


Compiladores e Intérpretes

Pautas Etapa 1 – Analizador Léxico

Universidad Nacional del Sur
Departamento de Ciencias e Ingeniería de la Computación
2022



Introducción

- El desarrollo de esta **etapa** del proyecto se **corresponde** con el **Tema 2 – Análisis Léxico** de la materia
- Te recomendamos que **hayas mirado y tengas frescos** todos los temas allí tratados **antes de arrancar** el diseño/desarrollo de esta etapa

Introducción

- En esta **etapa** hay que **diseñar e implementar** un **Analizador Léxico** para **MiniJava**
- Para poder usar el Léxico creado además van a tener que hacer un **Modulo Principal cliente** del Léxico que:
 - Manejará la **interfaz** con el **usuario** y
 - **Mostrará los tokens** reconocidos por el Léxico
 - **Reportará los errores léxicos** detectados por el Léxico

Indice

- En estas pautas vamos a ver lo siguiente:
 - [Diseño del Analizador Léxico](#)
 - [Estructura y Funcionalidad del Software a Desarrollar](#)
 - [Convenciones y Consideraciones](#)
 - [Informe](#)
 - [Logros disponibles para esta etapa](#)

Diseño de Analizador Léxico

Diseño del Analizador Léxico

- **Antes que nada:** leer detalladamente el documento de Aspectos Léxicos de MiniJava en el Moodle de la Materia

2 Componentes Léxicos

El primer paso para compilar un programa de MINIJAVA radica en convertir una entrada de caracteres en una entrada de tokens, donde cada token se corresponderá con un lexema de MINIJAVA. Los lexemas de MINIJAVA son palabras clave, nombres de tipos primitivos, literales, símbolos de puntuación, operadores e identificadores. Cada uno de estos elementos se describe en las siguientes secciones.

2.1 Espacios en Blanco

Los espacios en blanco en MINIJAVA, al igual que en Java, simplemente hacen más fácil la lectura del programa por un humano y **no** son tokens. Un espacio en blanco es un espacio, tab o un salto de línea.

2.2 Palabras Clave

MINIJAVA tiene las siguientes palabras clave:

Diseño del Analizador Léxico

- En base a la **información** de ese **documento** tienen que seguir los pasos para **diseñar el analizador léxico**
- Es decir llegar a **construir su autómata**, para eso deben:
 - 1) Identificar los **tokens** del lenguaje
 - 2) Especificar sus **Expresiones Regulares (ERs)**
 - 3) En base a esas ERs armar los **autómatas individuales**
 - 4) Con los autómatas individuales integrarlos en el **autómata general**

Diseño del Analizador Léxico

- En base a la **información** de seguir los pasos para **diseñar**

- Es decir llegar a **construir su**

- 1) Identificar los **tokens** del lenguaje,

- 2) Especificar sus **Expresiones Regulares (ERs)**

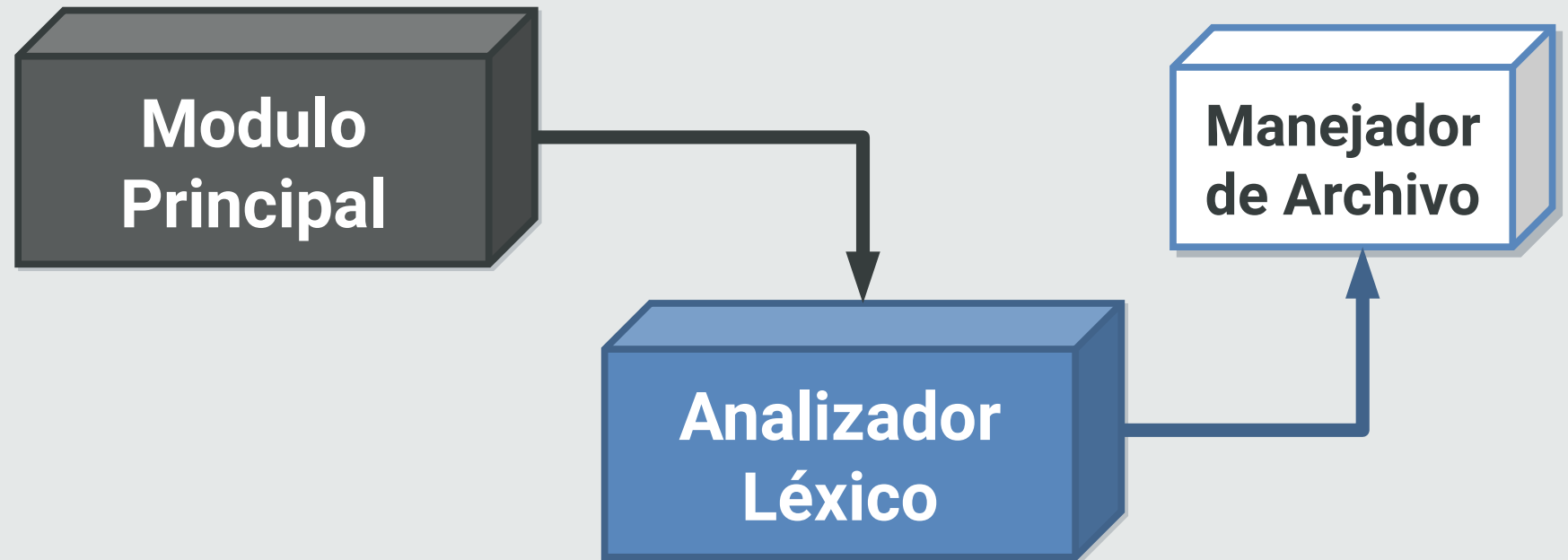
- 3) En base a esas ERs armar los **autómatas individuales**

- 4) Con los autómatas individuales integrarlos en el **autómata general**

Acá hay que omitir las **palabras reservadas** por que las vamos a tratar de una **forma especial** como fue explicado en el Tema 2.4 al final

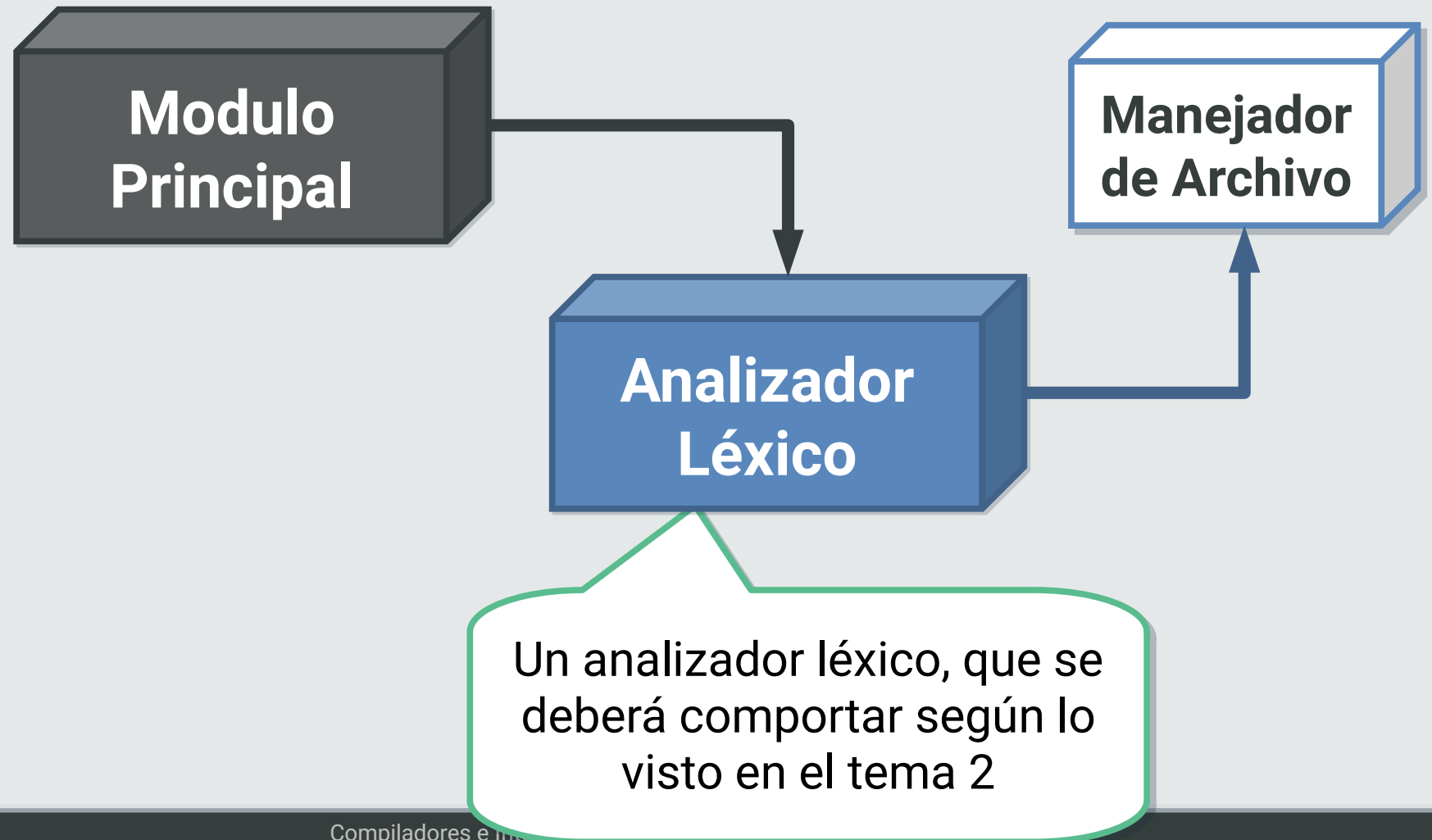
Estructura y Funcionalidad del Software a Desarrollar

Esquema General



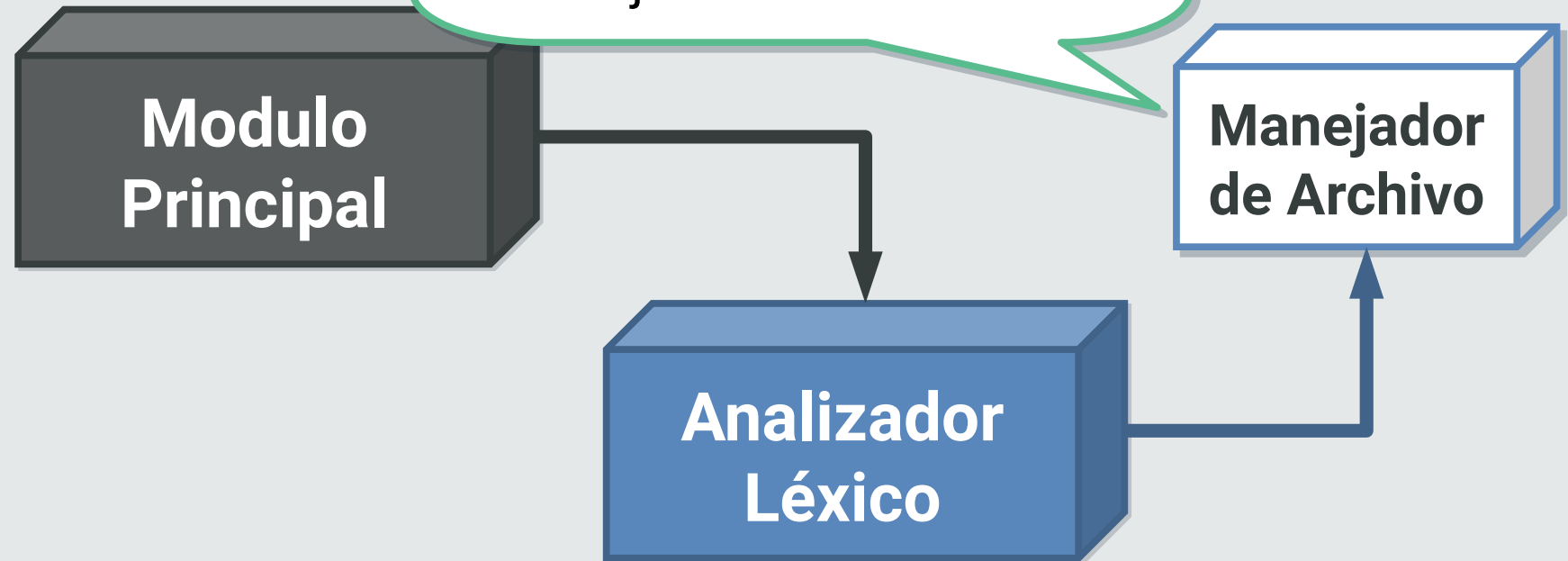
La base del proyecto
consistirá en implementar
estos tres módulos

Esquema General

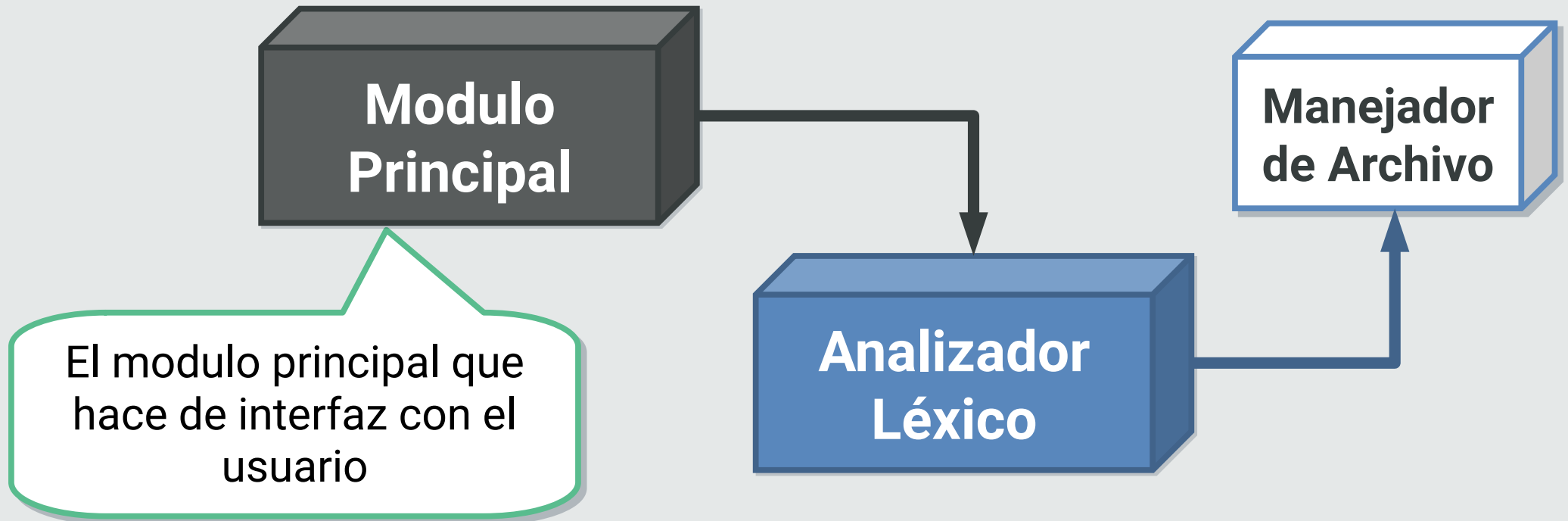


Esque

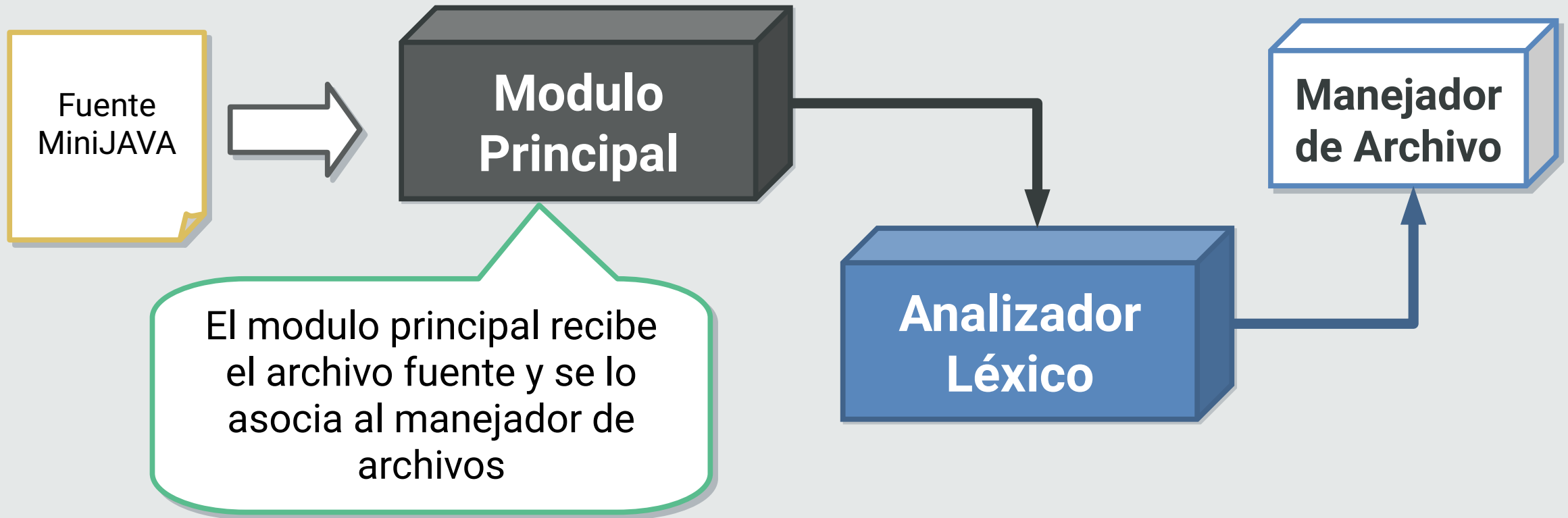
Un manejador de archivo que abstraerá al Léxico de los detalles de bajo nivel del manejo del archivo fuente



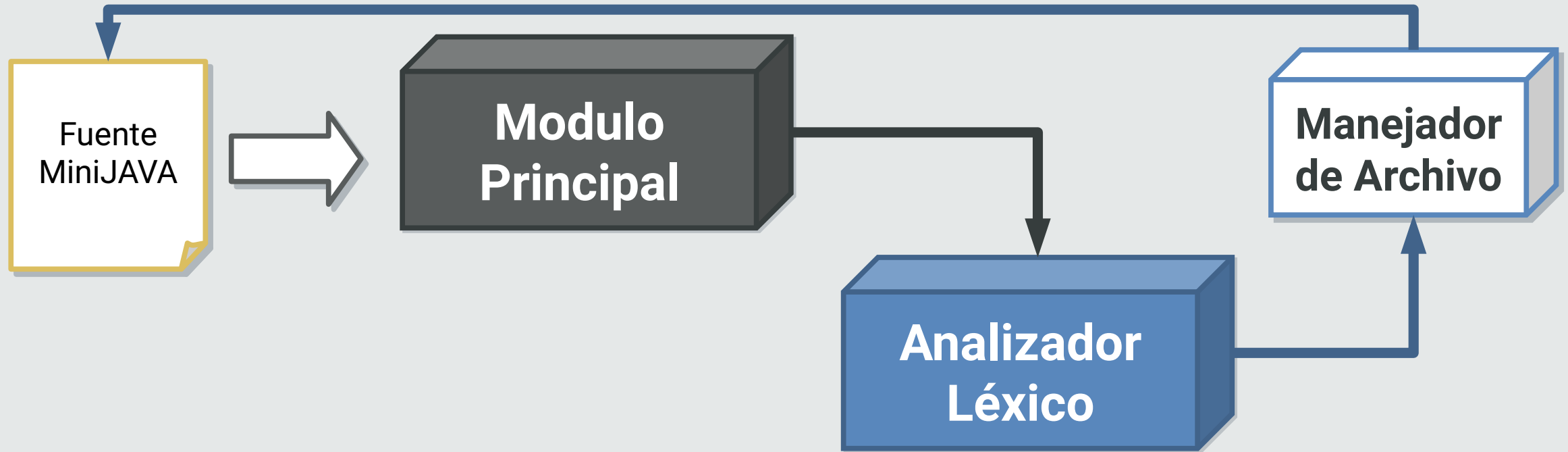
Esquema General



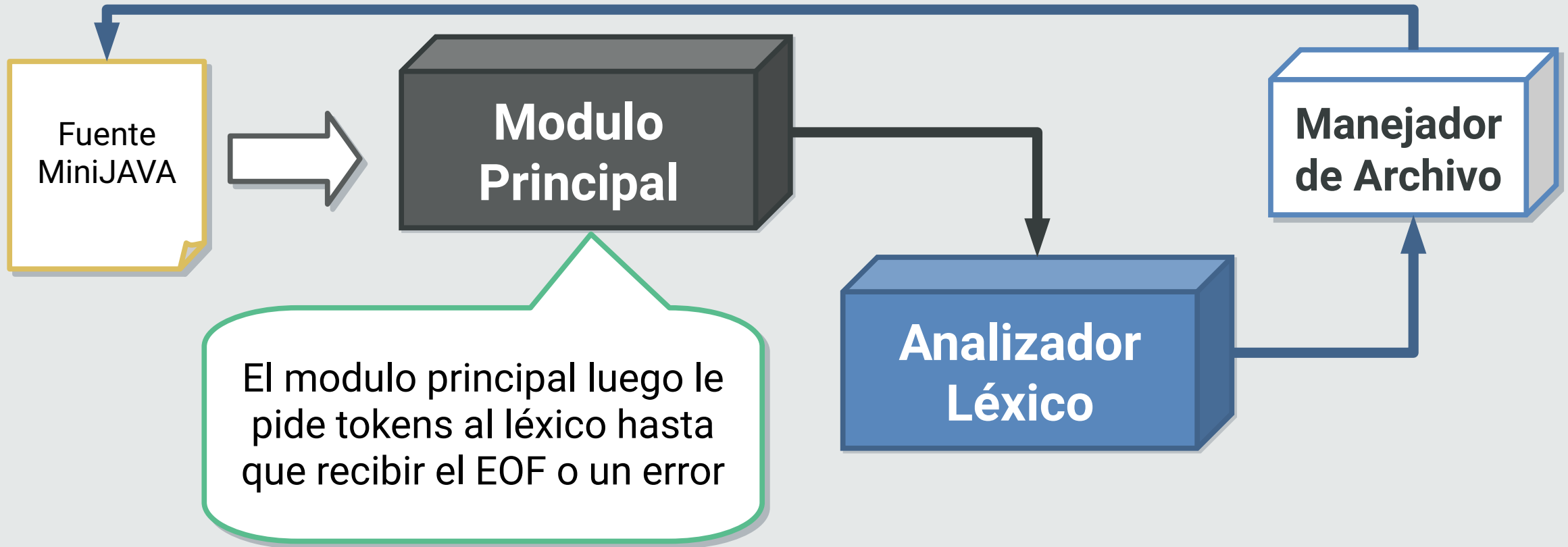
Funcionamiento e Interacción entre los Módulos



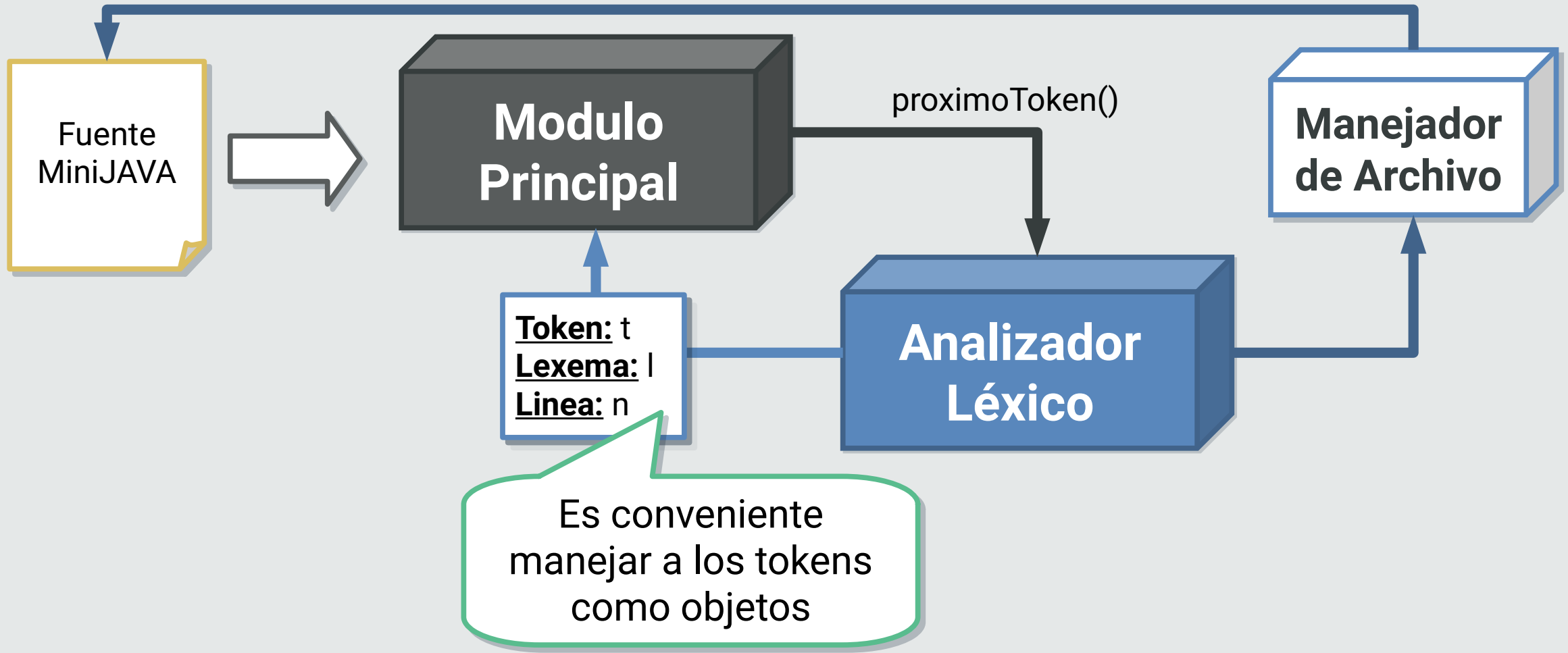
Funcionamiento e Interacción entre los Módulos



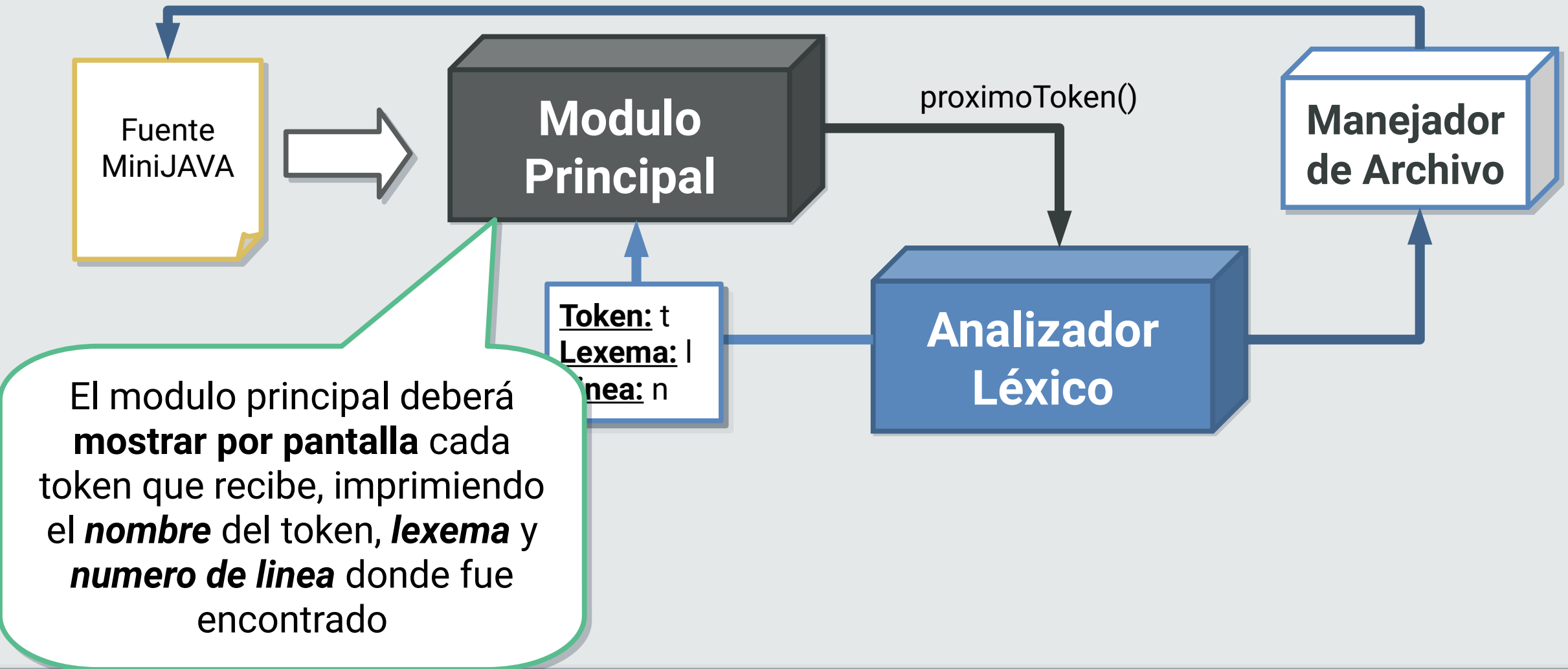
Funcionamiento e Interacción entre los Modulos



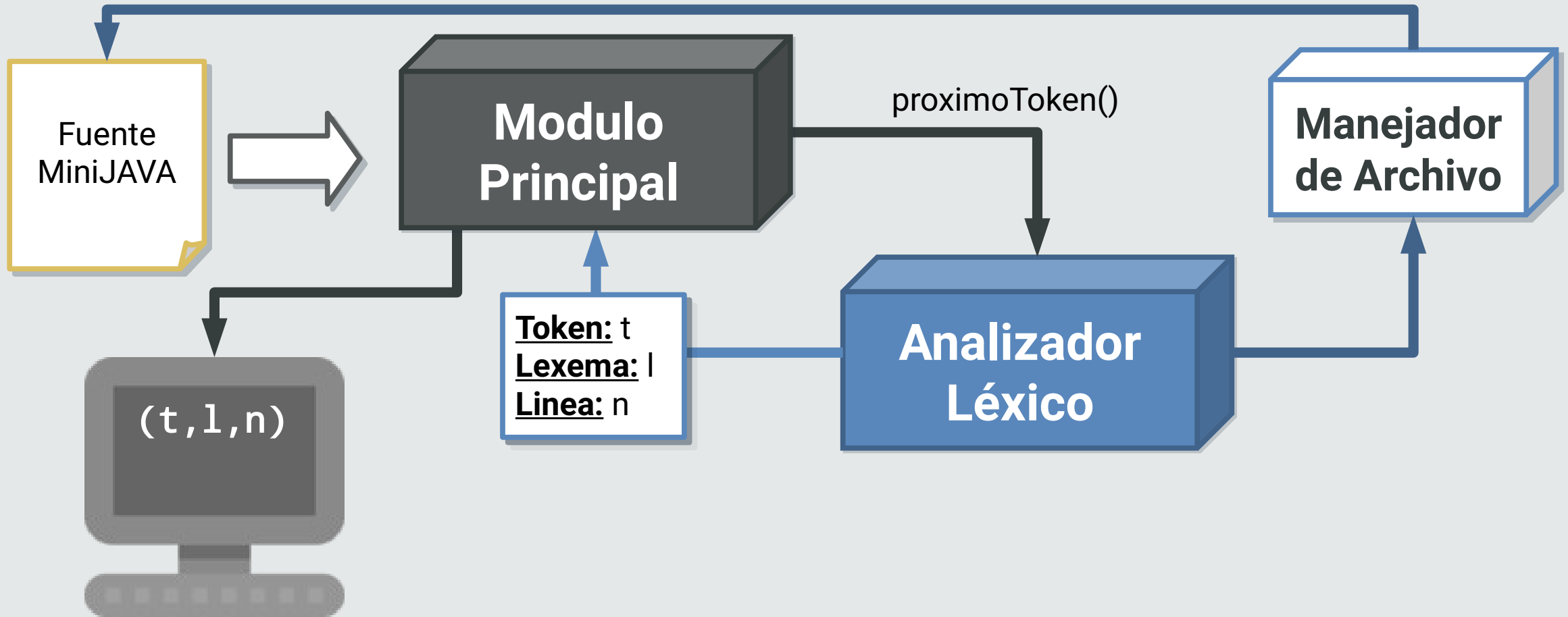
Funcionamiento e Interacción entre los Módulos



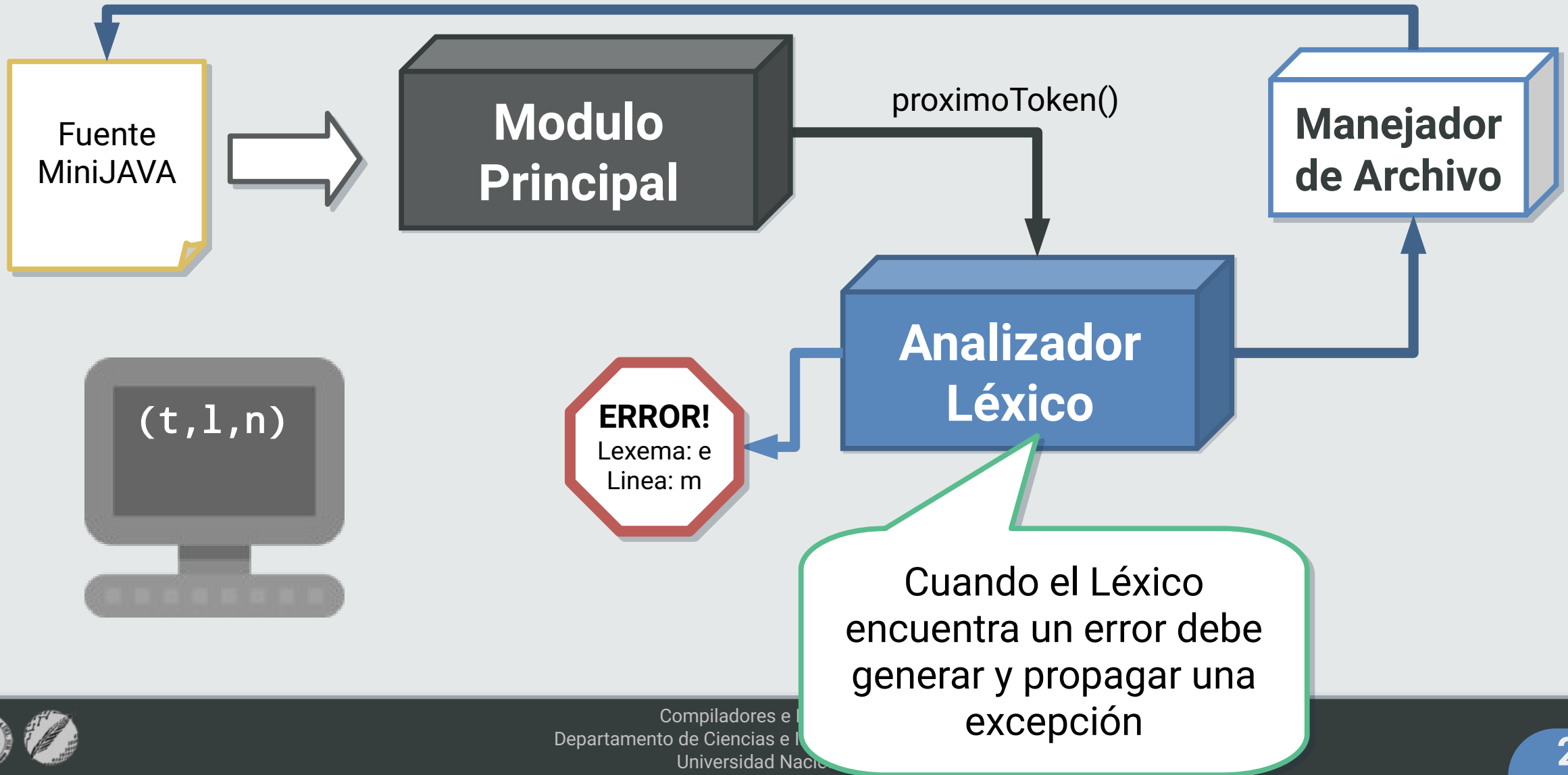
Funcionamiento e Interacción entre los Modulos



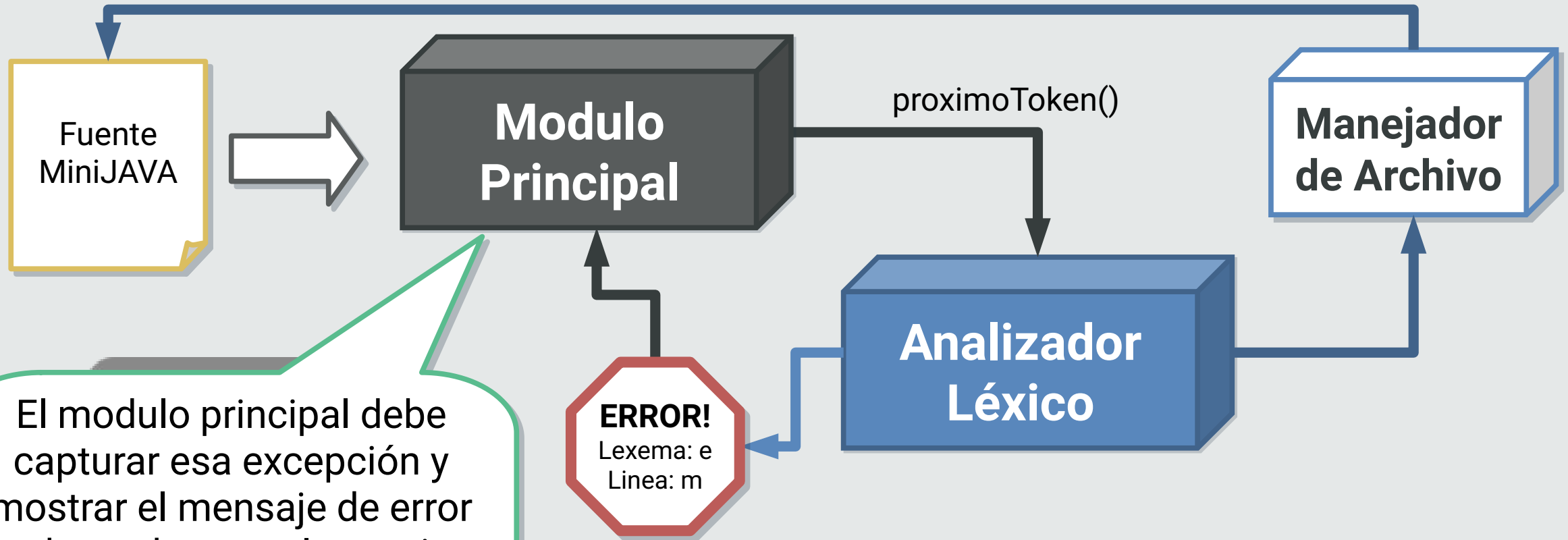
Funcionamiento e Interacción entre los Modulos



Funcionamiento e Interacción entre los Modulos



Funcionamiento e Interacción entre los Módulos



El modulo principal debe capturar esa excepción y mostrar el mensaje de error adecuado para el usuario por pantalla (y el código de error que indicaremos en las convenciones)

Convenciones y Consideraciones

Convenciones y Consideraciones

Fecha Limite de Entrega: 02/09/2022 a las 16hs

Forma de Entrega: En la actividad asociada a la entrega etapa 1 en Moodle, subiendo un zip con todo lo requerido

Convenciones y Consideraciones

La **interacción** de la aplicación entregada con el **usuario** deberá ser por **consola**



La entrega deberá consistir de: los archivos el **código fuente** (solo archivos **".java"**), un catálogo con los **casos de test** utilizados para la prueba del software y el **informe en formato pdf**



La **implementación** del **Analizador Léxico** deberá realizarse utilizando alguna de las **técnicas** presentadas en el **Tema 2** (sin utilizar herramientas de generación automática).



Convenciones y Consideraciones

Reportes de Errores Léxico

- Al detectar un error deberá reportarse y finalizar la ejecución
- Deberá reportarse un mensaje adecuado a la naturaleza del error indicando el lexema armado hasta el momento del error y en que numero de línea del fuente minijava se produjo.
- Al final del reporte deberá mostrarte un código de error con la siguiente estructura **[Error:Lexema|NroLinea]** donde Lexema y Nro de Linea son los reportados en el mensaje arriba mencionado.
- Ejemplo:



Fuente MiniJava

```
"hola"  
v1 + # chau  
if class}
```

El Reporte de
Error debería ser

```
Error Léxico en linea 2: # no es un  
símbolo valido
```

```
[Error:#|2]
```

Convenciones y Consideraciones

Mensaje de Análisis Exitoso y Muestra de Tokens Detectados

- El modulo principal deberá mostrar los tokens que recibe del Léxico de a uno por linea y usando la siguiente estructura (**Nombre Token, Lexema, Nro Linea**)
- Cuando no se detectan errores en el análisis el Modulo Principal deberá mostrar el código de éxito: **[SinErrores]**
- Por ejemplo

Fuente MiniJava

```
"hola"  
v1 + chau  
if class}
```

Debería mostrar

```
(String,hola,1)  
(idMV,v1,2)  
(op+,+,2)  
(idMV,chau,2)  
(pr_if,if,3)  
(pr_class,class,3)  
(llaveC,},3)
```

```
[SinErrores]
```



Convenciones y Consideraciones

Mensaje de Análisis Exitoso y Muestra de Tokens Detectados

- El modulo principal deberá mostrar los tokens que recibe del Léxico de a uno por linea y usando la siguiente estructura (**Nombre Token, Lexema, Nro Linea**)
- Cuando no se detectan errores en el análisis el Modulo Principal deberá mostrar el código de éxito: **[SinErrores]**
- Por ejemplo

Fuente MiniJava

```
"hola"  
v1 + chau  
if class}
```

Debería mostrar

```
(litString,hola,1)  
(idMV,v1,2)  
(op+,+,2)  
(idMV,chau,2)  
(pr_if,if,3)  
(pr_class,class,3)  
(llaveC,},3)
```

```
[SinErrores]
```

Toda lo que el compilador **muestra por pantalla** debe realizarse mediante la salida estándar (**System.out**)

Convenciones y Consideraciones

Invocación del Compilador

- Al ser compilado el software entregado poder invocarse de consola recibiendo como parámetro el archivo fuente MiniJava (debe aceptar cualquier tipo de extensión!)
- Por ejemplo: si al compilarlo lo llamamos Compilador y el fuente MiniJava se llama programa1.java, debería poder invocarse de la siguiente manera

```
java -jar Compilador.jar programa1.java
```



Convenciones y Consideraciones



No cumplir con estas convenciones llevará a incurrir en fallos y/o a la desaprobación de la entrega/reentrega

Convenciones y Consideraciones



No cumplir con estas convenciones llevará a incurrir en fallos y/o a la desaprobación de la entrega/reentrega

Las convenciones son esenciales para que **nosotros podamos corregir** bien la entrega...
por eso somos tan estrictos!

Si no la podemos corregir
va a estar **desaprobada!!!**

Herramientas de Test para la Evaluación

- En la tarea también les adjuntamos **herramientas de testing** de JUnit
- Estas herramientas las utilizamos en la cátedra como parte de la **evaluación** de los proyectos
- En el adjunto van a encontrar una **guía** de como agregarlas al proyecto (está armada respecto a **IntelliJ IDEA**)

Aclaración Importante: Las herramientas **no incluyen** los **casos de prueba** necesarios para **verificar adecuadamente** el compilador. El desarrollo de los casos es **responsabilidad del estudiante**. *Ademas, estas no son las únicas herramientas que utiliza la cátedra para la evaluación*

Informe

Informe

- El informe deberá indicar **como compilar el código fuente** entregado y como correrlo una vez compilado.
- Deben especificarse los **tokens reconocidos** por el analizador léxico, cada uno acompañado con la **expresión regular** que lo define.
- Incluir los **tipos de errores léxicos** que el analizador puede de detectar.
- Deberá incluirse la **documentación** usual, propia de un **proyecto de software** (decisiones de diseño, clases utilizadas, etc).
- Debe indicarse que **logros** se intentan alcanzar en la Etapa

Logros Disponibles para la Etapa 1

Logros Etapa 1



Entrega Anticipada Léxica

(no valido para la reentrega, ni etapas subsiguiente)

La entrega debe realizarse 48hs antes de la fecha limite



Imbatibilidad Léxica

(no valido para la reentrega, ni etapas subsiguiente)

El software entregado pasa correctamente toda la batería de prueba utilizada por la cátedra

Logros Etapa 1



Chars en Unicode

El Analizador Léxico permite literales carácter utilizando el el código unicode. Esto se hace mediante el escape \u seguido de un numero hexadecimal de 4 digitos. Por ejemplo '\u0A32' o '\uC90F'.

Logros Etapa 1

Reporte de Error Elegante

El compilador cuando reporta un error, además del numero de linea y la razón del error, muestra la linea en cuestión y apunta al lugar donde se produjo. Por ejemplo para el programa de la derecha debería mostrarse el mensaje:

Error Léxico en línea 2: # no es un símbolo valido

Detalle: v1 + # chau

^

[Error:#|2]

```
"hola"  
v1 + # chau  
if class}
```



Logros Etapa 1



Columnas

En los mensajes de error el compilador además del numero de linea indica el numero de columna donde se produjo el error. *Importante: esto afecta solo a los mensajes y no a los códigos de error!*



Multi-detección de Errores Lexicos

El compilador no finaliza la ejecución ante el primer error léxico (se recupera) y es capaz de reportar todos los errores léxico que tenga el programa fuente en una corrida