


Java & Clojure interop

Como ya sabemos, Clojure se compila en Java y se ejecuta sobre la JVM. Vamos a utilizar esto para poder tener las fuentes de nuestro proyecto tanto en Java como en Clojure y poder elegir el paradigma según lo que sea más conveniente para el dominio del problema que se quiera resolver.

Se podría tener un proyecto Clojure con dependencias a módulos en Java o un proyecto Java con dependencias a un módulo en Clojure.

El primer caso, es directo. Clojure reconoce de forma automática las bibliotecas de Java, por ejemplo:



```
Clojure 1.8.0
Java HotSpot(TM) 64-Bit Server VM 1.8.0_91-b14
> (.toUpperCase, "tecnicas de disenio 7510")
=> "TECNICAS DE DISENIO 7510"
```

Pero, dado que en este curso estamos más acostumbrados a los proyectos de Java, en este documento nos vamos a enfocar a utilizar Clojure desde un proyecto Java.

Nuestro proyecto Java va a utilizar un módulo de Clojure como dependencia y podrá hacer uso de las funciones definidas en el mismo.

En los siguientes ítems se va a explicar cómo hacer esto y qué puntos tener en cuenta para que todo funcione correctamente.

Proyecto en Clojure

Nuestro proyecto Clojure deberá tener algunas configuraciones adicionales dado que:

- **Necesitamos generar un .jar**
- **Necesitamos definir el namespace con el cual vamos a referenciar desde el módulo Java**

Primero, vamos a generar un proyecto Clojure utilizando Lein y luego, editaremos el archivo 'project.clj':

1. Generamos el proyecto

```
>lein new com.ar.fiuba.tdd.clojure.template
```

2. Entramos en el directorio y editamos el archivo project.clj

```
> cd com.ar.fiuba.tdd.clojure.template
> vim project.clj
```

Y lo dejamos de la siguiente forma⁽¹⁾:

```
(defproject com.ar.fiuba.tdd.clojure.template "0.1.0-SNAPSHOT"
  :description ""
  :url ""
  :license {:name "Eclipse Public License"
            :url "http://www.eclipse.org/legal/epl-v10.html"}
  :dependencies [[org.clojure/clojure "1.8.0"]]
  :aot :all
  :main ar.fiuba.tdd.clojure.template
)
```

(1) “:aot” significa Ahead-of-time Compilation, lo cual indica que queremos generar **inmediatamente** el bytecode para armar la clase, en lugar de esperar a que lo genere a medida que se necesita (Más información en la bibliografía).

3. Instalamos las dependencias

```
>lein deps
```

4. Creamos nuestros archivos .clj para generar un 'Hello world' desde clojure

```
(ns com.ar.fiuba.tdd.clojure-template  
  (:gen-class  
   :name com.ar.fiuba.tdd.clojure-template.clojure-template)  
  )
```

```
(defn -main []  
  (println "Hello, World!")  
  )
```

IMPORTANTE: la función 'main' deberá comenzar con - (guión medio) y gen-class indica que se debe generar un .class con el nombre detallado

5. Corremos para ver que todo funciona ok

```
> lein run  
> lein test
```

6. Generamos el Jar

```
> lein jar
```

7. Instalamos la dependencia en el repo local

```
> lein install
```

8. Una vez creado el jar, podemos agregarlo a nuestro proyecto java!

Proyecto Java usando Gradle

En nuestro proyecto Java + Gradle se deberá hacer lo siguiente:

- **Agregar el repo local en la configuración de Gradle, para que nos deje usar nuestro nuevo módulo como dependencia**
- **Configurar la dependencia**

1. Agregamos el repo local:

```
repositories {  
    ...  
    mavenLocal()  
}
```

2. Agregamos las dependencias:

```
dependencies {  
    compile group:'org.clojure', name:'clojure', version:'1.8.0'  
    compile group:'ar.fiuba.tdd.clojure.template',  
name:'ar.fiuba.tdd.clojure.template', version:'0.1.0-SNAPSHOT'  
    testCompile group: 'junit', name: 'junit', version: '4.11'  
}
```

3. Y lo usamos como cualquier otra dependencia

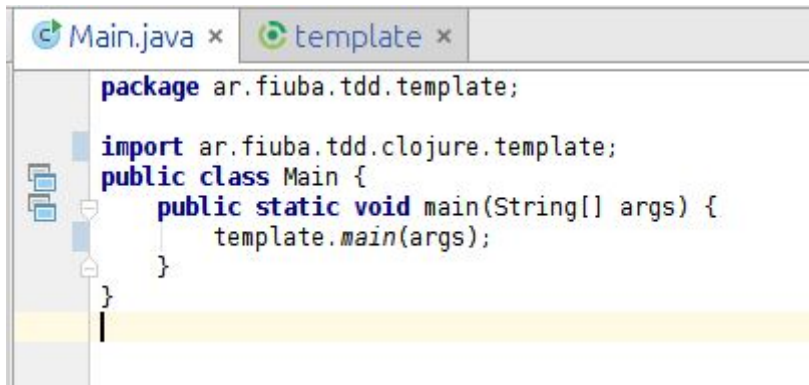
```
Import ar.fiuba.tdd.template.clojure
```

4. Cómo debería quedar la configuración de gradle:



```
main.java x template x  
apply from: 'gradle/tatjar.gradle'  
apply from: 'gradle/quality.gradle'  
  
sourceCompatibility = 1.8  
targetCompatibility = 1.8  
  
repositories {  
    jcenter()  
    maven {  
        url "http://nexus-tecnicas7510.rhcloud.com/nexus/content/repositories/third-party"  
    }  
    mavenLocal()  
}  
  
dependencies {  
    compile group:'org.clojure', name:'clojure', version:'1.8.0'  
    compile group:'ar.fiuba.tdd.clojure.template', name:'ar.fiuba.tdd.clojure.template', version:'0.1.0-SNAPSHOT'  
    testCompile group: 'junit', name: 'junit', version: '4.11'  
}  
  
configurations.all {  
    resolutionStrategy {  
        force 'xml-apis:xml-apis:1.4.01'  
    }  
}
```

5. Cómo lo usamos en nuestro código Java:



The screenshot shows an IDE with two tabs: 'Main.java' and 'template'. The 'Main.java' tab is active, displaying the following code:

```
package ar.fiuba.tdd.template;

import ar.fiuba.tdd.clojure.template;

public class Main {
    public static void main(String[] args) {
        template.main(args);
    }
}
```

The code is syntactically highlighted, and the 'template' package is visible in the left-hand sidebar.

Atención: son funciones estáticas!

Bibliográfia:

- <https://stackoverflow.com/questions/2181774/calling-clojure-from-java>
- <https://clojure.org/reference/compilation>
- <https://clojure.github.io/clojure/clojure.core-api.html#clojure.core/gen-class>
- https://kotka.de/blog/2010/02/gen-class_how_it_works_and_how_to_use_it.html
- <http://clojure-doc.org/articles/language/interop.html>