

# Escuela Colombiana de Ingeniería Julio Garavito

**Carrera:** INGENIERIA DE SISTEMAS.

**Materia:** CVDS

**Título:** LABORATORIO 2 - PATTERNS

**Estudiante:** SANTIAGO ARTEAGA GUTIERREZ & CRISTIAN DAVID POLO GARRIDO



ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO  
DEPARTAMENTO DE CIENCIAS NATURALES  
CICLOS DE VIDA DE DESARROLLO DE SOFTWARE  
GRUPO 02  
Bogotá, Colombia  
2025 – 1

## LABORATORIO 2 – PATTERNS – FACTORY

### 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](#) 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

```
C:\Users\Crisp>mvn -version
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: C:\Program Files\Apache\Maven
Java version: 1.8.0_441, vendor: Oracle Corporation, runtime: C:\Program Files (x86)\Java\jre1.8.0_441
Default locale: en_GB, platform encoding: Cp1252
OS name: "windows 11", version: "10.0", arch: "x86", family: "windows"
C:\Users\Crisp>
```

Hemos instalado Apache Maven 3.9.9 en nuestra máquina.

## EJERCICIO DE LAS FIGURAS

### CREAR UN PROYECTO CON MAVEN

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.

Para ejecutar el objetivo generate con los parámetros, debemos ejecutar este comando:

```
mvn archetype:generate -DgroupId=edu.eci.cvds -DartifactId=Patterns -
-Dpackage=edu.eci.cvds.patterns.archetype
-DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactId=maven-archetype-
quickstart -DarchetypeVersion=1.0 -DinteractiveMode=false
```

```
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.0
/maven-archetype-quickstart-1.0.jar (4.3 kB at 287 kB/s)
[INFO] -----
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO] -----
[INFO] Parameter: basedir, Value: C:\Users\Crisp\OneDrive\Campus\2025-1\Cvds\Lab02
[INFO] Parameter: package, Value: edu.eci.cvds.patterns.archetype
[INFO] Parameter: groupId, Value: edu.eci.cvds
[INFO] Parameter: artifactId, Value: Patterns
[INFO] Parameter: packageName, Value: edu.eci.cvds.patterns.archetype
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Users\Crisp\OneDrive\Campus\2025-1\Cvds\Lab02\Patterns
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 18.695 s
[INFO] Finished at: 2025-01-30T14:55:36-05:00
[INFO] -----
```

Finalmente, podremos ver la construcción completa de nuestro proyecto Maven.

Cambie al directorio **Patterns**:

```
$ cd Patterns
```

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

```
$ tree
```

```
C:\Users\CRISP\OneDrive\Campus\2025-1\CVDS\Lab02>cd Patterns
C:\Users\CRISP\OneDrive\Campus\2025-1\CVDS\Lab02\Patterns>tree /f
Folder PATH listing for volume Local Disk
Volume serial number is 5C3E-FC50
C:.
    pom.xml
    +-- src
        +-- main
            +-- java
                +-- edu
                    +-- eci
                        +-- cvds
                            +-- patterns
                                +-- archetype
                                    App.java
        +-- test
            +-- java
                +-- edu
                    +-- eci
                        +-- cvds
                            +-- patterns
                                +-- archetype
                                    AppTest.java
```

Así nos ha quedado el árbol de archivos y carpetas posterior a la generación y creación del proyecto.

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

```
.
+-- 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 realice la siguiente actualización:

Hay que cambiar la versión 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>
```

```

<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>edu.eci.cvds</groupId>
  <artifactId>Patterns</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Patterns</name>
  <url>http://maven.apache.org</url>
  <properties>
    <maven.compiler.target>1.8</maven.compiler.target>
    <maven.compiler.source>1.8</maven.compiler.source>
  </properties>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>

```

Hemos agregado la sección de *properties* que nos permite cambiar la versión del compilador de Java para nuestros proyectos.

## COMPILAR Y EJECUTAR

Para compilar ejecute el comando:

```
$ mvn package
```

```

PS C:\Users\Crisp\OneDrive\Campus\2025-1\CVds\Lab02\Patterns> mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Patterns 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO]   [ jar ]
[INFO] Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-resources-plugin/3.3.1/maven-resources-plugin-3.3.1.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-resources-plugin/3.3.1/maven-resources-plugin-3.3.1.pom (6.7 kB at 34 kB/s)
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:\Users\Criss\OneDrive\Campus\2025-1\CVds\Lab02\Patterns\src\main\resources
[INFO]
[INFO] --- compiler:3.13.0:compile (default-compile) @ Patterns ---
[INFO] Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-incremental/1.1/maven-shared-incremental-1.1.pom

```

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

```
$ mvn -U package
```

```

PS C:\Users\Criss\OneDrive\Campus\2025-1\CVds\Lab02\Patterns> mvn -U package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Patterns 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO]   [ jar ]
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ Patterns ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:\Users\Criss\OneDrive\Campus\2025-1\CVds\Lab02\Patterns\src\main\resources
[INFO]
[INFO] --- compiler:3.13.0:compile (default-compile) @ Patterns ---

```

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

El comando **mvn package** en Apache Maven ejecuta la fase package del ciclo de vida de construcción. Durante esta fase, Maven toma el código compilado y lo empaqueta en un formato distribuible, como un archivo JAR o WAR, listo para su distribución o implementación. Además de la fase package, Maven define varias fases en su ciclo de vida por defecto (default), cada una con un propósito específico:

- **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 unitarias utilizando frameworks como JUnit.
- **package**: Empaquea el código compilado en un formato específico, como JAR o WAR.

- **verify**: Realiza verificaciones para asegurar que el paquete cumple con los criterios de calidad.
- **install**: Instala el paquete en el repositorio local para que pueda ser utilizado como dependencia en otros proyectos.
- **deploy**: Copia el paquete final a un repositorio remoto para compartirlo con otros desarrolladores.

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>

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:

Buscar cómo enviar parámetros al plugin "exec".

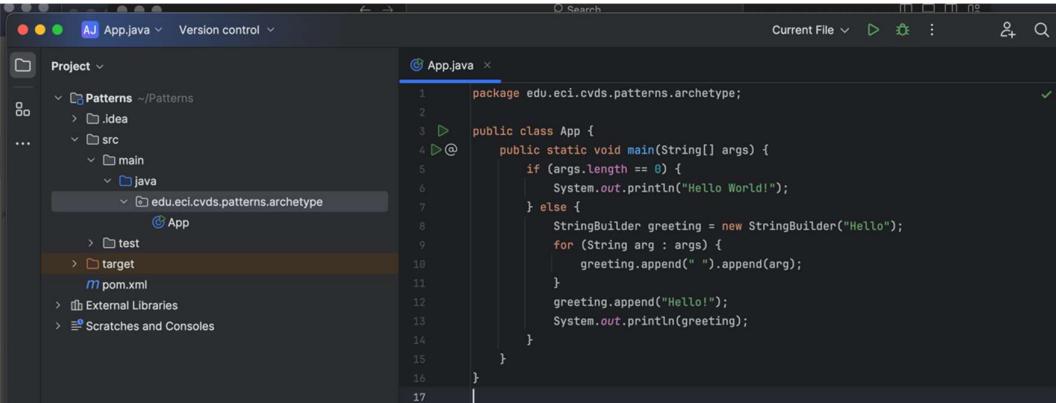
Ejecutar nuevamente la clase desde línea de comandos y verificar la salida: Hello World!

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

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

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

Ejecutar nuevamente y verificar la salida en consola. Ej: Hello Santiago Arteaga!



```

package edu.eci.cvds.patterns.archetype;
public class App {
    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Hello World!");
        } else {
            StringBuilder greeting = new StringBuilder("Hello");
            for (String arg : args) {
                greeting.append(" ").append(arg);
            }
            greeting.append("!");
            System.out.println(greeting);
        }
    }
}

```

```

[INFO] Scanning for projects...
[INFO]
[INFO] -----< edu.eci.cvds:Patterns >-----
[INFO] Building Patterns 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- exec:3.0.0:java (default-cli) @ Patterns ---
Hello Santiago Arteaga!
[INFO]
[INFO] BUILD SUCCESS
-----
```

```

Buscar cómo enviar parámetros al plugin "exec".
(base) santiagoarteaga@Santiagos-MacBook-Pro Patterns % mvn exec:java

[INFO] Scanning for projects...
[INFO]
[INFO] -----< edu.eci.cvds:Patterns >-----
[INFO] Building Patterns 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- exec:3.0.0:java (default-cli) @ Patterns ---
Hello World!
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time:  3.160 s

```

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

```

(base) santiagoarteaga@Santiagos-MacBook-Pro Patterns % mvn exec:java

[INFO] Scanning for projects...
[INFO]
[INFO] -----< edu.eci.cvds:Patterns >-----
[INFO] Building Patterns 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- exec:3.0.0:java (default-cli) @ Patterns ---
Hello World!
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time:  3.160 s

```

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

```

(base) santiagoarteaga@Santiagos-MacBook-Pro Patterns % mvn exec:java -Dexec.args="Pepito"

[INFO] Scanning for projects...
[INFO]
[INFO] -----< edu.eci.cvds:Patterns >-----
[INFO] Building Patterns 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- exec:3.0.0:java (default-cli) @ Patterns ---
Hello Pepito!
[INFO]
[INFO] BUILD SUCCESS

```

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

- **Por defecto, solo toma el primer argumento (Pepito). La salida será:**

```

[INFO] Scanning for projects...
[INFO]
[INFO] < edu.eci.cvds:Patterns >
[INFO] Building Patterns 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- exec:3.0.0:java (default-cli) @ Patterns ---
Hello Santiago!
[INFO]
[INFO] BUILD SUCCESS

```

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

- Para que el saludo funcione con **nombre y apellido**, debemos modificar la clase App para combinar todos los argumentos:

```

App.java x
1 package edu.eci.cvds.patterns.archetype;
2
3 public class App {
4     @
5         public static void main(String[] args) {
6             if (args.length == 0) {
7                 System.out.println("Hello World!");
8             } else {
9                 StringBuilder greeting = new StringBuilder("Hello");
10                for (String arg : args) {
11                    greeting.append(" ").append(arg);
12                }
13                greeting.append("!");
14                System.out.println(greeting);
15            }
16        }
17    }

```

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

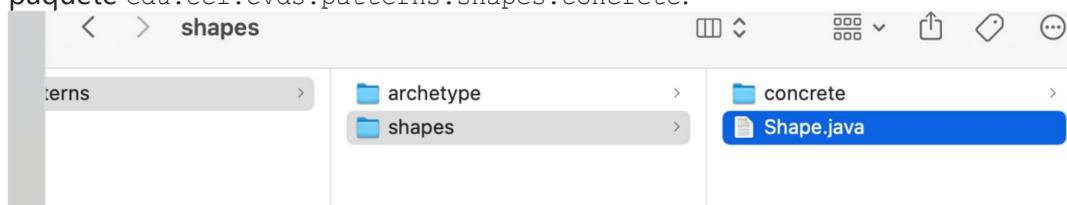
```

[INFO] -----
(base) santiagoarteaga@Santiagos-MacBook-Pro Patterns % mvn exec:java -Dexec.args="Santiago Arteaga"
Hello Santiago Arteaga!
[INFO] -----

```

## 1.1 HACER EL ESQUELETO DE LA APLICACIÓN

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



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();
}
```

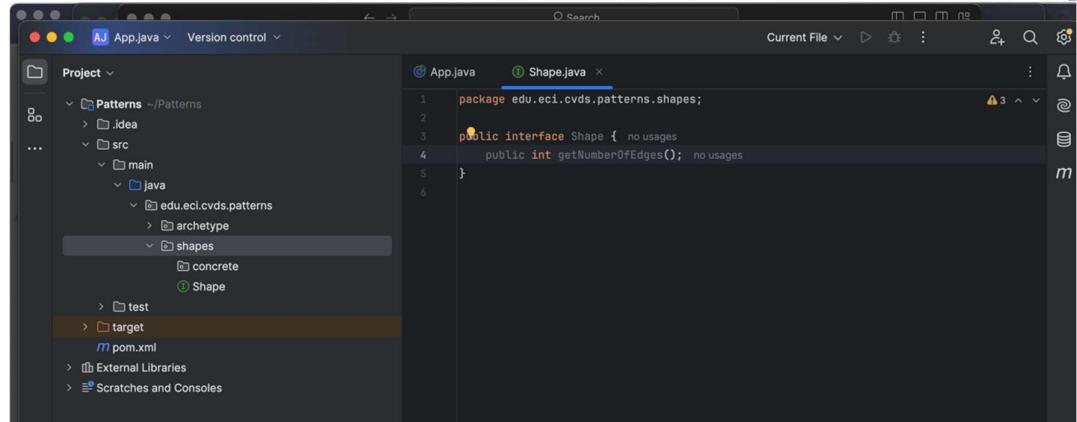
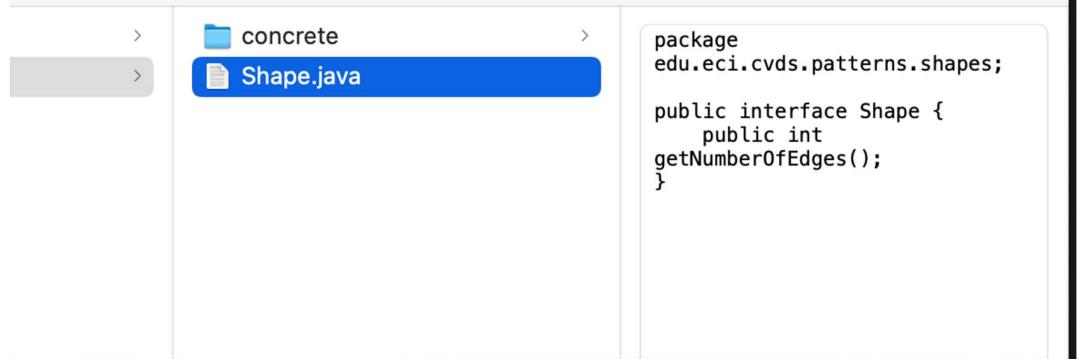
```
[INFO] Finished at: 2025-02-06T08:29:06-05:00
[INFO] -----
(base) santiagoarteaga@Santiagos-MacBook-Pro Patterns % cd src/main/java

(base) santiagoarteaga@Santiagos-MacBook-Pro java % mkdir -p edu/eci/cvds/patterns/shapes
mkdir -p edu/eci/cvds/patterns/shapes/concrete

(base) santiagoarteaga@Santiagos-MacBook-Pro java % cd edu/eci/cvds/patterns/shapes

(base) santiagoarteaga@Santiagos-MacBook-Pro shapes % touch Shape.java

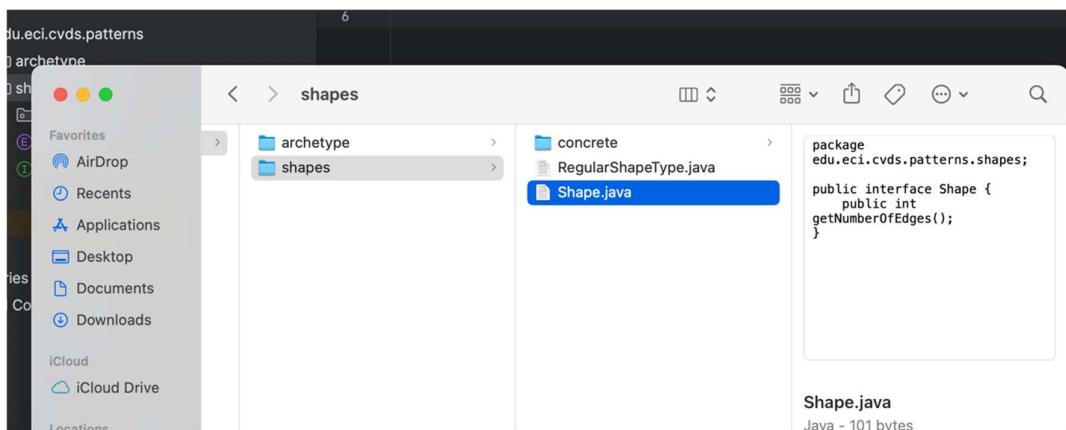
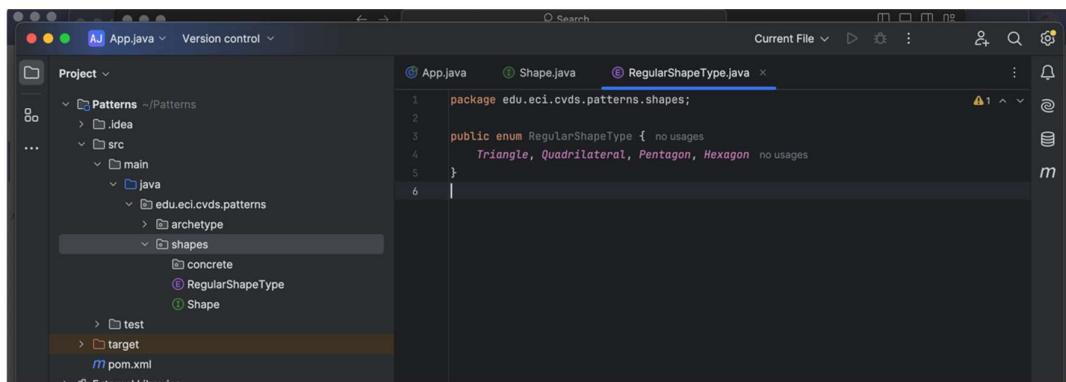
(base) santiagoarteaga@Santiagos-MacBook-Pro shapes %
```



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
}
```

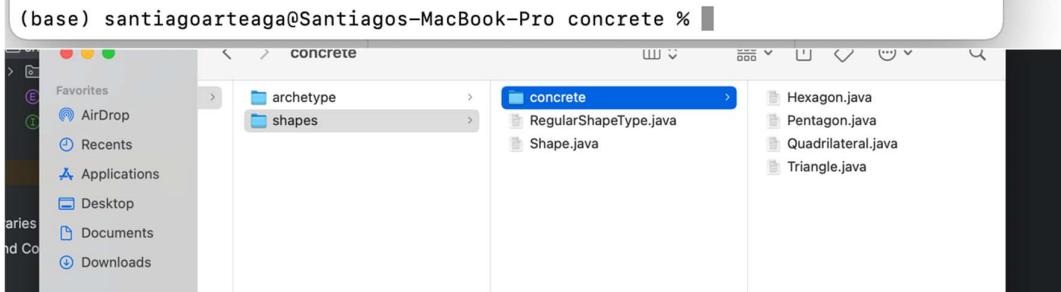


```
0 directories, 0 files
(base) santiagoarteaga@Santiagos-MacBook-Pro shapes % tree
.
└── RegularShapeType.java
└── Shape.java
└── concrete

2 directories, 2 files
(base) santiagoarteaga@Santiagos-MacBook-Pro shapes %
```

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.

```
(base) santiagoarteaga@Santiagos-MacBook-Pro main % cd ..
(base) santiagoarteaga@Santiagos-MacBook-Pro src % cd ..
(base) santiagoarteaga@Santiagos-MacBook-Pro Patterns % cd src/main/java/edu/eci
/cvds/patterns/shapes/concrete
touch Triangle.java Quadrilateral.java Pentagon.java Hexagon.java
```



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;
    }
}
```

The image displays four separate Java code editors, each showing a different concrete class implementation of the `Shape` interface. All code is contained within the package `edu.eci.cvds.patterns.shapes.concrete`.

- Hexagon.java:** Implements the `Shape` interface with 6 edges.

```
package edu.eci.cvds.patterns.shapes.concrete;  
import edu.eci.cvds.patterns.shapes.Shape;  
  
public class Hexagon implements Shape { no usages  
    @Override no usages  
    public int getNumberOfEdges() {  
        return 6;  
    }  
}
```

- Pentagon.java:** Implements the `Shape` interface with 5 edges.

```
package edu.eci.cvds.patterns.shapes.concrete;  
import edu.eci.cvds.patterns.shapes.Shape;  
  
public class Pentagon implements Shape { 1 usage  
    @Override 1 usage  
    public int getNumberOfEdges() {  
        return 5;  
    }  
}
```

- Quadrilateral.java:** Implements the `Shape` interface with 4 edges.

```
package edu.eci.cvds.patterns.shapes.concrete;  
import edu.eci.cvds.patterns.shapes.Shape;  
  
public class Quadrilateral implements Shape { no usages  
    @Override no usages  
    public int getNumberOfEdges() {  
        return 4;  
    }  
}
```

- Triangle.java:** Implements the `Shape` interface with 3 edges.

```
package edu.eci.cvds.patterns.shapes.concrete;  
import edu.eci.cvds.patterns.shapes.Shape;  
  
public class Triangle implements Shape { no usages  
    @Override no usages  
    public int getNumberOfEdges() {  
        return 3;  
    }  
}
```

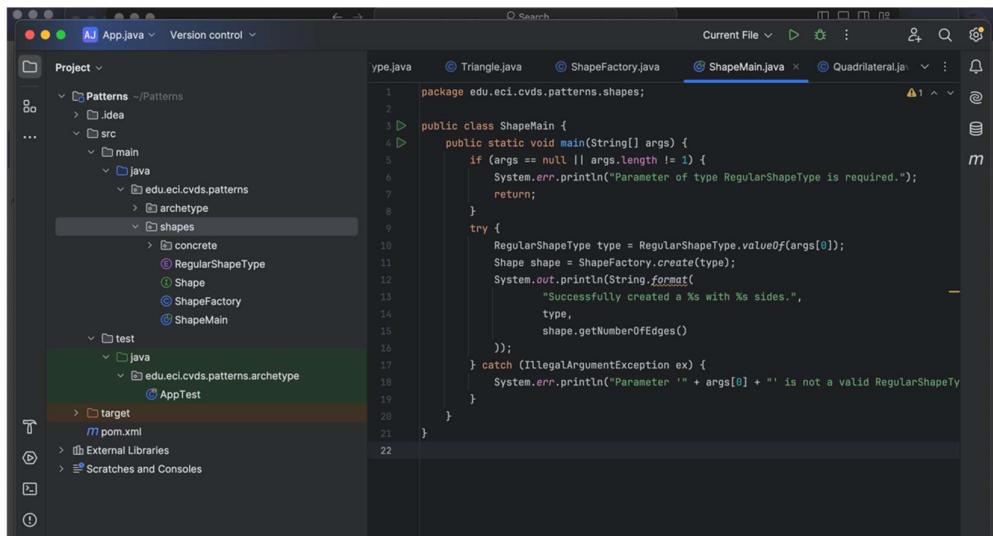
The project structure on the left shows the `Patterns` package containing `ShapeMain.java`, `ShapeFactory.java`, and `Shape`. The `Shape` class is further divided into `Abstract` and `Concrete` sub-directories, which correspond to the `ShapeMain`, `ShapeFactory`, and the four concrete shape classes (`Hexagon`, `Pentagon`, `Quadrilateral`, `Triangle`) respectively. The `test` directory contains `AppTest`.

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 {

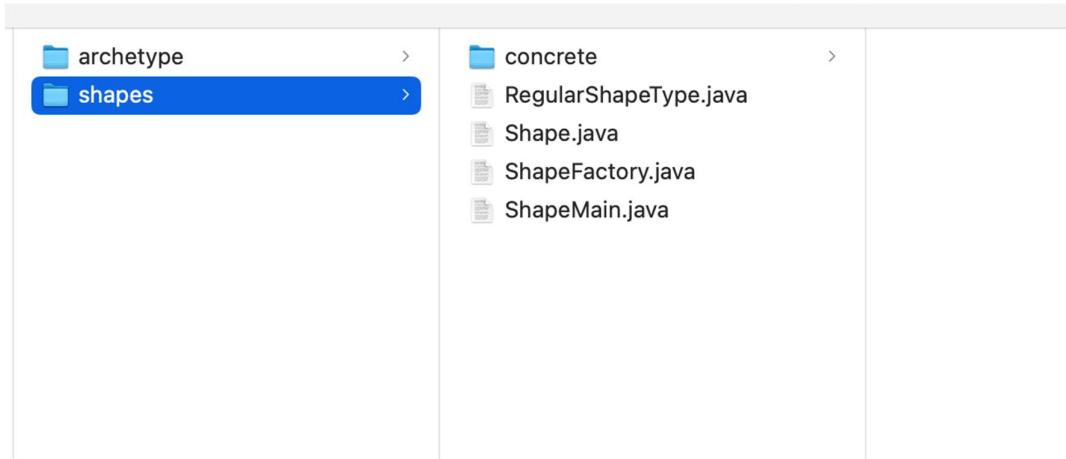
    public static void main(String[] args) {
        if (args == null || args.length != 1) {
            System.err.println("Parameter of type RegularShapeType is required.");
            return;
        }
        try {
            RegularShapeType type = RegularShapeType.valueOf(args[0]);
            Shape 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;
        }
    }
}
```



```
(base) santiagoarteaga@Santiagos-MacBook-Pro patterns % pwd
/Users/santiagoarteaga/Patterns/src/main/java/edu/eci/cvds/patterns
(base) santiagoarteaga@Santiagos-MacBook-Pro patterns % cd shapes
(base) santiagoarteaga@Santiagos-MacBook-Pro shapes % pwd
/Users/santiagoarteaga/Patterns/src/main/java/edu/eci/cvds/patterns/shapes
(base) santiagoarteaga@Santiagos-MacBook-Pro shapes % touch ShapeMain.java

(base) santiagoarteaga@Santiagos-MacBook-Pro shapes %
```

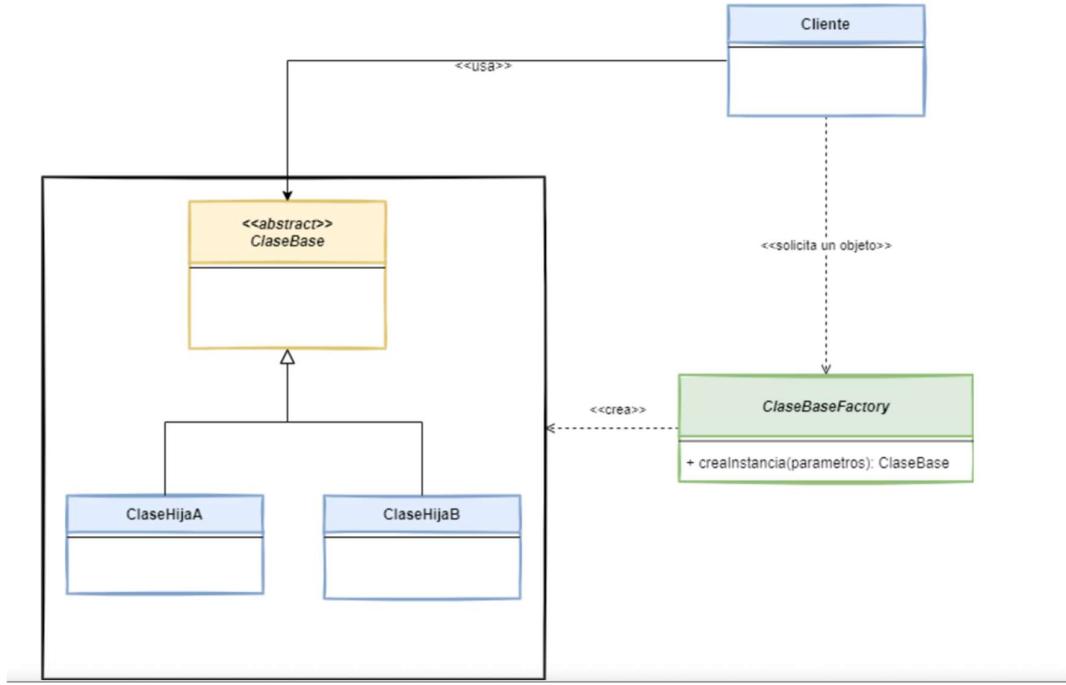
Analice y asegúrese de entender cada una de las instrucciones que se encuentran en todas las clases que se crearon anteriormente. Cree el archivo `ShapeFactory.java` en el directorio `src/main/java/edu/eci/cvds/patterns/shapes` implementando el patrón fábrica (Hint: <https://refactoring.guru/design-patterns/catalog>),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



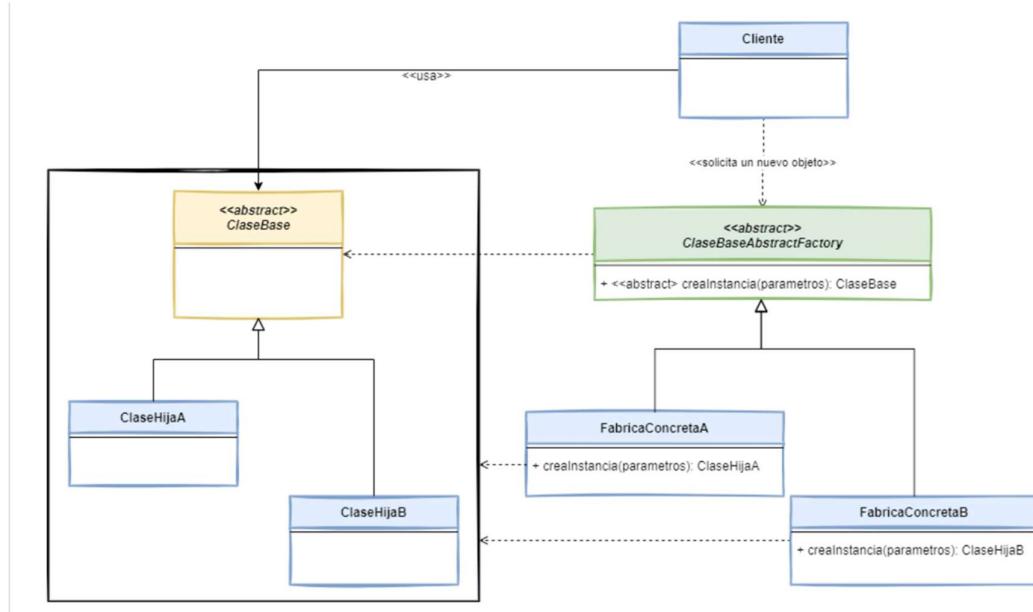
```
/Users/santiagoarteaga/Patterns  
[(base) santiagoarteaga@Santiagos-MacBook-Pro Patterns % mvn compile]  
  
[INFO] Scanning for projects...  
[INFO]  
[INFO] -----< edu.eci.cvds:Patterns >-----  
[INFO] Building Patterns 1.0-SNAPSHOT  
[INFO]   from pom.xml  
[INFO] -----[ jar ]-----  
[INFO]  
[INFO] --- resources:3.3.1:resources (default-resources) @ Patterns ---  
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!  
[INFO] skip non existing resourceDirectory /Users/santiagoarteaga/Patterns/src/main/resources  
[INFO]  
[INFO] --- compiler:3.13.0:compile (default-compile) @ Patterns ---  
[INFO] Recompiling the module because of changed source code.  
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!  
[INFO] Compiling 9 source files with javac [debug target 1.8] to target/classes  
[WARNING] bootstrap class path is not set in conjunction with -source 8
```

¿Cuál fábrica hiciste? y ¿Cuál es mejor?

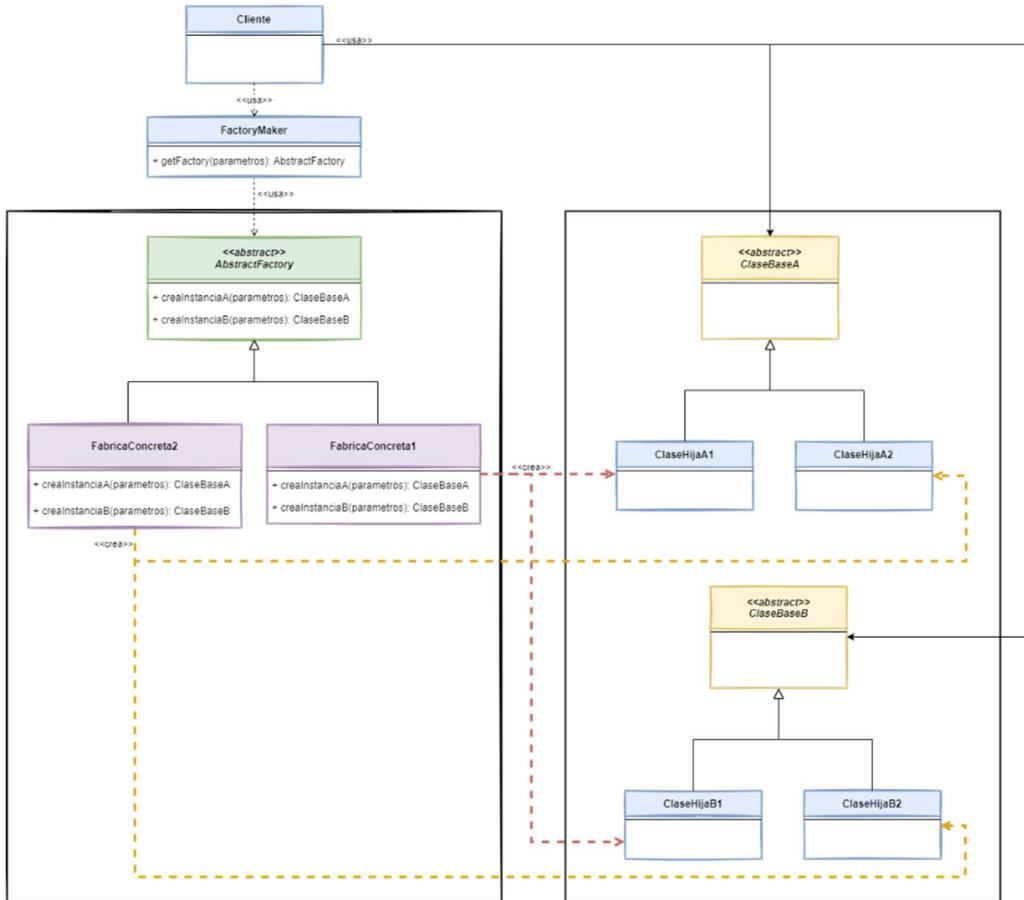
- Simple Factory:



- Factory Method:



- Abstract Factory:



## Cuál fábrica hiciste?

En este proyecto, implementamos una **Simple Factory** en el archivo `ShapeFactory.java`. Este patrón es conocido por su simplicidad y efectividad al proporcionar un único punto de creación para diferentes tipos de objetos. En este caso, la fábrica utiliza un método estático para recibir como parámetro una instancia de la enumeración `RegularShapeType` y, dependiendo del tipo recibido, retorna una instancia concreta de una de las clases `Triangle`, `Quadrilateral`, `Pentagon` o `Hexagon`. La lógica detrás de este enfoque es centralizar el proceso de creación de objetos, lo que facilita la gestión y el mantenimiento del código.

El método dentro de la fábrica emplea una estructura de control `switch-case`, evaluando el valor del tipo de figura geométrica solicitado y devolviendo la instancia adecuada. Si el tipo no coincide con ninguno de los casos definidos, puede lanzar una excepción o devolver un valor nulo. Este enfoque es práctico para manejar un conjunto limitado de clases, como en nuestro caso, donde cada clase representa una figura geométrica con un número fijo de lados.

### 1.1.1 ¿Cuál es mejor?

La elección del tipo de fábrica depende directamente del contexto y las necesidades del proyecto. Existen diferentes patrones de fábrica, cada uno con sus propias características, ventajas y desventajas:

- **Simple** **Factory:**  
Es la opción más sencilla y directa para centralizar la creación de objetos. Es adecuada cuando el número de clases a instanciar es limitado y no existe una gran variabilidad en la lógica de creación. Este patrón facilita el mantenimiento del código, ya que cualquier cambio en la creación de las instancias solo requiere modificar el método estático de la fábrica. Sin embargo, tiene la desventaja de ser menos flexible y no permite extender fácilmente las clases sin modificar la fábrica directamente.
- **Factory** **Method:**  
Este patrón proporciona mayor flexibilidad al delegar la responsabilidad de creación a subclases. Es útil cuando las clases a instanciar pueden cambiar o cuando cada clase requiere una lógica de creación personalizada. A diferencia de la Simple Factory, el Factory Method sigue el principio de inversión de dependencia, permitiendo que el código cliente dependa de abstracciones y no de implementaciones concretas. Aunque es más flexible, también es más complejo de implementar y mantener.
- **Abstract** **Factory:**  
Este patrón es el más avanzado y está diseñado para crear familias de objetos relacionados o dependientes entre sí. Es ideal cuando hay múltiples productos que deben ser compatibles entre ellos. Por ejemplo, si tuviéramos varias figuras geométricas agrupadas por categorías (polígonos regulares, polígonos irregulares, etc.), el Abstract Factory sería la elección adecuada. Sin embargo, su complejidad es significativamente mayor, y no siempre es necesario usarlo para problemas sencillos.

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:**

```
-----  
(base) santiagoarteaga@Santiagos-MacBook-Pro Patterns % mvn exec:java -Dexec.mainClass="edu.eci.cvds.patterns.shapes.ShapeMain"  
  
[INFO] Scanning for projects...  
[INFO]  
[INFO] -----< edu.eci.cvds:Patterns >-----  
[INFO] Building Patterns 1.0-SNAPSHOT  
[INFO]   from pom.xml  
[INFO] -----[ jar ]-----  
[INFO]  
[INFO] --- exec:3.0.0:java (default-cli) @ Patterns ---  
Parameter of type RegularShapeType is required.  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 1.426 s  
[INFO] Finished at: 2025-02-06T11:29:42-05:00  
[INFO] -----  
(base) santiagoarteaga@Santiagos-MacBook-Pro Patterns %
```

- Como no pasamos ningún argumento, el código entra en la primera condición:

```
java
CopiarEditar
if (args == null || args.length != 1) {
    System.err.println("Parameter of type RegularShapeType is required.");
    return;
}
```

args.length es 0, por lo que el programa muestra el mensaje de error y finaliza.

- **Parámetro: qwerty**

```
(base) santiagoarteaga@Santiagos-MacBook-Pro Patterns % mvn exec:java -Dexec.mainClass="edu.eci.cvds.patterns.shapes.ShapeMain" -Dexec.args="qwerty"

[INFO] Scanning for projects...
[INFO]
[INFO] -----< edu.eci.cvds:Patterns >-----
[INFO] Building Patterns 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- exec:3.0.0:java (default-cli) @ Patterns ---
Parameter 'qwerty' is not a valid RegularShapeType.
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time:  0.935 s
[INFO] Finished at: 2025-02-06T11:30:26-05:00
[INFO] -----
```

- args[0] es "qwerty", que no está en RegularShapeType (Triangle, Quadrilateral, Pentagon, Hexagon).

Al intentar convertir "qwerty" con RegularShapeType.valueOf(args[0]), se lanza una IllegalArgumentException, lo que provoca que el programa imprima este mensaje de error:

```
CopiarEditar
catch (IllegalArgumentException ex) {
    System.err.println("Parameter '" + args[0] + "' is not a valid RegularShapeType.");
}
```

- Parámetro: pentagon

```
(base) santiagoarteaga@Santiagos-MacBook-Pro Patterns % mvn exec:java -Dexec.mainClass="edu.eci.cvds.patterns.shapes.ShapeMain" -Dexec.args="pentagon"

[INFO] Scanning for projects...
[INFO]
[INFO] -----< edu.eci.cvds:Patterns >-----
[INFO] Building Patterns 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- exec:3.0.0:java (default-cli) @ Patterns ---
Parameter 'pentagon' is not a valid RegularShapeType.
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time:  1.139 s
[INFO] Finished at: 2025-02-06T11:30:57-05:00
[INFO] -----
```

- RegularShapeType.valueOf(args[0]) es sensible a mayúsculas y minúsculas. RegularShapeType está definido con Pentagon (con P mayúscula). Como "pentagon" no coincide exactamente con "Pentagon", se lanza una IllegalArgumentException y se imprime el mensaje de error.
- Parámetro: Hexagon

```

(base) santiagoarteaga@Santiagos-MacBook-Pro Patterns % mvn exec:java -Dexec.mainClass="edu.eci.cvds.patterns.shapes.ShapeMain" -Dexec.args="Hexagon"

[INFO] Scanning for projects...
[INFO]
[INFO] -----< edu.eci.cvds:Patterns >-----
[INFO] Building Patterns 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- exec:3.0.0:java (default-cli) @ Patterns ---
Successfully created a Hexagon with 6 sides.
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time:  1.184 s
[INFO] Finished at: 2025-02-06T11:31:38-05:00
[INFO]
(base) santiagoarteaga@Santiagos-MacBook-Pro Patterns %

```

args[0] es "Hexagon", que está definido en RegularShapeType.

Se llama a ShapeFactory.create(RegularShapeType.Hexagon), que devuelve una instancia de Hexagon.

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

**Las instrucciones que funcionan correctamente son aquellas que cumplen con los siguientes criterios:**

1. **El parámetro debe existir en RegularShapeType.**
  - La clase RegularShapeType solo contiene los valores definidos explícitamente (como Triangle, Quadrilateral, Pentagon, Hexagon).
  - Cualquier otro valor causará una IllegalArgumentException.
2. **La capitalización del parámetro debe coincidir exactamente con los valores en RegularShapeType.**
  - Java es sensible a mayúsculas y minúsculas, por lo que "pentagon" (minúsculas) **no** es igual a "Pentagon".

## **1.2 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.