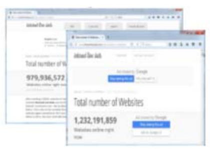
Spring – A Java Web Application Framework

- Introduction to Web Applications
- Introduction to Web Application Frameworks
- Spring – A Java WAF
  - Spring MVC overview
  - Dependency Injection (DI)
  - DI in Spring

We are living in Web era

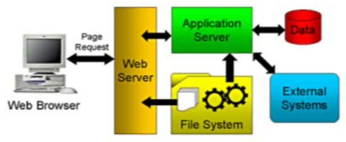


- Millions of websites created every minute
- Cover almost every application, such as news, online shopping, online banking, stock trading, entertainment, educations, government, social networks, etc.

Web application

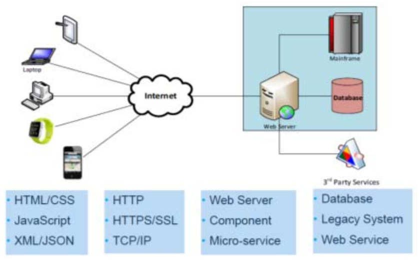
- A **web application** is a client–server computer program which the client (including the user interface and client-side logic) runs in a web browser. Common web applications include webmail, online retail sales, online auctions, wikis, instant messaging services and many other functions.
  - from https://en.wikipedia.org/wiki/Web_application
  - (instead of loading static web pages, run Java/Flash etc.)
- Client software is downloaded to the client machine when visiting a web page (e.g. using HTTP)
  - Software updates happen whenever browser is refreshed

But there are more behind the browsers

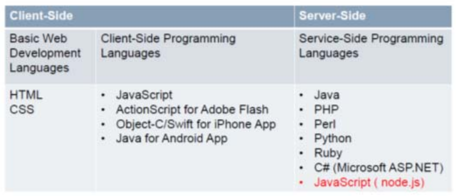


- These browser programs must be developed, stored and downloaded from the server-side via a web server;
- Many 'dynamic' contents/actions have to computed/generated on fly, such as shopping carts, making a order and a receipt after a payment.
- Some applications needs to get data from database, and even external systems at runtime.

It involves a lot of technologies

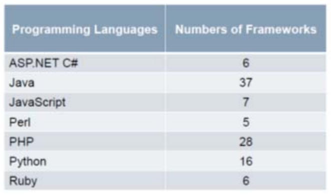


A lot of programming languages



Client-Side		Server-Side
Basic Web Development Languages	Client-Side Programming Languages	Service-Side Programming Languages
HTML CSS	• JavaScript • ActionScript for Adobe Flash • Object-C/Swift for iPhone App • Java for Android App	• Java • PHP • Perl • Python • Ruby • C# (Microsoft ASP.NET) • JavaScript (node.js)

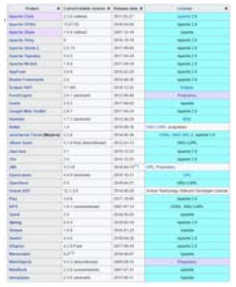
A lot of web application frameworks



Programming Languages	Numbers of Frameworks
ASP.NET C#	6
Java	37
JavaScript	7
Perl	5
PHP	28
Python	16
Ruby	6

- https://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks

Java Web Application Frameworks



Why Spring?

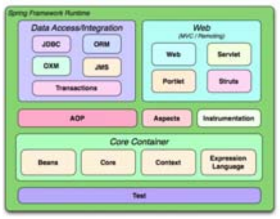
- Widely adopted, popular for Enterprise Java
- High performance, easy testing, reusable
- Simple, lightweight, organised
- Organised in modular fashion
- Built in unit testing
- Supports Model-View Control
- Good support of transactions
- Leverages other technology (JavaEE, hibernate,…)
- Active development
- Growing new projects

Spring projects/extensions

History of the Spring Framework

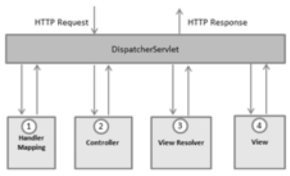
- 2002 Spring was first published by Rod Johnson in his book "Expert one-on-one J2EE Design and Development".
- 2003 Spring was first released as a framework under the Apache 2.0 license
- 2004 Spring v1.0 released
- 2005 Being recognized as a leading full-stack Java/J2EE application framework
- 2006 Spring v2.0 released
- 2009 Spring v3.0 released
- 2016 Spring v4.3 released
- 2017 Spring v5.0

Overview of the Spring Framework



https://docs.spring.io/spring/docs/5.0.0.RC2/spring-framework-reference/overview.html

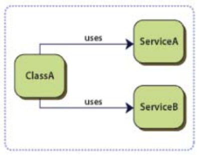
Spring Web MVC



- Browser sends request, received by DispatcherServlet(FC -Front Controller)
- Controller selected by HandlerMapping
- FC requests controller
- Controller returns ModelAndView
- If ModelAndViewcontains logical name for a View, FC queries viewResolverfor the View object that will render response
- FC requests the View Object.

https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm

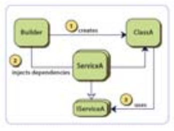
Dependency Injection - Problem



- To replace/update the dependencies, you must change your classes' source code.
- The concrete implementation of the dependencies must be available at compile time.
- Your classes are difficult to test in isolation because they have a direct reference to their dependencies. This means that these dependencies cannot be replaced with stubs or mocks.
- Your classes contain repetitive code for creating, locating, and managing their dependencies.

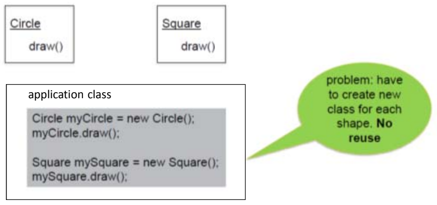
Source: https://msdn.microsoft.com/en-us/library/ff921152.aspx

Dependency Injection - What we want



- Decouple a class from its dependencies
  - dependencies can be replaced or updated with minimal or no changes to your classes' source code
- Don't have to know the dependencies implementation at the compile-time
- Be able to test with no dependence or using difference dependencies
- Removing the responsibility for managing the location & life cycle of dependencies

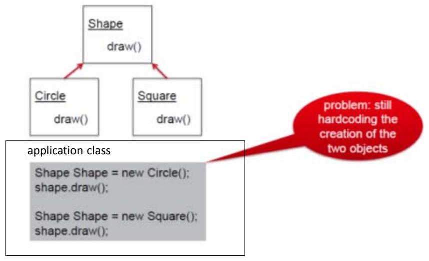
OO Refresh



```
Circle myCircle = new Circle();
myCircle.draw();

Square mySquare = new Square();
mySquare.draw();
```

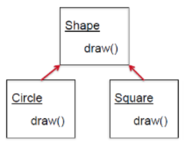
Class inheritance

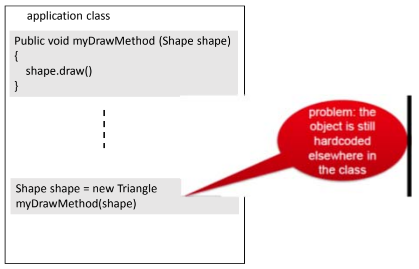


```
Shape Shape = new Circle();
shape.draw();

Shape Shape = new Square();
shape.draw();
```

Method Parameter

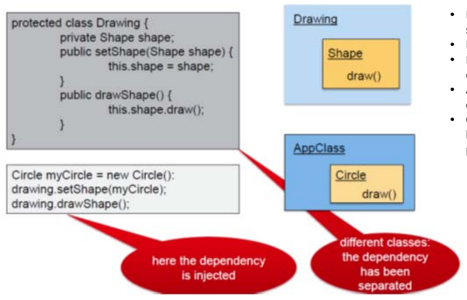



application class
problem: the object is still hardcoded elsewhere in the class

```
Public void myDrawMethod (Shape shape)
{
   shape.draw()
}
```

```
Shape shape = new Triangle
myDrawMethod(shape)
```

What we want


Drawing
Shape
draw()
AppClass
Circle
draw()
here the dependency is injected
different classes: the dependency has been separated

```
protected class Drawing {
        private Shape shape;
        public setShape(Shape shape) {
                this.shape = shape;
        }
        public drawShape() {
                this.shape.draw();
        }
}
```

```
Circle myCircle = new Circle();
drawing.setShape(myCircle);
drawing.drawShape();
```

- Drawing class does not know about shape, only application class.
- However, it can accept any shape.
- Drawing class assumes something else will initialise it
- Application class is what creates the circle (not drawing)
- Can easily add new shapes, don't have to modify drawing class, just pass in new shape
  - Reason: dependency of drawing class to shape object not owned by drawing class
    - Instead dependency is injected by external entity

POJO: Plain Old Java Object

```
Import java.io.Serializable;
Public class Product implementsSerializable
{
        private String description;
        private Double price;

        public void setDescription(String s) {
                description = s;
        }
        public String getDescription() {
                return description;
        }
        public void setPrice(Double d) {
                price = d;
        }
        public Double getPrice() {
                return price;
        }
}
```

Dependency Injection

```
<beans>
   <bean id="product1" class="bus.Product">
      <property name="description">
         <value>Lamp</value>
      </property>
      <property name="price">
         <value>5.75</value>
      </property>
   </bean>
   <bean id="product2" class="bus.Product">
      <property name="description">
         <value>Table</value>
      </property>
      <property name="price">
         <value>75.75</value>
      </property>
   </bean>
</beans>
```

Types of Dependency Injection

- **Constructor-based dependency injection**
  - Container invokes a class constructor with a number of arguments, each representing a dependency on other class.
- **Setter-based dependency injection**
  - Container calls setter methods on beans after invoking a no-argument constructor or no-argument static factory method to instantiate the bean.
- **Interface injection**
  - The dependency provides an injector method that will inject the dependency into any client passed to it. Clients must implement an interface that exposes a setter methodthat accepts the dependency

Constructor-based Dependency Injection

```
public class Foo {
        public Foo(int year, String name)
        { // ... }
}
```

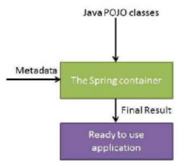
```
<beans>
  <bean id = "exampleBean" class = "examples.ExampleBean">
    <constructor-arg type = "int" value = "2001"/>
    <constructor-arg type = "java.lang.String" value = "Zara"/>
  </bean>
</beans>
```

Setter-based dependency injection

```
public class ExampleBean {
   private AnotherBean beanOne;
   private YetAnotherBean beanTwo;
   public void setBeanOne(AnotherBean beanOne) {
      this.beanOne = beanOne;
   }
   public void setBeanTwo(YetAnotherBean beanTwo) {
      this.beanTwo = beanTwo;
   }
}
```

```
<bean id="exampleBean" class="examples.ExampleBean">
   <property name="beanOne">
      <ref bean="anotherExampleBean"/>
   </property>
   <property name="beanTwo" ref="yetAnotherBean"/>
</bean>
```

Java Containers

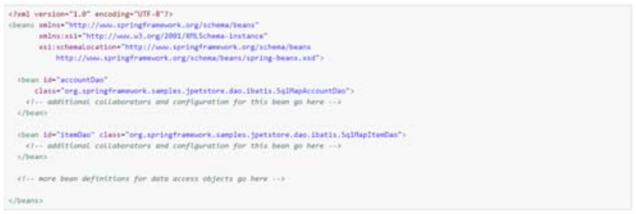


- Java containers manage Java objects: instantiation and lifecycle
- In this course, you will use :
  - Tomcat servlet container: hosts and processes web pages, such as HTML, JSP,etc.
  - Spring beans container: A bean is any *Plain Old Java Object* (POJO) , which can be used for:
    - Business components/services, such as AccountBean, BookingBeanetc.; or
    - Data Objects, such as Product, Contract etc.

Instantiating a Container

```
ApplicationContext context =
   new ClassPathXmlApplicationContext(new String[]
{"services.xml", ……});
```

An Example



```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean id="accountDao"
      class="org.springframework.samples.jpetstore.dao.ibatis.SqlMapAccountDao">
    <!-- additional collaborators and configuration for this bean go here -->
  </bean>

  <bean id="itemDao" class="org.springframework.samples.jpetstore.dao.ibatis.SqlMapItemDao">
    <!-- additional collaborators and configuration for this bean go here -->
  </bean>

  <!-- more bean definitions for data access objects go here -->

</beans>
```

More on Spring

- AOP –AspectJ/separation of concerns (Self-Study)
- Hibernate and ORM (next week)
- Spring-social, Spring-security, ... (based on your project)
- Framework comparison (next term)

Spring Resources

- https://spring.io/docs
- https://www.tutorialspoint.com/spring/index.htm
- http://www.springbyexample.org/
- http://www.freebookcentre.net/JavaTech/Free-Java-Spring-books-download.html
- http://learneasyspring.blogspot.com/2013/06/dependency-injection.html

Tutorial

- Start the 4-Part Spring MVC tutorial
  - Get set up on your own PC!

Basic Spring MVC Application

- Set up webserver
  - https://www.youtube.com/watch?v=n14rpj_08wM
- Set up spring web application
  - https://www.youtube.com/watch?v=S5cbm6SDyvU&feature=youtu.be