6. Maven

El propósito de la presente sección es introducir al estudiante de la herramienta de desarrollo en Java Maven.

6.1. Primer proyecto en Maven

Ejercicio 18. Responda a las siguientes preguntas:

1. ¿Qué es Maven?

   Maven es una herramienta de software para la gestión y construcción de proyectos Java

2. ¿Qué es el repositorio central de Maven?, ¿hasta qué punto son fiables las bibliotecas que hay en él?

   El repositorio central de Maven es una herramienta para crear aplicaciones la cual proporciona un repositorio. El repositorio central de Maven es una fuente confiable para obtener bibliotecas y componentes de software.

3. ¿Qué es el repositorio local?

   El repositorio local es un directorio en tu sistema de archivos que actúa como almacén local de todas las dependencias descargadas por Maven.

Ejercicio 19. Instale Maven. Por ejemplo, en OpenSuse, se ejecuta:

```
sudo zypper install maven
```

Para comprobar que se ha instalado correctamente:

```
mvn --version
```

Para otros sistemas operativos, seleccione la opción *download* de la página de Maven:

https://maven.apache.org/

Ejercicio 20. Realice los siguientes apartados:

1. Cree un proyecto en Maven ejecutando la siguiente instrucción[1]:

   `mvn archetype:generate -DgroupId=org.pr2 -DartifactId= miPrimeraAplicacion -DarchetypeArtifactId=maven-archetype -quickstart -DarchetypeVersion=1.4 -DinteractiveMode= false` y explore el árbol de directorios generado.

2. ¿Qué es un arquetipo en Maven?

Un arquetipo es una plantilla o patrón predefinido que se utiliza para crear nuevos proyectos.

3. Entre el el directorio miPrimeraAplicacion:

```
cd miPrimeraAplicacion
```

4. Explique el fichero *pom.xml*.

   El fichero pom.xml es un archivo que define la configuración y la estructura de un proyecto Maven.

5. Explore el árbol de directorios.

6. Compile el programa: mvn compile

7. Ejecute el programa:

```
mvn exec:java -D exec.mainClass=org.pr2.App
```

8. Elimine los artefactos generados anteriormente, vuelva compilar y ejecute de nuevo:

```
mvn clean compile
mvn exec:java -D exec.mainClass=org.pr2.App
```

9. Genere la documentación del proyecto, y explórela:

```
mvn site cd target/site firefox
index.html &
```

10. Explique en qué consisten las siguientes fases:

validate: En esta fase, Maven valida el proyecto para asegurarse de que todas las dependencias están disponibles y que el proyecto está configurado correctamente.

compile: Durante esta fase, Maven compila el código fuente del proyecto.

test: En esta fase, Maven ejecuta las pruebas unitarias definidas en el proyecto.

package: Durante esta fase, Maven empaqueta el código compilado y otros archivos de recursos en un formato específico.

install: En esta fase, Maven instala el paquete empaquetado en el repositorio local de Maven.

deploy: La fase de despliegue es similar a la fase de instalación, pero implica la copia del paquete empaquetado en un repositorio remoto.

clean: Durante esta fase, Maven elimina todos los archivos generados previamente durante la construcción del proyecto.

site: En esta fase, Maven genera documentación del proyecto, como informes de análisis estático de código, documentación de JavaDoc, y otros informes sobre la estructura y la salud del proyecto.

6.2. La libreta de contactos

Ejercicio 21. El propósito del presente ejercicio es la realización de una aplicación sencilla (una libreta de contactos) utilizando Maven. Se pide realizar los apartados que se muestran a continuación:

1. Genere el proyecto:

   ```
   mvn archetype:generate -DgroupId=org.pr2 -DartifactId=libreta -contactos -
       DarchetypeArtifactId=maven-archetypequickstart -DarchetypeVersion=1.4 -
       DinteractiveMode=false
   ```

2. Explore el árbol de directorios de *src*:

   ```
   cd libreta-contactos tree
   ```

3. Explore el código de la clase principal:

   ```
   vi src/main/java/org/pr2/App.java
   ```

4. Compile:

   ```
   mvn compile
   ```

5. Compruebe que la apliación ejecuta:

   ```
   mvn exec:java -D exec.mainClass=org.pr2.App
   ```

6. Cree un directorio para el dominio de la aplicación y otro para la interfaz de usuario:

   ```
   md src/main/java/org/pr2/dominio md
   src/main/java/org/pr2/interfaces
   ```

7. Escriba las clases *Contacto.java* en el directorio *src/main/java/org/pr2/dominio*

```
package org.pr2.dominio;

public class Contacto
{ private String nombre; private String
        telefono;

        public void setNombre(String nombre) { this.nombre = nombre;
        }

        public String getNombre() { return nombre;
        }

        public void setTelefono(String telefono) { this.telefono = telefono;
        }

        public String getTelefono() { return telefono;
        }

        @Override public String toString() { return "nombre: " +
        getNombre() + " " +
                                        "telefono: " + getTelefono();
        }

        public Contacto(String nombre, String telefono) { this.nombre =
                nombre; this.telefono = telefono;
        }

        public Contacto() {
        }

}
```

8. En el mismo directorio, escriba la clase *Libreta.java*

```
package org.pr2.dominio;

import java.io.File; import
java.io.FileWriter; import
java.io.IOException; import
java.util.ArrayList; import
java.util.Scanner;

public class Libreta
{ private String nombreFichero = "contactos.txt"; private ArrayList<Contacto> contactos =
        new ArrayList<>()
                ;
        public void addContacto(Contacto contacto)
        { contactos.add(contacto);
```

```
    }

    @Override public String toString() {
        StringBuilder sb = new StringBuilder(); for(Contacto contacto : contactos)
        { sb.append(contacto + "\n");
        } return sb.toString();
    }

    public Libreta() { try {
                File fichero = new File(nombreFichero);
                fichero.createNewFile(); Scanner sc = new
                Scanner(fichero); while(sc.hasNext()) {
                    Contacto contacto = new Contacto();
                    contacto.setNombre(sc.nextLine());
                    contacto.setTelefono(sc.nextLine()); contactos.add(contacto);
                }
        }catch(IOException ex) {
                System.err.println(ex);
        }
    }

    private void volcarContactos()
    { System.out.println(contactos); try {
                FileWriter fw = new FileWriter(nombreFichero); for(Contacto contacto :
                contactos)
                { fw.write(contacto.getNombre()+"\n");
                    fw.write(contacto.getTelefono()+"\n");
                } fw.close();
        }catch(IOException ex) {
                System.err.println(ex);
        } }

    public void annadirContacto(Contacto contacto)
    { contactos.add(contacto);
        this.volcarContactos();
    }

}
```

9. Escriba la clase *Interfaz.java* en el directorio *src/main/java/org/pr2/interfaces*

```
package org.pr2.interfaces;

import org.pr2.dominio.*;

public class Interfaz
{ public static void iniciar(String args[])
    {
        Libreta libreta = new Libreta(); if
        (args[0].equals("add"))
        {
```

```
                    Contacto contacto = new Contacto(args[1], args
                        [2]); libreta.annadirContacto(contacto);
                } else if (args[0].equals("show")) System.out.println( libreta);
                else System.out.println("Opción incorrecta"); }
}
```

10. Modifique la clase *App.java*

```
package org.pr2;

import org.pr2.interfaces.*;

public class App
{ public static void main( String[] args )
    {
        Interfaz.iniciar(args);
    }
}
```

11. compile:

```
mvn compile
```

12. Ejecute:

```
mvn exec:java -D exec.mainClass=org.pr2.App -D exec.args='add
    "Juan García Pérez" 65432145' mvn exec:java -D exec.mainClass=org.pr2.App -
Dexec.args=show
```

6.3. La libreta de contactos ejecutable con **java -jar**

Ejercicio 22. Lleve a cabo los siguientes apartados:

1. Añada en la sección de properties del pom.xml la siguiente línea:

```
<aplicacion>org.pr2.App</aplicacion>
```

2. Añada el código que se muestra a continuación al pom.xml en la sección de plugins:

```
<plugin> <artifactId>maven-assembly-plugin</artifactId>
    <configuration>
        <archive>
            <manifest>
                <mainClass>\${aplicacion}</mainClass>
            </manifest>
        </archive>
        <descriptorRefs>
            <descriptorRef>jar-with-dependencies</ descriptorRef>
        </descriptorRefs>
    </configuration>
```

```
</plugin>
```

3. Genere el .jar invocando el plugin:

   ```
   mvn compile assembly:single
   ```

4. Compruebe el funcionamiento de la aplicación:

   ```
   java -jar target/libreta-contactos-1.0-SNAPSHOT-jar-withdependencies.jar show
   ```

6.4. La libreta de contactos con uso de una biblioteca *jar*

Para que funcione el software generado en el presente ejercicio, es necesario haber incluido el plugin *maven-assembly-plugin* (véase ejercicio 22).

Ejercicio 23. Se pide realizar los apartados que se muestran a continuación:

1. Acceda al repositorio central de Maven:

   https://mvnrepository.com/repos/central

2. Busque la biblioteca *jopendocument*.

3. Incluya el código de la dependencia en el *pom.xml*.

4. Modifique el código de la clase *Libreta.java*

```
package org.pr2.dominio;

import java.io.File; import java.io.FileNotFoundException;
import java.io.FileWriter; import java.io.IOException; import
java.util.ArrayList; import java.util.Scanner; import
javax.swing.table.DefaultTableModel; import
javax.swing.table.TableModel;

import org.jopendocument.dom.spreadsheet.SpreadSheet;

/**
                * Clase responsable de mantener la libreta de contactos.
 */ public class Libreta
{ private String nombreFichero = "contactos.txt"; private ArrayList<Contacto> contactos =
            new ArrayList <>();

            /**
              * Genera una cadena de caracteres a partir de la libreta.
              * @return la cadena de caracteres.
              */
            @Override public String toString() {
```

```
            StringBuilder sb = new StringBuilder(); for(Contacto contacto :
            contactos)
            { sb.append(contacto + "\n");
            } return sb.toString();
}

/**
  * Lee la libreta con sus contactos del fichero <i> contactos.txt</i>.
  */ public Libreta() { try {
                                        File fichero = new File(nombreFichero
                                    ); fichero.createNewFile(); Scanner sc = new
                            Scanner(fichero); while(sc.hasNext()) {
                                            Contacto contacto = new
                                        Contacto(); contacto.setNombre(sc.
                                            nextLine());
                                    contacto.setTelefono(sc. nextLine());
                                    contactos.add(contacto);
                            }
            }catch(IOException ex)
            {
                                System.err.println(ex);
            } }

private void volcarContactos()
{ System.out.println(contactos); try {
                            FileWriter fw = new FileWriter( nombreFichero);
                            for(Contacto contacto : contactos)
                            { fw.write(contacto.getNombre()
                                        +"\n"); fw.write(contacto.getTelefono
                                        ()+"\n"); }
                            fw.close();
            }catch(IOException ex)
            {
                                System.err.println(ex);
            }
}

/**
  *   Añade el <i>contacto</i> a la libreta.
  */
public void annadirContacto(String nombre, String telefono)
{
            Contacto contacto = new Contacto(nombre, telefono);
            contactos.add(contacto); this.volcarContactos();
}

/**
  *   Borra el contacto cuyo nombre es el indicado en el parámetro
  *
  *   @param nombre del contacto a borrar.
```

```
        */
        public void borrarContacto(String nombre)
        {
                Contacto contacto = new Contacto();
                contacto.setNombre(nombre); contactos.remove(contacto);
                this.volcarContactos();
        }

    /**
         * Genera una hoja de cálculo
      */ public void generarHojaDeCalculo()
    { final Object[][] datos = new Object[contactos.size()
            ][2];
        int i = 0;
        for(Contacto contacto : contactos)
        { datos[i][0] = contacto.getNombre(); datos[i++][1] =
            contacto.getTelefono();
        }

        String[] columnas = new String[] { "Nombre", "Telé fono" };

        TableModel modelo = new DefaultTableModel(datos, columnas);

        final File file = new File("output/contactos.ods"); try {
            SpreadSheet.createEmpty(modelo).saveAs(file);
        }catch(IOException ex) {
            System.out.println(ex);
        }
    }

}
```

5. Modifique el código de la clase *Interfaz.java*

```
package org.pr2.interfaces;

import org.pr2.dominio.*;

/**
     * Implementa una interfaz texto para la libreta de contactos .
  */ public class Interfaz {
    /**
     * Inicia la interfaz con parámetros.
     * @param args puede ser <i>add nombre contacto</i> (p.ej
        . <i>add Juan
     * 653421367</i>) para añadir contacto, o <i>show</i> para mostrar todos
     * los contactos de la libreta. */
    public static void iniciar(String args[])
    {
```

```
        Libreta libreta = new Libreta(); if
        (args[0].equals("add"))
        { libreta.annadirContacto(args[1], args[2]);
        } else if (args[0].equals("rm"))
        { libreta.borrarContacto(args[1]);
        } else if (args[0].equals("show")) System.out.println( libreta);
        else if (args[0].equals("hoja"))
        { libreta.generarHojaDeCalculo();
              System.out.println("Hoja de cálculo generada en
                    output/contactos.ods");
        } else System.out.println("Opción incorrecta"); }
}
```

6. Compruebe que funciona correctamente:

   ```
   mvn clean compile assembly:single java -jar target/libreta-contactos-1.0-SNAPSHOT-
   jar-withdependencies.jar hoja
   ```

7. Genere el *javadoc* de la aplicación:

   ```
   mvn javadoc:javadoc
   ```

6.5. Arquetipos

Ejercicio 24. Acceda a la siguiente página Web y lea el inventario de arquetipos disponibles para Maven:

https://maven.apache.org/archetypes/index.html

Ejercicio 25. Cree un proyecto para una aplicación Web y compruebe su funcionamient:

```
mvn archetype:generate -DgroupId=org.pr2 -DartifactId=web-simple -
                              DarchetypeArtifactId=maven-archetype-webapp -DarchetypeVersion
    =1.4 -DinteractiveMode=false cd web-
simple/ mvn package
firefox target/web-simple/index.jsp &
```

Ejercicio 26. El propósito de este ejercicio es que el estudiante aprenda a elaborar su propio arquetipo. Se pide llevar a cabo los siguientes pasos:

1. Genere el arquetipo[1]:

   ```
   mvn archetype:generate -DarchetypeGroupId=org.apache.maven.
         archetypes -DarchetypeArtifactId=maven-archetypearchetype -
         DarchetypeVersion=1.4
   ```

[1] https://maven.apache.org/archetypes/maven-archetype-archetype/

La respuestas a las preguntas que le formule Maven son las siguientes:

- $groupId = org.pr2$ $artifactId =$
- $mi - arquetipo$ $version = 1.0$
- $'package' org.pr2$: Se deja vacío.
- $Y : Y$
-

2. Entre el directorio que se acaba de crear:

   cd mi-arquetipo

3. Abra el fichero

   src/main/resources/archetype-resources/pom.xml

4. Realice los cambios indicados en el ejercicio 22.

5. Valide el código:

   mvn validate

6. Instale el arquetipo en el repositorio local:

   mvn install

7. Explore el siguiente directorio a partir del $HOME:

   .m2/repository/org/pr2/mi-arquetipo/1.0/

8. Con el propósito de que pueda tener disponible el arquetipo creado en otros ordenadores o, en caso de que trabaje en modo kiosco en el laboratorio, pueda tenerlo disponible de una clase para otra, se recomienda también mantener los fuentes del arquetipo en un repositorio Git. Para ello, puede seguir los pasos que se muestran a continuación:

   a) Cree el repositorio remoto en Bitbucket.

   b) Clone el repositorio en el directorio donde ha creado el arquetipo.

   c) Cree un *.gitignore* para descartar el contenido del directorio *target* así como de los ficheros *.swp* temporales de Vi:

   ```
   /target/
   .*
   !/.gitignore
   ```

   d) Añada todos los ficheros y directorios del arquetipo: git add .

e) Realice el *commit* y el *push*:

```
git commit -am "Versión 1.0 de mi arquetipo" git push -u origin
master
```

9. Use el arquetipo. Para ello, en primer lugar, vaya a otro directorio y ejecute:

```
mvn archetype:generate -DgroupId=org.pr2 -DartifactId=prueba -
    DarchetypeGroupId=org.pr2 -DarchetypeArtifactId=miarquetipo -
    DarchetypeVersion=1.0 -DinteractiveMode=false
```

10. Compruebe el resultado de

```
cd prueba tree
```

11. Luego, genere el *.jar*:

```
mvn clean compile assembly:single
```

12. Compruebe que ejecuta el programa resultante:

```
java -jar target/prueba-1.0-jar-with-dependencies.jar
```

Se pretende codificar, de acuerdo con la técnica de desarrollo dirigido por pruebas (TDD), el método *static public BigDecimal sumaRecursivaElementosPila(Stack pila)* de tal forma que realice la misma función que el anterior, pero de manera recursiva. Se pide:

1. La secuencia de tests y de versiones del método a desarrollar según TDD que culmine con el método implementado que pase todos los tests. En la respuesta del estudiante debe quedar claro qué versión del software hace pasar cada test.

2. El análisis de la complejidad temporal asintótica tanto del método dado en el enunciado como del método que sea la versión final del estudiante.