WEB APPLICATION PROGRAMMING

Instructor: Yusuf Uğur SOYSAL
@n11.com

JAVA RECAP

Object Oriented Programming

classes: A class is a collection of data fields that hold values and methods that operate on those values. A class defines a **type**.

```
class Dog{
    String breed;
    int age;
    String color;

    void barking(){
    }

    void hungry(){
    }

    void sleeping(){
    }
}
```

Object Oriented Programming

object: An object is an instance of a class

```
Dog myDog = new Dog();
```

subclass:

```
class PaymentEntity{
    void pay(){
    }
}

class 3DPaymentEntity extends PaymentEntity{
    void checkConfirmationCode(){
    }
}
```

Object Oriented Programming

abstract: Any class with an abstract method is automatically abstract itself and must be declared as such. An abstract class cannot be instantiated.

```
abstract class PaymentEntity{
    abstract void pay(){
    }
}

class 3DPaymentEntity extends PaymentEntity{
    void pay(){
    }

    void checkConfirmationCode(){
    }
}
```

Object Oriented Programming

interface: An interface definition is much like a class definition in which all the (nondefault) methods are abstract

```
interface Centered {
  void setCenter(double x, double y);

  default void sort(Comparator<Centered> comparator) {
    Collections.sort(this, comparator);
  }
}

public class Circle implements Centered {
  private double centerX, centerY;

  public void setCenter(double x, double y) {
    centerX = x;
    centerY = y;
  }
}
```

Constructor / Destructor

```
public class Circle {
    protected double r;

    public Circle(double r) {
        this.r = r;
    }
}
```

No Destructor exists!

However, a finalization method exists (YOU WILL NOT NEED THIS - EVER!)

```
public class Circle {
    protected void finalize() throws Throwable {

    }
}
```

Modifiers

public / protected / private / package private

static

final

transient

volatile

Packages

A package is a named collection of classes, interfaces, and other reference types.

Packages serve to group related classes and define a namespace for the classes they contain.

The core classes of the Java platform are in packages whose names begin with java.

Extensions to the Java platform that have been standardized and typically have package names that begin with javax.

```
package org.apache.commons.net;

import java.io.File;

import java.io.*;

import java.util.List;
import java.awt.List; // compile error!!

import static java.lang.System.out;
```

Main Method

To create a program, you must define a class that has a special method with the following signature:

```
public static void main(String[] args) { }
```

This main() method is the main entry point for your program. It is where the Java interpreter starts running.

```
java -classpath /home/yusuf com.yusuf.TestApp argument1 argument2
```

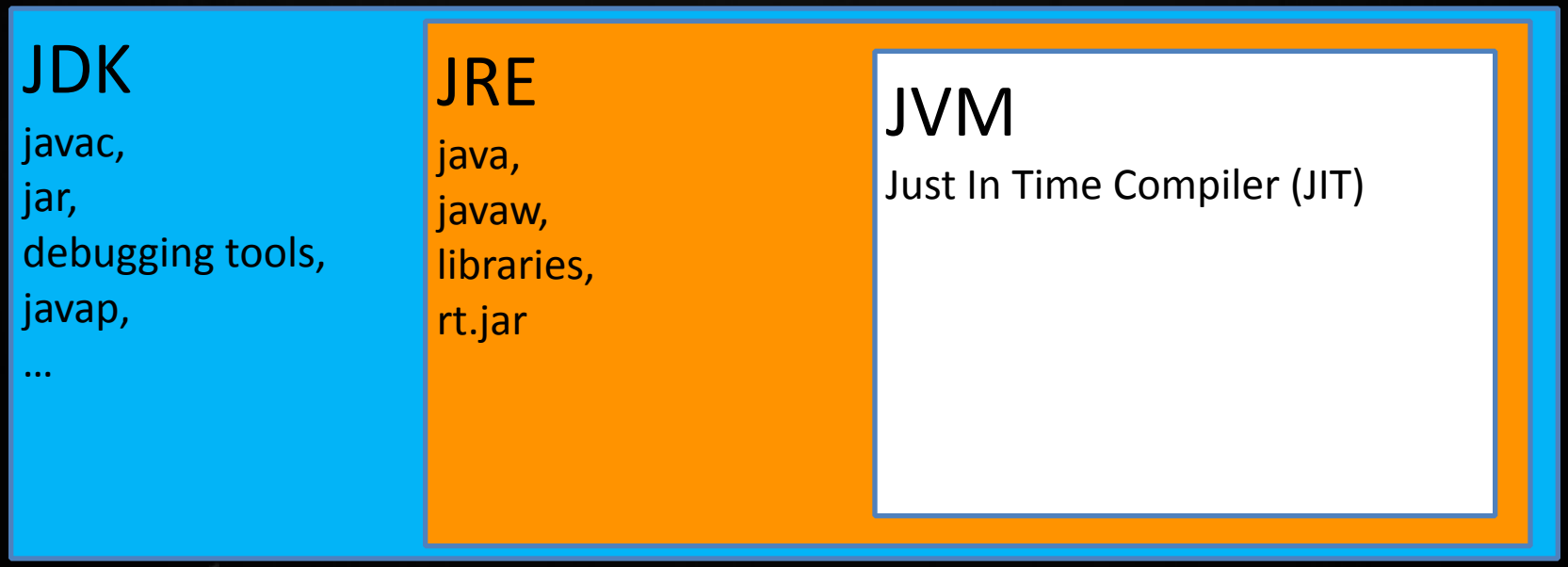
You can create a JAR (Java ARchive) file and aggregate all your class files into one.

JVM, JDK and JRE

JVM: Java Virtual Machine
- **Interprets byte code into machine code**
- **Responsible for Garbage Collection**
- **Platform Dependent**

JRE: Java Runtime Environment
- **Contains JVM and other class libraries**

JDK: Java Development Kit
- **Contains JRE and tools needed to develop Java applications**



Garbage Collections

Memory occupied by an object is automatically reclaimed when the object is no longer needed.

Memory Leaks??

Variable Scope

Instance Variables
Parameter Variables
Local Variables

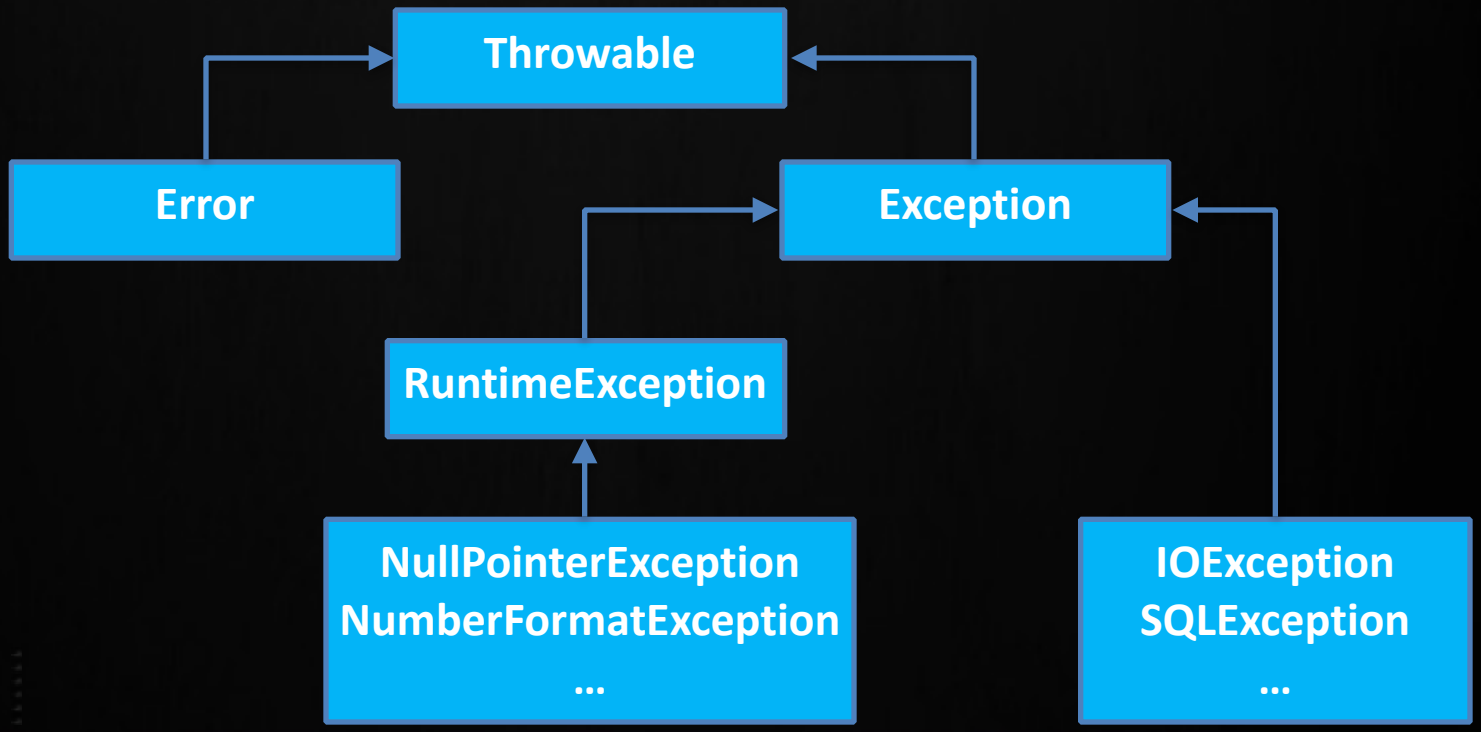
```
public class TwoSides {
    int side1, side2 ;
    public boolean testRightTriangle( int hypoteneuse ) {
        int side1Squared = side1 * side1;
        int side2Squared = side2 * side2;
        int hypSquared = hypoteneuse * hypoteneuse;

        return side1Squared + side2Squared == hypSquared;
    }
}
```

Pass by value or pass by reference?

Exception Handling

Checked Exceptions
Unchecked Exceptions

```
try {
  someMethodThatMightThrow();
} catch(Exception e){

}
```



Generics

```
List circles = new ArrayList();

circles.add(new Circle());
circles.add(new Dog());

Circle circle1 = (Circle) circles.add.get(0);
Circle circle2 = (Circle) circles.add.get(1);

//...

List<Circle> circles = new ArrayList<>();
```

Compile Time Checking
Type Erasure

Enums

Enums are a variation of classes that have limited functionality and that have only a small number of possible values that the type permits.

```
public enum PrimaryColor {
  // The ; is not required at the end of the list of
instances
  RED, GREEN, BLUE
}
```

Annotations

Annotations are a specialized kind of interface that annotate some part of a Java program.

```
@Override
@SuppressWarnings
@Deprecated
@NotNull
```

Lambda Expressions

Lambdas allow small bits of code to be written inline as literals in a program and facilitate a more functional style of programming Java.

```
(p, q) -> { /* method body */ }
```

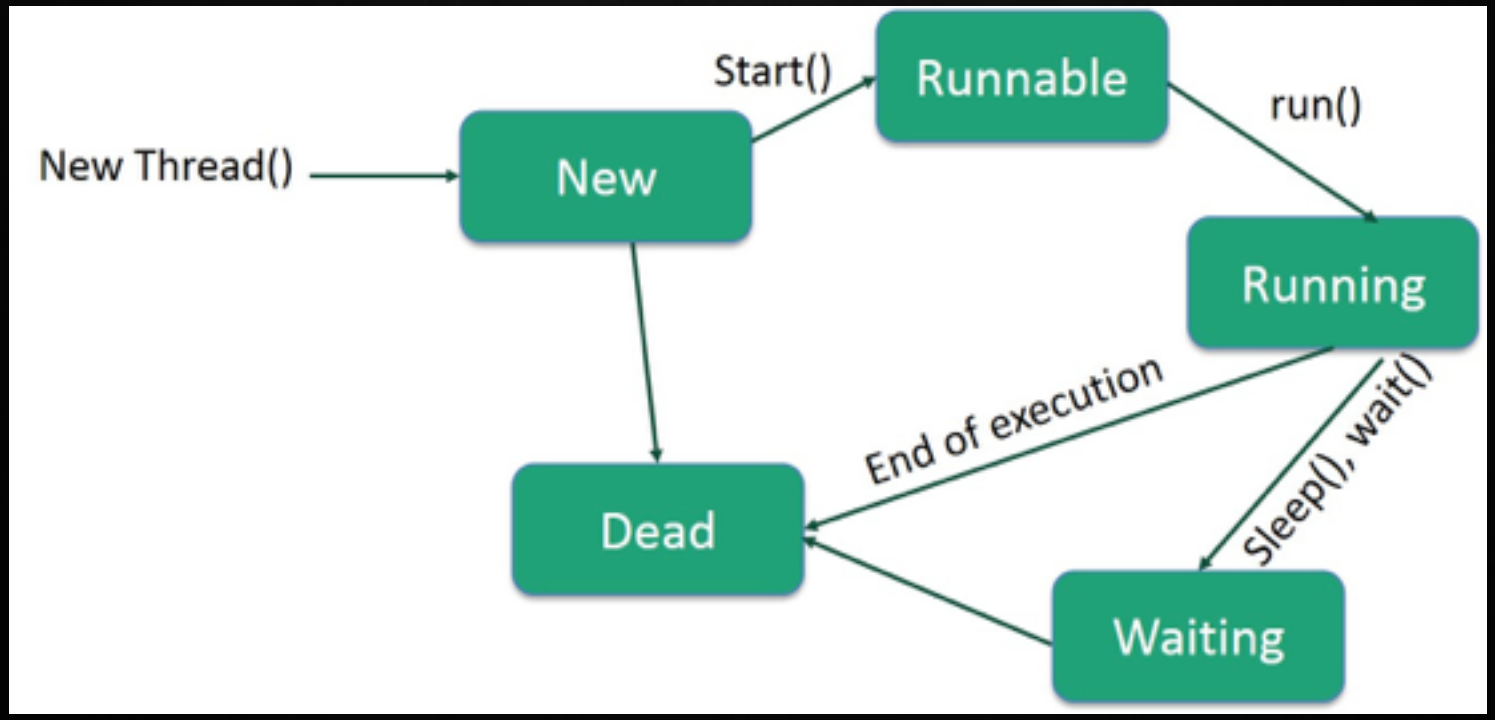
```
File dir = new File("/src");

String[] filelist = dir.list(new FilenameFilter() {
  public boolean accept(File f, String s) {
     return s.endsWith(".java");
  }
});
```

```
File dir = new File("/src");
String[] filelist = dir.list(
     (f,s) -> { return s.endsWith(".java"); }
);
```

Multithreading

Implementing Runnable Interface
Extending Thread Class

Never call run method directly
Always use a thread pool



Servlets

A servlet is a Java program.

A servlet application runs inside a servlet container and cannot run on its own.

A servlet instance is shared by all users in an application.

```
void init(ServletConfig config) throws ServletException

void service(ServletRequest request, ServletResponse
response)
        throws ServletException, java.io.IOException

void destroy()

java.lang.String getServletInfo()

ServletConfig getServletConfig()
```

HTTPServlet

doGet *
doPost *
doHead
doPut
doTrace
doOptions
doDelete

HTTPServletRequest
HTTPServletResponse

Deployment Descriptor

WEB-INF/web.xml

Configures application
Defines servlets and mappings
Specifies Filters, Error Pages, …

Annotating classes are also possible:
- @WebServlet

JavaServer Pages

A JSP page is essentially a servlet.

Basically text files with jsp extension.

The <% … %> block is called a scriplet.

```
<%@page import="java.util.Date"%>
<%@page import="“java.text.DateFormat"%>

<html>
<head><title>Today's date</title></head>
<body>
<%
    DateFormat dateFormat =
            DateFormat.getDateInstance(DateFormat.LONG);
    String s = dateFormat.format(new Date());
    out.println("Today is " + s);
%>
</body>
</html>
```

JavaServer Pages - Implicit Objects

request: javax.servlet.http.HttpServletRequest
response: javax.servlet.http.HttpServletResponse
out: javax.servlet.jsp.JspWriter
session: javax.servlet.http.HttpSession
application: javax.servlet.ServletContext
config: javax.servlet.ServletConfig
pageContext: javax.servlet.jsp.PageContext
page: javax.servlet.jsp.HttpJspPage
exception: java.lang.Throwable

JavaServer Pages - Include

Compile Time

```
<html>
<head><title>Including a file</title></head>
<body>
This is the included content: <hr/>
<%@ include file="copyright.jspf"%>
</body>
</html>
```

Request Time

```
<html>
<head>
<title>Include action</title>
</head>
<body>
<jsp:include page="jspf/menu.jsp">
    <jsp:param name="text" value="How are you?"/>
</jsp:include>
</body>
</html>
```

JavaServer Pages - Expressions

```
<%=java.util.Calendar.getInstance().getTime()%>
```

JavaServer Pages - Declarations

```
<%!
    public java.util.Date getTodaysDate() {
        return new java.util.Date();
    }
%>

<html>
<head><title>Declarations</title></head>
<body>
Today is <%=getTodaysDate()%>
</body>
</html>
```

JavaServer Pages - Tag Files

Placed in WEB-INF/tags directory

```
<%@ tag import="java.util.Date"
import="java.text.DateFormat"%>
<%
    DateFormat dateFormat =
            DateFormat.getDateInstance(DateFormat.LONG);
    Date now = new Date(System.currentTimeMillis());
    out.println(dateFormat.format(now));
%>
```

```
<%@ taglib prefix="easy" tagdir="/WEB-INF/tags" %>
Today is <easy:firstTag/>
```

WHEN IN DOUBT
GOOGLE IT!

INTRODUCTION TO SPRING

What is Spring Framework?

A platform that provides infrastructure support for developing Java applications.

Enables to build applications from “Plain Old Java Objects (POJO)”.

Has minimal external dependencies.

Has a lot of build in modules which uses existing technologies.

Inversion of Control (IoC)

IoC container manages java objects - from initialization to destruction.

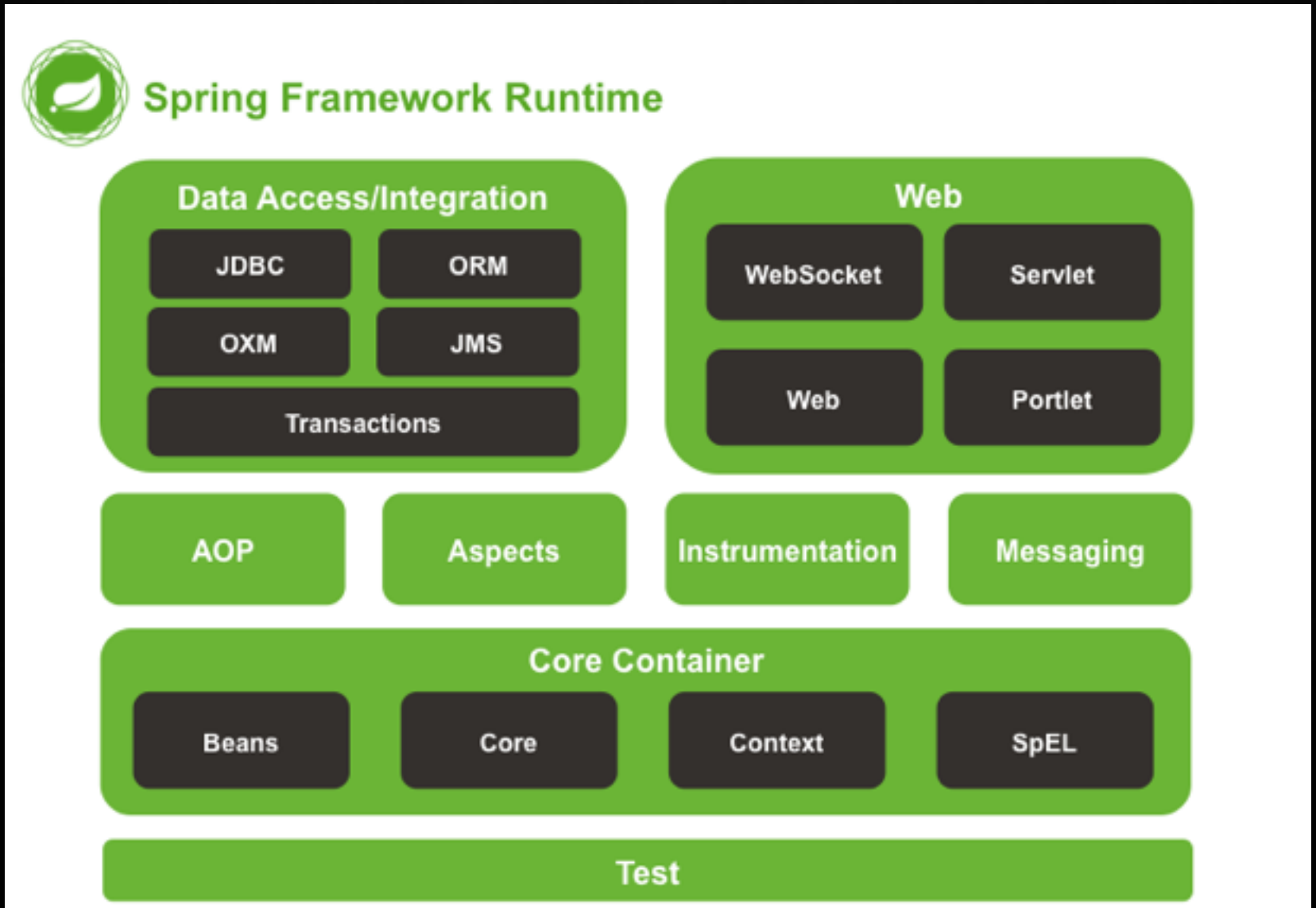
It is done via a “BeanFactory” - What is a Factory?

IoC container manages bean’s scope and lifecycle.

Spring framework enforces the dependency injection pattern.

Dependency injection allows to write loosely coupled classes

Core Spring Runtime



BeanFactory

BeanFactory provides basis for IoC functionality.

ApplicationContext is widely used instead.

Feature	*BeanFactory*	*ApplicationContext*
Bean installation/wiring	*Yes*	*Yes*
Automatic BeanPostProcessor registration	*No*	*Yes*
Automatic BeanFactoryPostProcessor registration	*No*	*Yes*
Convenient MessageSource access (for i18n)	*No*	*Yes*
ApplicationEvent publishing	*No*	*Yes*

Configuration Metadata

Spring consumes a form of configuration metadata.

It represents how the application should be initialized.

- XML based configuration
- Annotation based configuration
- Java based configuration

Beans

Objects created by Spring container.

They are POJOs.

Created via configuration metadata.

XML Based Configuration

```xml
<?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="..." class="...">
    </bean>

    <bean id="..." class="...">
    </bean>

</beans>
```

Using the IoC Container

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

PetStoreService service = context.getBean("petStore", PetStoreService.class);
```

Bean Names

```
<?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="bean1" class="...">
    </bean>

    <bean name="bean2" class="...">
    </bean>

    <bean name="bean3,bean3Alias" class="...">
    </bean>

</beans>
```

Initializing Beans

```
<bean id="exampleBean" class="examples.ExampleBean"/>

<bean id="clientService" class="examples.ClientService"
          factory-method="createInstance"/>
```

```
public class ClientService {
    private static ClientService clientService = new
ClientService();
    private ClientService() {}

    public static ClientService createInstance() {
        return clientService;
    }
}
```

Bean Dependencies

```
public class Product {
    private String name;
    private float price;

    public Product(String name, float price) {
        this.name = name;
        this.price = price;
    }
}
```

```
<bean name="featuredProduct" class="Product">
    <constructor-arg name="name" value="Keyboard"/>
    <constructor-arg name="price" value="89.99"/>
</bean>

……

<bean id="productName" class="java.lang.String">
    <constructor-arg value="Keyboard"/>
</bean>

<bean name="featuredProduct" class="Product">
    <constructor-arg name="name" ref="productName" />
    <constructor-arg name="price" value="89.99" />
</bean>
```

Bean Dependencies

```
public class Product {
    private String name;

    public void setName(String name) {
        this.name = name;
    }
}
```

```
<bean name="featuredProduct" class="Product">
    <property name="name" value="Keyboard"/>
</bean>
```

Circular Dependencies?

BeanCurrentlyInCreationException

Inner Beans

```
<bean id="outer" class="Product">
    <property name="variants">
        <bean class="Sku">
            <property name="name" value="..."/>
        </bean>
    </property>
</bean>
```

Collections

```
<bean id="moreComplexObject" class="example.ComplexObject">
    <property name="adminEmails">
        <props>
            <prop key="administrator">administrator@example.org</prop>
            <prop key="support">support@example.org</prop>
            <prop key="development">development@example.org</prop>
        </props>
    </property>
    <property name="someList">
        <list>
            <value>a list element followed by a reference</value>
            <ref bean="myDataSource" />
        </list>
    </property>
    <property name="someMap">
        <map>
            <entry key="an entry" value="just some string"/>
            <entry key="a ref" value-ref="myDataSource"/>
        </map>
    </property>
    <property name="someSet">
        <set>
            <value>just some string</value>
            <ref bean="myDataSource" />
        </set>
    </property>
</bean>
```

Lazy Initialize

```
<bean id="lazy" class="com.foo.ExpensiveToCreateBean" lazy-init="true"/>
```

Bean Scopes

singleton (Default)
prototype
request (valid for HTTP)
session (valid for HTTP Session)
global session (valid for Portlets)
application (valid for ServletContext)

Bean Lifecycle Callbacks

- org.springframework.beans.factory.InitializingBean => afterPropertiesSet()
  @javax.annotation.PostConstruct
  “init-method”

- org.springframework.beans.factory.DisposableBean => destroy()
  @javax.annotation.PreDestroy
  “destroy-method”

BeanPostProcessor

org.springframework.beans.factory.config.BeanPostProcessor

Defines callback methods that you can implement to provide your own instantiation logic

```
public class BeanPostProcessorExample
        implements BeanPostProcessor {

    public Object postProcessBeforeInitialization(Object bean, String
beanName) throws BeansException {
        return bean;
    }

    public Object postProcessAfterInitialization(Object bean, String beanName)
throws BeansException {
        System.out.println("Bean '" + beanName + "' created : " +
bean.toString());
        return bean;
    }

}
```

PropertyPlaceholderConfigurer

Externalize property values from a bean definition in a separate file.

Uses Java Properties format

Customize environment-specific properties

```
<bean
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"
>
    <property name="locations" value="classpath:com/foo/jdbc.properties"/>
</bean>

<bean id="dataSource" destroy-method="close"
        class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="${jdbc.driverClassName}"/>
    <property name="url" value="${jdbc.url}"/>
    <property name="username" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>
</bean>
```

```
jdbc.driverClassName=org.hsqldb.jdbcDriver
jdbc.url=jdbc:hsqldb:hsql://production:9002
jdbc.username=sa
jdbc.password=root
```

Annotation Based Configuration

Relies on Bytecode Metadata

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

    <context:annotation-config/>

</beans>
```

@Required

```
public class MyObject {

    private MyDependency myDependency;

    @Required
    public void setMyDependency(MyDependency myDependency) {
        this.myDependency = myDependency;
    }

}
```

@Autowired

```
public class SimpleMovieLister {
    @Autowired
    private MovieListHolder movieListHolder;
    private MovieFinder movieFinder;
    private MovieCatalog movieCatalog;
    private CustomerPreferenceDao customerPreferenceDao;

    @Autowired
    public void setMovieFinder(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

    @Autowired
    public void prepare(MovieCatalog movieCatalog,
            CustomerPreferenceDao customerPreferenceDao) {
        this.movieCatalog = movieCatalog;
        this.customerPreferenceDao = customerPreferenceDao;
    }

}
```

@Autowired

```
public class SimpleMovieLister {
    @Autowired(required=false)
    private MovieListHolder movieListHolder;
}
```

Component Scan

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

    <context:component-scan base-package="org.example"/>

</beans>
```

```xml
<beans>
    <context:component-scan base-package="org.example">
        <context:include-filter type="regex"
                expression=".*Stub.*Repository"/>
        <context:exclude-filter type="annotation"
                expression="org.springframework.stereotype.Repository"/>
    </context:component-scan>
</beans>
```

Annotations

Stereotype Annotations
- @Component
- @Controller
- @Service

@Repository

@Scope(“prototype”)

@Value("${jdbc.url}")

Java Based Configuration

```
@Configuration
public class AppConfig {
    @Bean
    public MyService myService() {
        return new MyServiceImpl();
    }
    @Bean
    public MyBean myBean(MyService myService) {
        return new MyBeanImpl(myService);
    }
}
```

```
public static void main(String[] args) {
    ApplicationContext ctx = new
AnnotationConfigApplicationContext(AppConfig.class);
    MyService myService = ctx.getBean(MyService.class);
}
```

Component Scan with @Configuration

```
@Configuration
@ComponentScan(basePackages = "com.foo")
public class AppConfig  {
    ...
}
```

Spring MVC

Designed around a DispatcherServlet

Based on @Controller and @RequestMapping annotations

You can use any object as form backing object - no framework specific interface

Data is cast automatically to the right type

Controller are not coupled with views. Clear separation of roles.

Handler mappings and view resolutions are customisable

DispatcherServlet

It is an actual Servlet

Dispatches requests to controllers

Completely integrated with Spring IoC container

```
<web-app>
    <servlet>
        <servlet-name>example</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</
servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>example</servlet-name>
        <url-pattern>/example/*</url-pattern>
    </servlet-mapping>

</web-app>
```

DispatcherServlet

```
public class MyWebApplicationInitializer implements WebApplicationInitializer
{

    @Override
    public void onStartup(ServletContext container) {
        ServletRegistration.Dynamic registration =
container.addServlet("dispatcher", new DispatcherServlet());
        registration.setLoadOnStartup(1);
        registration.addMapping("/example/*");
    }

}
```

DispatcherServlet

WEB-INF/[servlet name]-servlet.xml

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        classpath*:/appContext/appContext-*.xml,
    </param-value>
</context-param>
```

Controllers

Provides access to application behaviour

Interprets user input and transform them into a model

```
@Controller
public class HelloWorldController {

    @RequestMapping("/helloWorld")
    public String helloWorld(Model model) {
        model.addAttribute("message", "Hello World!");
        return "helloWorld";
    }
}
```

http://url/helloWorld

!! @RequestMapping maps all HTTP methods by default !!

Controllers - @RequestMapping

```
@Controller
@RequestMapping("/hesabim/yorumlarim-incelemelerim")
public class ProductReviewsController {

    @Autowired
    private BuyerVerificationService buyerVerificationService;

    @RequestMapping(method = RequestMethod.GET)
    public ModelAndView showAllOrderItems() {
        return ...;
    }

    @RequestMapping(value = "/yeni-inceleme", method = RequestMethod.POST)
    public ModelAndView saveProductReview() {
        return ...;
    }

}
```

http://url/hesabim/yorumlarim-incelemelerim

http://url/hesabim/yorumlarim-incelemelerim/yeni-inceleme

Controllers - URI Templates

URI templates can be used for convenient access to selected parts of a URL in a @RequestMapping method.

```
@RequestMapping(path="/owners/{ownerId}", method=RequestMethod.GET)
public String findOwner(@PathVariable String ownerId) {
    return "....";
}
```

http://url/owners/yusuf

```
@RequestMapping(path="/owners/{ownerId}", method=RequestMethod.GET)
public String findOwner(@PathVariable("ownerId") String theOwner) {
    return "....";
}

@RequestMapping(path="/owners/{ownerId}/pets/{petId}",
                method=RequestMethod.GET)
public String findPet(@PathVariable String ownerId,
                      @PathVariable String petId,
                      Model model) {
    ...
}
```

Controllers - URI Templates

```
@RequestMapping("/spring-web/{symbolicName:[a-z-]+}-{version:\\d\\.\\d\\.\\d}
{extension:\\.[a-z]+}")
public void handle(@PathVariable String version,
                   @PathVariable String extension) {
    // ...
}
```

```
@RequestMapping(“/spring-web/*")
public void handle() {
    // ...
}
```

```
@RequestMapping(“/spring-web/**”)
public void handle() {
    // ...
}
```

Controllers - Media Types

```
@RequestMapping(path = "/pets",
                method = RequestMethod.POST,
                consumes="application/json")
public void addPet(@RequestBody Pet pet) {
    ...
}

@RequestMapping(path = "/pets",
                method = RequestMethod.POST,
                consumes="!text/plain")
public void addPet(@RequestBody Pet pet) {
    ...
}
```

Controllers - Media Types

```
@RequestMapping(path = "/pets/{petId}",
                method = RequestMethod.GET,
                produces="application/json")
@ResponseBody
public Pet getPet(@PathVariable String petId) {
    ...
}
```

Controllers - @RequestParam

```
@RequestMapping(method = RequestMethod.GET)
public String setupForm(@RequestParam("petId") int petId) {
    ...
}
```

```
@RequestMapping(method = RequestMethod.GET)
public String setupForm(@RequestParam(value = “petId”,
required=false) int petId) {
    ...
}
```

Controllers - @RequestBody

```
@RequestMapping(path = "/something", method = RequestMethod.POST)
public void handle(@RequestBody String body){
    ...
}
```

Controllers - @RestController

```
@RestController
public class HelloWorldRestController {

    @RequestMapping("/hello/{player}")
    public Message message(@PathVariable String player) {

        Message msg = new Message(player, "Hello " + player);
        return msg;
    }

}
```

ANY QUESTIONS?
PING ME!

yusuf.soysal@n11.com