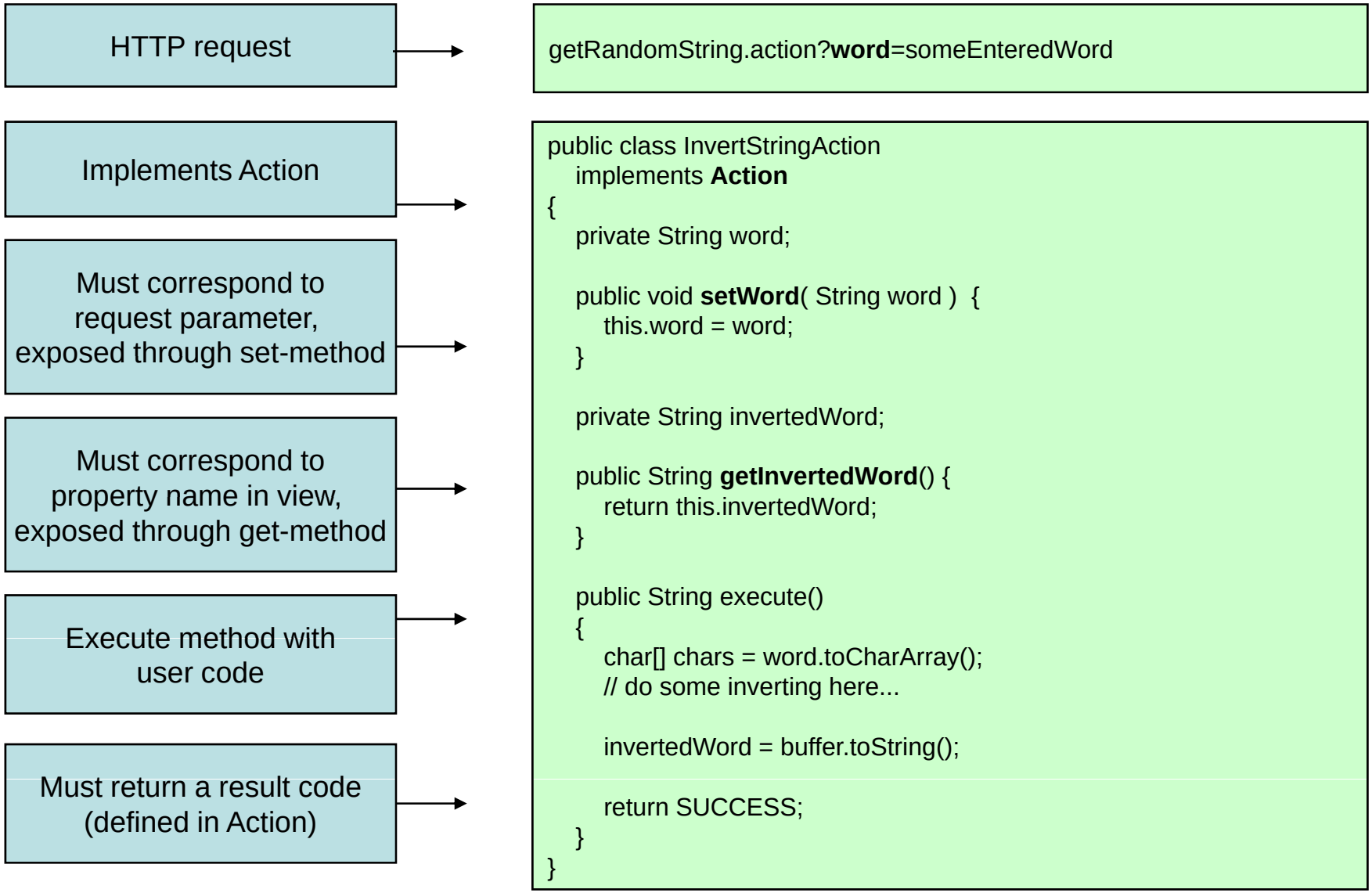
- Located in root of classpath
- *struts-default.xml* is included automatically
- Base package must extend *struts-default*
- Maps URLs to *action classes*
- Maps result codes to *results*

```
<struts>
  <package name="default" extends=”struts-default">

    <action name=”invertString" class="no.uio.inf5750.example.action.InvertStringAction">
      <result name="success" type="velocity">word.vm</result>
    </action>

  </package>
</struts>
```

Action classes

- Java code executed when a URL is requested
- Must implement the *Action* interface or extend *ActionSupport*
  - Provides the *execute* method
  - Must return a *result code* (SUCCESS, ERROR, INPUT)
  - Used to map to *results*
- Properties set by the request through public set-methods
- Properties made available to the response through public get-methods

Action classes

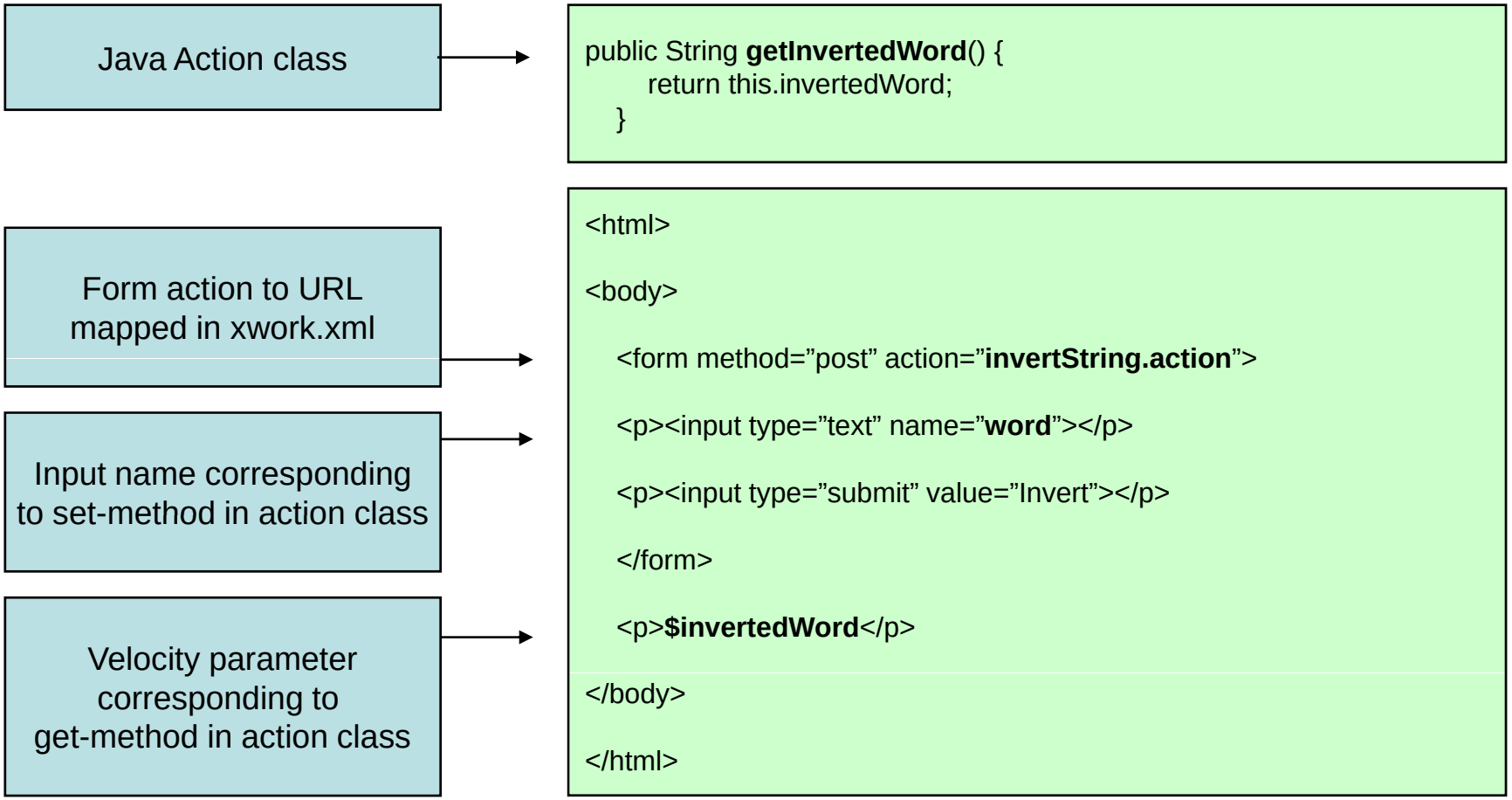


View

- Struts 2 integrates with many view technologies:
  - JSP
  - Velocity
  - Freemarker
  - JasperReports
- Values sent to controller with POST or GET as usual
- Values made available to the view by the controller

View

- *Velocity* is a popular template engine and -language



struts.xml (2)

- Different result codes can be mapped to different results

```
<struts>
  <package name="default" extends=”struts-default">

    <action name=”invertString" class="no.uio.inf5750.example.action.InvertStringAction">
      <result name="success" type="velocity">word.vm</result>
      <result name=”input” type=”velocity”>input.vm</result>
    </action>

  </package>
</struts>
```

struts.xml (3)

- *Static parameters* can be defined
- Requires public set-methods in action classes
- Automatic *type conversion* is provided

```
<struts>
  <package name="default" extends=”struts-default">

    <action name=”invertString" class="no.uio.inf5750.example.action.GetRandomStringAction">
      <result name="success" type="velocity">random.vm</result>
      <param name=”numberOfChars”>32</param>
    </action>

  </package>
</struts>
```

struts.xml (4)

- Files can *include* other files
  - Files are merged
- Facilitates breaking complex applications into manageable modules
  - Specified files are searched for in classpath
  - Configuration can be separated into multiple files / JARs

```
<struts>
   <include file=”struts-default.xml”/>

  <package name=”default" extends=”struts-default”>
    <!– Default action mappings -->
  </package>

  <include file=”struts-public.xml”/>
  <include file=”struts-secure.xml”/>
</struts>
```

struts.xml (5)

- Actions can be grouped in *packages*
- Useful for large systems to promote modular design
- A package can extend other packages
  - Definitions from the extended package are included
  - Configuration of commons elements can be centralized

```
<struts>
  <package name="default" extends=”struts-default">
    <action name=”invertString” class=”no.uio.no.example.action.InvertStringAction”> <!– mapping omitted -->
    </action>
  </package>

  <package name=”secure” extends=”default”>
    <!– Secure action mappings -->
  </package>
</struts>
```

struts.xml (6)

- Actions can be grouped in *namespaces*
- Namespaces map URLs to actions
  - Actions identified by the name and the namespace it belongs to
  - Facilitates modularization and maintainability

```
<xwork>
  <include file="webwork-default.xml"/>

  <package name="secure" extends="default" namespace="/secure">

    <action name="getUsername" class="no.uio.inf5750.example.action.GetUsernameAction">
        <result name="success" type="velocity">username.vm</result>
    </action>

  </package>

</xwork>
```

Interceptors

- Invoked before and/or after the execution of an action
- Enables centralization of concerns like security, logging

```
<struts>
  <package name="default" extends=”struts-default">

    <interceptors>
       <interceptor name=”profiling” class=”no.uio.example.interceptor.ProfilingInterceptor”/>
    </interceptors>

    <action name=”invertString” class=”no.uio.no.example.action.InvertStringAction”>
       <result name=”success” type=”velocity”>word.vm</result>
       <interceptor-ref name=”profiling”/>
    </action>

  </package>
</struts>
```

Provided interceptors

- Interceptors perform many tasks in Struts 2
  - ParametersInterceptor (HTTP request params)
  - StaticParametersInterceptor (config params)
  - ChainingInterceptor
- Many interceptor stacks provided in struts-default.xml
  - defaultStack
  - i18nStack
  - fileUploadStack and more…

Interceptor stacks

- Interceptors should be grouped in *stacks*
- A *default* interceptor stack can be defined
  - Should include the WebWork default stack

```
<struts> <!– package omitted →
   <interceptors>
      <interceptor name=”profiling” class=”no.uio.example.interceptor.ProfilingInterceptor”/>
      <interceptor name=”logging” class=”no.uio.example.logging.LoggingInterceptor”/>

      <interceptor-stack name=”exampleStack”>
         <interceptor-ref name=”defaultStack”/>
         <interceptor-ref name=”profiling”/>
         <interceptor-ref name=”logging”/>
      </interceptor-stack>

   </interceptors>

   <default-interceptor-ref name=”exampleStack”/>
</struts>
```

Result types

- Determines behaviour after the action is executed and the result is returned
- Several result types are bundled:
- Dispatcher (JSP)
  - Default - will generate a JSP view
- Velocity
  - Will generate a Velocity view
- Redirect
  - Will redirect the request to the specified action after execution
- Chain
  - Same as redirect but makes all parameters available to the following action

Result types

Chain result type.

The properties in GetRandomStringAction will be available for InvertStringAction.

Redirect result type.

Redirects the request to another action after being executed.

Velocity result type.

Generates a HTML response based on a Velocity template.

```
<struts>
  <package name="default" extends=”struts-default">

  <action name=”getRandomString” class=”no.uio...GetRandomStringAction”>
    <result name=”success” type=”chain”>invertString</result>
    <result name=”input” type=”redirect”>error.action</result>
  </action>

  <action name=”invertString” class=”no.uio...InvertStringAction”>
    <result name=”success” type=”velocity”>word.vm</result>
  </action>

  </package>
</struts>
```

Result types

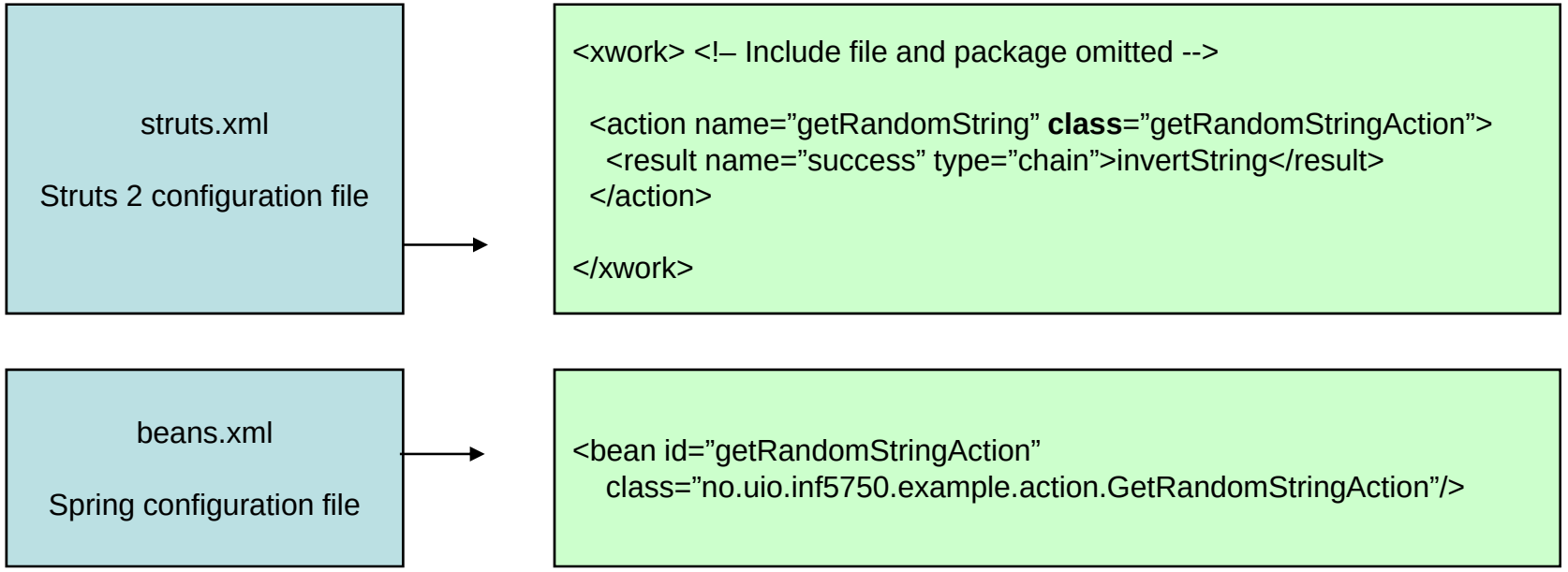
- Several provided result types integrated with ext tools
  - JasperReports
  - Flash
  - Freemarker
- Custom result types can be defined

```
<struts>
   <package name="default" extends="struts-default">

   <result-types>
      <result-type name="velocityXML"
         class="no.uio.inf5750.example.XMLResult"/>
   </result-types>

   </package>
</struts>
```

```
public class XMLResult
   implements Result
{
   public void execute( ActionInvocation invocation )
   {
      Action action = invocation.getAction();

      // Print to HTTPServletResponse or
      // modify action parameters or whatever..
   }
}
```

IoC

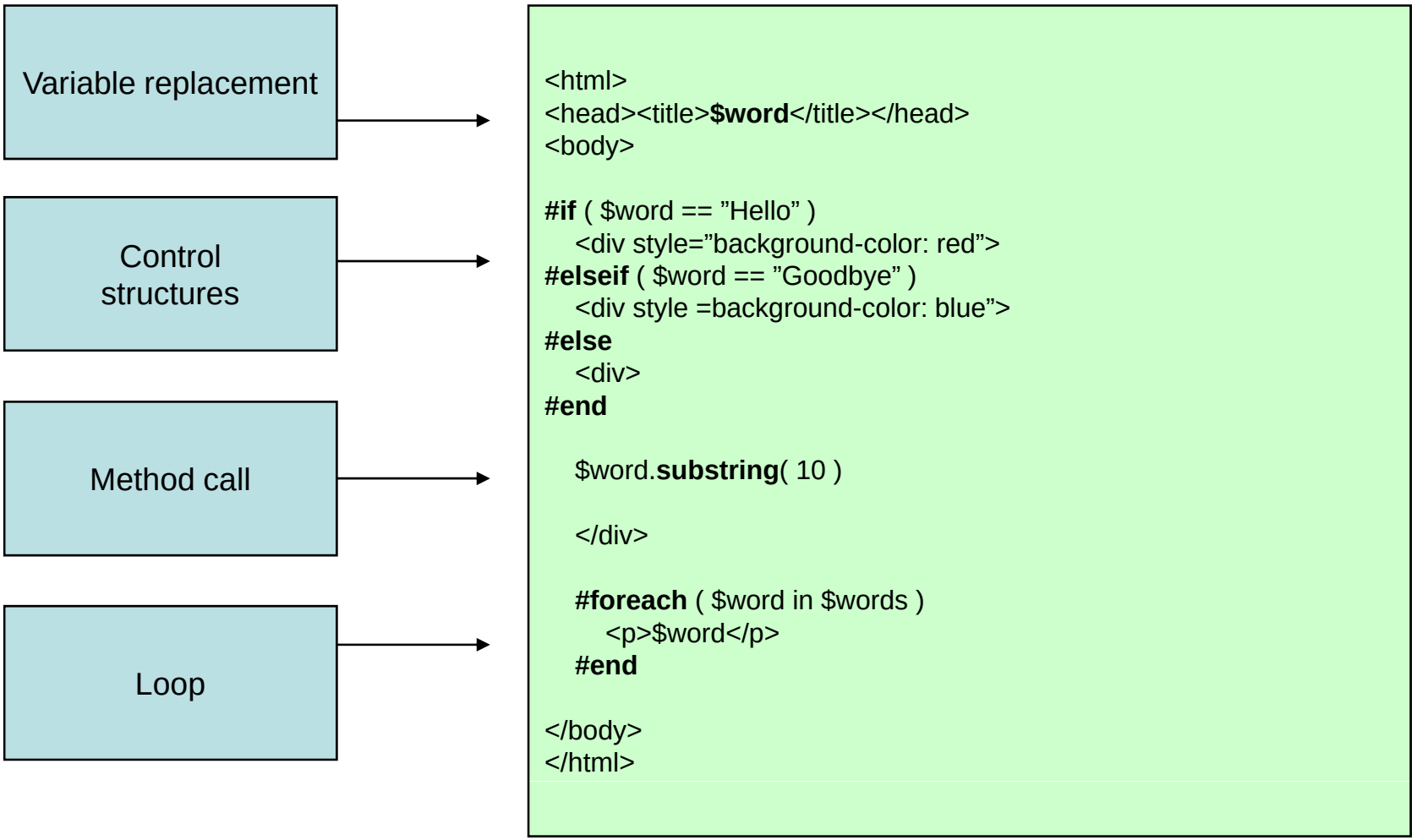
- Struts 2 integrates with the Spring IoC container
- The class property in action mappings refers to Spring bean identifiers instead of classes



Velocity

- Velocity is a *template* language
  - *Template:* basis for documents with similar structure
  - *Template language:* format defining where variables should be replaced in a document
- Features include:
  - Variable replacement
  - Simple control structures
  - Method invocation
- Velocity result is included in struts-default.xml
- Velocity is a *runtime* language
  - Fast
  - Error prone

Velocity


Variable replacement
Control structures
Method call
Loop
<html>
<head><title>$word</title></head>
<body>
#if ($word == ”Hello”)
<div style=”background-color: red”>
#elseif ($word == ”Goodbye”)
<div style =background-color: blue”>
#else

#end
$word.substring(10)

#foreach ($word in $words)
$word
#end
</body>
</html>

Struts 2 in DHIS 2

- Web commons project (dhis-web-commons)
  - Java code for widgets, security, portal
  - Interceptor, result configuration
  - Filters
  - Application logic interceptors
  - Custom results
- Web commons resources project (dhis-web-commons-resource)
  - Web resources like templates, javascripts, css

Web modules in DHIS 2

- Templates included in backbone template – main.vm
  - Static params in Struts 2 configuration for page and menu
- Must depend on dhis-web-commons and dhis-web-commons-resources
- Struts packages must
  - Include *dhis-web-commons.xml*
  - Extend *dhis-web-commons* package
  - Have the same package name as the artifact id
  - Have the same namespace as the artifact id
- Development tip: $ mvn jetty:run –war
  - Packages and deploys war file to Jetty for rapid development

Resources

- Brown, Davis, Stanlick: *Struts 2 in Action*
- Velocity user guide:
  - http://velocity.apache.org/engine/devel/user-guide.html
- Example code on course homepage