Laura Carratalá Castillo

Pruebas de caja negra y caja blanca

A continuación, se reflejarán de forma clara y concisa los pasos para la elaboración de las pruebas de caja negra y caja blanca de dos métodos pertenecientes a dos de nuestras aplicaciones que componen la Navaja Suiza. En nuestro caso, hemos elegido la aplicación 1, que consiste en la determinación de si un número positivo introducido es primo o no, y la aplicación 2, que consiste en mostrar los múltiplos de 3 y 5 encontrados hasta un número introducido, estando este comprendido entre el 1 y el 100.

Pruebas para la aplicación 1:

Como hemos indicado anteriormente, la aplicación 1 consiste en determinar si un número positivo introducido es primo o no. El método con el cual haremos las pruebas se llama **EsPrimo** y su código es el siguiente:

```
/// <summary>
/// Función que verifica si un número positivo introducido es o no es primo.
/// </summary>
/// <remarks>----</remarks>
/// <param name="numeroIntroducido">Número que introduce el usuario.</param>
/// <returns>Si es primo o no.</returns>
public static bool EsPrimo(int numeroIntroducido)
{
    bool esPrimo = true;

    if (numeroIntroducido > 0)
    {
        for (int i = 2; i < numeroIntroducido && esPrimo; i++)
        {
            if (numeroIntroducido % i == 0)
            {
                esPrimo = false;
            }
        }
    }
    return esPrimo;
}
```

En primer lugar, realizaremos las pruebas de caja negra. Para ello, definiremos las clases de equivalencia elegidas para cubrir el mayor número de pruebas posible. Cada prueba tendrá un identificador, en nuestro caso consistirá en la abreviatura “App”, más el número de la aplicación con la que estemos haciendo las pruebas (1 o 2), más un punto, más el número de la prueba, más la letra “N” o “B” en función de si estamos haciendo las pruebas de caja negra (N) o de caja blanca (B). P.ej: App1.1N, App1.1B.

Clases de equivalencia:

- **numeroIntroducido:** Es un int, deberá ser mayor que 0.
  App1.1N) numeroIntroducido >= 3 (que sea primo): Introducción de cualquier número entero positivo mayor o igual que 3 que sea primo. Valor límite: 3.
  App1.2N) numeroIntroducido >= 3 (que no sea primo): Introducción de cualquier número entero positivo mayor o igual que 3 que no sea primo. Valor límite: 4.
  App1.3N) 0 < numeroIntroducido < 3: Introducción de valores positivos menores que 3. Son valores especiales puesto que son susceptibles de dar algún error. Valores límite: 1, 2.

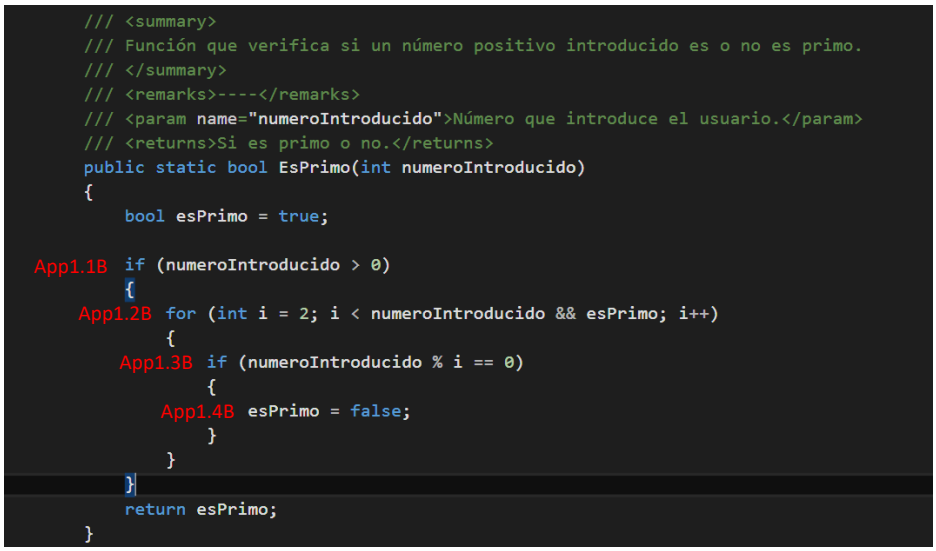
App1.4N) numeroIntroducido == 0: Es un valor especial. Ya no entra en el rango de valores permitidos.
App1.5N) numeroIntroducido < 0: Introducción de valores negativos. Valores límite: -1, - 2, -3, -4.
App1.6N) numeroIntroducido == decimal: Introducción de un número decimal, ya sea positivo o negativo.
App1.7N) numeroIntroducido == letra: Introducción de letras.
App1.8N) numeroIntroducido == símbolo: Introducción de caracteres no alfanuméricos.
App1.9N) NumeroIntroducido == "": No introducir ningún carácter.
App1.10N) NumeroIntroducido == MaxValue (2147483647): Introducción del máximo valor que puede tener un int, controlando también los límites. Valores límite: 2147483646, 2147483647, 2147483648.
App1.11N) NumeroIntroducido == MinValue (-2147483648): Introducción del mínimo valor que puede tener un int, controlando también los límites. Valores límite: -2147483647, -2147483648, -2147483649.

Prueba	Clase de equivalencia	Valores de entrada	Resultado	Comentarios
App1.1N	numeroIntroducido >= 3 (que sea primo)	3, 89	Válido	Los números son primos.
App1.2N	numeroIntroducido >= 3 (que no sea primo)	4, 100	Válido	Los números no son primos.
App1.3N	0 < numeroIntroducido < 3	1, 2	Para 2: Válido Para 1: No Válido	El 2 es primo, pero el 1 no se considera primo y el programa indica que si lo es
App1.4N	numeroIntroducido == 0	0	No válido	El 0 no es primo y el programa indica que si lo es. Además, no es un número permitido puesto que deben ser valores mayores a 0.
App1.5N	numeroIntroducido < 0	-1, -2, -3, -4, -50	No válido	El programa indica que son primos cuando no lo son.
App1.6N	numeroIntroducido == decimal	1.5, -1.5	Válido	El programa indica que se introduzca un elemento válido, puesto que solo acepta números enteros.
App1.7N	numeroIntroducido == letra	A, a, aa	Válido	El programa indica que se introduzca un elemento válido, puesto que solo acepta números enteros.
App1.8N	numeroIntroducido == simbolo	>, *	Válido	El programa indica que se introduzca un elemento válido, puesto

				que solo acepta números enteros.
App1.9N	numeroIntroducido == “”	(vacío)	Válido	El programa indica que se introduzca un elemento válido, puesto que solo acepta números enteros.
App1.10N	numeroIntroducido == MaxValue (2147483647)	2147483646, 2147483647, 2147483648	Válido	El programa indica que el 2147483646 no es primo, que el 2147483647 es primo y que el 2147483648 no es un elemento válido porque supera el máximo valor que puede tener un int.
App1.11N	numeroIntroducido == MinValue (-2147483647)	-2147483647, -2147483648, -2147483649	No válido	El programa indica que tanto el -2147483647, como el -2147483648 son primos, cuando al ser negativos no lo son. el -2147483649 no es un elemento válido porque supera el mínimo valor que puede tener un int.

Vemos que con todas estas pruebas hemos cubierto la totalidad del código de nuestro método, con lo que ya no sería necesario realizar más pruebas.

Sin embargo, realizaremos las pruebas de caja blanca, cubriendo las distintas partes de las que se compone nuestro método. En total hay 3 bucles que se enumeran en rojo de la forma que vemos en la siguiente imagen:



Para cubrir todos los bucles, será necesario realizar únicamente las siguientes pruebas:

Prueba	Valor de entrada	Resultado
App1.1B	0	No válido
App1.2B	1	No válido
App1.3B	3	Válido
App1.4B	4	Válido

A continuación, indicaremos las pruebas de caja negra que son cubiertas en cada prueba de caja blanca:

- App1.1B: Cubre las pruebas App1.4N, App1.5N y App1.11N.
- App1.2B: Cubre la prueba App1.3N.
- App1.3B: Cubre la prueba App1.1N.
- App1.4B: Cubre las pruebas App1.2N y App1.10N.

Pruebas para la aplicación 2:

Como hemos indicado anteriormente, la aplicación 2 consiste en mostrar los múltiplos de 3 y 5 encontrados hasta un número introducido, estando este comprendido entre el 1 y el 100. El método con el cual haremos las pruebas se llama **MostrarSerieMultiplos** y su código es el siguiente:

```
/// <summary>
/// Función que devuelve la cadena de números múltiplos de 3 y 5
/// comprendidos hasta el número que introduzca el usuario.
/// </summary>
/// <remarks>----</remarks>
/// <param name="numeroIntroducido">Número que introduce el usuario.</param>
/// <returns>Cadena con los múltiplos de 3 y 5.</returns>
public string MostrarSerieMultiplos(int numeroIntroducido)
{
    string cadenaTexto = "";

    for (int i = 1; i <= numeroIntroducido; i++)
    {
        if (i % 3 == 0 || i % 5 == 0)
        {
            cadenaTexto = cadenaTexto + i + ", ";
        }
    }
    return cadenaTexto;
}
```

En primer lugar, realizaremos las pruebas de caja negra. Para ello, definiremos las clases de equivalencia elegidas para cubrir el mayor número de pruebas posible. Al igual que en el caso anterior, usaremos la misma nomenclatura para dar un identificador a cada prueba.

Clases de equivalencia:

- **numeroIntroducido**: Es un int.

  App2.1N) 5 > numeroIntroducido >= 100: Introducción de cualquier número entero positivo mayor que 5 y menor o igual a 100. Valores límite: 6, 101.

  App2.2N) 3 <= numeroIntroducido <= 5: Introducción de valores enteros positivos entre el 3 y el 5 incluidos. Valores límite: 3, 4, 5.

  App2.3N) numeroIntroducido < 3: Introducción de valores positivos hasta el 2. Valores límite: 1, 2.

  App2.4N) numeroIntroducido == 0: Es un valor especial, ya no entra en el rango de valores permitidos.

  App2.5N) numeroIntroducido < 0: Introducción de valores negativos. Valores límite: -1, -3, -5.

  App2.6N) numeroIntroducido == decimal: Introducción de un número decimal, ya sea positivo o negativo.

  App2.7N) numeroIntroducido == letra: Introducción de letras.

  App2.8N) numeroIntroducido == símbolo: Introducción de caracteres no alfanuméricos.

  App2.9N) NumeroIntroducido == "": No introducir ningún carácter.

  App1.10N) NumeroIntroducido == MaxValue (2147483647): Introducción del máximo valor que puede tener un int, controlando también los límites. Valores límite: 2147483646, 2147483647, 2147483648.

  App1.11N) NumeroIntroducido == MinValue (-2147483648): Introducción del mínimo valor que puede tener un int, controlando también los límites. Valores límite: -2147483647, -2147483648, -2147483649.

Prueba	Clase de equivalencia	Valores de entrada	Resultado	Comentarios
App2.1N	5 > numeroIntroducido >= 100	6, 7, 100, 101	Válido	Como resultado obtenemos la cadena de múltiplos. Como el 7 no es múltiplo ni de 3 ni de 5, no se añade a la cadena. El resto de valores sí, excepto el 101, que se sale del rango de valores permitidos y el programa nos indica que introduzcamos un número entre el 1 y el 100.
App2.2N	3 <= numeroIntroducido <= 5	3, 4, 5	Válido	Como resultado obtenemos la cadena de múltiplos. Como el 4 no es múltiplo ni de 3 ni de 5, no se añade a la cadena.
App2.3N	numeroIntroducido < 3	1, 2	Válido	Como el 1 y el 2 no son múltiplos ni de 3 ni de 5, no se añaden a la cadena.
App2.4N	numeroIntroducido == 0	0	Válido	No entra en el rango de valores permitidos.

App2.5N	numeroIntroducido < 0	-1, -3, -5	Válido	No entran en el rango de valores permitidos.
App2.6N	numeroIntroducido == decimal	1.5, -1.5	Válido	El programa indica que se introduzca un elemento válido, puesto que solo acepta números enteros.
App2.7N	numeroIntroducido == letra	A, a, aa	Válido	El programa indica que se introduzca un elemento válido, puesto que solo acepta números enteros.
App2.8N	numeroIntroducido == símbolo	>, *	Válido	El programa indica que se introduzca un elemento válido, puesto que solo acepta números enteros.
App2.9N	numeroIntroducido == “”	(vacío)	Válido	El programa indica que se introduzca un elemento válido, puesto que solo acepta números enteros.
App2.10N	numeroIntroducido == MaxValue (2147483647)	2147483646, 2147483647, 2147483648	Válido	Como los valores 2147483646 y 2147483647 superan el rango de entre 1 y 100, el programa nos indica que introduzcamos un número que se encuentre en dicho rango. El 2147483648 supera el valor máximo que puede tener un int.
App2.11N	numeroIntroducido == MinValue (-2147483647)	-2147483647, -2147483648, -2147483649	Válido	Como los valores -2147483647 y -2147483648 no están en el rango de entre 1 y 100, el programa nos indica que introduzcamos un número que se encuentre en dicho rango. El -2147483649 es menor que el valor mínimo que puede tener un int.

Vemos que con todas estas pruebas hemos cubierto la totalidad del código de nuestro método, con lo que ya no sería necesario realizar más pruebas.

Sin embargo, realizaremos las pruebas de caja blanca, cubriendo las distintas partes de las que se compone nuestro método. En total hay 2 bucles que se enumeran en rojo de la forma que vemos en la siguiente imagen:

```
    /// <summary>
    /// Función que devuelve la cadena de números múltiplos de 3 y 5
    /// comprendidos hasta el número que introduzca el usuario.
    /// </summary>
    /// <remarks>----</remarks>
    /// <param name="numeroIntroducido">Número que introduce el usuario.</param>
    /// <returns>Cadena con los múltiplos de 3 y 5.</returns>
    public string MostrarSerieMultiplos(int numeroIntroducido)
    {
        string cadenaTexto = "";

App2.1B for (int i = 1; i <= numeroIntroducido; i++)
        {
 App2.2B if (i % 3 == 0 || i % 5 == 0)
            {
  App2.3B cadenaTexto = cadenaTexto + i + ", ";
            }
        }
        return cadenaTexto;
    }
```

Para cubrir todos los bucles, será necesario realizar únicamente las siguientes pruebas:

Prueba	Valor de entrada	Resultado
App2.1B	0	No válido, no permite entrar al bucle al no ser un valor permitido (no está entre 1 y 100)
App2.2B	1	Válido
App2.3B	3	Válido

A continuación, indicaremos las pruebas de caja negra que son cubiertas en cada prueba de caja blanca:

- App2.1B: Cubre las pruebas App2.4N, App2.5N, App2.11, y App2.10N.
- App2.2B: Cubre la prueba App2.3N.
- App2.3B: Cubre las pruebas App2.2N y App2.1N.