**웹 개발**

**HTML**

**JSTL EL 사용을 위한 Import**

<%@ **taglib** prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"* %>

라이브러리가 필요하다.

주소 get -> param.변수명

Request.set 방식 -> 변수명 or request.변수명

**HTML에서 CSS 적용하기**

<**link** rel=*"stylesheet"* href=*"css/style.css"*>

<div style=”color: blue; background-color:yellow;”>와 <style></style>방법이 있다.

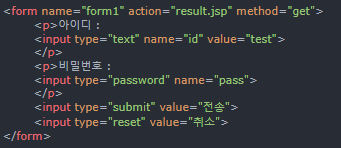
**HTML에서 script 적용하기**

<**script** src=*"파일경로/myscript2.js"* defer></**script**>

Defet는 async와 마찬가지로 비동기로 코드를 로딩해서 실행하지만, defer는 태그가 위치한 순서대로 실행 됩니다.

<script></script>와 <div onclick=”location.href=’../index.jsp’”></div>방식이 있다.

form태그 예제



**CSS**

**Css 주요 태그는 CSS3 Cheat Sheet을 참고할 것.**

<태그 속성(attribute)=”값”></태그>

**\* CSS 기본 셀렉터 (= 선택자)**

1) 태그 이름으로 선택

2) 아이디로 선택

3) 클래스로 선택

**\* 가상 요소**

- 가상 클래스가 실제로 존재하는 태그에 클래스 추가 없이 태그를 선택하는 것이라면,

가상 요소는 아예 실제로 존재하지 않는 요소를 선택할 때 사용한다.

- 가상 요소를 이용하면 특정 태그앞에 문자열이나 이미지를 넣을 수 있다.

- [형식] 선택자::가상요소 {......}

1) ::first-line : 태그의 첫번째 줄을 선택

2) ::first-letter : 태그의 첫번째 문자를 선택

3) ::before : 태그의 앞을 선택

4) ::after : 태그의 뒤를 선택

5) ::selection : 마우스 드래그 등으로 선택한 텍스트를 선택

예) p 태그의 첫번째 줄의 글자색을 빨간색으로 만든다.

p::first-line {color: red;}

예) p태그의 첫번째 문자의 색을 빨간색으로 만든다.

p::first-letter {color: red;}

예) p태그 앞에 "xyz"라는 단어를 넣고 색을 빨간색으로 만든다.

p::before { content: "xyz"; color: red; }

예) 마우스로 텍스트를 드래그 했을 때, 글자색은 빨강 바탕색을 노랑으로 만든다.

::selection { color: red; background: yellow; }

**JAVASCRIPT**

**\* alert() 함수의 특징**

=> <body>태그의 내용이 출력된 뒤에, alert() 함수가 호출되면,

배경과 함께 alert() 창이 같이 보이지만

<body>태그의 내용이 출력되기 전에 alert() 창이 동작되면, <body>쪽 내용이 안보인다.

=> alert를 사용했을 때, 배경이 사라질 경우에는 <dialog> 태그를 사용함

Javascript : https://htmlcheatsheet.com/js/

**JAVA**

**OOP – Object Oriented programming**

객체는 하나의 역할을 수행하는 메소드와 변수의 묶음이다. 프로그램을 객체라는 기본 단위로 나누어 이들의 상호작용으로 프로그램을 완성하는 방식이다.

특징으로는 캡슐화, 상속, 추상화, 다형성이 있다.

**클래스와 객체**

클래스란 객체를 정의해놓은 것으로 설계도라고 할 수 있다. 클래스는 객체를 생성하는데 사용한다.

객체는 속성(변수)와 기능(메서드)로 나눌 수 있다.

인스턴스는 특정 클래스로부터 생성된 객체를 뜻한다.

**메서드**

메서드는 작업단위로 문장들을 묶어서 이름 붙인 것이다. 값을 받아서 처리하고, 결과를 반환한다. 장점으로는 중복코드를 제거, 관리용이, 재사용 가능, 등이 있다.

**생성자 – constructor**

인스턴스가 생성될 때마다 호출되는 초기화 메서드이다.

생성자 this는 생성자에서 다른 생성자를 호출한다.

**Web-Info**

**\* eclipse에 서버 추가후 작업**

1) admin port : 8005 추가

=> 서버 아이콘 선택 -> "Tomcat v9.0 Server at localhost" 더블 클릭

-> "Modify the server ports"에서 Tomcat admin port를 8005로 설정

-> 저장 버튼 클릭

**\* port 번호**

- pc안의 프로그램에 부여된 번호

- pc안의 프로그램을 선택

**\* 이클립스에 등록된 아파치-톰캣이 관리하는 폴더**

- 이클립스에서 웹프로젝트를 실행하면, 이클립스는 .metadata폴더에 프로젝트를 복사해서 사용함

=> 이클립스는 프로젝트 원본은 건드리지 않음

\workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps

**\* 웹 프로젝트의 최종 결과물**

- war 파일

- war 파일을 아파치-톰캣이 관리하는 폴더에 저장하면

아파치-톰캣 프로그램은 클라이언트에게 요청이 올 때마다,

해당 파일이 존재하면, 응답으로 파일을 전송함

**\* 아파치-톰캣이 관리하는 폴더**

C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps

=> url : http://localhost:8080/5-test/hello.html

**\* <form> 태그의 데이터 전송 방식**

1) get 방식 (기본 방식)

=> 모든 데이터가 문자열로 전송

=> 주소창에 데이터가 표시됨

=> url : http://localhost:8080/5-test/hello.html?name=홍길동&age=25

2) post 방식

=> 모든 데이터가 스트림 형식으로 전송

=> 파일 또는 문자열을 전송할 때 사용

=> 주소창에 데이터가 표시안됨

**\* 서블릿 동작**

1) 브라우저 요청

http://localhost:8080/7-Servlet/HelloServlet

2) 톰캣서버는 web.xml 파일에 서블릿이 등록되어 있는 지 확인함

HelloServlet => com.hello.HelloServlet

3) 톰캣서버는 com.hello.HelloServlet 클래스를 구동시키고, 함수를 호출함

=> init() 함수 : 서블릿이 시작될 때, 제일 먼저 호출되는 함수,

처음에만 동작되고 그 다음부터는 호출안됨

=> doGet() 함수 : get 방식으로 요청이 오면 호출되는 함수

doPost() 함수 : post 방식으로 요청이 오면 호출되는 함수

4) 톰캣서버가 구동되고 있는 동안, 또 요청이 오면 doGet() 함수를 호출함

5) doGet() 함수에서 데이터를 처리하고, 처리결과를 HTML 문서로 작성해서 응답함

**\* 서블릿을 등록하는 방법**

1) web.xml에 등록

<servlet>

<servlet-name>hello</servlet-name>

<servlet-class>com.hello.HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>hello</servlet-name>

<url-pattern>/HelloServlet</url-pattern>

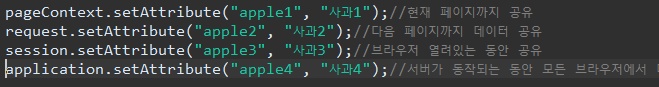
</servlet-mapping>

2) 어노테이션으로 등록

@WebServlet("/HelloServlet")

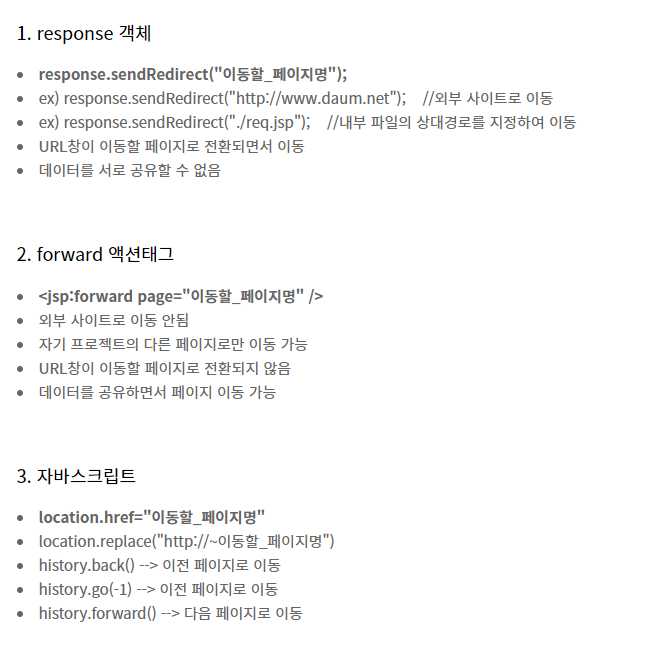
public class HelloServlet extends HttpServlet {}

**\* 웹페이지에서 공통데이터 사용하기**





**\* 화면 이동**



- forward 방식 사용 (jsp 태그)

=> 톰캣서버 내부에서 이동

<jsp:forward page="exam3Pro.jsp" />

<jsp:include page=””/>

=> 클래스 사용

RequestDispatcher dispatcher = request.getRequestDispatcher("forwardResult.jsp");

dispatcher.forward(request, response);

**\* 서블릿에 요청하는 방식**

1) 파라미터 이용

http://localhost:8080/18-mvc/Controller?command=message

2) 요청명 이용

<http://localhost:8080/18-mvc/message.do>

**\* <a> 태그 경로에 "/" 유무에 따른 URL**

1) view = /exam4/messageView.jsp

http://localhost:8080/exam4/messageView.jsp

2) view = exam4/messageView.jsp

http://localhost:8080/18-mvc/exam4/messageView.jsp

**\* forward 경로에 "/" 유무에 따른 URL**

1) view = /exam4/messageView.jsp

URL : http://localhost:8080/18-mvc/exam4/messageView.jsp

2) view = exam4/messageView.jsp

URL : http://localhost:8080/18-mvc/exam4/exam4/messageView.jsp

**\* 서블릿에서 요청 정보 확인 방법**

1) 프로젝트명 뒤의 요청 정보만 필요한 경우

=> /message.do

String command = request.getServletPath();

2) 폴더이름까지 요청 정보가 필요한 경우\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=> /18-mvc/message.do

String command = request.getRequestURI();

String contextPath = request.getContextPath();

int contextPathLength = request.getContextPath().length();

command = command.substring(contextPathLength);

**필수 암기 코드**

**JSP파일 업로드**

cos라이브러리 필수

텍스트, 모니터, 텔레비전, 화면이(가) 표시된 사진

자동 생성된 설명

🡪페이지 이동

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

🡪페이지 이동

텍스트이(가) 표시된 사진

자동 생성된 설명

**MVC – controller**

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

**DAO – connection pool**

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명