Cours / TP

Tutorial to Java Reflection API

Class / ClassLoader / Field-Method Introspection

Arnaud.nauwynck@gmail.com

This document:
http://arnaud.nauwynck.chez-alice.fr/
Intro-JavaIntrospection.pdf

Outline

- Introduction to Concepts
- Sample Class / ClassLoading reflection code
- Sample Method / Field reflection
- Sample Annotation reflections

KeyWords

- Object Oriented, UML
- Class
  - also: Type, Interface, Enum, Annotation
- Member : Field, Method, Constructor
- ClassLoader
- Reflection, Introspection, Invocation
- Meta-Data, Meta-Information
- Program / Langage Meta-Data

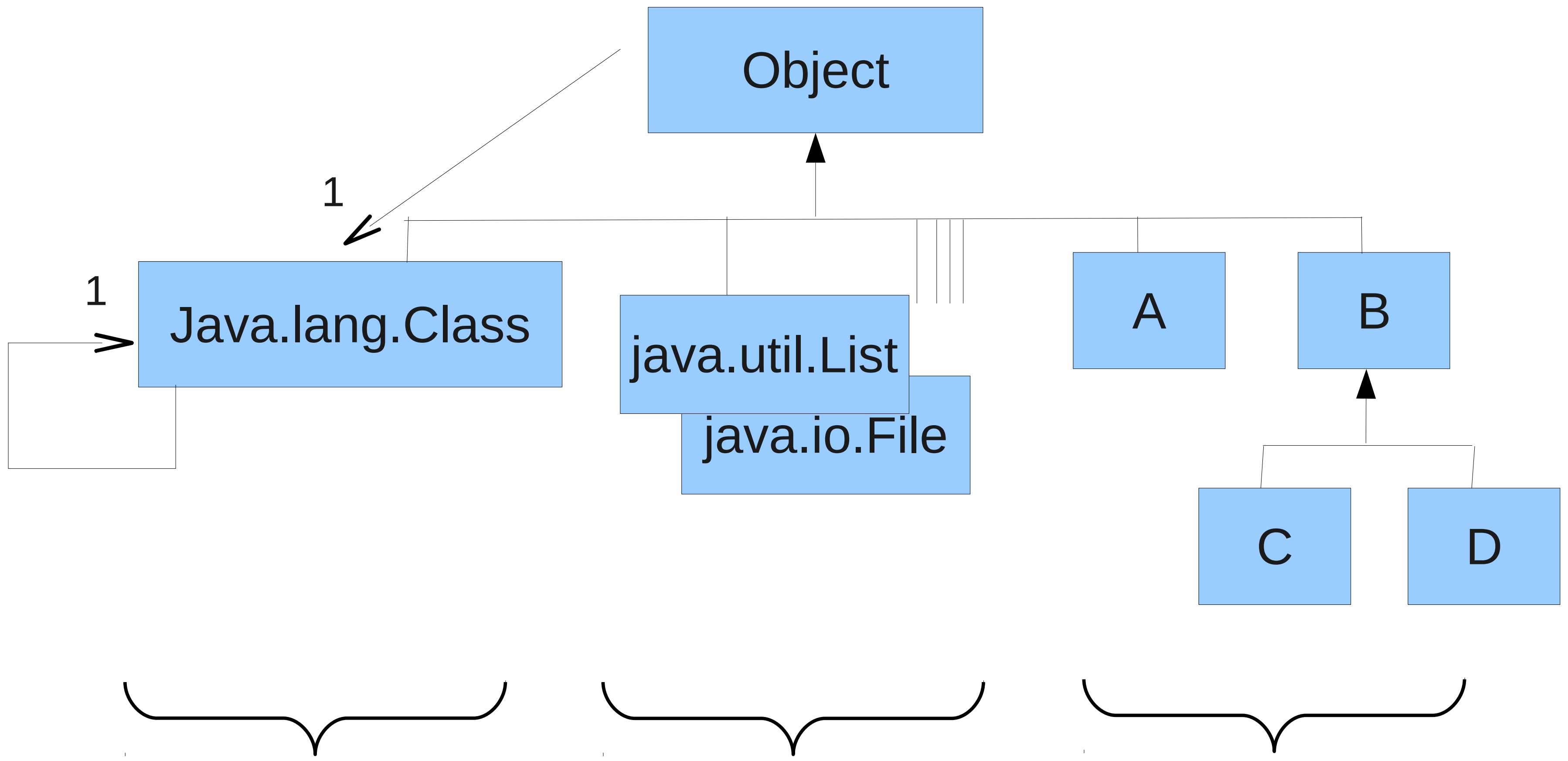
Object – Class
Class – SubClass Hierarchy

- Every Class extends “java.lang.Object”
  - … in java code:
    public class MyType { }
    …. implicitely: = class XXX extends Object {}

- Every Object instance has 1 and only 1 Type “java.lang.Class”
  - … in java code:
    MyType obj = …
    Class objClass = obj.getClass();

- Class objects are Objects … but there is NO sub-class class
  - final class Class extends Object {}
- Class Hierarchy Tree (class B extends A {} .. A extends Object)

UML Object - Class



Packages
  java.lang.*
  java.lang.reflect.*
Internal to JVM security

Other jdk packages
standard utility classes

User-defined classes

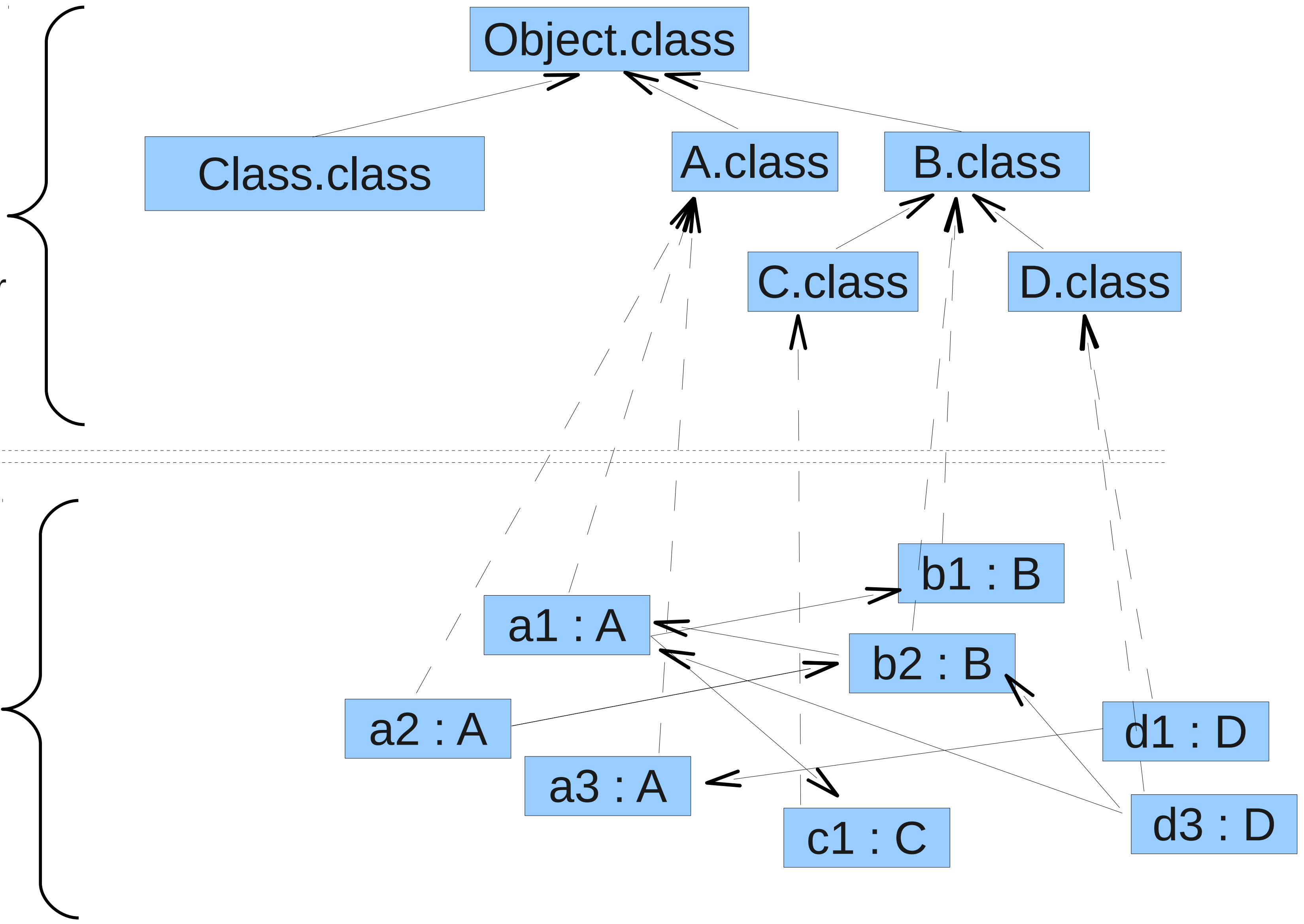
Data – MetaData
Object Instances – Meta Class

Meta-Classes

Namespace
= per ClassLoader

<u>**Objects Instances**</u>

Namespace
= pointers per jvm



Class Type-Checking Principles

- SubClass implies possibilities to downcast
  - Class B extends A { .. }
    A a1 = new B(); // implicit upcast
    B b1 = (B) a1;   // explicit downcast … Checked OK at runtime
- program semantic relies on ULTRA Strict Type-Checking (at compilation time + at runtime)
  - Class C {}
    A a2 = new C(); // … does NOT compile
    A a2 = (A) (Object) new C(); // ClassCastExption at runtime

Note: Additional Java Class Constraints

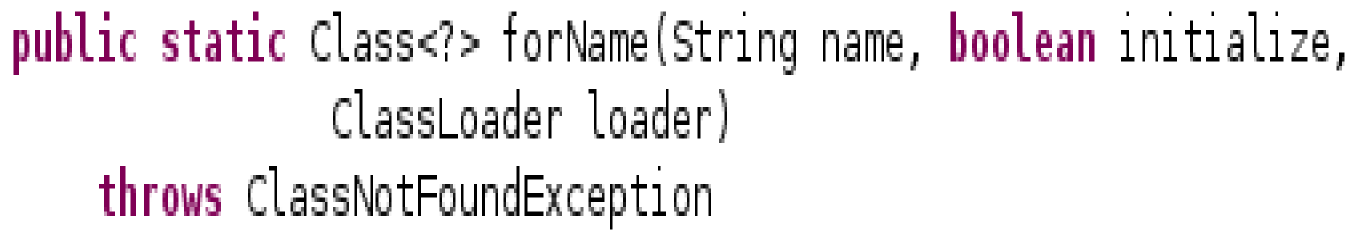
- A Class has 1 and only 1 super Class
  - No multi-inheritance like C++

- A Class may have 0..* interfaces
  - Class MyType … implements I1, I2, I3 { }
    interface I1 extends I4, I5 {}
- Interface contains NO data
  - Only “pure virtual” methods, or static constants
- primitive also have Type, but no pointers

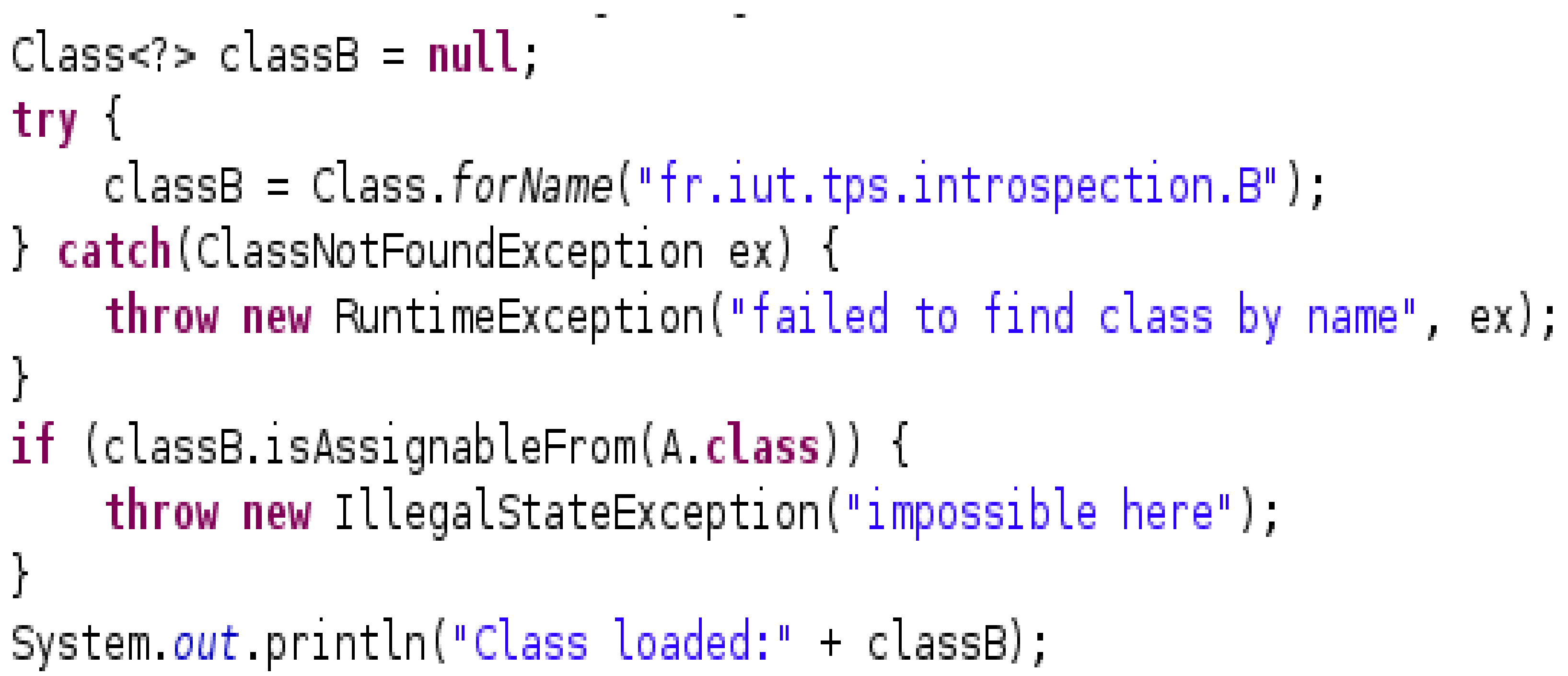
Secured Class instances
= JVM Managed “Constants”

- **For Security Reasons …**
  Classes are **SPECIAL** objects,
  entierely managed by the jvm
  - All fields are **private** final
  - Once constructed, they are **immuable**
- You can not instanciate “new Class()” !!
- BUT get like static constants :
  - **Class c = Class.forName(“comp.app.MyType”)**
  - **Class c = MyType.class;**

Detailed Class.forName()



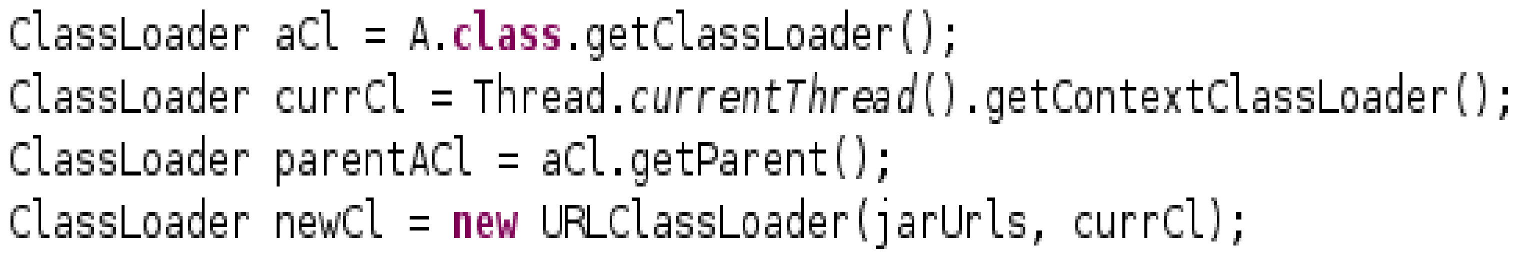
```java
public static Class<?> forName(String name, boolean initialize,
                               ClassLoader loader)
    throws ClassNotFoundException
```



```java
Class<?> classB = null;
try {
    classB = Class.forName("fr.iut.tps.introspection.B");
} catch(ClassNotFoundException ex) {
    throw new RuntimeException("failed to find class by name", ex);
}
if (classB.isAssignableFrom(A.class)) {
    throw new IllegalStateException("impossible here");
}
System.out.println("Class loaded:" + classB);
```

Class in ClassLoader ...

- 1 Class ==> 1 loading ClassLoader (!= 1 Defining ClassLoader)
- To obtain them:



```
ClassLoader aCl = A.class.getClassLoader();
ClassLoader currCl = Thread.currentThread().getContextClassLoader();
ClassLoader parentACl = aCl.getParent();
ClassLoader newCl = new URLClassLoader(jarUrls, currCl);
```

- ClassLoader can be chained in a strict Hierarchy... for security
  - BootClassLoader + SystemClassLoader
    + ApplicationClassLoader
    + ...Many on servers, 1 per isolated deployment (war, ear...)

- Classes are singleton by name … singleton PER ClassLoader !

  (1 ClassLoader , fullName) ===> 1 Class

ClassLoader and Thread Context

- Class A depends on class B at compilation time
  - If Pb loading B from A, … compilation contract is broken:

    NoClassDefFoundError() … not Exception !
    LinkageError, CompilationError, NoSuchFieldError, ...

- Whereas loading dynamic class
  - Pb if not found => ClassNotFoundException … not Error !

- To Load dynamic modules at runtime...
  - Use current ContextClassLoader

Thread.current ContextClassLoader

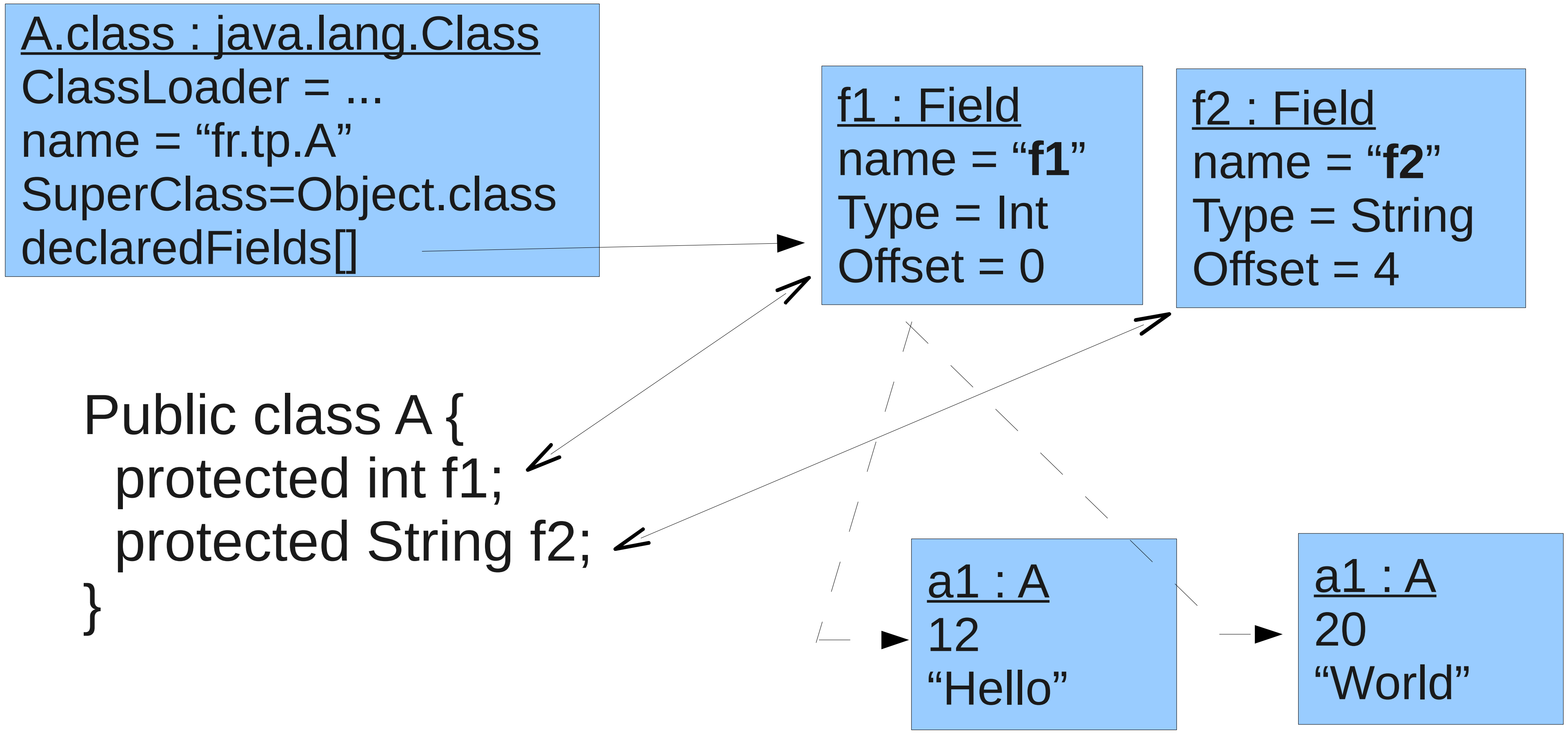
- Sample code... for plugins

```java
ClassLoader parentCl = A.class.getClassLoader();
URL[] jarUrls = new URL[] { new URL("myplugin.jar") };
ClassLoader newCl = new URLClassLoader(jarUrls, parentCl);

ClassLoader previousCl = null;
try {
    previousCl = Thread.currentThread().getContextClassLoader();
    Thread.currentThread().setContextClassLoader(newCl);
    // work with new classLoader...
    Class<?> a2Clss = Class.forName("fr.MyA2", false, newCl);
    A a2 = (A) a2Clss.newInstance();
    a2.doExecute();
} finally {
    Thread.currentThread().setContextClassLoader(previousCl);
}
```

Class Details

- Class contains Members : Fields, Method …
  - Field = description of a data slot inside object
  - Method = description of program fragment



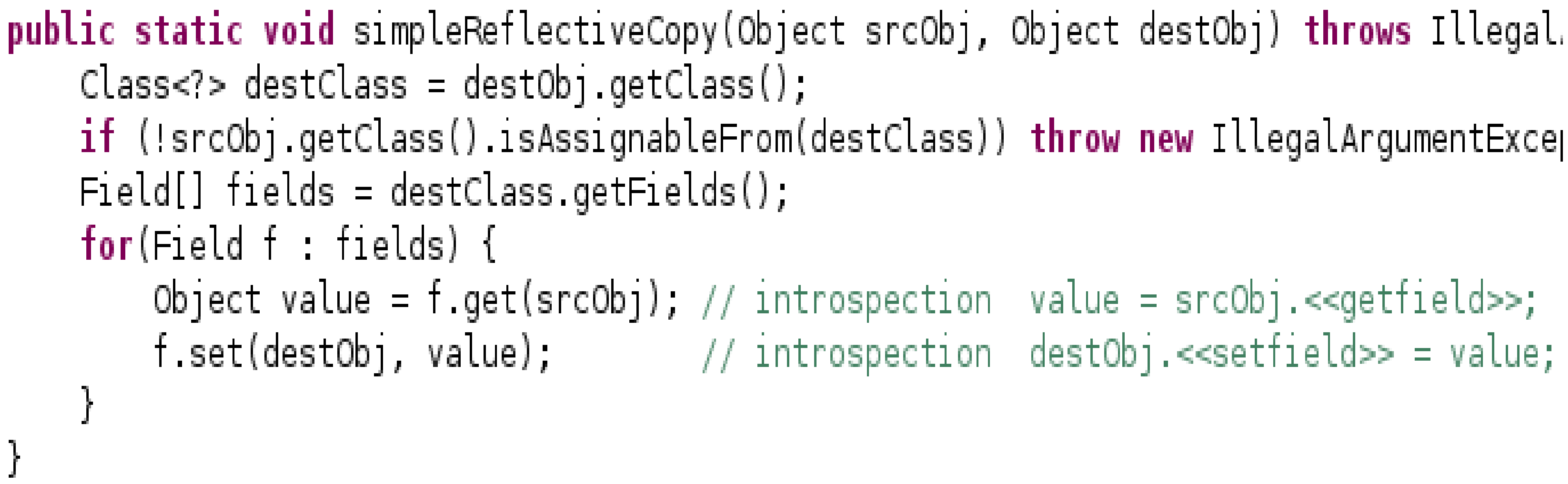
```
Public class A {
  protected int f1;
  protected String f2;
}
```

Introspection, Reflection

- Reflection, Introspection =
  - Speak about yourself
    Class c = Object.getClass()
    ==> Object knows himself
  - Deep explore himself :
    Fields[] fields = c.getFields();
    for (Field f : fields) { … recursive }
- Invocation
  - Possibility to trigger dynamic code execution
  - Similar to compiled code … but “not” compiled (hot loaded)

Field Value Get/Set Reflection

- Sample code … simple copy by reflection



```
public static void simpleReflectiveCopy(Object srcObj, Object destObj) throws Illegal
    Class<?> destClass = destObj.getClass();
    if (!srcObj.getClass().isAssignableFrom(destClass)) throw new IllegalArgumentExce
    Field[] fields = destClass.getFields();
    for(Field f : fields) {
        Object value = f.get(srcObj); // introspection  value = srcObj.<<getfield>>;
        f.set(destObj, value);        // introspection  destObj.<<setfield>> = value;
    }
}
```

- Pseudo Equivalent C/Assembly code

```
union_t* val =  wrap( *(&srcObj + f.offset) ) ;
*(&destObj + f.offset) = unwrap( *val );
```

Primitive Type Wrapper

- How to obtain a pointer to an “int” in Java ??
  - Pointer on stack => unsafe, crash
  - Pointer inside objects => unsafe, private
- All primitive types have a corresponding Wrapper value Class
  - int < = = > java.lang.Integer
    double <= = > java.lang.Double
    ...
- Wrap/unwrap conversion in built-in reflection methods

Primitive Type Wrapper, Auto Boxing

- Example for int

```
public final class Integer extends

    private final int value;

    public Integer(int value) {
        this.value = value;
    }
    public int intValue() {
        return value;
    }
```

- Typical code (Explicit)

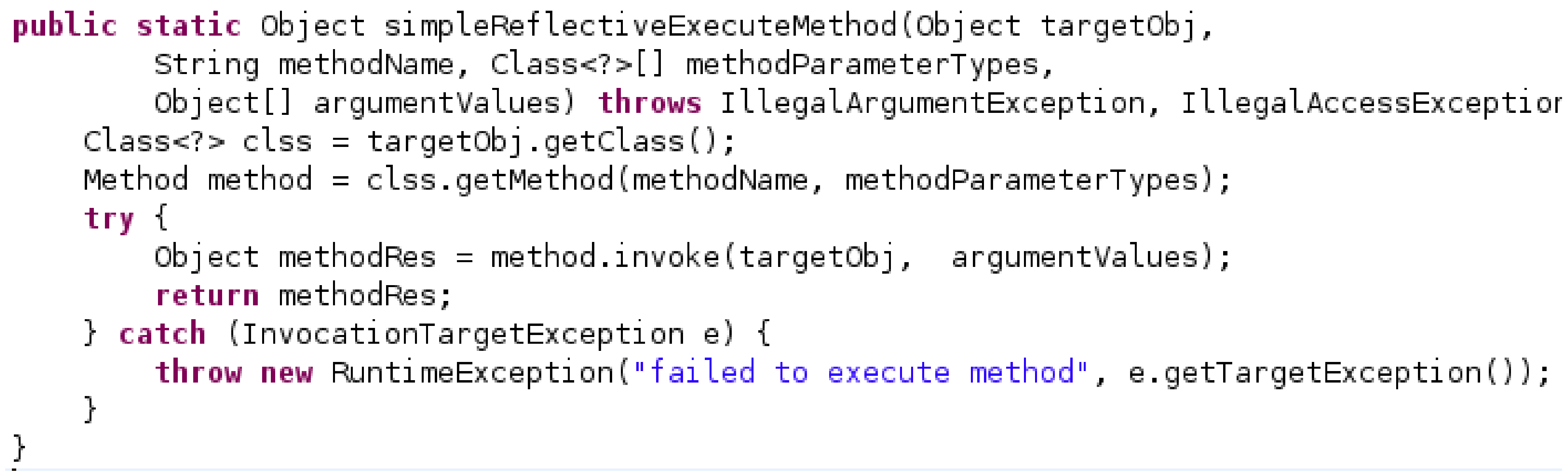
```
Object[] argumentValues = new Object[] { Integer.valueOf(intArg) };
Object methodRes = method.invoke(targetObj, argumentValues);
return ((Integer) methodRes).intValue();
```

- Implicit autoboxing code

```
int primitiveValue = value;
Integer obj = primitiveValue; // => boxing: new Integer(10)
int res = obj;  // => unboxing ((Integer)obj).intValue(), warn for NPE!
```

Method Invocation

- Sample code … execute method by name



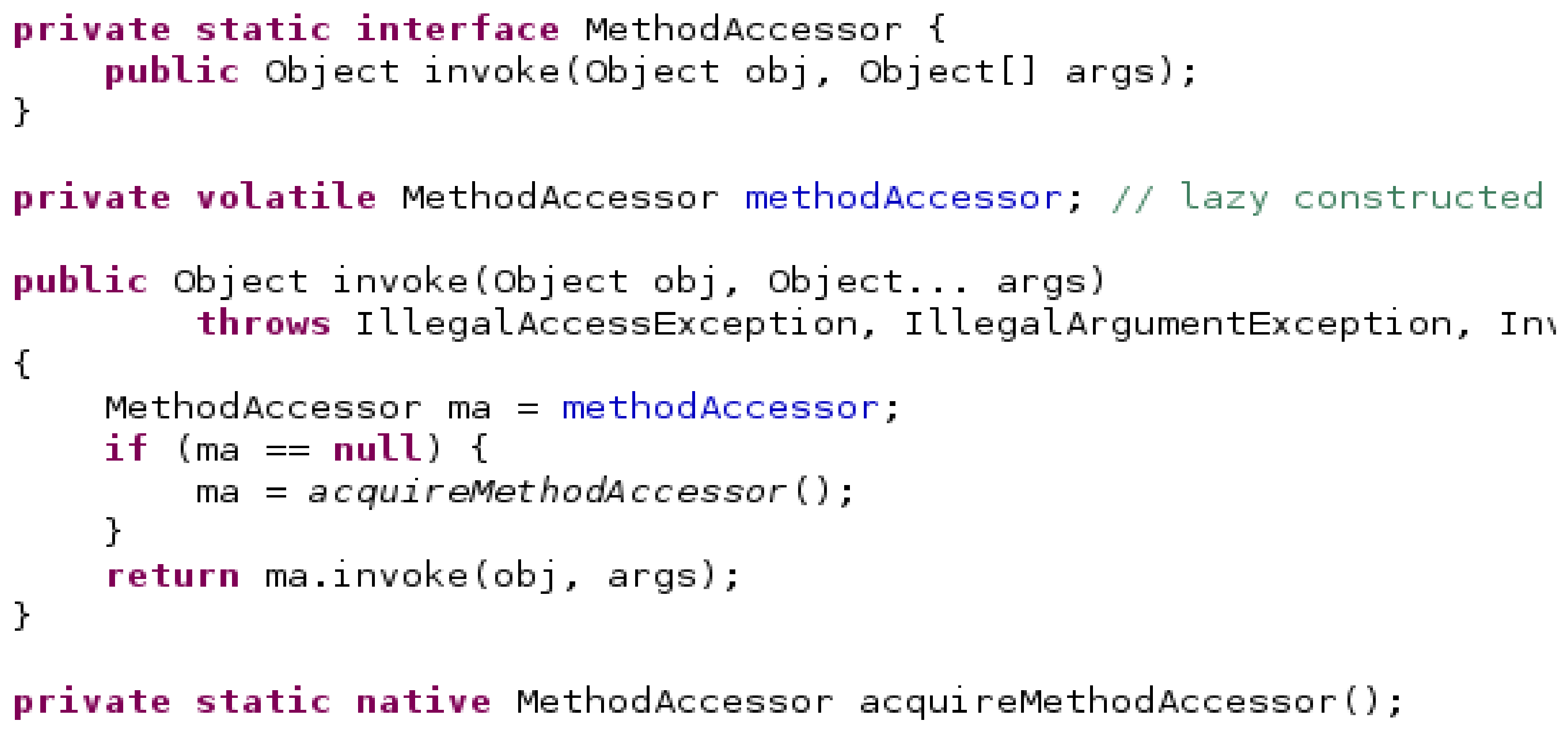
```
public static Object simpleReflectiveExecuteMethod(Object targetObj,
        String methodName, Class<?>[] methodParameterTypes,
        Object[] argumentValues) throws IllegalArgumentException, IllegalAccessExceptio
    Class<?> clss = targetObj.getClass();
    Method method = clss.getMethod(methodName, methodParameterTypes);
    try {
        Object methodRes = method.invoke(targetObj,  argumentValues);
        return methodRes;
    } catch (InvocationTargetException e) {
        throw new RuntimeException("failed to execute method", e.getTargetException());
    }
}
```

- Pseudo Equivalent C/Assembly code

```
typedef (ret) (*func_t)(args_t);
func_t  methodPointer = &(dynamicLoadedBytecodeMethod);
res = (*methodPointer)(args);
```

Method Invocation Details

- Internally... Method callbacks are loaded as a special anonymous inner Class (generated bytecode at runtime)
  - => efficient, but annoying to debug / view stack trace



```
private static interface MethodAccessor {
    public Object invoke(Object obj, Object[] args);
}

private volatile MethodAccessor methodAccessor; // lazy constructed

public Object invoke(Object obj, Object... args)
        throws IllegalAccessException, IllegalArgumentException, In
{
    MethodAccessor ma = methodAccessor;
    if (ma == null) {
        ma = acquireMethodAccessor();
    }
    return ma.invoke(obj, args);
}

private static native MethodAccessor acquireMethodAccessor();
```

Java Reflection Possibilities

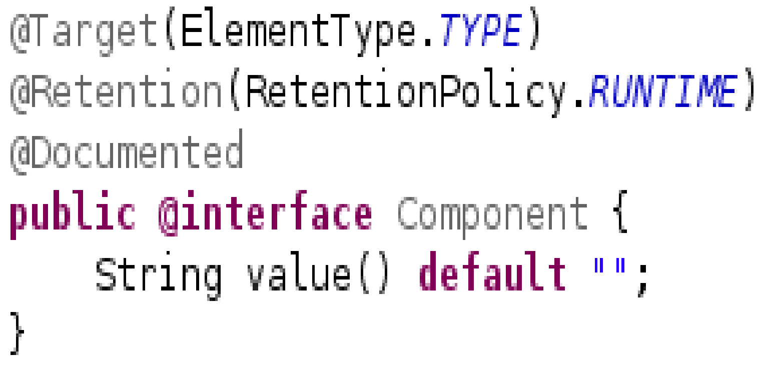
- Reflection offers Thousands possibilities
  - For runtime-Frameworks, (unkown user-defined code at compilation time)
  - Serialization (object to stream java.io.Serializable, Rmi)
  - Object Mapping (hibernate, Xml, …)
  - Injection (Spring, …)
- This is THE Why SO Many thousands library in Java … and Why 0.000 in C++   !!!

Annotations (Since Jdk 5)

- Frameworks used to have verbose config files
  - Hibernate.cfg.xml + ….*.hbm
  - Spring applicationContext.xml
- Purpose : add informatino on source code
  - JavaDoc is not usable, Xdoclet is nightmare...
- Since Jdk 5 … You can use Annotations !!

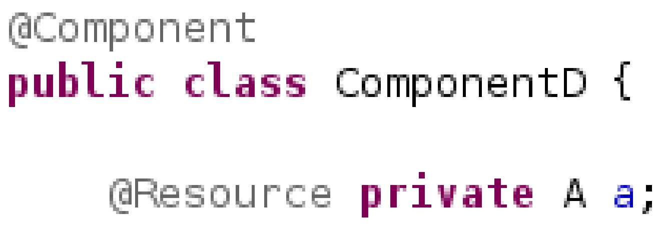
Annotation Sample API

Step 1 : Define annotation class



```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface Component {
    String value() default "";
}
```

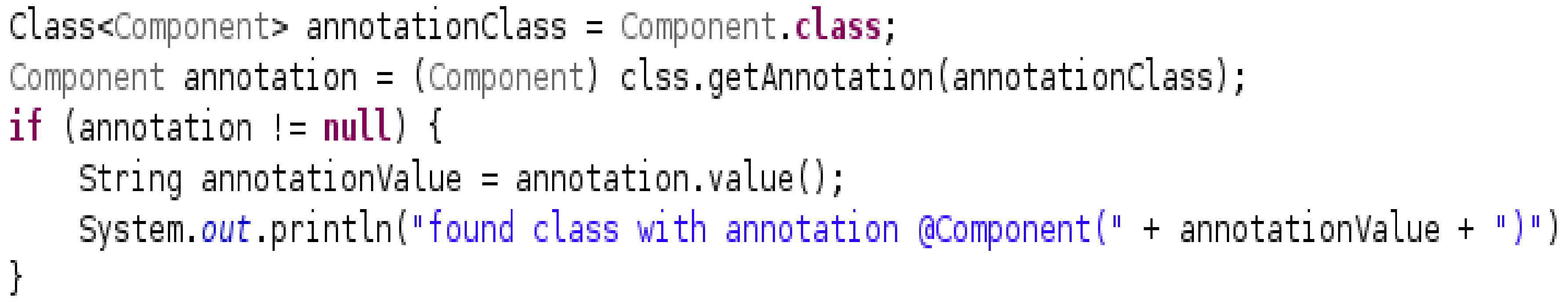
Step 2 : Annotate code with meta-data



```
@Component
public class ComponentD {

    @Resource private A a;
```

Step 3 : implement tools with meta-data



```
Class<Component> annotationClass = Component.class;
Component annotation = (Component) clss.getAnnotation(annotationClass);
if (annotation != null) {
    String annotationValue = annotation.value();
    System.out.println("found class with annotation @Component(" + annotationValue + ")")
}
```

Conclusion

- Conclusion
  - Java comes from Smalltalk (all is Object)
  - Is THE Type Checked – Reflective - Secure – Coherent – General Purpose Langage
  - With only syntax from C++
  - Huge Step Forward with Reflection !!!!!!!!!!
  - => Huge Frameworks Choice and Community
  - DotNet has copyed everything from Java
    (and added more features
    … but abandonning the OLE/ActiveX/Com/DCom tries)

More Info

- Questions ??
  arnaud.nauwynck@gmail.com

- TP …

- Ref:
  - Google java reflection api
  - This document:
    http://arnaud.nauwynck.chez-alice.fr/
    Intro-JavaIntrospection.pdf