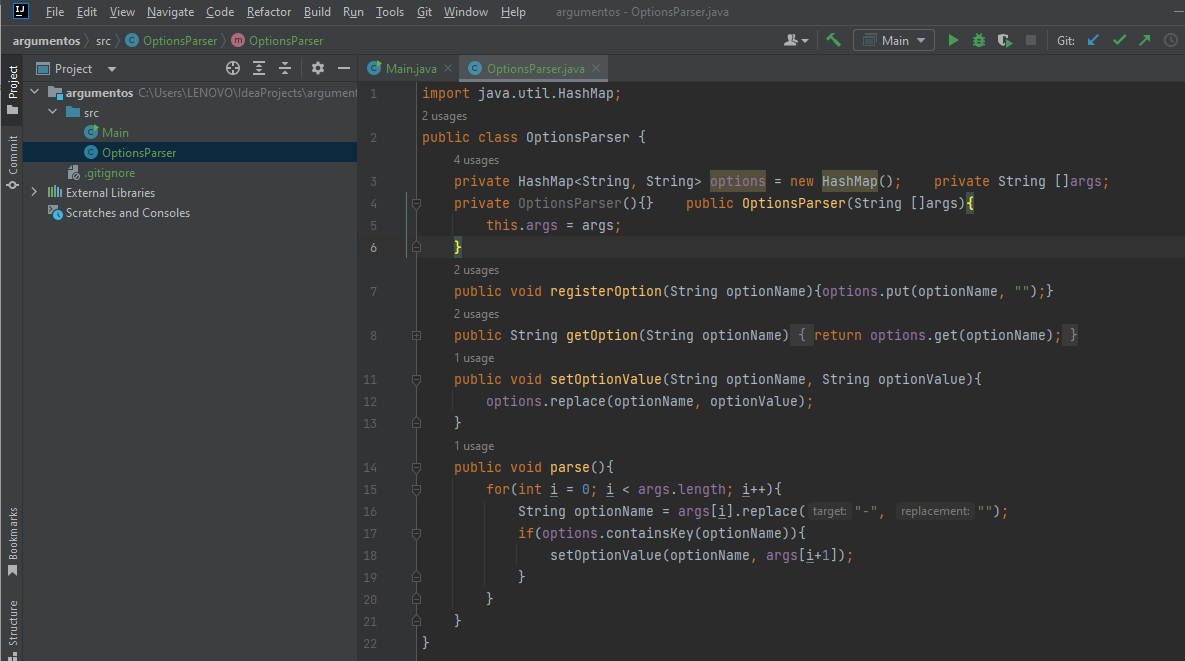
**Ejercicios sesiones 19 20 y 21**

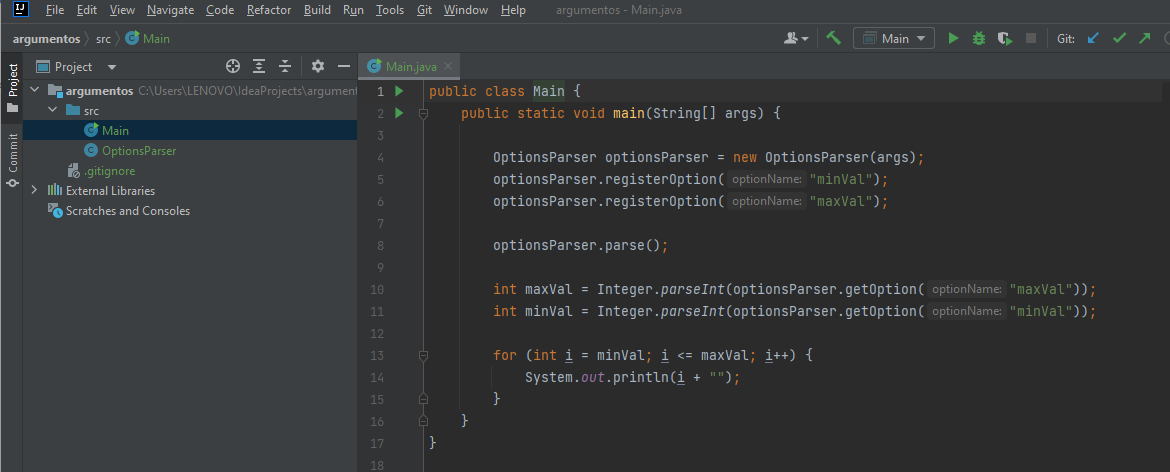
**19- Implementa Args en uno de los proyectos que se han ido trabajando y aplica las buenas prácticas que se han visto en la sesión 19.**

Implementacion para generar un arreglo y obtener sus valores

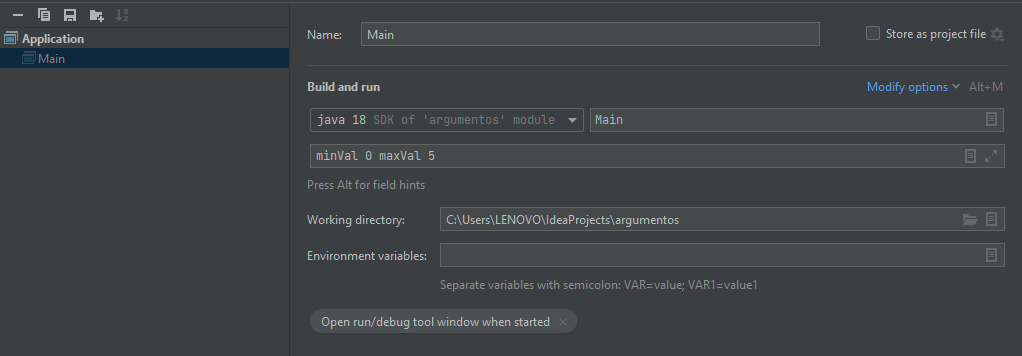
Calse OptionsParser. Contiene un HashMap donde con métodos de registro, obtención y seteo de valores. Ciclo para reemplazar caracteres en el parseo de la información.



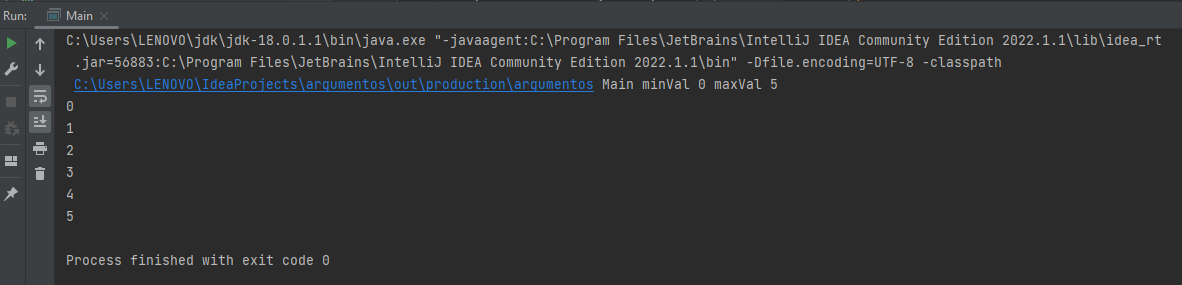
Clase Main



En Intellij Main 🡪 EditConfigurations configuramos los argumentos (minVal y maxVal).



La salida nos muestra los valores entre minVal y maxVal



**20- Identifica las características principales de la arquitectura limpia, indica en que casos se tendría que utilizar la arquitectura limpia y como lo harías.**

Los 5 principios SOLID de diseño de aplicaciones de software son:

S – Single Responsibility Principle (SRP)

Según este principio “una clase debería tener una, y solo una, razón para cambiar

Reúne las cosas que cambian por las mismas razones. Separa aquellas que cambian

por razones diferentes”

O – Open/Closed Principle (OCP)

Las clases que usas deberían estar abiertas para poder extenderse y cerradas para

modificarse.

L – Liskov Substitution Principle (LSP)

Dice que “las clases derivadas deben poder sustituirse por sus clases base”.

I – Interface Segregation Principle (ISP)

“Haz interfaces que sean específicas para un tipo de cliente”, es decir, para una finalidad concreta.

D – Dependency Inversion Principle (DIP)

“Depende de abstracciones, no de clases concretas”.

Así, Robert C. Martin recomienda:

Entre los objetivos de tener en cuenta estos 5 principios a la hora de escribir código encontramos:

Crear un software eficaz: que cumpla con su cometido y que sea robusto y estable.

Escribir un código limpio y flexible ante los cambios: que se pueda modificar fácilmente según necesidad, que sea reutilizable y mantenible.

Permitir escalabilidad: que acepte ser ampliado con nuevas funcionalidades de manera ágil.

En definitiva, desarrollar un software de calidad.

En este sentido la aplicación de los principios SOLID está muy relacionada con la comprensión y el uso de patrones de diseño, que nos permitirán mantener una alta cohesión y, por tanto, un bajo acoplamiento de software.

**Fuente: https://profile.es/blog/principios-solid-desarrollo-software-calidad/**

**21- Teniendo en cuenta lo visto en la sesión 21, utiliza un proyecto de los trabajados y límpialo. Si encuentras código duplicado, elimínalo y realiza la minificación de las clases y métodos.**

En el código original se utilizan muchas más líneas de código para listar, obtener y crear un Usuario respecto del código final

package Ejercicio21JavaAvanzado.Inicial;  
  
import java.io.File;  
import java.io.PrintStream;  
import java.util.ArrayList;  
import java.util.Scanner;  
  
public class Usuarios {  
 public String ficheroDatos = "usuarios.txt";  
  
 public ArrayList<Usuario> listarUsuarios() {  
 ArrayList<Usuario> usuarios = new ArrayList();  
  
 try {  
 Scanner scanner = new Scanner(new File(ficheroDatos));  
  
 while (scanner.hasNext()) {  
 String usuarioActual = scanner.next();  
 String []partes = usuarioActual.split(",");  
  
 Usuario usuario = new Usuario();  
 usuario.nombreUsuario = partes[0];  
 usuario.nombre = partes[1];  
 usuario.apellidos = partes[2];  
 usuario.email = partes[3];  
 usuario.nivelAcceso = Integer.*parseInt*(partes[4]);  
  
 usuarios.add(usuario);  
 }  
  
 scanner.close();  
 } catch (Exception e) {  
 }  
  
 return usuarios;  
 }

public Usuario obtenerUsuario(String username) {  
 try {  
 Scanner scanner = new Scanner(new File(ficheroDatos));  
 ArrayList<String> usuarios = new ArrayList();  
  
 while (scanner.hasNext()) {  
 usuarios.add(scanner.next());  
 }  
  
 scanner.close();  
  
 for (String usuario : usuarios) {  
 String []partes = usuario.split(",");  
 String nombreUsuarioActual = partes[0];  
 System.*out*.println(username + " " + nombreUsuarioActual);  
  
 if (!nombreUsuarioActual.equalsIgnoreCase(username)) {  
 continue;  
 }  
  
 Usuario usuarioRetorno = new Usuario();  
 usuarioRetorno.nombreUsuario = partes[0];  
 usuarioRetorno.nombre = partes[1];  
 usuarioRetorno.apellidos = partes[2];  
 usuarioRetorno.email = partes[3];  
 usuarioRetorno.nivelAcceso = Integer.*parseInt*(partes[4]);  
 return usuarioRetorno;  
 }  
  
 } catch (Exception e) {  
 }  
  
 return null;  
}  
  
public void crearUsuario(Usuario usuario) {  
 try {  
 Scanner scanner = new Scanner(new File(ficheroDatos));  
 ArrayList<String> usuarios = new ArrayList();  
  
 while (scanner.hasNext()) {  
 usuarios.add(scanner.next());  
 }  
  
 scanner.close();  
  
 for (String usuarioActual : usuarios) {  
 String []partes = usuarioActual.split(",");  
 String nombreUsuarioActual = partes[0];  
  
 if (nombreUsuarioActual.toLowerCase().equals(usuario.nombreUsuario.toLowerCase())) {  
 return;  
 }  
 }  
 } catch (Exception e) {  
 }  
  
 try {  
 PrintStream printStream = new PrintStream(ficheroDatos);  
 String buffer = "";  
  
 for (Usuario usuarioExistente : listarUsuarios()) {  
 buffer += usuarioExistente.nombreUsuario + ","  
 + usuarioExistente.nombre + ","  
 + usuarioExistente.apellidos + ","  
 + usuarioExistente.email + ","  
 + usuarioExistente.nivelAcceso;  
 }  
  
 buffer += usuario.nombreUsuario + ","  
 + usuario.nombre + ","  
 + usuario.apellidos + ","  
 + usuario.email + ","  
 + usuario.nivelAcceso;  
  
 printStream.println(buffer);  
 } catch (Exception e) {  
  
 }  
}

**Código final**

package Ejercicio21JavaAvanzado.Final;  
  
import java.io.File;  
import java.io.PrintStream;  
import java.util.ArrayList;  
import java.util.Scanner;  
  
public class Usuarios {  
 public String ficheroDatos = "usuarios.txt";  
  
 private ArrayList<Usuario> convertirUsuariosFicheroEnArrayList() {  
 ArrayList<Usuario> usuarios = new ArrayList();  
  
 try {  
 Scanner scanner = new Scanner(new File(ficheroDatos));  
  
 while (scanner.hasNext()) {  
 String usuarioActual = scanner.next();  
 String []partes = usuarioActual.split(",");  
  
 Usuario usuario = new Usuario();  
 usuario.nombreUsuario = partes[0];  
 usuario.nombre = partes[1];  
 usuario.apellidos = partes[2];  
 usuario.email = partes[3];  
 usuario.nivelAcceso = Integer.*parseInt*(partes[4]);  
  
 usuarios.add(usuario);  
 }  
  
 scanner.close();  
 } catch (Exception e) {  
 }  
  
 return usuarios;  
 }

public ArrayList<Usuario> listarUsuarios() {  
 return convertirUsuariosFicheroEnArrayList();  
}  
  
public Usuario obtenerUsuario(String username) {  
 ArrayList<Usuario> usuarios = convertirUsuariosFicheroEnArrayList();  
  
 for (Usuario usuarioActual : usuarios) {  
 if (usuarioActual.nombreUsuario.equalsIgnoreCase(username)) {  
 return usuarioActual;  
 }  
 }  
  
 return null;  
}  
  
public void crearUsuario(Usuario usuario) {  
 if (obtenerUsuario(usuario.nombreUsuario) != null) {  
 return;  
 }  
  
 try {  
 PrintStream printStream = new PrintStream(ficheroDatos);  
  
 for (Usuario usuarioExistente : convertirUsuariosFicheroEnArrayList()) {  
 printStream.println(separarUsuarioPorComas(usuarioExistente));  
 }  
  
 printStream.println(separarUsuarioPorComas(usuario));  
 } catch (Exception e) {  
 System.*out*.println("Error al escribir: " + e.getMessage());  
 }  
}