DOCUMENTO DE CASOS DE PRUEBAS

Equipo 2

14 de mayo de 2025

Realizó Mantenimiento:

Canul Ordoñez, Josué Israel

Garcilazo Cuevas, Mónica

Leo Fernández, José Carlos

Pool Flores, Endrick Alfredo

Rodríguez Coral, Samuel David

1. CONTROL DE DOCUMENTACIÓN

1.1 Control de Configuración

Título:	Casos de pruebas
Referencia:	N/A
Autor:	Adjany Armenta, Paulina Perera, Mónica Garcilazp
Fecha:	25 de febrero de 2025

1.2 Histórico de versiones

Versión	Fecha	Estado	Responsable	Nombre del archivo
1.0.1	24 – 02 – 2025	B	Adjany Armenta	Documento_casos_de_pruebas_1.0.1
1.0.2	25 – 02 – 2025	B	Paulina Perera	Documento_casos_de_pruebas_1.0.2
1.0.3	26 – 02 – 2025	B	Paulina Perera	Documento_casos_de_pruebas_1.0.3
1.0.4	26 – 02 – 2025	B	Paulina Perera	Documento_casos_de_pruebas_1.0.4
1.0.5	26 – 02 – 2025	A	Paulina Perera	Documento_casos_de_pruebas_1.0.5
2.0.0	03 – 03 – 2025	A	Jean Buenfil	Documento_casos_de_pruebas_2.0.0
2.1.0	10 – 05 – 2025	A	Mónica Garcilazo	Documento_casos_de_pruebas_2.1.0
3.0.0	14 – 05 – 2025	A	Mónica Garcilazo	Documento_casos_de_pruebas_3.0.0

1.3 Histórico de cambios

Versión	Fecha	Cambios
1.0.1	23 – 02 – 2025	Creación del archivo inicial
1.0.2	24 – 02 – 2025	Se añaden casos de prueba CP001, CP002, CP003, CP004, CP005, CP006
1.0.3	25 – 02 – 2025	Se añaden casos de prueba CP007, CP008, CP009
1.0.4	25 – 02 – 2025	Se añaden casos de prueba CP010, CP011
1.0.5	26 – 02 – 2025	Se añaden los resultados de las pruebas para cada paso, se corrigen detalles de redacción y de formato
2.0.0	03 – 04 – 2025	Se modifica los campos necesarios para cada caso de prueba, se modifican los casos de prueba del CP001 al CP011, se añaden los casos de prueba del CP012 al CP043, se añaden los resultados de las pruebas
2.1.0	10-05-2025	Se añaden casos de prueba CP042 al CP058
3.0.0	14-05-2025	Se añaden los resultados de las pruebas para cada paso, se corrigen detalles de redacción y de formato

CONTENIDO

2. CASOS DE PRUEBA

2.1 Caso de prueba 001: Procesar línea: Línea de comentario

Identificador caso de prueba: CP001

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Conteo de métodos

***Objetivo**:*

Probar que el sistema no cuenta las líneas de comentario.

Descripción:

Se le pasa una línea de comentario al sistema

Criterios de éxito:

- El valor de verdad sobre si la lista de métodos permanece vacía es igual a Verdadero.

Criterios de falla:

- El valor de verdad sobre si la lista de métodos permanece vacía es diferente a Verdadero.

Código utilizado:

```
@Test
@DisplayName("Procesar línea: línea de comentario")
void testProcessLine_CommentLine() throws InvalidFormatException {
  String line = "// Esto es un comentario";

  codeAnalyzer.processLine(line);

  assertTrue(structCounter.getClasses().isEmpty());
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

El conteo de clases no aumenta al procesar una línea de comentario.

2.2 Caso de prueba 002: Procesar línea: Línea vacía

Identificador caso de prueba: CP002

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Conteo de métodos

Objetivo*:*

Probar que el sistema no cuenta las líneas vacías.

Descripción:

Se le pasa una línea vacía al sistema

Criterios de éxito:

- El valor de verdad sobre si la lista de clases permanece vacía es igual a Verdadero.

Criterios de falla:

- El valor de verdad sobre si la lista de clases permanece vacía es diferente a Verdadero.

Código utilizado:

```
@Test
@DisplayName("Procesar línea: línea vacía")
void testProcessLine_EmptyLine() throws InvalidFormatException {
  String line = "    ";

  codeAnalyzer.processLine(line);

  assertTrue(structCounter.getClasses().isEmpty());
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

El conteo de clases no aumenta al procesar una línea vacía

2.3 Caso de prueba 003: Preprocesar línea: ignorar línea de comentario

Identificador caso de prueba: CP003

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Preprocesamiento de líneas

***Objetivo**:*

Probar que el sistema no preprocesa las líneas de comentario.

Descripción:

Se intenta preprocesar una línea de comentario

Criterios de éxito:

- El resultado al preprocesar una línea de comentario es Nulo.

Criterios de falla:

- El resultado al preprocesar una línea de comentario es diferente a Nulo.

Código utilizado:

```
@Test
@DisplayName("Preprocesar línea: ignorar línea de comentario")
void testPreprocessLine_IgnoreCommentLine() {
  String commentLine = "// Esto es un comentario";

  String result = codeAnalyzer.preprocessLine(commentLine);

  assertNull(result);
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

No se preprocesan las líneas de comentario

2.4 Caso de prueba 004: Preprocesar línea: ignorar línea vacía

Identificador caso de prueba: CP004

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Preprocesamiento de líneas

***Objetivo**:*

Probar que el sistema no preprocesa las líneas vacías.

Descripción:

Se intenta preprocesar una línea vacía

Criterios de éxito:

- El resultado al preprocesar una línea vacía es Nulo.

Criterios de falla:

- El resultado al preprocesar una línea vacía es diferente a Nulo.

Código utilizado:

```
@Test
@DisplayName("Preprocesar línea: ignorar línea vacía")
void testPreprocessLine_IgnoreEmptyLine() {
  String emptyLine = "    ";

  String result = codeAnalyzer.preprocessLine(emptyLine);

  assertNull(result);
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

No se preprocesan las líneas vacías

2.5 Caso de prueba 005: Preprocesar línea: línea válida

Identificador caso de prueba: CP005

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Preprocesamiento de líneas

***Objetivo**:*

Probar que el sistema preprocesa correctamente las líneas válidas.

Descripción:

Se intenta preprocesar una línea válida para comparar si el resultado es el correcto

Criterios de éxito:

- El resultado al preprocesar una línea válida es igual a la línea de entrada.

Criterios de falla:

- El resultado al preprocesar una línea válida es diferente a la línea de entrada.

Código utilizado:

```java
@Test
@DisplayName("Preprocesar línea: línea válida")
void testPreprocessLine_ValidLine() {
  String validLine = "public class TestClass {";

  String result = codeAnalyzer.preprocessLine(validLine);

  assertEquals("public class TestClass {", result);
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

"True"

Resultados de la prueba:

Las líneas válidas se procesan correctamente

2.6 Caso de prueba 006: Obtener contador

Identificador caso de prueba: CP006

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Obtener contador

Objetivo*:*

Validar que el analizador de código devuelve correctamente el contador de estructuras.

Descripción:

Se obtiene el contador de estructuras y se compara con la instancia original.

Criterios de éxito:

- El contador devuelto es igual al contador instanciado

Criterios de falla:

- El contador devuelto es diferente al contador instanciado.

Código utilizado:

```
@Test
@DisplayName("Obtener contador")
void testGetCounter() {
  assertEquals(structCounter, codeAnalyzer.getCounter());
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

"True"

Resultados de la prueba:

El contador se obtiene correctamente

2.7 Caso de prueba 007: Contar las clases anidadas

Identificador de caso de prueba: CP007

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Conteo de clases anidadas

Objetivo*:*

Verificar que el sistema cuenta correctamente las clases anidadas dentro de una estructura de código.

Descripción:

Se procesan líneas de código que contienen clases anidadas y se valida que el número total de clases detectadas sea correcto.

Criterios de éxito:

- El número de clases contadas es igual al número de declaraciones de clase en el código de entrada.

Criterios de falla:

- El número de clases contadas es diferente al número de declaraciones de clase en el código de entrada.

Código utilizado:

```
@Test
@DisplayName("Contar las clases anidadas")
void testCountNestedClasses() throws InvalidFormatException {
  String line1 = "public class OuterClass {";
  String line2 = "public class InnerClass {";
  String line3 = "public class InnerMostClass {";

  codeAnalyzer.processLine(line1);
  codeAnalyzer.processLine(line2);
  codeAnalyzer.processLine(line3);

  assertEquals(3, structCounter.getClassesCount());
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Se cuentan correctamente las clases anidadas.

2.8 Caso de prueba 008: Contar métodos en una clase

Identificador caso de prueba: CP008

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Conteo de métodos en una clase

***Objetivo**:*

Verificar que el sistema cuenta correctamente la cantidad de métodos en una clase.

Descripción:

Se procesa una clase con un método y se compara la cantidad de métodos detectados con el valor esperado.

Criterios de éxito:

- El número de métodos contados en la clase es igual al número de métodos en el código de entrada.

Criterios de falla:

- El número de métodos contados en la clase es diferente al número de métodos en el código de entrada.

Código utilizado:

```java
@Test
@DisplayName("Contar métodos en una clase")
void testCountMethodsInClass() throws InvalidFormatException {
  String line1 = "public class TestClass {";
  String line2 = "public void testMethod() {";
  String line3 = "return; } }";

  codeAnalyzer.processLine(line1);
  codeAnalyzer.processLine(line2);
  codeAnalyzer.processLine(line3);

  assertEquals(1, structCounter.getClasses().get(0).getMethodsAmount());
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Se cuenta correctamente la cantidad de métodos en una clase.

2.9 Caso de prueba 009: Contar líneas de código en una clase

Identificador de caso de prueba: CP009

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Conteo de líneas de código en una clase

Objetivo*:*

Verificar que el sistema cuenta correctamente la cantidad de líneas de código dentro de una clase.

Descripción:

Se procesa una clase con varias líneas de código y se compara la cantidad de líneas contadas con el valor esperado.

Criterios de éxito:

- El número de líneas de código contadas en la clase es igual al número de líneas de código en el código de entrada.

Criterios de falla:

- El número de líneas de código contadas en la clase es diferente al número de líneas de código en el código de entrada.

Código utilizado:

```
@Test
@DisplayName("Contar líneas de código en una clase")
void testCountLinesOfCodeInClass() throws InvalidFormatException {
  String line1 = "public class TestClass {";
  String line2 = "int a = 0;";
  String line3 = "return; } }";

  codeAnalyzer.processLine(line1);
  codeAnalyzer.processLine(line2);
  codeAnalyzer.processLine(line3);

  assertEquals(3, structCounter.getClasses().get(0).getLinesOfCode());
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Se cuentan correctamente las líneas de código en una clase.

2.10 Caso de prueba 010: Cargar el contenido de un archivo

Identificador de caso de prueba: CP010

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Carga de contenido de un archivo Java

Objetivo*:*

Verificar que el sistema carga correctamente el contenido de un archivo Java en memoria.

Descripción:

Se crea un archivo temporal con código fuente en Java y se carga su contenido. Luego, se valida que la lista de líneas no sea nula, no esté vacía y contenga la cantidad esperada de líneas.

Criterios de éxito:

- El contenido del archivo no es nulo
- El contenido del archivo no está vacío
- El número de líneas cargadas es igual a la cantidad esperada

Criterios de falla:

- El contenido del archivo es nulo
- El contenido del archivo está vacio
- El número de líneas cargadas es diferente a la cantidad esperada

Código utilizado:

```java
@Test
@DisplayName("Debe de cargar el contenido correctamente")
void testLoadFileContent() throws FileNotFoundException {
  JavaFile javaFile = new JavaFile(tempFile);
  List<String> content = javaFile.getContent();

  assertNotNull(content);
  assertFalse(content.isEmpty());
  assertEquals(12, content.size());
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

"True"

Resultados de la prueba:

El contenido del archivo se carga correctamente.

2.11 Caso de prueba 011: Eliminar comentarios en bloque y de línea

Identificador de caso de prueba: CP011

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Eliminación de comentarios

Objetivo*:*

Verificar que el sistema elimina correctamente los comentarios en bloque y de línea.

Descripción:

Se crea un archivo temporal con código fuente en Java que contiene comentarios en bloque y de línea. Luego, se procesa y se valida que los comentarios hayan sido eliminados.

Criterios de éxito:

- Ninguna línea de la lista de contenido contiene comentarios en línea
- Ninguna línea de la lista de contenido contiene comentarios en bloque

Criterios de falla:

- Alguna línea aún contiene comentarios en línea o en bloque después del procesamiento

Código utilizado:

```
@Test
@DisplayName("Debe de eliminar comentarios en bloque y de linea correctamente")
void testRemoveComments() throws FileNotFoundException {
  JavaFile javaFile = new JavaFile(this.tempFile).removeComments();
  List<String> content = javaFile.getContent();

  assertFalse(content.contains("// Esto es un comentario en línea"));
  assertFalse(content.contains("/**"));
  assertFalse(content.contains("*"));
  assertFalse(content.contains("* Comentario de bloque"));
  assertFalse(content.contains("* que ocupa varias líneas"));
  assertFalse(content.contains("*/"));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Los comentarios en línea y en bloque fueron eliminados correctamente.

2.12 Caso de prueba 012: Eliminar líneas en blanco

Identificador de caso de prueba: CP012

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Eliminación de líneas en blanco

***Objetivo**:*

Verificar que el sistema elimina correctamente las líneas en blanco.

Descripción:

Se crea un archivo temporal con código fuente en Java que contiene líneas en blanco. Luego, se procesa y se valida que los comentarios hayan sido eliminados.

Criterios de éxito:

- Ninguna línea en la lista de contenido está vacía.

Criterios de falla:

- Alguna línea en la lista de contenido sigue estando vacía después del procesamiento.

Código utilizado:

```java
@Test
@DisplayName("Debe de eliminar lineas en blanco correctamente")
void testRemoveBlankLines() throws FileNotFoundException {
  JavaFile javaFile = new JavaFile(this.tempFile).removeBlankLines();
  List<String> content = javaFile.getContent();

  assertFalse(content.contains(""));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Las líneas en blanco fueron eliminadas correctamente.

2.13 Caso de prueba 013: Eliminar comentarios y líneas en blanco

Identificador de caso de prueba: CP013

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Eliminación de comentarios y líneas en blanco

***Objetivo**:*

Verificar que el sistema elimina correctamente los comentarios y las líneas en blanco.

Descripción:

Se crea un archivo temporal con código fuente en Java que contiene comentarios y líneas en blanco. Luego, se procesa y se valida que los comentarios y líneas en blanco hayan sido eliminados.

Criterios de éxito:

- Ninguna línea de la lista de contenido contiene comentarios.
- Ninguna línea de la lista de contenido está vacía.
- La cantidad final de líneas corresponde con la cantidad esperada tras la eliminación de comentarios y líneas en blanco.

Criterios de falla:

- Alguna línea en la lista de contenido sigue estando vacía después del procesamiento.
- Alguna línea en la lista de contenido aún contiene comentarios después del procesamiento.
- La cantidad final de líneas es diferente a la esperada

Código utilizado:

```java
@Test
@DisplayName("Debe de remover lineas en blanco y comentarios")
void testRemoveCommentsAndBlankLines() throws FileNotFoundException {
  JavaFile javaFile = new JavaFile(tempFile).removeComments().removeBlankLines();
  List<String> content = javaFile.getContent();

  assertFalse(content.contains("// Esto es un comentario en línea"));
  assertFalse(content.contains("/* Comentario de bloque"));
  assertFalse(content.contains("que ocupa varias líneas */"));
  assertFalse(content.contains(""));

  assertEquals(5, content.size());
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Los comentarios y líneas en blanco fueron eliminados correctamente.

2.14 Caso de prueba 014: Validar declaración de import

Identificador de caso de prueba: CP014

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaraciones import

Objetivo*:*

Verificar que el sistema acepte correctamente las declaraciones de import válidas.

Descripción:

Se validan diferentes declaraciones de import

Criterios de éxito:

- Las import son válidos

Criterios de falla:

- Los import no son válidos

Código utilizado:

```
@Test
@DisplayName("Debe de aceptar la declaracion de import")
void testValidImport() throws InvalidFormatException {
  ImportValidator validator = new ImportValidator();
  assertTrue(validator.isValid("import java.util.List;"));
  assertTrue(validator.isValid("import static java.util.List;"));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Las declaraciones de import válidas fueron aceptadas correctamente.

2.15 Caso de prueba 015: Lanzar excepción al detectar comodín en import

Identificador de caso de prueba: CP015

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaraciones import con comodines

***Objetivo**:*

Verificar que el sistema rechace declaraciones de import que utilicen comodines (*) y lance la excepción correspondiente.

Descripción:

Se validan diferentes declaraciones de import con comodín (*) y se espera que el sistema lance una excepción.

Criterios de éxito:

- Se lanza la excepción cuando se pasa un import con comodín

Criterios de falla:

- No se lanza la excepción cuando se pasa un import con comodín

Código utilizado:

```
@Test
@DisplayName("Debe de lanzar la excepion al detectar comodin en import")
void importWithWildcardThrowsException() {
  ImportValidator validator = new ImportValidator();
  assertThrows(InvalidFormatException.class, () -> validator.isValid("import java.util.*;"));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Se lanzó correctamente la excepción al detectar un comodín en import.

2.16 Caso de prueba 016: Lanzar excepción al detectar comodín en import static

Identificador de caso de prueba: CP016

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaraciones import static con comodines

***Objetivo**:*

Verificar que el sistema rechace declaraciones de import static que utilicen comodines (*) y lance la excepción correspondiente.

Descripción:

Se validan diferentes declaraciones de import static con comodín (*) y se espera que el sistema lance una excepción.

Criterios de éxito:

- Se lanza la excepción cuando se pasa un import static con comodín

Criterios de falla:

- No se lanza la excepción cuando se pasa un import static con comodín

Código utilizado:

```
@Test
@DisplayName("Debe de lanzar la excepion al detectar comodin en import static")
void testStaticImportWithWildcardThrowsException() {
  ImportValidator validator = new ImportValidator();
  assertThrows(
      InvalidFormatException.class, () -> validator.isValid("import static java.lang.Math.*;"));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

"True"

Resultados de la prueba:

Se lanzó correctamente la excepción al detectar un comodín en import static.

2.17 Caso de prueba 017: Validar anotación @Override

Identificador de caso de prueba: CP017

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de anotaciones

Objetivo*:*

Verificar que el sistema acepte correctamente una anotación válida en una sola línea, en este caso, @Override.

Descripción:

Se valida la declaración @override

Criterios de éxito:

- La validación de @Override retorna Verdadero

Criterios de falla:

- La validación de @Override retorna Falso o lanza una excepción

Código utilizado:

```java
@Test
@DisplayName("Debe devolver true para una anotación válida en una sola línea: @Override")
void testValidOverrideAnnotation() throws InvalidFormatException {
  assertTrue(validator.isValid("@Override"));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

"True"

Resultados de la prueba:

La anotación @Override fue validada correctamente.

2.18 Caso de prueba 018: Validar anotación @Rule con parámetros

Identificador de caso de prueba: CP018

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de anotaciones

Objetivo*:*

Verificar que el sistema acepte correctamente una anotación @Rule con parámetros en una sola línea

Descripción:

Se valida la declaración @Rule con parámetros

Criterios de éxito:

- La validación de @Rule retorna Verdadero

Criterios de falla:

- La validación de @Rule retorna Falso o lanza una excepción

Código utilizado:

```
@Test
@DisplayName(
    "Debe devolver true para una anotación válida en una sola línea: @Rule(expected ="
        + " IllegalArgumentException.class)")
void testValidTestAnnotation() throws InvalidFormatException {
  assertTrue(validator.isValid("@Rule(expected = IllegalArgumentException.class)"));
}
```

Resultados de ejecución

Fecha de ejecución: 31 de marzo de 2025

Salida:

“True”

Resultados de la prueba:

La anotación @Rule fue validada correctamente.

2.19 Caso de prueba 019: Lanzar excepción para múltiples anotaciones en la misma línea

Identificador de caso de prueba: CP019

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de anotaciones

Objetivo*:*

Verificar que el sistema rechace múltiples anotaciones en la misma línea y lance una excepción.

Descripción:

Se valida una línea que contiene múltiples anotaciones. Se espera que el sistema lance una excepción.

Criterios de éxito:

- La validación de la línea con múltiples anotaciones lanza una excepción

Criterios de falla:

- La validación de la línea con múltiples anotaciones retorna Verdadero

Código utilizado:

```
@Test
@DisplayName("Debe lanzar excepción para múltiples anotaciones en la misma línea")
void testMultipleAnnotationsThrowsException() {
  String line = "@Override @Test @Something";
  assertThrows(InvalidFormatException.class, () -> validator.isValid(line));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Se lanzó correctamente la excepción al detectar múltiples anotaciones en la misma línea.

2.20 Caso de prueba 020: Lanzar excepción para anotación seguida de código en la misma línea

Identificador de caso de prueba: CP020

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de anotaciones

Objetivo*:*

Verificar que el sistema rechace una anotación seguida de código en la misma línea y lance una excepción.

Descripción:

Se valida una línea que contiene una anotación seguida de código. Se espera que el sistema lance una excepción.

Criterios de éxito:

- La validación de la línea con una anotación seguida de código lanza una excepción

Criterios de falla:

- La validación de la línea con una anotación seguida de código retorna Verdadero

Código utilizado:

```
@Test
@DisplayName("Debe lanzar excepción para anotación seguida de código en la misma línea")
void testAnnotationWithCodeThrowsException() {
  String line = "@Override public void doSomething(){";
  assertThrows(InvalidFormatException.class, () -> validator.isValid(line));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Se lanzó correctamente la excepción al detectar una anotación seguida de código en la misma línea.

2.21 Caso de prueba 021: Lanzar excepción si la anotación está mal formada

Identificador de caso de prueba: CP021

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de anotaciones

***Objetivo**:*

Verificar que el sistema rechace anotaciones mal formadas y lance una excepción.

Descripción:

Se valida líneas con anotaciones mal formadas. Se espera que el sistema lance una excepción.

Criterios de éxito:

- La validación de las líneas con anotaciones mal formadas lanza una excepción

Criterios de falla:

- La validación de las líneas con anotaciones mal formadas retorna Verdadero

Código utilizado:

```
@Test
@DisplayName("Debe lanzar excepción si la anotación es mal formada")
void testMalformedAnnotationThrowsException() {
  assertThrows(InvalidFormatException.class, () -> validator.isValid("@"));
  assertThrows(InvalidFormatException.class, () -> validator.isValid("@123Invalid"));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Se lanzó correctamente la excepción al detectar una anotación mal formada.

2.22 Caso de prueba 022: Validar declaración única de variable

Identificador de caso de prueba: CP022

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaraciones de variables

***Objetivo**:*

Verificar que el sistema acepte correctamente una declaración única de variable en una sola línea.

Descripción:

Se valida una línea con declaración

Criterios de éxito:

- La validación de la línea con declaración retorna Verdadero

Criterios de falla:

- La validación de la línea con declaración retorna Falso o lanza una excepción

Código utilizado:

```
@Test
void testValidSingleDeclaration() throws InvalidFormatException {
  SingleDeclarationValidator validator = new SingleDeclarationValidator();
  assertTrue(validator.isValid("int x = 10;"));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

"True"

Resultados de la prueba:

La declaración única de variable fue validada correctamente.

2.23 Caso de prueba 023: Lanzar excepción al detectar inicialización de múltiples variables con punto y coma

Identificador de caso de prueba: CP023

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaraciones de variables

Objetivo*:*

Verificar que el sistema rechace correctamente declaraciones múltiples de variables separadas por punto y coma y lance una excepción

Descripción:

Se valida que al analizar líneas con más de una declaración se lance una excepción

Criterios de éxito:

- La validación de las líneas con declaraciones múltiples lanza una excepción

Criterios de falla:

- La validación de las líneas con declaraciones múltiples retorna Verdadero

Código utilizado:

```java
@Test
@DisplayName("Debe de lanzar la excepion al detectar incialización de varias variables")
void testInvalidMultipleDeclarationsWithSemicolon() {
  SingleDeclarationValidator validator = new SingleDeclarationValidator();
  assertThrows(
      InvalidFormatException.class,
      () -> {
        validator.isValid("int value 1 = 10; double value2 = 12.8;");
      });
  assertThrows(
      InvalidFormatException.class,
      () -> {
        validator.isValid("value; value;");
      });
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Se lanzó correctamente la excepción al detectar múltiples declaraciones separadas por punto y coma.

2.24 Caso de prueba 024: Lanzar excepción al detectar inicialización de múltiples variables con punto y coma

Identificador de caso de prueba: CP024

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaraciones de variables

***Objetivo**:*

Verificar que el sistema rechace correctamente declaraciones múltiples de variables separadas por comas y lance una excepción

Descripción:

Se valida que al analizar líneas con más de una declaración se lance una excepción

Criterios de éxito:

- La validación de las líneas con declaraciones múltiples lanza una excepción

Criterios de falla:

- La validación de las líneas con declaraciones múltiples retorna Verdadero

Código utilizado:

```
@Test
@DisplayName("Debe de lanzar la excepion al detectar incialización de varias variables ")
void testInvalidMultipleDeclarationsWithCommas() {
  SingleDeclarationValidator validator = new SingleDeclarationValidator();
  assertThrows(
      InvalidFormatException.class,
      () -> {
        validator.isValid("int value1, value2, value3;");
      });
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Se lanzó correctamente la excepción al detectar múltiples declaraciones separadas por comas.

2.25 Caso de prueba 025: Lanzar excepción si la línea tiene un corchete de apertura inválido

Identificador de caso de prueba: CP025

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de corchetes en líneas de código

***Objetivo**:*

Verificar que el sistema detecte y rechace correctamente una línea con un corchete de apertura { inválido.

Descripción:

Se valida que al analizar una línea con un corchete de apertura invalido se lance una excepción

Criterios de éxito:

- La validación de la línea con un corchete de apertura inválido lanza una excepción

Criterios de falla:

- La validación de la línea con un corchete de apertura inválido retorna Verdadero

Código utilizado:

```
@Test
@DisplayName(
    "Debe lanzar InvalidFormatException si la línea tiene un corchete de apertura inválido")
void testInvalidOpeningBracketThrowsException() {
  String invalidLine = " {";
  assertThrows(InvalidFormatException.class, () -> validator.isValid(invalidLine));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Se lanzó correctamente la excepción al detectar un corchete de apertura inválido.

2.26 Caso de prueba 026: Lanzar excepción si la línea tiene un corchete de cierre inválido

Identificador de caso de prueba: CP026

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de corchetes en líneas de código

Objetivo*:*

Verificar que el sistema detecte y rechace correctamente una línea con un corchete de cierre } inválido.

Descripción:

Se valida que al analizar una línea con un corchete de cierre invalido se lance una excepción

Criterios de éxito:

- La validación de la línea con un corchete de cierre inválido lanza una excepción

Criterios de falla:

- La validación de la línea con un corchete de cierre inválido retorna Verdadero

Código utilizado:

```java
@Test
@DisplayName(
    "Debe lanzar InvalidFormatException si la línea tiene un corchete de cierre inválido")
void testInvalidClosingBracketThrowsException() {
  String invalidLine = "} if";
  assertThrows(InvalidFormatException.class, () -> validator.isValid(invalidLine));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Se lanzó correctamente la excepción al detectar un corchete de cierre inválido.

2.27 Caso de prueba 027: Lanzar excepción si la línea contiene corchetes vacíos

Identificador de caso de prueba: CP027

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de corchetes en líneas de código

Objetivo*:*

Verificar que el sistema detecte y rechace correctamente una línea con corchetes {} vacíos.

Descripción:

Se valida que al analizar una línea con corchetes vacíos se lance una excepción

Criterios de éxito:

- La validación de la línea con corchetes vacíos lanza una excepción

Criterios de falla:

- La validación de la línea con corchetes vacíos retorna Verdadero

Código utilizado:

```
@Test
@DisplayName("Debe lanzar InvalidFormatException si la línea contiene corchetes vacíos")
void testEmptyBracketsThrowsException() {
  String invalidLine = "{}";
  assertThrows(InvalidFormatException.class, () -> validator.isValid(invalidLine));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Se lanzó correctamente la excepción al detectar corchetes vacíos.

2.28 Caso de prueba 028: Lanzar excepción si existen {} en una sola línea con o sin contenido

Identificador de caso de prueba: CP028

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de corchetes en líneas de código

Objetivo*:*

Verificar que el sistema detecte y rechace correctamente una línea que contiene {} en una sola línea, con o sin contenido.

Descripción:

Se valida que al analizar una línea con corchetes en una sola línea, con o sin contenido, se lance una excepción

Criterios de éxito:

- La validación de la línea con corchetes en una sola línea, con o sin contenido, lanza una excepción

Criterios de falla:

- La validación de la línea con corchetes en una sola línea, con o sin contenido, retorna Verdadero

Código utilizado:

```
@Test
@DisplayName(
    "Debe devolver lanzar InvalidFormatException si existen {} en una sola línea con o sin"
        + " contenido")
void testValidLineReturnsTrue() {
  String validLine = "if (x > 0) { doSomething(); }";
  assertThrows(InvalidFormatException.class, () -> assertTrue(validator.isValid(validLine)));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Se lanzó correctamente la excepción al detectar corchetes en una sola línea.

2.29 Caso de prueba 029: Validar correctamente una declaración de clase

Identificador de caso de prueba: CP029

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaración de clase

***Objetivo**:*

Verificar que el sistema valide correctamente una declaración de clase en Java.

Descripción:

Se valida una declaración de clase válida

Criterios de éxito:

- La validación de la declaración de clase válida retorna Verdadero

Criterios de falla:

- La validación de la declaración de clase válida retorna Falso o lanza una excepción

Código utilizado:

```
@Test
@DisplayName("Debe de validar correctamente una declaración de clase")
void testValidateTypeWithValidClassDeclaration() {
  String validClassDeclaration = "public class MyClass {";
  assertTrue(validator.validateType(validClassDeclaration));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

La declaración de clase válida fue validada correctamente.

2.30 Caso de prueba 030: Validar correctamente una declaración de interfaz

Identificador de caso de prueba: CP030

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaración de interfaz

***Objetivo**:*

Verificar que el sistema valide correctamente una declaración de interfaz en Java.

Descripción:

Se valida una declaración de interfaz válida

Criterios de éxito:

- La validación de la declaración de interfaz válida retorna Verdadero

Criterios de falla:

- La validación de la declaración de interfaz válida retorna Falso o lanza una excepción

Código utilizado:

```
@Test
@DisplayName("Debe de validar correctamente una declaración de interfaz")
void testValidateTypeWithValidInterfaceDeclaration() {
  String validInterfaceDeclaration = "public interface MyInterface {";
  assertTrue(validator.validateType(validInterfaceDeclaration));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

"True"

Resultados de la prueba:

La declaración de interfaz válida fue validada correctamente.

2.31 Caso de prueba 031: Validar correctamente una declaración de enum

Identificador de caso de prueba: CP031

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaración de enum

Objetivo*:*

Verificar que el sistema valide correctamente una declaración de enum en Java.

Descripción:

Se valida una declaración de enum válido

Criterios de éxito:

- La validación de la declaración de enum válido retorna Verdadero

Criterios de falla:

- La validación de la declaración de enum válido retorna Falso o lanza una excepción

Código utilizado:

```java
@Test
@DisplayName("Debe de validar correctamente una declaración de enum")
void testValidateTypeWithValidEnumDeclaration() {
  String validEnumDeclaration = "public enum MyEnum {";
  assertTrue(validator.validateType(validEnumDeclaration));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

La declaración de enum válido fue validada correctamente.

2.32 Caso de prueba 032: No validar incorrectamente una declaración de método inválido

Identificador de caso de prueba: CP032

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaración de método

Objetivo*:*

Verificar que el sistema no valide incorrectamente una declaración de método inválido en Java.

Descripción:

No se valida una declaración de método invalido.

Criterios de éxito:

- La validación de la declaración de método inválido retorna Falso

Criterios de falla:

- La validación de la declaración de método inválido retorna Verdadero o lanza una excepción

Código utilizado:

```
@Test
@DisplayName("Debe de validar correctamente una declaración de método")
void testValidateTypeWithInvalidDeclaration() {
  String invalidDeclaration = "public void myMethod() {";
  assertFalse(validator.validateType(invalidDeclaration));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

"True"

Resultados de la prueba:

La declaración de método inválido no fue validada incorrectamente.

2.33 Caso de prueba 033: No validar incorrectamente una línea vacía

Identificador de caso de prueba: CP033

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de tipo

Objetivo*:*

Verificar que el sistema no valide incorrectamente una línea vacía en Java.

Descripción:

No se valida una línea vacía.

Criterios de éxito:

- La validación de la línea vacía retorna Falso

Criterios de falla:

- La validación de la línea vacía retorna Verdadero o lanza una excepción

Código utilizado:

```
@Test
@DisplayName("Debe de validar correctamente una declaración con liea vacía")
void testValidateTypeWithEmptyLine() {
  String emptyLine = "";
  assertFalse(validator.validateType(emptyLine));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

La línea vacía no fue validada incorrectamente.

2.34 Caso de prueba 034: No validar incorrectamente una línea con solo espacios en blanco

Identificador de caso de prueba: CP034

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de tipo

***Objetivo**:*

Verificar que el sistema no valide incorrectamente una línea con solo espacios vacíos en Java.

Descripción:

No se valida una línea con solo espacios vacíos.

Criterios de éxito:

- La validación de la línea con solo espacios vacíos retorna Falso

Criterios de falla:

- La validación de la línea con solo espacios vacíos retorna Verdadero o lanza una excepción

Código utilizado:

```
@Test
@DisplayName("Debe de validar correctamente una declaración con espacios en blanco")
void testValidateTypeWithWhitespaceOnly() {
  String whitespaceOnly = "    ";
  assertFalse(validator.validateType(whitespaceOnly));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

La línea con solo espacios vacíos no fue validada incorrectamente.

2.35 Caso de prueba 035: No validar incorrectamente un comentario

Identificador de caso de prueba: CP035

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de tipo

***Objetivo**:*

Verificar que el sistema no valide incorrectamente una línea de comentario en Java.

Descripción:

No se valida una línea de comentario.

Criterios de éxito:

- La validación de la línea de comentario retorna Falso

Criterios de falla:

- La validación de la línea de comentario retorna Verdadero o lanza una excepción

Código utilizado:

```
@Test
@DisplayName("Debe de validar correctamente una declaración con comentario de bloque")
void testValidateTypeWithCommentLine() {
  String commentLine = "// This is a comment";
  assertFalse(validator.validateType(commentLine));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

La línea de comentario no fue validada incorrectamente.

2.36 Caso de prueba 036: No validar incorrectamente una declaración parcial

Identificador de caso de prueba: CP036

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de tipo

***Objetivo**:*

Verificar que el sistema no valide incorrectamente una declaración parcial en Java.

Descripción:

No se valida una declaración parcial.

Criterios de éxito:

- La validación de la declaración parcial retorna Falso

Criterios de falla:

- La validación de la declaración parcial retorna Verdadero o lanza una excepción

Código utilizado:

```
@Test
@DisplayName("Debe de validar correctamente una declaración parcial")
void testValidateTypeWithPartialDeclaration() {
  String partialDeclaration = "class";
  assertFalse(validator.validateType(partialDeclaration));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

"True"

Resultados de la prueba:

La declaración parcial no fue validada incorrectamente.

2.37 Caso de prueba 037: Validar correctamente una declaración de método válida

Identificador de caso de prueba: CP037

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaración de método

Objetivo*:*

Verificar que el sistema valide correctamente una declaración de método válida en Java.

Descripción:

Se valida una declaración de método válida

Criterios de éxito:

- La validación de la declaración de método válido retorna Verdadero

Criterios de falla:

- La validación de la declaración de método válido retorna Falso o lanza una excepción

Código utilizado:

```java
@Test
@DisplayName("Debe retornar true para una declaración de método válida")
void testValidateType_ValidMethodDeclaration() {
  String validMethod = "public static void main(String[] args)";
  assertTrue(
      validator.validateType(validMethod), "Should return true for a valid method declaration");
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

La declaración de método válido fue validada correctamente.

2.38 Caso de prueba 038: Validar correctamente una declaración de método sin modificadores

Identificador de caso de prueba: CP038

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaración de método

Objetivo*:*

Verificar que el sistema valide correctamente una declaración de método sin modificadores válida en Java.

Descripción:

Se valida una declaración de método sin modificadores válida

Criterios de éxito:

- La validación de la declaración de método sin modificaciones válido retorna Verdadero

Criterios de falla:

- La validación de la declaración de método sin modificadores válido retorna Falso o lanza una excepción

Código utilizado:

```
@Test
@DisplayName("Debe retornar true para una declaración de método válida sin modificadores")
void testValidateType_ValidMethodWithoutModifiers() {
  String validMethod = "void doSomething()";
  assertTrue(
      validator.validateType(validMethod),
      "Should return true for a valid method declaration without modifiers");
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

"True"

Resultados de la prueba:

La declaración de método sin modificadores válido fue validada correctamente.

2.39 Caso de prueba 039: No validar una declaración de método con punto y coma

Identificador de caso de prueba: CP039

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaración de método

Objetivo:

Verificar que el sistema no valide incorrectamente una declaración de método con punto y coma.

Descripción:

Se valida una declaración de método con punto y coma

Criterios de éxito:

- La validación de la declaración de método con punto y coma válido retorna Falso

Criterios de falla:

- La validación de la declaración de método con punto y coma retorna Verdadero o lanza una excepción

Código utilizado:

```
@Test
@DisplayName("Debe retornar false para una declaración de método con punto y coma")
void testValidateType_InvalidMethodWithSemicolon() {
  String invalidMethod = "public void doSomething();";
  assertFalse(
      validator.validateType(invalidMethod),
      "Should return false for a method declaration with a semicolon");
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

La declaración de método con punto y coma no fue validada incorrectamente.

2.40 Caso de prueba 040: No validar una línea de código que no es una declaración de método

Identificador de caso de prueba: CP040

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaración de método

***Objetivo**:*

Verificar que el sistema no valide incorrectamente una línea de código que no es una declaración de método.

Descripción:

Se valida una línea de código que no es una declaración de método

Criterios de éxito:

- La validación de la línea de código que no es una declaración de método retorna Falso

Criterios de falla:

- La validación de la línea de código que no es una declaración de método retorna Verdadero o lanza una excepción

Código utilizado:

```
@Test
@DisplayName("Debe retornar false para una línea de código que no es una declaración de método")
void testValidateType_InvalidLineOfCode() {
  String invalidCode = "int x = 0;";
  assertFalse(
      validator.validateType(invalidCode),
      "Should return false for a non-method declaration line of code");
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

"True"

Resultados de la prueba:

La línea de código que no es una declaración de método no fue validada incorrectamente.

2.41 Caso de prueba 041: No validar una línea vacía como declaración de método

Identificador de caso de prueba: CP041

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaración de método

Objetivo*:*

Verificar que el sistema no valide incorrectamente una línea vacía que no es una declaración de método.

Descripción:

Se valida una línea vacía que no es una declaración de método

Criterios de éxito:

- La validación de la línea vacía que no es una declaración de método retorna Falso

Criterios de falla:

- La validación de la línea vacía que no es una declaración de método retorna Verdadero o lanza una excepción

Código utilizado:

```java
@Test
@DisplayName("Debe retornar false para una línea vacía")
void testValidateType_EmptyLine() {
  String emptyLine = "";
  assertFalse(validator.validateType(emptyLine), "Should return false for an empty line");
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

La línea vacía que no es una declaración de método no fue validada incorrectamente.

2.42 Caso de prueba 042: No validar una línea con solo espacios en blanco como declaración de método

Identificador de caso de prueba: CP042

Autor*:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaración de método

Objetivo*:*

Verificar que el sistema no valide incorrectamente una línea con solo espacios en blanco que no es una declaración de método.

Descripción:

Se valida una línea con solo espacios en blanco que no es una declaración de método

Criterios de éxito:

- La validación de la línea con solo espacios en blanco que no es una declaración de método retorna Falso

Criterios de falla:

- La validación de la línea con solo espacios en blanco que no es una declaración de método retorna Verdadero o lanza una excepción

Código utilizado:

```
@Test
@DisplayName("Debe retornar false para una línea con solo espacios en blanco")
void testValidateType_WhitespaceOnly() {
  String whitespaceOnly = "    ";
  assertFalse(
      validator.validateType(whitespaceOnly),
      "Should return false for a line with only whitespace");
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

"True"

Resultados de la prueba:

La línea con solo espacios en blanco que no es una declaración de método no fue validada incorrectamente.

2.43 Caso de prueba 043: No validar una declaración de método sin paréntesis

Identificador de caso de prueba: CP043

***Autor**:* José Chi

Fecha de creación: 31 de marzo de 2025

Función para probar: Validación de declaración de método

***Objetivo**:*

Verificar que el sistema no valide incorrectamente una declaración de método sin paréntesis.

Descripción:

Se valida una declaración de método sin paréntesis.

Criterios de éxito:

- La validación de la declaración de método sin paréntesis retorna Falso

Criterios de falla:

- La validación de la declaración de método sin paréntesis retorna Verdadero o lanza una excepción

Código utilizado:

```
@Test
@DisplayName("Debe retornar false para una declaración de método sin paréntesis")
void testValidateType_InvalidMethodWithoutParentheses() {
  String invalidMethod = "public void doSomething";
  assertFalse(
      validator.validateType(invalidMethod),
      "Should return false for a method declaration without parentheses");
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

La declaración de método sin paréntesis no fue validada incorrectamente.

2.44 Caso de prueba 044: Detectar línea sin cambio entre versiones

Identificador de caso de prueba: CP044

***Autor**:* Mónica Garcilazo Cuevas

Fecha de creación: 14 de mayo de 2025

Función para probar: Detección de líneas originales sin cambios

***Objetivo**:*

Verificar que una línea presente en ambas versiones se clasifique como [ORIGINAL]

Descripción:

Cuando se comparan dos archivos con líneas exactamente iguales, cada línea debe marcarse como ORIGINAL en ambos archivos.

Criterios de éxito:

- Se etiqueta como [ORIGINAL] en ambas líneas.

Criterios de falla:

- Se etiqueta como otra categoría

Código utilizado:

```java
@Test
public void testCompareContent_identicalContent() {
    List<String> contentA = Arrays.asList(
            ...a:"public class Test {",
            "    int x = 5;",
            "}"
    );
    List<String> contentB = Arrays.asList(
            ...a:"public class Test {",
            "    int x = 5;",
            "}"
    );

    JavaFileComparator comparator = new JavaFileComparator(contentA, contentB);
    comparator.compareContent();

    ComparationReport report = comparator.getComparationReport();

    for (LineRecord line : report.getSourceContentReport()) {
        assertEquals(Status.ORIGINAL, line.status());
    }

    for (LineRecord line : report.getTargetContentReport()) {
        assertEquals(Status.ORIGINAL, line.status());
    }
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Los dos líneas fueron etiquetadas como ORIGINAL correctamente

2.45 Caso de prueba 045: Detección de línea nueva.

Identificador de caso de prueba: CP045

Autor*:* Mónica Garcilazo Cuevas

Fecha de creación: 14 de mayo de 2025

Función para probar: Detección de líneas nuevas.

Objetivo*:*

Verificar que la línea agregada se etiquete como [NEW]

Descripción:

Cuando se detecta una línea nueva en el segundo archivo esta se etiqueta como nueva.

Criterios de éxito:

- Se etiqueta como [NEW] la línea.

Criterios de falla:

- Se etiqueta como otra categoría

Código utilizado:

```
@Test
public void testCompareContent_newLine() {
    List<String> contentA = Arrays.asList(
            ...a:"public class Test {"
    );
    List<String> contentB = Arrays.asList(
            ...a:"public class Test {",
            "    int y = 10;"
    );

    JavaFileComparator comparator = new JavaFileComparator(contentA, contentB);
    comparator.compareContent();

    ComparationReport report = comparator.getComparationReport();
    assertTrue(report.getTargetContentReport().stream().anyMatch(r -> r.status() == Status.NEW));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

La nueva línea fue etiquetada como [New] correctamente.

2.46 Caso de prueba 046: Detección de archivo más corto (líneas eliminadas)

Identificador de caso de prueba: CP046

***Autor**:* Mónica Garcilazo Cuevas

Fecha de creación: 14 de mayo de 2025

Función para probar: Detección de líneas eliminadas

***Objetivo**:*

Verificar que si el archivo original tiene más líneas que el archivo nuevo, estas líneas se clasifiquen como [DELETED].

Descripción:

Cuando hay líneas al final del archivo original que no existen en el archivo nuevo, deben marcarse como [DELETED].

Criterios de éxito:

- Las líneas faltantes se clasifican como [DELETED]
- Se actualiza correctamente el contador de líneas eliminadas.

Criterios de falla:

- No se detectan líneas eliminadas
- Las líneas son etiquetadas erróneamente.

Código utilizado:

```java
@Test
public void testCompareContent_differentSizes() {
    List<String> contentA = Arrays.asList(
        ...a:"public class Test {",
        "    int x = 5;",
        "}"
    );
    List<String> contentB = Arrays.asList(
        ...a:"public class Test {",
        "}"
    );

    JavaFileComparator comparator = new JavaFileComparator(contentA, contentB);
    comparator.compareContent();

    ComparationReport report = comparator.getComparationReport();
    assertFalse(report.getSourceContentReport().isEmpty());
    assertFalse(report.getTargetContentReport().isEmpty());
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Líneas adicionales en archivo anterior correctamente marcadas como eliminadas.

2.47 Caso de prueba 047: Detectar línea eliminada en la versión nueva

Identificador de caso de prueba: CP047

Autor*:* Mónica Garcilazo Cuevas

Fecha de creación: 14 de mayo de 2025

Función para probar: Detección de líneas eliminadas

Objetivo*:*

Verificar que si una línea presente en el archivo original no se encuentra en la versión nueva, sea etiquetada como [DELETED].

Descripción:

El archivo nuevo tiene una línea menos que el original, indicando que fue eliminada.

Criterios de éxito:

- Se etiqueta como [DELETED]
- Se incrementa el contador de líneas eliminadas.

Criterios de falla:

- Se etiqueta como otra categoría
- No se detecta la eliminación.

Código utilizado:

```
@Test
public void testCompareContent_deletedLine() {
    List<String> contentA = Arrays.asList(
            ...a:"public class Test {",
            "    int z = 9;"
    );
    List<String> contentB = Arrays.asList(
            ...a:"public class Test {"
    );

    JavaFileComparator comparator = new JavaFileComparator(contentA, contentB);
    comparator.compareContent();

    ComparationReport report = comparator.getComparationReport();
    assertTrue(report.getSourceContentReport().stream().anyMatch(r -> r.status() == Status.DELETED));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Línea eliminada correctamente detectada.

2.48 Caso de prueba 048: Detectar línea modificada entre versiones

Identificador de caso de prueba: CP048

Autor*:* Mónica Garcilazo Cuevas

Fecha de creación: 14 de mayo de 2025

Función para probar: Detección de líneas modificadas

Objetivo*:*

Verificar que una línea modificada en la versión reciente se clasifique como [MODIFIED].+

Descripción:

Cuando una línea ha sido modificada (texto distinto, pero posición similar), debe marcarse como ORIGINAL en el archivo anterior y MODIFIED en el nuevo archivo.

Criterios de éxito:

- Se etiqueta como [MODIFIED] en el archivo generado.
- Se incrementa el contador de líneas modificadas.

Criterios de falla:

- Se etiqueta como otra categoría
- No se contabiliza correctamente.

Código utilizado:

```
@Test
public void testCompareContent_modifiedLine() {
    List<String> contentA = Arrays.asList(
            ...a:"public class Test {",
            "    int x = 4;",
            "}"
    );
    List<String> contentB = Arrays.asList(
            ...a:"public class Test {",
            "    int x = 5;",
            "}"
    );

    JavaFileComparator comparator = new JavaFileComparator(contentA, contentB);
    comparator.compareContent();

    ComparationReport report = comparator.getComparationReport();
    assertTrue(report.getSourceContentReport().stream().anyMatch(r -> r.status() == Status.ORIGINAL));
    assertTrue(report.getTargetContentReport().stream().anyMatch(r -> r.status() == Status.MODIFIED));
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

La línea modificada fue detectada correctamente.

2.49 Caso de prueba 049: Manejo de líneas vacías

Identificador de caso de prueba: CP049

***Autor**:* Mónica Garcilazo Cuevas

Fecha de creación: 14 de mayo de 2025

Función para probar: Comparación de líneas vacías

***Objetivo**:*

Verificar que líneas vacías en uno u ambos archivos se procesen correctamente y no generen errores.

Descripción:

Una o más líneas están vacías en alguna versión del archivo. Se debe determinar si fueron agregadas, eliminadas o sin cambios.

Criterios de éxito:

- No se producen excepción.
- Se incrementa el contador de líneas nuevas.

Criterios de falla:

- Se etiqueta como otra categoría
- No se contabiliza correctamente.

Código utilizado:

```java
@Test
public void testCompareContent_withEmptyLines() {
    List<String> contentA = Arrays.asList(
        ...a:"System.out.println(\"Hello\");",
        "",
        "System.out.println(\"World\");"
    );
    List<String> contentB = Arrays.asList(
        ...a:"System.out.println(\"Hello\");",
        "System.out.println(\"World\");"
    );

    JavaFileComparator comparator = new JavaFileComparator(contentA, contentB);
    comparator.compareContent();

    ComparationReport report = comparator.getComparationReport();
    assertEquals(3, report.getSourceContentReport().size());
    assertEquals(Status.DELETED, report.getSourceContentReport().get(index:1).status());
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

Las líneas vacías fueron manejadas correctamente.

2.50 Caso de prueba 050: Resumen global correcto

Identificador de caso de prueba: CP050

Autor*:* Mónica Garcilazo Cuevas

Fecha de creación: 14 de mayo de 2025

Función para probar: Detección de líneas eliminadas

Objetivo*:*

Verificar que una línea que existía solo en la versión previa se etiquete como [DELETED].

Descripción:

Se compara una línea idéntica en ambos archivos

Criterios de éxito:

- Se etiqueta como [NEW] en el archivo generado.
- Se incrementa el contador de líneas nuevas.

Criterios de falla:

- Se etiqueta como otra categoría
- No se contabiliza correctamente.

Código utilizado:

```
@Test
public void testGlobalSummaryCountsWithoutExporting() {
    ComparationReport report = new ComparationReport();

    report.makeReportLine(Status.ORIGINAL, sourceLine:"a", targetLine:"a");
    report.makeReportLine(Status.MODIFIED, sourceLine:"b", targetLine:"c");
    report.makeReportLine(Status.NEW, sourceLine:"", targetLine:"d");
    report.makeReportLine(Status.NEW, sourceLine:"", targetLine:"e");
    report.makeReportLine(Status.DELETED, sourceLine:"f", targetLine:"");

    Map<Status, Long> counts = report.getTargetContentReport().stream()
        .collect(java.util.stream.Collectors.groupingBy(LineRecord::status, java.util.stream.Collectors.counting()));

    assertEquals(1, counts.getOrDefault(Status.ORIGINAL, defaultValue:0L));
    assertEquals(1, counts.getOrDefault(Status.MODIFIED, defaultValue:0L));
    assertEquals(2, counts.getOrDefault(Status.NEW, defaultValue:0L));
    assertEquals(0, counts.getOrDefault(Status.DELETED, defaultValue:0L));

    long deletedCount = report.getSourceContentReport().stream()
        .filter(lr -> lr.status() == Status.DELETED)
        .count();
    assertEquals(2, deletedCount);
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

La declaración de método sin paréntesis no fue validada incorrectamente.

2.51 Caso de prueba 051: Generar archivos con etiquetas por línea

Identificador de caso de prueba: CP051

***Autor**:* Mónica Garcilazo Cuevas

Fecha de creación: 14 de mayo de 2025

Función para probar: Detección de líneas eliminadas

***Objetivo**:*

Verificar que una línea que existía solo en la versión previa se etiquete como [DELETED].

Descripción:

Se compara una línea idéntica en ambos archivos

Criterios de éxito:

- Se etiqueta como [NEW] en el archivo generado.
- Se incrementa el contador de líneas nuevas.

Criterios de falla:

- Se etiqueta como otra categoría
- No se contabiliza correctamente.

Código utilizado:

```java
@Test
public void testLinesHaveCorrectTagsWithoutExporting() {
    ComparationReport report = new ComparationReport();

    report.makeReportLine(Status.ORIGINAL, sourceLine:"int x = 1;", targetLine:"int x = 1;");
    report.makeReportLine(Status.MODIFIED, sourceLine:"int y = 2;", targetLine:"int y = 3;");
    report.makeReportLine(Status.NEW, sourceLine:"", targetLine:"int z = 4;");
    report.makeReportLine(Status.DELETED, sourceLine:"int a = 5;", targetLine:"");

    List<LineRecord> target = report.getTargetContentReport();
    List<LineRecord> source = report.getSourceContentReport();

    assertEquals(Status.ORIGINAL, target.get(index:0).status());
    assertEquals(Status.MODIFIED, target.get(index:1).status());
    assertEquals(Status.NEW, target.get(index:2).status());

    assertEquals(Status.ORIGINAL, source.get(index:0).status());
    assertEquals(Status.ORIGINAL, source.get(index:1).status());
    assertEquals(Status.DELETED, source.get(index:2).status());
}
```

Resultados de ejecución

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

La declaración de método sin paréntesis no fue validada incorrectamente.

2.52 Caso de prueba 052: Crear línea [ORIGINAL]

Identificador de caso de prueba: CP052

Autor*:* Mónica Garcilazo Cuevas

Fecha de creación: 14 de mayo de 2025

Función para probar: Generación de línea sin cambios.

Objetivo*:*

Verificar que se agregue correctamente un registro con estado [ORIGINAL] tanto en el contenido original como en el comparado

Descripción:

Genera un registro cuando una línea no ha cambiado entre versiones.

Criterios de éxito:

- Se etiqueta como [ORIGINAL] en el reporte generado.

Criterios de falla:

- Se etiqueta como otra categoría

Código utilizado:

```java
@Test
public void testMakeReportLine_Original() {
    ComparationReport report = new ComparationReport();

    report.makeReportLine(Status.ORIGINAL, sourceLine:"System.out.println(\"Hola\");", targetLine:"System.out.println(\"Hola\");");

    List<LineRecord> current = report.getSourceContentReport();
    List<LineRecord> compare = report.getTargetContentReport();

    assertEquals(1, current.size());
    assertEquals(Status.ORIGINAL, current.get(index:0).status());
    assertEquals("System.out.println(\"Hola\");", current.get(index:0).content());

    assertEquals(1, compare.size());
    assertEquals(Status.ORIGINAL, compare.get(index:0).status());
}
```

Resultados de ejecución:

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

La línea fue etiquetada correctamente como [ORIGINAL].

La declaración de método sin paréntesis no fue validada incorrectamente.

2.53 Caso de prueba 053: Crear línea [MODIFIED]

Identificador de caso de prueba: CP053

***Autor**:* Mónica Garcilazo Cuevas

Fecha de creación: 14 de mayo de 2025

Función para probar: Generación de línea cuando hay una modificación.

***Objetivo**:*

Verificar que se agregue correctamente un registro con estado [MODIFIED] en el contenido comparado.

Descripción:

Genera un registro cuando una línea cambia de contenido entre versiones.

Criterios de éxito:

- Se etiqueta como [MODIFIED] en el reporte generado.

Criterios de falla:

- Se etiqueta como otra categoría

Código utilizado:

```
@Test
public void testMakeReportLine_Modified() {
    ComparationReport report = new ComparationReport();

    report.makeReportLine(Status.MODIFIED, sourceLine:"int x = 1;", targetLine:"int x = 2;");

    List<LineRecord> current = report.getSourceContentReport();
    List<LineRecord> compare = report.getTargetContentReport();

    assertEquals(1, current.size());
    assertEquals(Status.ORIGINAL, current.get(index:0).status());

    assertEquals(1, compare.size());
    assertEquals(Status.MODIFIED, compare.get(index:0).status());
}
```

Resultados de ejecución:

Fecha de ejecución: 15 de mayo de 2025

Salida:

"True"

Resultados de la prueba:

La línea modificada fue detectada y etiquetada correctamente.

2.54 Caso de prueba 054: Crear línea [NEW]

Identificador de caso de prueba: CP054

***Autor**:* Mónica Garcilazo Cuevas

Fecha de creación: 14 de mayo de 2025

Función para probar: Detección de líneas nuevas.

***Objetivo**:*

Verificar que se agregue correctamente un registro con estado [NEW] en el contenido comparado.

Descripción:

Genera un registro para una línea que aparece solo en la nueva versión.

Criterios de éxito:

- Se etiqueta como [NEW] en el reporte generado.

Criterios de falla:

- Se etiqueta como otra categoría

Código utilizado:

```java
@Test
public void testMakeReportLine_New() {
    ComparationReport report = new ComparationReport();

    report.makeReportLine(Status.NEW, sourceLine:"int z = 3;", targetLine:"int z = 3;");

    List<LineRecord> current = report.getSourceContentReport();
    List<LineRecord> compare = report.getTargetContentReport();

    assertEquals(1, current.size());
    assertEquals(Status.DELETED, current.get(index:0).status());

    assertEquals(1, compare.size());
    assertEquals(Status.NEW, compare.get(index:0).status());
}
```

Resultados de ejecución:

Fecha de ejecución: 15 de mayo de 2025

Salida:

"True"

Resultados de la prueba:

La línea fue correctamente identificada como nueva.

2.55 Caso de prueba 055: Crear línea [DELETED]

Identificador de caso de prueba: CP055

Autor*:* Mónica Garcilazo Cuevas

Fecha de creación: 14 de mayo de 2025

Función para probar: Detección de líneas eliminadas.

Objetivo*:*

Verificar que se agregue correctamente un registro con estado [DELETED] en el contenido original.

Descripción:

Registra líneas que estaban en la versión antigua y ya no existen.

Criterios de éxito:

- Se etiqueta como [DELETED] en el reporte generado.

Criterios de falla:

- Se etiqueta como otra categoría

Código utilizado:

```
@Test
public void testUpdateReport_DeletedLines() {
    ComparationReport report = new ComparationReport();

    List<String> content = Arrays.asList(...a:"line1", "line2", "line3", "line4");

    report.updateReport(content, difference:2);

    List<LineRecord> current = report.getSourceContentReport();

    assertEquals(2, current.size());
    assertEquals("line3", current.get(index:0).content());
    assertEquals(Status.DELETED, current.get(index:0).status());

    assertEquals("line4", current.get(index:1).content());
    assertEquals(Status.DELETED, current.get(index:1).status());
}
```

Resultados de ejecución:

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

La eliminación fue registrada correctamente.

2.56 Caso de prueba 056: Agregar múltiples líneas nuevas.

Identificador de caso de prueba: CP056

***Autor**:* Mónica Garcilazo Cuevas

Fecha de creación: 14 de mayo de 2025

Función para probar: Detección de líneas eliminadas.

***Objetivo**:*

Verificar que múltiples líneas nuevas se agreguen al reporte con la etiqueta [NEW].

Descripción:

Procesa múltiples líneas nuevas y las agrega como [NEW].

Criterios de éxito:

- Se etiquetan como [NEW] en el reporte generado, todas las líneas nuevas.

Criterios de falla:

- Se etiqueta como otra categoría.
- Se omite alguna línea.

Código utilizado:

```
@Test
public void testUpdateReport_NewLines_Multiple() {
    ComparationReport report = new ComparationReport();

    List<String> content = Arrays.asList(...a:"a", "b");
    List<String> contentToCompare = Arrays.asList(...a:"a", "b", "c", "d");

    int difference = 2;

    report.updateReport(content, contentToCompare, difference);

    List<LineRecord> current = report.getSourceContentReport();

    assertEquals(2, current.size());
    assertEquals("c", current.get(index:0).content());
    assertEquals(Status.NEW, current.get(index:0).status());
    assertEquals("d", current.get(index:1).content());
    assertEquals(Status.NEW, current.get(index:1).status());
}
```

Resultados de ejecución:

Fecha de ejecución: 15 de mayo de 2025

Salida:

"True"

Resultados de la prueba:

Todas las líneas nuevas fueron registradas correctamente.

2.57 Caso de prueba 057: Dividir línea larga preservando contenido lógico.

Identificador de caso de prueba: CP057

Autor*:* Mónica Garcilazo Cuevas

Fecha de creación: 14 de mayo de 2025

Función para probar: Detección de líneas eliminadas.

Objetivo*:*

Verificar que una línea que excede los 80 caracteres sea dividida correctamente en fragmentos más pequeños, manteniendo su contenido lógico y estado.

Descripción:

Se proporciona una línea de más de 80 caracteres para asegurar que se divida la línea correctamente. El primer fragmento debe marcarse como SPLITED, y los fragmentos posteriores deben conservar el estado original (ORIGINAL). Además, al unirlos, el contenido debe coincidir con la línea original.

Criterios de éxito:

- Generar múltiples fragmentos.

- El primer fragmento tiene estado [SPLITTED]
- Los demás fragmentos tienen estado [ORIGINAL]

Criterios de falla:

- Se genera solo un fragmento para una línea muy larga.
- Los estados asignados no son los correctos.
- La reconstrucción del contenido no coincide con el original.
- Se cuentan como dos líneas.

Código utilizado:

```java
@Test
public void testSplitLongLine_ShouldSplitAndPreserveLogicalLine() {
    String longLine = "Esto es una línea muy larga que definitivamente excede los ochenta caracteres, y debe ser dividida en múltiples part
    LineRecord original = new LineRecord(Status.ORIGINAL, longLine);

    List<LineRecord> result = LineSplitter.splitLongLines(original);

    assertTrue(result.size() > 1, "La línea debería haber sido dividida en múltiples fragmentos");

    assertEquals(Status.SPLITED, result.get(index:0).status(), "El primer fragmento debe tener estado SPLITED");

    for (int i = 1; i < result.size(); i++) {
        assertEquals(Status.ORIGINAL, result.get(i).status(), "Los fragmentos siguientes deben mantener el estado ORIGINAL");
    }

    StringBuilder reconstructed = new StringBuilder();
    for (LineRecord part : result) {
        reconstructed.append(part.content()).append(str:" ");
    }
    String reconstructedLine = reconstructed.toString().trim().replaceAll(regex:"\\s+", replacement:" ");
    String originalNormalized = longLine.trim().replaceAll(regex:"\\s+", replacement:" ");
    assertEquals(originalNormalized, reconstructedLine, "El contenido reconstruido debe ser igual al original");
}
```

Resultados de ejecución:

Fecha de ejecución: 15 de mayo de 2025

Salida:

“True”

Resultados de la prueba:

La línea fue correctamente dividida y reconstruida, cumpliendo con los criterios establecidos.

3. RESUMEN DE RESULTADOS

ID Caso	Resultado (Éxito / Fallo)	Fecha
CP001	Éxito	15 de mayo de 2025
CP002	Éxito	15 de mayo de 2025
CP003	Éxito	15 de mayo de 2025
CP004	Éxito	15 de mayo de 2025
CP005	Éxito	15 de mayo de 2025
CP006	Éxito	15 de mayo de 2025
CP007	Éxito	15 de mayo de 2025
CP008	Éxito	15 de mayo de 2025
CP009	Éxito	15 de mayo de 2025
CP010	Éxito	15 de mayo de 2025
CP011	Éxito	15 de mayo de 2025
CP012	Éxito	15 de mayo de 2025
CP013	Éxito	15 de mayo de 2025
CP014	Éxito	15 de mayo de 2025
CP015	Éxito	15 de mayo de 2025
CP016	Éxito	15 de mayo de 2025
CP017	Éxito	15 de mayo de 2025
CP018	Éxito	15 de mayo de 2025
CP019	Éxito	15 de mayo de 2025
CP020	Éxito	15 de mayo de 2025

CP021	Éxito	15 de mayo de 2025
CP022	Éxito	15 de mayo de 2025
CP023	Éxito	15 de mayo de 2025
CP024	Éxito	15 de mayo de 2025
CP025	Éxito	15 de mayo de 2025
CP026	Éxito	15 de mayo de 2025
CP027	Éxito	15 de mayo de 2025
CP028	Éxito	15 de mayo de 2025
CP029	Éxito	15 de mayo de 2025
CP030	Éxito	15 de mayo de 2025
CP031	Éxito	15 de mayo de 2025
CP032	Éxito	15 de mayo de 2025
CP033	Éxito	15 de mayo de 2025
CP034	Éxito	15 de mayo de 2025
CP035	Éxito	15 de mayo de 2025
CP036	Éxito	15 de mayo de 2025
CP037	Éxito	15 de mayo de 2025
CP038	Éxito	15 de mayo de 2025
CP039	Éxito	15 de mayo de 2025
CP040	Éxito	15 de mayo de 2025
CP041	Éxito	15 de mayo de 2025
CP042	Éxito	15 de mayo de 2025
CP043	Éxito	15 de mayo de 2025

CP044	Éxito	15 de mayo de 2025
CP045	Éxito	15 de mayo de 2025
CP046	Éxito	15 de mayo de 2025
CP047	Éxito	15 de mayo de 2025
CP048	Éxito	15 de mayo de 2025
CP049	Éxito	15 de mayo de 2025
CP050	Éxito	15 de mayo de 2025
CP051	Éxito	15 de mayo de 2025
CP052	Éxito	15 de mayo de 2025
CP053	Éxito	15 de mayo de 2025
CP054	Éxito	15 de mayo de 2025
CP055	Éxito	15 de mayo de 2025
CP056	Éxito	15 de mayo de 2025
CP057	Éxito	15 de mayo de 2025
CP058	Éxito	15 de mayo de 2025