Cours IUT CSID – January 2012

Introduction to Java Annotations

Sample Library Usages

Arnaud Nauwynck

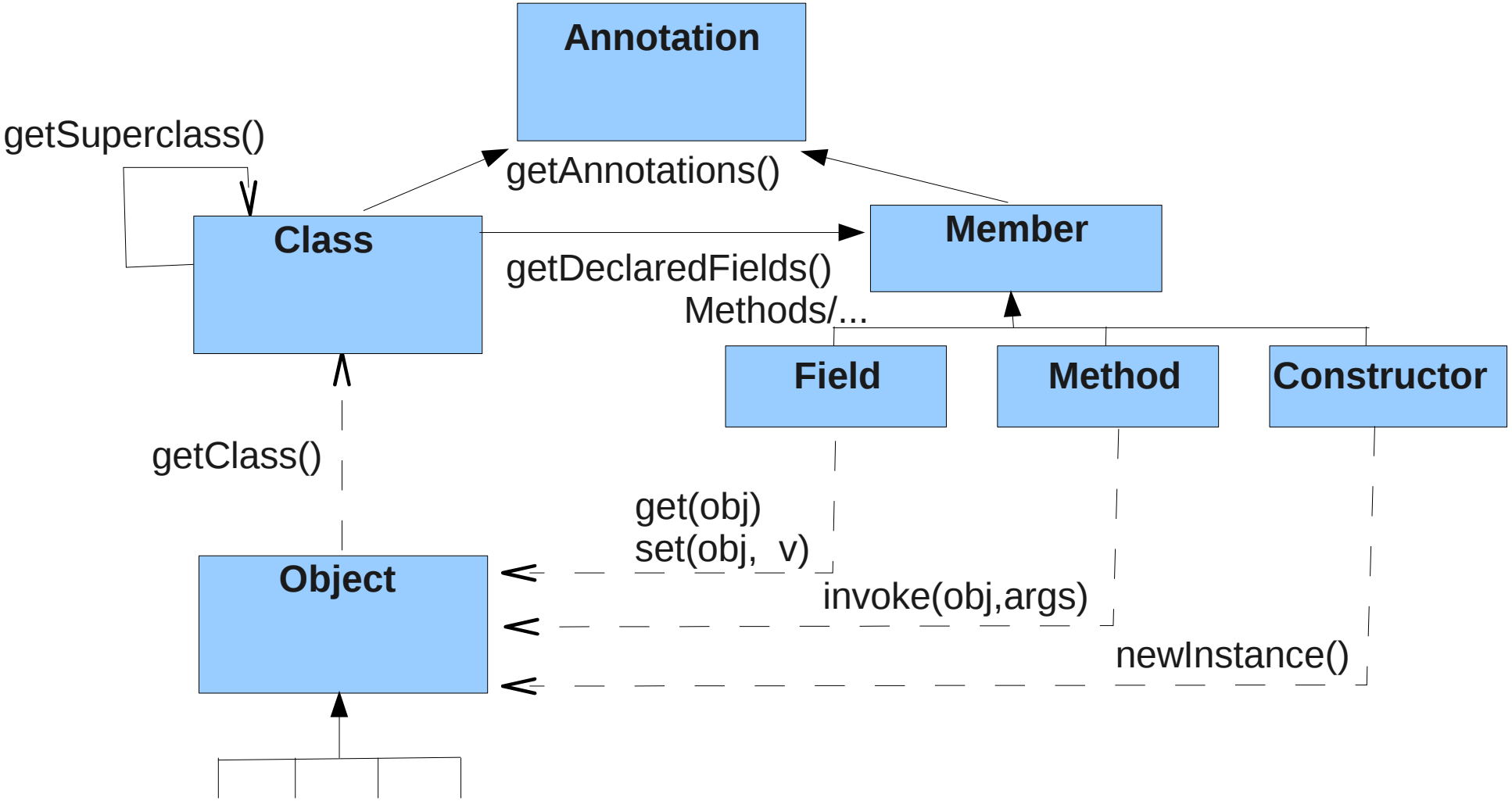
This document:
http://arnaud.nauwynck.chez-alice.fr/devPerso/Pres/Intro-Java-Annotations.pdf

Table Of Content

Annotation History

- Literate Programming
  - Inspired by D.E. Knuth ...~ 1970 (!)
- Principle : 1 Program => 2 Outputs
  - Code (tangle) => compile + run-it
  - Doc Comment (weave) => extract TeX + print it
- Java …. JavaDoc /** @Author … */
- Xdoclet … compiler for some JavaDoc @Tags
- Then Java 5 … Spring, Apt, AspectJ, Lombok...

Introspection API



Sample Introspection Usage

- Serializing Objects
  - ObjectOutputStream.writeObject(obj)
  - Obj = ObjectInputStream.readObject()
- Used in Rmi (Marshall / Unmarshall)
- JMX : expose object attributes + methods
- Spring : instanciate + configure objects as Xml
- Hibernate : mapping Java – Jdbc

  (1 class = 1 table.... 1 field = 1 column)
- ...

Sample Introspection Code

- Object to XML recursive dump

```
public void dump(Object obj, PrintStream out) throws Exception {
    if (obj == null) { out.print("<null/>"); return; }
    Class<?> clss = obj.getClass();
    Format fmt = tryLookupClassFormatter(clss);
    if (fmt != null) {
        out.print("<value>" + fmt.format(obj) + "</value>");
    } else {
        out.print("<obj className='" + clss.getName() + "'>");
        for(Field field : collectFieldsAndSuperClassFields(clss)) {
            out.print("<" + field.getName() + ">");
            Object fieldValue = field.get(obj);
            dump(fieldValue, out); // recurse
            out.print("</>");
        }
        out.println("</obj>");
    }
}
```

Java and Introspection

- Introspection is a fundamental feature of Java

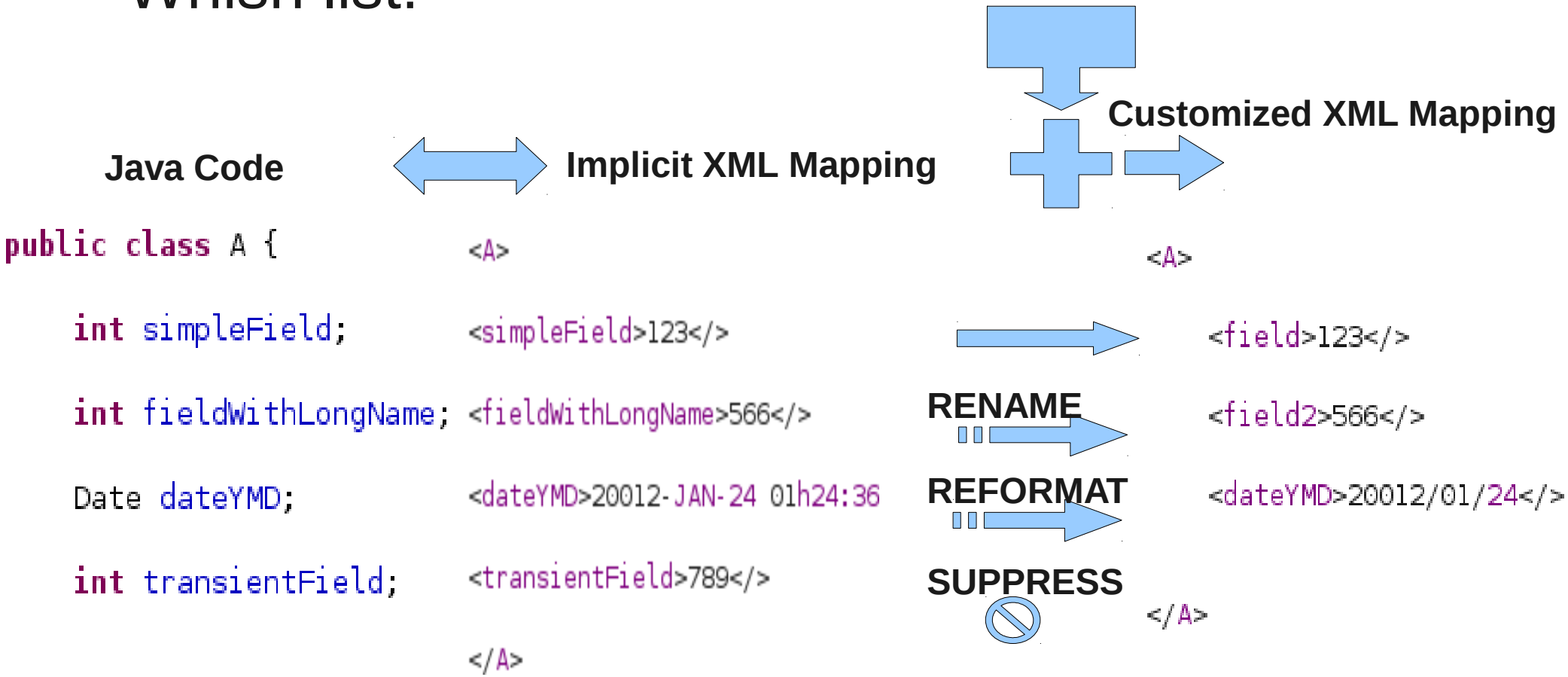
  … missing in C++ (!!) … success of java

  Lot of Frameworks use it

- configuration possibilities:
  - NOT configurable
    (example RMI) … but magic enough
  - Configurable
    - with external Xml/Prop Files
      (example: old Hibernate, old spring...)
    - with @Annotations

Sample Code Customization Goal

- Problem:
  - How to configure name mappings and converters...
- Whish list:



Adding / Mixing Extra Configurations

- Annotations do not interfere with code:
  - code ignore them !
  - read on demand
- like JavaDoc .. but typed & present
- Like external files

Annoted INFO

```
@XmlRootElement
@MyClassAnnotation("A")
public class A {
```

External Files INFO

ext-file.xml

ext-file1.properties

ext-file2.properties

For JAXB → `@XmlElement(name="field2")`

user-defined → `@MyFieldAnnotation("field2")`

For Spring → `@Value("${key}")`

```
int fieldWithLongName;
```

Syntax & Reference

Reading Annotation Values From Introspection

```java
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface MyFieldAnnotation {

    String value();

    String format() default "";

    int precision() default 10;
```

```java
for(Field field : collectFieldsAndSuperClassFields(clss)) {
    MyFieldAnnotation myAn = field.getAnnotation(MyFieldAnnotation.class);
    if (myAn != null) { // found annotation @MyFieldAnnotation(...)
        String value = myAn.value(); // read main value @MyFieldAnnotation("value")
        String format = myAn.format(); // read @...(myFormat="myFmt")
        int prec = myAn.precision(); // read @...(precision=123)  ... default 10
        dumpFieldWith(obj, field, value, format, prec);
    } else { // specific annotation not found... see all others
        Annotation[] annotations = field.getAnnotations();
```

Syntax

```java
@Retention(RetentionPolicy.SOURCE)
@Target(ElementType.TYPE)
public @interface MyClassAnnotation {

    String value();

    int order() default 100;

}
```

declare

use

```java
@MyClassAnnotation("B")
public class B {

    @MyFieldAnnotation
    private int field;

    @MyConstructorAnnotation public B() {
    }

    @MyMethodAnnotation
    public void f(@MyParamAnnotation int p) {
        @MyVarAnnotation int loc = 1;
        g(loc);
    }
```

Annotation of Annotation... @Target(ElementType)

```java
package java.lang.annotation;

 * Indicates the kinds of program element to which an annotation type
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
public @interface Target {
    ElementType[] value();
}
```

```java
package java.lang.annotation;

 * A program element type.  The constants of this enumerated type
public enum ElementType {
    TYPE,
    FIELD, METHOD, CONSTRUCTOR,
    PARAMETER, LOCAL_VARIABLE, ANNOTATION_TYPE, PACKAGE
}
```

Samples :

```java
@Target(ElementType.TYPE)
public @interface MyClassAnnotation
```

```java
@Target({ ElementType.METHOD, ElementType.CONSTRUCTOR } )
public @interface MyCodeAnnotation {
```

Annotation of Annotation... @Rentention(RetentionPolicy)

```
@Retention(RetentionPolicy.SOURCE)
public @interface MySourceAnnotation {

    public enum RetentionPolicy {
        /**
         * Annotations are to be discarded by the compiler.
         */
        SOURCE,

        /**
         * Annotations are to be recorded in the class file by the compiler
         * but need not be retained by the VM at run time.  This is the default
         * behavior.
         */
        CLASS,

        /**
         * Annotations are to be recorded in the class file by the compiler and
         * retained by the VM at run time, so they may be read reflectively.
         *
         * @see java.lang.reflect.AnnotatedElement
         */
        RUNTIME
    }
```

Sample Libraries Usages

SpringFramework

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>3.1.0.RELEASE</version>
</dependency>
```

- spring-context-support-3.1.0.RE
- spring-beans-3.1.0.RELEASE.jar
- spring-context-3.1.0.RELEASE.ja
- spring-expression-3.1.0.RELEAS
- spring-core-3.1.0.RELEASE.jar -
- spring-asm-3.1.0.RELEASE.jar -
- spring-aop-3.1.0.RELEASE.jar - /
- aopalliance-1.0.jar - /home/arna
- spring-orm-3.1.0.RELEASE.jar - /
- spring-jdbc-3.1.0.RELEASE.jar - /
- spring-tx-3.1.0.RELEASE.jar - /hc

SpringFramework IOC

- @Component … Spring calls “new ()”
- @Injected … Spring calls “set()”
- @Value

- Now Standards (EJB3, Java6):
  - @Stateless, @Service, @Local, @Remote …
  - @Resource, @EJB

SpringFramework Sample

```
<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-3.0.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <!-- enable annotation and dependency injection by @Resource -->
    <context:annotation-config />

    <!-- scan classes by @Component -->
    <context:component-scan base-package="fr.an.test" />
```

Scan + instanciate + Dependency injection (IOC)

```
@Component
public class MyClient {

    @Resource
    protected MyService myService;

    public void call() {
        myService.compute(1);
    }
```

```
@Component
public class MyService {

    public int compute(int param) {
        return param*param;
    }
```

Spring Web MVC

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>3.1.0.RELEASE</version>
</dependency>
```

Spring Web – MVC Controler @Controler, @ContextPath ...

```
@Controller
public class HomeController {

    /**
     * handler for http GET "/hello" .. render with "WEB-INF/jsp/helloWorldView.jsp" view
     */
    @RequestMapping(value = "/hello", method = RequestMethod.GET)
    public String handleHome(Locale locale, Model model) {
        model.addAttribute("serverMessage", "server msg" );
        return "helloWorldView"; // => "WEB-INF/jsp/helloWorldView.jsp"
    }
```

Spring MVC Configuration


```
<context:component-scan base-package="fr.an.test.springweb" />

<mvc:annotation-driven />

<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <beans:property name="prefix" value="/WEB-INF/jsp/" />
    <beans:property name="suffix" value=".jsp" />
</beans:bean>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

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

    <servlet-mapping>
        <servlet-name>springDispatcherServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```

Junit 4

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.10</version>
    <scope>test</scope>
</dependency>
```

Junit Annotations

- Before E.Gamma & K. Bent EPOCH ... no tests?
- Junit 3: A simple piece as small as revolutionnary....

```
import junit.framework.TestCase;
import org.junit.Assert;

public class Mytest extends TestCase {

    public void test1() {
        Assert.assertEquals(2, 1+1);
    }
```

- Junit 4: Simpler that simple...

```
public class MyTest {

    @Test
    public void f() {
        Assert.assertEquals(2, 1+1);
    }
```

Junit4 : @RunWith(...)
(+ Spring @ContextConfiguration)

- @RunWith() = customize Junit4 runner
  - With Spring runner + @ContextConfiguration(...)
  - Or others (MockitoRunner.class, etc.)

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "/applicationContext-test1.xml" })
public class MySpringTest {

    @Resource
    protected MySpringObj obj;

    @Test
    public void test() {
        obj.callMyEJB(1);
    }
```

Mockito

```
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-all</artifactId>
    <version>1.9.0-rc1</version>
    <scope>test</scope>
</dependency>
```

Mockito (Mock for JUnit)

- @Mock to instanciate mock proxy on interfaces
- @InjectMock to inject dependency into object

```
public class MyObjMockTest {

    // @Mock is equivalent to create a mock proxy
    // init from MockitoAnnotations.initMocks(this);
    @Mock private MyEJB myEJB;

    // SUT = System Under Test
    // @InjectMocks is equivalent to autowiring setter "sut.set(...)"
    //  init from MockitoAnnotations.initMocks(this);
    @InjectMocks
    private MySpringObj sut = new MySpringObj();

    @Before public void setup() {
        // equivalent to @RunWith(MockitoJUnitRunner.class) in test class
        MockitoAnnotations.initMocks(this);
    }

    @Test
    public void test1() {
```

Lombok

```
<!-- cf http://projectlombok.org/ for installation
   (need to add lombok.jar in eclipse
    need to add "-javaagent:lombok.jar" in eclipse launcher)
-->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>0.10.6</version>
    <scope>provided</scope>
</dependency>
```

Lombok Sample

You write .java file

```java
import lombok.Data;
import lombok.Getter;
import lombok.Setter;

public class LombokPOJO {

    @Getter @Setter
    private int myField;

    @Data
    public static class OtherPOJO {
        private int field1, field2;
    }

}
```

You get in compiled .class

- LombokPOJO
  - myField : int
  - getMyField() : int
  - setMyField(int) : void
  - OtherPOJO
    - OtherPOJO()
    - getField1() : int
    - getField2() : int
    - setField1(int) : void
    - setField2(int) : void
    - equals(Object) : boolean
    - canEqual(Object) : boolean
    - hashCode() : int
    - toString() : String
    - field1 : int
    - field2 : int

never write Getter-Setter anymore Because Small is Beautifull (http://projectlombok.org)

Project Lombok - Download

Download lombok.jar 0.10.6

Feeling adventurous? Downl

Maven or Ivy
Javac
NetBeans
Eclipse and STS

Project Lombok

Lombok features

@Getter / @Setter
Never write `public int getFoo() {return foo;}` again.

@Getter(lazy=true)
Laziness is a virtue!

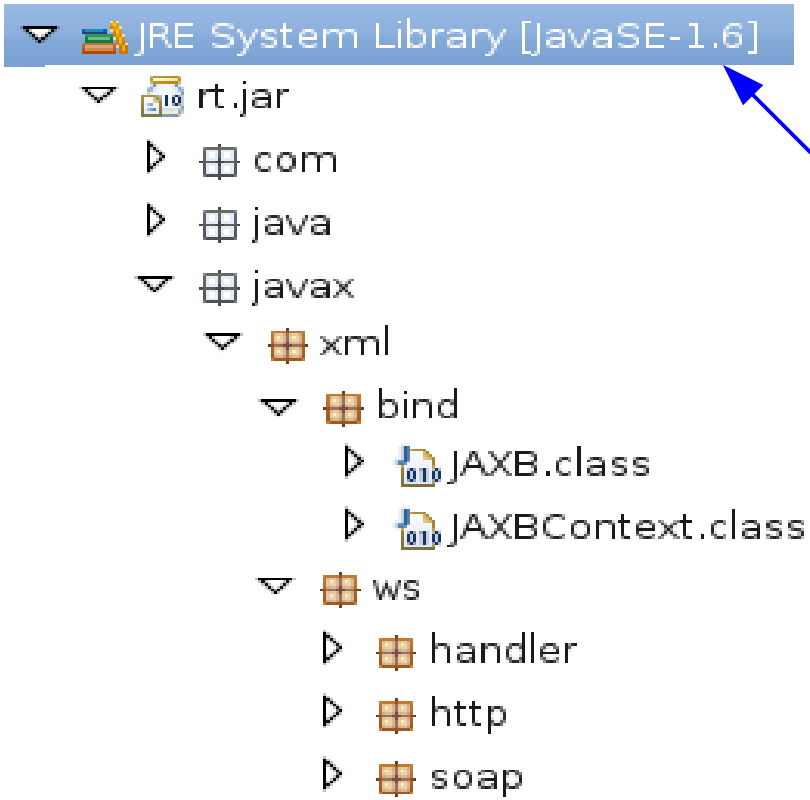
@ToString
No need to start a debugger to see your fields: Just let lombok generate a `toString` for you!

@EqualsAndHashCode
Equality made easy: Generates `hashCode` and `equals` implementations from the fields of your object.

@NoArgsConstructor, @RequiredArgsConstructor and @AllArgsConstructor
Constructors made to order: Generates constructors that take no arguments, one argument per final / non-null field, or one argument for every field.

@Data
All together now: A shortcut for `@ToString`, `@EqualsAndHashCode`, `@Getter` on all fields, and `@Setter` on all non-final fields, and `@RequiredArgsConstructor`!

JAXB, JAX-WS




```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
        <source>1.6</source>
        <target>1.6</target>
    </configuration>
</plugin>
```

JAXB = Xml Binding


```java
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement( name="doc" )
public class MyJaxbObj {

    @XmlElement
    protected Foo foo;
```

```java
public void obj2xml(MyJaxbObj obj, ObjectOutputStream out) throw
    JAXBContext ctx = JAXBContext.newInstance(MyJaxbObj.class);
    Marshaller marshaller = ctx.createMarshaller();
    marshaller.marshal(obj, out);
}
```

Java2Xsd
Xsd2Java

JaxbContext
marshal() / unmarshal()

```xsd
<?xml version="1.0" encoding="UTF-8"?>

<xsd:complexType name="Foo">
</xsd:complexType>

<xsd:complexType name="Document">
  <xsd:sequence>
    <xsd:element name="foo" type="Foo"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="doc" type="Document"/>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <foo>...</foo>
</doc>
```

JAX-WS = Java Web Service


```java
import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService(targetNamespace = "http://duke.org", name="AddNumbers")
public interface MyWebService extends Remote {

    @WebMethod(operationName="add", action="urn:addNumbers")
    public int addNumbers(int number1, int number2) throws RemoteException;
```

```java
@javax.jws.WebService(serviceName = "MyWebService",
    endpointInterface = "fr.an.test.MyWebService",
    targetNamespace="http://mywebservice.test.fr/",
    portName="MyWebServicePort")
public class MyWebServiceImpl implements MyWebService {

    @Override
    public int addNumbers(int number1, int number2) thr
        return number1 + number2;
    }
```

Java2wsdl (jaxws:wsgen)
Wsdl2Java (jaxws:wsimport)

Sample wsgen
(Java → Wsdl + Port classes)

```
<plugin>
  <groupId>org.jvnet.jax-ws-commons</groupId>
  <artifactId>jaxws-maven-plugin</artifactId>
  <version>2.1</version>
  <configuration>
      <sei>fr.an.test.MyWebServiceImpl</sei>
      <genWsdl>true</genWsdl>
  </configuration>
  <executions>
      <execution>
          <id>wsgen</id>
          <goals>
              <goal>wsgen</goal>
          </goals>
      </execution>
  </executions>
</plugin>
```

WSDL

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.7-b01-. -->
<definitions targetNamespace="http://mywebservice.test.fr/" name="MyWebService" xmlns="http://schema
  <import namespace="http://duke.org" location="AddNumbers.wsdl"/>
  <binding name="MyWebServicePortBinding" type="ns1:AddNumbers" xmlns:ns1="http://duke.org">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="add">
      <soap:operation soapAction="urn:addNumbers"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="MyWebService">
    <port name="MyWebServicePort" binding="tns:MyWebServicePortBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
    </port>
  </service>
</definitions>
```

JPA = Java Persistence API

```
<dependency>
  <groupId>javax.persistence</groupId>
  <artifactId>persistence-api</artifactId>
  <version>1.0.2</version>
</dependency>
```

JPA Sample Entity Annotations

```
@Entity
@Table(name="T_USER")
public class User {

    @Id
    @GeneratedValue(generator="SEQ")
    @SequenceGenerator(name="SEQ", sequenceName="SEQ_USER", allocationSize=1)
    private int id;

    @Version
    private int version;

    @Column(name="LOGIN", length=50, nullable=false)
    private String login;

    @ManyToOne(fetch=FetchType.LAZY)
    private Department department;

    @OneToMany(fetch=FetchType.LAZY, cascade={ CascadeType.ALL }, mappedBy="user")
    private List<UserRole> roles;
```

minimal @Entity : @Id + @Version

- Minimal mapping : @Entity
  - Need to specify a @Id !!
    (object is an entity If-and-Only-If it has a unique Id)
  - Better with @Version … for optimisitc-lock / cache...
- By default,
  - Table name = shortname of class
  - All fields are persistent...
    column name = name of field

Entity Custom mapping

- Override Table name or entity-table mapping...

  optionnal **@Table**()

- Override field mapping

  optionall **@Column**()

- Override Id generation

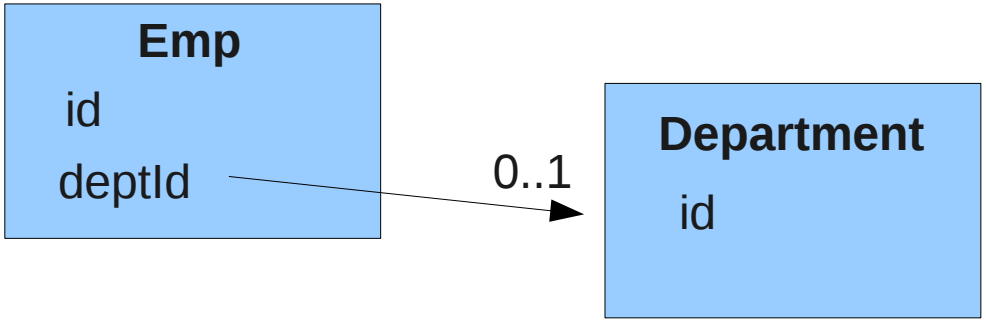
  @GeneratedValue
  @SequenceGenerator

Relation @ManyToOne + optional @JoinColumn (= Pointer / Foreign Key to PK)

```java
@Entity
public class Emp {

    private int id;
    private int version;

    @ManyToOne(fetch = FetchType.LAZY)
    private Dept dept;
```



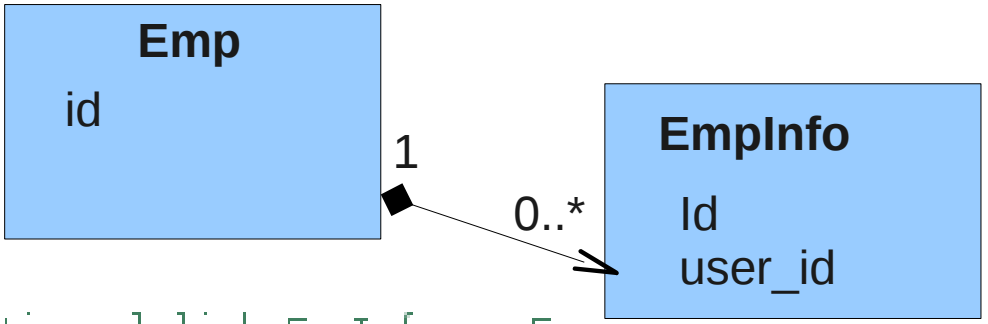

```java
public void userToDept(User u) {
    Department dept = u.getDepartment();
    // SQL equivalent: "select * from DEPT d where d.ID = ?user_dept_id"
}
```

@OneToMany
(= List / where Foreign Key=PK)

```java
@Entity
public class Emp {

    private int id;
    private int version;

    @OneToMany(fetch = FetchType.LAZY,
            mappedBy="emp", // <= bidirectionnal link EmpInfo -> Emp
            cascade=CascadeType.ALL // strong aggregation : delete Emp => delete EmpInfo
            )
    private Collection<EmpInfo> infos = new LinkedHashSet<EmpInfo>();
```

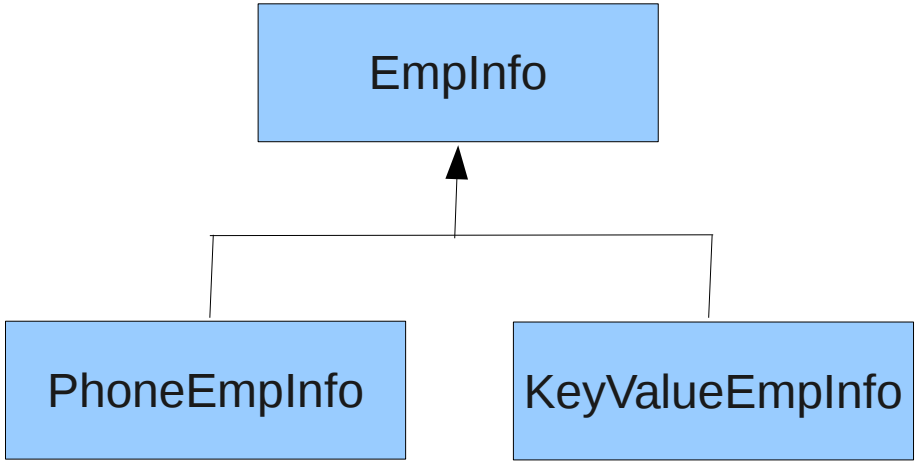



```java
public void printEmpInfo(Emp emp, OutputStream out) {
    // sql equivalent: "select * from EMP_INFO ei where ei.EMP_ID = ?"
    Collection<EmpInfo> infos = emp.getInfos();
    for (EmpInfo info : infos) { printInfo(out, info); }
}

public void deleteEmp(Emp emp) {
    em.remove(emp);
    // ==> with CascadeType.ALL ... implies "delete EMP_INFO ei where ei.EMP_ID=?"
}
```

@Inheritance, @DiscriminatorValue @DiscriminatorColumn

```
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
@DiscriminatorColumn(name="TYPE",
    discriminatorType=DiscriminatorType.CHAR)
public abstract class EmpInfo {
```




```
@Entity
@DiscriminatorValue("P")
public class PhoneEmpInfo extends EmpInfo {
```

```
@Entity
@DiscriminatorValue("K")
public class KeyValueEmpInfo extends EmpInfo {
```

Questions ?

Alors TP ! …

This document :

http://arnaud.nauwynck.chez-alice.fr/devPerso/Pres/Intro-Java-Annotations.pdf