Annotations used in RMC

1. @Entity

JPA will include any class annotated with @Entity in the persistence management setup. You don't need persistence.xmlif you use annotations

1. @JsonAutoDetect

We use annotations to describe how to perform the mapping between the JSON data and the Person object. We use @JsonAutoDetect to say that field member names map directly to JSON field names.

1. @XStreamAlias

The idea of XStream is to annotate your domain classes with xstream annotations which map the attributes and references to XML.

<http://xstream.codehaus.org/index.html>

**Java notes:**

Some advance technologies that gaining popularity now a days are

1. JSF
2. ORM frameworks: iBatis, Hibernate
3. Frameworks: Spring
4. AOP:aspectj, springAOP
5. Advance java programming languages like Grails, Ruby

**Eclipse history**

**Release Date Platform version Projects**

Indigo June 2011 3.7 Indigo projects

Helios 23 June 2010 3.6 Helios projects

Galileo 24 June 2009 3.5 Galileo projects

Ganymede 25 June 2008 3.4 Ganymede projects

Europa 29 June 2007 3.3 Europa projects

Callisto 30 June 2006 3.2 Callisto projects

Eclipse 3.1 28 June 2005 3.1

Loath

SOA:

Service-oriented architecture (SOA) is an evolution of distributed computing based on the request/reply design paradigm for synchronous and asynchronous applications. An application's business logic or individual functions are modularized and presented as services for consumer/client applications. What's key to these services is their loosely coupled nature; i.e., the service interface is independent of the implementation. Application developers or system integrators can build applications by composing one or more services without knowing the services' underlying implementations. For example, a service can be implemented either in .Net or J2EE, and the application consuming the service can be on a different platform or language..

Service-oriented architectures have the following key characteristics:

* SOA services have self-describing interfaces in platform-independent XML documents. Web Services Description Language ([WSDL](http://www.javaworld.com/javaworld/jw-06-2005/jw-0613-soa.html#resources)) is the standard used to describe the services.
* SOA services communicate with messages formally defined via XML Schema (also called [XSD](http://www.javaworld.com/javaworld/jw-06-2005/jw-0613-soa.html#resources)). Communication among consumers and providers or services typically happens in heterogeneous environments, with little or no knowledge about the provider. Messages between services can be viewed as key business documents processed in an enterprise.
* SOA services are maintained in the enterprise by a registry that acts as a directory listing. Applications can look up the services in the registry and invoke the service. Universal Description, Definition, and Integration ([UDDI](http://www.javaworld.com/javaworld/jw-06-2005/jw-0613-soa.html#resources)) is the standard used for service registry.
* Each SOA service has a quality of service (QoS) associated with it. Some of the key QoS elements are security requirements, such as authentication and authorization, reliable messaging, and policies regarding who can invoke services.

Asynchronous apps:

Asynchronous APIs do not block. Every synchronous call waits and blocks for your results to come back. This is just a sleeping thread and wasted computation.

If you need something to happen, send of an asynchronous request and do further computation when the request returns. This means your thread sits idle and can pick up other work.

Asynchronous requests is the way to scale to thousands of concurrent users.

Spring Framework

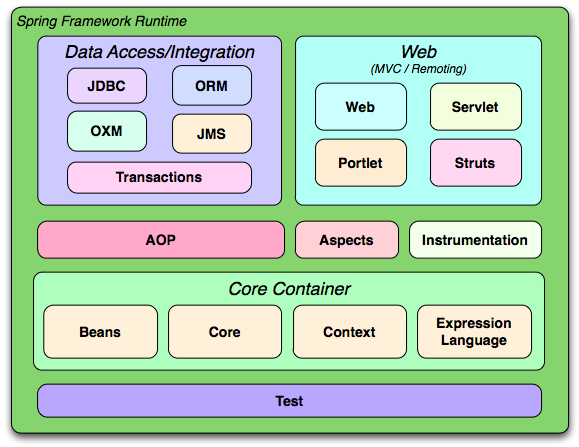
Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure so you can focus on your application.

Spring enables you to build applications from “plain old Java objects” (POJOs) and to apply enterprise services non-invasively to POJOs. This capability applies to the Java SE programming model and to full and partial Java EE.

Examples of how you, as an application developer, can use the Spring platform advantage:

* Make a Java method execute in a database transaction without having to deal with transaction APIs.
* Make a local Java method a remote procedure without having to deal with remote APIs.
* Make a local Java method a management operation without having to deal with JMX APIs.
* Make a local Java method a message handler without having to deal with JMS APIs.

The Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, and Test, as shown in the following diagram.



Overview of the Spring Framework

[Difference between DTO, VO, POJO, JavaBeans?](http://stackoverflow.com/questions/1612334/difference-between-dto-vo-pojo-javabeans)

DTO: "Data transfer objects " can travel between seperate layers in software architecture.

VO: "Value objects " hold a object such as Integer,Money etc.

POJO: Plain Old Java Object which is not a special object.

Java Beans: requires a Java Class to be serializable, have a no-arg constructor and a getter and setter for each field

[difference between java bean and java class?](http://stackoverflow.com/questions/8629173/difference-between-java-bean-and-java-class)

Same mostly

Quality analysis and measurement tools Findbugs, PMD (

* [JarAnalyzer](http://www.kirkk.com/main/Main/JarAnalyzer) ([JarAnalyzer](http://www.kirkk.com/main/Main/JarAnalyzer) – Is a dependency management utility for jar files. It’s primary purpose is to traverse through a directory, parse each of the jar files in that directory, and identify the dependencies between the jar files.

Java design pattern

|  |
| --- |
| [About Design Patterns](http://www.javacamp.org/designPattern/about.html) |
| Creational Patterns |
| ---[Abstract Factory](http://www.javacamp.org/designPattern/abstractfactory.html) ( which is a layer higher than a factory method.) |
| ---[Builder](http://www.javacamp.org/designPattern/builder.html) |
| ---[Factory Method](http://www.javacamp.org/designPattern/factory.html) (like interface to hook subclasses into one) |
| ---[Prototype](http://www.javacamp.org/designPattern/prototype.html) ( which creates a new object by copying an instance, so it reduces subclasses, Overloading method is a kind of prototype too) |
| ---[Singleton](http://www.javacamp.org/designPattern/singleton.html) |
| Structural Patterns |
| ---[Adapter](http://www.javacamp.org/designPattern/adapter.html) |
| ---[Bridge](http://www.javacamp.org/designPattern/bridge.html) |
| ---[Composite](http://www.javacamp.org/designPattern/composite.html) (A component has many elements and itself which has many elements and itself, etc. A file system is a typical example. Directory is a composite pattern ) |
| ---[Decorator](http://www.javacamp.org/designPattern/decorator.html) |
| ---[Façade](http://www.javacamp.org/designPattern/facade.html) ( Make a complex system simpler by providing a unified or general interface, which is a higher layer to these subsystems).  JDBC design is a good example of Façade pattern. A database design is complicated. JDBC is used to connect the database and manipulate data without exposing details to the clients. |
| ---[Flyweight](http://www.javacamp.org/designPattern/flyweight.html) |
| ---[Proxy](http://www.javacamp.org/designPattern/proxy.html) ( Use a simple object to represent a complex one or provide a placeholder for another object to control access to it. Image example) |
| Behavioral Patterns |
| ---[Chain of Responsibility](http://www.javacamp.org/designPattern/chains.html) |
| ---[Command](http://www.javacamp.org/designPattern/command.html) |
| ---[Interpreter](http://www.javacamp.org/designPattern/interpreter.html) |
| ---[Iterator](http://www.javacamp.org/designPattern/iterator.html) ( Provide a way to move through a list of collection or aggregated objects without knowing its internal representations) |
| ---[Mediator](http://www.javacamp.org/designPattern/mediator.html) |
| ---[Memento](http://www.javacamp.org/designPattern/memento.html) |
| ---[Observer](http://www.javacamp.org/designPattern/observer.html) |
| ---[State](http://www.javacamp.org/designPattern/state.html) ( An object's behavior change is represented by its member classes, which share the same super class.)   * Need to control many states without using if-else or switch statements. * Use a class to represent a state, not a constant or something else.   Exampl.e  Users connect to a database to do some jobs. Users from Management department may focus on management. Users from Sales department may focus on sales information. Every connection has to perform similar functions like open, log and close. Suppose we have an abstract Connection class and have these functions listed. Thus, every subclass of Connection must implement these functions. We list three subclasses Management, Sales and Accounting for example, just to show the State pattern concept. The Controller class contains each state of connection. Its behavior is decided by another object, which is a Test class. All the details have been hidden from the Test class. Suppose we have a server which is a singleton. Which connection is made depends on the user. We use a Test class which makes a trigger from command line. In the real program, the trigger should be made by the user.  -------- |
| ---[Strategy](http://www.javacamp.org/designPattern/strategy.html) ( Group several algorithms in a single module to provide alternatives. Also known as policy. ) Compress files using different algorithms or save files in different formats or draw graphic in different presentations. Here is a simple example. Just to show the concept of a strategy pattern. |
| ---[Template Method](http://www.javacamp.org/designPattern/template.html) |
| ---[Visitor](http://www.javacamp.org/designPattern/visitor.html) |
| J2EE Patterns |
| ---[MVC](http://www.javacamp.org/designPattern/mvc.html) |
| ---[Business Delegate](http://www.javacamp.org/designPattern/businessdgt.html) ( An intermediate class decouples between presentation-tier clients and business services.) Definition An intermediate class decouples between presentation-tier clients and business services. Where to use & benefits  * Simplify the complicated relationship. * Reduce coupling. * Cache results and references to remote business services. * Cut potentially costly round trips * Hide the underlying implementation details of business service. * Related patterns include   + [Proxy](http://www.javacamp.org/designPattern/proxy.html) combined to simplify the complexity.  Example Make a class deal with lookups and exception, acting as a representative of the client components  Cache results and references to remote business services |
| ---[Composite Entity](http://www.javacamp.org/designPattern/compositeentity.html) |
| ---[Data Access Object](http://www.javacamp.org/designPattern/dao.html) ( Adapt a uniform interface to access multiple databases like relational, unrelational, object-oriented, etc.  ) |
| ---[Front Controller](http://www.javacamp.org/designPattern/frontcontrol.html) ( Using a single component to process application requests) |
| ---[Intercepting Filter](http://www.javacamp.org/designPattern/interfilter.html) |
| ---[Service Locator](http://www.javacamp.org/designPattern/servicelocator.html) |
| ---[Transfer Object](http://www.javacamp.org/designPattern/transferobj.html) |
| Misc |
| ---[typesafe enum](http://www.javacamp.org/designPattern/enum.html) |
|  |
| ---[RESTful WS](http://www.javacamp.org/designPattern/rest.html) |

They are basicall for context,problem,solution

Singleton makes objects responsible for themselves

#### Example 21-1: Singleton Pattern

[TOP](http://www.netobjectives.com/resources/books/design-patterns-explained/java-code-examples/chapter21/#top)

public class USTax extends Tax {

private static USTax instance;

private USTax() {

}

public static USTax getInstance() {

if (instance== null) instance= new USTax();

return instance;

}

}

1. The difference between aggregation and composition is:

Answer: Composition is an "owning" relationship

Note:In composition, if an object is removed, it's related objects are also removed

Note:In aggregations, related objects can exist on their own

1. Refactoring is:
   * Answer: c. A re-architecture of the code that does not change it's behavior

**Java Programming Language** By *Ken Arnold, James Gosling, David Holmes*

**Head First Java** By *Kathy Sierra, Bert Bates*

**Thinking In Java** By *Bruce Eckel*

**The elements of Java style**   By *Scott Ambler, Alan Vermeulen*

**strictfp** is a [keyword](https://en.wikipedia.org/wiki/Java_keyword) in the [Java programming language](https://en.wikipedia.org/wiki/Java_(programming_language)) that restricts [floating-point](https://en.wikipedia.org/wiki/Floating-point) calculations to ensure portability.

* Vectors are synchronized, ArrayLists are not.
* Data Growth Methods

 multiple threads access an ArrayList concurrently then we must externally synchronize the block of code which modifies the list either structurally or simply modifies an element. Structural modification means addition or deletion of element(s) from the list. Setting the value of an existing element is not a structural modification.

**A Vector defaults to doubling the size of its array, while the ArrayList increases its array size by 50 percent.**

# [why java is not pure object oriented language? [closed]](http://stackoverflow.com/questions/6151497/why-java-is-not-pure-object-oriented-language)

The usual objection is that Java is not "purely" OO because it has primitive types (int, double, etc.), which are not objects.

int i = 42;

System.err.println(i.toString());

With question 3, [this article](http://www.infoq.com/articles/rest-soap-when-to-use-each) states that RESTful Services are appropriate in this scenarios:

* If you have limited bandwidth
* If your operations are stateless
* If your clients require caching.

While SOAP is the way to go when:

* If you require asynchronous processing
* If you need formal contract/Interfaces
* In your service operations are statefull

[**Difference between SOAP webservice and RESTFUL webservice**](http://stackoverflow.com/questions/7561130/difference-between-soap-webservice-and-restful-webservice)

As the first answer already explains, SOAP Webservices and REST Webservices differ in various points. SOAP:

* you define your interface in a .wsdl file, which describes exactly which input parameters are expected and how the return values will look like
* there are tools to generate the .wsdl files out of java class hirarchies. [JAXB for example](http://jaxb.java.net/)
* there are also tools to generate java objects/classes as part of eclipse for example (don't know the name in the moment).
* SOAP is very strict. Every request is validatet against the wsdl before processing.

A good but not so easy to start with framework for SOAP WS is [Apache CXF](http://cxf.apache.org/)

REST (no hands on experience up to now, feel free to correct and improve ;) ):

* a way to access a webserver or web application to retrieve data from or send to it.
* it's only negotiated, how it is accessed.
* common is something like this <http://server.domain.com/app/type/id=123> to retrieve object of type type with id=123
* very intuitive, but no automatic validation of requests.

# [What is the difference between a top-down web service and a bottom-up web service?](http://stackoverflow.com/questions/5890438/what-is-the-difference-between-a-top-down-web-service-and-a-bottom-up-web-servic)

Top-down means you start with a WSDL and then create all the necessary scaffolding in Java all the way down.

Bottom-up means you start with a Java method, and generate the WSDL from it.

SOAP means that the URL is the same for all invocations, and only the parameters to the Java method differs. REST means that the URL reflects the operation to be done