Java Fundamentals: the Java Reflection API and Method Handles

INTRODUCING JAVA REFLECTION

José Paumard
PHD, JAVA CHAMPION, JAVA ROCK STAR
@JosePaumard https://github.com/JosePaumard

Reflection API and Method Handles

- Reflection API introduced in Java 1
- Method Handles introduced in Java 7
- read and modify the content of an object
- without knowing its class or structure
- how to discover the content of an object

Why is this API so important?

- all the major Java frameworks use it!
- Hibernate, EclipseLink
- JAX-B, JSON-B
- Spring, CDI, Guice
- JAX-RS, JAX-WS
- JUnit, ...

This is a Java course

Fair knowledge of the language and its main API

How to write classes, what is an annotation

This is a fundamental course

Agenda of the Course

This course is application oriented

Introducing the API

How it works on 2 examples:

- Object Relational Mapping

- Dependency Injection

Performances and Method Handles

Agenda

Let us define the technical terms!

Classes: Class, Field, Method

How to get information on a class

Understanding the patterns

Introducing The Reflection API

There are several classes:

- the class Class
- the class Field
- the class Method
- the class Constructor
- the class Annotation

Each of those classes provides a model

For a fundamental element of a class

The Class Named Class

So there is a class named Class

How to get an instance?

What can be done with it?

You cannot create a class instance

You can query an object for its class

You can get a class by its name, known at compile time

You can get a class by its name, known at runtime

```java
String hello = "Hello";
Class helloClass = hello.getClass();

String world = "World";
Class worldClass = world.getClass();
```

This getClass() method is declared on the Object class

There is only one instance of Class for a given class

```java
Class<?> getClass();

Class<?> helloClass = "Hello".getClass();

Class<String> helloClass = "Hello".getClass(); // Compile ERROR!!!
```

Class is a class with a parameter

So, some affectations do compile

```
Class<?> getClass();

Class<?> helloClass = "Hello".getClass();

Class<String> helloClass = "Hello".getClass(); // Compile ERROR!!!
Class<Object> helloClass = "Hello".getClass(); // Compile ERROR!!!
```

Class is a class with a parameter

So, some affectations do compile

And some do not!

```java
Class<?> getClass();

Class<?> helloClass = hello.getClass();

Class<? extends String> helloClass = "Hello".getClass();
Class<? extends Object> helloClass = "Hello".getClass();
```

Class is a class with a parameter

So, some affectations do compile

And some do not!

```
Class<?> stringClass = String.class;

String className = "java.lang.String";
Class<?> stringClass = Class.forName(className);
```

You can also get a Class object from a known class

And from the name of a class

(Beware of exceptions)

```java
Class<?> clss = "Hello".getClass();
Class<?> clss = String.class;
Class<?> clss = Class.forName("java.lang.String");
```

Here are the three patterns to get a Class instance:

- from an object

- from a known class

- from the name of a class

Getting Information on a Class

From the Class object, we can:

- get the super classes

- get the implemented interfaces, if any

```
Class<?> clss = "Hello".getClass();
Class<?> superClass = clss.getSuperClass();

Class<?>[] interfaces = clss.getInterfaces();
```

getSuperClass(): returns the only super class

The super class of Object is null

getInterfaces(): returns the interfaces, or an empty array

Getting the Fields of a Class

There are many methods in Class

- fields

- methods and constructors

```java
Class<?> cls = Person.class;

Field field             = cls.getField("age");
Field[] declaredFields = cls.getDeclaredFields();
Field[] fields          = cls.getFields();
```

Three methods to get the fields of a class:

- getField(name)

- getDeclaredFields(): declared in the class

- getFields(): public fields, including inherited

```java
public class Person {

    private int age;
    private String name;

    // getters and setters
}
```

```java
Class<?> clss = Person.class;

Field[] fields = clss.getFields();
```

Suppose we have a Person class

And we get the fields of this class

Then what we get is an empty array

```java
public class Person {

    private int age;
    private String name;

    // getters and setters
}
```

```java
Class<?> clss = Person.class;

Field[] fields = clss.getDeclaredFields();
```

Suppose we have a Person class

And we get the declared fields of this class

Then we get the two fields age and name in the array

The “non-declared” elements of a class

Are the elements declared in this class and all the super classes

But only the public ones

The “declared” elements of a class

Are the elements declared in this class:

- private
- protected
- public

With no inherited element

Getting the Methods of a Class

```java
Class<?> clss = Person.class;

Method method =
      clss.getMethod("setName", String.class);

Method[] declaredMethods = clss.getDeclaredMethods();
Method[] methods         = clss.getMethods();
```

Three methods to get the methods of a class:

- getMethod(name, types)

- getDeclaredMethods(): declared in the class

- getMethods(): public methods, including inherited

```java
Class<?> clss = Person.class;

Constructor constructor =
      clss.getConstructor(Class<?>... types);

Constructor[] declaredConstructors =
      clss.getDeclaredConstructors();
Constructor[] constructors          = clss.getConstructors();
```

Three methods to get the constructors of a class:

- getConstructor(types)

- getDeclaredConstructors(): declared in the class

- getConstructors(): public constructors declared in the class

Reading the Modifiers

The modifiers tell if a field or a method is:

- static or not
- abstract or not
- final or not
- public / private / protected
- synchronized / volatile
- native

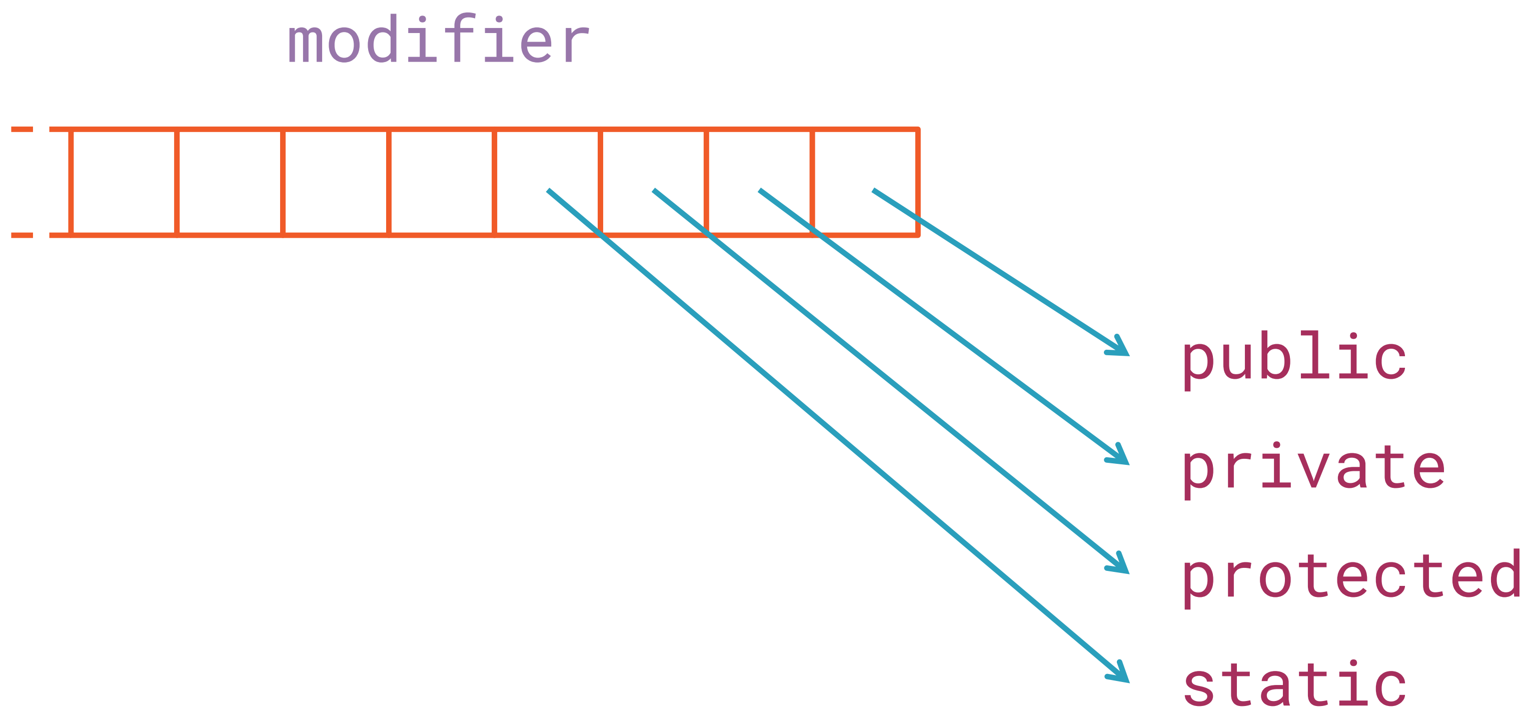
There is a method for that

Available on Field, Method, Constructor

getModifiers()

That returns an int

Understanding Modifiers



```java
Field field = clss.getField("name");
int modifiers = field.getModifiers();

boolean isPublic = modifiers && 0x00000001;

boolean isPublic = Modifier.isPublic(modifiers);
```

It is possible to check if a field is public by using the correct bit mask

Very tedious and error prone…

There is a Modifier class to do that

Demo

Let us see some code!

Let us create a simple bean

And see how we can use reflection on it

Module Wrap Up

What did you learn?

How to access the elements of a class

- the super classes and interfaces

- the fields

- the methods and constructors

- the modifiers