

Grado en Ingeniería de Computadores – Curso 2021-22

Ingeniería del Software: Laboratorio

PECL2 Pruebas y métricas de mantenimiento

Enunciado:

Se trata de un trabajo individual, no se puede hacer en grupo. Consiste en realizar tres ejercicios independientes.

Contenido

Ejercicio 1. Pruebas unitarias y de integración.....	2
PARTE A. PRUEBAS UNITARIAS.....	2
Clase SecuenciaADN:.....	3
PARTE B. PRUEBAS DE INTEGRACIÓN.....	9
Ejercicio 2. Pruebas unitarias con objetos simulados (mocks).....	14
Caso 1 del método solicitarPrestamoPersonal:.....	18
Caso 2 del método solicitarPrestamoPersonal:.....	19
Caso 3 del método solicitarPrestamoPersonal:.....	20
Caso 4 del método solicitarPrestamoPersonal:.....	21
Caso 5 del método solicitarPrestamoPersonal:.....	22
Caso 6 del método solicitarPrestamoPersonal:.....	23
Caso 7 del método solicitarPrestamoPersonal:.....	24
Caso 8 del método solicitarPrestamoPersonal:.....	25
Ejercicio 3. Métricas de mantenimiento del software:.....	26
Ejecución del programa JavaNCSS:.....	26
Ejecución del programa CKJM:.....	29
Ejecución del programa Dependency Finder:.....	30
Forma de entrega:.....	35
Criterios de valoración.....	35
Anexo. Código fuente para el Ejercicio 1.....	36

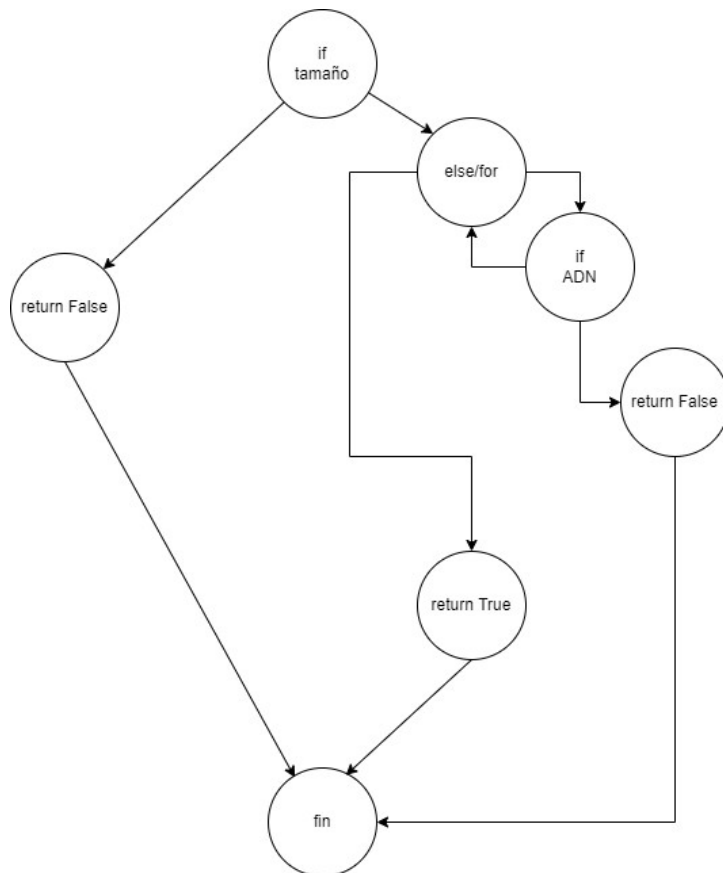
Ejercicio 1. Pruebas unitarias y de integración

PARTE A. PRUEBAS UNITARIAS

1. Calcular la **complejidad ciclomática** de los métodos de la clase *SecuenciaADN* (ver código fuente en Anexo), dibujando previamente un grafo para cada método.

Clase SecuenciaADN:

- Método esValida:

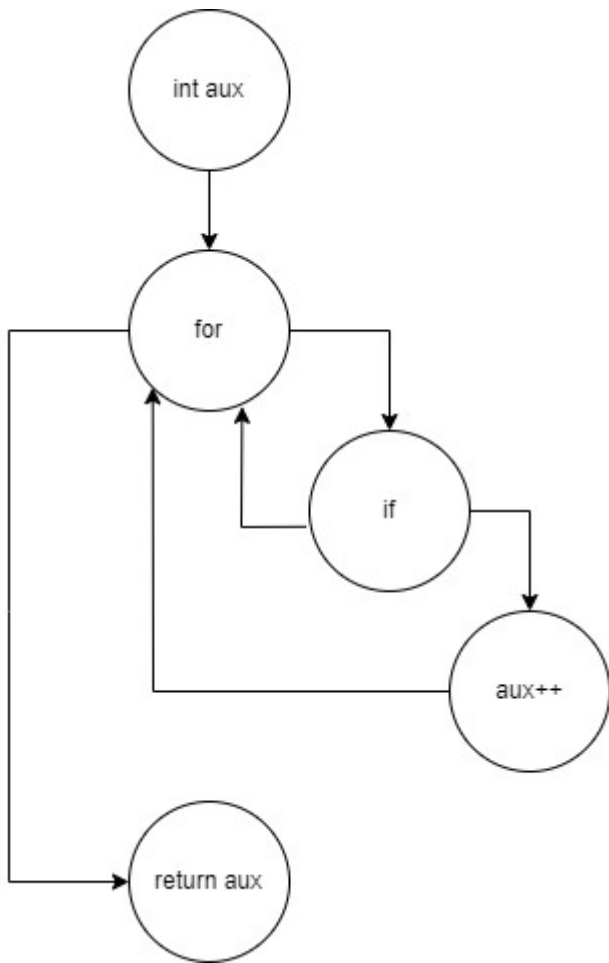


Cálculo: 9 aristas - 7 nodos + 2 = 4.

Hay que preparar 4 casos de prueba:

- Uno para cuando el tamaño de la variable secuencia sea 0, retornando False.
 - testEsValida_Sec_0
- Uno para cuando el tamaño de la secuencia sea ≥ 1 y tenga caracteres válidos, retornando True.
 - testEsValida_Validos
- Uno para cuando el tamaño de la secuencia sea ≥ 1 y tenga algún carácter inválido, retornando False.
 - testEsValida_Invalidos
- Uno para cuando el tamaño de la secuencia sea ≥ 1 y no entre en el bucle for, retornando True. ESTE CAMINO ES IMPOSIBLE EN LA PRÁCTICA, PERO HAY QUE CONTARLO.
 - testEsValida_No_For

- Método contar:



Cálculo: 6 aristas - 5 nodos + 2 = 3.

Hay que preparar 3 casos de prueba:

- Uno para cuando no se cumple la condición for, y se retornaría con `aux = 0`.
 - o `testContar_No_For`
- Uno para cuando se cumple el for, pero no se cumplen los if, retornando `aux = 0`.
 - o `testContar_No_Encontrado`
- Uno para cuando se cumple el for y algún if, retornando `[aux != 0]`.
 - o `testContar_Encontrado`

2. Diseñar casos de prueba para cada uno de los métodos de la clase *SecuenciaADN* aplicando la técnica de **caminos básicos**, con tantos casos de prueba como indique la complejidad ciclomática.

No	Nombre/ identificador	Descripción	Pre- condiciones	Entradas	Pasos	Resultados Esperados	Resultados de error
1	testEsValida_Sec_0	Se prueba que el resultado sea False al tener tamaño 0 secuencia.	Instanciar Secuencia con String() [Tamaño 0]	-	Llamar SecuenciaADN.esValida()	False	True
2	testEsValida_Validos	Se prueba que el resultado sea True al meter una secuencia válida.	Instanciar Secuencia con String("") [ENTRADA]	ACGTT TTTT TAGCC CCCA	Llamar SecuenciaADN.esValida()	True	False
3	testEsValida_Invalidos	Se prueba que el resultado sea False al meter un carácter inválido.	Instanciar Secuencia con String("") [ENTRADA]	ACGTT TTTT ZTAGC CCCA	Llamar SecuenciaADN.esValida()	False	True
4	testEsValida_No_For	Se prueba que siempre se entra en el for.	Instanciar Secuencia con String("") [ENTRADA]	A	Llamar SecuenciaADN.esValida()	True	False
5	testContar_No_For	Se prueba la secuencia con tamaño 0.	Instanciar Secuencia con String() [Tamaño 0], nucleotido = 'A'	-	Llamar SecuenciaADN.esValida()	0	!=0
6	testContar_No_Encontrado	Se prueba el conteo con un carácter no presente en la secuencia.	Instanciar Secuencia con String() [ENTRADA], nucleotido = 'A'	CGTTT TTTTZ TGCCC C	Llamar SecuenciaADN.esValida()	0	!=0
7	testContar_Encontrado	Se prueba el conteo con un carácter presente en la secuencia.	Instanciar Secuencia con String() [ENTRADA], nucleotido = 'A'	CGTTT TATTT ZTGCC ACCA	Llamar SecuenciaADN.esValida()	3	!=3

Los casos de prueba:

- 1,2,3,4 – Método esValida
- 5,6,7 – Método contar

3. Crear un proyecto nuevo y copiar el código de la clase *SecuenciaADN*. Programar con JUnit 5 las **pruebas unitarias** necesarias para cubrir todos los caminos básicos, y ejecutar todos los casos de prueba.

Proyecto adjuntado en la entrega nombrado como cadenaADN.zip

- Método testEsValida_Sec_0

```
/**
 * Tests of esValida method, of class SecuenciaADN.
 */
@org.junit.jupiter.api.Test
public void testEsValida_Sec_0() {
    System.out.println("testEsValida_Sec_0");

    SecuenciaADN instance = new SecuenciaADN(new String());

    boolean expectedResult = false;
    boolean result = instance.esValida();
    assertEquals(expectedResult, result);
}
```

- Método testEsValida_Validos

```
@org.junit.jupiter.api.Test
public void testEsValida_Validos() {
    System.out.println("testEsValida_Validos");

    SecuenciaADN instance = new SecuenciaADN(new String("ACGTTTTTTTTTAGCCCCCA"));

    boolean expectedResult = true;
    boolean result = instance.esValida();
    assertEquals(expectedResult, result);
}
```

- Método testEsValida_Invalidos

```
@org.junit.jupiter.api.Test
public void testEsValida_Invalidos() {
    System.out.println("testEsValida_Invalidos");

    SecuenciaADN instance = new SecuenciaADN(new String("ACGTTTTTTTTZTAGCCCCCA"));

    boolean expectedResult = false;
    boolean result = instance.esValida();
    assertEquals(expectedResult, result);
}
```

- Método testEsValida_No_For

```
@org.junit.jupiter.api.Test
public void testEsValida_No_For() {
    System.out.println("testEsValida_No_For");

    SecuenciaADN instance = new SecuenciaADN(new String("A"));

    boolean expectedResult = true;
    boolean result = instance.esValida();
    assertEquals(expectedResult, result);
}
```

- Método testContar_No_For

```
/**
 * Test of contar method, of class SecuenciaADN.
 */
@org.junit.jupiter.api.Test
public void testContar_No_For() {
    System.out.println("testContar_No_For");

    char nucleotido = 'A';

    SecuenciaADN instance = new SecuenciaADN(new String(""));

    int expectedResult = 0;
    int result = instance.contar(nucleotido);

    assertEquals(expectedResult, result);
}
```

- Método testContar_No_Encontrado

```
@org.junit.jupiter.api.Test
public void testContar_No_Encontrado() {
    System.out.println("testContar_No_Encontrado");

    char nucleotido = 'A';

    SecuenciaADN instance = new SecuenciaADN(new String("CGTTTTTTTGTGCCCC"));

    int expectedResult = 0;
    int result = instance.contar(nucleotido);

    assertEquals(expectedResult, result);
}
```


- Método testContar_Encontrado

```
@org.junit.jupiter.api.Test
public void testContar_Encontrado() {
    System.out.println("testContar_Encontrado");

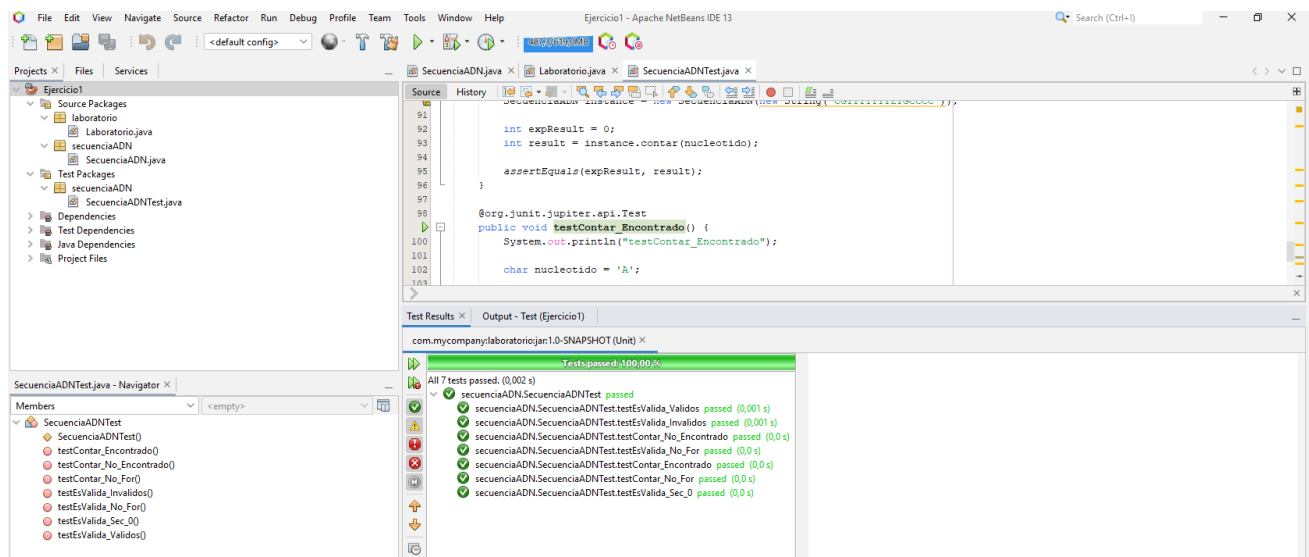
    char nucleotido = 'A';

    SecuenciaADN instance = new SecuenciaADN(new String("CGTTTTATTTZTGCCACCA"));

    int expectedResult = 3;
    int result = instance.contar(nucleotido);

    assertEquals(expectedResult, result);
}
```

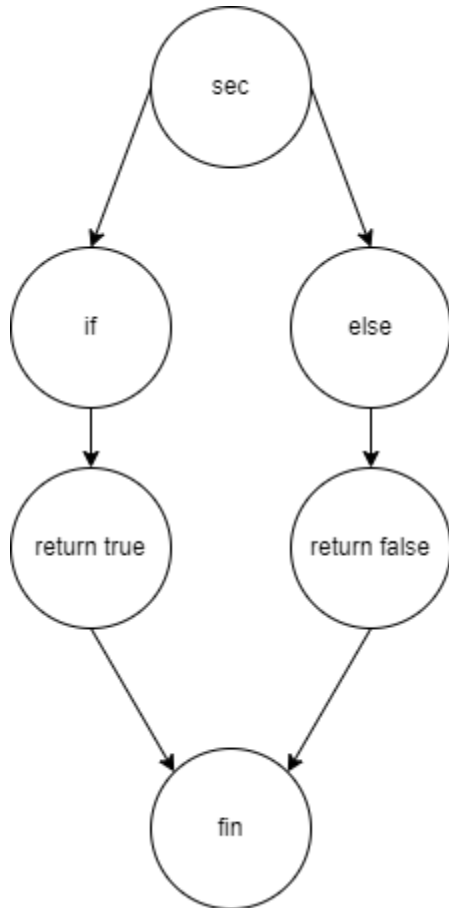
Ejecución de las pruebas:



PARTE B. PRUEBAS DE INTEGRACIÓN

4. Calcular la **complejidad ciclomática** del método *analizarADN* de la clase Laboratorio (ver código fuente en Anexo), dibujando previamente un grafo.

- Método analizarADN:



Cálculo: $6 \text{ aristas} - 6 \text{ nodos} + 2 = 2$.

Hay que preparar 2 casos de prueba:

- Uno para cuando se cumple la condición if, y se retornaría True.
 - o testAnalizarADN_valido
- Uno para cuando no se cumple el if, retornando False.
 - o testAnalizarADN_no_valido

5. Diseñar casos de prueba para el método *analizarADN* aplicando la técnica de **caminos básicos**, con tantos casos de prueba como indique la complejidad ciclomática.

No	Nombre / identificador	Descripción	Pre-condiciones	Entradas	Pasos	Resultados Esperados	Resultados de error
1	testAnalizarADN_valido	Se prueba que el resultado sea True al meter una secuencia válida.	Instanciar Secuencia con adn	ACGTT TTTTTT AGCCC CCA	Llamar SecuenciaADN.es Valida()	True	False
2	testAnalizarADN_no_valido	Se prueba que el resultado sea False al meter una secuencia inválida.	Instanciar Secuencia con adn	ACGTT TTTTTT AZCCC CCA	Llamar SecuenciaADN.es Valida()	False	True

Los casos de prueba:

- 1,2 – Método analizarADN

6. Crear un proyecto nuevo y copiar el código de la clase *Laboratorio*. Programar con JUnit 5 las **pruebas de integración** de la clase *Laboratorio* con la clase *SecuenciaADN*, necesarias para cubrir todos los caminos básicos, y ejecutar todos los casos de prueba.

Proyecto adjuntado en la entrega nombrado como laboratorio.zip

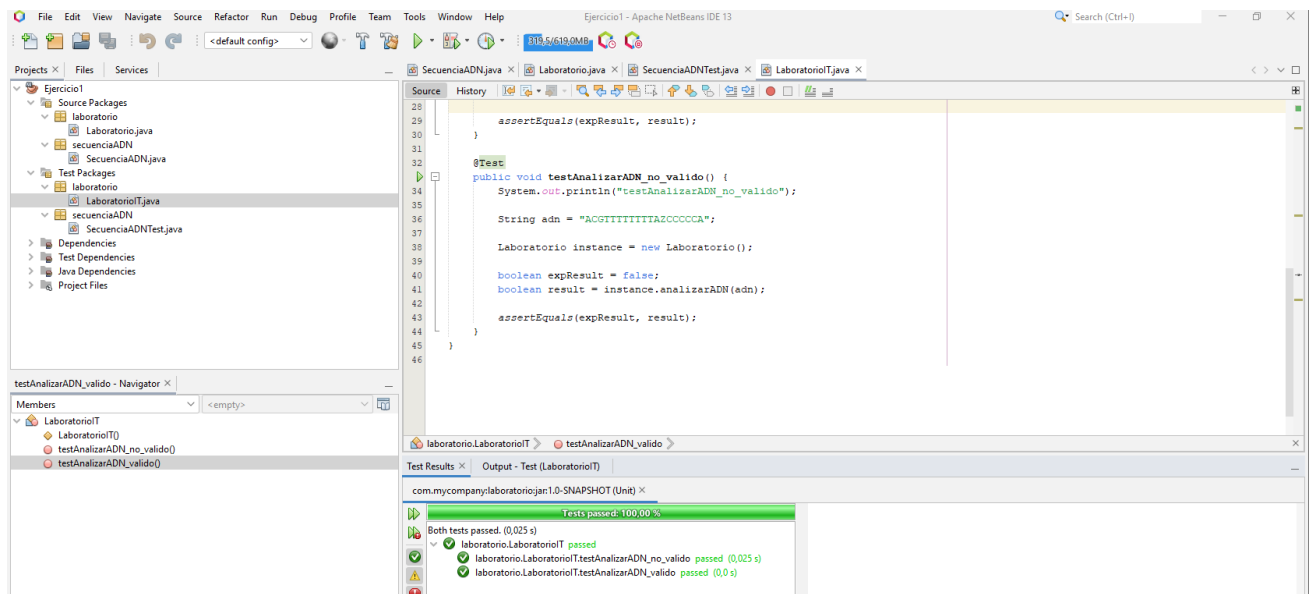
- Método testAnalizarADN_valido:

```
public class LaboratorioIT {  
  
    public LaboratorioIT() {  
    }  
  
    /**  
     * Tests of analizarADN method, of class Laboratorio.  
     */  
    @Test  
    public void testAnalizarADN_valido() {  
        System.out.println("testAnalizarADN_valido");  
  
        String adn = "ACGTTTTTTTAGCCCCCA";  
  
        Laboratorio instance = new Laboratorio();  
  
        boolean expResult = true;  
        boolean result = instance.analizarADN(adn);  
  
        assertEquals(expResult, result);  
    }  
}
```

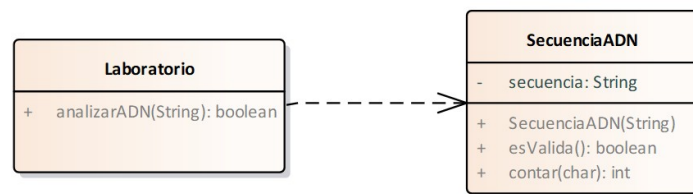
- Método testAnalizarADN_no_valido:

```
@Test  
public void testAnalizarADN_no_valido() {  
    System.out.println("testAnalizarADN_no_valido");  
  
    String adn = "ACGTTTTTTTAZCCCCCA";  
  
    Laboratorio instance = new Laboratorio();  
  
    boolean expResult = false;  
    boolean result = instance.analizarADN(adn);  
  
    assertEquals(expResult, result);  
}
```

Ejecución de las pruebas:

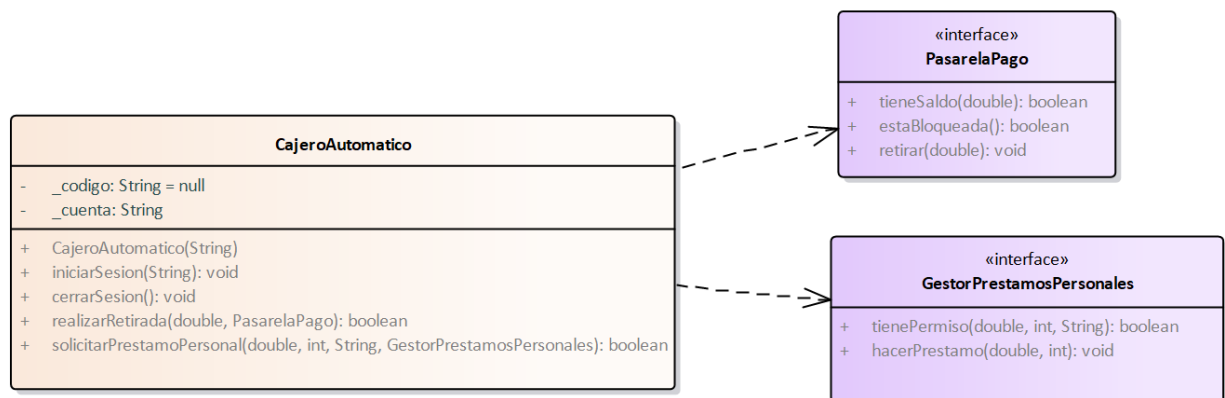


En el siguiente diagrama se muestran las clases del ejercicio.



Ejercicio 2. Pruebas unitarias con objetos simulados (mocks)

Se debe partir del código finalizado de la práctica P6 de la asignatura, y añadir una nueva clase denominada *GestorPrestamosPersonales*, que será utilizada por *CajeroAutomático*. Esta nueva clase aún no está desarrollada, por lo que se especificará su comportamiento mediante una interfaz.



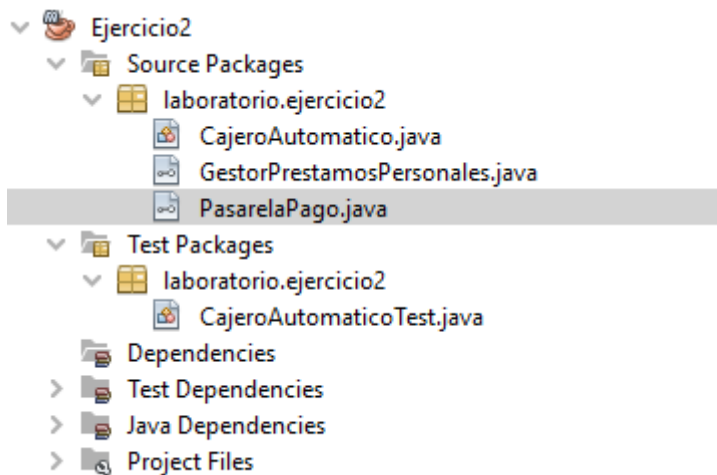
El gestor de préstamos permitirá desde el cajero solicitar un préstamo personal (especificando la cantidad, el plazo de amortización y la fecha de solicitud) con las siguientes reglas:

- No se puede pedir préstamos de más de 3000 EUR o menos de 1000 EUR.
- No se pueden pedir préstamos a más de 12 meses ni a menos de 6 meses.
- No se puede conceder más de un préstamo dentro del mismo año (por eso se proporciona la fecha de solicitud al gestor de préstamos para que lo compruebe).

Se pide lo siguiente:

1. Codificar la interfaz *GestorPrestamosPersonales* con dos métodos:

- tienePermiso*, que recibe como parámetros la cantidad pedida, el plazo de amortización en meses y la fecha de solicitud (por simplicidad se puede utilizar un String como fecha). El tipo de resultado es boolean para indicar si se tiene o no permiso para obtener el préstamo.
- hacerPrestamo*, que recibe como parámetros la cantidad pedida y el plazo, y realiza el préstamo solicitado.



```
1 package laboratorio.ejercicio2;
2
3 /**
4  *
5  * @author usuario
6  */
7 public interface GestorPrestamosPersonales {
8
9     /**
10      * tienePermiso, que recibe como parámetros
11      * la cantidad pedida, el plazo de amortización en meses y la fecha de solicitud
12      * (por simplicidad se puede utilizar un String como fecha).
13      * El tipo de resultado es boolean para indicar si se tiene o no permiso para obtener el préstamo.
14      */
15     public boolean tienePermiso(double cantidad, int plazoPagoMeses, String fechaSolicitud);
16
17     /**
18      * hacerPrestamo, que recibe como parámetros
19      * la cantidad pedida y el plazo,
20      * y realiza el préstamo solicitado.
21      */
22     public void hacerPrestamo(double cantidad, int plazoPagoMeses);
23 }
```

2. Codificar el método *solicitarPrestamoPersonal* en la clase *CajeroAutomático*, que utilice los métodos *tienePermiso* y *hacerPrestamo* de la interfaz *GestorPrestamosPersonales*.

Este método tiene como parámetros:

- La cantidad solicitada de préstamo (double)
- El plazo de amortización (int)
- La fecha de solicitud (String)
- El gestor de préstamos personales (de la clase *GestorPrestamosPersonales*) que se encargará de gestionar esa solicitud de préstamo


```

/*
2.Codificar el método solicitarPrestamoPersonal en la clase CajeroAutomático, que utilice los
métodos tienePermiso y hacerPrestamo de la interfaz GestorPrestamosPersonales.
Este método tiene como parámetros:
* La cantidad solicitada de préstamo (doublé)
* El plazo de amortización (int)
* La fecha de solicitud (String)
* El gestor de préstamos personales (de la clase GestorPrestamosPersonales) que se encargará de gestionar esa solicitud de pr

Condiciones:
a. No se puede pedir préstamos de más de 3000 EUR o menos de 1000 EUR.
b. No se pueden pedir préstamos a más de 12 meses ni a menos de 6 meses.
c. No se puede conceder más de un préstamo dentro del mismo año
(por eso se proporciona la fecha de solicitud al gestor de préstamos para que lo compruebe).

*/

public boolean solicitarPrestamoPersonal(double cantidadPrestamo, int plazoPagoMeses, String fecha, GestorPrestamosPersonales gestor){
    // Comprobaciones
    assert (_cuenta != null); // Comprobación cuenta operativa
    if (Objects.isNull(gestor)) // Comprobación gestor
        return false;
    if ((cantidadPrestamo>3000) || (cantidadPrestamo<1000)) //Comprobación cantidad válida
        return false;
    if ((plazoPagoMeses>12) || (plazoPagoMeses<6)) //Comprobación meses a pagar.
        return false;
    if (!gestor.tienePermiso(cantidadPrestamo, plazoPagoMeses, fecha)) // Comprobación c para el préstamo.
        return false;

    // Realización
    gestor.hacerPrestamo(cantidadPrestamo, plazoPagoMeses);
    return true;
}

```

Se ha usado el método `Objects.isNull(Object)` para comprobar que el gestor no sea null.

3. Diseñar los casos de prueba para el método *solicitarPrestamoPersonal* del cajero, considerando la técnica de clases de equivalencia. Realizar una tabla con los casos válidos y no válidos.

Método `solicitarPrestamoPersonal`:

Parámetros	Clases válidas	Clases no válidas
cantidadPrestamo	(1) $1000 \leq \text{cantidadPrestamo} \leq 3000$	(2) $\text{cantidadPrestamo} < 1000$ (3) $\text{cantidadPrestamo} > 3000$
plazoPagoMeses	(4) $6 \leq \text{plazoPagoMeses} \leq 12$	(5) $\text{plazoPagoMeses} < 6$ (6) $\text{plazoPagoMeses} > 12$
fecha	(7) "fecha" válida y préstamo no pedido	(8) "fecha" no válida (9) "fecha" préstamo pedido anteriormente
gestor	(10) Objeto "gestor" instanciado de la clase <code>GestorPrestamosPersonales</code>	(11) Objeto "gestor" con valor null

Casos válidos:

- (1) (4) (7) (10): $\text{cantidadPrestamo} = 1500$, $\text{plazoPagoMeses} = 11$, $\text{fecha} = \text{"15/07/2021"}$, $\text{gestor} \neq \text{null}$

Casos no válidos:

- (2) (4) (7) (10): cantidadPrestamo = 900, plazoPagoMeses = 11, fecha = "15/07/2021", gestor != null
- (3) (4) (7) (10): cantidadPrestamo = 3300, plazoPagoMeses = 11, fecha = "15/07/2021", gestor != null
- (1) (5) (7) (10): cantidadPrestamo = 1500, plazoPagoMeses = 4, fecha = "15/07/2021", gestor != null
- (1) (6) (7) (10): cantidadPrestamo = 1500, plazoPagoMeses = 13, fecha = "15/07/2021", gestor != null
- (1) (4) (8) (10): cantidadPrestamo = 1500, plazoPagoMeses = 11, fecha = "hola", gestor != null
- (1) (4) (9) (10): cantidadPrestamo = 1500, plazoPagoMeses = 11, fecha = "18/09/2021", gestor != null
- (1) (4) (7) (11): cantidadPrestamo = 1500, plazoPagoMeses = 11, fecha = "15/07/2021", gestor == null

Se han optado por estas clases de equivalencia, debido a que hay que tener en cuenta que los parámetros cuenten con las siguientes condiciones impuestas:

- cantidadPrestamo debe tener un valor double o fraccionario entre 1000 y 3000.
- plazoPagoMeses debe tener un valor int o entero entre 6 y 12.
- fecha debe tener un valor String con un formato de fecha válido dd/mm/yyyy y que en ese mismo año no se haya solicitado un prestamo por el mismo usuario.
- gestor debe estar instanciado como objeto de la clase GestorPrestamosPersonales y por lo tanto no puede ser null.

4. Implementar, utilizando JUnit e EasyMock, los casos de pruebas establecidos en el apartado anterior, utilizando objetos simulados (mock) para *GestorPrestamosPersonales*.

Caso 1 del método solicitarPrestamoPersonal:

(1) (4) (7) (10): cantidadPrestamo = 1500, plazoPagoMeses = 11, fecha = "15/07/2021", gestor != null

```
111  /*
112  Casos válidos:
113
114  * (1) (4) (7) (10): cantidadPrestamo = 1500, plazoPagoMeses = 11, fecha = "15/07/2021", gestor != null
115  */
116  @Test
117  public void solicitarPrestamoPersonal() {
118      System.out.println("solicitarPrestamoPersonal: caso 1 cantidadPrestamo = 1500, plazoPagoMeses = 11, fecha = '15/07/2021', gestor!=null");
119      /*(1) Se crea el objeto de la clase a probar y un mock para simular la clase PasarelaPago*/
120
121      unCajeroAProbar = new CajeroAutomatico("1111111111");
122      unCajeroAProbar.iniciarSesion("1234");
123      gestorMock = createMock(GestorPrestamosPersonales.class);
124
125      /*(2) En estado de "grabación", se le dice al objeto Simulado las llamadas que debe esperar y cómo responder a ellas.*/
126      expect(gestorMock.tienePermiso(1500, 11, "15/07/2021")).andReturn(true);
127
128      gestorMock.hacerPrestamo(1500, 11);
129      /*(3) Ahora, el objeto simulado comienza a esperar las llamadas*/
130      replay(gestorMock);
131
132      /*(4) Se programa la prueba del objeto de la clase a probar*/
133      boolean result = unCajeroAProbar.solicitarPrestamoPersonal(1500,11,"15/07/2021",gestorMock);
134      assertTrue(result);
135
136      /*(5) Forzamos a que la ausencia de todas las llamadas previstas sea un error también:*/
137      verify(gestorMock);
138
139      /*(6) Se ejecutan instrucciones necesarias de finalización de la prueba y se resetea el mock*/
140      unCajeroAProbar.cerrarSesion();
141      reset(gestorMock);
142      System.out.println("Fin del caso de prueba 1");
143  }
```

Caso 2 del método solicitarPrestamoPersonal:

(2) (4) (7) (10): cantidadPrestamo = 900, plazoPagoMeses = 11, fecha = "15/07/2021", gestor != null

```
145      /*
146      Casos no válidos:
147
148      * (2) (4) (7) (10): cantidadPrestamo = 900, plazoPagoMeses = 11, fecha = "15/07/2021", gestor != null
149      */
150      @Test
151      public void solicitarPrestamoPersonal2() {
152          System.out.println("solicitarPrestamoPersonal: caso 1 cantidadPrestamo = 900, plazoPagoMeses = 11, fecha = '15/07/2021', gestor!=null");
153          /*(1) Se crea el objeto de la clase a probar y un mock para simular la clase PasarelaPago*/
154
155          unCajeroAProbar = new CajeroAutomatico("1111111111");
156          unCajeroAProbar.iniciarSesion("1234");
157          gestorMock = createMock(GestorPrestamosPersonales.class);
158
159          /*(2) En estado de "grabación", se le dice al objeto Simulado las llamadas que debe esperar y cómo responder a ellas.*/
160          // No se debería realizar ninguna llamada a gestor, debido a que se ha programado la comprobación en el
161          // método CajeroAutomatico.solicitarPrestamoPersonal()
162
163          /*(3) Ahora, el objeto simulado comienza a esperar las llamadas*/
164          replay(gestorMock);
165
166          /*(4) Se programa la prueba del objeto de la clase a probar*/
167          boolean result = unCajeroAProbar.solicitarPrestamoPersonal(900,11,"15/07/2021",gestorMock);
168          assertFalse(result);
169
170          /*(5) Forzamos a que la ausencia de todas las llamadas previstas sea un error también:*/
171          verify(gestorMock);
172
173          /*(6) Se ejecutan instrucciones necesarias de finalización de la prueba y se resetea el mock*/
174          unCajeroAProbar.cerrarSesion();
175          reset(gestorMock);
176          System.out.println("Fin del caso de prueba 1");
177      }
```

Nota: No se debe programar ningún expect en el gestorMock, debido a que la comprobación de la cantidadPrestamo se realiza en el método CajeroAutomatico.solicitarPrestamoPersonal(...)

Caso 3 del método solicitarPrestamoPersonal:

(3) (4) (7) (10): cantidadPrestamo = 3300, plazoPagoMeses = 11, fecha = "15/07/2021", gestor != null

```
/*
    Casos no válidos:

    * (3) (4) (7) (10): cantidadPrestamo = 3300, plazoPagoMeses = 11, fecha = "15/07/2021", gestor != null
*/
@Test
public void solicitarPrestamoPersonal3() {
    System.out.println("solicitarPrestamoPersonal3: caso 1 cantidadPrestamo = 3300, plazoPagoMeses = 11, fecha = '15/07/2021', gestor != null");
    /* (1) Se crea el objeto de la clase a probar y un mock para simular la clase PasarelaPago */

    unCajeroAProbar = new CajeroAutomatico("1111111111");
    unCajeroAProbar.iniciarSesion("1234");
    gestorMock = createMock(GestorPrestamosPersonales.class);

    /* (2) En estado de "grabación", se le dice al objeto Simulado las llamadas que debe esperar y cómo responder a ellas. */
    // No se debería realizar ninguna llamada a gestor, debido a que se ha programado la comprobación en el
    // método CajeroAutomatico.solicitarPrestamoPersonal()

    /* (3) Ahora, el objeto simulado comienza a esperar las llamadas */
    replay(gestorMock);

    /* (4) Se programa la prueba del objeto de la clase a probar */
    boolean result = unCajeroAProbar.solicitarPrestamoPersonal(3300, 11, "15/07/2021", gestorMock);
    assertFalse(result);
    /* (5) Forzamos a que la ausencia de todas las llamadas previstas sea un error también */
    verify(gestorMock);

    /* (6) Se ejecutan instrucciones necesarias de finalización de la prueba y se resetea el mock */
    unCajeroAProbar.cerrarSesion();
    reset(gestorMock);
    System.out.println("Fin del caso de prueba 1");
}
```

Nota: No se debe programar ningún expect en el gestorMock, debido a que la comprobación de la cantidadPrestamo se realiza en el método CajeroAutomatico.solicitarPrestamoPersonal(...)

Caso 4 del método solicitarPrestamoPersonal:

(1) (5) (7) (10): cantidadPrestamo = 1500, plazoPagoMeses = 4, fecha = "15/07/2021", gestor != null

Nota: No se debe programar ningún expect en el gestorMock, debido a que la comprobación de la cantidadPrestamo se realiza en el método CajeroAutomatico.solicitarPrestamoPersonal(...)

```
/*
    Casos no válidos:

    * (1) (5) (7) (10): cantidadPrestamo = 1500, plazoPagoMeses = 4, fecha = "15/07/2021", gestor != null
*/
@Test
public void solicitarPrestamoPersonal4() {
    System.out.println("solicitarPrestamoPersonal4: caso 4 cantidadPrestamo = 1500, plazoPagoMeses = 4, fecha = '15/07/2021', gestor!="
    /*(1) Se crea el objeto de la clase a probar y un mock para simular la clase PasarelaPago*/

    unCajeroAProbar = new CajeroAutomatico("1111111111");
    unCajeroAProbar.iniciarSesion("1234");
    gestorMock = createMock(GestorPrestamosPersonales.class);

    /*(2) En estado de "grabación", se le dice al objeto Simulado las llamadas que debe esperar y cómo responder a ellas.*/
    // No se debería realizar ninguna llamada a gestor, debido a que se ha programado la comprobación en el
    // método CajeroAutomatico.solicitarPrestamoPersonal()

    /*(3) Ahora, el objeto simulado comienza a esperar las llamadas*/
    replay(gestorMock);

    /*(4) Se programa la prueba del objeto de la clase a probar*/
    boolean result = unCajeroAProbar.solicitarPrestamoPersonal(1500,4,"15/07/2021",gestorMock);
    assertFalse(result);

    /*(5) Forzamos a que la ausencia de todas las llamadas previstas sea un error también:*/
    verify(gestorMock);

    /*(6) Se ejecutan instrucciones necesarias de finalización de la prueba y se resetea el mock*/
    unCajeroAProbar.cerrarSesion();
    reset(gestorMock);
    System.out.println("Fin del caso de prueba 1");
}
```

Caso 5 del método solicitarPrestamoPersonal:

(1) (6) (7) (10): cantidadPrestamo = 1500, plazoPagoMeses = 13, fecha = "15/07/2021", gestor != null

Nota: No se debe programar ningún expect en el gestorMock, debido a que la comprobación de la cantidadPrestamo se realiza en el método CajeroAutomatico.solicitarPrestamoPersonal(...)

```
/*
    Casos no válidos:

    * (1) (6) (7) (10): cantidadPrestamo = 1500, plazoPagoMeses = 13, fecha = "15/07/2021", gestor != null
*/
@Test
public void solicitarPrestamoPersonal5() {
    System.out.println("solicitarPrestamoPersonal5: caso 5 cantidadPrestamo = 1500, plazoPagoMeses = 13, fecha = '15/07/2021', gestor != null");
    /*(1) Se crea el objeto de la clase a probar y un mock para simular la clase PasarelaPago*/

    unCajeroAProbar = new CajeroAutomatico("1111111111");
    unCajeroAProbar.iniciarSesion("1234");
    gestorMock = createMock(GestorPrestamosPersonales.class);

    /*(2) En estado de "grabación", se le dice al objeto Simulado las llamadas que debe esperar y cómo responder a ellas.*/
    // No se debería realizar ninguna llamada a gestor, debido a que se ha programado la comprobación en el
    // método CajeroAutomatico.solicitarPrestamoPersonal()

    /*(3) Ahora, el objeto simulado comienza a esperar las llamadas*/
    replay(gestorMock);

    /*(4) Se programa la prueba del objeto de la clase a probar*/
    boolean result = unCajeroAProbar.solicitarPrestamoPersonal(1500,13,"15/07/2021",gestorMock);
    assertFalse(result);

    /*(5) Forzamos a que la ausencia de todas las llamadas previstas sea un error también:*/
    verify(gestorMock);

    /*(6) Se ejecutan instrucciones necesarias de finalización de la prueba y se resetea el mock*/
    unCajeroAProbar.cerrarSesion();
    reset(gestorMock);
    System.out.println("Fin del caso de prueba 5");
}
```


Caso 6 del método solicitarPrestamoPersonal:

(1) (4) (8) (10): cantidadPrestamo = 1500, plazoPagoMeses = 11, fecha = "hola", gestor != null

```
/*
    Casos no válidos:

    * (1) (4) (8) (10): cantidadPrestamo = 1500, plazoPagoMeses = 11, fecha = "hola", gestor != null
*/
@Test
public void solicitarPrestamoPersonal6() {
    System.out.println("solicitarPrestamoPersonal6: caso 6 cantidadPrestamo = 1500, plazoPagoMeses = 11, fecha = 'hola', gestor!=null"
    /* (1) Se crea el objeto de la clase a probar y un mock para simular la clase PasarelaPago*/

    unCajeroAProbar = new CajeroAutomatico("1111111111");
    unCajeroAProbar.iniciarSesion("1234");
    gestorMock = createMock(GestorPrestamosPersonales.class);

    /* (2) En estado de "grabación", se le dice al objeto Simulado las llamadas que debe esperar y cómo responder a ellas.*/
    expect(gestorMock.tienePermiso(1500, 11, "hola")).andReturn(false);
    /* (3) Ahora, el objeto simulado comienza a esperar las llamadas*/
    replay(gestorMock);

    /* (4) Se programa la prueba del objeto de la clase a probar*/
    boolean result = unCajeroAProbar.solicitarPrestamoPersonal(1500,11,"hola",gestorMock);
    assertFalse(result);

    /* (5) Forzamos a que la ausencia de todas las llamadas previstas sea un error también:*/
    verify(gestorMock);

    /* (6) Se ejecutan instrucciones necesarias de finalización de la prueba y se resetea el mock*/
    unCajeroAProbar.cerrarSesion();
    reset(gestorMock);
    System.out.println("Fin del caso de prueba 6");
}
```

Caso 7 del método solicitarPrestamoPersonal:

(1) (4) (9) (10): cantidadPrestamo = 1500, plazoPagoMeses = 11, fecha = "18/09/2021", gestor != null

NOTA: Se supone que el usuario ha pedido un prestamo en ese mismo año antes de ejecutar este caso de prueba, por eso debe salir false.

```
/*
Casos no válidos:

* (1) (4) (9) (10): cantidadPrestamo = 1500, plazoPagoMeses = 11, fecha = "18/09/2021", gestor != null
*/
@Test
public void solicitarPrestamoPersonal7() {
    System.out.println("solicitarPrestamoPersonal: caso 7 cantidadPrestamo = 1500, plazoPagoMeses = 11, fecha = '18/09/2021', gestor!=
    /*(1) Se crea el objeto de la clase a probar y un mock para simular la clase PasarelaPago*/

    unCajeroAProbar = new CajeroAutomatico("1111111111");
    unCajeroAProbar.iniciarSesion("1234");
    gestorMock = createMock(GestorPrestamosPersonales.class);

    /*(2) En estado de "grabación", se le dice al objeto Simulado las llamadas que debe esperar y cómo responder a ellas.*/
    expect(gestorMock.tienePermiso(1500, 11, "18/09/2021")).andReturn(false);

    /*(3) Ahora, el objeto simulado comienza a esperar las llamadas*/
    replay(gestorMock);

    /*(4) Se programa la prueba del objeto de la clase a probar*/
    boolean result = unCajeroAProbar.solicitarPrestamoPersonal(1500, 11, "18/09/2021", gestorMock);
    assertFalse(result);

    /*(5) Forzamos a que la ausencia de todas las llamadas previstas sea un error también:*/
    verify(gestorMock);

    /*(6) Se ejecutan instrucciones necesarias de finalización de la prueba y se resetea el mock*/
    unCajeroAProbar.cerrarSesion();
    reset(gestorMock);
    System.out.println("Fin del caso de prueba 7");
}
```

Caso 8 del método solicitarPrestamoPersonal:

(1) (4) (7) (11): cantidadPrestamo = 1500, plazoPagoMeses = 11, fecha = "15/07/2021", gestor == null

NOTA: Se supone que se ha pasado la variable "gestorMock" con valor Null, por eso debería retornar un false.

```
/*
    Casos no válidos:

    * (1) (4) (7) (11): cantidadPrestamo = 1500, plazoPagoMeses = 11, fecha = "15/07/2021", gestor == null
*/
@Test
public void solicitarPrestamoPersonal8() {
    System.out.println("solicitarPrestamoPersonal8: caso 8 cantidadPrestamo = 1500, plazoPagoMeses = 11, fecha = '15/07/2021', gestor="
    /*(1) Se crea el objeto de la clase a probar y un mock para simular la clase PasarelaPago*/

    unCajeroAProbar = new CajeroAutomatico("1111111111");
    unCajeroAProbar.iniciarSesion("1234");
    gestorMock = null;

    /*(2) En estado de "grabación", se le dice al objeto Simulado las llamadas que debe esperar y cómo responder a ellas.*/
    // No se debería realizar ninguna llamada a gestor, debido a que se ha programado la comprobación en el
    // método CajeroAutomatico.solicitarPrestamoPersonal()

    /*(3) Ahora, el objeto simulado comienza a esperar las llamadas*/
    // Comprobación de gestor == null

    /*(4) Se programa la prueba del objeto de la clase a probar*/

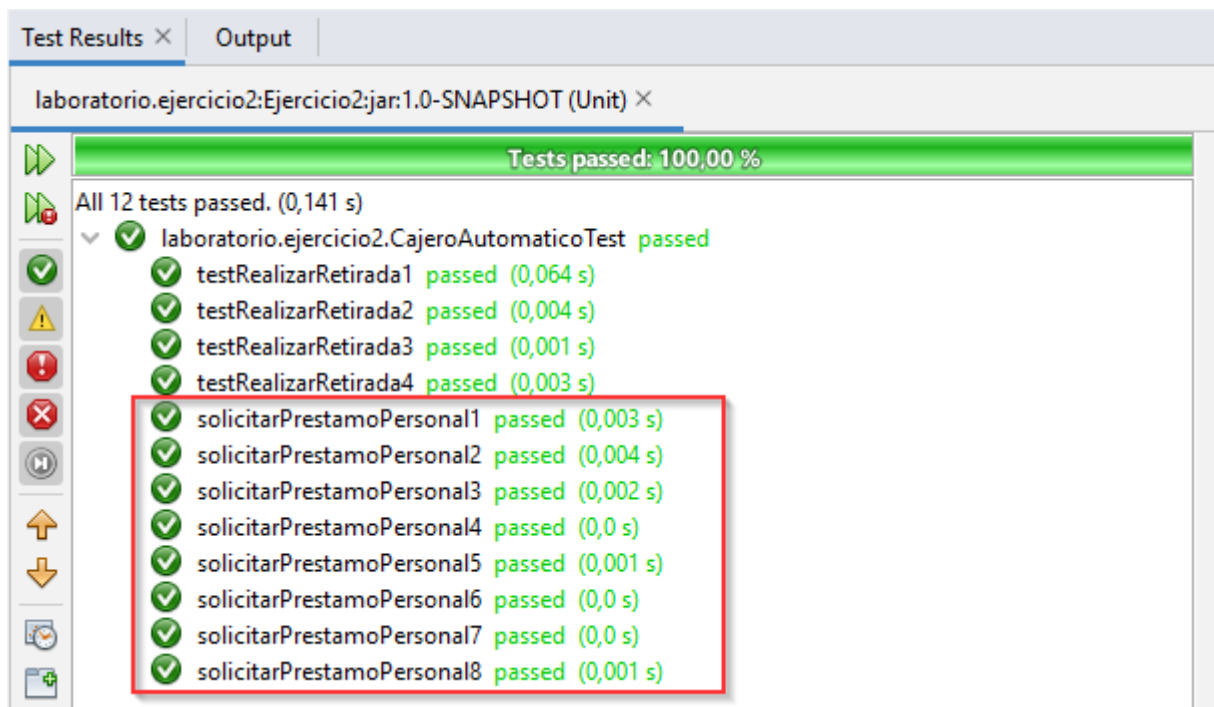
    boolean result = unCajeroAProbar.solicitarPrestamoPersonal(1500,11,"15/07/2021",gestorMock);
    assertFalse(result);

    /*(5) Forzamos a que la ausencia de todas las llamadas previstas sea un error también:*/
    // Comprobación de gestor == null

    /*(6) Se ejecutan instrucciones necesarias de finalización de la prueba y se resetea el mock*/
    unCajeroAProbar.cerrarSesion();

    System.out.println("Fin del caso de prueba 8");
}
```

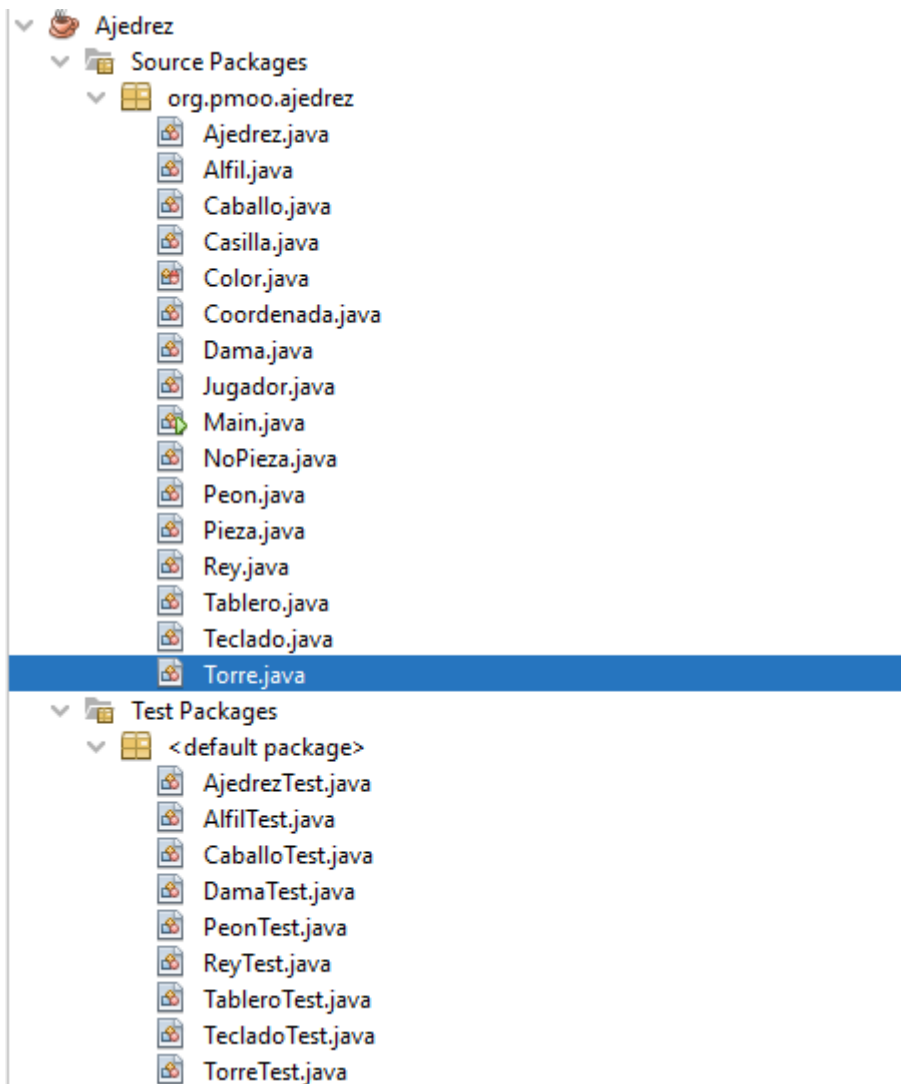
Ejecución de las pruebas:



Ejercicio 3. Métricas de mantenimiento del software:

1. Seleccionar un programa en Java, que tenga más de 10 clases, de un repositorio de código abierto (SourceForge.net, GoogleCode, etc.), e identificar las clases que pueden ser más problemáticas en mantenimiento utilizando como herramientas de análisis **JavaNCSS**, **CKJM** y **Dependency Finder**.

Se ha utilizado el proyecto: <https://github.com/Danelitos/Ajedrez>



Ejecución del programa JavaNCSS:

Comando:

```
java -classpath .;C:\javancss-32.53\lib\javancss.jar;C:\javancss-32.53\lib\ccl.jar;C:\javancss-32.53\lib\jhbasic.jar;C:\javancss-32.53\lib\javacc.jar javancss.Main -all -recursive -gui C:\Ajedrez\src\org\pmoo\ajedrez
```

```
C:\Windows\system32\cmd.exe - java -classpath .;C:\javancss-32.53\lib\javancss.jar;C:\javancss-32.53\lib\ccl.jar;C:\javancss-32.53\lib\jhbasic.jar;C:\jav...
Microsoft Windows [Versión 10.0.19044.1682]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\usuario>java -classpath .;C:\javancss-32.53\lib\javancss.jar;C:\javancss-32.53\lib\ccl.jar;C:\javancss-32.53\lib\jhbasic.jar;C:\javancss-32.53\lib\javacc.jar javancss.Main -all -recursive -gui C:\Ajedrez\src\org\pmoo\ajedrez
Warning: could not find image for 'JavancssFrame' application main frame.
```

JavaNCSS:

File Help

Packages Classes Methods

Tue, May 10, 2022 21:17:08 Europe/Madrid

Nr.	Classes	Functions	NCSS	Javadocs	Package
1	16	66	956	0	org.pmoo.ajedrez
	16	66	956	0	Total

Packages	Classes	Functions	NCSS	Javadocs	per
1.00	16.00	66.00	956.00	0.00	Project
	16.00	66.00	956.00	0.00	Package
		4.13	59.75	0.00	Class
			14.48	0.00	Function

JavaNCSS:

File Help

Packages Classes Methods

Tue, May 10, 2022 21:17:08 Europe/Madrid

Nr.	NCSS	Functions	Classes	Javadocs	Class
1	55	5	0	0	org.pmoo.ajedrez.Ajedrez
2	90	3	0	0	org.pmoo.ajedrez.Alfil
3	66	3	0	0	org.pmoo.ajedrez.Caballo
4	10	4	0	0	org.pmoo.ajedrez.Casilla
5	5	2	0	0	org.pmoo.ajedrez.Color
6	10	3	0	0	org.pmoo.ajedrez.Coordenada
7	150	3	0	0	org.pmoo.ajedrez.Dama
8	12	4	0	0	org.pmoo.ajedrez.Jugador
9	3	1	0	0	org.pmoo.ajedrez.Main
10	8	3	0	0	org.pmoo.ajedrez.NoPieza
11	35	3	0	0	org.pmoo.ajedrez.Peon
12	27	11	0	0	org.pmoo.ajedrez.Pieza
13	66	3	0	0	org.pmoo.ajedrez.Rey
14	310	12	0	0	org.pmoo.ajedrez.Tablero
15	12	3	0	0	org.pmoo.ajedrez.Teclado
16	78	3	0	0	org.pmoo.ajedrez.Torre

Average Object NCSS: 58.56

Average Object Functions: 4.13

Average Object Inner Classes: 0.00

Average Object Javadoc Comments: 0.00

Program NCSS: 956.00

JavaNCSS:

File Help

Packages Classes Methods

Tue, May 10, 2022 21:17:08 Europe/Madrid

Nr.	NCSS	CCN	JVDC	Function
1	3	1	0	org.pmoo.ajedrez.Ajedrez.Ajedrez()
2	4	2	0	org.pmoo.ajedrez.Ajedrez.getMiAjedrez()
3	15	4	0	org.pmoo.ajedrez.Ajedrez.jugarPartida()
4	15	2	0	org.pmoo.ajedrez.Ajedrez.crearJugadores()
5	12	3	0	org.pmoo.ajedrez.Ajedrez.turno()
6	6	2	0	org.pmoo.ajedrez.Alfil.Alfil(Color)
7	27	30	0	org.pmoo.ajedrez.Alfil.puedeMover(Coordenada, Coordenada, Casilla)
8	54	62	0	org.pmoo.ajedrez.Alfil.puedeComer(Coordenada, Coordenada, Casilla)
9	6	2	0	org.pmoo.ajedrez.Caballo.Caballo(Color)
10	19	26	0	org.pmoo.ajedrez.Caballo.puedeMover(Coordenada, Coordenada, Casilla)
11	38	54	0	org.pmoo.ajedrez.Caballo.puedeComer(Coordenada, Coordenada, Casilla)
12	2	1	0	org.pmoo.ajedrez.Casilla.Casilla()
13	2	1	0	org.pmoo.ajedrez.Casilla.imprimirPieza()
14	2	1	0	org.pmoo.ajedrez.Casilla.setPieza(Pieza)
15	2	1	0	org.pmoo.ajedrez.Casilla.getPieza()
16	2	1	0	org.pmoo.ajedrez.Color.Color(String)
17	2	1	0	org.pmoo.ajedrez.Color.getColor()
18	3	1	0	org.pmoo.ajedrez.Coordenada.Coordenada(int, int)
19	2	1	0	org.pmoo.ajedrez.Coordenada.getFila()
20	2	1	0	org.pmoo.ajedrez.Coordenada.getColumna()
21	6	2	0	org.pmoo.ajedrez.Dama.Dama(Color)
22	47	54	0	org.pmoo.ajedrez.Dama.puedeMover(Coordenada, Coordenada, Casilla)
23	94	110	0	org.pmoo.ajedrez.Dama.puedeComer(Coordenada, Coordenada, Casilla)
24	3	1	0	org.pmoo.ajedrez.Jugador.Jugador(String, Color)
25	2	1	0	org.pmoo.ajedrez.Jugador.getNombre()
26	2	1	0	org.pmoo.ajedrez.Jugador.imprimirColor()
27	2	1	0	org.pmoo.ajedrez.Jugador.getColor()
28	2	1	0	org.pmoo.ajedrez.Main.main(String[])
29	2	1	0	org.pmoo.ajedrez.NoPieza.NoPieza(Color)
30	2	1	0	org.pmoo.ajedrez.NoPieza.NoPieza()
31	2	1	0	org.pmoo.ajedrez.NoPieza.visualizarPieza()

Se ha importado las métricas al excel como se indica en la práctica:

Autoguardado Métricas-Ajedrez.xlsx

Archivo Inicio Insertar Disposición de página Fórmulas Datos Revisar Vista Ayuda

Calibri 11 A* Fuente Alineación Número Estilos

General Formato condicional Dar formato como tabla Estilos de celda Insertar Eliminar Formato Ordenar y filtrar Buscar y seleccionar Analizar datos Comentarios Compartir

B13

Nr.	Classes	Functions	NCSS	Javadocs	Package
1	16	66	956	0	org.pmoo.ajedrez
16	66	956	0	Total	
Packages Classes Functions NCSS Javadocs per					
1.00	16.00	66.00	956.00	0.00	Project
16.00	66.00	956.00	0.00		Package
4.13	59.75	0.00			Class
14.48	0.00				Function

javancss.packages javancss.classes javancss.methods

Accesibilidad: todo correcto Configuración de visualización 100%

Ejecución del programa CKJM:

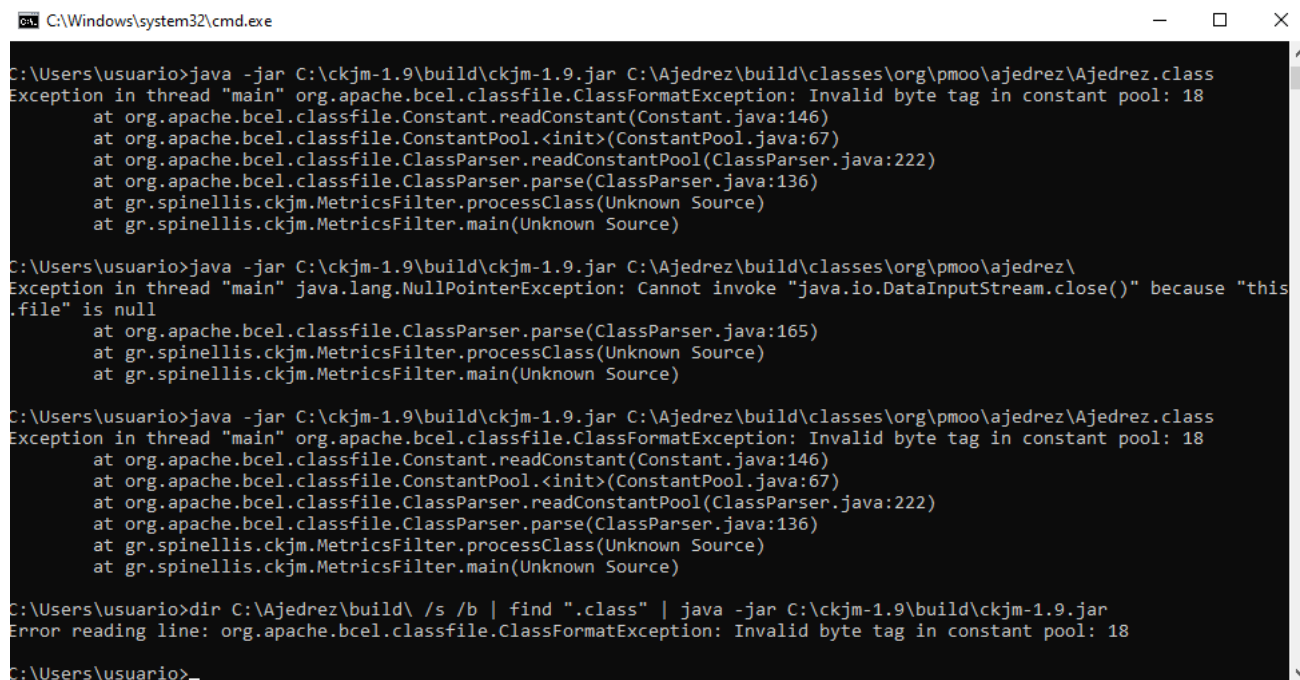
Comandos ejecutados:

```
java -jar C:\ckjm-1.9\build\ckjm-1.9.jar C:\Ajedrez\build\classes\org\pmoo\ajedrez\*.class
```

```
dir C:\Ajedrez\build\ /s /b | find ".class" | java -jar C:\ckjm-1.9\build\ckjm-1.9.jar
```

Al intentar ejecutar los comandos del programa, da el siguiente error:

Por lo que no se ha podido sacar las métricas del programa con CKJM.



```
C:\Windows\system32\cmd.exe

C:\Users\usuario>java -jar C:\ckjm-1.9\build\ckjm-1.9.jar C:\Ajedrez\build\classes\org\pmoo\ajedrez\Ajedrez.class
Exception in thread "main" org.apache.bcel.classfile.ClassFormatException: Invalid byte tag in constant pool: 18
    at org.apache.bcel.classfile.Constant.readConstant(Constant.java:146)
    at org.apache.bcel.classfile.ConstantPool.<init>(ConstantPool.java:67)
    at org.apache.bcel.classfile.ClassParser.readConstantPool(ClassParser.java:222)
    at org.apache.bcel.classfile.ClassParser.parse(ClassParser.java:136)
    at gr.spinellis.ckjm.MetricsFilter.processClass(Unknown Source)
    at gr.spinellis.ckjm.MetricsFilter.main(Unknown Source)

C:\Users\usuario>java -jar C:\ckjm-1.9\build\ckjm-1.9.jar C:\Ajedrez\build\classes\org\pmoo\ajedrez\
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "java.io.DataInputStream.close()" because "this
.file" is null
    at org.apache.bcel.classfile.ClassParser.parse(ClassParser.java:165)
    at gr.spinellis.ckjm.MetricsFilter.processClass(Unknown Source)
    at gr.spinellis.ckjm.MetricsFilter.main(Unknown Source)

C:\Users\usuario>java -jar C:\ckjm-1.9\build\ckjm-1.9.jar C:\Ajedrez\build\classes\org\pmoo\ajedrez\Ajedrez.class
Exception in thread "main" org.apache.bcel.classfile.ClassFormatException: Invalid byte tag in constant pool: 18
    at org.apache.bcel.classfile.Constant.readConstant(Constant.java:146)
    at org.apache.bcel.classfile.ConstantPool.<init>(ConstantPool.java:67)
    at org.apache.bcel.classfile.ClassParser.readConstantPool(ClassParser.java:222)
    at org.apache.bcel.classfile.ClassParser.parse(ClassParser.java:136)
    at gr.spinellis.ckjm.MetricsFilter.processClass(Unknown Source)
    at gr.spinellis.ckjm.MetricsFilter.main(Unknown Source)

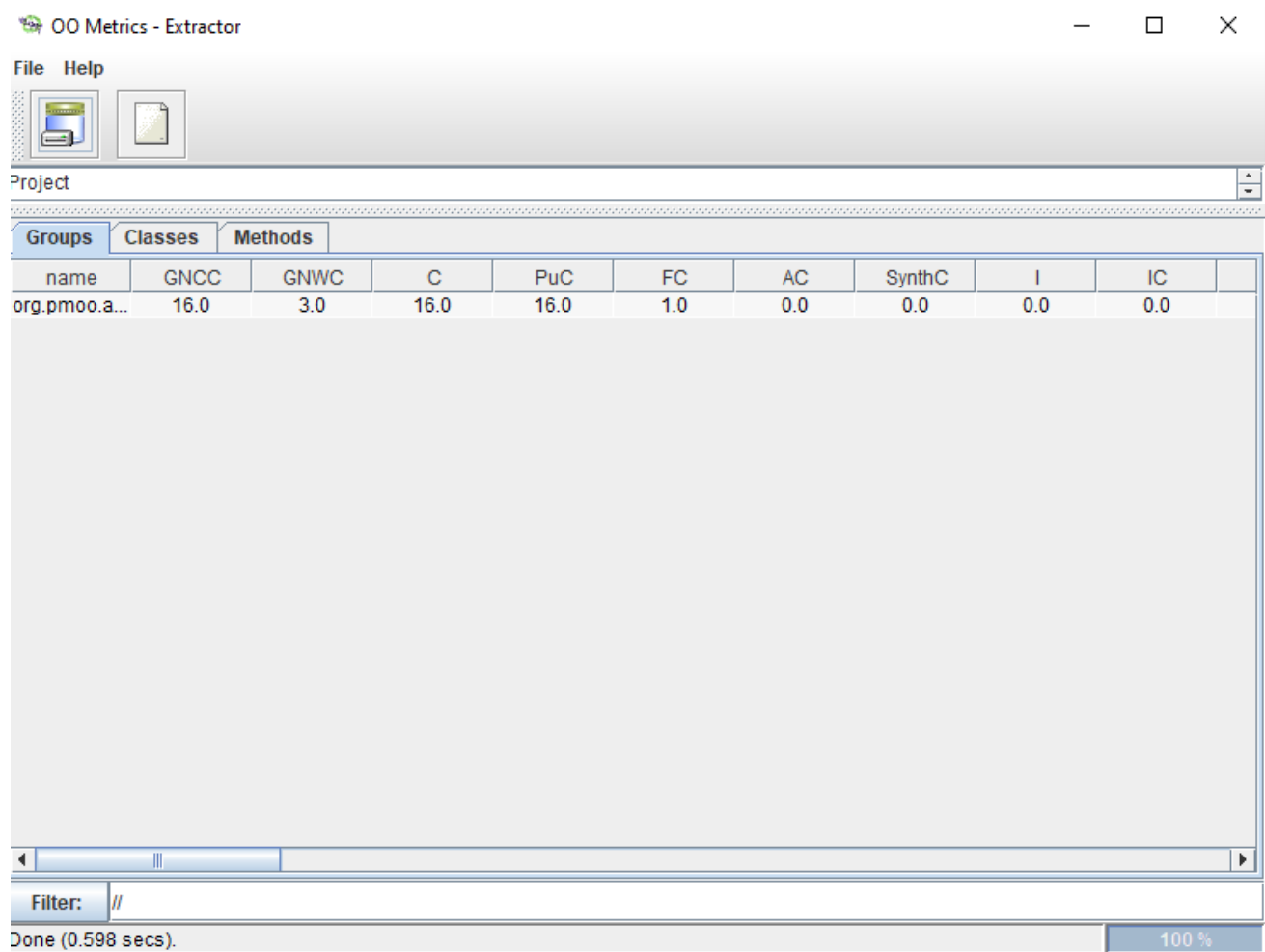
C:\Users\usuario>dir C:\Ajedrez\build\ /s /b | find ".class" | java -jar C:\ckjm-1.9\build\ckjm-1.9.jar
Error reading line: org.apache.bcel.classfile.ClassFormatException: Invalid byte tag in constant pool: 18

C:\Users\usuario>
```


Ejecución del programa Dependency Finder:

Comando:

OOMetricsGUI.bat

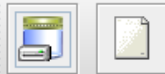


Groups										
Classes										
Methods										
name	GNCC	GNWC	C	PuC	FC	AC	SynthC	I	IC	
org.pmoo.a...	16.0	3.0	16.0	16.0	1.0	0.0	0.0	0.0	0.0	

Filter: //

Done (0.598 secs). 100 %

File Help



Project

Groups	Classes	Methods								
name	CNCC	CNWC	SLOC	M	PuM	ProM	PriM	PaM	FM	
org.pmoo.a...	7.0	1.0	51.0	5.0	3.0	0.0	2.0	0.0	0.0	
org.pmoo.a...	5.0	1.0	100.0	5.0	3.0	0.0	0.0	0.0	0.0	
org.pmoo.a...	7.0	1.0	64.0	5.0	3.0	0.0	0.0	0.0	0.0	
org.pmoo.a...	7.0	1.0	10.0	4.0	4.0	0.0	0.0	0.0	0.0	
org.pmoo.a...	5.0	1.0	12.0	7.0	3.0	0.0	2.0	1.0	0.0	
org.pmoo.a...	10.0	1.0	9.0	3.0	3.0	0.0	0.0	0.0	0.0	
org.pmoo.a...	4.0	1.0	172.0	5.0	3.0	0.0	0.0	0.0	0.0	
org.pmoo.a...	7.0	1.0	10.0	4.0	4.0	0.0	0.0	0.0	0.0	
org.pmoo.a...	4.0	1.0	4.0	2.0	2.0	0.0	0.0	0.0	0.0	
org.pmoo.a...	7.0	2.0	8.0	3.0	3.0	0.0	0.0	0.0	0.0	
org.pmoo.a...	4.0	1.0	38.0	5.0	3.0	0.0	0.0	0.0	0.0	
org.pmoo.a...	5.0	1.0	23.0	11.0	11.0	0.0	0.0	0.0	0.0	
org.pmoo.a...	3.0	1.0	64.0	5.0	3.0	0.0	0.0	0.0	0.0	
org.pmoo.a...	7.0	1.0	382.0	12.0	9.0	0.0	3.0	0.0	0.0	
org.pmoo.a...	7.0	1.0	11.0	3.0	2.0	0.0	1.0	0.0	0.0	
org.pmoo.a...	5.0	1.0	88.0	5.0	3.0	0.0	0.0	0.0	0.0	

Filter: //

Done (0.598 secs).

100 %

OO Metrics - Extractor

File Help

Project

Groups	Classes	Methods							
name	MNCC	MNWC	SLOC	PARAM	LVAR	IICM	IIPM	IEPM	OICF
org.pmoo.a...	7.0	1.0	4.0	0.0	1.0	1	0	0	0
org.pmoo.a...	14.0	2.0	14.0	0.0	5.0	1	0	0	0
org.pmoo.a...	12.0	3.0	3.0	0.0	0.0	0	1	0	1
org.pmoo.a...	12.0	2.0	14.0	0.0	1.0	0	1	0	2
org.pmoo.a...	5.0	1.0	10.0	0.0	4.0	1	0	0	0
org.pmoo.a...	5.0	1.0	7.0	1.0	2.0	0	1	0	1
org.pmoo.a...	10.0	2.0	60.0	3.0	12.0	0	0	0	1
org.pmoo.a...	10.0	2.0	30.0	3.0	8.0	0	0	0	0
org.pmoo.a...	7.0	1.0	7.0	1.0	2.0	0	1	0	1
org.pmoo.a...	10.0	2.0	36.0	3.0	4.0	0	0	0	1
org.pmoo.a...	10.0	2.0	18.0	3.0	4.0	0	0	0	0
org.pmoo.a...	7.0	1.0	3.0	0.0	1.0	0	1	0	0
org.pmoo.a...	8.0	2.0	1.0	0.0	1.0	0	21	0	0
org.pmoo.a...	13.0	2.0	2.0	0.0	1.0	0	2	0	0
org.pmoo.a...	8.0	2.0	2.0	1.0	2.0	0	4	0	0
org.pmoo.a...	7.0	1.0	0.0	0.0	0.0	1	0	0	0
org.pmoo.a...	5.0	1.0	3.0	3.0	2.0	1	0	0	0
org.pmoo.a...	8.0	2.0	1.0	0.0	1.0	0	1	0	0
org.pmoo.a...	6.0	1.0	2.0	0.0	0.0	0	0	0	2
org.pmoo.a...	7.0	2.0	1.0	1.0	1.0	0	0	0	0
org.pmoo.a...	6.0	1.0	1.0	0.0	0.0	0	0	0	0
org.pmoo.a...	10.0	1.0	4.0	2.0	2.0	0	11	0	0

Filter: //

Done (0.598 secs). 100 %

Para obtener las métricas se ha utilizado el comando:

```
OOMetrics.bat -csv -out informe C:\Ajedrez\build
```

```
C:\Users\usuario>00Metrics.bat -csv -out informe C:\Ajedrez\build
C:\Users\usuario>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 9CDB-FF76

Directorio de C:\Users\usuario

10/05/2022  22:15          10.943 informe_classes.csv
10/05/2022  22:15           2.334 informe_groups.csv
10/05/2022  22:15           8.609 informe_methods.csv
10/05/2022  22:15           3.655 informe_project.csv
```

Se han insertado los datos en Métricas-Ajedrez.xlsx adjuntado a la práctica.

2. **Razonar las decisiones** basándose en el tipo de métricas utilizadas y su significado. Citar referencias de posibles autores que proponen rangos de valores recomendados para cada métrica. Se recomienda consultar el documento "[Medición en la Orientación a Objetos](#)" como punto de partida.

Nota: No se puede usar un proyecto ya usado por otros compañeros ni el utilizado en la práctica de la asignatura (JChecs). Para ello, se debe anotar el nombre del proyecto en el Foro "PECL2 – Lista de proyectos de código abierto", creado en Blackboard, siguiendo las instrucciones que allí se recogen.

Forma de entrega:

El trabajo debe enviarse a través del Campus virtual, desde la sección Trabajos > PECL2.

Debe enviarse un único archivo comprimido (zip o rar) con el siguiente contenido:

- *PECL2(ApellidoNombre del Alumno).docx*. Con el informe del trabajo, incluyendo copias de pantalla, tablas, fragmentos de código y tantos elementos explicativos como considere necesario. Debe tener índice y tres secciones:
 - o Ejercicio 1: Con los 6 apartados indicados en el enunciado del ejercicio. Es necesario razonar los casos de prueba diseñados, e incluir las evidencias necesarias (capturas de pantallas del código Java de las pruebas y de su ejecución mostrando que se han superado todas), para que el profesor pueda comprobar que es correcto sin tener que abrir los proyectos en Netbeans.
 - o Ejercicio 2: Con los 4 apartados indicados en el enunciado del ejercicio. Incluyendo las clases, métodos y casos de prueba implementados. Deben razonarse los casos de prueba diseñados así como las clases de equivalencia utilizadas. Deben incluirse en el documento todas las evidencias necesarias (capturas de pantallas del código Java de las pruebas y de su ejecución), para que el profesor pueda comprobar que es correcto sin tener que abrir los proyectos en Netbeans.
 - o Ejercicio 3: Con los 2 apartados indicados en el enunciado del ejercicio. Indicando la URL desde la que se ha descargado el programa analizado. Incluyendo capturas de pantalla de las tres herramientas utilizadas, y razonando las conclusiones más relevantes sobre los valores obtenidos y su repercusión en el mantenimiento de la aplicación. Hay que indicar, al menos, qué clase(s) tiene(n) el valor más alto en métricas relevantes, y aquellas que están fuera de rangos recomendados por expertos, y citar la fuente de los expertos.
- *Proyectos de NetBeans*. Tres archivos comprimidos (zip o rar), uno con cada proyecto Java realizado en NetBeans: cadenaADN.zip, laboratorio.zip, cajero.zip.
- *Archivo Excel con métricas*. Que incluya los valores de las métricas del ejercicio 3, con un formato similar al ejemplo de la práctica 7.
- *Archivo comprimido con el código fuente del programa analizado en el ejercicio 3*.

Criterios de valoración

Se valorará la legibilidad, claridad, concisión y pertinencia respecto a lo solicitado así como la precisión de los documentos entregados y de las explicaciones correspondientes. El peso de cada ejercicio en la calificación es el siguiente: Ejercicio 1: 40%, Ejercicio 2: 30%, Ejercicio 3:

30%.

Anexo. Código fuente para el Ejercicio 1

```
package secuenciaADN;

/**
 * Representa una secuencia de ADN o secuencia de nucleótidos,
 * que consiste en una sucesión de letras A, C, G y T, que simbolizan las cuatro
 * subunidades de nucleótidos: Adenina, Citosina, Guanina y Timina.
 * Las secuencias se presentan pegadas unas a las otras, sin espacios.
 * Por ejemplo, la secuencia AAAGTCTGAC.
 */

public class SecuenciaADN {

    private String secuencia;

    /**
     * Crea la secuencia convirtiendo a mayúsculas todas las letras de la secuencia
     * que se recibe como parámetro, que no debe ser nulo.
     */
    public SecuenciaADN(String sec) {
        assert (sec != null);
        secuencia = sec.toUpperCase();
    }

    /**
     * Comprueba si una secuencia es válida, es decir, si sólo contiene
     * los caracteres A, C, G y T.
     */
    public boolean esValida() {
        if (secuencia.length() == 0) {
            return false;
        }
        else {
            for (int i = 0; i < secuencia.length(); i++) {
                if ("ACGT".indexOf(secuencia.charAt(i)) == -1) {
                    return false;
                }
            }
            return true;
        }
    }

    /**
     * Cuenta las veces que aparece un nucleótido válido en la secuencia.
     * Los nucleótidos válidos son: A, C, G y T.
     */
    public int contar(char nucleotido) {
        int aux = 0;
        for (int i = 0; i < secuencia.length(); i++) {
            if (secuencia.charAt(i) == nucleotido) {
                aux++;
            }
        }
        return aux;
    }
}
```

```
package laboratorio;

import secuenciaADN.SecuenciaADN;

public class Laboratorio {

    public boolean analizarADN(String adn) {

        SecuenciaADN sec = new SecuenciaADN(adn);

        if (sec.esValida()) {
```



```
        return true;
    } else {
        return false;
    }
}
```