Inside Java MVC Frameworks

Appreciating All Levels From Surface to Depth

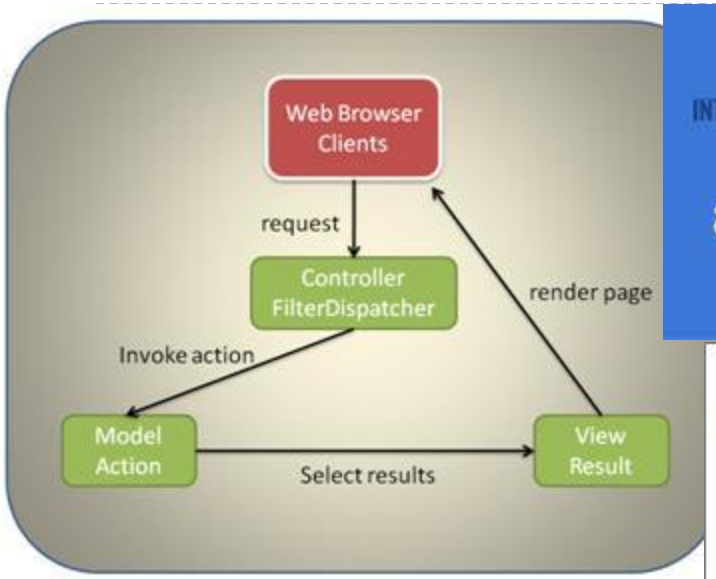
What is a [MVC] Web Framework?

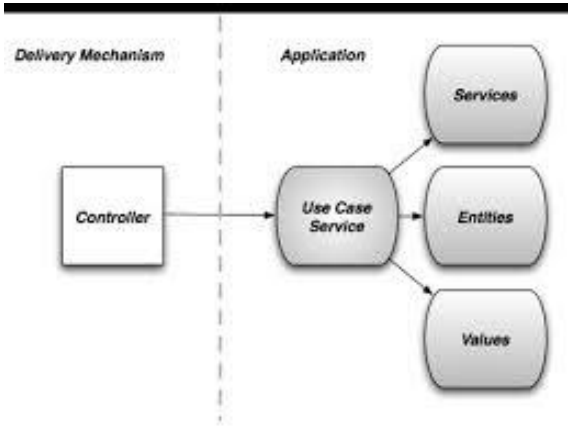
- Designed to simplify development
  - Has already been built, tested, and industry hardened
  - Increases reliability and reduces programming time
  - Adheres to DRY principle
  - Helps enforce best practices and rules

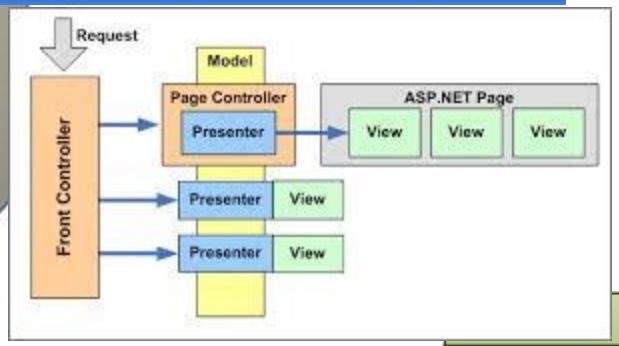
- Common Features
  - ***MVC Front Controller Pattern***
  - ***Validation Framework***
  - Declarative Routing
  - Session Management
  - Security
  - Data Persistence

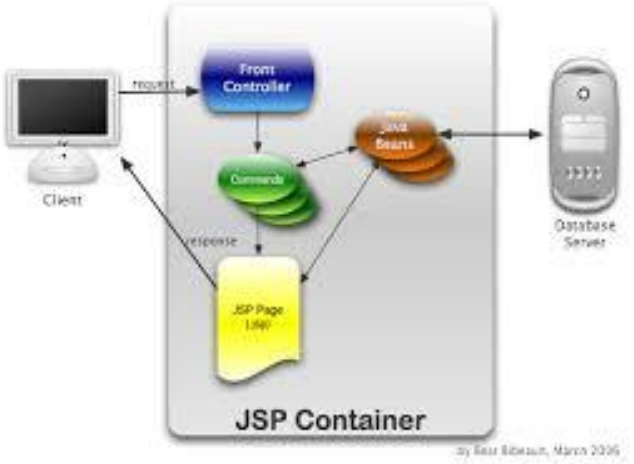
- NOTE: All Frameworks have: Learning Curves"

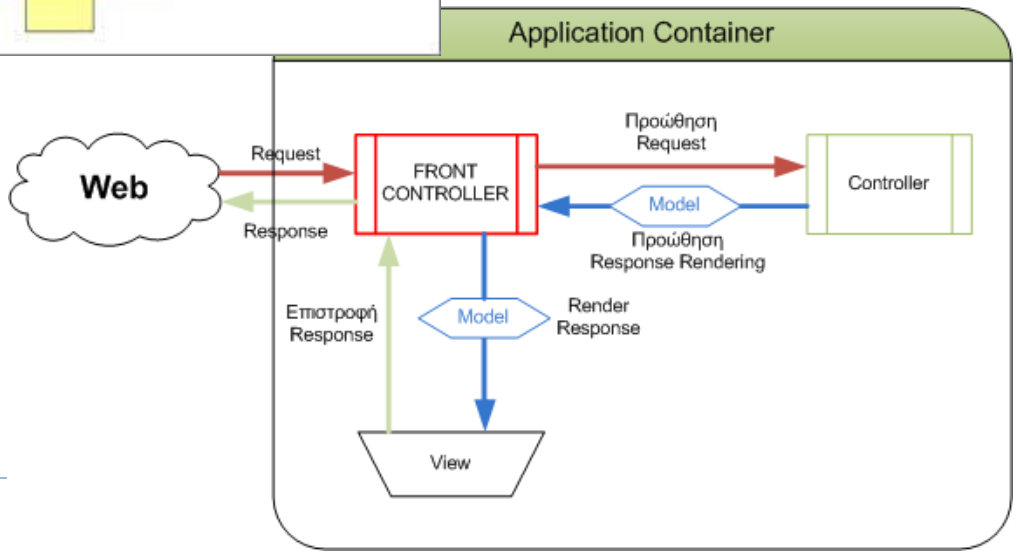
FRONT CONTROLLER



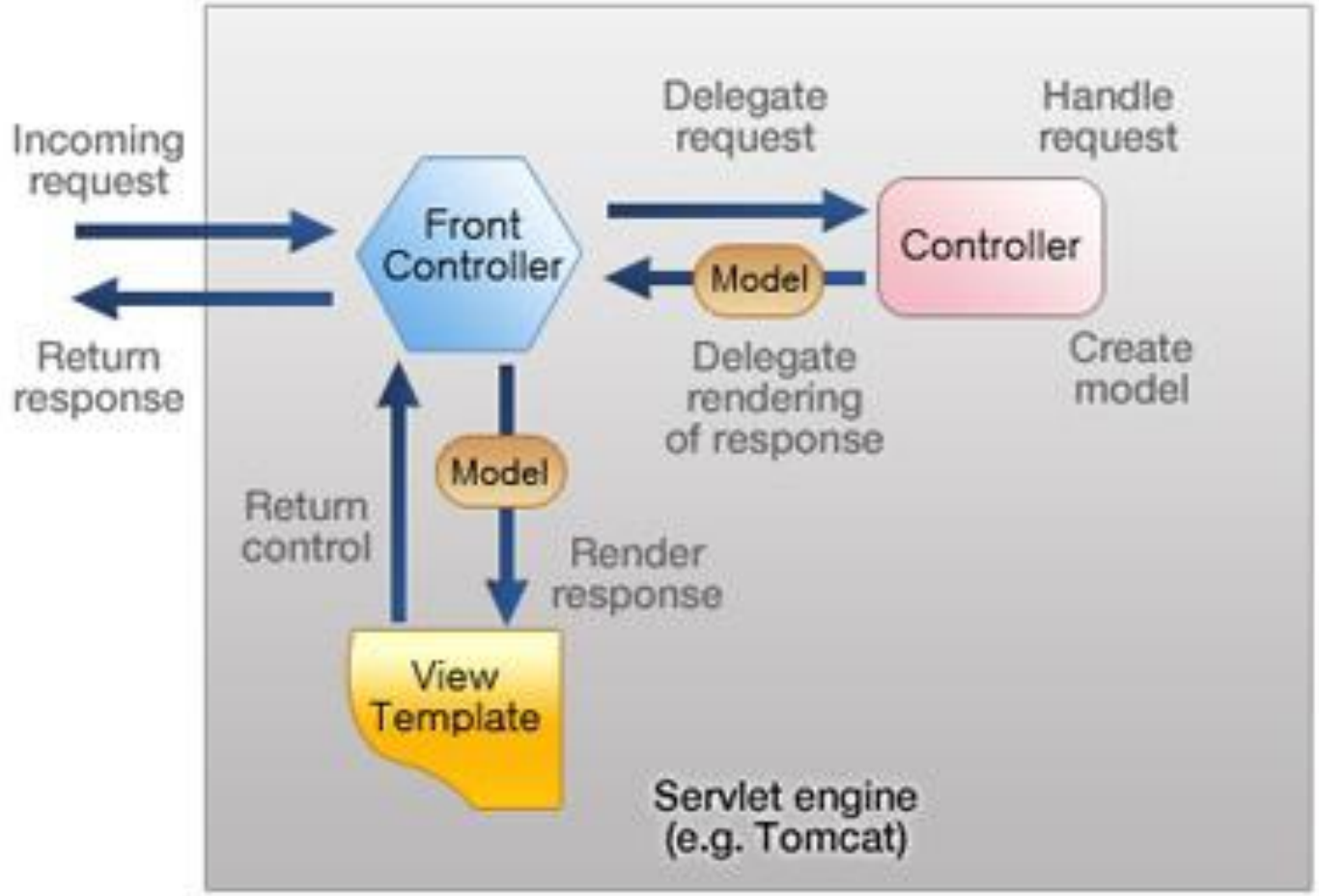








Spring MVC Front Controller



PHASE I - Front Controller & Validation

- **web.xml:**

```
<servlet>
  <servlet-name>DispatcherServlet</servlet-name>
  <servlet-
    class>mum.edu.servlet.DispatcherServlet</servlet-
    class>
</servlet>
<servlet-mapping>
  <servlet-name>DispatcherServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

DispatcherServlet

```
public class DispatcherServlet extends HttpServlet {
    @Override
    public void doGet(...) {
        process(request, response);
    }
    @Override
    public void doPost(...) {
        process(request, response);
    }
    private void process(...) {
        if (action.equals("/product_input") || action.equals("/")) {
        InputProductController controller = new InputProductController();
        dispatchUrl = controller.handleRequest(request, response);
        } else if (action.equals("/product_save")) {
        SaveProductController controller = new SaveProductController();
        dispatchUrl = controller.handleRequest(request, response);
        }
        if (dispatchUrl != null) {
            RequestDispatcher requestDispatcher =
                    request.getRequestDispatcher(dispatchUrl);
            requestDispatcher.forward(request, response);
        }
    }
```

SaveProductController

```
public String handleRequest(...) {
        ProductForm productForm = new ProductForm();
        productForm.setName(request.getParameter("name"));
        productForm.setDescription(request.getParameter("description"));
        productForm.setPrice(request.getParameter("price"));
        // validate ProductForm
        ProductValidator productValidator = new ProductValidator();
        List<String> errors = productValidator.validate(productForm);
        if (errors.isEmpty()) {
            Product product = new Product();
            product.setName(productForm.getName());
            product.setDescription(productForm.getDescription());
            product.setPrice(Float.parseFloat(productForm.getPrice()));

            request.setAttribute("product", product);
            return "/WEB-INF/jsp/ProductDetails.jsp";
        } else {
            request.setAttribute("errors", errors);
            request.setAttribute("form", productForm);
            return "/WEB-INF/jsp/ProductForm.jsp";
        }
```

X

ProductValidator

```
public class ProductValidator {
   public List<String> validate(ProductForm productForm) {
      List<String> errors = new ArrayList<String>();
      String name = productForm.getName();
      if (name == null || name.trim().isEmpty()) {
        errors.add("Product must have a name");
      }
      String price = productForm.getPrice();
      if (price == null || price.trim().isEmpty()) {
        errors.add("Product must have a price");
      } else {
         try {
           Float.parseFloat(price);
         } catch (NumberFormatException e) {
           errors.add("Invalid price value");
         }
      }
      return errors;
   }
8}
```

PHASE II - Declarative Routing

- Generalize the URL-to-Controller Mapping.
- Access a config file through `web.xml` declaration

web.xml:

```
<servlet>
  <servlet-name>DispatcherServlet</servlet-name>
  <servlet-class>mum.edu.servlet.DispatcherServlet</servlet-
    class>
  <init-param>
    <param-name>configFile</param-name>
    <param-value>
      config.properties
    </param-value>
  </init-param>
</servlet>
```

- Load & instantiate Controllers at Startup

PHASE II - Declarative Routing [cont.]

- `config.properties` File data:

```
/product_input=mum.edu.controller.InputProductController
/product_save=mum.edu.controller.SaveProductController
/=mum.edu.controller.InputProductController
```

- `DispatcherServlet.java`

```
public class DispatcherServlet extends HttpServlet {

   Map<String, Controller> controllerDispatch = null;

   @Override
   public void init() throws ServletException {

      String configFile = getServletConfig().getInitParameter("configFile");
      LoadServletProperties loadServletProperties = new
        LoadServletProperties(configFile);
      controllerDispatch = loadServletProperties.loadControllers();
   }
...
```

Dispatcher Routing Change

```
if (action.equals("/product_input") || action.equals("/")) {
   InputProductController controller = new InputProductController();
   dispatchUrl = controller.handleRequest(request, response);
} else if (action.equals("/product_save")) {
   SaveProductController controller = new SaveProductController();
   dispatchUrl = controller.handleRequest(request, response);
}
```

- **REDUCES TO THIS:**

```
Controller controller = controllerDispatch.get(action);
dispatchUrl = controller.handleRequest(request, response);
```

Main Point

- Frameworks make Web development easier and more effective by providing a secure and reliable foundation on which to build upon.
- *The simplest form of awareness, Transcendental Consciousness, provides a strong foundation for a rewarding and successful life.*

- There is MORE that we can do !!!
- WE can:
  - Have MULTIPLE URIs route to a SINGLE Controller
  - AUTOMATICALLY BIND the Domain Object to JSP form
- AND Eventually:
  - Implement Dependency Injection
  - Employ Annotations

But FIRST:

Java Frameworks & Reflection API

- Reflection is a fundamental aspect of Java frameworks
- Reflection allows frameworks to deal with any class at runtime without prior knowledge of it[class].

- The Reflection API provides the following functions:
  - Examine an object's class at runtime
  - Construct an object for a class at runtime
  - Examine a class's field and method at runtime
  - Invoke any method of an object at runtime

- NOTE: Reflection can have a Performance cost

Java “meta-Class”

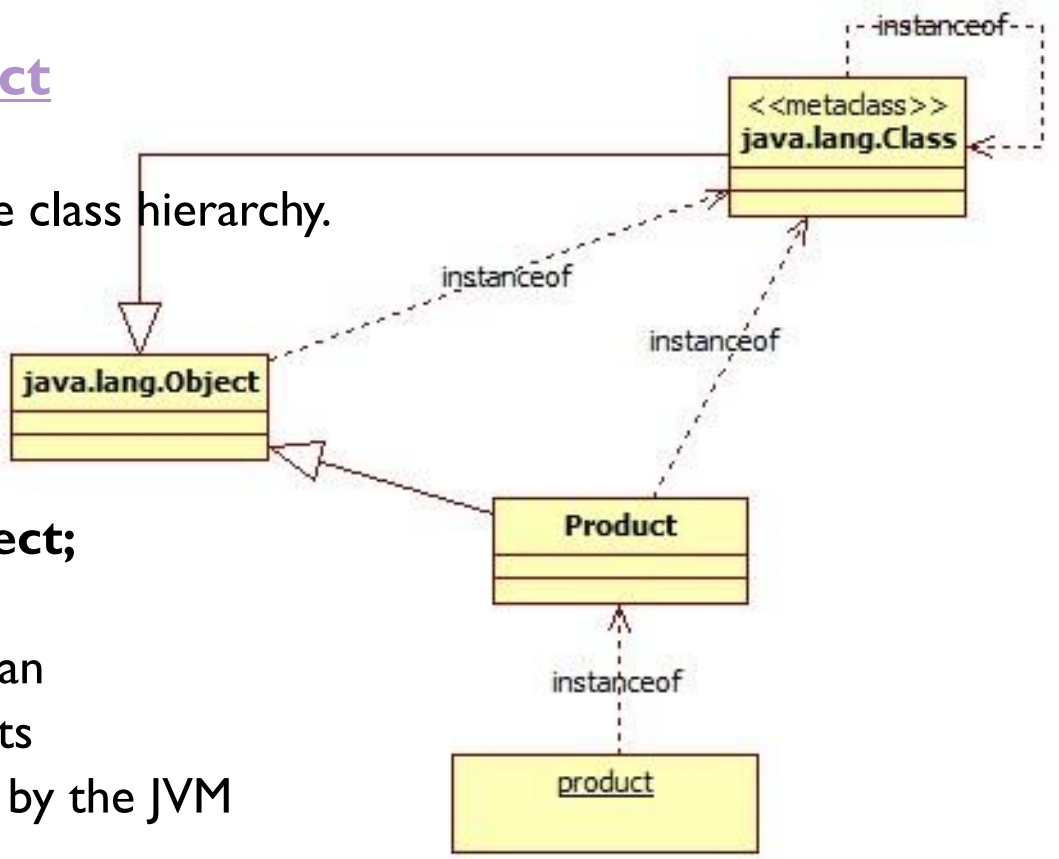
- All objects are instances of a class, and all classes are objects.

- **Class java.lang. Object**

public class **Object**
Class Object is the root of the class hierarchy.
Every class has Object
as a superclass.

- **Class java.lang. Class**

final class **Class** extends **Object;**
Instances of Class represent
classes & interfaces[Object is an
instance of Class]. Class objects
are constructed automatically by the JVM
as classes are loaded.



PHASE III Reflection API

- Add functionality [through config file] to match URI to controller/method name
- Merge InputProductController & SaveProductController into single ProductController
- Performed DATA BINDING on Product Domain Object

Access Config File through Servlet init()

- **DispatcherServlet.java**

```
public class DispatcherServlet extends HttpServlet {
   Map<String, Controller> handlers = new HashMap<String, Controller>();
   Map<String, ControllerMethod> handlerMethods = new HashMap<String,
      ControllerMethod>();

   @Override
   public void init() throws ServletException {
      String configFile =
        getServletConfig().getInitParameter("configFile");
      LoadServletProperties loadServletProperties = new
        LoadServletProperties();
      loadServletProperties.loadControllers(configFile, handlers,
        handlerMethods);
   }
…
```

Process Config File

```
 // Enumerate thru Controllers,handlers...
Enumeration  enumeration = prop.keys();
while (enumeration.hasMoreElements()) {
String key =
   (String) enumeration.nextElement();
if (prop.get(key).equals("Start")) {
    type = key;
    continue;
}
if (type.equals("Controllers"))
   controller =
     getControllerInstance((String)prop.get(key));
   controllers.put(key, controller);
else if (type.equals("Handlers")) {
   controller = controllers.get((String)prop.get(key));
   handlers.put(key, controller);
else if (type.equals("Methods")) {
   String temp = (String) prop.get(key);
   ControllerMethod controllerMethod = getMethodDetails(controller,temp);
   handlerMethods.put(key, controllerMethod); }
```

`config.properties` File data:

```
Controllers=Start
ProductController=mum.edu.controller.ProductController

Handlers=Start
/product_input=ProductController
/product_save=ProductController
/=ProductController

Methods=Start
M/product_input=inputProduct
M/product_save=saveProduct
M/=inputProduct
```

DisplatcherServlet

```
// Get Controller Method parameters – identified in Config
Method method = controllerMethod.getMethod();
Map<String, Object> params = controllerMethod.getParams();

// To be filled in with the parameters from request
Object[] methodParams = new Object[method.getParameterTypes().length];

// ORDER IS IMPORTANT [KLUDGE!!! we are taking a short cut by enforcing the order]
int n = 0;
if (params.get("domainObject") != null)
    methodParams[n++] = params.get("domainObject");
if (params.get("request") != null)
    methodParams[n++] = request;
if (params.get("response") != null)
    methodParams[n++] = response;

// If it is a POST, we want to BIND the request parameters to the Domain Object
if (request.getMethod().equals("POST")) {
    domainDataBinding(request, controllerMethod);
}
// call the controller method with the appropriate "args"
// for example, productController.saveProduct(product,request,response)
dispatchUrl = (String) method.invoke(controller, methodParams);
```

Data Binding

```
Enumeration<String> parameterNames = request.getParameterNames();
Object domainObject = controllerMethod.getParams().get("domainObject");
Map<String,Method> domainObjectSetters =
                                controllerMethod.getDomainObjectSetters();
while (parameterNames.hasMoreElements()) {
   String fieldName = (String) parameterNames.nextElement();
   // value of the form field, e.g., name,description OR price
   Object[] value = (Object[])parameterMap.get(fieldName);
   domainMethod=domainObjectSetters.get(fieldName) //Method e.g.,setName()
   Class<?>[] parameterTypes  = domainMethod.getParameterTypes();
    String strVal = ((String)value[0]).trim();
   if (parameterTypes[0]  == String.class)
       domainMethod.invoke(domainObject, strVal); //invoke method W/string
   else if (parameterTypes[0]  == Double.class)
       Double val  Double.valueOf(strVal);
      domainMethod.invoke(domainObject, val); //invoke method W/Double
   else if (parameterTypes[0]  == Integer.class) {
    Integer val = Integer.valueOf(strVal);
    domainMethod.invoke(domainObject, val); //invoke method W/Integer
   }
   …
```

ProductController

```
public String saveProduct(Product product, HttpServletRequest request) {
        // validate Product
        ProductValidator productValidator = new ProductValidator();
        List<String> errors = productValidator.validate(product);
        if (errors.isEmpty()) {
            request.setAttribute("product", product);
            return "/WEB-INF/jsp/ProductDetails.jsp";
        } else {
            // store errors and form in a scope variable for the view
            request.setAttribute("errors", errors);
            request.setAttribute("form", product);
            return "/WEB-INF/jsp/ProductForm.jsp";
```

- Compare with Slide 7

Main Point

- ALL OO constructs of Java are defined by the circular and reflexive aspects of their fundamental design.
- *In this case, we see a clear example of the concept of self-referral that characterizes all of Life at its basis.*

PHASE IV DI & Annotations

DEPENDENCY INJECTION

Whenever we create object using

new()

we violate the

principle of programming to an interface rather than implementation

which eventually results in code that is inflexible and difficult to maintain.

Annotations

- Metadata - to *describe* the usage and meaning of entities like methods and classes
- No direct effect on the operation of the code they annotate
- Can be evaluated by "others" (e.g., frameworks)
- Usage: "inline" configuration; control of lifecycle behavior

We are going to use an Annotation to implement Dependency Injection

@Autowired

```
@Documented
@Retention(java.lang.annotation.RetentionPolicy.RUNTIME)
@Target({java.lang.annotation.ElementType.FIELD})
public @interface AutoWired {}
```

Usage in ProductController.java

```
@AutoWired
Validator productValidator;
...
public String saveProduct(Product product...) {
//ProductValidator productValidator = new ProductValidator();
  List<String> errors = productValidator.validate(product);
```

@Autowired processing

- Backed by configure time processing using Reflection API

```
public class ProcessAnnotations {
```

```
Set<String> keys = (Set<String>) controllers.keySet();
for (String key : keys) {
    Controller controller = controllers.get(key);
    Field[] fields = controller.getClass().getDeclaredFields();
    for (Field field : fields) {
        if (field.isAnnotationPresent(AutoWired.class)) {
            AutoWired dependency = field.getAnnotation(AutoWired.class);
            if (dependency != null) {
                try {
                    Class<?> fieldClass = field.getType();
                    Object setter = InjectionFactory.getInstance(fieldClass.getName());
                    field.setAccessible(true);
                    field.set(controller, setter);
                } catch (Exception e) {
                    e.printStackTrace();
                }
```

NOTE: MORE Reflection….

PHASE V More Annotation

- **Annotate the Controller method with URL mapping**

```
@RequestMapping(value={"/","/product_input"})
public String inputProduct(HttpServletRequest request,
                           HttpServletResponse response) {
```

- **Simplifies Config file**

```
Controllers=Start
Controller=mum.edu.controller
```

Main Point

- Variations on the Reflection API usage coupled with Annotations allow us to apply best practices W/R to Java Object construction and lifecycle management.
- *Understanding more fundamental aspects of "**any thing**" makes us able to put those principles to proper use. Transcendental Consciousness is the ultimate fundamental aspect of Nature.*