

## SECCIÓN 1: INTRODUCCIÓN (1-2 minutos)

[PANTALLA: Logo/Título del proyecto]

### NARRACIÓN:

"Hola, bienvenidos. Hoy les voy a presentar un compilador completo de Java a Java, desarrollado en Python. Este proyecto implementa todas las fases tradicionales de compilación: análisis léxico, sintáctico, semántico, generación de código intermedio, optimización y generación del código final."

### PUNTOS CLAVE A MENCIONAR:

- Es un compilador educativo pero funcional
- Traduce código Java a código Java optimizado
- Incluye interfaz gráfica y línea de comandos
- Implementa optimizaciones automáticas

[PANTALLA: Mostrar estructura de carpetas]

### NARRACIÓN:

"El proyecto está organizado en módulos bien definidos. Tenemos el paquete compiler con todos los componentes del compilador, el paquete utils con utilidades compartidas, una interfaz gráfica en el paquete gui, y una suite completa de pruebas y ejemplos."

---

## SECCIÓN 2: CARACTERÍSTICAS DEL COMPILADOR (2-3 minutos)

[PANTALLA: Tabla o diagrama de características]

### NARRACIÓN:

"Veamos qué puede compilar este sistema. Soporta los tipos básicos de Java: enteros, doubles, booleanos y arrays unidimensionales."

### MOSTRAR EN PANTALLA:

```
int edad = 25;
double altura = 1.75;
boolean activo = true;
int[] numeros = new int[5];
```

### NARRACIÓN:

"Implementa todos los operadores aritméticos, de comparación y lógicos. Aunque el operador módulo no está directamente soportado, se puede simular usando división y resta."

### MOSTRAR:

```
int suma = a + b;
boolean mayor = x > y;
boolean resultado = condicion1 && condicion2;
// Módulo simulado
int residuo = a - ((a / b) * b);
```

**NARRACIÓN:**

"En cuanto a estructuras de control, tenemos condicionales if-else, bucles while y for, y los statements de control break, continue y return."

**MOSTRAR:**

```
if (edad >= 18) {
    System.out.println("Mayor de edad");
}

for (int i = 0; i < 10; i = i + 1) {
    if (i == 5) {
        break;
    }
    System.out.println(i);
}
```

**NARRACIÓN:**

"Una característica importante es el soporte completo para métodos con parámetros, tipos de retorno, y lo más interesante: recursión. Aquí un ejemplo del clásico factorial."

**MOSTRAR:**

```
public static int factorial(int n) {
    if (n <= 1) {
        return 1;
    }
    return n * factorial(n - 1);
}
```

---

## SECCIÓN 3: ARQUITECTURA DEL COMPILADOR (3-4 minutos)

[PANTALLA: Diagrama de flujo del pipeline]

**NARRACIÓN:**

"Ahora veamos cómo funciona internamente. El compilador sigue una arquitectura de pipeline con siete fases principales."

**[MOSTRAR DIAGRAMA ANIMADO]:**

```
Código Fuente → LEXER → Tokens → PARSER → AST →  
SEMANTICS → AST Tipado → IR → OPTIMIZER → IR Optimizado →  
CODEGEN → Código Java → FORMATTER → Código Final
```

**NARRACIÓN:**

"Primera fase: El LEXER o analizador léxico. Lee el código fuente carácter por carácter y lo convierte en tokens. Por ejemplo, la expresión 'int x = 5;' se convierte en los tokens: INT\_T, IDENT, ASSIGN, INT\_LIT, SEMICOLON."

**[MOSTRAR EJEMPLO VISUAL DE TOKENIZACIÓN]****NARRACIÓN:**

"Segunda fase: El PARSER o analizador sintáctico. Toma esos tokens y construye un Árbol de Sintaxis Abstracta o AST. Este árbol representa la estructura jerárquica del programa."

**[MOSTRAR AST EN PANTALLA - PUEDE SER JSON O DIAGRAMA]****NARRACIÓN:**

"Tercera fase: Análisis semántico. Aquí verificamos que el programa tenga sentido: que las variables estén declaradas antes de usarse, que los tipos sean compatibles, que los métodos existan y tengan el número correcto de parámetros."

**MOSTRAR EJEMPLO:**

```
int x = 5;  
double y = x; // ✓ Conversión implícita permitida  
boolean z = x; // ✗ Error: tipos incompatibles
```

**NARRACIÓN:**

"Cuarta fase: Generación de representación intermedia. El AST se convierte en un código de tres direcciones con un grafo de flujo de control. Esto facilita las optimizaciones."

**MOSTRAR IR:**

```
t0 = 5 + 3  
x = t0
```

**NARRACIÓN:**

"Quinta fase: Optimización. Con el nivel O1 activado, el compilador realiza optimizaciones como constant folding, donde evalúa expresiones constantes en tiempo de compilación, eliminación de código muerto, y simplificación algebraica."

**MOSTRAR ANTES/DESPUÉS:**

```
// Antes
int x = 2 + 3;
int y = x * 1;

// Después de 01
int x = 5;
int y = x;
```

**NARRACIÓN:**

"Sexta fase: Generación de código Java. Aquí traducimos el AST optimizado de vuelta a código Java válido."

**NARRACIÓN:**

"Séptima y última fase: Formateo. El código se formatea con indentación correcta y espacios alrededor de operadores para mejor legibilidad."

---

## SECCIÓN 4: DEMOSTRACIÓN EN GUI (3-4 minutos)

[PANTALLA: Abrir la interfaz gráfica]

**NARRACIÓN:**

"Ahora veamos la interfaz gráfica en acción. Para iniciarla ejecutamos: python main.py --gui"

**[EJECUTAR COMANDO Y MOSTRAR GUI]****NARRACIÓN:**

"La interfaz tiene tres áreas principales: el editor de código a la izquierda con números de línea y syntax highlighting, el panel de salida a la derecha, y el panel de diagnósticos en la parte inferior."

**[NAVEGAR POR LA INTERFAZ]****NARRACIÓN:**

"Vamos a abrir uno de los ejemplos incluidos. Click en Abrir, seleccionamos ejemplo\_metodos.java"

**[ABRIR ARCHIVO]****NARRACIÓN:**

"Fíjense cómo el editor colorea automáticamente las palabras clave en azul, los strings en rojo, y los comentarios en verde. Esto es el syntax highlighting automático."

**[SEÑALAR CÓDIGO COLOREADO]****NARRACIÓN:**

"Ahora compilamos. Click en Compilar."

#### [COMPILAR]

##### NARRACIÓN:

"En el panel derecho vemos el código Java generado, formateado y listo para ejecutar. La barra de estado muestra que la compilación fue exitosa, sin errores ni warnings."

#### [MOSTRAR PANEL DE SALIDA Y BARRA DE ESTADO]

##### NARRACIÓN:

"Ahora vamos a ejecutarlo directamente desde la GUI. Click en Ejecutar."

#### [EJECUTAR]

##### NARRACIÓN:

"El sistema compila el código con javac y lo ejecuta automáticamente. Aquí vemos la salida del programa."

#### [MOSTRAR SALIDA]

##### NARRACIÓN:

"Ahora introduzcamos un error intencional para ver el sistema de diagnósticos."

#### [EDITAR CÓDIGO PARA CREAR ERROR - por ejemplo: usar variable no declarada]

```
public static void main(String[] args) {  
    int x = 5;  
    System.out.println(y); // y no declarada  
}
```

#### [COMPILAR]

##### NARRACIÓN:

"Al compilar, vemos que aparece un error en rojo en el panel de diagnósticos: 'Variable y no declarada'. Si hacemos click en el error..."

#### [CLICK EN ERROR]

##### NARRACIÓN:

"El editor automáticamente salta a la línea del error y la resalta. Esto facilita mucho la depuración."

#### [MOSTRAR LÍNEA RESALTADA]

##### NARRACIÓN:

"El panel de diagnósticos tiene tres pestañas: Todos, Errores, y Warnings. Esto permite filtrar rápidamente solo los errores o solo las advertencias."

**[MOSTRAR TABS]**

---

**SECCIÓN 5: DEMOSTRACIÓN CLI (2 minutos)**

[PANTALLA: Terminal]

**NARRACIÓN:**

"El compilador también funciona desde línea de comandos. Veamos algunos ejemplos."

**NARRACIÓN:**

"Compilación básica. Especificamos archivo de entrada y archivo de salida con -o."

**[EJECUTAR]:**

```
python main.py examples/ejemplo_basico.java -o output/Basico.java
```

**MOSTRAR SALIDA EN TERMINAL****NARRACIÓN:**

"Para compilar con optimizaciones, agregamos --O1."

**[EJECUTAR]:**

```
python main.py examples/ejemplo_matematicas.java --O1 -o output/Optimizado.java
```

**NARRACIÓN:**

"Para ver la estructura interna, podemos usar --ast para ver el árbol de sintaxis, o --ir para ver la representación intermedia."

**[EJECUTAR Y MOSTRAR]:**

```
python main.py examples/ejemplo_basico.java --ast
```

**MOSTRAR JSON DEL AST****NARRACIÓN:**

"Y finalmente, compilamos el código generado con javac y lo ejecutamos con java."

**[EJECUTAR]:**

```
cd output
javac Basico.java
java EjemploBasico
```

## **MOSTRAR SALIDA DEL PROGRAMA**

---

### SECCIÓN 6: CASOS DE USO AVANZADOS (2 minutos)

[PANTALLA: Editor o terminal]

#### **NARRACIÓN:**

"Veamos algunos ejemplos más complejos incluidos en el proyecto. Aquí tenemos un programa que implementa el algoritmo de Fibonacci de forma recursiva."

**[ABRIR ejemplo\_completo.java]**

#### **NARRACIÓN:**

"Este ejemplo incluye búsqueda en arrays, ordenamiento burbuja, y contador de números pares. Es un buen ejemplo de cómo combinar todas las características del compilador."

**[MOSTRAR CÓDIGO BREVEMENTE]**

#### **NARRACIÓN:**

"Aquí otro ejemplo: implementación de funciones matemáticas como cálculo de potencias, máximo común divisor usando el algoritmo de Euclides, y verificación de números primos."

**[ABRIR ejemplo\_matematicas.java]**

#### **NARRACIÓN:**

"Lo interesante de este ejemplo es cómo simula el operador módulo, que no está directamente soportado. Usa la fórmula:  $a \text{ menos } a \text{ dividido } b \text{ por } b$ ."

**[SEÑALAR CÓDIGO DEL MÓDULO SIMULADO]**

---

### SECCIÓN 7: ASPECTOS TÉCNICOS (1-2 minutos)

[PANTALLA: Código del compilador o diagramas]

#### **NARRACIÓN:**

"Hablemos brevemente de la implementación técnica. El compilador está escrito en Python usando type hints para mejor mantenibilidad. Utiliza dataclasses para definir los nodos del AST y las instrucciones IR."

**[MOSTRAR ast\_nodes.py O CÓDIGO RELEVANTE]**

#### **NARRACIÓN:**

"El parser es un recursive descent parser con precedencia de operadores. Cada función de parsing corresponde a una regla de la gramática."

**[MOSTRAR FRAGMENTO DE parser.py]**

**NARRACIÓN:**

"El análisis semántico usa un patrón visitor para recorrer el AST, manteniendo una tabla de símbolos organizada por niveles de scope."

**[MOSTRAR FRAGMENTO DE semantics.py]**

**NARRACIÓN:**

"El proyecto incluye una suite completa de pruebas unitarias que cubren todas las fases del compilador, garantizando que cada componente funcione correctamente."

**[MOSTRAR CARPETA tests/]**

---

## SECCIÓN 8: CONCLUSIÓN Y PRÓXIMOS PASOS (1 minuto)

[PANTALLA: Resumen o pantalla final]

**NARRACIÓN:**

"Para resumir: hemos creado un compilador completo de Java a Java que implementa todas las fases tradicionales de compilación. Soporta tipos básicos, estructuras de control, métodos con recursión, arrays, y optimizaciones automáticas."

**PUNTOS CLAVE:**

- ✓ Pipeline completo de compilación
- ✓ Análisis léxico, sintáctico y semántico
- ✓ Optimizaciones de código
- ✓ Interfaz gráfica intuitiva
- ✓ Sistema de diagnósticos detallado
- ✓ Suite de pruebas completa

**NARRACIÓN:**

"El proyecto está completamente documentado con documentación técnica, manual de usuario, y ejemplos listos para ejecutar. Todo el código está disponible en GitHub bajo licencia MIT."

**NARRACIÓN:**

"Posibles mejoras futuras incluyen: soporte para Strings como tipo completo, arrays multidimensionales, operador módulo nativo, switch-case, y más optimizaciones avanzadas como optimización de loops y eliminación de código inalcanzable."

**NARRACIÓN:**

"El repositorio está abierto a contribuciones. Si encuentran bugs o quieren agregar características, son bienvenidos a crear issues o pull requests."

### [MOSTRAR:]

GitHub: [github.com/Yahir750/Compilador](https://github.com/Yahir750/Compilador)  
Documentación completa incluida  
Licencia: MIT

### NARRACIÓN:

"Gracias por ver esta presentación. Si tienen preguntas o comentarios, déjenlos en los comentarios o en el repositorio de GitHub. ¡Hasta luego!"

---

## NOTAS PARA LA GRABACIÓN

### Preparación Previa

#### Archivos a tener listos:

1. GUI ya abierta en una ventana
2. Terminal en la carpeta del proyecto
3. Editor de texto con ejemplos abiertos
4. Diagramas/slides para secciones teóricas

#### Configuración de Pantalla:

- Resolución clara (1920x1080 recomendado)
- Fuente grande en terminal (tamaño 14-16)
- GUI maximizada
- Cerrar aplicaciones innecesarias

### Durante la Grabación

#### Ritmo:

- Hablar claro y pausado
- Dar tiempo para que se vean los resultados en pantalla
- No apresurarse en las demos

#### Transiciones:

- Usar frases de transición entre secciones
- "Ahora veamos...", "Pasemos a...", "Observe cómo..."

#### Enfasis:

- Resaltar puntos clave
- Repetir conceptos importantes

- Usar ejemplos visuales

## Post-Producción

### Edición:

- Agregar títulos/subtítulos para cada sección
- Resaltar áreas importantes con zoom o recuadros
- Acelerar partes repetitivas si es necesario

### Efectos:

- Flecha/cursor destacado en demos
- Transiciones suaves entre secciones
- Música de fondo suave (opcional)

### Subtítulos:

- Agregar subtítulos para términos técnicos
- Traducir conceptos complejos visualmente

---

## GUIÓN ALTERNATIVO (VERSIÓN CORTA - 5 minutos)

Para video más breve, usar esta estructura:

**Minuto 1:** Introducción + características principales **Minuto 2:** Demo rápida GUI (compilar y ejecutar) **Minuto 3:** Mostrar arquitectura con diagrama **Minuto 4:** Demo CLI + ejemplos avanzados **Minuto 5:** Conclusión + recursos

---

## MATERIAL DE APOYO VISUAL

### Diapositivas Recomendadas

#### Slide 1: Título

- Logo del compilador
- "Compilador Java → Java"
- "Pipeline completo de compilación"

#### Slide 2: Características

- Lista de tipos soportados
- Operadores
- Estructuras de control

#### Slide 3: Arquitectura

- Diagrama de flujo del pipeline
- 7 fases claramente marcadas

**Slide 4: Optimizaciones**

- Antes/Después ejemplos
- Constant folding, propagation, DCE

**Slide 5: Recursos**

- GitHub URL
- Documentación
- Contacto

---

**Versión:** 1.0 **Duración sugerida:** 10-15 minutos **Formato:** Tutorial técnico **Nivel:** Intermedio-Avanzado