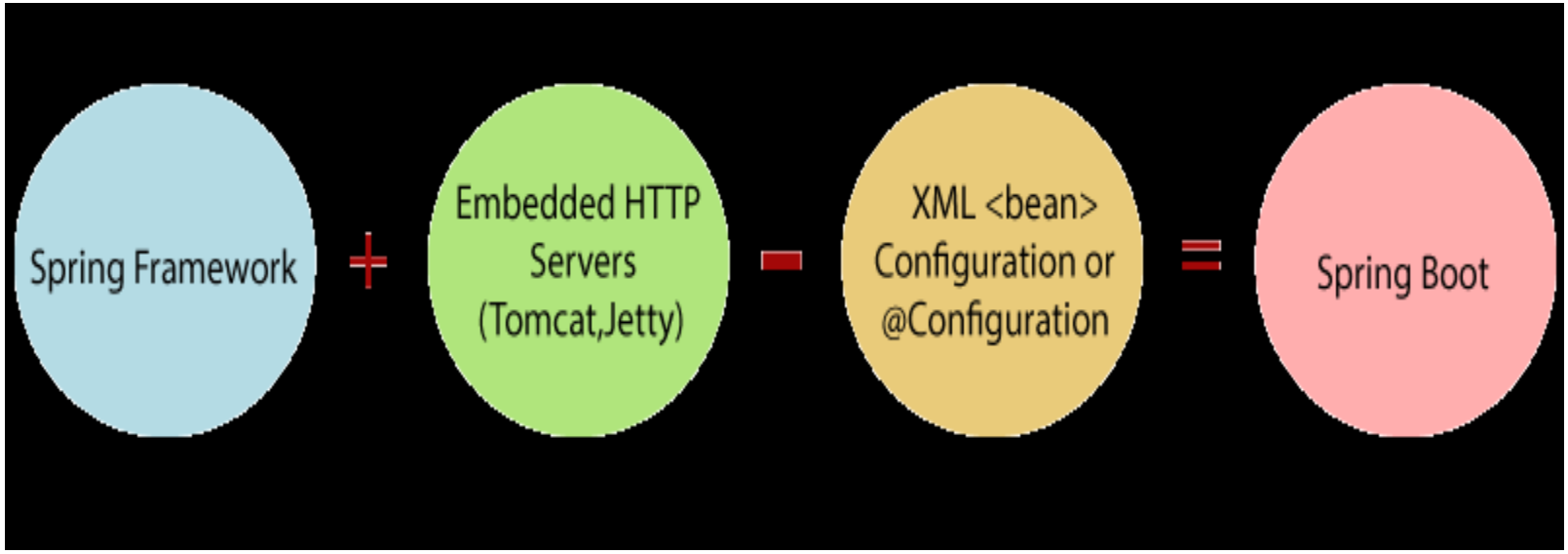
KLEFSPRO
Learn & Grow

Introduction to springBoot

What is Spring Boot

- **Spring Boot is a project that is built on the top of the Spring Framework.**
- **It provides an easier and faster way to set up, configure, and run both simple and web-based applications.**
- **It provides the RAD (Rapid Application Development) feature to the Spring Framework.**
- **It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.**
- **It is the combination of Spring Framework and Embedded Servers.**
- **In Spring Boot, there is no requirement for XML configuration (deployment descriptor). It uses convention over configuration software design paradigm that means it decreases the effort of the developer.**

- **We can use Spring STS IDE or Spring Initializr to develop Spring Boot Java applications.**


Spring Framework
+
Embedded HTTP Servers (Tomcat,Jetty)
-
XML <bean> Configuration or @Configuration
=
Spring Boot

Why should we use Spring Boot Framework?

- The dependency injection approach is used in Spring Boot.
- It contains powerful database transaction management capabilities.
- It simplifies integration with other Java frameworks like JPA/Hibernate ORM, Struts, etc.
- It reduces the cost and development time of the application.

Advantages of Spring Boot

- **It creates stand-alone Spring applications that can be started using Java -jar.**
- **It tests web applications easily with the help of different Embedded HTTP servers such as Tomcat, Jetty, etc. We don't need to deploy WAR files.**
- **It provides opinionated 'starter' POMs to simplify our Maven configuration.**
- **It provides production-ready features such as metrics, health checks, and externalized configuration.**
- **There is no requirement for XML configuration.**
- **It offers a CLI tool for developing and testing the Spring Boot application.**
- **It offers the number of plug-ins.**
- **It also minimizes writing multiple boilerplate codes (the code that has to be included in many places with little or no alteration), XML configuration, and annotations.**
- **It increases productivity and reduces development time.**

Limitations of Spring Boot

- Spring Boot can use dependencies that are not going to be used in the application. These dependencies increase the size of the application.

Goals of Spring Boot

- **The main goal of Spring Boot is to reduce development, unit test, and integration test time.**
- **Provides Opinionated Development approach**
- **Avoids defining more Annotation Configuration**
- **Avoids writing lots of import statements**
- **Avoids XML Configuration.**
- **By providing or avoiding the above points, Spring Boot Framework reduces Development time, Developer Effort, and increases productivity.**

Spring MVC framework

Lesson Objectives

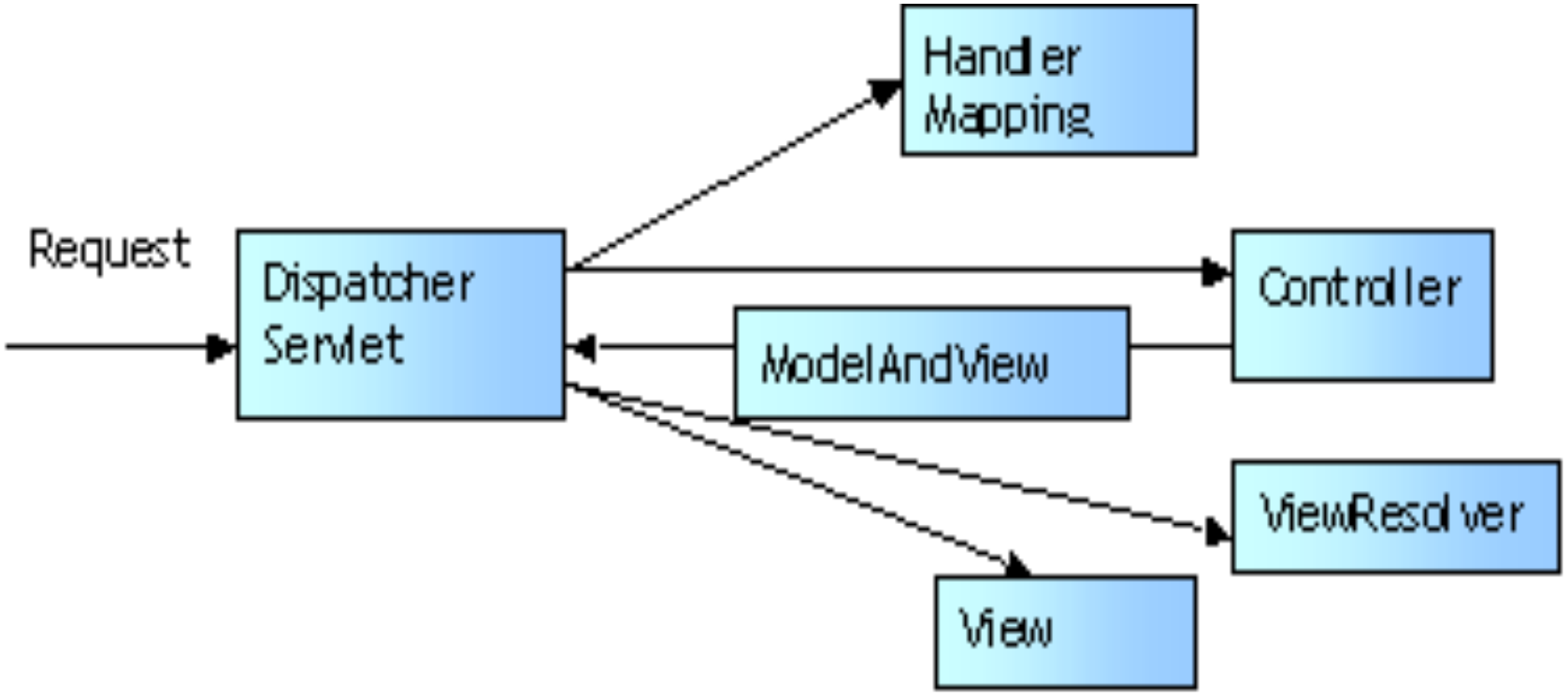
- **Introduction to Spring MVC framework**
  - Learn how to develop web applications using Spring
  - Understand the Spring MVC architecture and the request cycle of Spring web applications
  - Understand components like handler mappings, ViewResolvers and controllers
  - Use MVC Annotations like @Controller, @RequestMapping and @RequestParam

Spring MVC Framework Features

- **Provides you with an out-of-the-box implementations of workflow typical to web applications**
- **Allows you to use a variety of different view technologies**
- **Enables you to fully integrate with your Spring based, middle-tier logic through the use of dependency injection**
- **Displays modular framework, with each set of components having specific roles and completely decoupled from the rest of the framework**

Spring MVC lifecycle

- **Life cycle of a Request in Spring MVC**



Traditional configuration
Configuring the DispatcherServlet in web.xml

```
<servlet>
   <servlet-name>basicspring</servlet-name>
   <servlet-class> org.springframework.web.servlet.DispatcherServlet
   </servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>basicspring</servlet-name>
   <url-pattern>*.obj</url-pattern>
 </servlet-mapping>
```

The servlet-name given to the servlet is significant

- **Steps to build a homepage in Spring MVC:**
  - Write the controller class that performs the logic behind the homepage
  - Configure controller in the DispatcherServlet's context configuration file
  - Configure a view resolver to tie the controller to the JSP
  - Write the JSP that will render the homepage to the user

Building a Homepage in Spring MVC (Contd..)

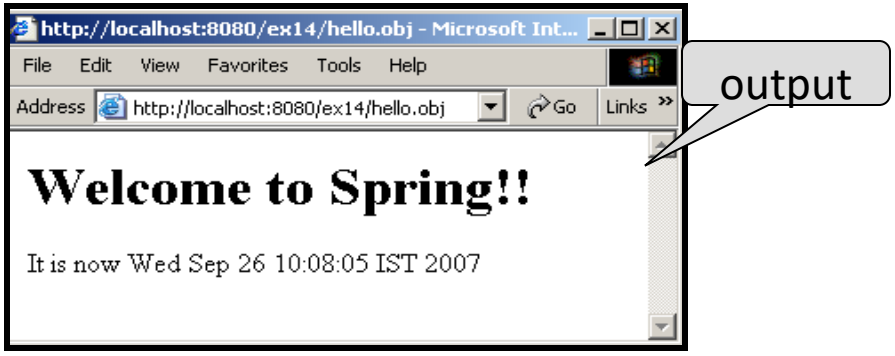
- **Step 3 : Configure a view resolver to bind controller to the JSP**

```
<bean id="viewResolver"
  class="org.springframework.web.servlet.view.InternalResourceViewResolver ">
    <property name="viewClass">
       <value>org.springframework.web.servlet.view.JstlView </value>
    </property>
    <property name="prefix"><value>/WEB-INF/views/</value></property>
    <property name="suffix"><value>.jsp</value></property>
</bean>
```

/hello.jsp

- **Step 4 : Write the JSP that will render the homepage to the user**

```
<html>
 <body>
  <h1>Welcome to Spring!! </h1>
   Now it is ${now}
 </body>
</html>
```



ModelAndView Class

- **This class fully encapsulates the view and model data that is to be displayed by the view. Eg:**

```
ModelAndView("hello","now",now);
```

```
Map myModel = new HashMap();
myModel.put("now",now);
myModel.put("products",getProductManager().getProducts());
return new ModelAndView("product","model",myModel);
```

  - Every controller returns a ModelAndView
  - Views in Spring are addressed by a view name and are resolved by a view resolver

Resolving Views : The ViewResolver

View resolver	How it works
InternalResourceViewResolver	Resolves logical view names into View objects that are rendered using template file resources
BeanNameViewResolver	Looks up implementations of the View interface as beans in the Spring context, assuming that the bean name is the logical view name
ResourceBundleViewResolver	Uses a resource bundle that maps logical view names to implementations of the View interface
XmlViewResolver	Resolves View beans from an XML file that is defined separately from the application context definition files

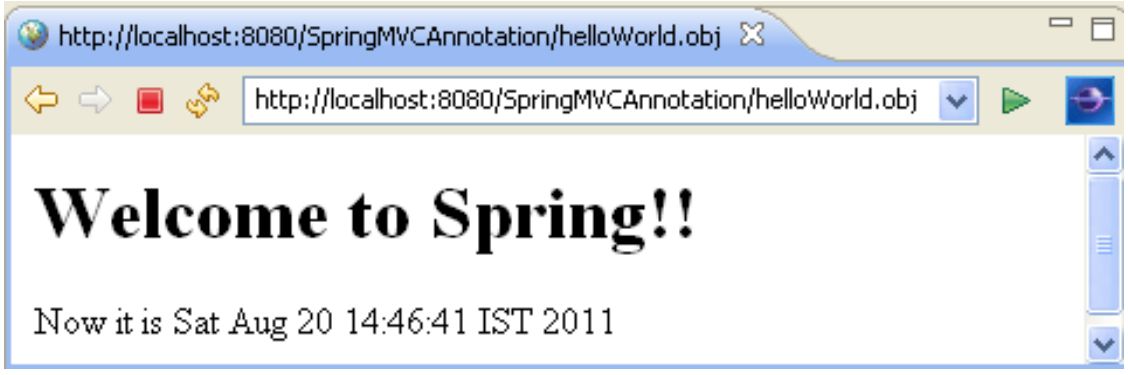
Implementing Controllers

```
@Controller
public class HelloController1 {
  @RequestMapping("/helloWorld")
   public ModelAndView handleRequest(HttpServletRequest request,
                    HttpServletResponse response) throws ….. {
       String now = new java.util.Date().toString();
       return new ModelAndView("hello", "now", now);
    }}
```

```
  @Controller
  public class HelloController {
     @RequestMapping("/helloWorld")
     public String handleMyRequest(Map<String, Object> model) {
          String now = new java.util.Date().toString();
          model.put("now", now);
          return "hello";
}}
```

Demo

- **Execute the applications in the SpringMVCBootAnnotation web project**



Using Spring boot

- It is a well-suited Spring module for web application development. We can easily create a self-contained HTTP application that uses embedded servers like Tomcat, Jetty, or Undertow. We can use the spring-boot-starter-web module to start and run the application quickly.

- SpringApplication

- The SpringApplication is a class that provides a convenient way to bootstrap a Spring application. It can be started from the main method. We can call the application just by calling a static run() method.

- public static void main(String[] args)
- {
- SpringApplication.run(ClassName.class, args);
- }

Spring boot MVC

Properties File

- **Spring Boot provides a rich set of Application Properties. So, we can use that in the properties file of our project.**
- **The properties file is used to set properties**
- **Example**

server-port =8082

spring.mvc.view.prefix=/WEB-INF/views/

spring.mvc.view.suffix=.jsp

POM.xml

```
<dependencies>

        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-devtools</artifactId>
                <scope>runtime</scope>
                <optional>true</optional>
        </dependency>
```

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
</dependency>

<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
</dependency>

<dependency>
        <groupId>org.apache.tomcat.embed</groupId>
        <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
</dependencies>
```

HelloController

- @Controller
- public class HelloController {

- @GetMapping("/")
- public String sayHello(Map<String, Object> model) {
- model.put("userDate",new Date());
- model.put("name", "Kishori");
- return "hello";
- }

View file hello.jsp

- <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
- pageEncoding="ISO-8859-1"%>
- <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
- <html>
- <head>
- <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
- <title>Insert title here</title>
- </head>
- <body>
- hello world
- Hello ${name}
- Hello ${userDate}
- </body>
- </html>

RestWebService

- **@RestController annotation is a combination of Spring's @Controller and @ResponseBody annotations.**

```
@RestController
public class HelloController {

        @GetMapping("/")
        public String sayHello() {
                return "Hello world!";
        }
```

Aspect Oriented Programming (AOP)

Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.demo.springbootAOP/groupId>
   <artifactId>spring-boot-tutorial-basics</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <packaging>jar</packaging>
   <name>spring-boot-tutorial-basics</name>
   <description>Spring Boot Tutorial - Basic Concept Examples</description>
   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.0.0.M6</version>
      <relativePath />
      <!-- lookup parent from repository -->
```

```
</parent>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-aop</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
```

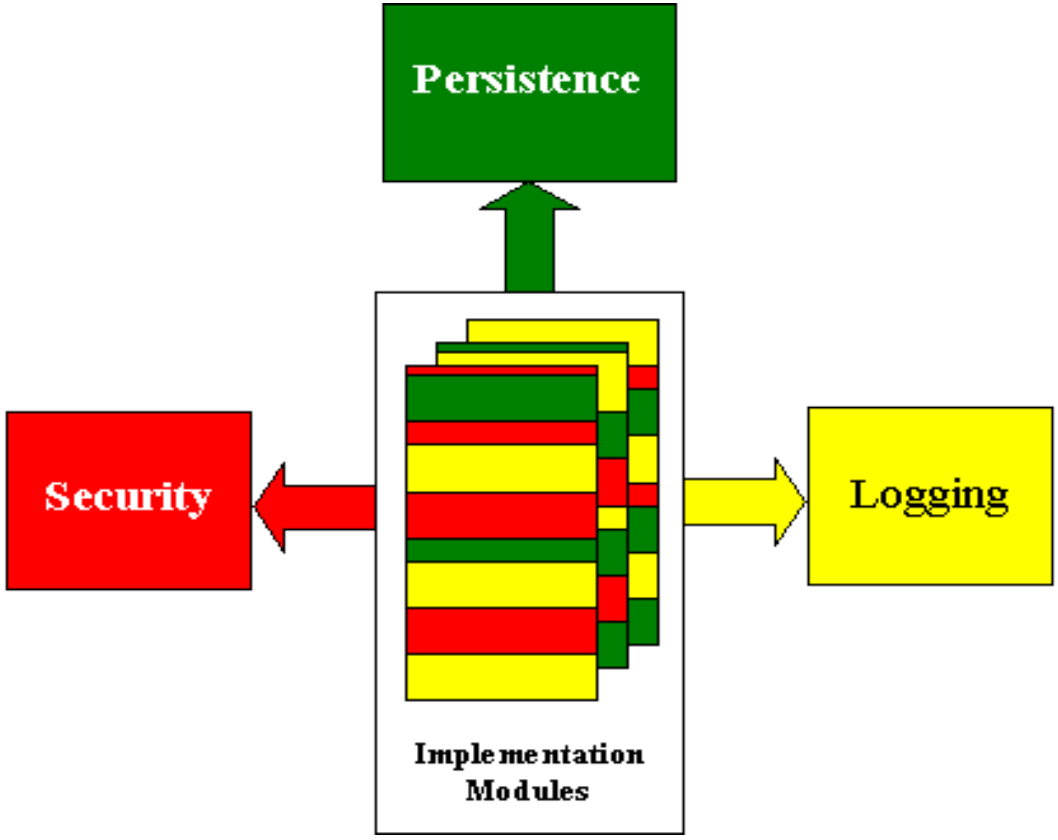
```
</dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
  </project>
```

Lesson Objectives

- **Introduction to Spring AOP**
  - Learn AOP basics and terminologies
  - Understand key AOP terminologies
  - Understand the different ways that Spring supports AOP

Introduction to AOP

- AOP complements OOP
- Aspects enable the modularization of concerns that cut across multiple types and objects
- AOP complements Spring IoC to provide a very capable middleware solution



Understanding AOP

- **An Example:**

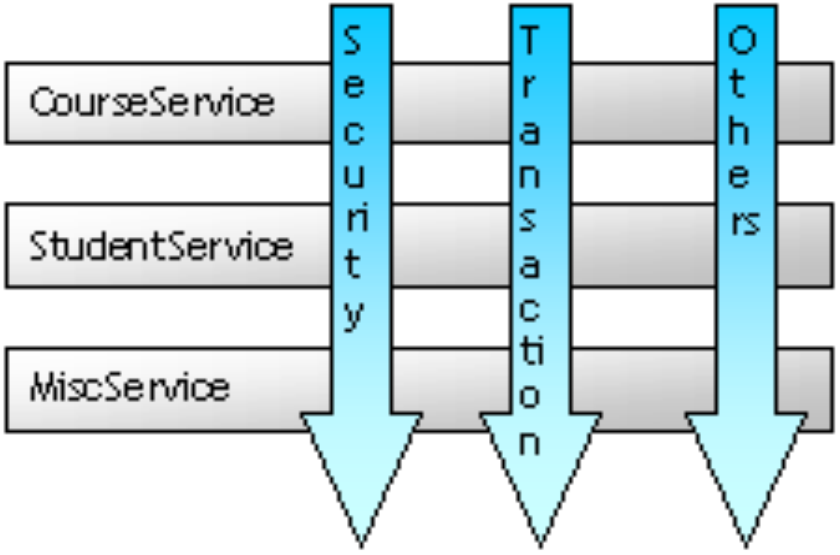
```
void transfer(Account src, Account tgt, int amount) {
  if (src.getBalance() < amount) {
    throw new InsufficientFundsException();
  }
   src.withdraw(amount);
  tgt.deposit(amount);
}
```

Understanding AOP: Example

```
void transfer(Account src, Account tgt, int amount) {
    if (!getCurrentUser().canPerform(OP_TRANSFER))
                throw new SecurityException();
    if (amount < 0)
                throw new NegativeTransferException();
   if (src.getBalance() < amount) {
                throw new InsufficientFundsException();  }
   Transaction tx = database.newTransaction();
    try {
        src.withdraw(amount);
           tgt.deposit(amount);
           tx.commit();
       systemLog.logOperation(OP_TRANSFER, src, tgt, amount);
 }
 catch(Exception e) {   tx.rollback();  }
```

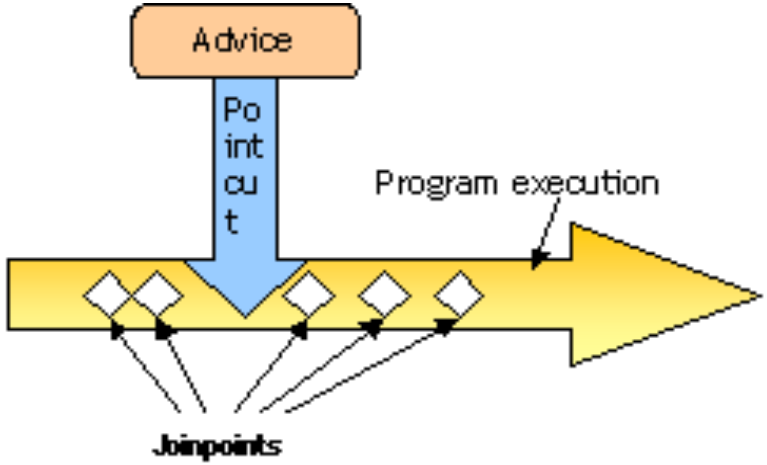
AOP and Spring

- **AOP attempts to separate concerns, that is, break down a program into distinct parts that overlap in functionality sparingly.**
- **In particular, AOP focuses on the modularization and encapsulation of cross-cutting concerns.**



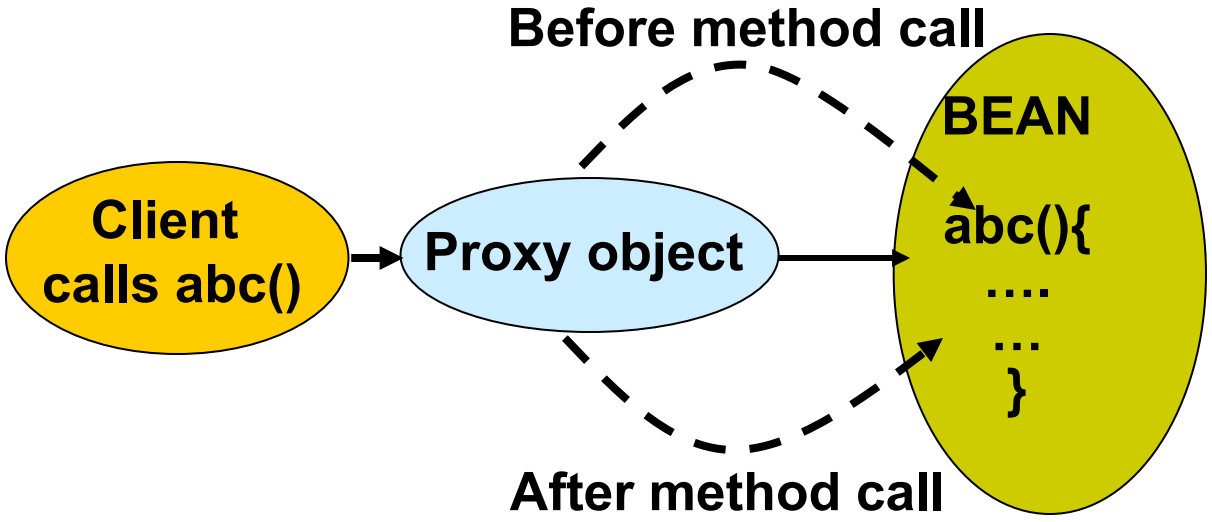
AOP Terminology

- **Aspect :**
  - the cross-cutting functionality being implemented
- **Advice :**
  - the actual implementation of aspect that is advising your application of a new behavior. It is inserted into application at joinpoints
- **Join-point :**
  - a point in the execution of the application where an aspect can be plugged in
- **Point-cut :**
  - defines at what joinpoints an advice should be applied
- **Target :**
  - the class being advised
- **Proxy :**
  - the object created after applying advise to the target



AOP Terminology

- **Types of advices:**
  - Before advice
  - After advice
  - After-returning advice
  - Around advice
  - After-throwing advice



AOP Frameworks

- **AOP support in Spring borrows a lot from the AspectJ project.**
- **Spring supports AOP in the following four flavors:**
  - Classic Spring proxy-based AOP
  - @AspectJ annotation-driven aspects
  - Schema-based AOP support

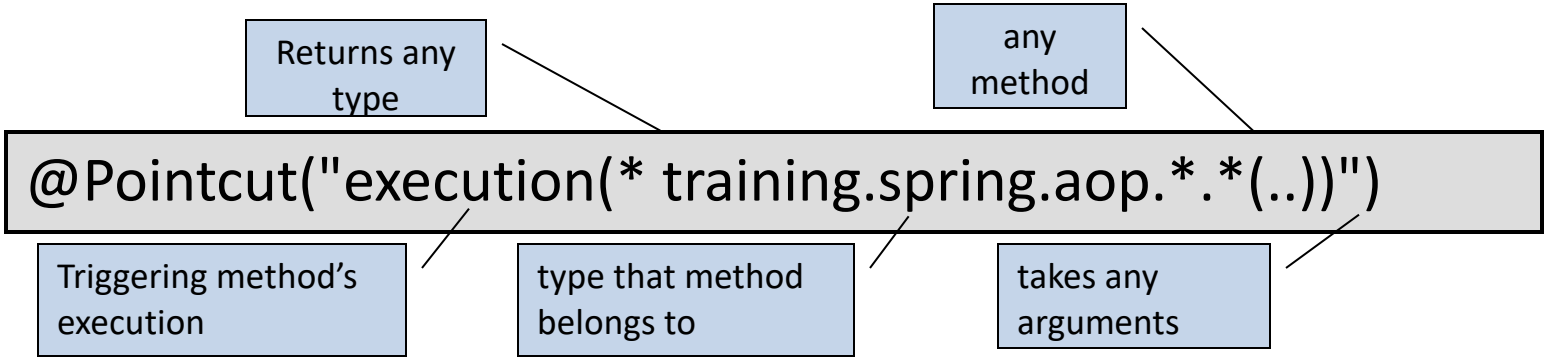
- **Key points of Spring's AOP framework:**
  - All advices are written in Java
  - Spring advises objects at runtime
  - Spring's AOP support is limited to method interception

AspectJ's pointcut expression

- **AspectJ pointcut designators supported in Spring AOP**

AspectJ	Description
args()	Limits matching to the execution of methods whose arguments are instances of the given types
@args()	Limits matching to the execution of methods whose arguments are annotated with the given annotation types
execution()	Matches join points that are method executions
this()	Limits matching to those where the bean reference of the AOP proxy is of a given type
target()	Limits matching to those where the target object is of a given type
@target	Limits matching to join points where the class of the executing object has an annotation of the given type
within()	Limits matching to join points within certain types
@within	Limits matching to join points within types that have the given annotation
@annotation	Limits matching to those where the subject of the join point has the given annotation

- **Writing pointcuts**

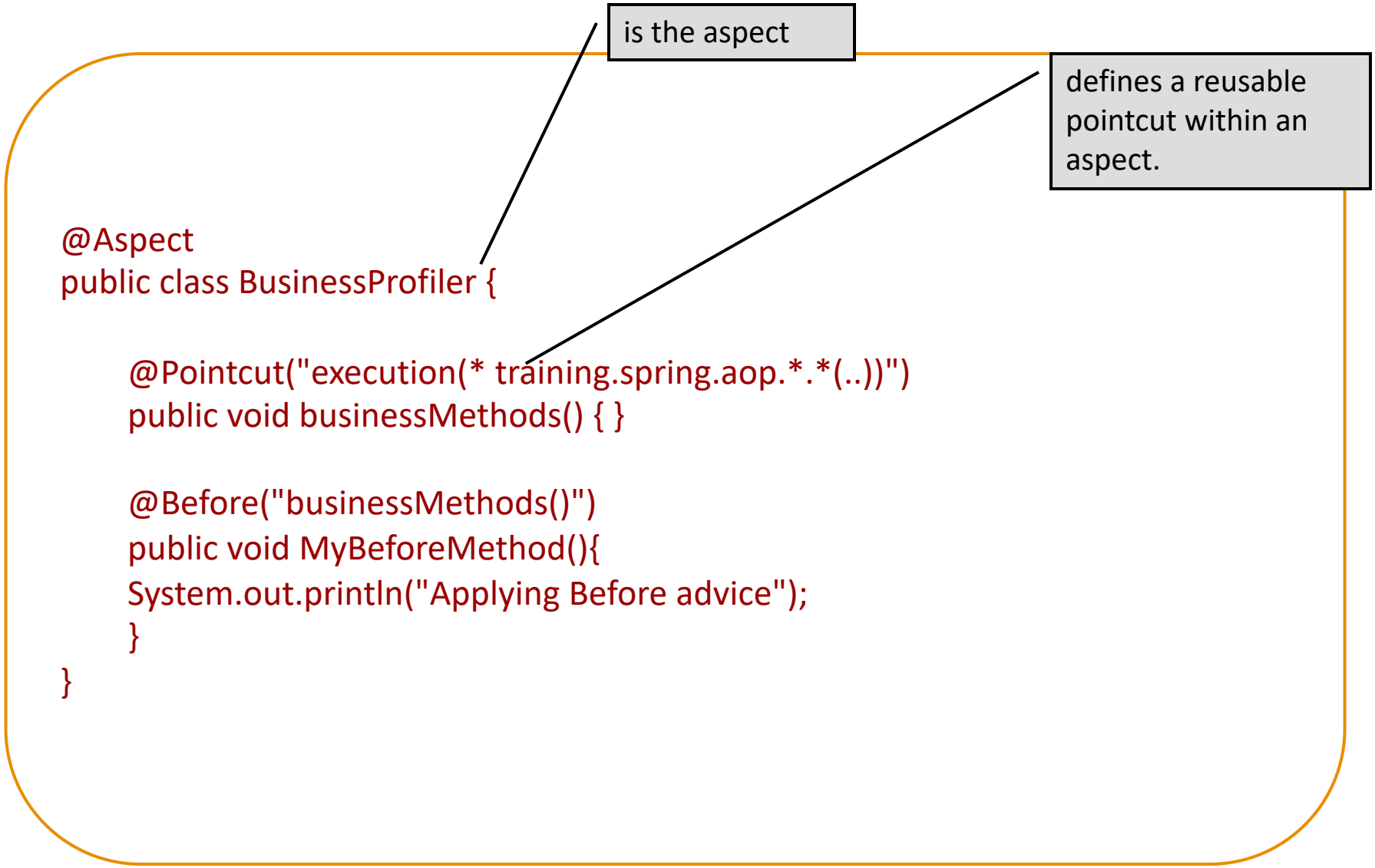


Spring's @AspectJ

```
package training.spring.aop;;
public interface Business {
      void doSomeOperation();
}
```

```
package training.spring.aop;
public class BusinessImpl implements Business {
      public void doSomeOperation() {
            System.out.println("I do what I do best, i.e sleep.");
            try { Thread.sleep(2000);
            } catch (InterruptedException e) {
                  System.out.println("How dare you to wake me up?"); }
            System.out.println("Done with sleeping.");
    }}
```

Spring's @AspectJ



Spring's @AspectJ