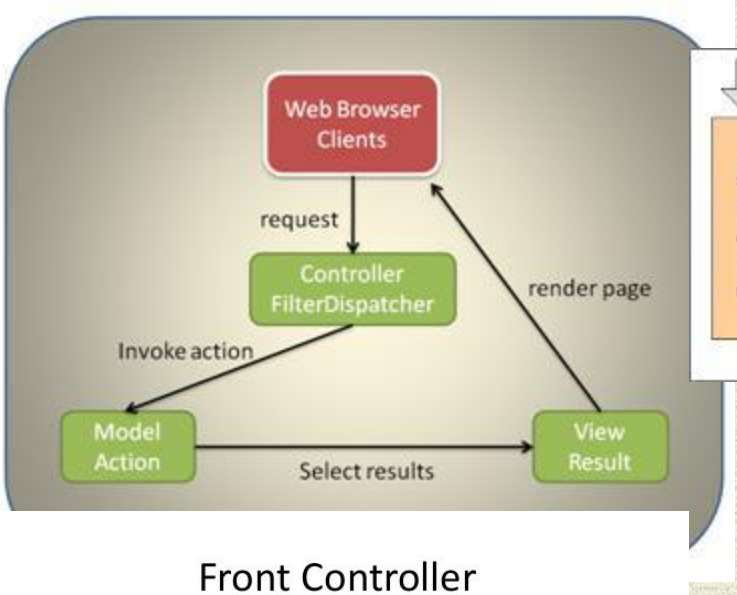
INSIDE JAVA MVC FRAMEWORKS

Appreciating All Levels From Surface to Depth

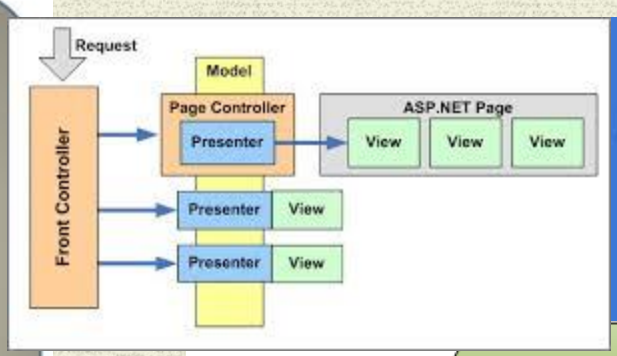
What is a Web Framework?

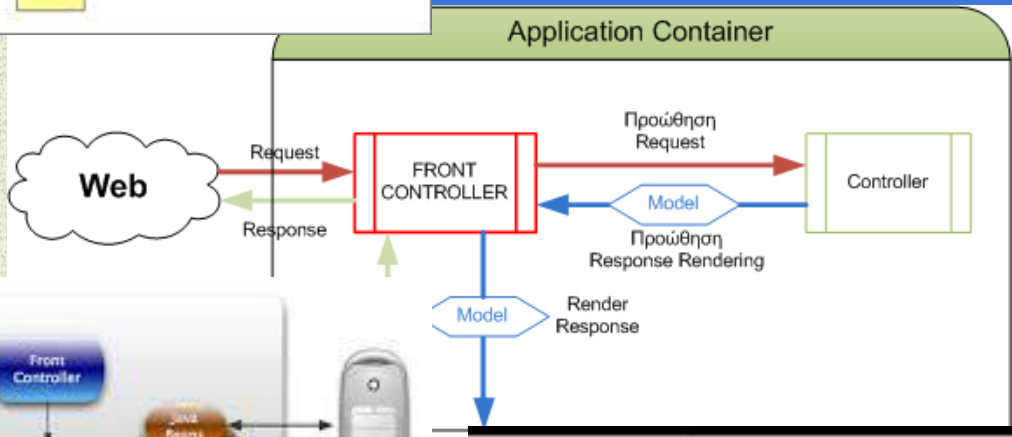
- Designed to simplify development
  - Has already been built, tested, and industry hardened
  - Increases reliability and reduces programming time
  - Adheres to DRY principle
  - Helps enforce best practices and rules
- Common Features
  - ***MVC Front Controller Pattern***
  - ***Validation Framework***
  - ***Declarative Routing***
  - Session Management
  - Security
  - Data Persistence
- NOTE: All Frameworks have: Learning Curves”

FRONT CONTROLLER



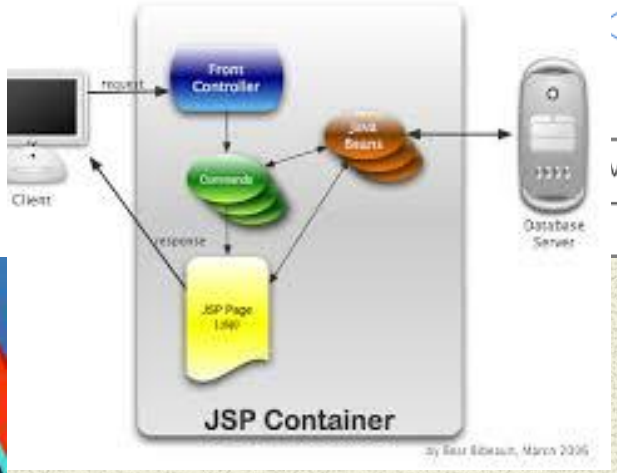
Front Controller

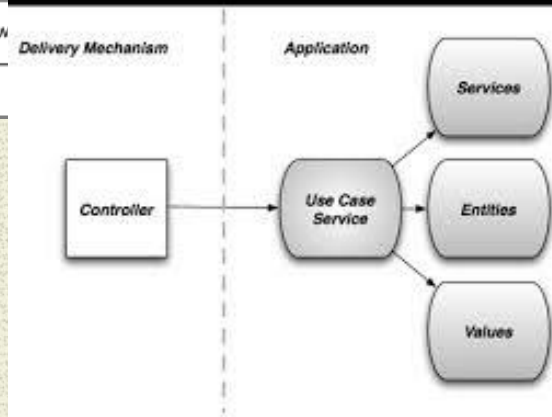


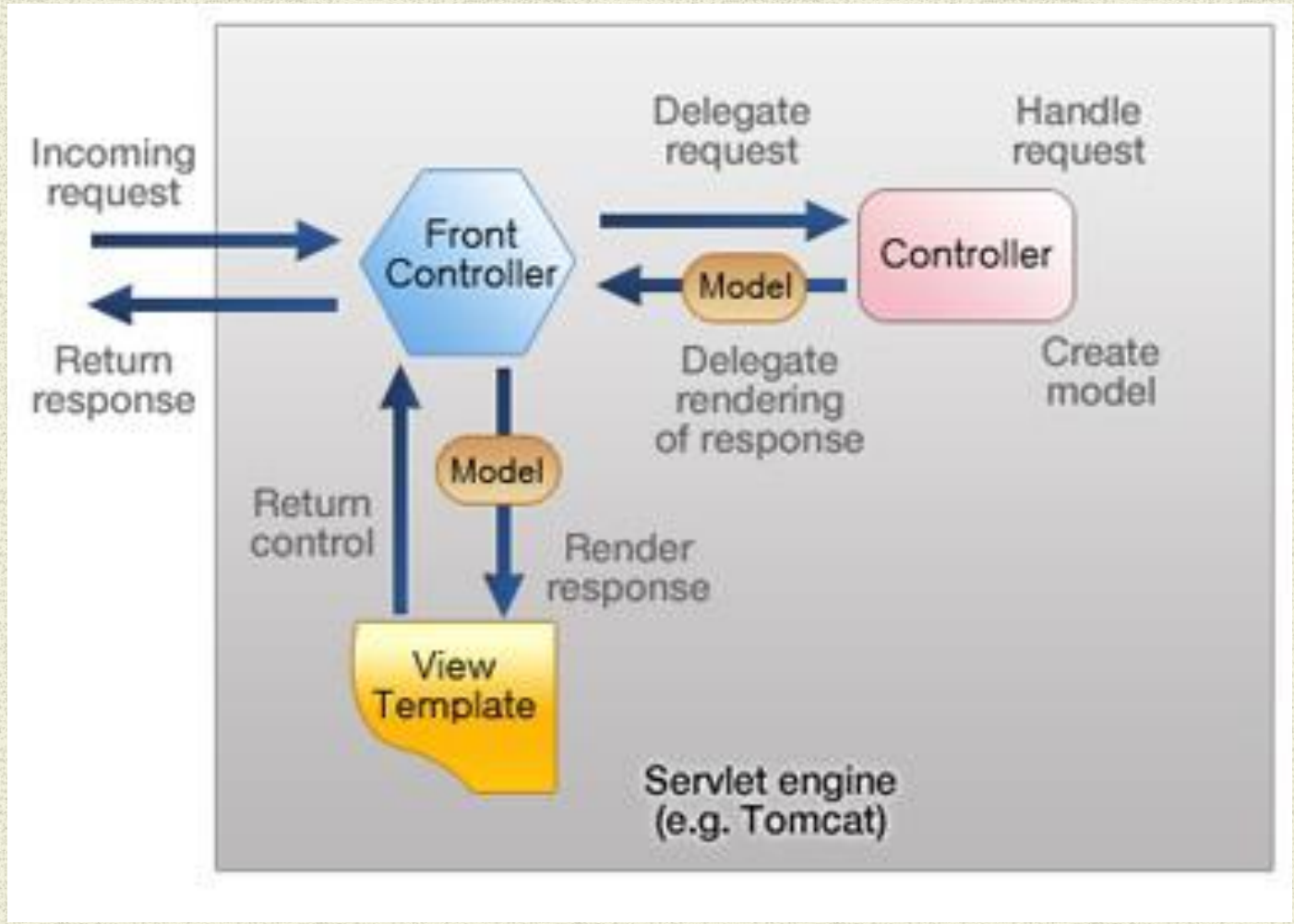


Problema

- Si vuole fornire un punto di accesso centralizzato per la gestione delle richieste al livello dello strato di presentazione, in modo da sparare la logica di presentazione da quella di *processamento* delle richieste stesse.
- Inoltre si vuole evitare di avere del codice duplicato e si vuole applicare una logica comune a più richieste.







DEMO Time - Let's take a Look!

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;
```

PHASE II - Declarative Routing

- Generalize the URL-to-Controller Mapping.
- Access a config file through WEB.XML declaration

web.xml:

```
<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>
```

- Load & instantiate Controllers at Startup

PHASE II - Declarative Routing [cont.]

- Config File data:

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

```
public class DispatcherServlet extends HttpServlet {
     Map controllerDispatch = null;

   @Override
    public void init( ) throws ServletException {

    LoadServletProperties  loadServletProperties=
                                   new LoadServletProperties();
    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 = (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:

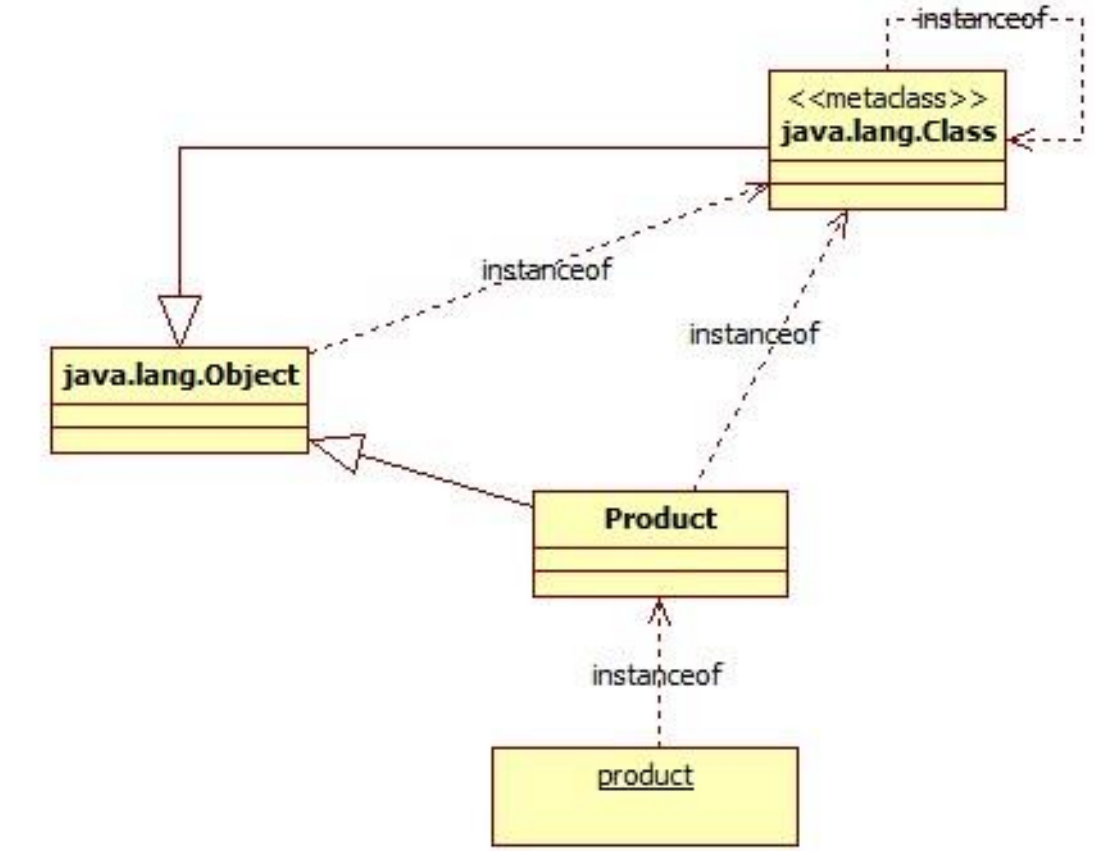
Java Frameworks & Reflection API

- 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:**

```
public void init( ) throws ServletException {

  Map<String, Controller> controllers= new HashMap<String, Controller>();
  Map<String, Controller> dispatchers = new HashMap<String, Controller>();
  Map<String, String> dispatcherMethods = new HashMap<String, String>();

    String configFile = super.getInitParameter("configFile");
  LoadServletProperties loadServletProperties=
                                        new LoadServletProperties();
   loadServletProperties.loadControllers(configFile,
              controllers,dispatchers,dispatcherMethods);
}
```

Process Config File

```
// Enumerate thru Controllers,Dispatchers...
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("Dispatchers")) {
     controller = controllers.get((String)prop.get(key));
     dispatchers.put(key, controller);
  else if (type.equals("Methods")) {
     String temp = (String) prop.get(key);
     key = key.substring(1);
     dispatcherMethods.put(key, temp);
```

CONFIG FILE:

```
Controllers=Start
ProductController=mum.ProductControlle

Dispatchers=Start
/product_input=ProductController
/product_save=ProductController
/=ProductController

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

DispatcherServlet Process Request

```
 Controller controller =  (Controller) dispatchers.get(action);
try{
// Find Controller method... ASSUMES: only one method with this name
 String methodName = dispatcherMethods.get(action);
 Class classObject = controller.getClass();
Method method = null;
for(Method m : classObject.getMethods()) {
    if (m.getName().equals(methodName)) {
    method = m;
     break;
    }
// Get the parameters...ASSUMES if NOT HTTP arguments then create ONE...
Object[] methodParams = new Object[method.getParameterTypes().length];
Object domainObject = null;
for (int i = 0; i < method.getParameterTypes().length; i ++) {
    Class<?> paramClass = method.getParameterTypes()[i];
    if (paramClass == HttpServletRequest.class
        methodParams[n++] = request;
   else if (paramClass == HttpServletResponse.class)
        methodParams[n++] = response;
   else { //Save extra parameter as this is our domain object ASSUMES: ONLY ONE
        domainObject =   paramClass.getConstructor().newInstance();
        methodParams[n++] = domainObject;
```

Performance Consideration: Move as much Reflection to "Configuration" - LoadServletProperties as possible

```
// If it is a POST,  we want to BIND the request parameters to the Domain Object
 if (request.getMethod().equals("POST")) {
    Method domainMethod = null;
   Map parameterMap = request.getParameterMap();
   Enumeration parameterNames = request.getParameterNames();
  while (parameterNames.hasMoreElements()) {
   String name = (String) parameterNames.nextElement();
   Object[] value = (Object[])parameterMap.get(name);

   name = Character.toUpperCase(name.charAt(0)) + name.substring(1);
   String domainObjectSetter = "set" + name;
  Class<?>[] parameterTypes = null;
  for(Method m : domainObject.getClass().getMethods()) {
   if (m.getName().equals(domainObjectSetter)) {
   domainMethod = m;
   break;
   }
   parameterTypes = domainMethod.getParameterTypes();
   if (parameterTypes[0]  == String.class)
        domainMethod.invoke(domainObject, value[0] );
   else if (parameterTypes[0]  == Double.class) {
       Double val = Double.valueOf((String)value[0]);
       domainMethod.invoke(domainObject, val);
   }
dispatchUrl = (String) method.invoke(controller, methodParams);
```

ProductController

```
public String saveProduct(Product product, HttpServletRequest request,
HttpServletResponse response) {
        // 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

- The OO constructs of Java are defined by the circular and reflexive aspects of their basic design.
- *In this case, we can see a powerful example of the concept of self-referral that characterizes 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})
@Inherited
public @interface AutoWired {}
```

Usage in ProductController.java

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

@Autowired processing

- Backed by configure time processing using Reflection API

```
public class ProcessAnnotations {
/*
 * Loop through Controllers looking for Annotations [@Autowired)
 */
      public static void handleAnnotations(
            Map<String,Controller> controllers ) {
            .....
      }
```

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
ProductController=mum.edu.controller.ProductController
```

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.*