

Propuesta de ejercicio

Propuesta de ejercicio	1
¿Qué aprenderás?	2
Introducción	2
Desarrollo	3
Conclusiones y observaciones	8



¡Comencemos!

¿Qué aprenderás?

- Crear un algoritmo desde el enfoque de inyección de dependencias.

Introducción

Implementaremos una aplicación que simula un Torneo de Artes Marciales, donde cada personaje se moverá en base a librerías con funciones, basándonos en la siguiente descripción, cada personaje, posee un ataque básico, un ataque avanzado, una protección contra hechizos, energía y poder.

Basándose en el concepto de Interfaces, crear dos interfaces, una para mover al personaje y otra para realizar las acciones.

Se debe crear un campo de combate donde podrán interactuar los personajes y ambos personajes deben realizar distintas tareas.

¡Vamos con todo!



Desarrollo

Comencemos creando 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();
}
```

Para la clase de personaje, utilizaremos la librería de [Maven Apache Commons Math](#) versión 3.6.1 para utilizar el método para obtener un valor factorial. Este, lo usaremos en nuestra clase personaje para calcular el poder y la energía en base a los valores entregados en el constructor.

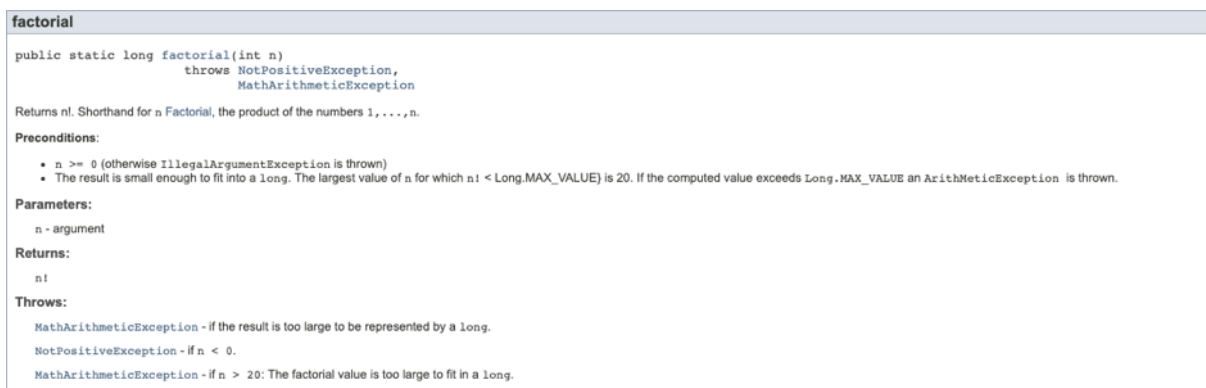


Imagen 1. Método factorial
Fuente: Desafío Latam.

El método factorial, recibe un parámetro integer retornando un valor de tipo de dato *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-math3 -->
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-math3</artifactId>
      <version>3.6.1</version>
    </dependency>
  </dependencies>
</project>
```

Generamos nuestra clase de personajes, que implementa ambas interfaces además de incorporar el llamado a la librería de Maven.

```
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");
}
}
```

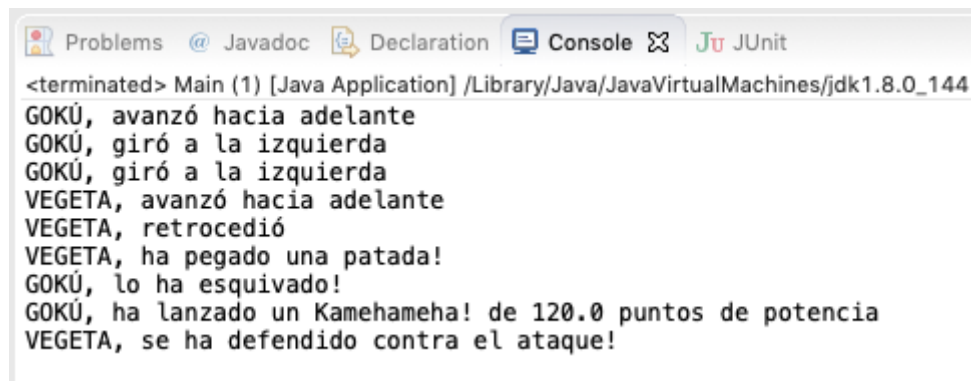
Ahora, crearemos nuestra clase del torneo, donde los personajes interactúan:

```
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();
    }
}
```

Finalmente, generamos nuestra clase *Main*, donde creamos nuestros personajes, creamos nuestro torneo, lo ejecutamos y ¡a la batalla!

```
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();
    }
}
```

Obtendremos como resultado, lo siguiente:



```
<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!
```

Imagen 2. Resultado del ejercicio.
Fuente: Desafío Latam.

Conclusiones y observaciones

Vimos que se puede hacer cualquier cosa con inyección de dependencias, incluso mezclando las disponibles de Maven con las que nosotros podemos generar, esto no nos muestra limitantes de dónde o cómo podemos implementar librerías o proyectos desde cualquier parte del código, ahora bien, es importante recordar que el código tendrá un flujo definido y siempre estará ordenado.