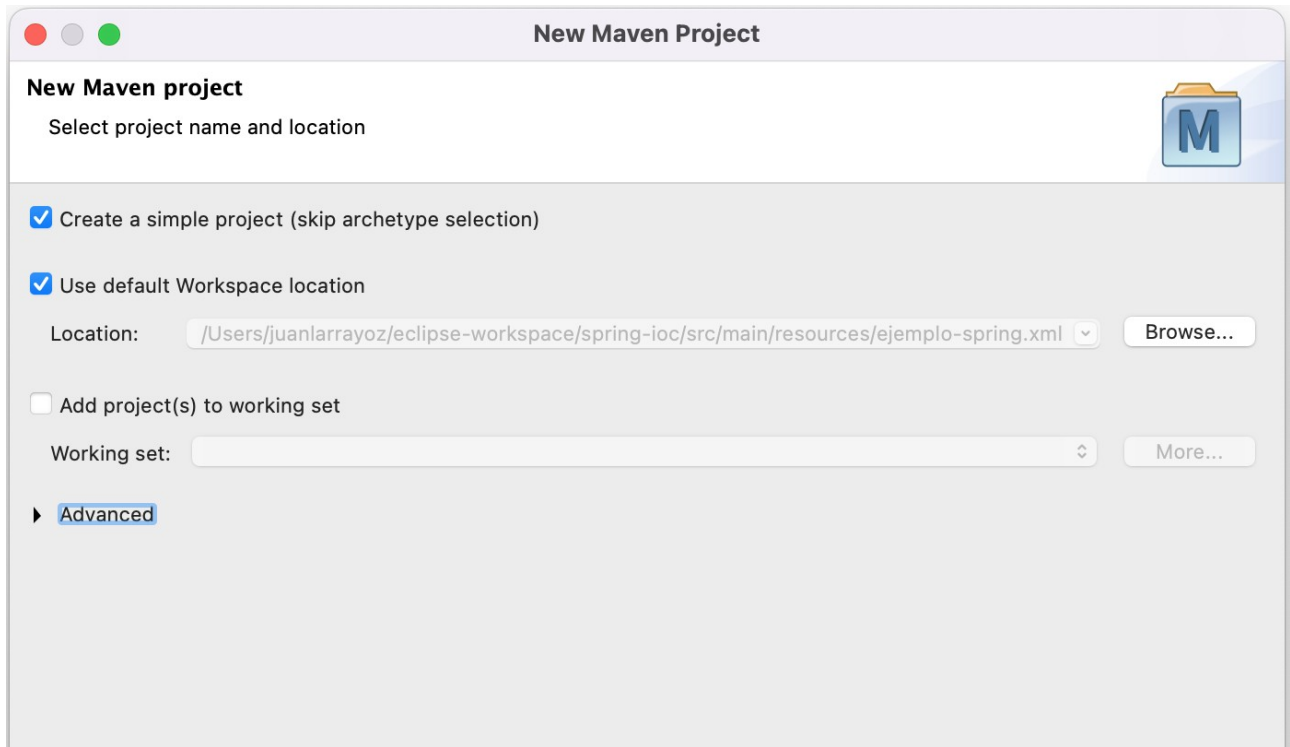


Ejercicio – Spring IoC

1) Vamos a crear un nuevo proyecto maven para poder trabajar en el.



The screenshot shows the 'New Maven Project' dialog box. The title bar says 'New Maven Project'. Inside, the subtitle is 'New Maven project' and the instruction is 'Select project name and location'. There is a folder icon with an 'M' on the right. The options are:

- ☒ Create a simple project (skip archetype selection)
- ☒ Use default Workspace location

The 'Location' field is set to '/Users/juanlarrayoz/eclipse-workspace/spring-ioc/src/main/resources/ejemplo-spring.xml' with a 'Browse...' button next to it.

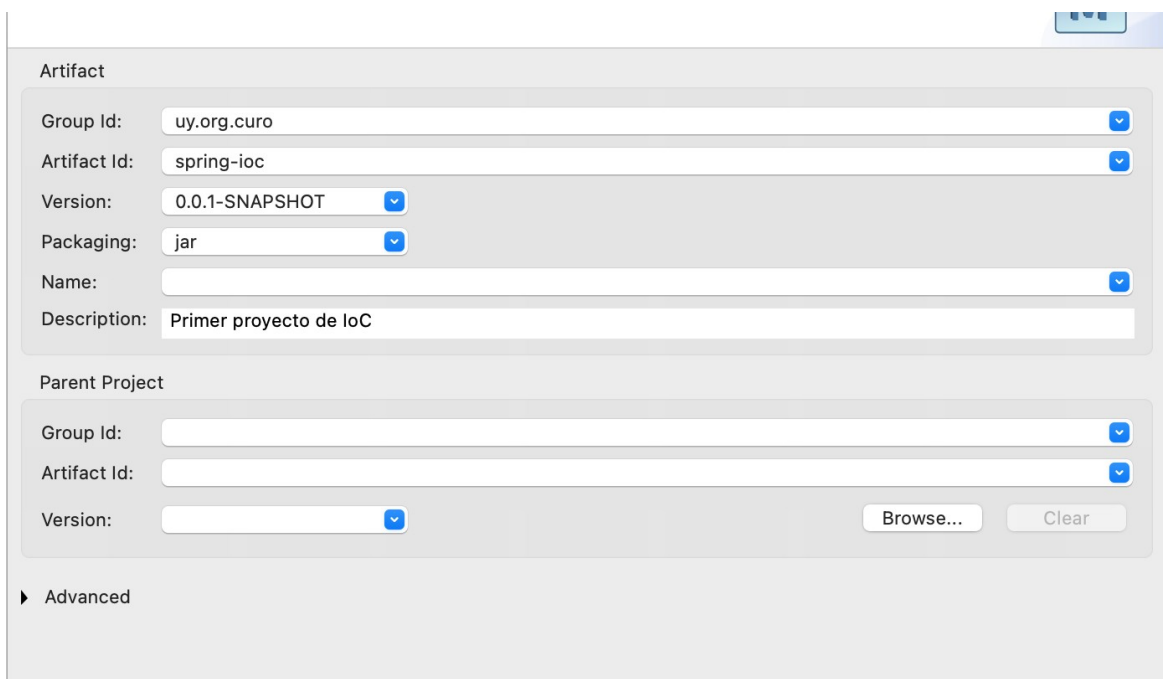
☐ Add project(s) to working set

The 'Working set' field is empty with a 'More...' button next to it.

There is a section titled 'Advanced' with a right-pointing arrow.

Debemos marcar la opción “Create a simple”

Luego debemos completar los valores solicitados por el siguiente wizard:



The screenshot shows the 'Artifact' and 'Parent Project' sections of the Maven wizard.

Artifact

- Group Id: uy.org.curo
- Artifact Id: spring-ioc
- Version: 0.0.1-SNAPSHOT
- Packaging: jar
- Name:
- Description: Primer proyecto de IoC

Parent Project

- Group Id:
- Artifact Id:
- Version:

There are 'Browse...' and 'Clear' buttons next to the 'Version' field.

There is a section titled 'Advanced' with a right-pointing arrow.

2) Agregamos las dependencias necesarias.

Vamos a agregar las siguientes dependencias y properties en el pom.xml de nuestro proyecto:

```
<properties>
  <java.version>17</java.version>
</properties>

<dependencies>
  <!--
https://mvnrepository.com/artifact/org.springframework/spring-core
-->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>6.0.0</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>6.0.0</version>
  </dependency>
</dependencies>
```

3) Vamos a agregar al proyecto la siguiente estructura de packages:



Inyección por constructor

4) Creamos la clase ConstructorMessage.java en el package beans:

```
public class ConstructorMessage {
    private String mensaje = null;

    public ConstructorMessage(String mensaje) {
        super();
        this.mensaje = mensaje;
    }

    public String getMensaje() {
        return mensaje;
    }
}
```

5) Creamos el archivo "ejemplo-message.xml" el cual oficiara de archivo de configuración para Spring IoC. Este archivo debe estar contenido en la carpeta resources.

```
<?xml version="1.0" encoding="UTF-8"?>
<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
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        https://www.springframework.org/schema/context/spring-context.xsd">

    <!-- Definimos la DI para nuestro bean -->
    <bean id="consMessage" class="uy.edu.curso.example.ConstructorMessage">
        <constructor-arg value="Mensaje inyectado por constructor"></constructor-arg>
    </bean>
</beans>
```

Nota: corregir el fully qualified name del bean

6) Una vez que tenemos el bean de ejemplo y la configuración XML para la inyección, vamos a necesitar correr un programa para realizar la prueba. Con ese objetivo creamos la clase MessageMain.java en el package mains:

```
public class MessageMain {
    public static void main(String[] args) {

        //Vamos a acceder al contenedor IOC a través de la interfaz ApplicationContext
        ApplicationContext container = new ClassPathXmlApplicationContext("ejemplo-message.xml");

    }
}
```

7) Ahora vamos a pedirle al container el bean utilizando 3 métodos diferentes:

```
ConstructorMessage consMessage =  
container.getBean(ConstructorMessage.class);  
  
ConstructorMessage consMessage1 = (ConstructorMessage)  
container.getBean("consMessage");  
  
ConstructorMessage consMessage2 =  
container.getBean("consMessage", ConstructorMessage.class);
```

8) Verificar por código si las instancias obtenidas son objetos diferentes o no.

Ejecutar el método getMessage de cualquiera de los beans para verificar que efectivamente se haya inyectado en el constructor el String definido en el XML.

Inyección por setter

9) Ahora vamos a probar la inyección mediante el método setter. Para eso, vamos a realizar una copia de ConstructorMessage.java y la vamos a llamar setterMessage.java. La clase debe ver se la siguiente forma:

```
public class setterMessage {  
    private String mensaje = null;  
  
    public setterMessage() {  
        super();  
    }  
  
    public String getMensaje() {  
        return mensaje;  
    }  
  
    public void setMensaje(String mensaje) {  
        this.mensaje = mensaje;  
    }  
}
```

10) Debemos modificar el archivo XML definido previamente y agregar la config para el nuevo bean:

```
<!-- Definimos la DI para nuestro bean mediante setter-->
<bean id="consMessage" class="uy.org.curso.beans.ConstructorMessage">
  <property name="mensaje" value="Mensaje inyectado por setter"></property>
</bean>
```

11) Agregar en el main el código para obtener del container una instancia del nuevo bean. ¿Funcionó?, corregir el error.

12) Colocar un breakpoint en el constructor de ConstructorMessage y otro en el setMensaje de SetterMessage.

Analizar cuando el debug para en los breakpoint en que momento lo hace: (cuando se llama al getBean, cuando se crea el container)

13) Modificar el XML del bean setMessage y agregar la propiedad:

```
lazy-init="true"
```

Volver a ejecutar el main en debug y volver a analizar el comportamiento del punto 12.

14) Definir en el archivo XML de configuración otro bean por ejemplo de consMessage y ponerle como id "auxMessage". Cambiar el mensaje que se inyecta.

15) Volver a ejecutar el main, ¿qué paso?. Prestar atención al stacktrace de la consola.