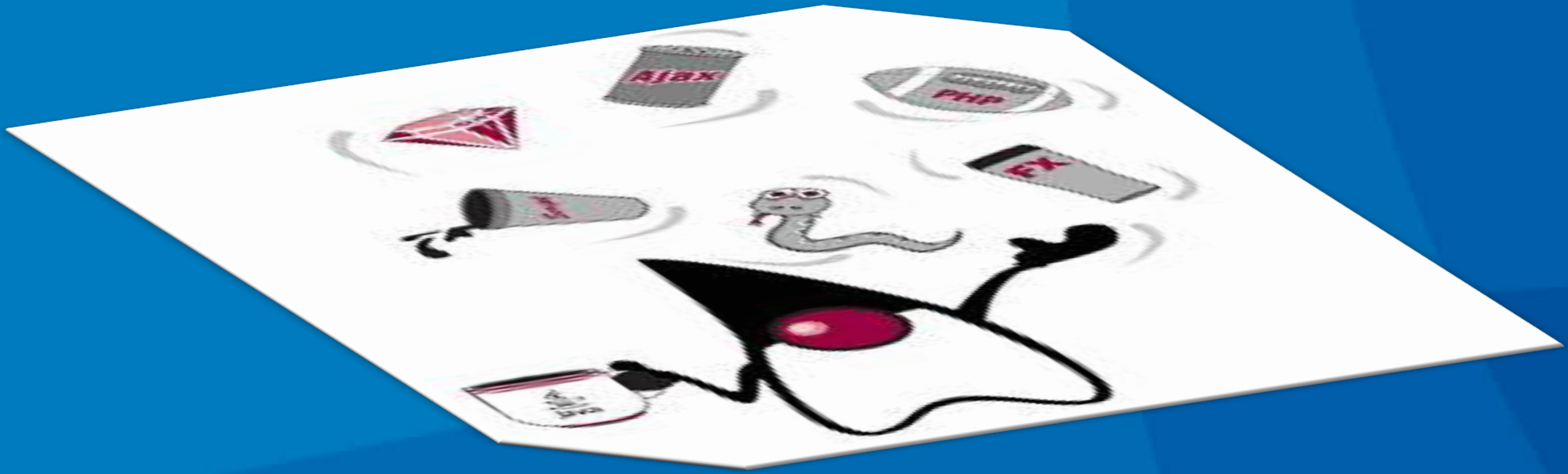
# Basic
# Java Web



## Course contents

- Web Basics
- The WebContainer

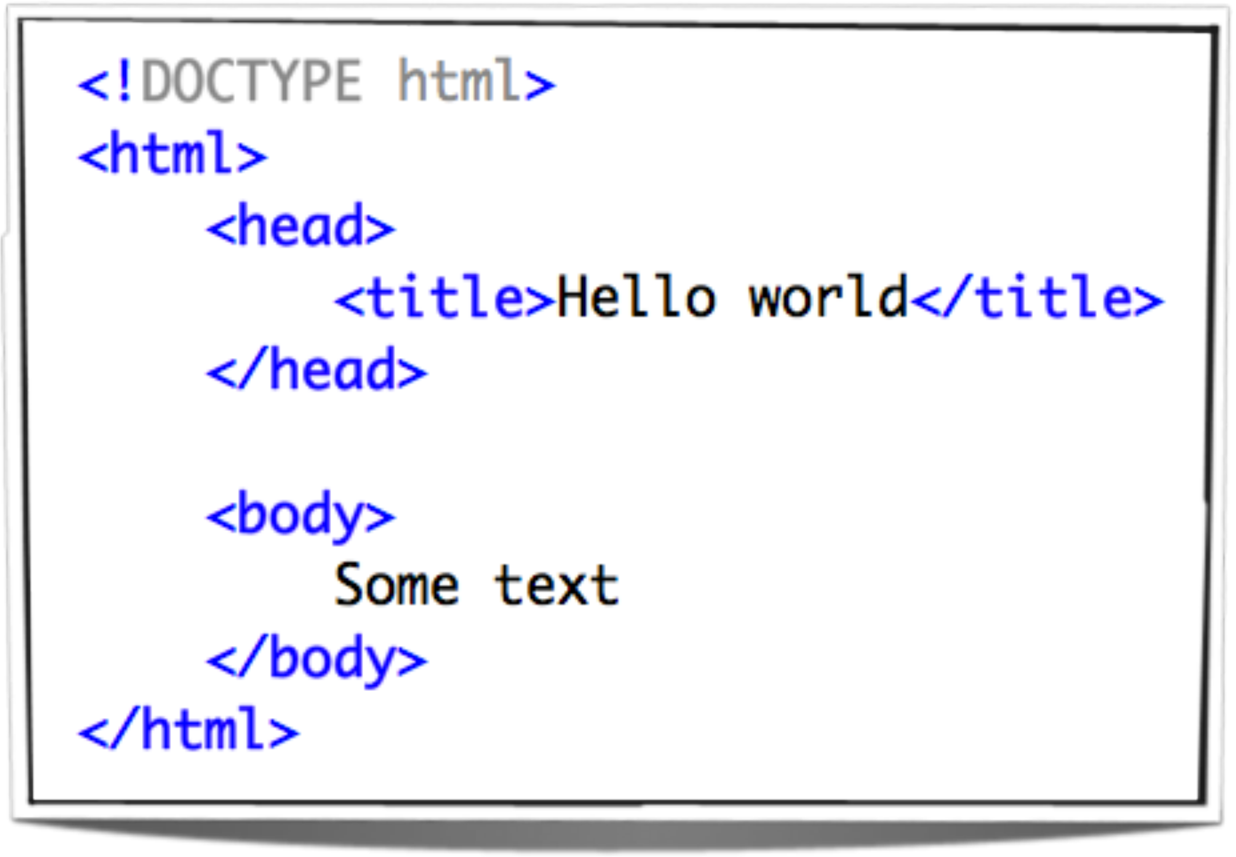
# Web basics

InfoSupport
Solid Innovator

## Related Background

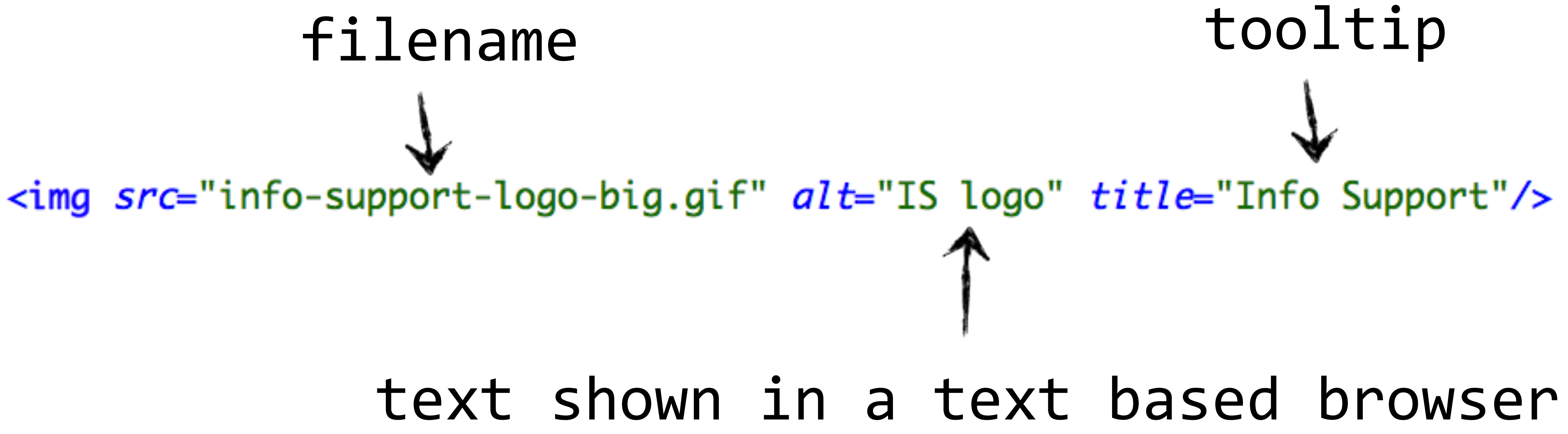
- HTML
  - HyperText Markup Language
- HTTP
  - HyperText Transfer Protocol

# Basic HTML elements



```
<!DOCTYPE html>
<html>
    <head>
        <title>Hello world</title>
    </head>

    <body>
        Some text
    </body>
</html>
```

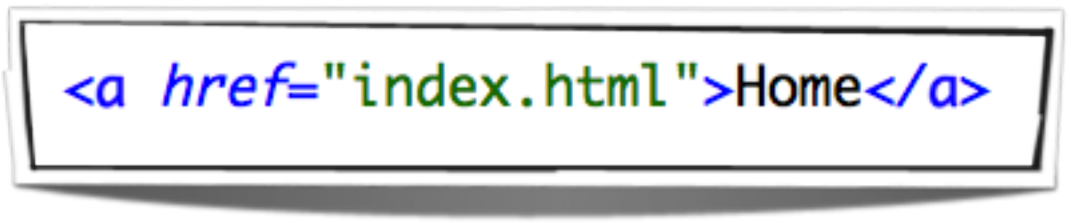
# Images



```html
<img src="info-support-logo-big.gif" alt="IS logo" title="Info Support"/>
```

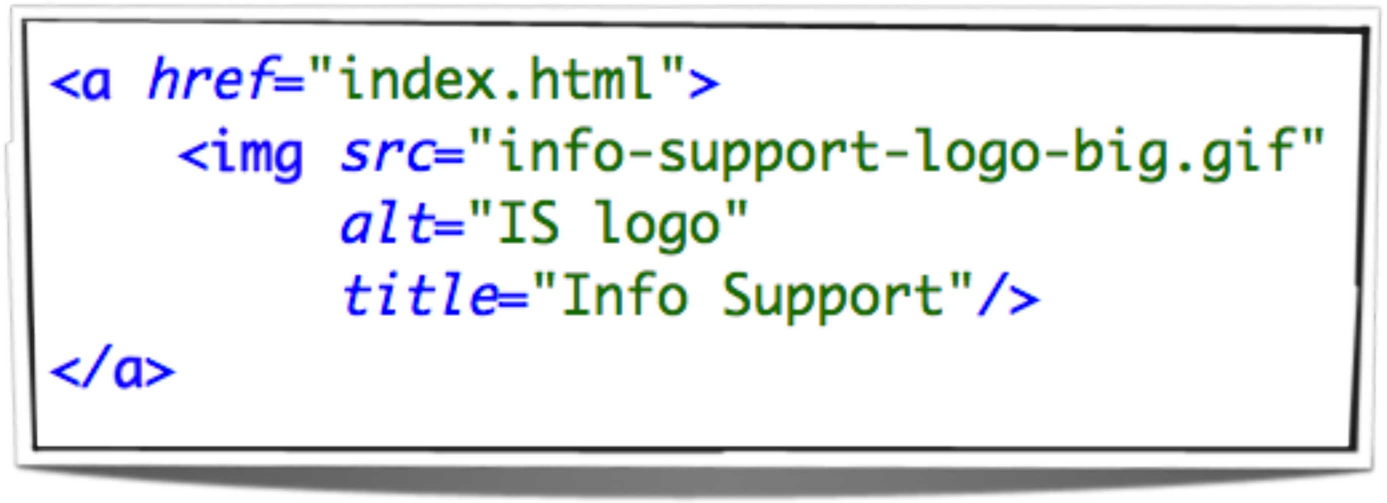
# Links

Text link



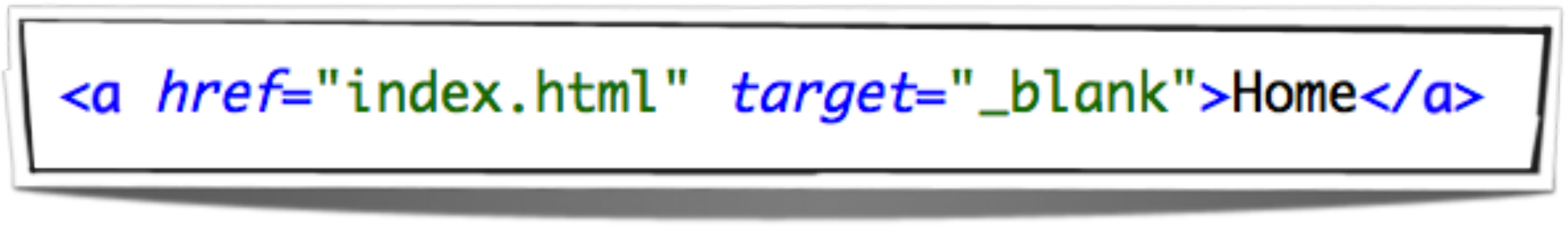
```html
<a href="index.html">Home</a>
```

Clickable image



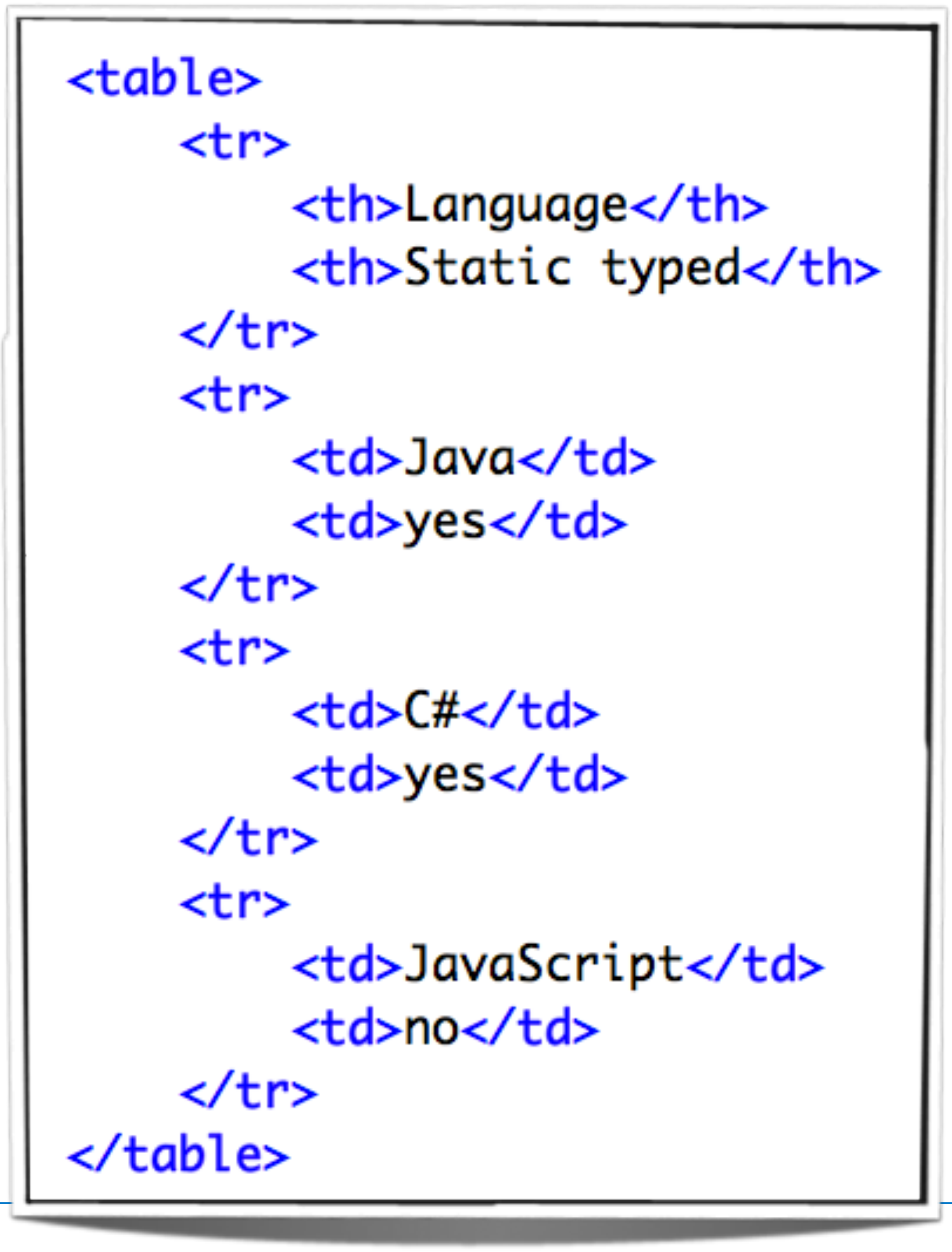
```html
<a href="index.html">
    <img src="info-support-logo-big.gif"
         alt="IS logo"
         title="Info Support"/>
</a>
```

Open in new window



```html
<a href="index.html" target="_blank">Home</a>
```

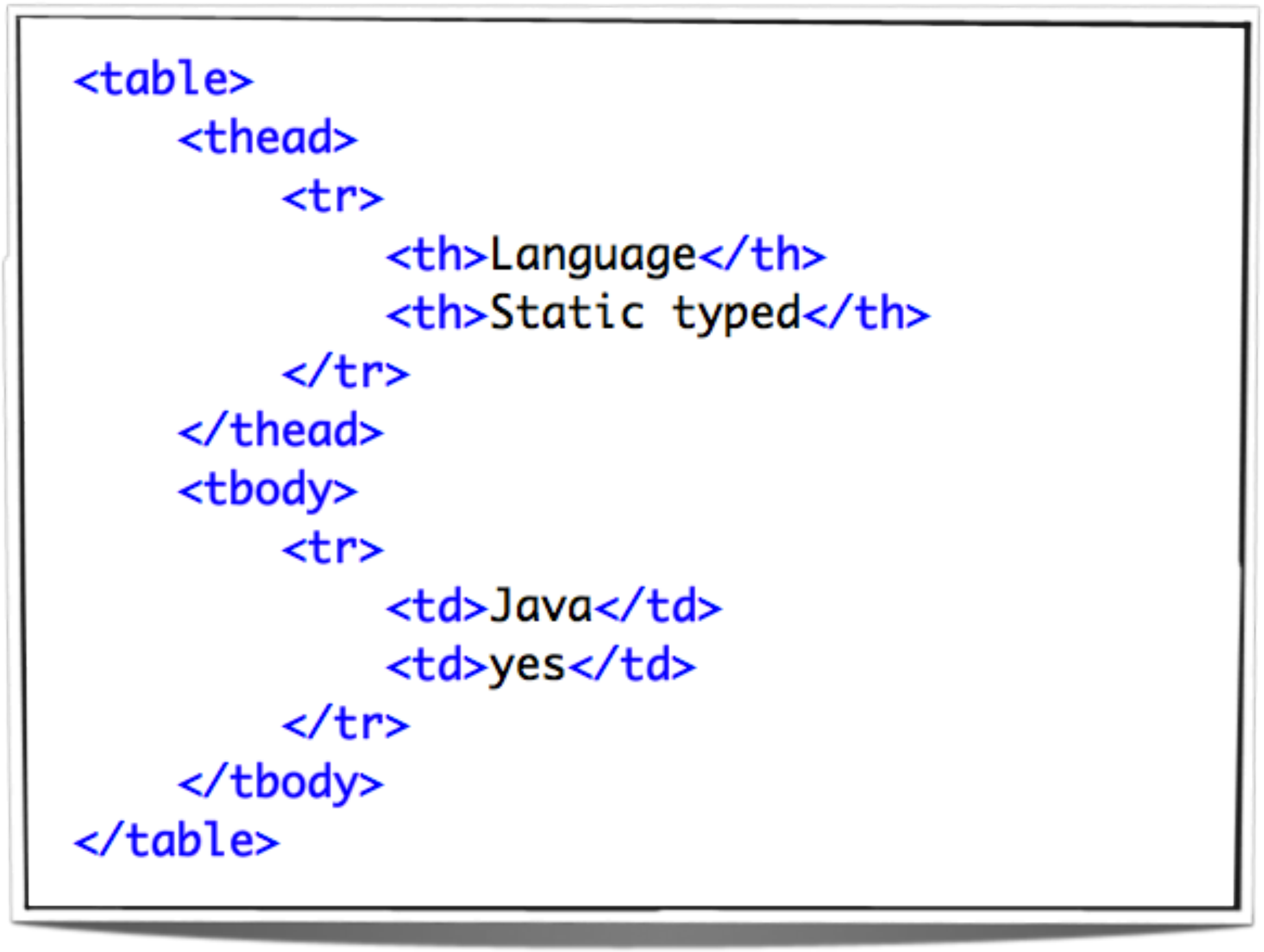
## Tables



```html
<table>
    <tr>
        <th>Language</th>
        <th>Static typed</th>
    </tr>
    <tr>
        <td>Java</td>
        <td>yes</td>
    </tr>
    <tr>
        <td>C#</td>
        <td>yes</td>
    </tr>
    <tr>
        <td>JavaScript</td>
        <td>no</td>
    </tr>
</table>
```

# thead / tbody

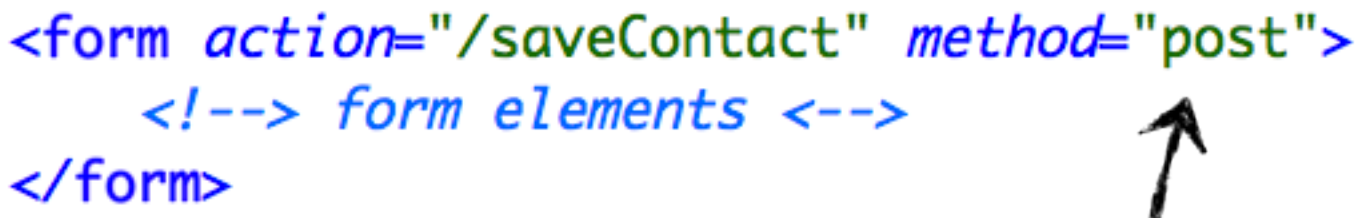
- Not required, but adds semantics to the page



```
<table>
    <thead>
        <tr>
            <th>Language</th>
            <th>Static typed</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Java</td>
            <td>yes</td>
        </tr>
    </tbody>
</table>
```

# Forms

url to post to

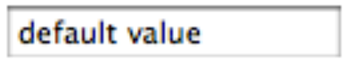


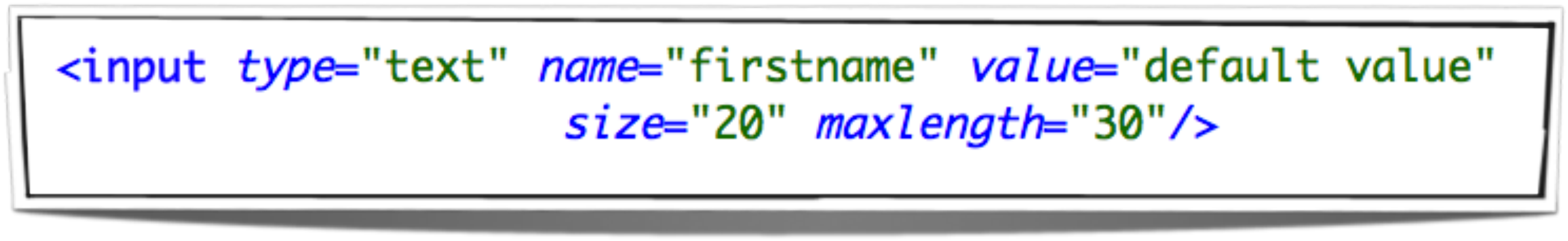
```
<form action="/saveContact" method="post">
    <!--> form elements <-->
</form>
```

should always be post
(but defaults to get)

# Form elements

Text input





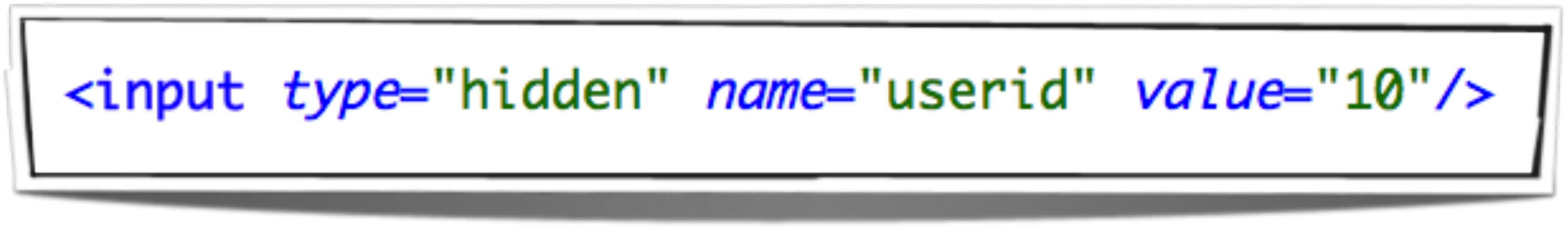
```
<input type="text" name="firstname" value="default value"
                    size="20" maxlength="30"/>
```

value will be posted

# Form elements

Hidden input

Doesn't render to a user.
Often used by server side frameworks

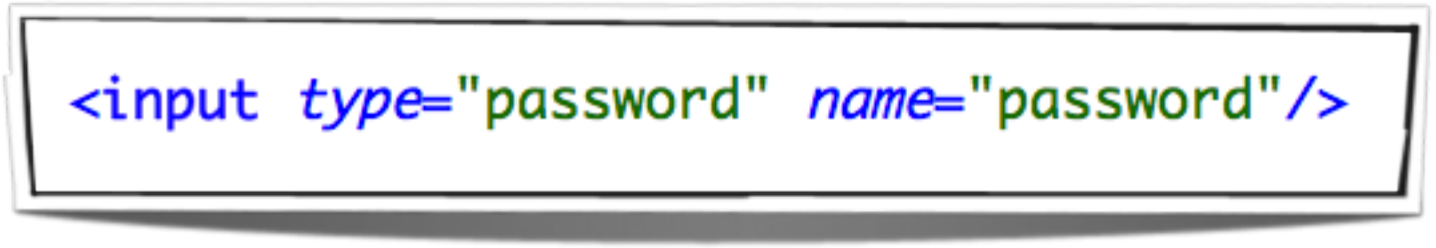


```
<input type="hidden" name="userid" value="10"/>
```

value will be posted

## Form elements

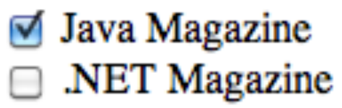
Password input



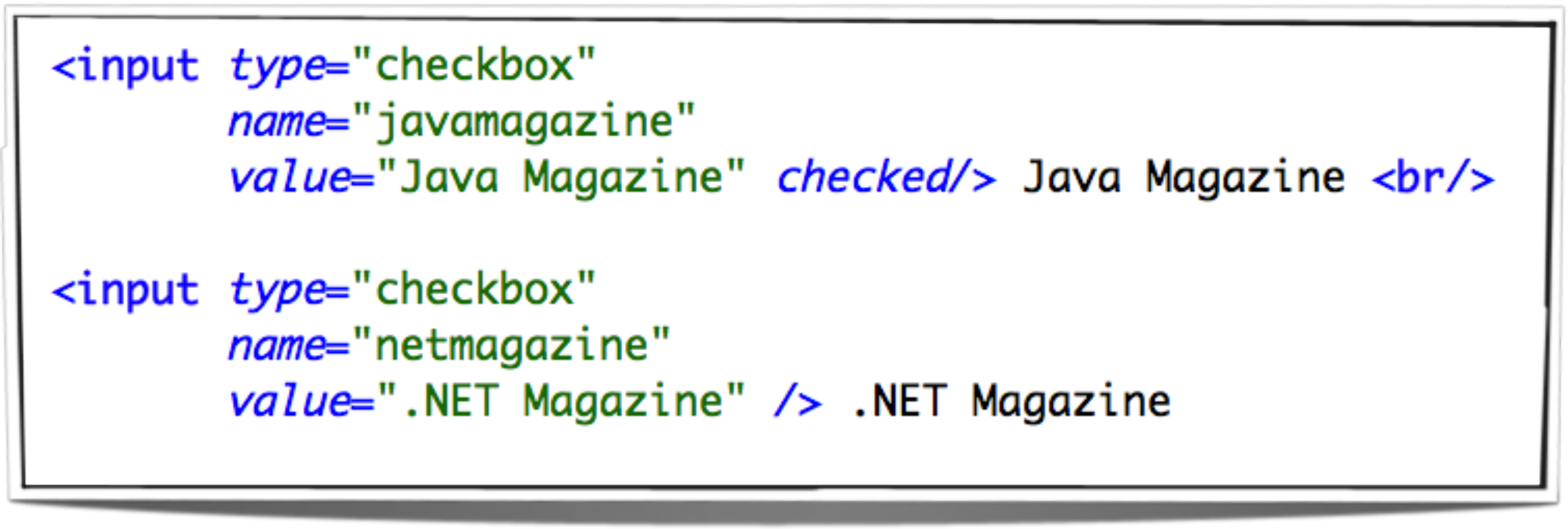
```
<input type="password" name="password"/>
```

value will be posted

# Form elements

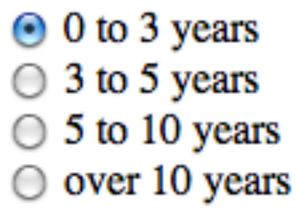


checkboxes

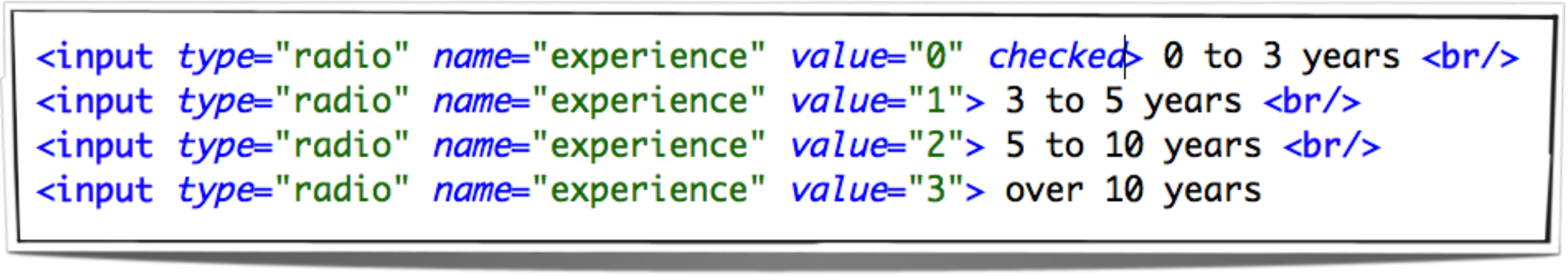


```
<input type="checkbox"
       name="javamagazine"
       value="Java Magazine" checked/> Java Magazine <br/>

<input type="checkbox"
       name="netmagazine"
       value=".NET Magazine" /> .NET Magazine
```

## Form elements

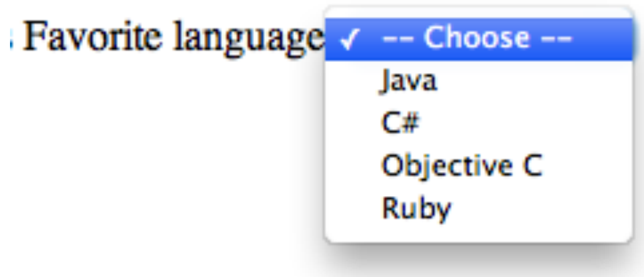

◉ 0 to 3 years
○ 3 to 5 years
○ 5 to 10 years
○ over 10 years

radio buttons

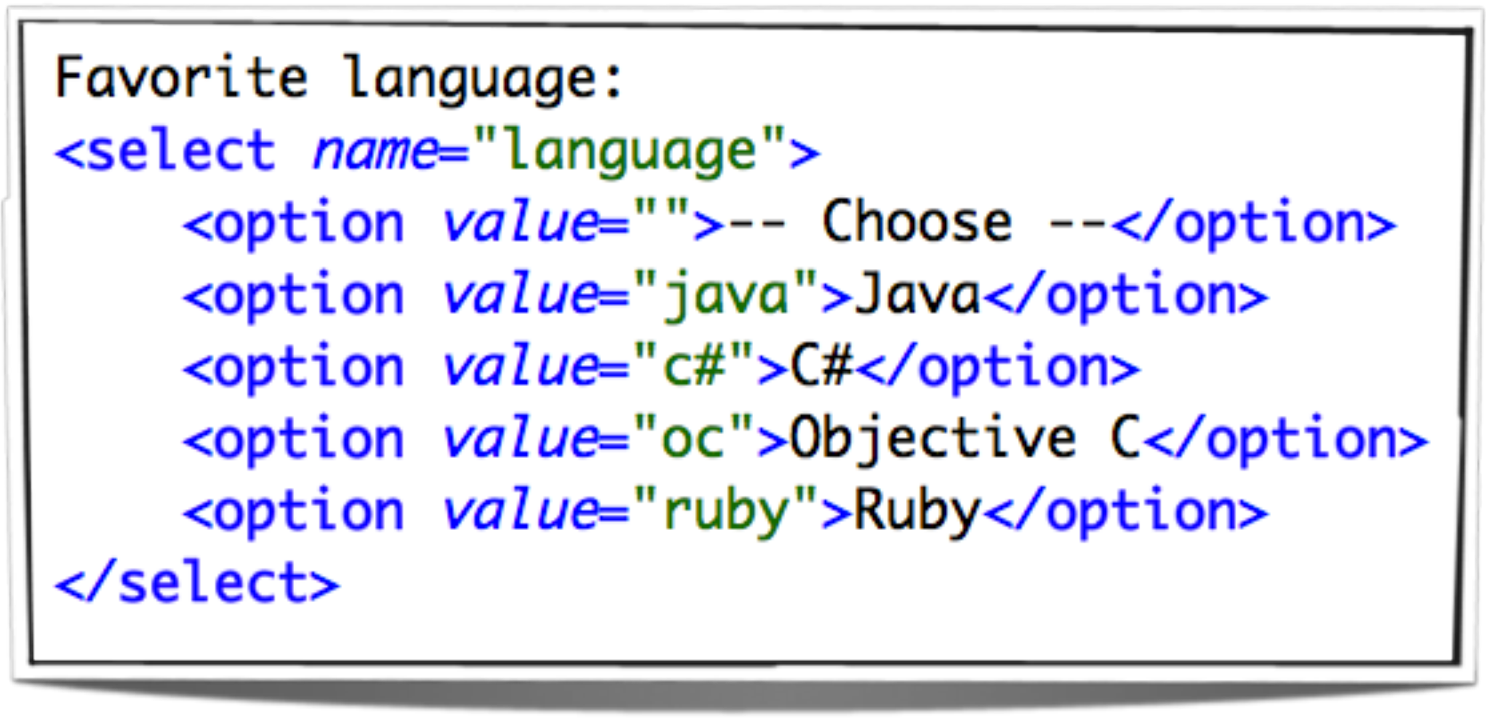


```
<input type="radio" name="experience" value="0" checked> 0 to 3 years <br/>
<input type="radio" name="experience" value="1"> 3 to 5 years <br/>
<input type="radio" name="experience" value="2"> 5 to 10 years <br/>
<input type="radio" name="experience" value="3"> over 10 years
```

# Form elements



select menu



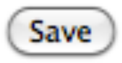
```
Favorite language:
<select name="language">
    <option value="">-- Choose --</option>
    <option value="java">Java</option>
    <option value="c#">C#</option>
    <option value="oc">Objective C</option>
    <option value="ruby">Ruby</option>
</select>
```

## Submitting a form

label of the button

$\downarrow$

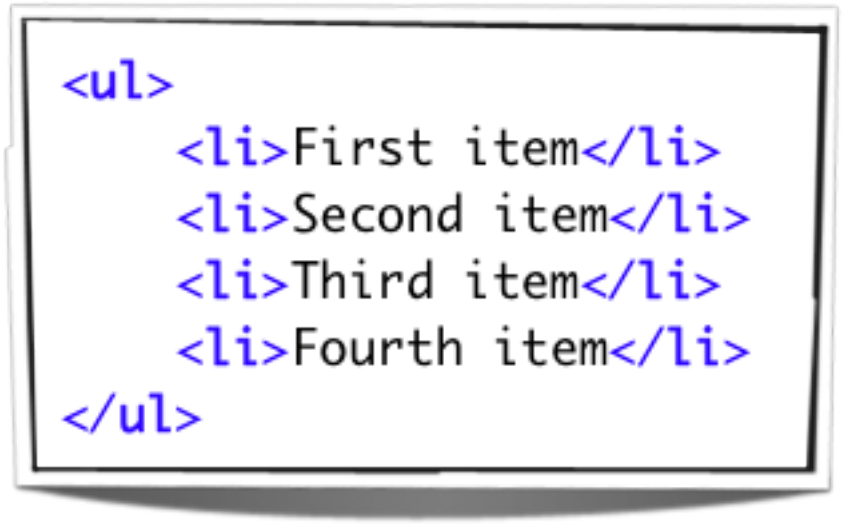
```
<input type="submit" value="Save"/>
```



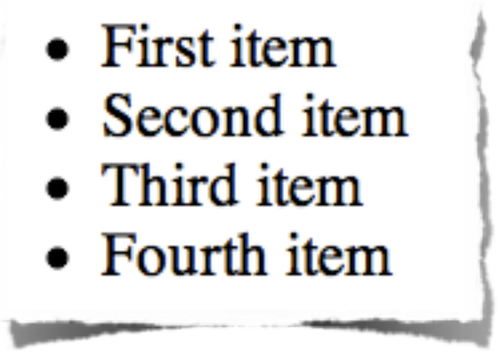
## Lists

- Unordered lists <ul>
- Ordered lists <ol>
- Definition lists <dl>

## Unordered list

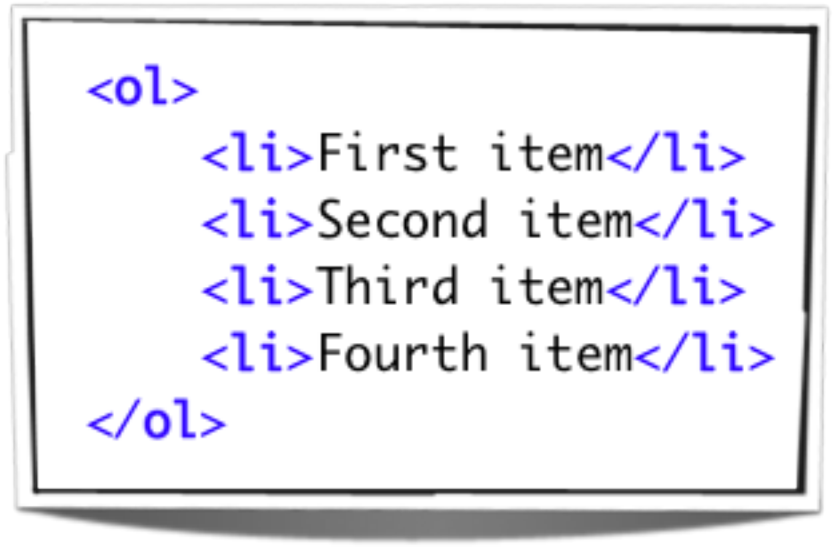


```
<ul>
    <li>First item</li>
    <li>Second item</li>
    <li>Third item</li>
    <li>Fourth item</li>
</ul>
```

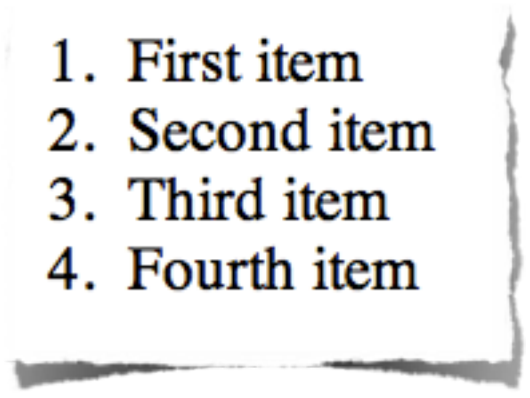


- First item
- Second item
- Third item
- Fourth item

# Ordered list

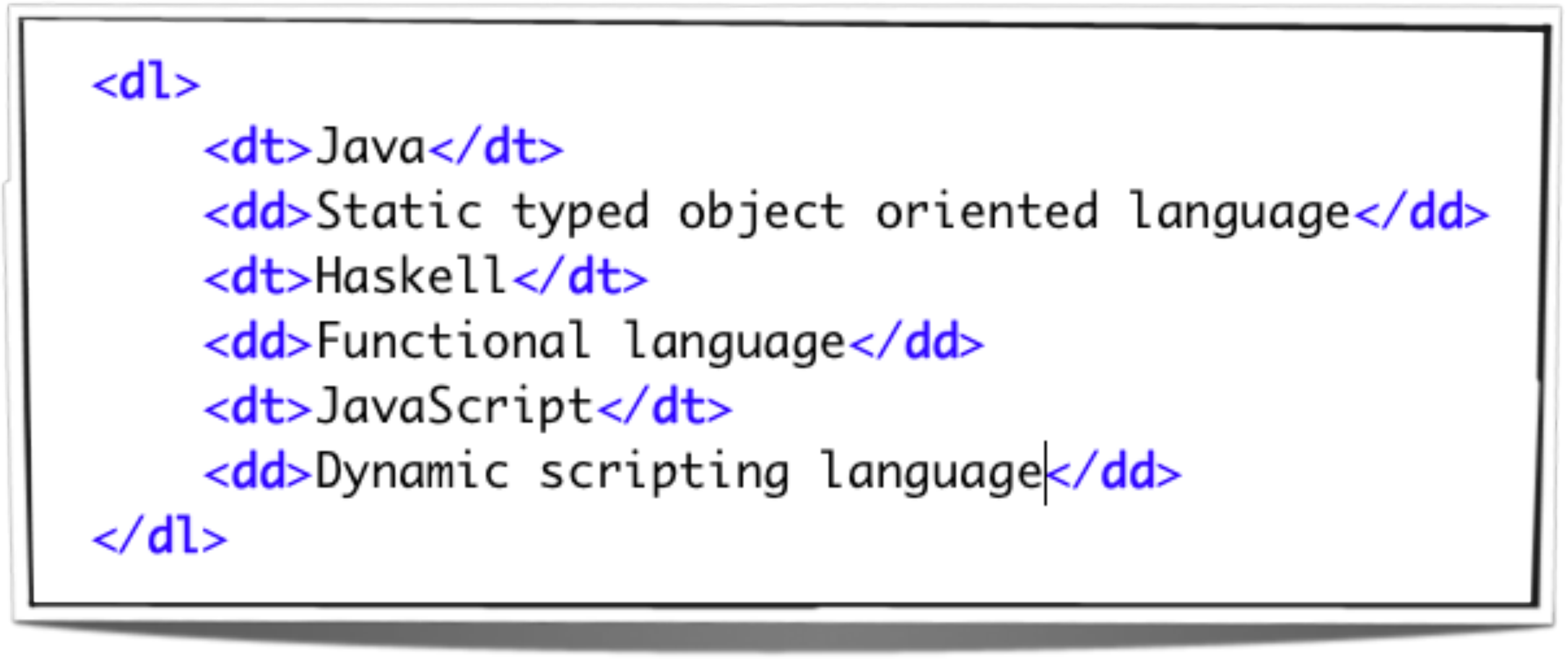


```
<ol>
    <li>First item</li>
    <li>Second item</li>
    <li>Third item</li>
    <li>Fourth item</li>
</ol>
```

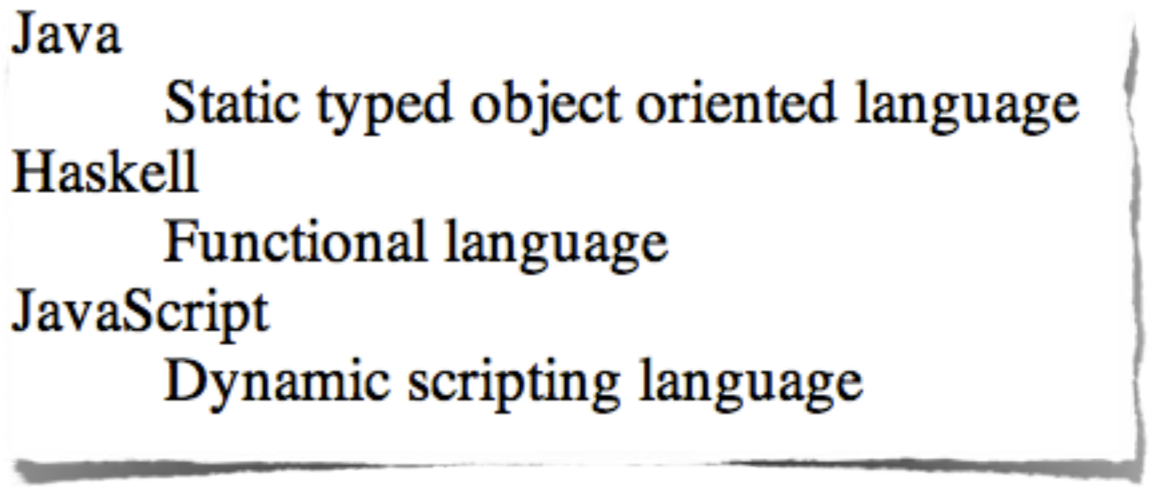


1. First item
2. Second item
3. Third item
4. Fourth item

# Definition list



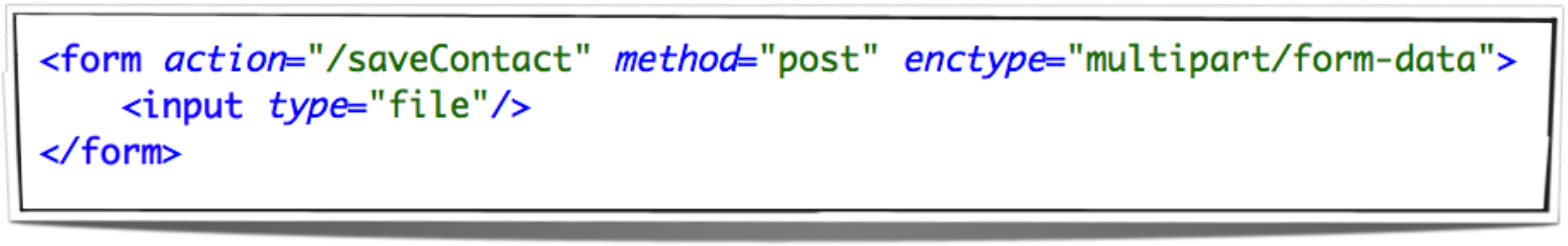
```html
<dl>
    <dt>Java</dt>
    <dd>Static typed object oriented language</dd>
    <dt>Haskell</dt>
    <dd>Functional language</dd>
    <dt>JavaScript</dt>
    <dd>Dynamic scripting language</dd>
</dl>
```



```
Java
        Static typed object oriented language
Haskell
        Functional language
JavaScript
        Dynamic scripting language
```

## Form enctype

- Set enctype to support file upload
- Defaults to application/x-www-form-urlencoded



```
<form action="/saveContact" method="post" enctype="multipart/form-data">
    <input type="file"/>
</form>
```

## HyperText Transfer Protocol

- HTTP functions as a request-response protocol in the client-server computing model
  - Example: a browser acts as a client also commonly referred to as User Agent
  - an application running on a computer hosting a web site functions as a server
- The client submits an HTTP request message to the server
- The server returns an HTTP response message to the client

## HTTP Request methods

| HTTP Method | Description |
|---|---|
| **GET** | Requests a representation of the specified resource. |
| **POST** | Submits data to be processed to the identified resource |
| **PUT** | Uploads a representation of the specified resource |
| **DELETE** | Deletes the specified resource |
| **HEAD** | Requests meta-information written in response headers |

- Note: a browser supports only the GET and POST method

## The GET & POST

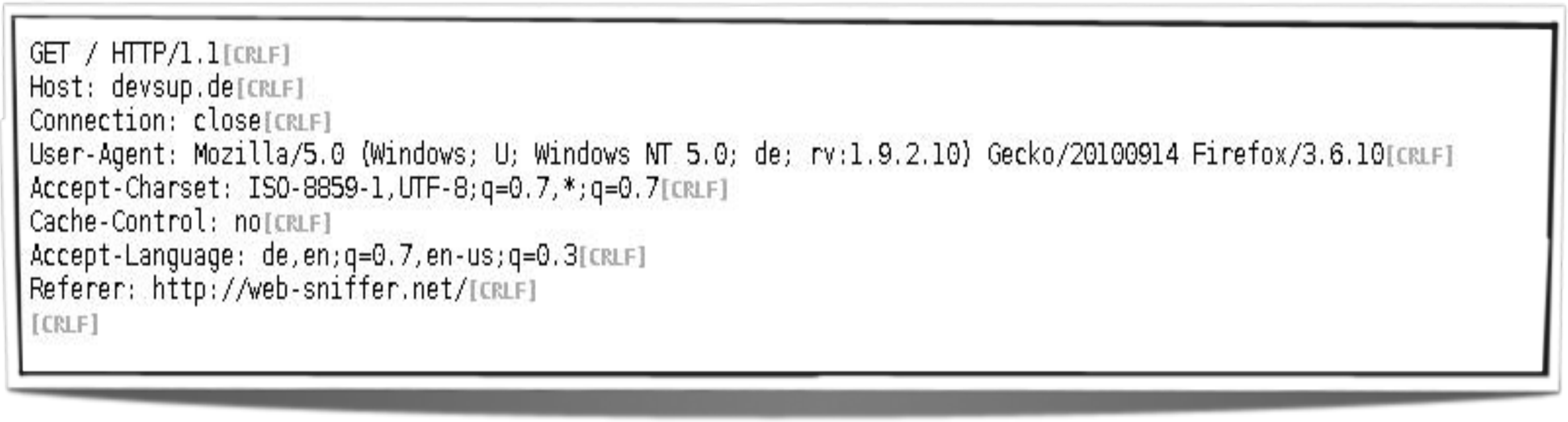
- Requests using GET  "SHOULD NOT have the significance of taking an action other than RETRIEVAL
  - In other words, they should not have side effects
  - the handling of the GET request by the server is not technically limited in any way
  - therefore it is the responsibility of the programmer to make the GET safe.
- Request using POST is intended for actions that may cause side effects

# Request Headers

- Metadata is sent to the server in the form of headers.
- Some typical headers are:

| header | meaning | example |
|---|---|---|
| Accept | Content-Types that are acceptable for the browser | `Accept: text/plain` |
| Accept-Charset | Character sets that are acceptable | `Accept-Charset: utf-8` |
| Content-Length | The length of the request body in bytes | `Content-Length: 348` |
| Content-Type | The mime type of the body of the request (used with POST and PUT requests) | `Content-Type: application/x-www-form-urlencoded` |
| Host | The domain name of the server for virtual hosting | |
| User-Agent | The user agent string | `User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)` |

# Example Request



```
GET / HTTP/1.1[CRLF]
Host: devsup.de[CRLF]
Connection: close[CRLF]
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; de; rv:1.9.2.10) Gecko/20100914 Firefox/3.6.10[CRLF]
Accept-Charset: ISO-8859-1,UTF-8;q=0.7,*;q=0.7[CRLF]
Cache-Control: no[CRLF]
Accept-Language: de,en;q=0.7,en-us;q=0.3[CRLF]
Referer: http://web-sniffer.net/[CRLF]
[CRLF]
```

## Status codes

| Status code | value |
|---|---|
| `200 OK` | Standard response for successful HTTP requests |
| `303 See Other` | Redirect web applications to a new uri |
| `400 Bad Request` | The request cannot be fulfilled due to bad syntax |
| `403 Forbidden` | The request was a legal request, but the server is refusing to respond to it |
| `404  Not Found` | The requested resource could not be found but may be available again in the future |
| `500 Internal Server Error` | A generic error message, given when no more specific message is suitable |

# The Web Container

infoSupport
Solid Innovator

## Problem (1)

- We need a process on the server side to act on the received messages
  - to save submitted data as a Contact entity in the database
  - to read on uploaded file and act appropriately depending on the content
  - etc. etc. etc...

## Problem (2)

- When we like to implement this process in Java our building blocks are classes on which we implement methods
- The HTTP protocol only lets us direct to resources which are bound to an URL
- In Java the way to execute some functionality is by calling a method on an object
- How to map the  request for a resource in HTTP to a method call in Java?

## Intro Web Container

- A lot of plumbing has to be done to bridge the gap between the HTTP and Java world
- This plumbing is delegated to a specialized process, the so called Web Container a.k.a. Web Server
- The Web Container receives the requests and translates it in a java accessible form
- The Web Container also provides a java infrastructure which can be used on the java side to tackle common web problems

## Popular Java Web Containers

Full JEE compliant app servers:

Glassfish

JBoss™

ORACLE WebLogic

IBM WebSphere Application Server Version 1

APACHE GERONIMO

Web profile vendors:

Apache Tomcat

jetty://

## Web Applications

- Our Web application actually runs inside the Web Container, it is not a standalone application
  - it becomes a “part” of the container
  - it supplies the “customized” behavior which represents the business functionality
  - it must conform to predefined standards to be able to interact with the container
- This standard of how to build web applications is described in the Servlet Specification

## The Servlet Specification (1)

- The Servlet Specification describes how the web container will interact with our web application
- The Servlet Specification is part of a much larger specification, the JEE specification, which describes how Java Enterprise Applications should be build in a standard, compatible way.

## The Servlet Specification (2)

- Other, typical sub specs from the JEE spec are (among many others):
  - JDBC
  - JPA
  - EJB
- Note: in an earlier release the JEE spec was called the J2EE specification

## The Servlet

- The Servlet spec. dictates that a Java class that can be the target of a HTTP request must implement the Servlet interface
  - This java class is subsequently called a Servlet
  - It is also called a web component
  - It is also an example of a managed component,

## Servlet as Managed Component (1)

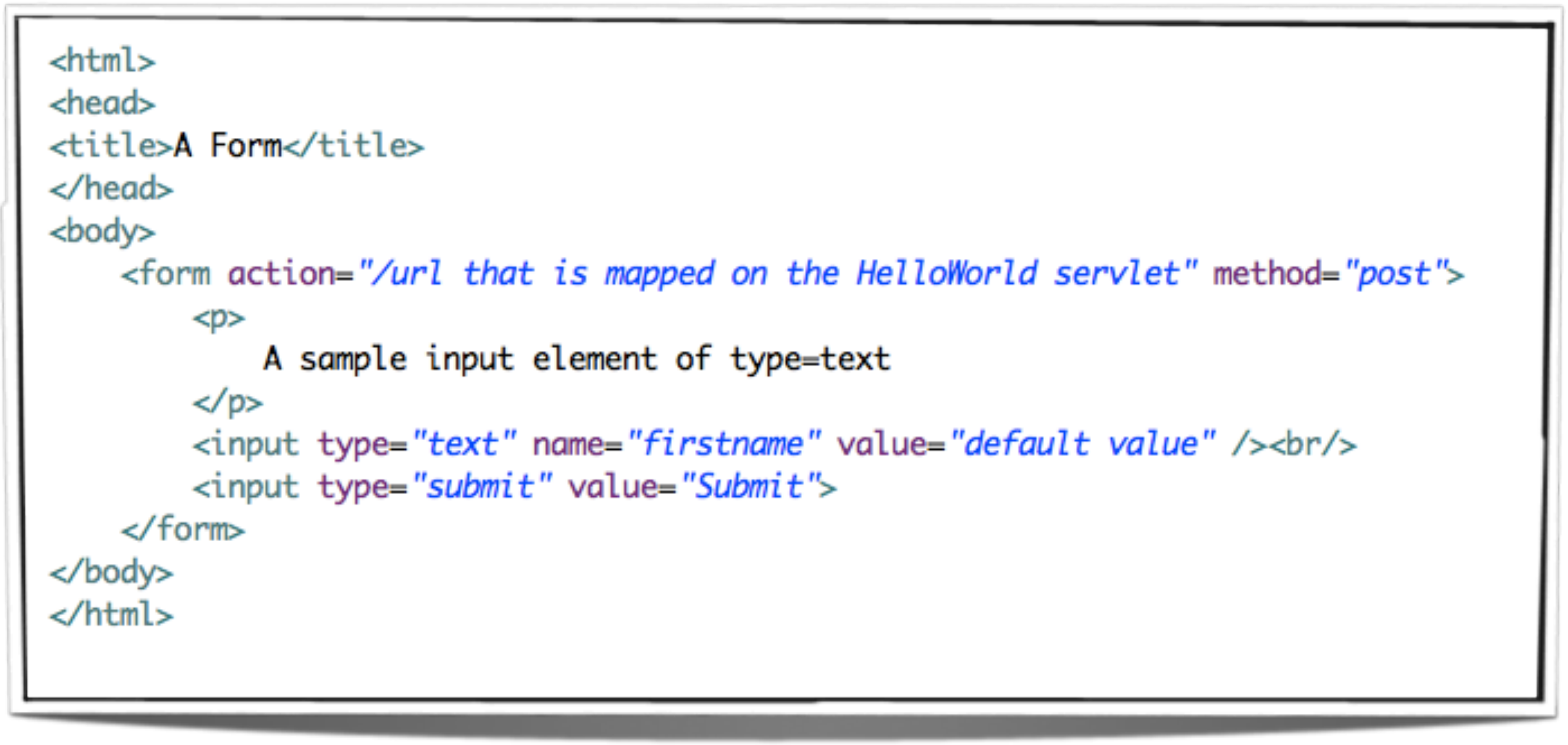
- A Servlet is a component managed by the web container
  - the container instantiates the Servlet
  - initializes it
  - supplies, if wanted, necessary dependencies
  - routes the http requests to the specified Servlet
  - destroys it

## Servlet as Managed Component (2)

- The container also hides the HTTP details behind a convenient O.O. model
  - the details of the HTTP request are translated to a Request object
  - the data written in the Java Response object by our web application are automatically translated into a HTTP Response by the container

## Example

- We want to add a Servlet which reads the input parameter `firstname` from the submitted form and prints it to the server console

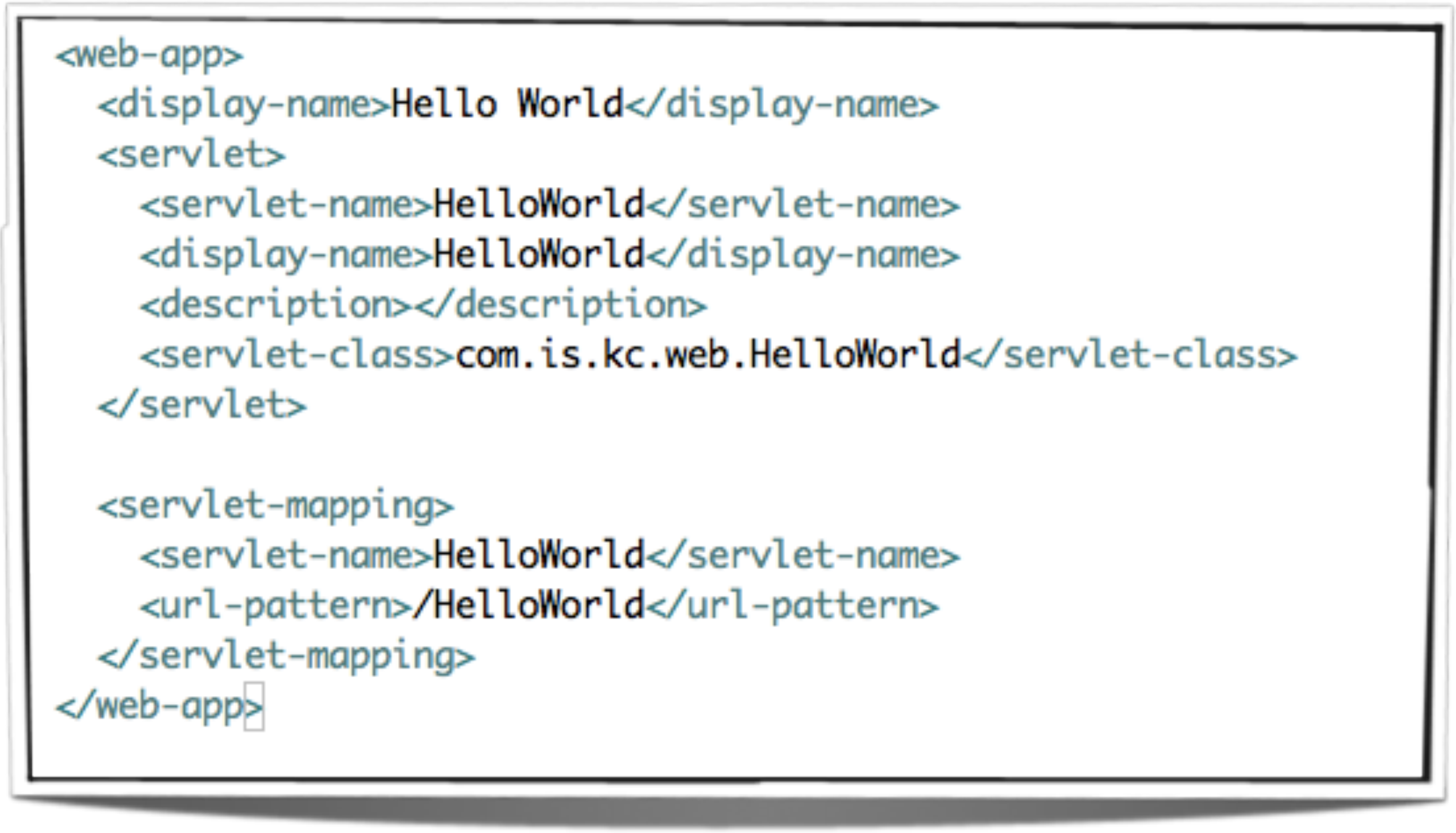


```
<html>
<head>
<title>A Form</title>
</head>
<body>
    <form action="/url that is mapped on the HelloWorld servlet" method="post">
        <p>
            A sample input element of type=text
        </p>
        <input type="text" name="firstname" value="default value" /><br/>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

## web.xml

- Register the servlet class with the container
- Inform the container which url's are handled by this servlet

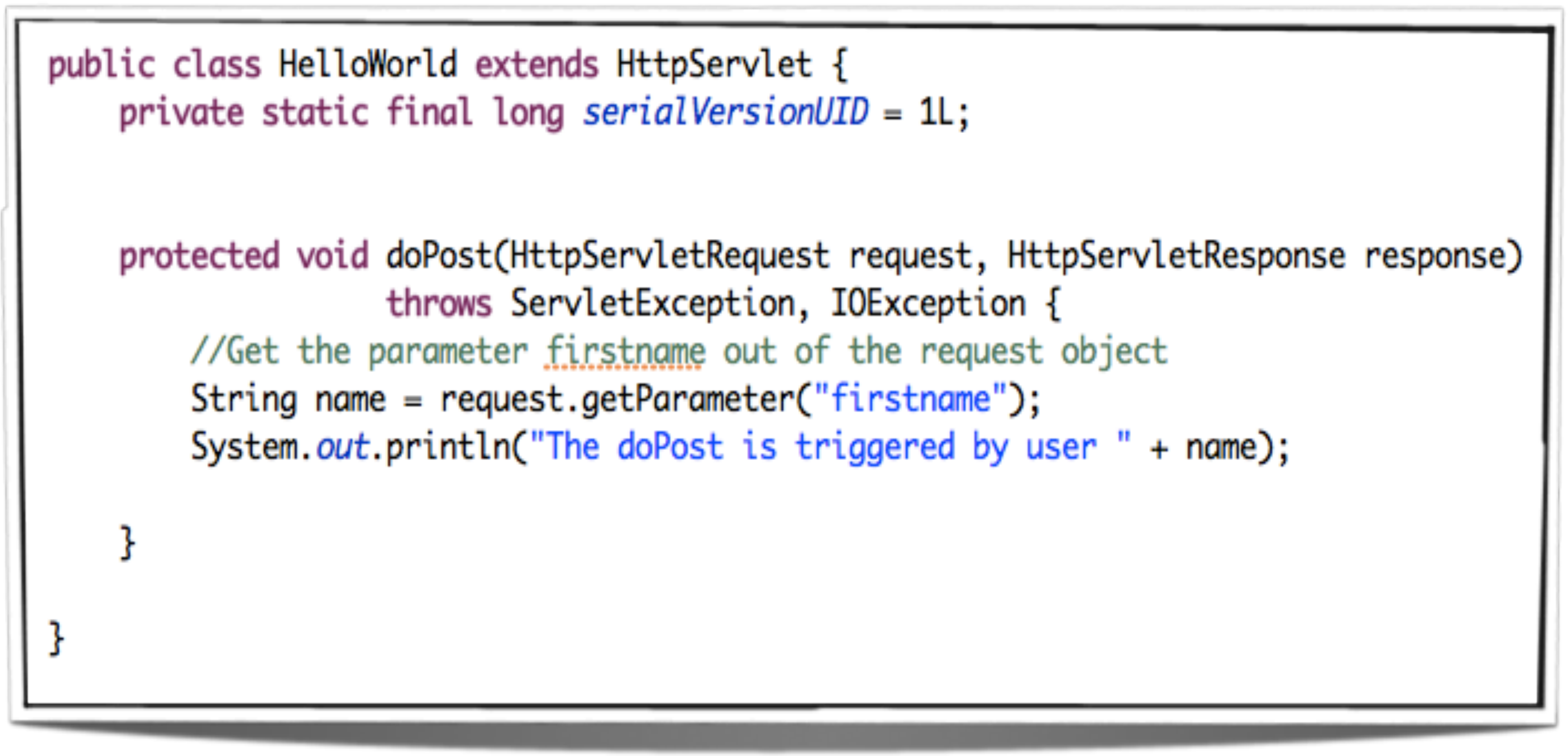
The configuration file of the web application



```
<web-app>
  <display-name>Hello World</display-name>
  <servlet>
    <servlet-name>HelloWorld</servlet-name>
    <display-name>HelloWorld</display-name>
    <description></description>
    <servlet-class>com.is.kc.web.HelloWorld</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/HelloWorld</url-pattern>
  </servlet-mapping>
</web-app>
```

Note: the "/" in the url-pattern /HelloWorld refers to the root of the web application

## Example Servlet

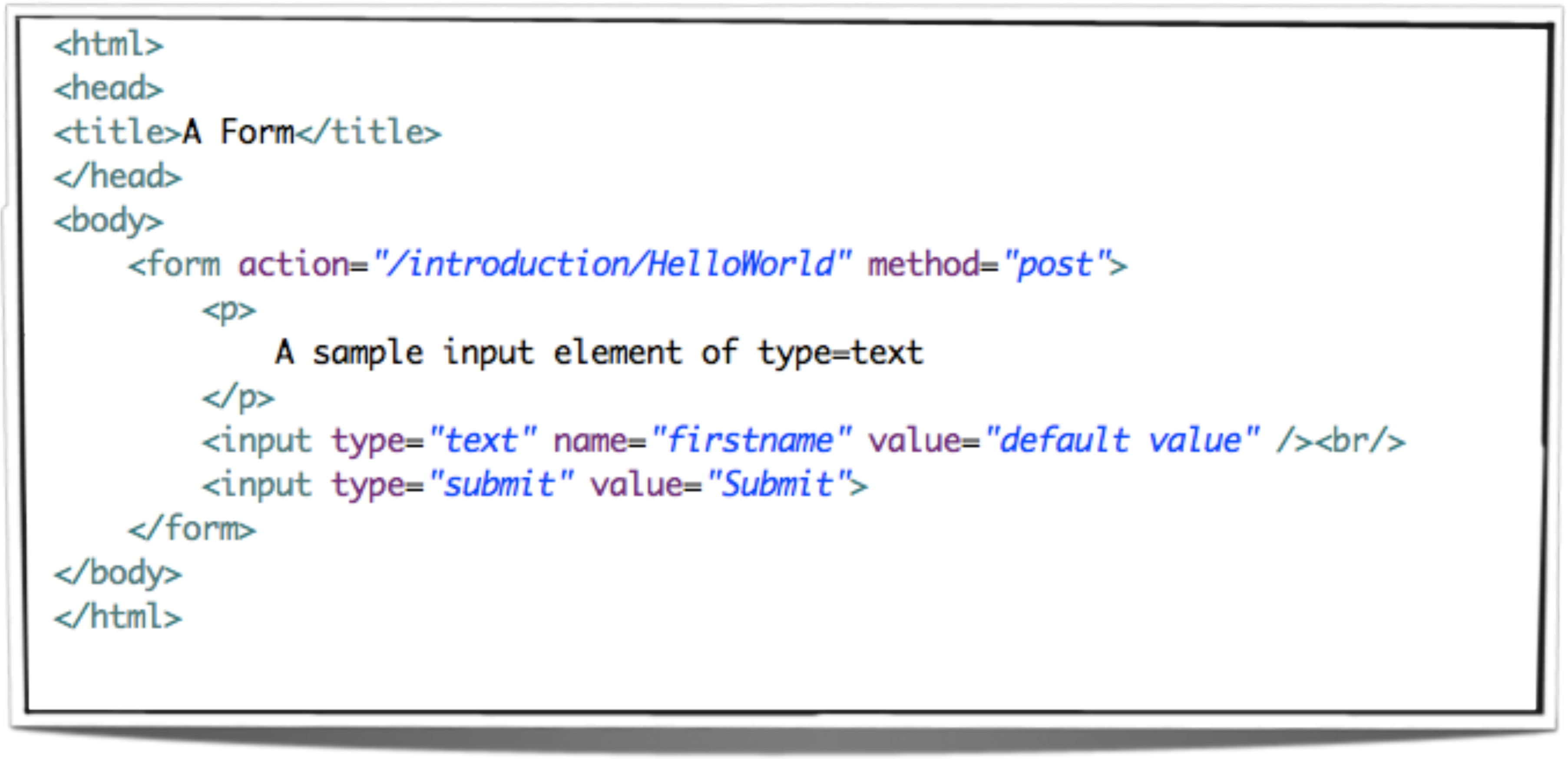


```
public class HelloWorld extends HttpServlet {
    private static final long serialVersionUID = 1L;


    protected void doPost(HttpServletRequest request, HttpServletResponse response)
                   throws ServletException, IOException {
        //Get the parameter firstname out of the request object
        String name = request.getParameter("firstname");
        System.out.println("The doPost is triggered by user " + name);

    }

}
```

- Details see next slide

## Example Servlet

- The HelloWorld Servlet extends HttpServlet
- The container will call the doPost method on this servlet when a post Request is made for the mapped url
- The HttpServletRequest object is supplied by the container to make request details available to the doPost method
- The HttpServletResponse object can be used to store all relevant data which must be sent back to the client

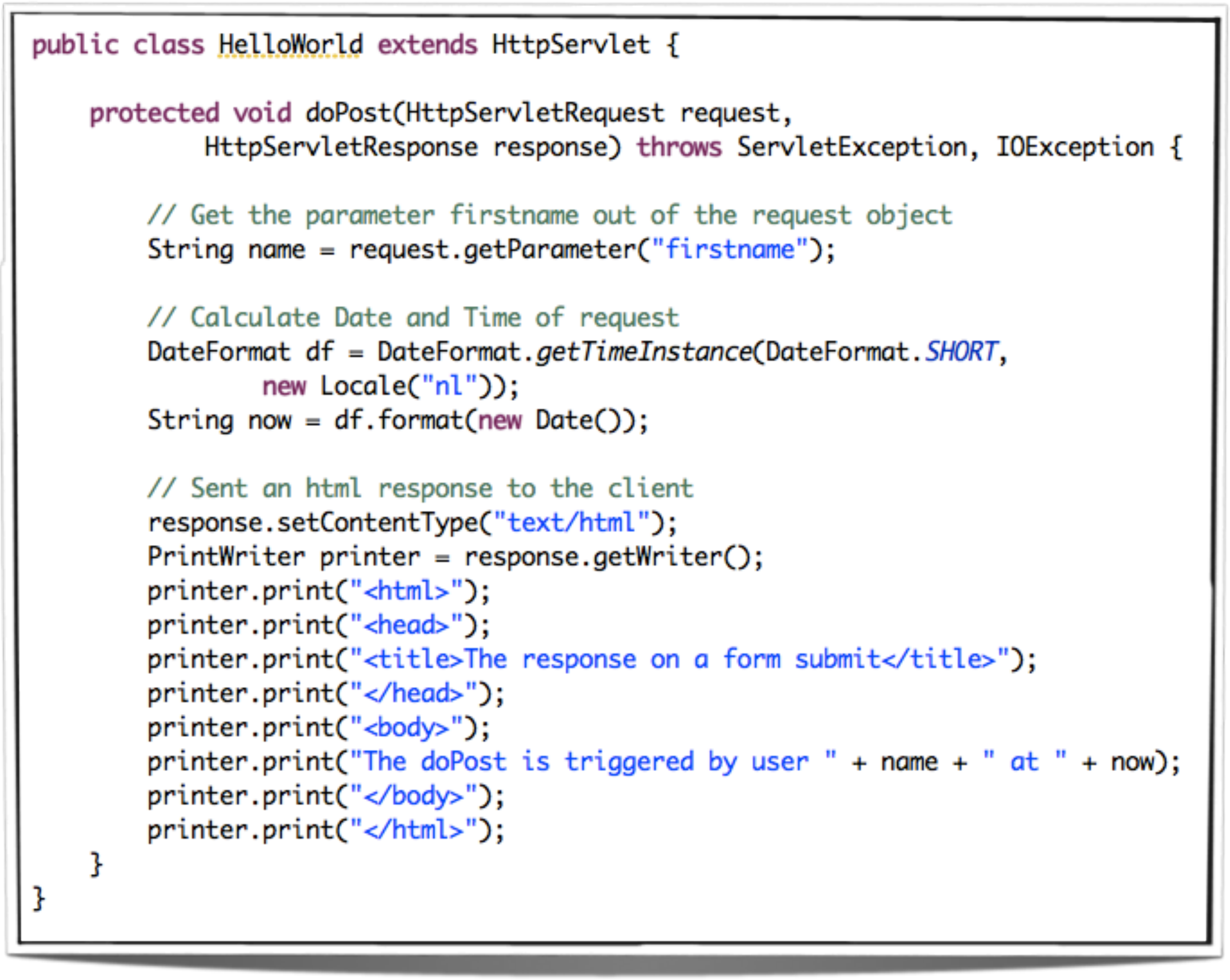


```
<html>
<head>
<title>A Form</title>
</head>
<body>
    <form action="/introduction/HelloWorld" method="post">
        <p>
            A sample input element of type=text
        </p>
        <input type="text" name="firstname" value="default value" /><br/>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

## Change the action attribute

- Note: an absolute path in the action attribute must contain the name of the web application (introduction)
- Note: the url that appears in the address bar when submitting the form
  - http://localhost:8080/introduction/HelloWorld
  - the server has prefixed it with the protocol server name and port
  - the data (the firstname submitted) is not shown in the url because it is a post

# Build a HTML response



```
public class HelloWorld extends HttpServlet {

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {

        // Get the parameter firstname out of the request object
        String name = request.getParameter("firstname");

        // Calculate Date and Time of request
        DateFormat df = DateFormat.getTimeInstance(DateFormat.SHORT,
                new Locale("nl"));
        String now = df.format(new Date());

        // Sent an html response to the client
        response.setContentType("text/html");
        PrintWriter printer = response.getWriter();
        printer.print("<html>");
        printer.print("<head>");
        printer.print("<title>The response on a form submit</title>");
        printer.print("</head>");
        printer.print("<body>");
        printer.print("The doPost is triggered by user " + name + " at " + now);
        printer.print("</body>");
        printer.print("</html>");
    }
}
```

## Problems with the Servlet only approach

- Statements aimed at describing the markup of the html page (presentation) are mixed with business code, this makes reading and maintaining the code more difficult

- Enter a web framework -> there are hundreds of them
- We will look at Spring Web MVC