Práctica Evaluable Temas 9 y 12.

Manejo de ficheros y bibliotecas de uso frecuente

Objetivos.

- Repasar los conceptos estudiados hasta ahora.

Consideraciones iniciales.

- Deberéis realizar un único proyecto en Visual Studio o similar, con el nombre que se indica en este enunciado.
- Cada clase que se indica debe crearse en su propio archivo fuente, con el mismo nombre de la clase y con extensión ".cs". Así, por ejemplo, la clase "Elemento" deberá crearse en un archivo llamado "Elemento.cs". El programa principal (la clase con el método "Main") podrá guardarse en el archivo que se crea por defecto, llamado "Program.cs".
- Cada clase debe tener un comentario inicial explicando su utilidad. Se deben aclarar con comentarios también los fragmentos de código que no quede muy claro qué hacen.
- Como en otras prácticas evaluables anteriores, el proyecto debe compilar, ya que de lo contrario se evaluará con un 0.

Entrega.

Se debe entregar un archivo ZIP con el proyecto completo comprimido.

Nombre del archivo: **Grupo_Apellidos_Nombre_PracT9T12.zip**

Por ejemplo, si te llamas Andrés Pérez Gómez y eres de 1º de DAM B, el archivo debe llamarse *B_Perez_Gomez_Andres_PracT9T12.zip*. Dentro, debe contener el proyecto completo.

Desarrollo.

Ejercicio "Tabla periódica de elementos químicos".

Nombre del proyecto: "TablaPeriodica"

Puntuación máxima: 10 puntos

La práctica de este tema, consistirá en realizar un programa para el seguimiento por parte de un estudiante de la ESO, del estudio que ha realizado de la tabla periódica de elementos químicos, según la clasificación de elementos propuesta por Antoine Lavoisier. El estudiante podrá guardar y cargar el conjunto de elementos químicos estudiados, y además visualizarlos según diferentes posibilidades.

1. Tipos de elementos (1,5 puntos)

(0,75 puntos) Clase abstracta **Elemento**

Partiremos de una clase base abstracta que almacenará los datos generales de cualquier elemento químico:

- Nombre.
- Símbolo químico.
- Número atómico.
- Año de descubrimiento: será de tipo DateTime para fechas D.C. Para elementos químicos descubiertos en la Prehistoria o en la Edad Antigua (antes de Cristo) este dato no será tenido en cuenta.
- Elemento antiguo: será cierto si la fecha de descubrimiento es A.C, falso en caso contrario.

Se deberán poder definir todos los atributos mediante el/los constructor/es de la clase. También se deberán definir las propiedades get/set públicas para acceder a los atributos (protegidos) de la clase.

(0,75 puntos) Subclases

Habrá tres subtipos (subclases de la anterior) de los cuales nos interesa conocer estas características específicas:

- Metal
  - es liquido?
  - color
  - es metal noble?
- Metaloide
  - es metal refractario?
  - es metal noble?
- No metal
  - es gas noble?

Se creará una clase para cada uno de estos subtipos, heredando de la clase padre, y definiendo los correspondientes constructores (basados en el padre), atributos privados y propiedades get/set públicas.

2. La carga y guardado de elementos (2,25 puntos)

Se debe crear una clase llamada ***GestorElementos***, que tendrá esta descripción:

- Una constante privada con el nombre del fichero donde se guardarán y cargarán los elementos químicos. El nombre de dicho fichero será “elementos.txt”, y se guardará en la misma carpeta que el ejecutable de nuestra aplicación.
- Un método estático llamado ***Cargar***, sin parámetros, que leerá el contenido del archivo “elementos.txt” (si existe), creará un elemento del tipo correspondiente con cada línea que lea del fichero, y lo almacenará en una colección de elementos del tipo adecuado.

  Para almacenar los elementos químicos internamente en la aplicación, el alumno deberá escoger el tipo de colección concreta que más se ajuste a la resolución del problema. No se podrá utilizar un array de objetos Elemento.

  Al final del proceso, devolverá (*return*) esa colección como resultado. Notar que, si el archivo no existe o está vacío, deberá devolver una colección vacía, sin mostrar ningún mensaje de error por pantalla ni producir ninguna excepción.
- Otro método estático llamado ***Guardar****,* que recibirá como parámetro una colección de elementos y, si no está vacía, los almacenará en el archivo anterior (“elementos.txt”), con el mismo formato con el que luego se cargarán.

Descripción archivo almacenamiento elementos

En el archivo “elementos.txt” se almacenarán los datos, con la siguiente estructura:

```
Tipo_de_Elemento;Nombre;Símbolo químico;Número atómico;Año de
descubrimiento;...
```

- Una línea por cada elemento químico.
- Los diferentes campos estarán separados por “punto y coma” (;).
- El formato con que se guarda la fecha deberá ser únicamente el año con 4 cifras numéricas (*aaaa*).
- Los puntos suspensivos (...) representan el resto de campos del elemento, que variarán dependiendo del tipo de elemento que sea.
- No es necesario que los elementos se guarden ordenados.

Aquí se muestra un ejemplo del posible contenido de ese archivo:

```
Metal;Mercurio;Hg;80;A.C.;liquido;rojo;no noble
Metal;Oro;Au;79;A.C.;no;amarillo;noble
Metaloide;Cobalto;Co;27;1730;no;no noble
Metaloide;Rutenio;Ru;44;1807;no;noble
NoMetal;Helio;He;2;1895;gas noble
NoMetal;Hidrógeno;H;1;1766;no noble
```

3. La tabla periódica (2,5 puntos)

Crearemos una clase **Tabla** para mostrar los elementos químicos agrupados por los siglos o los años en que fueron descubiertos.

- **(0,25 puntos)** Un método estático llamado ***MostrarSigloActual***, que recibirá como parámetro una colección de objetos *Elemento* (de cualquier tipo). Mostrará en pantalla, en la parte izquierda, los años del siglo 21, en bloques de 5 años por fila, quedando algo así

```
Siglo 21

2000 2001 2002 2003 2004
2005 2006 2007 2008 2009
2010 2011 2012 2013 2014
...
2090 2091 2092 2093 2094
2095 2096 2097 2098 2099
```

- **(1,25 puntos)** Un método estático llamado ***MostrarSiglo***, que recibirá como parámetro una colección de objetos *Elemento* (de cualquier tipo) y el siglo a visualizar. Mostrará en pantalla, en la parte izquierda, los años del siglo, con un aspecto similar al anterior. Por ejemplo:

```
Siglo 19

1800 1801 1802 1803 1804
1805 1806 1807 1808 1809
1810 1811 1812 1813 1814
...
1890 1891 1892 1893 1894
1895 1896 1897 1898 1899
```

- **(1 punto)** Un método estático llamado ***MostrarTodosSiglos***, que recibirá como parámetros una colección de elementos como el método anterior. Mostrará en pantalla los siglos y la cantidad de elementos descubiertos en ese siglo, también en la parte izquierda, con un formato similar a este:

```
Edad Antigua : 11 elementos
Siglo 1      :
Siglo 2      :
Siglo 3      :
...
Siglo 19     : 50 elementos
Siglo 20     : 30 elementos
Siglo 21     : 5 elementos
```

En todos estos métodos, se escribirán en diferentes colores los años o siglos en función del tipo de elemento químico descubierto en ese año o siglo:

- Rojo si se descubrieron solo metales.
- Azul si se descubrieron solo metaloides.
- Verde si se descubrieron solo gases.
- En color invertido (texto negro sobre fondo blanco) en el caso de que en un mismo año o siglo haya más de un tipo de elemento.
- En color estándar (texto blanco sobre fondo negro), los años o siglos donde no haya nada que mostrar.

4. El programa principal (3,75 puntos)

Desde el programa principal, inicialmente se cargará la colección de elementos químicos con el método *Cargar* de *GestorElementos*. Recuerda que esta colección puede estar vacía si el fichero inicialmente no existe o no tiene nada. A continuación, se mostrará por pantalla:

- En la parte superior izquierda el siglo actual con sus años y colores pertinentes.
- En la parte derecha un menú con las opciones:
  1. Ver todos los siglos.
  2. Ver siglo.
  3. Nuevo Elemento.
  4. Salir.

Por ejemplo, debe quedar algo así:

```
Siglo 21                        1. Ver todos los siglos
                                2. Ver siglo
2000 2001 2002 2003 2004        3. Nuevo Elemento
2005 2006 2007 2008 2009        4. Salir
2010 2011 2012 2013 2014
2015 2016 2017 2018 2019        Elige una opción: _
...
2090 2091 2092 2093 2094
2095 2096 2097 2098 2099
```

Las opciones del menú desarrollarán los siguientes aspectos:

1. **(0,5 puntos) Ver todos los siglos.** Mostrará todos los siglos y el número de elementos descubiertos agrupados por siglo en la mitad izquierda, y el menú de la parte derecha.

2. **(1,25 puntos) Ver siglo**. Pedirá al usuario que introduzca un siglo, del 0 al 21, (representando el 0 la “Prehistoria y La Edad Antigua”). Si los datos son válidos, limpiará la consola y mostrará:
   - En la mitad izquierda el siglo indicado, con sus años y colores pertinentes.
   - En la parte derecha:
     - En caso de no haber elementos descubiertos en ese siglo, se mostrará el mensaje “No hay elementos descubiertos en este siglo”
     - En otro caso, se pedirá al usuario que introduzca un año válido para mostrar la información completa de todos los elementos descubiertos en ese año.

   Ejemplo:

```
Siglo 21                        1. Ver todos los siglos
                                2. Ver siglo
2000 2001 2002 2003 2004        3. Nuevo Elemento
2005 2006 2007 2008 2009        4. Salir
2010 2011 2012 2013 2014
2015 2016 2017 2018 2019        Elige una opción: 2
2020 2021 2022 2023 2024
2025 2026 2027 2028 2029        Indica el siglo: 19
...
2090 2091 2092 2093 2094
2095 2096 2097 2098 2099
```

   Esta elección provocaría que se limpiase la pantalla y se mostrase esta otra:

```
Siglo 19                        Elige un año: 1807

1800 1801 1802 1803 1804        Descubierto Rutenio (Ru, 44)
1805 1806 1807 1808 1809        No refractario. Noble
1810 1811 1812 1813 1814
1815 1816 1817 1818 1819        Pulsa una tecla para continuar...
...
1890 1891 1892 1893 1894
1895 1896 1897 1898 1899
```

3. **(1,5 puntos) Nuevo elemento**. Limpiará la pantalla y pedirá al usuario los datos de un nuevo elemento, añadiéndolo a la colección. Se deberá verificar sobre la marcha que cada dato del elemento sea correcto (que los textos no estén vacíos, la fecha sea válida y los datos numéricos sean válidos), y en caso de no serlo, se le volverá a pedir hasta que lo sea.
4. **(0,5 punto) Salir**. Llamará al método *Guardar* de la clase *GestorElementos* para guardar los datos (la colección de elementos) y saldrá del programa.

Tras mostrar los mensajes de información correspondientes, se pedirá al usuario que pulse una tecla (Console.ReadKey) para continuar, volviendo a la pantalla inicial.

Observaciones generales de la práctica.

- Se deben definir las propiedades get/set públicas para acceder a los atributos (protegidos o privados) de todas las clases creadas.
- Además de los atributos indicados para cada clase, se pueden añadir otros atributos, constructores o métodos que se consideren oportunos. Posteriormente, se valorará si esos elementos añadidos son de utilidad o no.
- **Para almacenar los datos internamente en la aplicación, el alumno deberá escoger el tipo de colección concreta que más se ajuste a la resolución del problema.**
- Recordad, que se **valorarán negativamente algunas malas prácticas** de programación, como por ejemplo:
  - La *suciedad* de código, en lo referente a lo explicado en el Tema 9 del bloque 1 del módulo de Entornos de Desarrollo:
    - Nombres poco apropiados de variables, funciones o clases (ya se venía penalizando con anterioridad).
    - Espaciado y alineación vertical (separación entre funciones y entre bloques de código con propósito diferente).
    - Espaciado y alineación horizontal (incluyendo que las líneas de código no excedan del ancho recomendado).
  - Utilización incorrecta de las estructuras de control, estructuras repetitivas, las instrucciones,... Algunos casos típicos que conviene tener en cuenta:
    - Utilización incorrecta de la estructura de control "*switch-case*".
    - La instrucción "*goto*" no debe utilizarse salvo en "*switch-case*".
    - Los bucles "*for*" solo deberían utilizarse cuando se conoce el principio y fin del bucle, es decir, cuántas repeticiones van a realizarse.
  - Las funciones que devuelvan algún tipo de dato (*return*), no deberían tener más de punto de salida, es decir, sólo deberían tener una única instrucción *return*.
  - Evitar la repetición de código que pueda ser implementado en funciones.