MANTENIMIENTO Y PRUEBAS DEL SOFTWARE
(Prueba realizada el 5 de Septiembre de 2016)

1. El código de la derecha muestra parte de la implementación de la clase CircularBuffer. En un búfer circular los mensajes se van grabando secuencialmente hasta alcanzar el límite del búfer (SIZE), momento a partir del cual un nuevo mensaje se escribe sobre el mensaje más antiguo. El búfer puede mantener un número fijo de mensajes, los más recientes. El método displayLastMsg(n) muestra los n mensajes más recientes, en orden inverso (del más reciente al más antiguo). Si el búfer está vacío no se muestra ningún mensaje, y si n es mayor que el número de mensajes disponibles se muestran todos los mensajes. El método supone que la clase mantiene en todo momento el índice del último mensaje introducido en el búfer (lastMsg).

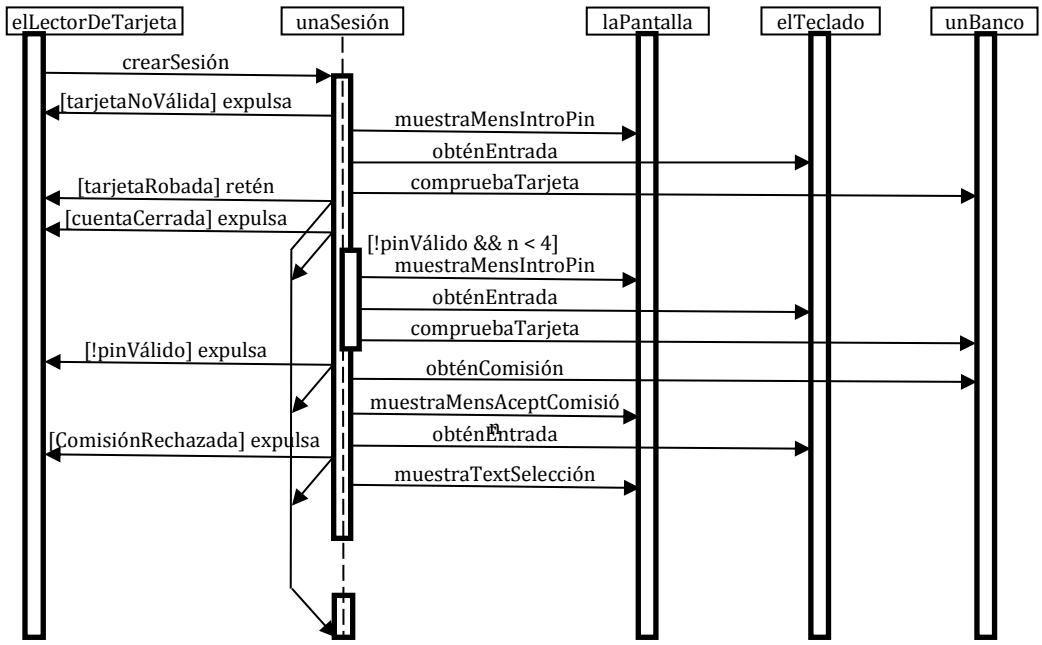
```
public class CircularBuffer {
   int    lastMsg,           // índice del último mensaje
          msgCounter;        // número de mensajes registrados
   final int SIZE = 1000;    // tamaño del búfer de mensajes
   String[] messageBuffer = new String[SIZE];

   // ... otros métodos

   public int displayLastMsg(int nToPrint) {
      int np = 0;
      if ((msgCounter > 0) && (nToPrint > 0)) {
         if (nToPrint > msgCounter)
            nToPrint = msgCounter;
         for (int j = lastMsg; ((j >= 0) && (np < nToPrint)); --j) {
            System.out.println(messageBuffer[j]);
            ++np;
         }
         if (np < nToPrint) {
            for (int j = SIZE - 1; ((j >= 0) && (np < nToPrint); --j) {
               System.out.println(messageBuffer[j]);
               ++np;
            }
         }
      }
      return np;
   }
}
```

3p a) Proporciona 2 casos de prueba a partir de un análisis combinatorio.

1,5p b) Proporciona el menor número posible de casos de prueba para obtener 100% de cobertura de condiciones compuestas.

2,5p c) Proporciona el menor número posible de casos de prueba para obtener 100% de cobertura de caminos DU a partir de un análisis del flujo de datos.

2. El diagrama de secuencia de la derecha muestra la interacción entre los componentes encargados de la gestión de un cajero automático. En ella podemos ver un objeto elLectorDeTarjeta, responsable de la interacción con el usuario, un objeto unaSesión, instancia de una clase Sesión que representa la sesión abierta por el usuario, y objetos laPantalla, elTeclado y unBanco que modelan, respectivamente, la pantalla del cajero, el teclado y el banco. Supongamos que disponemos de todos los componentes menos el del banco, el cual es proporcionado por el propio banco.



2p a) Diseña los casos de prueba necesarios para comprobar la correcta integración de las clases LectorDeTarjeta, Sesión, Pantalla y Teclado con la clase Banco.

2p b) Implementa uno de estos casos de prueba utilizando JUnit+Mockito.

1p 3. Similitudes y diferencias entre pruebas basadas en cobertura de código y pruebas basadas en mutantes.

El apartado 2.b) solo tienen que hacerlo aquellos alumnos que no han seguido la parte práctica de la asignatura