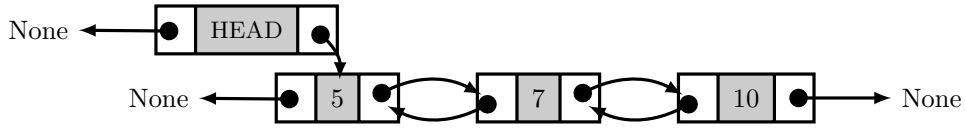
Tecnología de la Programación **16 de enero de 2023**
Duración: 3h:00m Evaluación Final.

Aquellos ejercicios, que, a pesar de estar resueltos correctamente, no estén razonados por el alumno, se considerarán no válidos. Comenta el código. Justifica, en su caso, el valor de retorno de los métodos. Se asume que todos los atributos privados tienen propiedades get/set (no los escribas). ENTREGA EL ENUNCIADO JUNTO CON TUS SOLUCIONES.

Apellidos y Nombre: .....

Ejercicio 1 (7 puntos) Se quiere implementar una **lista ordenada creciente** usando una estructura doblemente enlazada con cabecera.



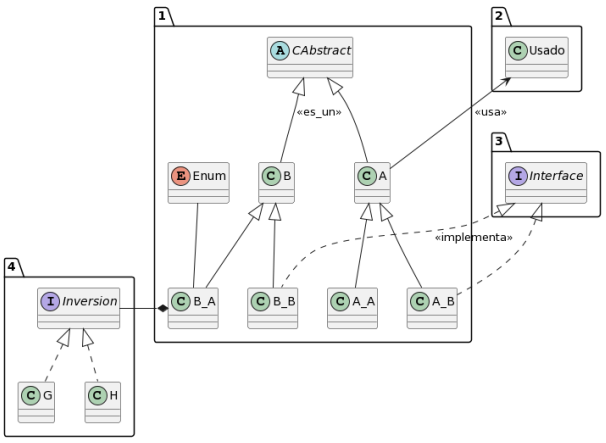
Cada nodo consta de 3 atributos: **valor**, representa a un elemento de la lista. P.e., 5, 7, 10, ...; **prev**, guarda la referencia al nodo anterior en la lista - representado por las flechas orientadas hacia la izquierda; **next**, guarda la referencia al nodo siguiente en la lista - representado por las flechas orientadas hacia la derecha.
Una lista se representa con los siguientes campos: **head**, guarda la referencia al nodo cabecera de la estructura; **last**, guarda la referencia al nodo de la estructura que tienen el máximo valor; **len**, almacena el número de elementos de la lista.
A. implementa los siguientes operadores del TDA Lista Ordenada Creciente, `SortedList`. Se indican con notación punto:

1. **(0.25 Pts.)** El `constructor` de listas. Inicialmente es una lista vacía. Es decir, solo constará del nodo cabecera.
2. **(2.5 Pts.)** `.append(a):None`. Dado un **valor** a, lo añade en una posición n tal que $a_{n-1} \leq a_n = a \leq a_{n+1}$, siendo a_k el valor de la posición k. Es decir, lo añade respetando el orden creciente.
   Para la implementación se invocará al operador privado `._get_node_prev(a):Node` que retorna, para un **valor** a, el nodo N que verifica: $N = arg_M \max\{valor(M) |\ valor(M) < a, M \in Nodos\}$. P.e. en la figura, `.get_node_prev(8)` y `.get_node_prev(10)` es el nodo que contiene al 7 y para `.get_node_prev(3)` o `.get_node_prev(5)` es el nodo cabecera.
3. **(0.75 Pt.)** `.search(a):bool`. Indica si existe el **valor** a en la lista. Para su implementación se usará una variante de la búsqueda binaria llamada **búsqueda por interpolación** donde se suponen que los valores están distribuidos uniformemente. En concreto, considera que la posición, p, donde se puede encontrar el valor a viene dado por $p = min + \left\lfloor \frac{(a - L[min])(max - min)}{L[max] - L[min]} \right\rfloor$; siendo $[min, max]$ las **posiciones** (índices enteros) de la lista L donde se está buscando al elemento a.
   Sobreescribe el método `__getitem(self,indice)__`.
4. **(0.5 Pt.)** `.save(file): None`. Guarda el contenido de la lista en un fichero de texto. Para la implementación haz control de excepciones **propio** y ten en cuenta que los valores de la lista pueden ser de cualquier tipo mientras estén ordenados.

C. (0.5 Pts.) ¿A qué TDA respondería esta representación si tuviese un único método de consulta/extracción de sus elementos y ello lo hiciese por solo uno de sus "extremos" sin acceso aleatorio? Para este caso ¿cómo sería `.search()` para buscar un elemento?
D. (2.5 Pts.) Haz un programa que contemple los siguientes módulos: 1 el TDA indicado con las clases `Node` y `SortedList` -es **A.**-, 2 la clase `Alumno` que se caracterizan por su nombre y calificación para ser usada con `SortedList`. 3 para la gestión de un menú simple de texto con las opciones: `0-Salir`, `1-Crear nueva lista`, `2-Añadir alumno`, `3-Mostrar alumnos` y todo lo que consideres necesario para el uso del menú -p.e. excepciones sobre opciones no existentes-, 4 el que relaciona los módulos anteriores - es un presentador o controlador y 5 el programa principal que se ejecuta sin fin hasta que se decida salir.
NO se necesita COPIAR lo que ya se haya escrito en apartados anteriores; pero si fuera necesario **añade en este apartado** los métodos que consideraras necesario. Controla todas las excepciones. Añade más módulos si fuera necesario.

Ejercicio 2 (3 puntos)

Invéntate un problema que responda al esquema general de la figura, y que no se corresponda con el proyecto práctico de programación.



1. **(0.5 Pt.)** Constará de una clase abstracta de las que derivan dos clases y de cada una de ellas otras dos. En alguna(s) clase(s) debe(rán) existir atributo(s) de instancia, atributo(s) de clase, método(s) de instancia, método(s) de clase y método(s) estático(s). Alguna clase deberá usar un enumerado (en la figura se usa en B_A)
2. **(0.5 Pts.)** Una de las clases usará otra clase no perteneciente a la jerarquía de clases. En la figura se ha optado por la clase A, pero puedes hacerlo con otra.
3. **(1 Pts.)** Implementa una interface en dos clases de la jerarquía que no compartan una clase padre común. En la figura se ha representado por las clases A_B y B_B.
4. **(1 Pt.)** Haz que una clase tenga un miembro delegado y que responda al principio SOLI**D** (inversión de dependencias). En la figura se aplica sobre la clase B_A.

Se valorará la originalidad, escritura legible y **la definición correcta del problema sin ambigüedades de ningún tipo** para que responda a dicho esquema general. Escribe el código solución a la definición de tu problema donde se vea claramente cómo se utiliza la clase de uso, la interface y la inversión/delegación.