1. Clase Main

Esta clase es la que inicia la aplicación. Lo que hace es ejecutar la interfaz gráfica de forma segura usando el método invokeLater de Swing. Así, se asegura que toda la UI se maneje en el hilo de eventos.

Código:

```
public class Main {
   public static void main(String[] args) {
      javax.swing.SwingUtilities.invokeLater(new Runnable() {
         public void run() {
            new VentanaTraductor();
         }
      });
   }
}
```

- Se usa SwingUtilities.invokeLater para iniciar la ventana del traductor en el hilo adecuado.
- Se crea una instancia de VentanaTraductor, que es la ventana principal de la aplicación.

2. Clase Traductor

Esta clase se encarga de la lógica de traducción del código. Está pensada para analizar el código fuente, validarlo según el lenguaje de origen y luego aplicar ciertas reglas de conversión para transformar el código al lenguaje destino.

Estructura y elementos principales:

- Clases internas:
    - ErrorInfo:
      Guarda información sobre errores encontrados durante la traducción (número de línea y mensaje).

- ResultadoTraduccion:
  Contiene el resultado final de la traducción y la lista de errores detectados.

- Método traducir:
  Es el método principal que recibe el código, el idioma de origen y el idioma destino. Primero valida que el código tenga características mínimas del lenguaje indicado (por ejemplo, en Java se espera que aparezca una declaración de clase y el método main).

  Luego, se recorre el código línea a línea:
  - Se verifica si hay errores simples (como la presencia de la palabra "error").
  - Se traduce cada línea usando un método auxiliar que selecciona la conversión adecuada según el par de lenguajes.

- Métodos auxiliares de traducción:
  Se han implementado funciones específicas para convertir de:

  - Java a JavaScript: Se omite la declaración de clase y se convierte el main y System.out.println a console.log.
  - Java a C++: Se traduce el main a int main() y se convierte System.out.println en una expresión con std::cout << y << std::endl;.
  - C++ a JavaScript: Se omiten directivas de preprocesador y se cambia std::cout por console.log.
  - Otros casos (como de C++ a Java o de JS a Java/C++) se hacen de forma básica, y se puede ampliar según las necesidades.

- Fragmento representativo:

```
public class Traductor {
   // Clase para almacenar errores en la traducción.
   public static class ErrorInfo {
      public int linea;
      public String mensaje;
      public ErrorInfo(int linea, String mensaje) {
         this.linea = linea;
         this.mensaje = mensaje;
      }
   }

   // Clase que representa el resultado de la traducción.
   public static class ResultadoTraduccion {
      public String traduccion;
      public List<ErrorInfo> errores;
      public ResultadoTraduccion(String traduccion, List<ErrorInfo> errores) {
         this.traduccion = traduccion;
         this.errores = errores;
      }
   }

   /**
    * Traduce el código fuente verificando primero que corresponda al lenguaje indicado.
    * Si no cumple con las características mínimas, se añade un error y se detiene la traducción.
    */
   public static ResultadoTraduccion traducir(String codigo, String idiomaOrigen, String idiomaDestino) throws Exception {
      // ... (validación y traducción línea a línea)
   }
```

```
    // Métodos auxiliares para traducir de un lenguaje a otro.
    private static String traducirLineaJavaToJS(String linea, int numLinea,
List<ErrorInfo> errores) { /*...*/ }
    private static String traducirLineaJavaToCpp(String linea, int numLinea,
List<ErrorInfo> errores) { /*...*/ }
    private static String traducirLineaCppToJS(String linea, int numLinea,
List<ErrorInfo> errores) { /*...*/ }
    private static String traducirLineaCppToJava(String linea, int numLinea,
List<ErrorInfo> errores) { /*...*/ }
    private static String traducirLineaJSToJava(String linea, int numLinea,
List<ErrorInfo> errores) { /*...*/ }
    private static String traducirLineaJSToCpp(String linea, int numLinea,
List<ErrorInfo> errores) { /*...*/ }
}
```

3. Clase VentanaTraductor

Esta es la interfaz gráfica del traductor, implementada en Java Swing. La idea fue crear una ventana moderna y sencilla, con:

- Un área de texto para el código fuente:
  Con un *placeholder* que dice "Copia o pega tu código aquí". Cuando el usuario hace foco, el placeholder se borra para facilitar la edición.
- Controles para seleccionar idiomas:
  Dos combo boxes para elegir el idioma de origen y el destino (por ejemplo, Java, C++ o JS).
- Botones de acción:
  - Traducir: Ejecuta el traductor.
  - Limpiar: Limpia el área de código fuente.
  - Nuevo: Reinicia la ventana (limpia el área de código, la tabla de errores y cierra la ventana de traducción si está abierta).

- Tabla de errores:
  Muestra en un panel inferior los errores detectados durante la traducción (con número de línea y descripción).
- Ventana emergente para el código traducido:
  Se abre una nueva ventana con el resultado traducido y se le agrega un botón "Copiar" para copiar el resultado al portapapeles.

- Fragmento representativo:

```
public class VentanaTraductor extends JFrame {
   // Variables para componentes de la interfaz (áreas de texto, tablas, botones, etc.)
   // ...
   private final String PLACEHOLDER = "Copia o pega tu código aquí";

   public VentanaTraductor() {
      // Configuración de la ventana, tamaño, título, etc.
      // Llama al método initComponentes() para montar toda la UI.
   }

   private void initComponentes() {
      // Se configura el look and feel, se crean paneles, se añaden los componentes.
      // Se crea el área de código fuente con placeholder.
      // Se configuran los botones y se añaden los listeners para las acciones.
   }

   private void traducirCodigo() {
      // Se recoge el código fuente, se valida, se llama al método Traductor.traducir()
      // Si hay errores, se muestran en la tabla; si no, se muestra el código traducido en una nueva ventana.
   }
```

```
    private void mostrarTraduccion(String codigoTraducido) {
        // Se crea una nueva ventana con un área de texto que muestra el código traducido.
        // Se añade un botón "Copiar" para pasar el código al portapapeles.
    }

    // Métodos para limpiar y reiniciar la ventana.
}
```

Notas adicionales:

- Se ha aplicado el estilo Nimbus para darle un toque moderno a la interfaz.
- La ventana emergente para el código traducido tiene un diseño simple y funcional, y el botón "Copiar" usa el portapapeles de Java para facilitar su uso.