COMPRUEBA TU APRENDIZAJE

1. ¿Qué estrategias se siguen para probar el software? Si las pruebas de unidad funcionan, ¿es necesario hacer la prueba de integración?

2. Considérese una aplicación bancaria, donde el usuario puede conectarse al banco a través de Internet y realizar una serie de operaciones bancarias. Una vez que ha accedido al banco con las consiguientes medidas de seguridad puede realizar una serie de operaciones. La operación que se va a gestionar requiere la siguiente entrada:

   - *Código del banco:* puede estar en blanco o puede ser un número de 3 dígitos. En este último caso, el primero de ellos tiene que ser mayor que 1.
   - *Código de sucursal*: número de cuatro dígitos. El primero de ellos mayor de 0.
   - *Número de cuenta*: número de cinco dígitos.
   - *Clave personal*: valor alfanumérico de cinco posiciones.
   - *Orden*: puede estar en blanco o ser uno de los valores siguientes: "Talonario" o "Movimientos".

   El programa responde de la siguiente manera:

   - Si *Orden* tiene el valor "Talonario", el usuario recibe un talonario de cheques.
   - Si *Orden* tiene el valor "Movimientos", el usuario recibe los movimientos del mes en curso.
   - Si *Orden* está en blanco, el usuario recibe los dos documentos.
   - Si ocurre algún error en la entrada de datos, el programa muestra un mensaje de error sobre el dato implicado.

   Se pide definir las clases de equivalencia, casos de prueba válidos y casos de prueba no válidos que cubran una sola clase no válida.

3. Rellena en la siguiente tabla los casos de prueba tomando como referencia las reglas del análisis de valores límite:

Condiciones de entrada y de salida	Casos de prueba
Una variable toma valores comprendidos entre -4 y 4 (enteros)	
El programa lee un fichero que contiene de 1 a 100 registros	
El programa deberá generar de 1 a 5 listados	
El número de alumnos para calcular la nota media es 35	
La función deberá devolver un array de enteros, de 1 a 10 valores	

   Realiza después un programa Java para probar la función que devuelve el array de enteros. Utiliza los casos de prueba que hayas definido.

4. A partir del diagrama de flujo mostrado en la Figura 3.29 construye el grafo de flujo. Indica el número de nodos, aristas, regiones, nodos predicado, la complejidad ciclomática y el conjunto de caminos independientes.

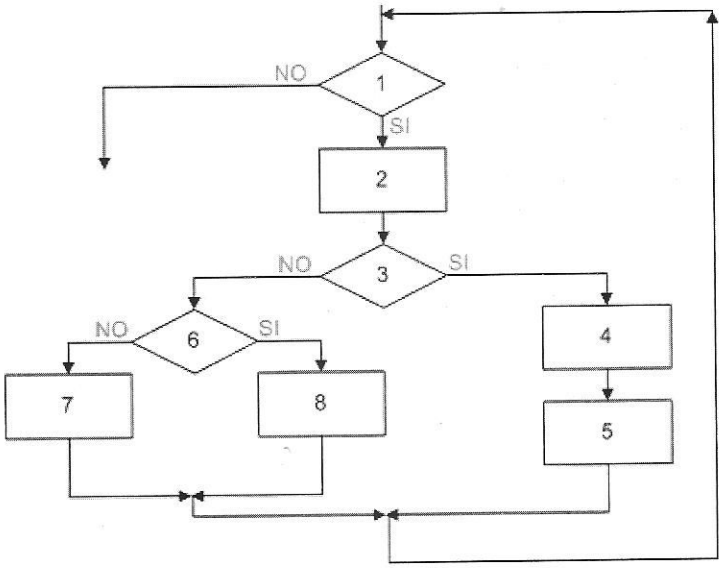

Figura 3.29. Ejercicio 4.

5. Realiza el grafo de flujo, calcula la complejidad ciclomática, define el conjunto básico de caminos, elabora los casos de prueba para cada camino y evalúa el riesgo para la siguiente función Java:

```
static int Contador1(int x, int y) {
  Scanner entrada = new Scanner(System.in);
  int num, c = 0;
  if (x > 0 && y > 0) {
      System.out.println("Escribe un número");
      num = entrada.nextInt();
      if (num >= x && num <= y){
           System.out.println("\tNúmero en el rango");
           c++;
      }
      else
           System.out.println("\tNúmero fuera de rango");

      }
  else
      c = -1;
  entrada.close();
  return c;
}//
```

6. Desarrolla una batería de pruebas para probar el método *DevuelveFecha()* de la clase *Fecha* que se expone a continuación. El método recibe un número entero y devuelve un String con un formato de fecha que dependerá del valor de dicho número. Si el número recibido es distinto de 1, 2 y 3 el método devuelve ERROR. La clase es la siguiente:

```
import java.text.SimpleDateFormat;
```

```
import java.util.Date;

public class Fecha {
     SimpleDateFormat formato;
     Date hoy;

     public Fecha() {
          hoy = new Date();
     }

     public String DevuelveFecha(int tipo) {
          String cad = "";
          switch (tipo) {
          case 1: {
               formato = new SimpleDateFormat("yyyy/MM");
               cad = formato.format(hoy);
               break;
          }
          case 2: {
               formato = new SimpleDateFormat("MM/yyyy");
               cad = formato.format(hoy);
               break;
          }
          case 3: {
               formato = new SimpleDateFormat("MM/yy");
               cad = formato.format(hoy);
               break;
          }
          default: {
               cad = "ERROR";
          }
          }
          return cad;
     }
}//
```

7. Escribe una clase de pruebas para probar el método *calculo()* de la clase *Factorial*. En el método se comprueba si el número es menor que 0, en este caso se lanza la excepción *IllegalArgumentException* con el mensaje *Número n no puede ser < 0*. Si el valor del factorial calculado es menor que 0 es que ha ocurrido un error de desbordamiento, en este caso se lanza la excepción *ArithmeticException* y se lanza el mensaje *Overflow, número n demasiado grande*. La clase a probar es la siguiente:

```
public class Factorial {
    public static int calculo(int n) {
       if (n < 0 ) {
            throw new IllegalArgumentException("Número "+ n +
                                  " no puede ser < 0");
        }
        int fact = 1;
        for (int i = 2; i <= n; i++)
            fact *= i;
```

```
            if (fact < 0 ) {
                throw new ArithmeticException("Overflow, número "+
                                n +" demasiado grande");
            }
            return fact;
        }
    }//
```

8. Se trata de implementar la entrada de datos desde una web para la petición de un CD de música. El cliente introduce el código del CD y una cantidad. El código deberá comenzar por un dígito, seguido de un guión y 7 dígitos más. Solo se podrá pedir un máximo de 3 CDs. Si el código del CD no está en el catálogo se produce un error (ER1) y si la cantidad solicitada supera el stock de dicho CD se produce otro error (ER2). En el caso de que todo vaya bien se enviará el CD al solicitante con el precio mostrado en la web, salvo que tenga oferta; en este caso se le aplicará al precio la oferta.

   Realiza la tabla con las condiciónes de entrada y las clases de equivalencia válidas y no válidas. Muestra varios casos de prueba, en los que se valide todo correctamente y que den error y no se lleve a cabo el pedido, en este caso indica por qué no se lleva a cabo el pedido.

ACTIVIDADES DE AMPLIACIÓN

1. Realiza el grafo de flujo, calcula la complejidad ciclomática, define el conjunto básico de caminos, elabora los casos de prueba para cada camino y evalúa el riesgo para la siguiente función Java:

```
static int Contador2(int x, int y) {
  Scanner entrada = new Scanner(System.in);
  int num, c = 0;
  if ( x > 0  &&  y > 0) {
     System.out.println("Escribe un número");
     num = entrada.nextInt();
     while (num != 0) {
           if ( num >= x  &&  num <= y ) {
                 System.out.println("\tNúmero en el rango");
                 c++;
           } else
                 System.out.println("\tNúmero fuera de rango");
           System.out.println("Escribe un número");
           num = entrada.nextInt();
     }//fin while
  }
  else
     c = -1;
  entrada.close();
  return c;
}//
```

2. Escribe una clase de pruebas para probar los métodos de la clase *TablaEnteros*. En esta clase de prueba crea un método con la anotación **@BeforeClass** en el que inicialices un array de enteros para usarlo en las pruebas de los métodos. El método *sumaTabla()* suma los elementos del array y devuelve la suma. El método *mayorTabla()* devuelve el elemento mayor de la tabla. Y el método *posicionTabla()* devuelve la posición ocupada por el elemento cuyo valor se envía.

   En el constructor se comprueba si el número de elementos de la tabla es nulo o 0, en este caso se lanza la excepción *IllegalArgumentException* con el mensaje *No hay elementos*. El método *posiciónTabla* también lanza la excepción, *java.util. NoSuchElementException*, en el caso de que no se encuentre el elemento en la tabla. Hay que añadir otros dos métodos de prueba para probar estas excepciones. La clase a probar es la siguiente:

```
public class TablaEnteros {
  private Integer[] tabla;

  TablaEnteros(Integer[] tabla) {
    if (tabla == null || tabla.length == 0)
      throw new IllegalArgumentException("No hay elementos");
    this.tabla = tabla;
  }

  //devuelve la suma de los elementos de la tabla
  public int sumaTabla() {
    int suma = 0;
    for (int i = 0; i < tabla.length; i++)
      suma += tabla[i];
    return suma;
  }

  //devuelve el mayor elemento de la tabla
  public int mayorTabla() {
    int max = -999;
    for (int i = 0; i < tabla.length; i++)
      if (tabla[i] > max)
          max = tabla[i];
    return max;
  }

  //devuelve la posición de un elemento cuyo valor se pasa
  public int posicionTabla(int n) {
    for (int i = 0; i < tabla.length; i++)
      if (tabla[i] == n)
          return i;
    throw new java.util.NoSuchElementException("No existe:" + n);
  }
}//
```