



# Spring\_

Parte I

{desafío}  
latam\_







# Inicio

{desafío}  
latam\_



15 Minutos

*/\** Crear un algoritmo desde el enfoque de inyección de dependencias.*\*/*

**Objetivo**



# Desarrollo

{desafío}  
latam\_



150 Minutos

**/\*Inyección de dependencias\*/**

# ¿Qué es inyección de dependencias?

- Patrón de diseño orientado a objetos que permite dividir y repartir las responsabilidades de un programa.

## Ejemplo:

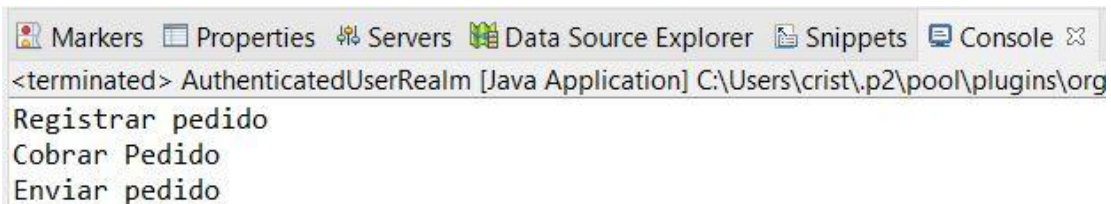
Pedidos de comida, necesitamos registrar los pedidos, para luego cobrarlos y enviarlos. ServicioComidaDomicilio tiene tres tareas:

```
package
com.desafiolatam.inyeccion_dependencias_ejemplo_uno;
public class ServicioComidaDomicilio {
    public void enviar() {
        System.out.println("Registrar pedido");
        System.out.println("Cobrar Pedido");
        System.out.println("Enviar pedido");
    }
}
```

## Método Main:

```
package com.desafiolatam.inyeccion_dependencias_ejemplo_uno;
public class Main {
    public static void main(String[] args) {
        ServicioComidaDomicilio servicio = new
ServicioComidaDomicilio();
        servicio.enviar();
    }
}
```

## Resultado:



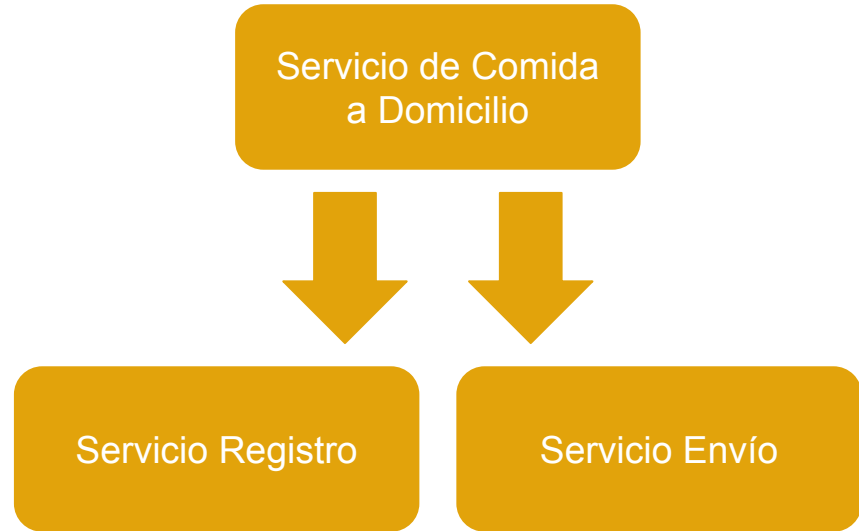
The screenshot shows an IDE console window with the following tabs: Markers, Properties, Servers, Data Source Explorer, Snippets, and Console. The console output is as follows:

```
<terminated> AuthenticatedUserRealm [Java Application] C:\Users\cris\p2\pool\plugins\org
Registrar pedido
Cobrar Pedido
Enviar pedido
```



# Separando los servicios

- Cada servicio se especializa en una función.
- Cada uno se comporta de forma independiente.



**Ejemplo:**

**(ServicioRegistroPedido):**

```
package
com.desafiolatam.inyeccion_dependencias_ejemplo_dos;
public class ServicioRegistroPedido {
    public void registrarPedido() {
        System.out.println("Registrar Pedido");
    }
}
```

**(ServicioCobroPedido):**

```
package
com.desafiolatam.inyeccion_dependencias_ejemplo_dos;
public class ServicioCobroPedido {
    public void cobrarPedido() {
        System.out.println("Cobrar Pedido");
    }
}
```

## (ServicioEnvioPedido)

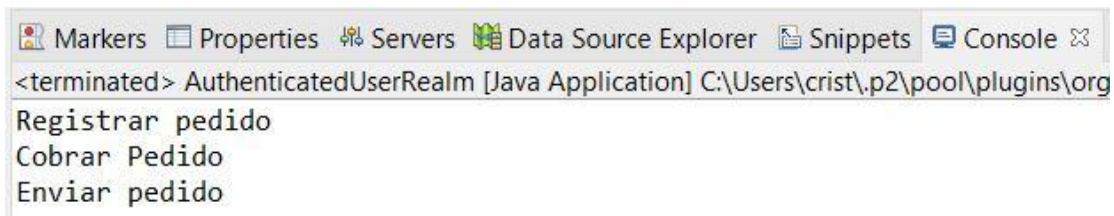
```
package  
com.desafiolatam.inyeccion_dependencias_ejemplo_dos;  
public class ServicioEnvioPedido {  
    public void enviarPedido() {  
        System.out.println("Enviar Pedido");  
    }  
}
```

## Se definen los objetos en el constructor de la clase ServicioComidaDomicilio:

```
package com.desafiolatam.inyeccion_dependencias_ejemplo_dos;

public class ServicioComidaDomicilio {
    ServicioRegistroPedido servicio_registro;
    ServicioCobroPedido servicio_cobro;
    ServicioEnvioPedido servicio_envio;
    public ServicioComidaDomicilio() {
        this.servicio_registro = new ServicioRegistroPedido();
        this.servicio_cobro = new ServicioCobroPedido();
        this.servicio_envio = new ServicioEnvioPedido();
    }
    public void enviar() {
        servicio_registro.registrarPedido();
        servicio_cobro.cobrarPedido();
        servicio_envio.enviarPedido();
    }
}
```

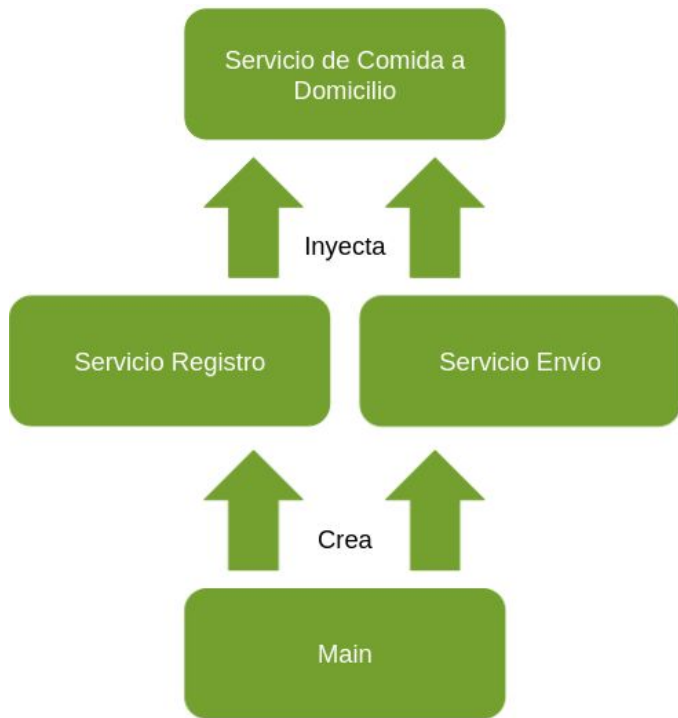
## Resultado:



The screenshot shows an IDE console window with a toolbar at the top containing icons for Markers, Properties, Servers, Data Source Explorer, Snippets, and Console. The console text displays the application's output, which includes a terminated status, the application name, its path, and three method calls.

```
<terminated> AuthenticatedUserRealm [Java Application] C:\Users\cris\p2\pool\plugins\org  
Registrar pedido  
Cobrar Pedido  
Enviar pedido
```

Se puede realizar la operación anterior inyectando las dependencias al servicio de correo.



**Modificación  
diagrama  
Servicio de  
Comida a  
Domicilio**

El servicio si depende de los otros tres servicios, pero él no los genera, sólo los utiliza.

```
package com.desafiolatam.inyeccion_dependencias_ejemplo_tres;
public class ServicioComidaDomicilio {
    ServicioRegistroPedido servicio_registro;
    ServicioCobroPedido servicio_cobro;
    ServicioEnvioPedido servicio_envio;
    public ServicioComidaDomicilio(ServicioRegistroPedido
servicio_registro, ServicioCobroPedido servicio_cobro, ServicioEnvioPedido
servicio_envio) {
        this.servicio_registro = servicio_registro;
        this.servicio_cobro = servicio_cobro;
        this.servicio_envio = servicio_envio;
    }
    public void enviar() {
        servicio_registro.registrarPedido();
        servicio_cobro.cobrarPedido();
        servicio_envio.enviarPedido();
    }
}
```

- (ServicioRegistroPedido), (ServicioEnvioPedido), (ServicioCobroPedido), son definidos en el método Main y son utilizados por (ServicioComidaDomicilio).

```
package com.desafiolatam.inyeccion_dependencias_ejemplo_tres;
public class Main {
    public static void main(String[] args) {
        ServicioComidaDomicilio servicio = new
ServicioComidaDomicilio( new ServicioRegistroPedido(), new
ServicioCobroPedido(), new ServicioEnvioPedido());
        servicio.enviar();
    }
}
```



## ¿De qué sirve esto?

- Permite la extensibilidad de los servicios (capacidad de un programa para soportar nuevas funcionalidades).

### Continuemos con el Ejemplo:

Verificar el pedido antes de enviarlo.

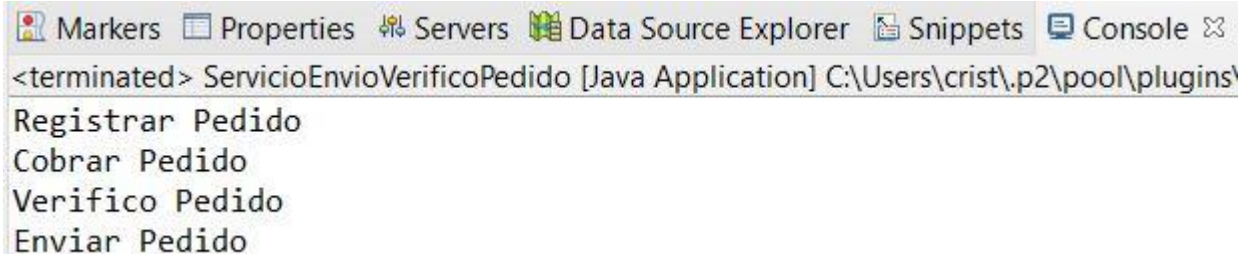
```
package com.desafiolatam.inyeccion_dependencias_ejemplo_tres;
public class ServicioEnvioVerificoPedido extends
ServicioEnvioPedido {
    public void enviarPedido() {
        System.out.println("Verifico Pedido");
        super.enviarPedido();
    }
}
```

Modificando la definición en el método Main:

```
package com.desafiolatam.inyeccion_dependencias_ejemplo_tres;

public class Main {
    public static void main(String[] args) {
        ServicioComidaDomicilio servicio = new ServicioComidaDomicilio( new
        ServicioRegistroPedido(), new ServicioCobroPedido(), new
        ServicioEnvioVerificoPedido());
        servicio.enviar();
    }
}
```

Resultado:



The screenshot shows an IDE interface with a toolbar at the top containing icons for Markers, Properties, Servers, Data Source Explorer, Snippets, and Console. The Console window is active and displays the output of a Java application. The output starts with a terminated status and then lists four actions: Registrar Pedido, Cobrar Pedido, Verifico Pedido, and Enviar Pedido.

```
<terminated> ServicioEnvioVerificoPedido [Java Application] C:\Users\crist\.p2\pool\plugins\
Registrar Pedido
Cobrar Pedido
Verifico Pedido
Enviar Pedido
```

# A través de interfaces

- Es un objeto abstracto.
- Conjunto de atributos que especifica lo que se debe hacer y no cómo se debe hacer.

## Ejemplo:

Interfaz que posee dos métodos.

```
package com.inyeccion_dependencias_ejemplo_cinco;
public interface IServicioEnvioPedido {
    public void enviarPedido();
    public void verificarPedido();
}
```

Implementando la interfaz, generando la clase ServicioEnvioPedido.

```
package com.inyeccion_dependencias_ejemplo_cinco;
public class ServicioEnvioPedido implements
IServicioEnvioPedido{
    public void enviarPedido() {
        System.out.println("Enviar Pedido");
    }
    public void verificarPedido() {
        System.out.println("Verifico pedido");
    }
}
```

```
package com.desafiolatam.inyeccion_dependencias_ejemplo_cuatro;

public class ServicioComidaDomicilio {
    ServicioRegistroPedido servicio_registro;
    ServicioCobroPedido servicio_cobro;
    IServicioEnvioPedido servicio_envio;

    public ServicioComidaDomicilio(ServicioRegistroPedido
servicio_registro, ServicioCobroPedido servicio_cobro,
                                IServicioEnvioPedido servicio_envio)
    {
        this.servicio_registro = servicio_registro;
        this.servicio_cobro = servicio_cobro;
        this.servicio_envio = servicio_envio;
    }

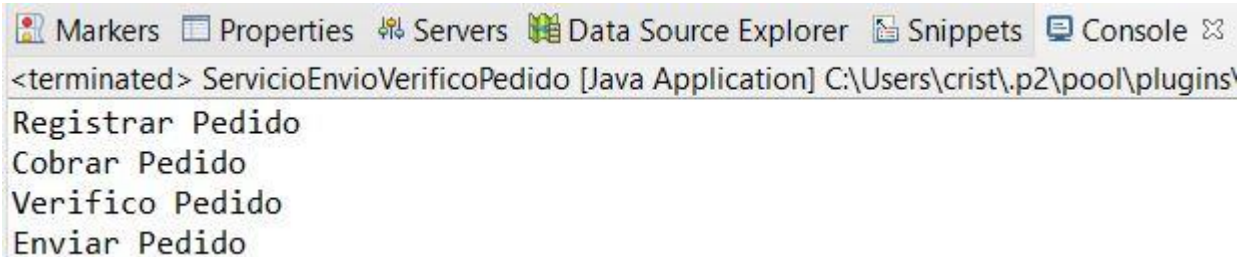
    public void enviar() {
        servicio_registro.registrarPedido();
        servicio_cobro.cobrarPedido();
        servicio_envio.verificarPedido();
        servicio_envio.enviarPedido();
    }
}
```

En el  
ServicioComidaDomicilio,  
asignamos la interfaz como  
servicio de envío  
(IServicioEnvioPedido)

Se corrige el método Main :

```
package com.inyeccion_dependencias_ejemplo_cinco;
public class Main {
    public static void main(String[] args) {
        ServicioComidaDomicilio servicio= new
        ServicioComidaDomicilio(
            new ServicioRegistroPedido(),
            new ServicioCobroPedido(),
            new ServicioEnvioPedido()
        );
        servicio.enviar();
    }
}
```

Resultado:



```
<terminated> ServicioEnvioVerificoPedido [Java Application] C:\Users\crist\.p2\pool\plugins\
Registrar Pedido
Cobrar Pedido
Verifico Pedido
Enviar Pedido
```

**/\*Maven y manejo de dependencias\*/**

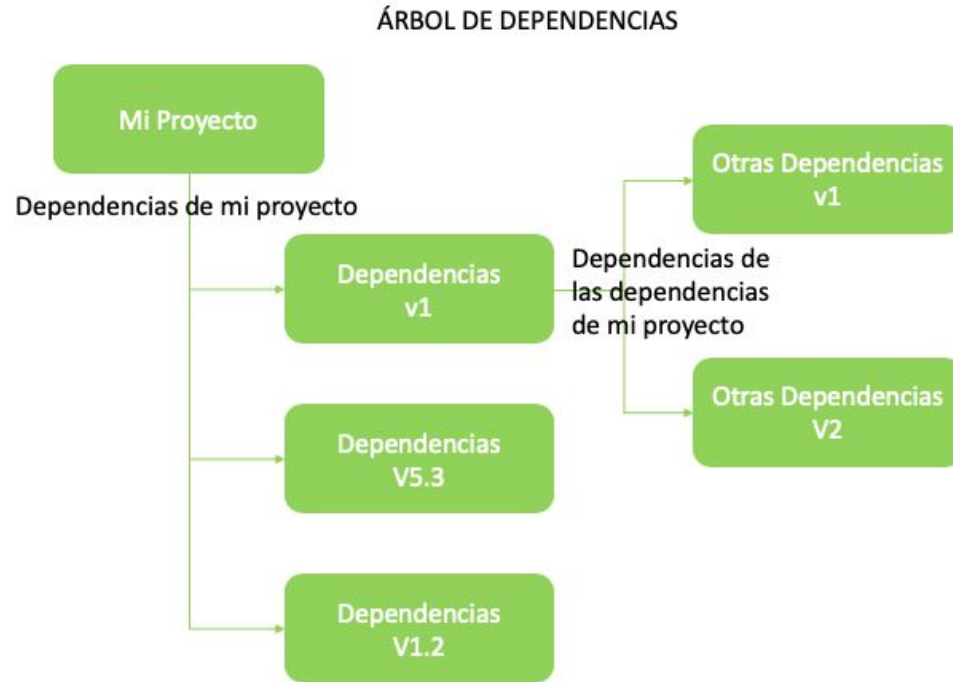
# ¿Qué es Maven?

- Herramienta para la gestión y el manejo de librerías.
- Define un ciclo de vida para la ejecución de un objeto.

Se basa en siete etapas:

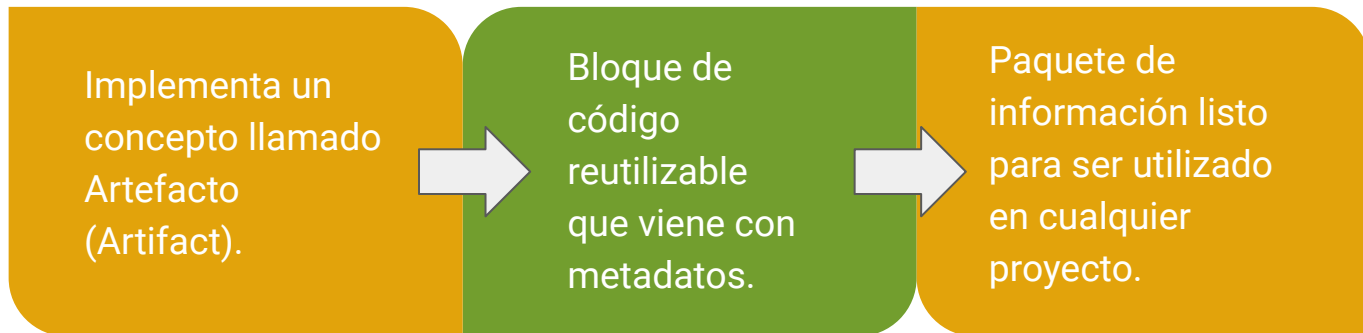
Validar	Que la estructura del proyecto esté correcta.
Compilar	Compila el código.
Probar	Genera pruebas unitarias en el código compilado.
Empaquetar	Toma el código compilado y genera paquetes en formato para la distribución.
Verificar	Ejecuta la verificación de los resultados de las pruebas.
Instalar	Paquetes en una carpeta local para poder utilizarlos como dependencias.
Implementación	Copia el paquete final en un repositorio remoto para compartirlo.

Permite importar repositorios o librerías al sistema que estemos desarrollando de manera rápida y sencilla.





# ¿Cómo hace maven para conocer la dependencia de las librerías que estoy utilizando?



# Concepto de Artefacto

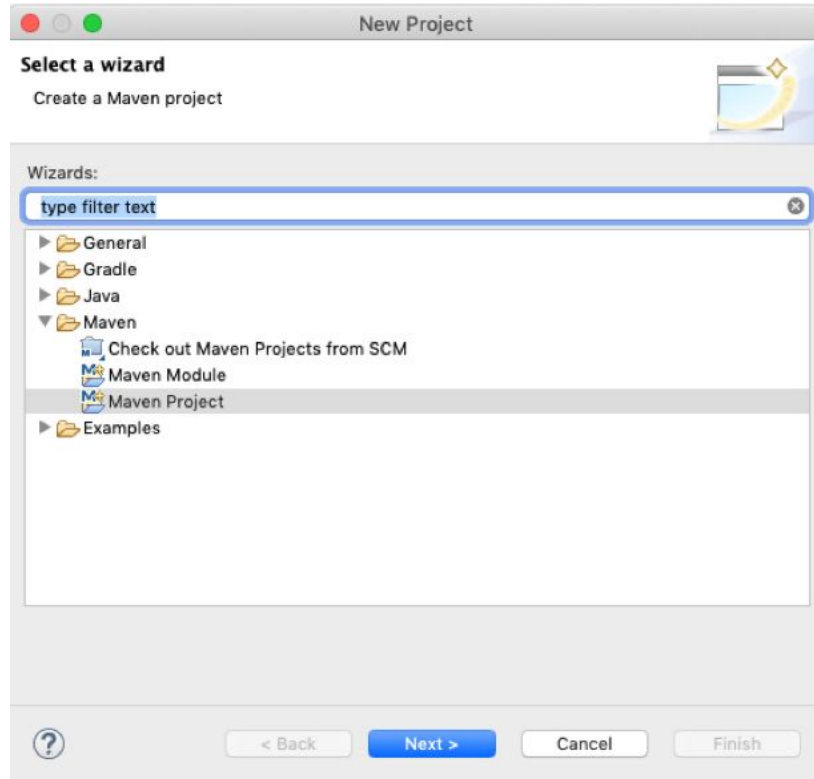
- Proyecto que gestiona Maven.
- Permite conocer todo lo relacionado a la librería o directorio.



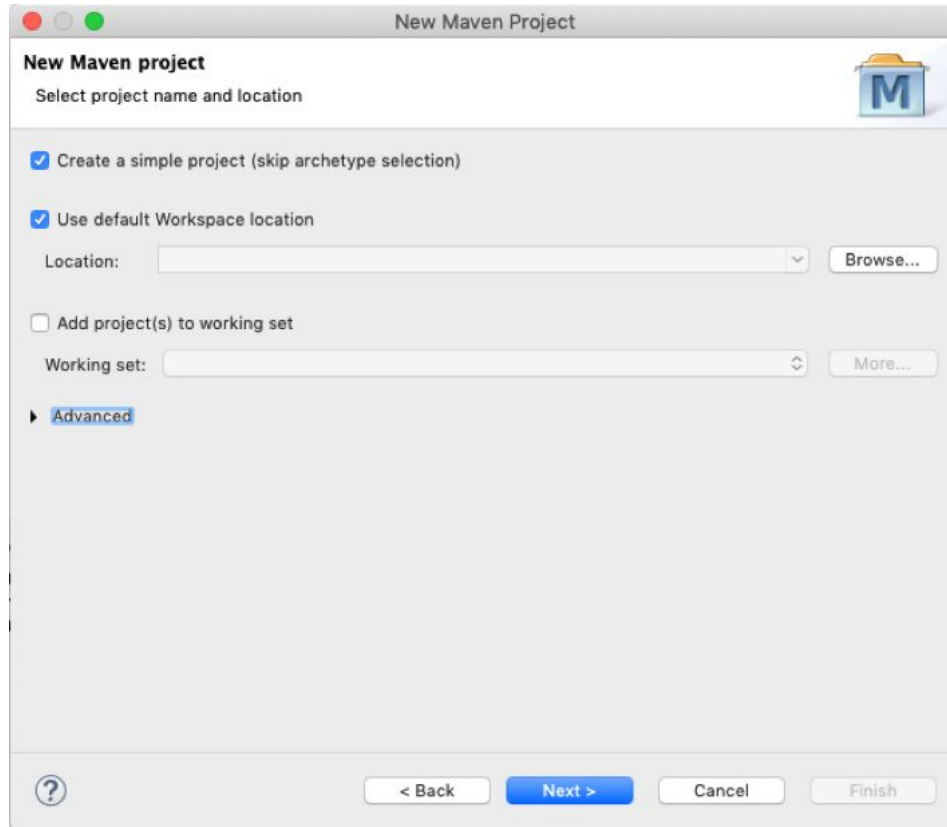
- **(Project Object Model):**
- Maneja toda la información relacionada al proyecto.

**pom.xml**

# Creando un proyecto maven



- “Create a simple project”:



- Definimos la identificación del grupo y la identificación artefacto.

**New Maven Project**  
Configure project

**Artifact**

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

**Parent Project**

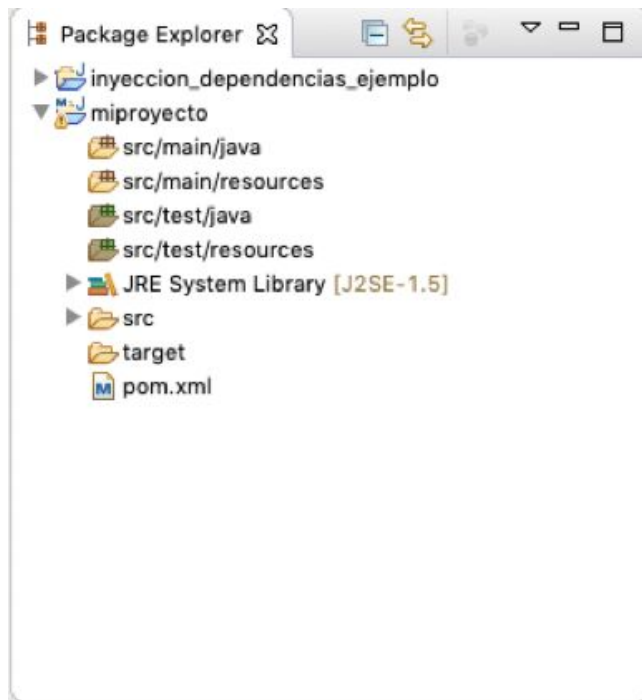
Group Id:

Artifact Id:

Version:

Advanced

- Creará la estructura básica de un proyecto Maven.



- Archivo pom.xml, que generó eclipse automáticamente.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.maven.ejemplo</groupId>
  <artifactId>miproyecto</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</project>
```

**Maven**, utiliza esta información para generar artefactos que podrías utilizar como dependencias de otro proyecto.



# Ejemplo de inyección de dependencias

- Calculadora con operaciones básicas.

```
package com.micalculadora;
public class Calculadora{
    public static double suma (double a, double b) {
        return (a+b);
    }
    public static double multiplica (double a, double b) {
        return (a*b);
    }
    public static double resta (double a, double b) {
        return (a-b);
    }
    public static double divide (double a, double b) {
        return (a/b);
    }
    public static double resto (double a, double b) {
        return (a%b);
    }
}
```

- Buscamos JUnit y accedemos a la segunda opción.

The screenshot shows a web browser window with the URL <https://mvnrepository.com/search?q=junit>. The page features the Maven Repository logo and a search bar containing the text 'junit'. On the left side, there are two panels: 'Repository' and 'Group'. The 'Repository' panel lists various repositories like Central, Sonatype, Spring Plugins, etc. The 'Group' panel lists groups like com.github, org.eclipse, org.apache, etc. The main content area displays 'Found 1715 results' and a sort dropdown set to 'relevance'. Three results are shown:

- JUnit Jupiter API** (3,580 usages, EPL license)  
org.junit.jupiter » junit-jupiter-api  
Module "junit-jupiter-api" of JUnit 5.  
Last Release on Mar 19, 2019
- JUnit** (88,793 usages, EPL license)  
junit » junit  
JUnit is a unit testing framework for Java, created by Erich Gamma and Kent Beck.  
Last Release on May 5, 2019
- JUnit** (1,452 usages, CPL and CPAL licenses)  
junit » junit-dep  
JUnit is a regression testing framework written by Erich Gamma and Kent Beck. It is used by the developer who implements unit tests in Java.  
Last Release on Nov 14, 2012

- Buscamos la última versión estable.

← → ↻ 🏠 🔒 https://mvnrepository.com/artifact/junit/junit

**Indexed Artifacts (14.4M)**

**Popular Categories**

- Aspect Oriented
- Actor Frameworks
- Application Metrics
- Build Tools
- Bytecode Libraries
- Command Line Parsers
- Cache Implementations
- Cloud Computing
- Code Analyzers
- Collections
- Configuration Libraries
- Core Utilities
- Date and Time Utilities
- Dependency Injection
- Embedded SQL Databases
- HTML Parsers
- HTTP Clients
- I/O Utilities

Home » junit » junit

## JUnit

JUnit is a unit testing framework for Java, created by Erich Gamma and Kent Beck.

License	EPL 1.0
Categories	Testing Frameworks
Tags	testing junit
Used By	88,793 artifacts

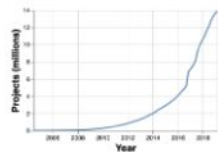
**Note:** This artifact was moved to:

[org.junit.jupiter » junit-jupiter-api](#)

Central (27) Redhat GA (3) Redhat EA (2) JBoss 3rd-party (1) ICM (6) Alfresco (1)

	Version	Repository	Usages	Date
4.13.x	4.13-beta-3	Central	7	May, 2019
	4.13-beta-2	Central	71	Feb, 2019
	4.13-beta-1	Central	55	Nov, 2018
4.12.x	4.12	Central	42,474	Dec, 2014
	4.12-beta-3	Central	30	Nov, 2014
	4.12-beta-2	Central	31	Sep, 2014
	4.12-beta-1	Central	31	Jul, 2014

Indexed Artifacts (14.4M)



Popular Categories

- Aspect Oriented
- Actor Frameworks
- Application Metrics
- Build Tools
- Bytecode Libraries
- Command Line Parsers
- Cache Implementations
- Cloud Computing
- Code Analyzers
- Collections
- Configuration Libraries
- Core Utilities
- Date and Time Utilities
- Dependency Injection
- Embedded SQL Databases
- HTML Parsers
- HTTP Clients
- I/O Utilities

{desafio}  
latam\_

Home » junit » junit » 4.11



## JUnit » 4.11

JUnit is a unit testing framework for Java, created by Erich Gamma and Kent Beck.

License

CPAL 1.0 CPL 1.0

Categories

Testing Frameworks

Organization

JUnit

HomePage

<http://junit.org>

Date

(Nov 14, 2012)

Files

[pom \(2 KB\)](#) [jar \(239 KB\)](#) [View All](#)

Repositories

[Central](#) [AdobePublic](#) [Aspose](#) [Geomajas](#) [Redhat GA](#) [Sonatype](#)

Used By

88,793 artifacts

**Note:** There is a new version for this artifact

New Version

4.13-beta-3

Maven

[Gradle](#)

[SBT](#)

[Ivy](#)

[Grape](#)

[Leiningen](#)

[Buildr](#)

```
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
```

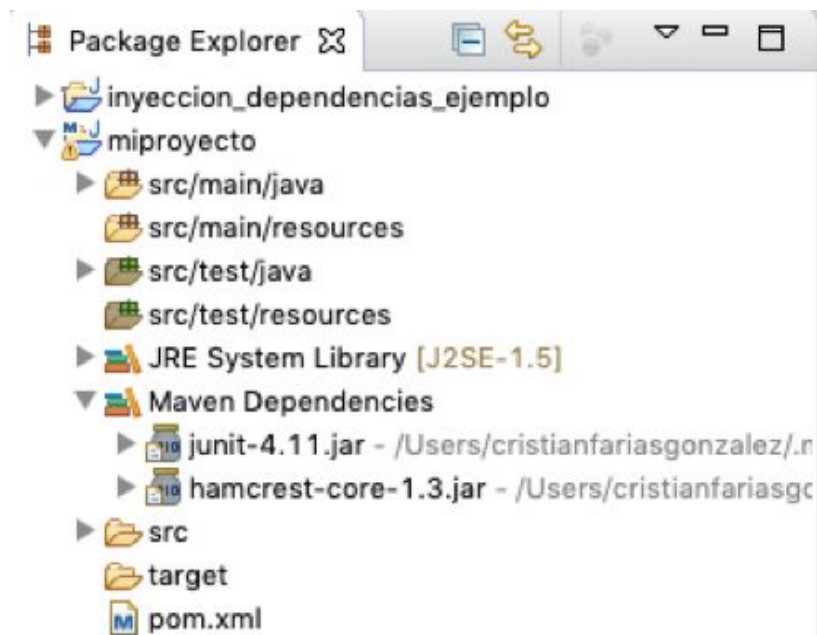
☒ Include comment with link to declaration

Incorporando  
el repositorio  
con Maven

- El archivo pom.xml se verá así.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.calculadora</groupId>
  <artifactId>micalculadora</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
    <!-- https://mvnrepository.com/artifact/junit/junit -->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

- Estructura del proyecto.



- Método assertEquals:

### **assertEquals**

```
public static void assertEquals(double expected,  
                               double actual,  
                               double delta)
```

Asserts that two doubles or floats are equal to within a positive delta. If they are not, an `AssertionError` is thrown. If the expected value is infinity then the delta value is ignored. NaNs are considered equal:

```
assertEquals(Double.NaN, Double.NaN, *) passes
```

#### **Parameters:**

`expected` - expected value

`actual` - the value to check against `expected`

`delta` - the maximum delta between `expected` and `actual` for which both numbers are still considered equal.

Recibe 3 parámetros:

- Valor que espera.
- Valor actual, valor resultado de la operación.
- Delta, que es porcentaje de error de la igualación.

- Archivo de pruebas unitarias:

```
package com.micalculadora;
import org.junit.Test;
import static org.junit.Assert.*;
import com.micalculadora.Calculadora;
public class CalculadoraPrueba {
    @Test
    public void prueba() {
        assertEquals(4,Calculadora.suma(2,2),0);
        assertEquals(4,Calculadora.multiplica(2,2),0);
        assertEquals(0,Calculadora.resta(2,2),0);
        assertEquals(1,Calculadora.divide(2,2),0);
        assertEquals(0,Calculadora.resto(2,2),0);
    }
}
```

- El resultado de esto nos refleja que nuestras pruebas son correctas.





- Generemos el error, para comprobar que la librería realmente funciona:



- La clase que se utilizó para generar el error:

```
package com.miproyecto.prueba;
import org.junit.Test;
import static org.junit.Assert.*;
import com.miproyecto.Calculadora;
public class CalculadoraPrueba {
    @Test
    public void prueba() {
        assertEquals(2,Calculadora.suma(2,2),0);
    }
}
```

**/\*Propuesta de ejercicio\*/**

# Enunciado

- Implementaremos una aplicación que simula un Torneo de Artes Marciales, donde cada personaje se moverá en base a librerías con funciones.

# Desarrollo

- Creamos nuestra interfaces de movimiento y de actividades.

```
package com.batalla;  
public interface IMovimiento {  
    void avanzar();  
    void derecha();  
    void izquierda();  
    void retroceder();  
}  
  
package com.batalla;  
public interface IActividades {  
    void ataqueBasico();  
    void ataqueAvanzado();  
    void defenderAtaque();  
    void esquivarAtaque();  
}
```

- Personaje: utilizaremos la librería de maven Apache Commons Math versión 3.6.1 para utilizar el método para obtener un valor factorial.

#### factorial

```
public static long factorial(int n)
    throws NotPositiveException,
           MathArithmeticException
```

Returns n!. Shorthand for n Factorial, the product of the numbers 1, ..., n.

#### Preconditions:

- $n \geq 0$  (otherwise `IllegalArgumentException` is thrown)
- The result is small enough to fit into a long. The largest value of  $n$  for which  $n! < \text{Long.MAX\_VALUE}$  is 20. If the computed value exceeds `Long.MAX\_VALUE` an `ArithmeticException` is thrown.

#### Parameters:

n - argument

#### Returns:

n!

#### Throws:

`MathArithmeticException` - if the result is too large to be represented by a long.

`NotPositiveException` - if  $n < 0$ .

`MathArithmeticException` - if  $n > 20$ : The factorial value is too large to fit in a long.

- El archivo POM.xml se verá así:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.ejercicio.batalla</groupId>
  <artifactId>ejercicio_batalla</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
    <!--
https://mvnrepository.com/artifact/org.apache.commons/commons-math
3 -->
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-math3</artifactId>
      <version>3.6.1</version>
    </dependency>
  </dependencies>
</project>
```

- Generamos nuestra clase de personajes.

```
package com.batalla;
import org.apache.commons.math3.util.CombinatoricsUtils;
public class Personaje implements IActividades, IMovimiento {
    String nombre;
    double energia;
    double poder;
    Personaje(){}
    Personaje(String nombre, int poder, int energia){
        this.nombre = nombre;
        this.poder = CombinatoricsUtils.factorial(poder);
        this.energia = CombinatoricsUtils.factorial(energia);
    }
    public String getNombre () {
        return this.nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public double getEnergia () {
        return this.energia;
    }
    public void setEnergia (double vida) {
        this.energia = vida;
    }
}
```

```
}  
public double getPoder() {  
    return this.poder;  
}  
public void setPoder(double poder) {  
    this.poder = poder;  
}  
public void ataqueBasico() {  
    System.out.println(this.nombre+", ha pegado una patada!");  
}  
public void ataqueAvanzado() {  
    System.out.println(this.nombre+", ha lanzado un Kamehameha! de "+this.poder+" puntos de potencia");  
}  
public void defenderAtaque() {  
    System.out.println(this.nombre+", se ha defendido contra el ataque!");  
}  
public void esquivarAtaque() {  
    System.out.println(this.nombre+ ", lo ha esquivado!");  
}  
public String toString() {  
    return "Hola, soy "+this.nombre+", mi poder alcanza "+this.poder+" y mi energía no supera los  
"+this.energia;
```



```
    }  
    public void avanzar() {  
        System.out.println(this.nombre+", avanzó hacia adelante");  
    }  
    public void derecha() {  
        System.out.println(this.nombre+", giró a la derecha");  
    }  
    public void izquierda() {  
        System.out.println(this.nombre+", giró a la izquierda");  
    }  
    public void retroceder() {  
        System.out.println(this.nombre+", retrocedió");  
    }  
}
```

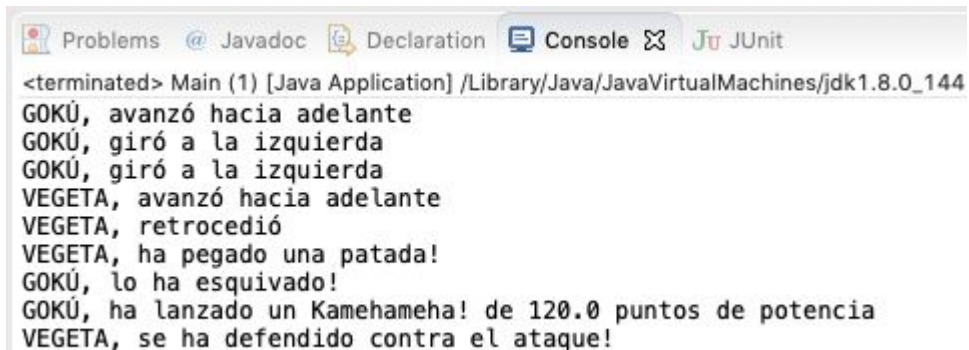
```
package com.batalla;
public class Torneo {
    Personaje p1;
    Personaje p2;
    Torneo(Personaje p1, Personaje p2){
        this.p1 = p1;
        this.p2 = p2;
    }
    public void pelea() {
        p1.avanzar();
        p1.izquierda();
        p1.izquierda();
        p2.avanzar();
        p2.retroceder();
        p2.ataqueBasico();
        p1.esquivarAtaque();
        p1.ataqueAvanzado();
        p2.defenderAtaque();
    }
    public void presentarContrincantes() {
        p1.toString();
        p2.toString();
    }
}
```

Crearemos  
nuestra clase  
del torneo

- Generamos nuestra clase Main.

```
package com.batalla;  
public class Main {  
    public static void main(String[] args) {  
        Personaje goku = new Personaje("GOKÚ",5,10);  
        Personaje vegeta = new Personaje("VEGETA",6,9);  
        Torneo torneo = new Torneo(goku,vegeta);  
        torneo.presentarContrincantes();  
        torneo.pelea();  
    }  
}
```

- Resultado del ejercicio.



The screenshot shows a Java IDE with a console window open. The console displays the output of the program, which simulates a battle between GOKÚ and VEGETA. The output is as follows:

```
<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_144  
GOKÚ, avanzó hacia adelante  
GOKÚ, giró a la izquierda  
GOKÚ, giró a la izquierda  
VEGETA, avanzó hacia adelante  
VEGETA, retrocedió  
VEGETA, ha pegado una patada!  
GOKÚ, lo ha esquivado!  
GOKÚ, ha lanzado un Kamehameha! de 120.0 puntos de potencia  
VEGETA, se ha defendido contra el ataque!
```



# Cierre

{desafío}  
latam\_



15 Minutos

**¿Existe algún concepto que  
no hayas comprendido?**

**Volvamos a revisar los conceptos que más te  
hayan costado antes de seguir adelante**

**Reflexionemos**





*Academia de  
talentos digitales*

[www.desafiolatam.com](http://www.desafiolatam.com)



/DesafioLatam



/DesafioLatam



/DesafioLatam



/DesafioLatam