Texto

Descripción generada automáticamente

Escuela Colombiana de Ingeniería Julio Garavito

Laboratorio 2. Ciclos de vida y desarrollo de software

ISIS CVDS-201

**PATTERNS**

Docente

Oscar David Ospina Rodríguez

Estudiantes

Angie Julieth Ramos Cortes

Alexandra Moreno Latorre

30 Enero 2025-1

Bogotá

**TALLER 2**

**PATTERNS - FACTORY**

**PRE-RREQUISITOS**

* Java OpenJDK Runtime Environment: 17.x.x
* Apache Maven: 3.9.x

**OBJETIVOS**

1. Entender ¿qué es Maven?
2. Usar comandos de generación de arquetipos, compilación y ejecución de un proyecto usando Maven
3. Obtener puntos adicionales por PR qué corrijan o mejoren los laboratorios

**LA HERRAMIENTA MAVEN**

La herramienta [Apache Maven](https://maven.apache.org/what-is-maven.html) se usa para gestionar y manejar proyectos de software. La base de maven para un proyecto es el concepto de un modelo de objeto de proyecto (POM), Maven puede gestionar la compilación, los informes y la documentación de un proyecto a partir de este modelo, que se concreta en el archivo pom.xml.

Ingresar a la página de la herramienta y entender:

* Cuál es su mayor utilidad
* Fases de maven
* Ciclo de vida de la construcción
* Para qué sirven los plugins
* Qué es y para qué sirve el repositorio central de Maven

Apache Maven es una herramienta de gestión y comprensión de proyectos de software que se basa en el concepto de un Modelo de Objeto de Proyecto (POM). A partir de este modelo, Maven puede gestionar la construcción, los informes y la documentación de un proyecto.

[maven.apache.org](https://maven.apache.org/?utm_source=chatgpt.com)

**Utilidad principal:**

Maven se utiliza para gestionar la construcción, documentación, informes, dependencias, gestión de control de versiones, liberaciones y distribución de proyectos. Su objetivo es aplicar patrones a la infraestructura de construcción de un proyecto para promover la comprensión y la productividad, proporcionando una ruta clara en el uso de las mejores prácticas.

[maven.apache.org](https://maven.apache.org/guides/getting-started/?utm_source=chatgpt.com)

**Fases de Maven:**

Maven organiza el proceso de construcción en una serie de fases secuenciales, conocidas como el ciclo de vida de construcción. Algunas de las fases más comunes incluyen:

* **validate**: Verifica que el proyecto es correcto y que toda la información necesaria está disponible.
* **compile**: Compila el código fuente del proyecto.
* **test**: Ejecuta pruebas utilizando un marco de pruebas adecuado. Estas pruebas no requieren que el código sea empaquetado o desplegado.
* **package**: Toma el código compilado y lo empaqueta en su formato distribuible, como un JAR.
* **install**: Instala el paquete en el repositorio local, para usarlo como una dependencia en otros proyectos localmente.
* **deploy**: Copia el paquete final en el repositorio remoto para compartirlo con otros desarrolladores y proyectos.

Estas fases permiten a Maven gestionar de manera eficiente el ciclo de vida completo de la construcción de un proyecto.

[maven.apache.org](https://maven.apache.org/guides/index.html?utm_source=chatgpt.com)

**Ciclo de vida de la construcción:**

El ciclo de vida de construcción de Maven es una secuencia de fases que definen el proceso de construcción de un proyecto. Maven tiene tres ciclos de vida integrados:

1. **default**: Maneja la construcción del proyecto.
2. **clean**: Maneja la limpieza del proyecto.
3. **site**: Maneja la generación de la documentación del proyecto.

Cada ciclo de vida está compuesto por una serie de fases, y cada fase puede estar asociada a uno o más objetivos (goals) proporcionados por plugins. Por ejemplo, la fase compile en el ciclo de vida default está asociada al objetivo compile del plugin maven-compiler-plugin.

[maven.apache.org](https://maven.apache.org/guides/index.html?utm_source=chatgpt.com)

**Plugins:**

Los plugins en Maven son componentes que proporcionan una serie de objetivos que pueden ser ejecutados durante las diferentes fases del ciclo de vida de construcción. La mayoría de las funcionalidades de Maven son proporcionadas por plugins, incluyendo la compilación del código, la ejecución de pruebas, la generación de documentación y más. Los plugins permiten a Maven ser altamente flexible y extensible, ya que los usuarios pueden configurar y utilizar diferentes plugins según las necesidades específicas de su proyecto.

[maven.apache.org](https://maven.apache.org/guides/index.html?utm_source=chatgpt.com)

**Repositorio central de Maven:**

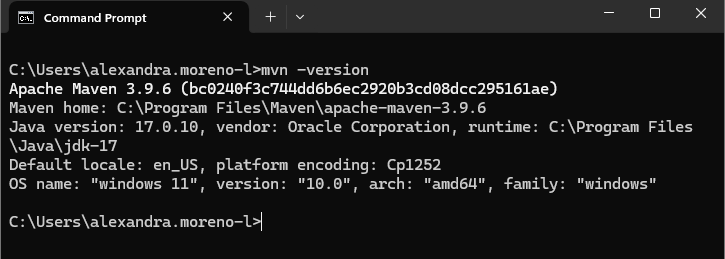
El repositorio central de Maven es una colección pública de artefactos de software, como bibliotecas y plugins, que Maven utiliza para resolver dependencias y obtener los recursos necesarios para la construcción de proyectos. Cuando un proyecto declara una dependencia en su POM, Maven busca el artefacto correspondiente en el repositorio central y lo descarga al repositorio local del usuario para su uso. Esto facilita la gestión de dependencias y asegura que los proyectos utilicen versiones consistentes de las bibliotecas necesarias.

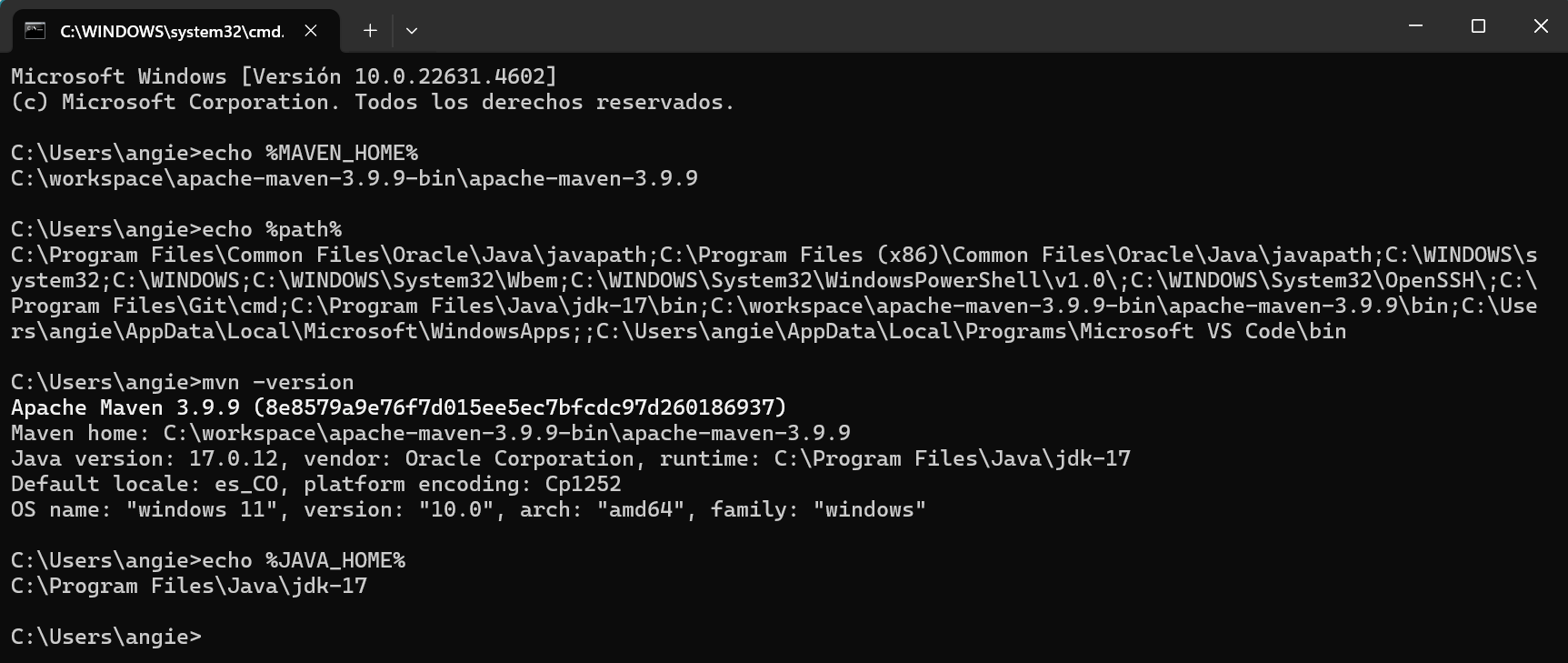
[central.sonatype.org](https://central.sonatype.org/publish/publish-maven/?utm_source=chatgpt.com)

**EJERCICIO DE LAS FIGURAS**

**CREAR UN PROYECTO CON MAVEN**

*Primero revisamos que Maven si se encuentre instalado*





Buscar cómo se crea un proyecto maven con ayuda de los arquetipos (archetypes).

Busque cómo ejecutar desde línea de comandos el objetivo "generate" del plugin "archetype", con los siguientes parámetros:

*ProjectId: org.apache.maven.archetypes:maven-archetype-quickstart:1.0*

*Id del Grupo: edu.eci.cvds*

*Id del Artefacto: Patterns*

*Paquete: edu.eci.cvds.patterns.archetype*

Se debió haber creado en el directorio, un nuevo proyecto Patterns a partir de un modelo o arquetipo, que crea un conjunto de directorios con un conjunto de archivos básicos.

Ejecutamos el siguiente comando en la terminal:

mvn archetype:generate -DgroupId=edu.eci.cvds -DartifactId=Patterns -Dpackage=edu.eci.cvds.patterns.archetype -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

Texto

Descripción generada automáticamente

Así hemos creado una estructura de proyecto Maven con el arquetipo quickstart.

Cambie al directorio Patterns:

*$ cd Patterns*



Para ver el conjunto de archivos y directorios creados por el comando mvn ejecute el comando tree.

*$ tree*

*Texto

Descripción generada automáticamente*

En caso de que no funcione en git bash, otra herramienta que se puede usar es PowerShell ya que ésta maneja el comando "tree".

En algunos sistemas operativos, este comando no funciona correctamente o puede requerir un parámetro (por ejemplo: tree /f). En caso que funcione, la salida muestra la estructura del proyecto, similar a como se muestra a continuación:

*.*

*│ pom.xml*

*└───src*

*├───main*

*│ └───java*

*│ └───edu*

*│ └───eci*

*│ └───cvds*

*│ └───patterns*

*└───archetype*

*│ App.java*

*│*

*└───test*

*└───java*

*└───edu*

*└───eci*

*└───cvds*

*└───patterns*

*└───archetype*

*AppTest.java*

**AJUSTAR ALGUNAS CONFIGURACIONES EN EL PROYECTO**

Edite el archivo pom.xml y realize la siguiente actualización:



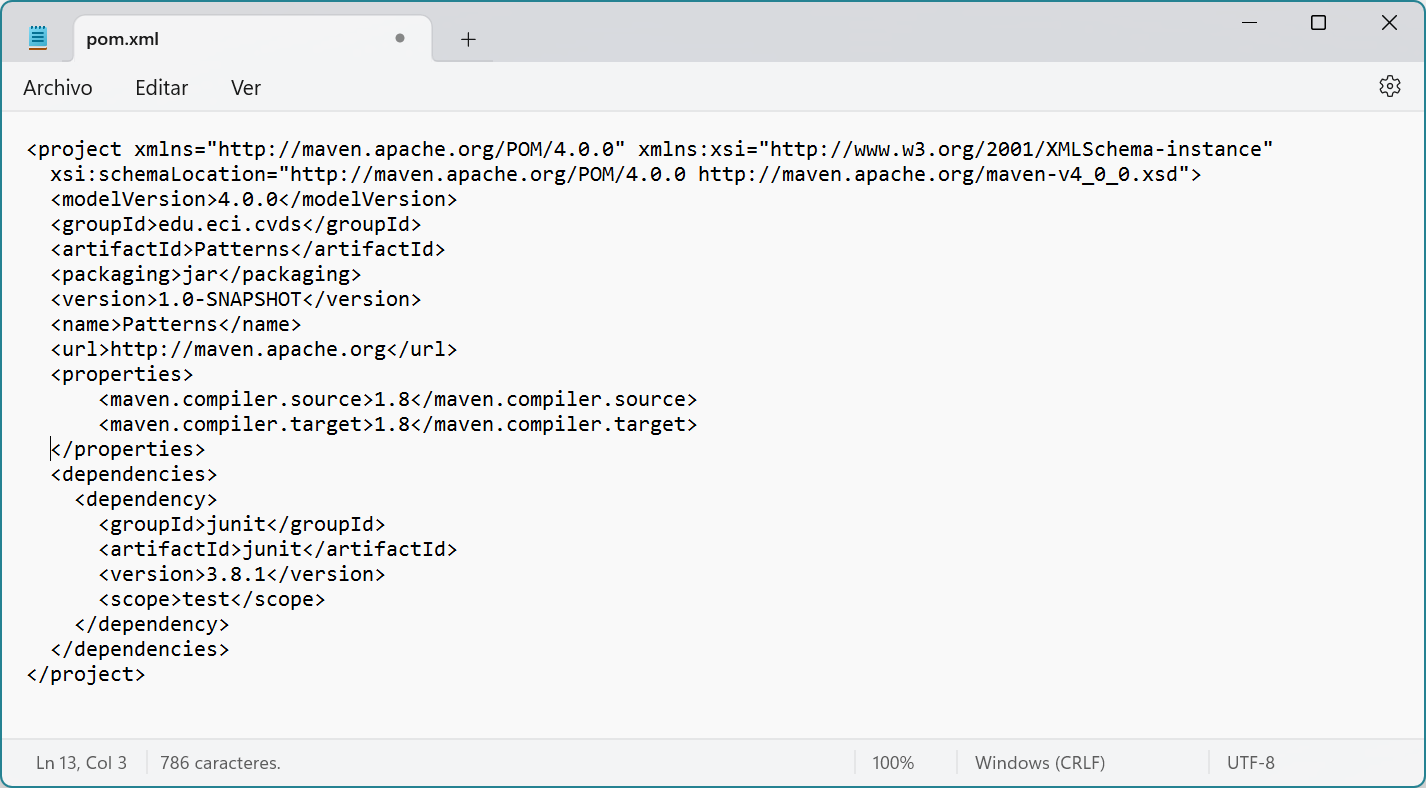
Hay que cambiar la version del compilador de Java a la versión 8, para ello, agregue la sección properties antes de la sección de dependencias:

<properties>

<maven.compiler.target>1.8</maven.compiler.target>

<maven.compiler.source>1.8</maven.compiler.source>

</properties>



Ejecutamos el siguiente comando en la terminal para asegurarnos de que el proyecto se compila con Java 8:

mvn compile

Con el siguiente mensaje confirmamos que todo salió bien:

[INFO] BUILD SUCCESS

Texto

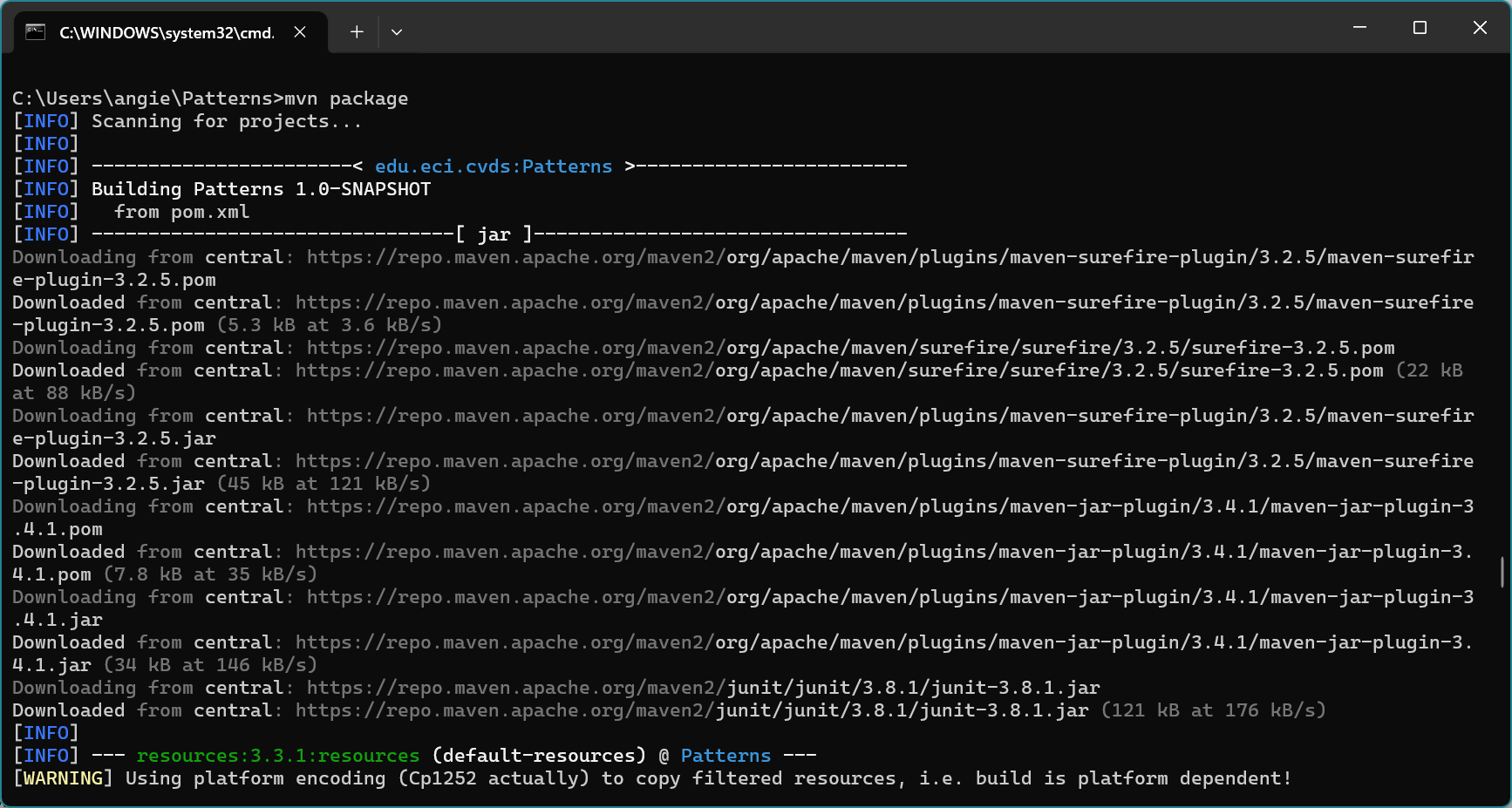
Descripción generada automáticamente

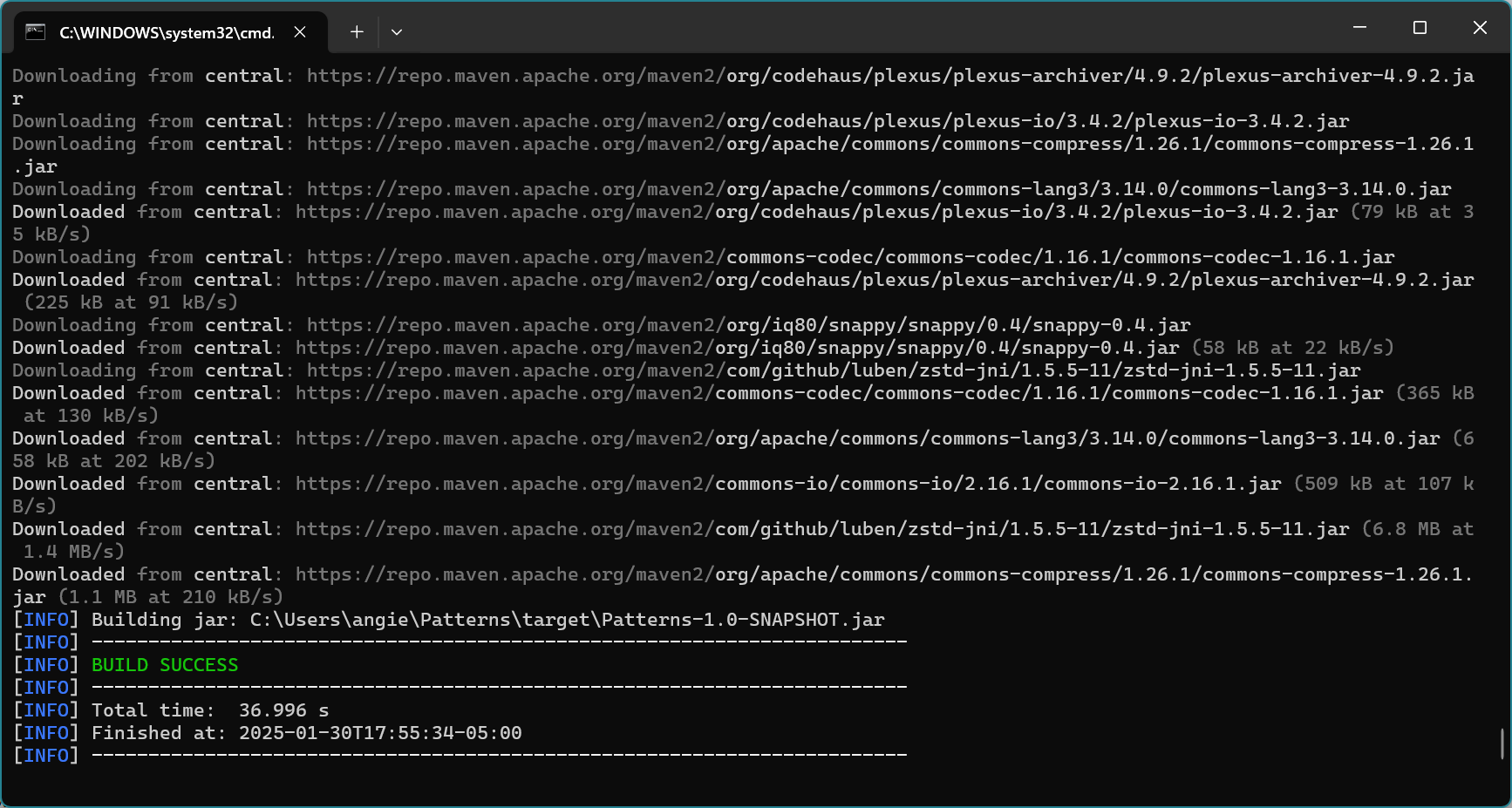
**COMPILAR Y EJECUTAR**

Para compilar ejecute el comando:

$ mvn package

El comando mvn package compila el proyecto y empaqueta los archivos compilados en un archivo JAR (Java ARchive), que se encuentra en el directorio target/.





Si maven no actualiza las dependencias utilice la opción -U así:

$ mvn -U package

Busque cuál es el objetivo del parámetro "package" y qué otros parámetros se podrían enviar al comando mvn.

**¿Qué hace el parámetro package?**

El objetivo package de Maven es una fase del ciclo de vida de Maven que compila el código fuente, ejecuta las pruebas y empaqueta el proyecto (por lo general, en un archivo JAR o WAR). Básicamente, es la etapa que prepara la aplicación para su distribución.

Busque cómo ejecutar desde línea de comandos, un proyecto maven y verifique la salida cuando se ejecuta con la clase App.java como parámetro en "mainClass". Tip: <https://www.mojohaus.org/exec-maven-plugin/usage.html>

Ahora, vamos a usar el **exec-maven-plugin** para ejecutar el proyecto desde la línea de comandos.

**Verificamos que el archivo pom.xml tenga configurado el plugin exec-maven-plugin.**

Abrimos el pom.xml y agregamos la siguiente configuración dentro de la sección <build>

Agregamos el siguiente código debajo de las dependencias:

<build>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>3.5.0</version>

<executions>

<execution>

<goals>

<goal>java</goal>

</goals>

</execution>

</executions>

<configuration>

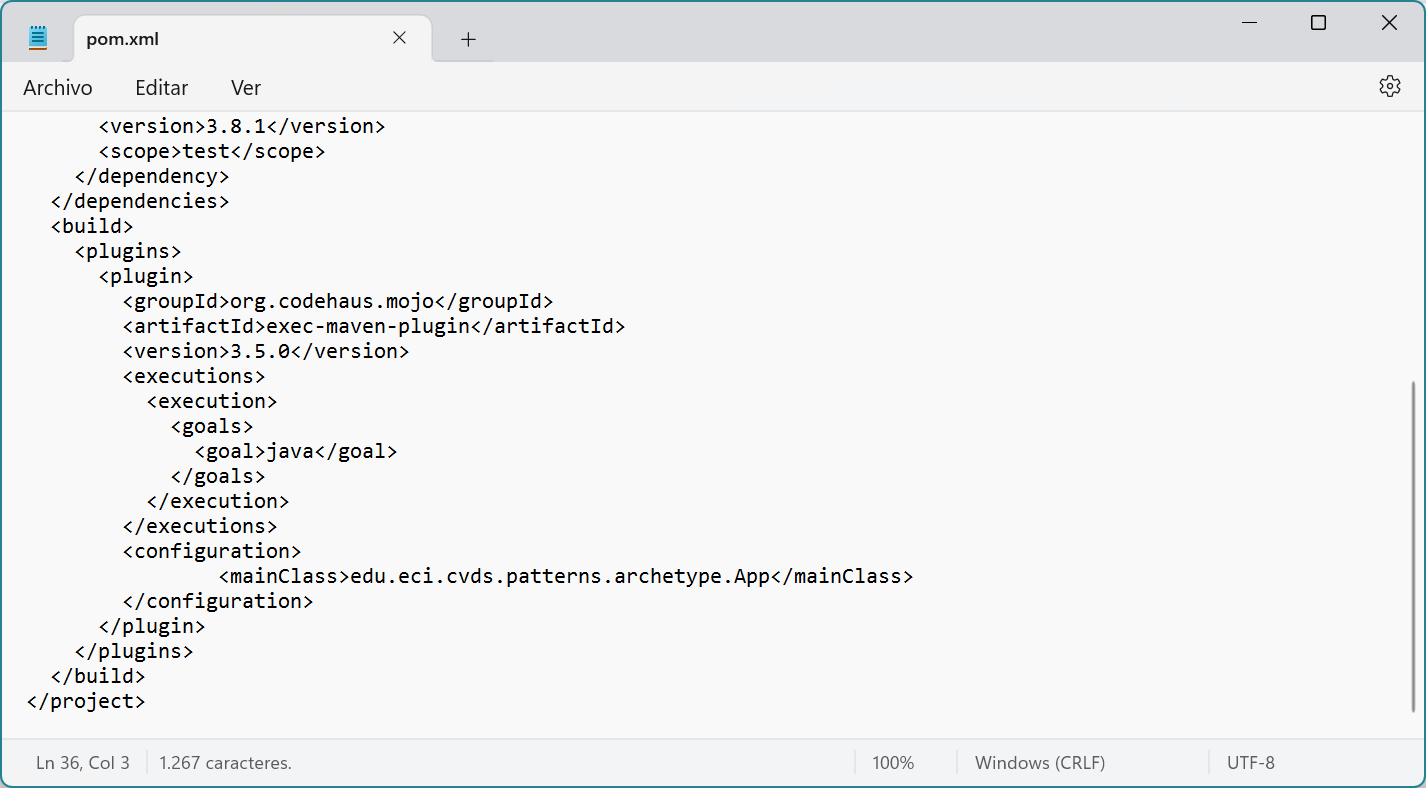
<mainClass>edu.eci.cvds.patterns.archetype.App</mainClass>

</configuration>

</plugin>

</plugins>

</build>



**Para ejecutar el proyecto con la clase *App* como parámetro en mainClass:** Ejecutamos el siguiente comando:

mvn exec:java

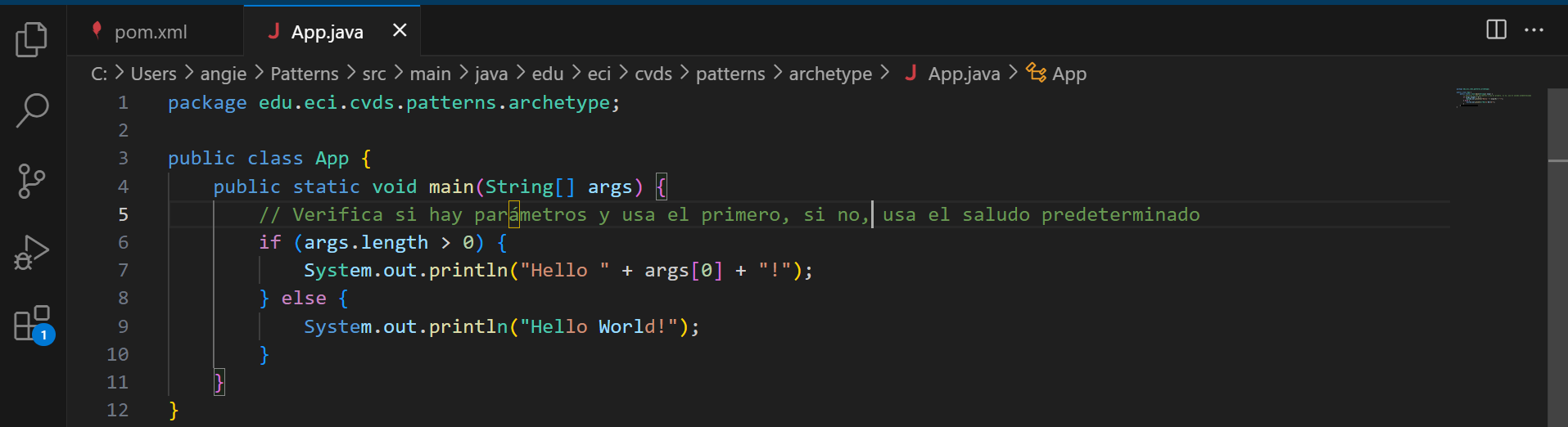
Imagen que contiene Tabla

Descripción generada automáticamente

Realice el cambio en la clase App.java para crear un saludo personalizado, basado en los parámetros de entrada a la aplicación.

Utilizar la primera posición del parámetro que llega al método "main" para realizar el saludo personalizado, en caso que no sea posible, se debe mantener el saludo como se encuentra actualmente

Abrimos la clase App.java y realizamos el siguiente cambio en el método main para que el saludo sea personalizado:



Ahora, el programa verificará si se pasa un parámetro y usará ese parámetro para saludar. Si no hay parámetro, mantiene el saludo predeterminado "Hello World!".

Buscar cómo enviar parámetros al plugin "exec".

Para enviar parámetros al plugin exec (exec-maven-plugin), lo hacemos con el siguiente comando:

mvn exec:java -Dexec.args="Julieth "

Texto

Descripción generada automáticamente

Ejecutar nuevamente la clase desde línea de comandos y verificar la salida: Hello World!

Ejecutamos con el comando: mvnexec:java

La salida es: Hello World!

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Ejecutar la clase desde línea de comandos enviando su nombre como parámetro y verificar la salida. Ej: Hello Pepito!

En este caso, el parámetro "Angie" se pasa al programa y la salida será: Hello Angie!

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ejecutar la clase con su nombre y apellido como parámetro. ¿Qué sucedió?

Sucede que sólo se envía el primer parámetro

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Verifique cómo enviar los parámetros de forma "compuesta" para que el saludo se realice con nombre y apellido.

Captura de pantalla de computadora

Descripción generada automáticamente

Ejecutar nuevamente y verificar la salida en consola. Ej: Hello Pepito Perez!



**HACER EL ESQUELETO DE LA APLICACIÓN**

Cree el paquete edu.eci.cvds.patterns.shapes y el paquete edu.eci.cvds.patterns.shapes.concrete.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente





Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente



Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

Cree una interfaz llamada Shape.java en el directorio src/main/java/edu/eci/cvds/patterns/shapes de la siguiente manera:

package edu.eci.cvds.patterns.shapes;

public interface Shape {

public int getNumberOfEdges();

}

Texto

Descripción generada automáticamente

Cree una enumeración llamada RegularShapeType.java en el directorio src/main/java/edu/eci/cvds/patterns/shapes así:

package edu.eci.cvds.patterns.shapes;

public enum RegularShapeType {

Triangle, Quadrilateral, Pentagon, Hexagon

}

Texto

Descripción generada automáticamente

Así debería de quedarnos:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

En el directorio src/main/java/edu/eci/cvds/patterns/shapes/concrete cree las diferentes clases (Triangle, Quadrilateral, Pentagon, Hexagon), que implementen la interfaz creada y retornen el número correspondiente de vértices que tiene la figura.

Siguiendo el ejemplo del triángulo:

package edu.eci.cvds.patterns.shapes.concrete;

import edu.eci.cvds.patterns.shapes.Shape;

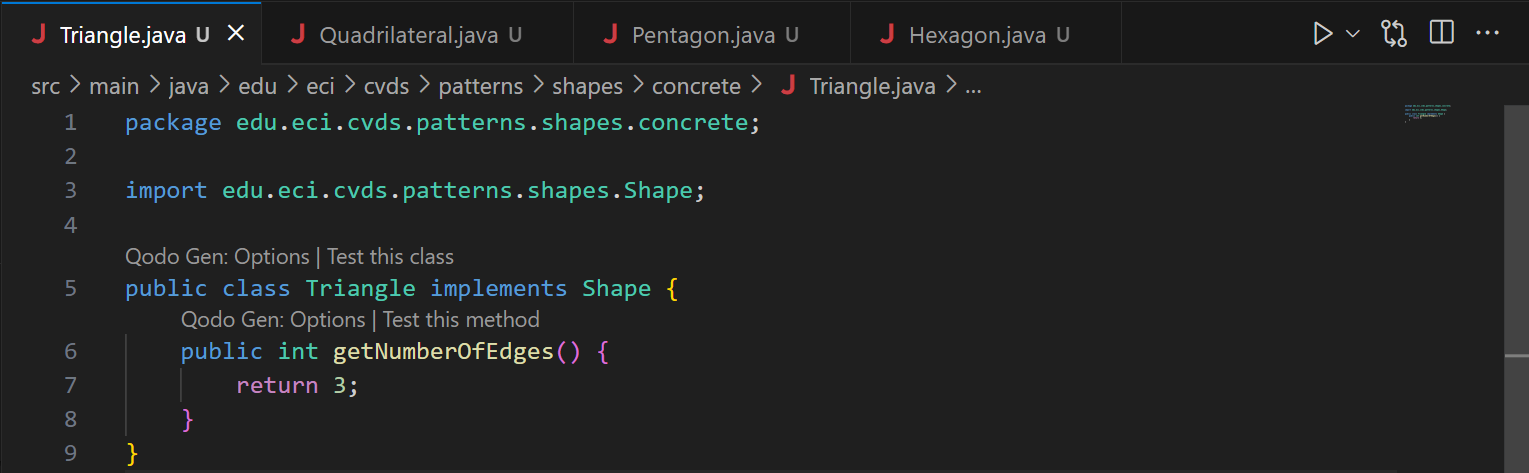
public class Triangle implements Shape {

public int getNumberOfEdges() {

return 3;

}

}



Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Cree el archivo ShapeMain.java en el directorio src/main/java/edu/eci/cvds/patterns/shapes con el metodo main:

package edu.eci.cvds.patterns.shapes;

public class ShapeMain {

public static void main(String[] args) {

if (args == null || args.length != 1) {

System.err.println("Parameter of type RegularShapeType is required.");

return;

}

try {

RegularShapeType type = RegularShapeType.valueOf(args[0]);

Shape shape = ShapeFactory.create(type);

System.out.println(

String.format(

"Successfully created a %s with %s sides.",

type,

shape.getNumberOfEdges()

)

);

} catch (IllegalArgumentException ex) {

System.err.println(

"Parameter '" + args[0] + "' is not a valid RegularShapeType"

);

return;

}

}

}

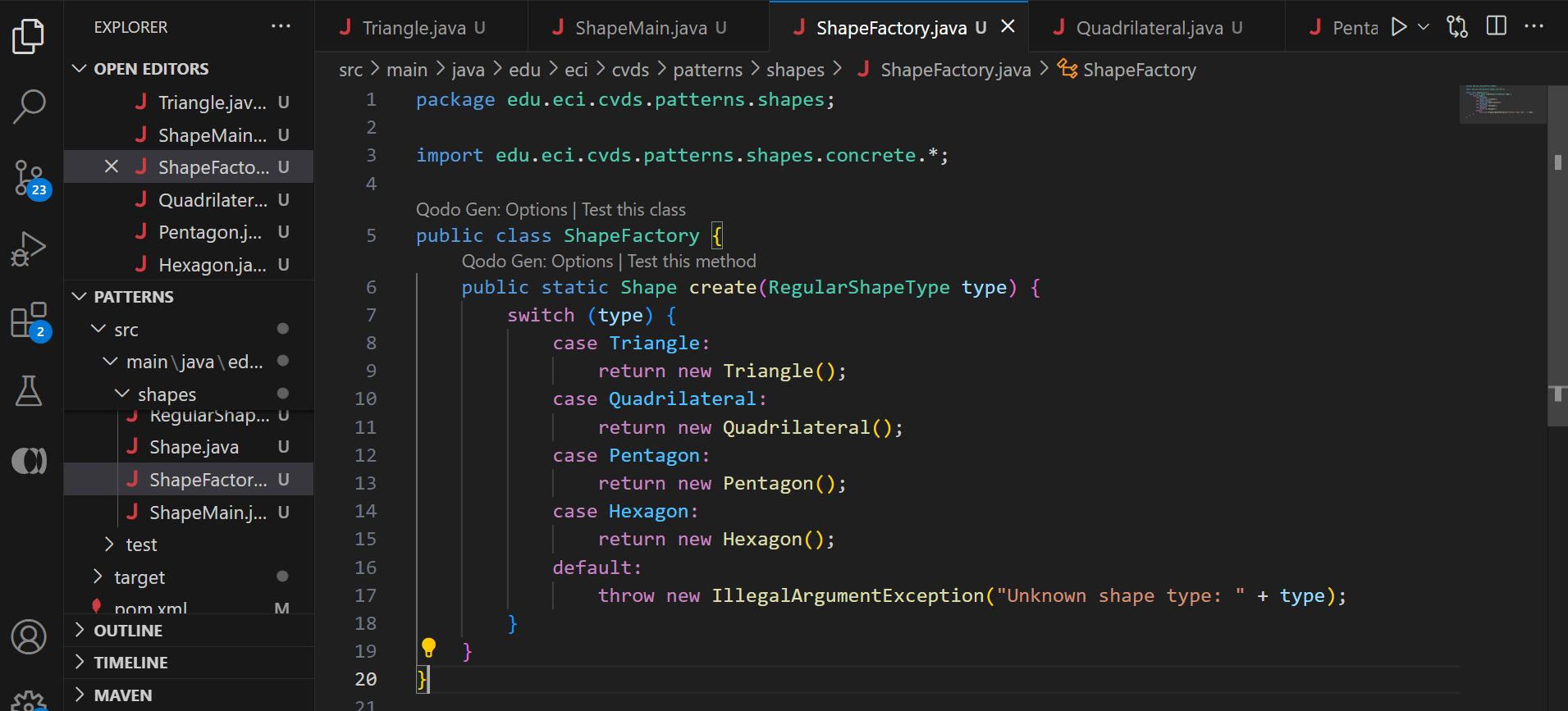
Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Analice y asegúrese de entender cada una de las instrucciones que se encuentran en todas las clases que se crearon anteriormente. Cree el archivo ShapeFactory.java en el directorio src/main/java/edu/eci/cvds/patterns/shapes implementando el patrón fábrica (Hint: <https://refactoring.guru/design-patterns/catalog>), haciendo uso de la instrucción switch-case de Java y usando las enumeraciones.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente



¿Cuál fábrica hiciste? y ¿Cuál es mejor?

* Simple Factory:
* Factory Method:
* Abstract Factory:

**¿Qué tipo de fábrica hicimos?**

La implementación de ShapeFactory sigue el **patrón Simple Factory**.

**Simple Factory:**

* Usa un solo método create(RegularShapeType type) para instanciar objetos según el tipo solicitado.
* Centraliza la creación de objetos en una sola clase.
* Es una solución sencilla y fácil de entender.

**¿Cuál es mejor?**

✅ **Depende del contexto**

Tabla

Descripción generada automáticamente

Ejecute múltiples veces la clase ShapeMain, usando el plugin exec de maven con los siguientes parámetros y verifique la salida en consola para cada una:

* Sin parámetros

mvn exec:java -Dexec.mainClass="edu.eci.cvds.patterns.shapes.ShapeMain"

Texto

Descripción generada automáticamente

* Parámetro: qwerty

mvn exec:java -Dexec.mainClass="edu.eci.cvds.patterns.shapes.ShapeMain" -Dexec.args="qwerty"

Texto

Descripción generada automáticamente con confianza media

* Parámetro: pentagon

mvn exec:java -Dexec.mainClass="edu.eci.cvds.patterns.shapes.ShapeMain" -Dexec.args="pentagon"

Texto

Descripción generada automáticamente

* Parámetro: Hexagon

mvn exec:java -Dexec.mainClass="edu.eci.cvds.patterns.shapes.ShapeMain" -Dexec.args="Hexagon"

Texto

Descripción generada automáticamente

¿Cuál(es) de las anteriores instrucciones se ejecutan y funcionan correctamente y por qué?

* 1. Error: El código verifica if (args == null || args.length != 1), y como no hay argumentos, muestra un error.
  2. Error: qwerty no está en RegularShapeType, por lo que RegularShapeType.valueOf(args[0]) lanza una excepción IllegalArgumentException.
  3. Error: RegularShapeType.valueOf(args[0]) es sensible a mayúsculas y pentagon no coincide con Pentagon, lanzando una excepción.
  4. Exitosa: Hexagon existe en RegularShapeType, por lo que el ShapeFactory.create(type) devuelve un objeto Hexagon correctamente.

**ENTREGAR**

* Se espera al menos que durante la sesión de laboratorio, se termine el ejercicio del saludo y haya un avance significativo del ejercicio de las figuras. Dentro del directorio del proyecto, cree un archivo de texto integrantes.txt con el nombre de los dos integrantes del taller.
* Crear un repositorio para este proyecto y agregar la url del mismo, como entrega del laboratorio.
* La entrega final se realizará en Moodle.
* NOTA: Investigue para qué sirve "gitignore" y configurelo en su proyecto para evitar adjuntar archivos que no son relevantes para el proyecto.

El archivo .gitignore es un archivo de texto que le dice a Git qué otros archivos y carpetas de un proyecto debe ignorar.

Un archivo .gitignore local se coloca normalmente en el directorio de origen de un proyecto. También puedes crear un archivo .gitignore global y cualquier entrada que contenga será ignorada en todos tus repositorios Git por igual.

Para crear un archivo .gitignore local, crea un archivo de texto y llámalo .gitignore (recuerda incluir el . al principio). Después edita el archivo como sea necesario. Cada línea nueva debería contener el archivo o carpeta que quieres que Git ignore.

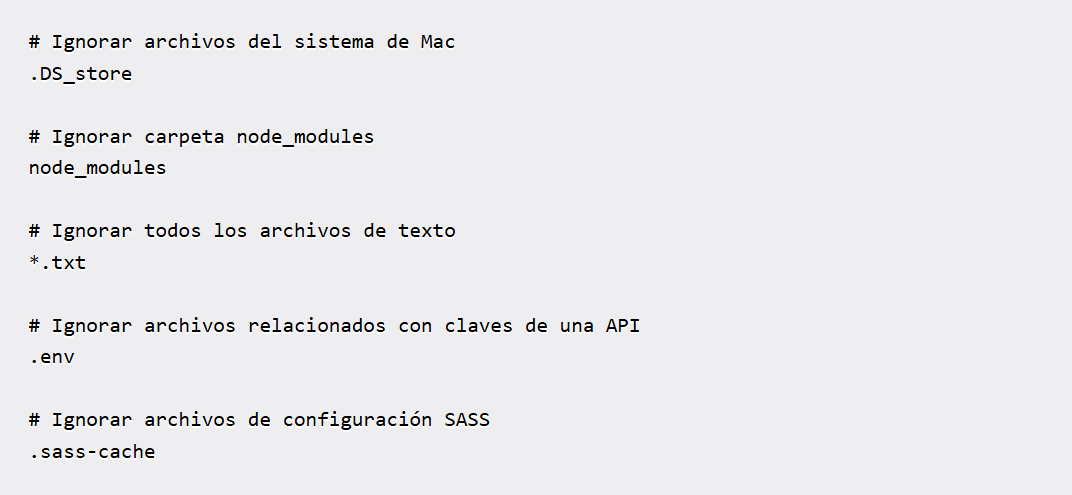
Las entradas en este archivo también pueden seguir un modelo similar.

\* se usa para encontrar coincidencias

/ se usa para ignorar nombres de ruta relacionados con el archivo .gitignore

# se usa para añadir comentarios al archivo .gitignore

Este es un ejemplo de cómo se vería un archivo .gitignore:



El .gitignore que hemos creado está configurado para:

* Ignorar la carpeta target/ y otros archivos generados por Maven
* Ignorar archivos específicos de IDEs como IntelliJ IDEA o Eclipse
* Ignorar archivos compilados .class
* Ignorar logs y archivos comprimidos
* Ignorar archivos específicos del sistema operativo