ABSTRACT CLASS: Etiqueta

Etiqueta es una clase abstracta, pues nunca se van a instanciar objetos de dicha clase, pero sí de sus subclases: **EtiquetaSimple, EtiquetaPersona, EtiquetaEspecial**.

Esta clase tampoco tiene atributos, pues estas tres clases solo tienen en común que son etiquetas. La razón por la cual es necesario trabajar esta clase de esta forma es para que en la clase **Imagen** se pueda crear una sola lista con todas las etiquetas asignadas a cada objeto de esa clase **(polimorfismo).** Además, no sabemos cómo se va a comportar esta clase cuando agreguemos la interfaz de usuario y es posible que tengamos que agregarle métodos a dicha clase. Por comodidad, creamos esta clase.

CLASS: EtiquetaSimple

Esta clase se usa para agregar las Etiquetas simples en una Imagen, y **hereda de Etiqueta**. Tal como dice el enunciado, esta etiqueta únicamente debe tener una frase. De esta forma, tiene un atributo que sirve para almacenar dicha frase y un valor DEFAULT para su longitud, pues puede llegar a ser tan largo como quiera el usuario. Además, tiene constructor y getter/setter para dicho atributo.

Atributos:

- **- frase : string** => Atributo donde se almacena la frase de la etiqueta
- **- DEFAULT_LONGITUD_MAX_FRASE:int** => Atributo para la longitud default de la frase (por fijar con el equipo)
- **- serialNumber : int** => Atributo que es un int entre 1 y 1000 que sirve para identificar y diferenciar etiquetas asignadas a una misma imagen

Metodos:

- **+ EtiquetaSimple()** => Constructor parameterless.
- **+ EtiquetaSimple(string frase)** => Sobrecarga constructor
- **+ EtiquetaSimple(string frase, int serialNumber)** => Sobrecarga constructor
- **+ Frase(string frase):bool** => Setter de frase. En el se debe controlar que la etiqueta no supere una cierta longitud máxima soportada por todas las etiquetas simples. Retorna true si el cambio se realizó con éxito y false si no es así.
- **+ Frase() : string** => Getter de frase
- **+ SerialNumber() : int** => Getter del serial number.
- **+ SerialNumber(int serialNumber) : bool** => Setter del serial number. La seleccion de este serialNumber se realiza fuera de la clase. Retorna true si lo logra hacer y false si no. Es posible asignarlo al crear la etiqueta mediante el constructor.

CLASS: EtiquetaPersona

Esta clase sirve para etiquetar personas en una Imagen, y hereda de Etiqueta. Tal como dice el enunciado, en esta etiqueta se deben poder agregar múltiples atributos sobre la persona que se está etiquetando. Además, para crear la etiqueta se debe seleccionar el rostro de la persona en la imagen. Así, uno de los atributos de esta clase es un array de 4 doublés, donde el primero y el

segundo representan el largo y el ancho del rectángulo del rostro y el tercero y el cuarto la distancia del borde izquierdo y superior de la imagen hasta dicho rectángulo (todas medidas en pixeles). Se asume que la interacción para seleccionar el rectángulo lo realiza la interfaz de usuario, cosa que aun no se desarrolla en esta entrega. Además, los colores posibles se guardan en un archivo EColores, las dos opciones de sexo en un ESexo y las nacionalidades en ENacionalidades. Para poder crear una etiqueta de persona, el mínimo parámetro necesario es el nombre y las coordenadas del rectángulo de la cara de la persona. Además, se observa que ningún atributo puede ser tan largo como para necesitar una longitud máxima default.

Atributos:

- **- nombre : string** => Atributo obligatorio que toda etiqueta de persona debe tener
- **- DEFAULT_APELLIDO : string** => Atributo de apellido por default. Deberá ser null.
- **- apellido : string** => Atributo para almacenar el apellido de la persona
- **- DEFAULT_NACIONALIDAD : ENacionalidades** => Atributo de nacionalidad por default. Debera ser none (none pertenece a ENacionalidades)
- **- nacionalidad : ENacionalidades** => Atributo de nacionalidad de las etiquetas, cuyo tipo debe ser el enum ENacionalidades
- **- DEFAULT_COLOR : EColores** => Atributo de color por default para colorOjos y colorCabello. Debera ser none (none pertenece a EColores)
- **- colorOjos : EColores** => Atributo de color de ojos de las etiquetas, cuyo tipo debe ser el enum EColores
- **- colorCabello : EColores** => Atributo de color de cabello de las etiquetas, cuyo tipo debe ser el enum EColores
- **- DEFAULT_SEXO : ESexo** => Atributo de sexo por default. Deberá ser none (none pertenece a ESexo)
- **- sexo : ESexo** => Atributo de sexo de las etiquetas, cuyo tipo debe ser el enum ESexo
- **- DEFAULT_FECHANACIMIENTO : DateTime** => Atributo de fecha de nacimiento por default. Debera ser 01-01-1900.
- **- fechaNacimiento : DateTime** => Atributo de fecha de nacimiento de las etiquetas, cuyo tipo debe ser DateTime
- **- coordenadas : double[4]** => Atributo obligatorio que toda etiqueta de persona debe tener
- **- serialNumber : int** => Atributo que es un int entre 1 y 1000 que sirve para identificar y diferenciar etiquetas asignadas a una misma imagen

Métodos:

- **+ EtiquetaPersona(string nombre, double[4] coordenadas)** => Constructor con menos parámetros posibles
- **+ EtiquetaPersona(string nombre, string apellido, double[4] coordenadas)** => Sobrecarga del constructor
- **+ EtiquetaPersona(string nombre, string apellido, ENacionalidades nacionalidad, double[4] coordenadas)** => Sobrecarga del constructor
- **… Resto de sobrecargas de constructores encadenados entre sí, llegando al último constructor que debe asignar valores para todos los atributos de la clase. El**

encadenamiento se debe realizar con los DEFAULT values. Debe haber un constructor al final que acepte como parámetro además el SerialNumber.

- **+ Nombre(string nombre) : bool** => Setter del atributo nombre. Retorna true si se logra hacer el cambio y false si no
- **+ Nombre() : string** => Getter del atributo nombre
- **+ Apellido(string apellido) : bool** => Setter del atributo apellido. Retorna true si se logra hacer el cambio y false si no
- **+ Apellido() : string** => Getter del atributo apellido
- **+ Nacionalidad(ENacionalidades nacionalidad) : bool** => Setter del atributo nacionalidad. Retorna true si se logra hacer el cambio y false si no
- **+ Nacionalidad() : ENacionalidades** => Getter del atributo nacionalidad
- **+ ColorOjos(EColores color_ojos) : bool** => Setter del atributo colorOjos. Retorna true si se logra hacer el cambio y false si no
- **+ ColorOjos() : EColores** => Getter del atributo colorOjos
- **+ ColorCabello(EColores color_cabello) : bool** => Setter del atributo colorCabello. Retorna true si se logra hacer el cambio y false si no
- **+ ColorCabello() : EColores** => Getter del atributo colorCabello
- **+ Sexo(ESexo sexo) : bool** => Setter del atributo sexo. Retorna true si se logra hacer el cambio y false si no
- **+ Sexo() : ESexo** => Getter del atributo sexo
- **+ FechaNacimiento(DateTime fecha_nacimiento) : bool** => Setter del atributo fechaNacimiento. Retorna true si se logra hacer el cambio y false si no
- **+ FechaNacimiento() : DateTime** => Getter del atributo fechaNacimiento
- **+ Coordenadas(double[4] coordenadas) : bool** => Setter del atributo coordenadas. Retorna true si se logra hacer el cambio y false si no
- **+ Coordenadas() : double[4]** => Getter del atributo coordenadas
- **+ SerialNumber() : int** => Getter del serial number.
- **+ SerialNumber(int serialNumber) : bool** => Setter del serial number. La selección de este serialNumber se realiza fuera de la clase. Retorna true si lo logra hacer y false si no. Ademas, en el caso en que se cargue del archivo csv, es posible asignar el serialNumber mediante el constructor

CLASS: EtiquetaEspecial

Esta clase sirve para agregar etiquetas especiales a una imagen. Según el enunciado, esta debe contener ubicación geográfica, la dirección y el fotógrafo, dejando libre para que el equipo elija otros dos atributos. En este caso, elegimos usar agregar el motivo de la foto (string) y si la foto es una selfie o no (un bool). En este caso, todos los valores son opcionales, así que todos tienen valores default. Para la ubicación geográfica, planeamos usar GMap.NET, que permite insertar mapas en forms, para que el usuario pueda seleccionar la ubicación geográfica de la foto. En este caso, la ubicación esta dada por dos doubles, que corresponden a los valores de latitud y longitud. Ademas, el valor mas largo que se puede ingresar es el motivo, por lo que se agrega un default para su longitud máxima, para evitar cualquier tipo de problemas.

Atributos:

- **- ubicacionGeografica : double[2]** => Atributo de ubicacionGeografica de la etiqueta.
- **- DEFAULT_UBICACION_GEOGRAFICA : double[2]** => Valor default de la ubicación geográfica. Debera ser [0.0 , 0.0].
- **- dirección : string** => Atributo de dirección de la etiqueta
- **- DEFAULT_DIRECCION : string** => Atributo default de la dirección. Debera ser null
- **- fotógrafo : string** => Atributo de fotógrafo de la etiqueta
- **- DEFAULT_FOTOGRAFO : string** => Atributo default de fotógrafo. Debera ser null
- **- motivo : string** => Atributo de motivo para la etiqueta
- **-DEFAULT_MOTIVO : string** => Atributo default del motivo. Debera ser null
- **-DEFAULT_LONG_MAX_MOTIVO : int** => Atributo de la longitud máxima que puede tener motivo
- **- selfie : bool** => Atributo de selfie de la etiqueta. Es true si lo es, y false si no.
- **- DEFAULT_SELFIE : bool** => Atributo default de selfie. Debera ser false
- **- serialNumber : int** => Atributo que es un int entre 1 y 1000 que sirve para identificar y diferenciar etiquetas asignadas a una misma imagen

Métodos

- **+ EtiquetaEspecial()** => Constructor parameterless
- **+ EtiquetaEspecial(double[2] ubicacionGeografica)** => Constructor sobrecargado
- **+ EtiquetaEspecial(double[2] ubicacionGeografica, string dirección)** => Constructor sobrecargado
- **… Resto de sobrecargas de constructores encadenados entre sí, llegando al último constructor que debe asignar valores para todos los atributos de la clase. El encadenamiento se debe realizar con los DEFAULT values cuando alguno de ellos no este dado. Debe haber un constructor que acepte el serialNumber como parámetro.**
- **+ UbicacionGeografica(double[2] ubicacionGeografica) : bool** => Setter de la ubicación geográfica. En el se debe revisar que la ubicación geográfica ingresada es válida (es una ubicación que existe). Retorna true si se logra hacer el cambio y false si no
- **+ UbicacionGeografica() : double[2]** => Getter de la ubicación geografica
- **+ Direccion(string dirección) : bool** => Setter de la dirección. Retorna true si se logra hacer el cambio y false si no (por cualquier motivo).
- **+ Direccion() : string** => Getter de la dirección.
- **+ Fotografo(string fotografo) : bool** => Setter del fotografo. Retorna true si se logra hacer el cambio y false si no (por cualquier motivo).
- **+ Fotografo() : string** => Getter del fotógrafo
- **+ Motivo(string motivo) : bool** => Setter del motivo. Retorna true si se logra hacer el cambio y false si no (por cualquier motivo).
- **+ Motivo() : string** => Getter del motvio.
- **+ Selfie(bool selfie) : bool** => Setter de selfie. Retorna true si se logra hacer el cambio y false si no (por cualquier motivo).
- **+ Selfie() : bool** => Getter de selfie
- **+ SerialNumber() : int** => Getter del serial number.

- **+ SerialNumber(int serialNumber) : bool** => Setter del serial number. La seleccion de este serialNumber se realiza fuera de la clase. Retorna true si lo logra hacer y false si no. Ademas, es posible asignar el serialNumber cuando se crea la etiqueta, mediante el constructor.

CLASS: Imagen

El objetivo de esta clase es poder concentrar en **un solo objeto la imagen en si** (System.Drawing.Bitmap), **las propiedades asignadas por el usuario** (Etiquetas y Valoracion), **las propiedades intrínsecas de la imagen** (EXIF, saturación, resolución, aspect ratio) que se piden por enunciado, **las dos propiedades extra elegidas por el equipo** (identificar si la imagen es clara u oscura, y si usa tecnología HDR), **y el nombre del archivo al que corresponde la imagen**. De esta forma, las propiedades asignadas por el usuario son opcionales en la clase Imagen, mientras que las propiedades intrínsecas se deben asignar cada vez que se crea un objeto Imagen (en el constructor, llamando métodos private dentro de la misma clase para identificar cada propiedad intrínseca. Para el constructor de la clase, es necesario pasarle como parametro el nombre del archivo.

El equipo **supone que hay una carpeta dentro del proyecto denominada Files, en la cual están todos los archivos bitmap que se han cargado** hasta el momento en el programa (la parte de la importación se explica más adelante). Además, debe existir en esta carpeta el archivo **properties.csv** . En este csv, se almacenan (con un formato que se explicara más adelante) todas las propiedades asignadas por el usuario a cada imagen, para poder reiniciar el programa y no perder todas las etiquetas y valoración asignadas a cada imagen, y además, los patrones de búsqueda cargados hasta el momento para las listas inteligentes. En el constructor, a partir del nombre del archivo, se busca dentro de la carpeta Files el bitmap. Desde el bitmap se obtienen y asignan todas las propiedades intrínsecas, y desde el csv todas las propiedades asignadas por el usuario. Es decir, las propiedades intrínsecas de la imagen se calculan cada vez que se crea el objeto Imagen.

Atributos

- **- etiquetas : List<Etiqueta>** => Atributo en el que se almacenan todas las etiquetas asignadas a la imagen
- **- nombre : string** => Atributo en el que se almacena el nombre de la imagen
- **- imagen : System.Drawing.Bitmap** => Atributo en el que se almacena la imagen en si
- **- calificación : int** => Atributo en el que se almacena la calificación hecha por el usuario sobre la imagen. Esta debe estar entre 1 y 5
- **- exif : Dictionary<string atributo, string valor>** => Atributo en el que se almacenan (con un cierto orden aun no establecido) todos los valores EXIF que se puedan obtener del bitmap
- **- saturación : double** => Atributo en el que se almacena la saturación de la imagen
- **- resolucion : int[2]** => Atributo en el que se almacena la resolución de la imagen. Es un array de dos int, pues la resolución se mide de esa forma. Por ejemplo, una resolución de 1920x1080 se debe almacenar como (1920,1080).
- **- relacionAspecto : int[2]** => Atributo en el que se almacena la relación de aspecto de la imagen. Es un array de dos int, pues la relación de aspecto normalmente se mide de la forma k:m, donde k y m son el ancho y la altura de la imagen, que normalmente son enteros. De esta forma, una relación de aspecto 16:9, se almacenaría como (16,9).

- **- hdr : bool** => Atributo que sirve para saber si la imagen usa tecnología HDR o no. Si toma el valor true, usa HDR, y toma el valor false en caso contrario.
- **- claroOscuro : bool** => Atributo que sirve para saber si la imagen se puede catalogar como clara u oscura. Si toma el valor true, es clara, y toma el valor false en caso contrario.

Métodos

- **+ Imagen(string nombreImagen)** => Único constructor de la clase, pues el resto de atributos se obtienen o del csv, o del mismo bitmap en la carpeta Files. Desde este constructor se debe llamar a CargarEtiquetas, CargarValoracion, y el resto de métodos que fijan los atributos de la imagen.
- **- CargarImagen(string nombreImagen) : bool** => Método para cargar el atributo imagen de la clase. Retorna true si lo logra hacer y false si no.
- **- CargarEtiquetas(string nombreImagen) : bool** => Método para cargar todas las etiquetas para nombreImagen que se encuentran dentro del csv. Retorna true si se logró cargar al menos una etiqueta y false si no.
- **- CargarValoracion(string nombreImagen) : bool** => Método para cargar la valoración para nombreImagen que se encuentra dentro del csv. Retorna true si se logro hacerlo y false si no. Si no se logra hacer, el atributo se deja como -1.
- **- CalcularEXIF(string nombreImagen) : bool** => Método para calcular y fijar los valores EXIF de la imagen. Los que no se puedan obtener se dejan como null. Retorna true si se pudo calcular al menos un valor y false si no.
- **- CalcularSaturacion(string nombreImagen) : bool** => Método para calcular y fijar el atributo de la saturación de la imagen. Retorna true si lo logra calcular y false si no. En el caso en que no, el atributo se deja como -1.
- **- CalcularResolucion(string nombreImagen) : bool** => Método para calcular y fijar el atributo resolución. Retorna true si lo logra hacer y false si no.
- **- CalcularRelacionAspecto(string nombreImagen) : bool** => Método para calcular y fijar el atributo relacionAspecto. Retorna true si lo logra hacer y false si no.
- **- CalcularHdr(string nombreImagen) : bool** => Método para saber y fijar el atributo hdr. Retorna true si lo logra hacer y false si no.
- **- CalcularClaroOscuro(string nombreImagen) : bool** => Método para saber y fijar el atributo claroOscuro. Retorna true si lo logra hacer y false si no.
- **+ AgregarEtiqueta(Etiqueta etiqueta) : bool** => Método para agregar una etiqueta al atributo etiquetas de la imagen. Retorna true si se logra hacer y false si no. Este método debe, además, asignar el serialNumber a la etiqueta que se esta agregando, pues conoce que otras etiquetas se han agregado a la imagen y puede asignarle un numero aleatorio diferente.
- **+ EliminarEtiqueta(int serialNumber) : bool** => Metodo para eliminar una etiqueta del atributo etiquetas de la imagen. Retorna true si se logra hacer y false si no. Las etiquetas se diferencian entre si dentro de la misma imagen por el serialNumber.
- **+ Etiquetas() : List<Etiqueta>** => Getter del atributo etiquetas, que retorna el atributo de etiquetas. No hace falta agregar un setter, pues solo se pueden agregar etiquetas y ya existe un método para ello.
- **+ Calificacion() : int** => Getter del atributo valoración. Retorna un numero entre 1 y 5.

- **+ Calificacion(int valoracion) : bool** => Setter del atributo valoracion. En el se debe evaluar si el valor se encuentra entre 1 y 5. Si se logra asignar, retorna true, sino retorna false.
- **+ GuardarPropiedades() : bool** => Metodo para guardar todas las propiedades modificadas en la imagen en el archivo csv (se agregaron o quitaron etiquetas o se cambió la valoracion de la imagen). Retorna true si se logra hacer y false si no.
- **+ Imagen(System.Drawing.Bitmap imagen) : bool** => Metodo para cambiar el atributo imagen del objeto, es el setter de imagen. Es necesario esto porque se espera que otros objetos accedan a la imagen ya sea para aplicarle filtros, agregarle texto, u otros cambios. Entonces, es vital poder cambiar este atributo. Retorna true si se logra setear dicho atributo y false si no.
- **+ Imagen() : System.Drawing.Bitmap** => Getter del atributo imagen. Se espera que otros objetos puedan obtener este atributo para modificarlo (filtros, texto, etc…).
- **+ GuardarImagen(string nombreImagen) : bool** => Metodo para guardar el atributo System.Drawing.Bitmap en la carpeta Files, por si se realiza algún cambio en el, con el nombre nombreImagen. Ademas, este método debe cambiar el atributo nombre del objeto, haciendo que coincida con el nuevo nombre de la imagen
- **- RecalcularAtributos(string nombreImagen) : bool** => Metodo para volver a calcular todos los atributos (saturación, resolución, relacionAspecto,Hdr…) y cargarlos en el objeto. Esto es necesario hacerlo cada vez que se llama a GuardarImagen, pues asumimos que se han realizado cambios en esta.
- **+ Nombre(string nombre) : bool** => Setter del nombre de la imagen. Retorna true si se logro cambiar y false si no
- **+ Nombre() : string** => Getter del nombre de la imagen
- **+ Exif() : Dictionary<string propiedad, string valor>** => Getter de los valores EXIF de la imagen. No hace falta un setter, pues estos valores se calculan a partir de la misma imagen
- **+ Saturacion() : double** => Getter de la saturación de la imagen. No hace falta un setter, pues este valor se calcula a partir de la misma imagen
- **+ Resolucion() : int[2]** => Getter de la saturación de la imagen. No hace falta un setter, pues este valor se calcula a partir de la misma imagen
- **+ RelacionAspecto() : int[2]** => Getter de la relación de aspecto de la imagen. No hace falta un setter, pues este valor se calcula a partir de la misma imagen
- **+ Hdr() : bool** => Getter del atributo hdr de la imagen. No hace falta un setter, pues este valor se calcula a partir de la misma imagen
- **+ ClaroOscuro() : bool** => Getter del atributo claroOscuro de la imagen. No hace falta un setter, pues este valor se calcula a partir de la misma imagen

CLASS: Biblioteca

Esta clase sirve como dice su propio nombre, como Biblioteca del programa. En ella, **se deben guardar todas las imágenes que hasta el momento se han trabajado en el programa**. Las imágenes importadas se guardan en la carpeta Temp (se especifica más adelante). Cada vez que se inicia el programa, se cargan los objetos Imagen (es) a la biblioteca. Además, cada vez que se realizan cambios en alguna imagen la biblioteca debe actualizarse. **De la biblioteca se seleccionan la o las**

imágenes que se quieren llevar al AreaTrabajo (que se define más adelante). En biblioteca se almacenan las listas inteligentes que se piden en el enunciado, mediante un diccionario, cuyas claves son los patrones de búsqueda (un string que se define mas adelante) y los valores son listas de imágenes que cumplen dichos patrones de búsqueda. Cada vez que se actualiza la biblioteca también se debe actualizar la lista inteligente. Biblioteca, además, se encarga de agregar/quitar etiquetas y valoración a las imágenes que tiene.

Atributos

- **- imágenes : List<Imagen>** => Atributo en el que se almacenan todas las imágenes de la biblioteca.
- **- listaInteligente : Dictionary<string patron, List<Image>>** => Atributo donde se almacenan todas las listas inteligentes, con patrones de búsqueda definidos por el usuario. La forma en la que se debe especificar el patron de búsqueda se explica mas adelante.

Métodos

- **+ Biblioteca()** => Constructor de la clase. En este, se deben instanciar los atributos imágenes y listaInteligente, mediante el método CargarBiblioteca().
- **- CargarBiblioteca() : bool** => Metodo que carga la biblioteca. Esto se debe realizar cada vez que se inicia el programa. Se debe buscar en la carpeta Files y en el archivo properties.csv, recorrerlos y crear objetos Imagen que se agregan al atributo imágenes. Además, en properties.csv están definidos los patrones de búsqueda agregados hasta el momento en las listas inteligentes, así que en este método deben cargarse dichos patrones y cargar las listas inteligentes, usando la clase Buscador (que se especifica mas adelante) y cada patrón de búsqueda descrito hasta el momento. Retorna true si se logra hacer sin problemas y false si encuentra algún problema (incoherencia entre el csv y la carpeta, por ejemplo).
- **+ ActualizarBiblioteca() : bool** => Metodo que sirve para actualizar la biblioteca. Cumple la misma función que CargarBiblioteca, pero este método se puede usar durante la ejecución en si del programa, no como CargarBiblioteca, que solo se ejecuta al principio. Retorna true si logra hacerlo sin problemas y false si no. Notar que la única forma de agregar imágenes nuevas a la biblioteca es que esta se encuentre en la carpeta Files.
- **+ AgregarEtiqueta(string nombreImagen, Etiqueta etiqueta) : bool** => Método que sirve para agregar una etiqueta a una determinada imagen. Retorna true si se logra hacer y false si no
- **+ EliminarEtiqueta(string nombreImagen, int serialNumber) : bool** => Método que sirve para eliminar una etiqueta determinada a una determinada imagen, donde serialNumber es el numero serial de la etiqueta a eliminar. Retorna true si se logra hacer y false si no.
- **+ ModificarValoracion(string nombreImagen, int valoración):bool** => Metodo que sirve para modificar la valoración de una determinada imagen.
- **+ AgregarPatronBusqueda(string patron) : bool** => Metodo que sirve para agregar un patron de búsqueda a las listasInteligentes. Retorna true si se logra hacer y false si no. Por defecto, cuando se agrega un patrón de búsqueda y no se han actualizado aun las listas inteligentes, se debe asignar como value al nuevo patron de búsqueda un List<Image> vacía.

- **+ ActualizarListasInteligentes() : bool** => Metodo que sirve para actualizar las listas inteligentes (esto se debe hacer cada vez que se agrega un patron de búsqueda, o cuando se agregan imágenes nuevas a la biblioteca
- **+ Imágenes () : List<Image>** => Getter de las imágenes de la biblioteca. Notar que no hace falta un setter, pues estas se actualizan cada vez que se actualiza la biblioteca
- **+ ListasInteligentes() : Dictionary<string, List<Image>>** => Getter de las listas inteligentes agregadas hasta ahora

CLASS: AreaTrabajo

Esta clase modela un objeto en el cual **se almacenan todas las imágenes a las que se les realiza cambios o se realiza una presentación con ellas**. El encargado de trabajar en ella es el Productor, por lo que el Productor tendrá un AreaTrabajo. **Suponemos que existe una carpeta denominada Temp, en la que se moverán todos los archivos que se quieran de la biblioteca.** Es decir, cuando se quiera trabajar con algunas imágenes, estas deben copiarse del directorio Files y moverse al directorio Temp. AreaTrabajo, contendrá todas las imágenes que existen en Temp.

Atributos

- **+ imágenes : Dictionary <System.Drawing.Bitmap, string>** => En este atributo se almacenan todos los bitmaps que se importen de la biblioteca. El string es el nombre de cada uno.

Metodos

- **+ AreaTrabajo()** => Constructor del área. Únicamente se debe crear el atributo imágenes.
- **+ CargarImagenes(List<string>) : bool** => Este método mueve las imágenes (cuyos nombres se le pasan como una lista de string) de la biblioteca al AreaTrabajo. Es decir, realiza copias de todas las imágenes que se le pidan, y mueve dichas copias al directorio Temp. Luego, además, carga todas esas imágenes al atributo imágenes (desde el directorio Temp). Retorna true si se logra hacer y false si no
- **+ Imagenes() : Dictionary<System.Drawing.Bitmap, string>** => Getter del atributo imagenes
- **+ GuardarImagen(List<string>) : bool** => Metodo que guarda las imágenes cuyos nombres se le pasan en la lista en Temp. Es decir, el Productor modifica el objeto System.Drawing.Bitmap y este método esta encargado de guardar dichas modificaciones hechas a los archivos en la carpeta Temp.
- **+ AgregarImagen(string nombreImagen, System.Drawing.Bitmap imagen) : bool** => Este método sirve para cuando el productor realice cree imágenes nuevas, como en el caso de los mosaicos o de los collage, y necesite agregarlas al AreaTrabajo. Este método, guarda la imagen cuyo nombre es parámetro en el directorio Temp, y la agrega al atributo imágenes de esta misma clase.

- **+ QuitarAreaTrabajo(List<string>) : bool** => Método que sirve para quitar del atributo imágenes aquellas cuyos nombres se le pasan como parámetro. Estas imágenes deben devolverse a Files y se debe actualizar la biblioteca. Retorna true si se logra hacer y false si no.
- **+ EliminarAreaTrabajo(List<string>) : bool** => Método que sirve para eliminar elementos del área de trabajo que no se quieran devolver a Files. Es decir, en caso en que el usuario realice cambios en las imágenes que no se desean guardar, no es necesario guardar los cambios, así que este método elimina del atributo imágenes las imágenes cuyos nombres se le pasan como parámetro. Retorna true si se logra hacer y false si no.

CLASS: Productor

El productor es en el programa el ente encargado de modificar las imágenes (aplicar filtros, realizar mosaicos, collage, así como hacer slideshow o presentaciones). Para ello, se supone que existe una carpeta llamada Temp, en la que se encuentran todas las imágenes que el productor necesita para trabajar. El Productor **no esta encargado de la comunicación con el usuario,** para ello se utilizará otra clase que se definirá mas adelante. El productor utiliza las imágenes cargadas en el AreaTrabajo para modificarlas, y trabajar con ellas. **Una vez realizados todos los cambios, estos se guardan en AreaTrabajo, y luego se agregan a la biblioteca las imágenes que el usuario quiera.** Ademas, el Productor tiene una serie de herramientas que se definirán mas adelante para realizar cambios sobre las imágenes que se encuentran en el AreaTrabajo.

Atributos

- **- areaTrabajo : AreaTrabajo** => Área de donde el productor obtiene las imágenes para su modificación
- **- herramientas : List<Herramienta>** => Herramientas que el productor tiene para poder realizar cambios sobre las imágenes que se agreguen al área de trabajo

Método

- **+ Productor(AreaTrabajo área, List<Herramienta> herramientas)** => Constructor de la clase.
- **+ Presentacion(List<string> nombresImágenes) : bool** => Este método sirve para crear presentaciones con las imágenes cuyos nombres se encuentran en la lista imágenes. Obviamente, estas imágenes deben estar previamente cargadas al AreaTrabajo. En este método, el usuario puede realizar una presentación, seleccionando el mismo cuando quiere cambiar entre las imágenes, tal como se pide en el enunciado. Retorna true si se logró hacer la presentación sin ningún problema y false si no.
- **+ SlideShow(List<string> nombresImágenes) : bool** => Este método sirve para crear un slideshow con imágenes cuyos nombres se encuentran en la lista imágenes. Obviamente, estas imágenes deben estar previamente cargadas al AreaTrabajo. En este método, el

usuario elige las imágenes que quiere que aparezcan y el tiempo que debe durar cada una. Retorna true si se logro hacer el slideshow sin ningún problema y false si no.

- **+ Fusionar(List<string> nombresImágenes) : System.Drawing.Bitmap** => Este método sirve para fusionar las imágenes que se le pasen como parámetro, y retorna un nuevo bitmap, que es la fusión.
- **+ Mosaico(string imagenBase, List<string> nombresImagenes) : System.Drawing.Bitmmap** => Este método sirve para realizar un mosaico de la imagen base con las imágenes que se le pasen como parámetro. Retorna el mosaico nuevo.
- **+ Collage(List<string> nombresImágenes, double[2] tamanoFondo, double[] tamanosImagenes, EColores colorFondo) : System.Drawing.Bitmap** => Este método sirve para realizar un collage con las imágenes cuyos nombres se reciben como parámetro, cuyo tamaño del fondo también, así como el tamaño de cada imagen (el orden de los tamaños debe ser el mismo que el de los nombres de las imágenes) y el color de fondo también se recibe como parámetro. Se debe usar este método si se quiere tener un fondo solido en el collage. Retorna el collage ya hecho.
- **+ Collage(List<string> nombresImágenes, double[2] tamanoFondo, double[] tamanosImagenes, System.Drawing.Bitmap imagenFondo) : System.Drawing.Bitmap** => Cumple la misma función que el método anterior, pero sirve para tener como fondo una imagen, no un fondo sólido.
- **+ Album(List<string> nombresImagenes, int cantFotosXPagina) : System.Drawing.Bitmap** => Metodo que sirve para generar un álbum con las fotos cuyos nombres se le pasan como parámetro así como la cantidad de fotos por página. Retorna el álbum listo.
- **+ Calendario(string[12] nombresImágenes, int anio) : System.Drawing.Bitmap** => Metodo que sirve para generar un calendario con las imágenes cuyos nombres se le pasan como parámetro. Notar que estos deben ser necesariamente 12, una por mes, y que en los parámetros se debe especificar de que anio se quiere el calendario. Retorna el Calendario listo.
- **+ AplicarFiltro(string nombreImagen, EFiltro filtro, string Texto : null) : System.Drawing.Bitmap** => Metodo que sirve para aplicar un filtro especifico a una imagen cuyo nombre se le pasa como parámetro. Se supone que la imagen ya pertenece al AreaTrabajo. El filtro tiene que estar dentro de los posibles, y el productor dentro de sus herramientas tiene todos los filtros posibles. Estos se detallan en el enum EFiltro. En el caso en que se seleccione el filtro para agregar texto a la imagen, es necesario que se le pase dicho texto como parámetro. Como no es necesario para todos los filtros, el texto se deja como un parámetro opcional que por omisión es null.
- **+ ReconocimientoSexoEdad(string nombreImagen) : Dictionary<string propiedad, string valor>** => Este método sirve para aplicar un feature que el equipo decidió hacer, y es reconocer el sexo y la edad de las personas que se encuentran en la foto. Esto se piensa hacer mediante IBMCloud, con una cuenta free creada por el equipo, a través de la cual se pueden hacer peticiones a IBMWatson para que clasifique ciertas imágenes. Específicamente, IBMWatson es capaz sin ningún tipo de entrenamiento por parte del equipo de reconocer la edad y el sexo de las personas que aparecen en una imagen. Retorna un diccionario, que corresponde a las propiedades y valores obtenidos por Watson.

- **+ CensuraPixel(string nombreImagen, double[4] posicion) : System.Drawing.Bitmap** => Método que es parte de los feature que el equipo decidió hacer, que sirve para *censurar* una imagen. En otras palabras, este método pixela la el recuadro que el usuario quiera dentro de la imagen. Los valores de posición corresponden al alto y ancho del recuadro y la distancia desde el borde izquierdo y superior del recuadro.
- **+ CensuraNegro(string nombreImagen, double[4] posicion) : System.Drawing.Bitmap** => Método que funciona igual que el anterior, pero en lugar de pixelar, dibuja un recuadro negro en el lugar a censurar.

ABSTRACT CLASS: Herramienta

Clase abstracta que sirve **para clasificar de la misma forma todos los filtros que existen y las herramientas para censurar y enviar peticiones a IBMWatson**. No es necesario crear objetos de esta clase, por tal razón es abstracta.

CLASS: FiltroWindows

Clase que hereda de Herramienta, y es una herramienta: FiltroWindows. Esta clase tiene pocos métodos.

Métodos

- **+ FiltroWindows()** => Constructor de la clase
- **+ AplicarFiltro(System.Drawing.Bitmap imagen) : System.Drawing.Bitmap** => Metodo que sirve para aplicar el filtro a una imagen, y devuelve el resultado de aplicar el filtro

CLASS: FiltroMirror

Clase que hereda de Herramienta, y es una herramienta: FiltroMirror. Esta clase tiene pocos métodos.

Métodos

- **+ FiltroMirror()** => Constructor de la clase
- **+ AplicarFiltro(System.Drawing.Bitmap imagen) : System.Drawing.Bitmap** => Método que sirve para aplicar el filtro a una imagen, y devuelve el resultado de aplicar el filtro

CLASS: FiltroOldFilm

Clase que hereda de Herramienta, y es una herramienta: FiltroOldFilm. Esta clase tiene pocos métodos.

Métodos

- **+ FiltroOldFilm()** => Constructor de la clase
- **+ AplicarFiltro(System.Drawing.Bitmap imagen) : System.Drawing.Bitmap** => Método que sirve para aplicar el filtro a una imagen, y devuelve el resultado de aplicar el filtro

CLASS: FiltroAjusteAutomatico

Clase que hereda de Herramienta, y es una herramienta: FiltroAjusteAutomatico. Esta clase tiene pocos métodos.

Métodos

- **+ FiltroAjusteAutomatico()** => Constructor de la clase
- **+ AplicarFiltro(System.Drawing.Bitmap imagen) : System.Drawing.Bitmap** => Método que sirve para aplicar el filtro a una imagen, y devuelve el resultado de aplicar el filtro

CLASS: FiltroAgregarTexto

Clase que hereda de Herramienta, y es una herramienta: FiltroAgregarTexto. Esta clase tiene pocos métodos.

Métodos

- **+ FiltroAgregarTexto()** => Constructor de la clase
- **+ AplicarFiltro(System.Drawing.Bitmap imagen, string texto, double[4] posicion) : System.Drawing.Bitmap** => Método que sirve para aplicar el filtro a una imagen, y devuelve el resultado de aplicar el filtro. Recibe además el texto a agregar y la posición en la que hay que agregarlo

CLASS: FiltroBlancoNegro

Clase que hereda de Herramienta, y es una herramienta: FiltroBlancoNegro. Esta clase tiene pocos métodos.

Métodos

- **+ FiltroBlancoNegro()** => Constructor de la clase
- **+ AplicarFiltro(System.Drawing.Bitmap imagen) : System.Drawing.Bitmap** => Método que sirve para aplicar el filtro a una imagen, y devuelve el resultado de aplicar el filtro

CLASS: FiltroSepia

Clase que hereda de Herramienta, y es una herramienta: FiltroSepia. Esta clase tiene pocos métodos.

Métodos

- **+ FiltroSepia()** => Constructor de la clase
- **+ AplicarFiltro(System.Drawing.Bitmap imagen) : System.Drawing.Bitmap** => Método que sirve para aplicar el filtro a una imagen, y devuelve el resultado de aplicar el filtro

CLASS: FiltroCensura

Clase que hereda de Herramienta, y es una herramienta: FiltroCensura. Esta clase tiene pocos métodos.

Métodos

- **+ FiltroCensura()** => Constructor de la clase
- **+ AplicarFiltroPixel(System.Drawing.Bitmap, double[4] posicion) : System.Drawing.Bitmap** => Método que sirve para aplicar el filtro censura de pixel a una imagen, y devuelve el resultado de aplicar el filtro
- **+ AplicarFiltroNegro(System.Drawing.Bitmap, double[4] posicion) : System.Drawing.Bitmap** => Método que sirve para aplicar el filtro censura negro a una imagen, y devuelve el resultado de aplicar el filtro

CLASS: FiltroWatson

Clase que hereda de Herramienta, y es una herramienta: FiltroWatson. Esta clase tiene pocos métodos.

Atributos

- **- ultimaPeticion : Dictionary<string propiedad, string valor>** => Atributo para guardar el resultado de la última petición realizada a Watson, para que los rostros puedan ser recortados si es necesario.
- **- tijera : Tijera** => Tijera para realizar recortes de los rostros encontrados por Watson

Métodos

- **+ FiltroWatson()** => Constructor de la clase. En el se deben instanciar los atributos tijera y ultimaPeticion
- **+ AplicarFiltro(System.Drawing.Bitmap imagen) : Dictionary<string, string>** => Método que sirve para aplicar el filtro a una imagen, devuelve el resultado de aplicar el filtro y lo guarda en ultimaPeticion
- **+ RecortarRostros(System.Drawing.Bitmap imagen) : List<System.Drawing.Bitmap>** => Una vez usado el método anterior, este método sirve para recortar los rostros de las personas que Watson encontró en la imagen. Retorna una lista con todas las imágenes.

CLASS: Tijera

Clase que es usada por Buscador y por el FiltroWatson. El Buscador, la usa para recortar rostros de las personas cuando se quiere realizar la búsqueda especial por persona, como dice el enunciado. Por otro lado, el FiltroWatson la usa para recortar los rostros encontrados a través del método RecortarRostros.

Métodos

- **+ Tijera()** => Constructor de la clase
- **+ Recortar(Dictionary<System.Drawing.Bitmap imagen, double[4] coordenadas>) : List<System.Drawing.Bitmap>** => Metodo que sirve para recortar las imágenes que se le

pasan como parámetro en las coordenadas indicadas (la imagen a recortar es el key y las coordenadas es el key). Retorna una lista con los recortes pedidos.

CLASS: Buscador

Clase que sirve para realizar búsquedas según parámetros que se le especifican.

Atributos

- **- tijera : Tijera** => Tijera para realizar recortes para lo búsqueda especial que se pide por enunciado

Métodos

- **+ Buscador()** => Constructor de la clase
- **+ Buscar(List<Imagen> imagenes, string declaracionBusqueda) : List<Imagen>** => Método que sirve para buscar imágenes dentro de una lista que se le pasa como parámetro, según el patrón de búsqueda que se necesita. Aun no hemos definido como se debe escribir dicho patrón, pero si sabemos que debe incluir and y or, y mezclas de cualquier cantidad de parámetros de búsqueda. Retorna una lista con los objetos Imagen que cumplen con la declaraciónBusqueda.
- **+ BuscarRostro(string nombrePersona, List<Imagen> imágenes) : List<System.Drawing.Bitmap>** => Método que sirve para buscar los rostros de la persona que se le pase como parámetro dentro de la lista de imágenes. Retorna una lista con los recortes de los rostros de la persona buscada.

CLASS: FlujoPrograma

Clase que administra el flujo de ejecución del programa. **Esta clase administra todos los recursos del mismo, así como la interacción con el usuario**. **Todos los métodos de esta clase administran el resto de recursos del programa y mantienen la comunicación con el usuario**

Metodos

- **+ FlujoPrograma()** => Constructor de la clase
- **+ Ejecutar() : void** => Ejecutar el programa
- **- Presentacion() : void** => Ejecutar la presentación del programa
- **- MenuPrincipal() : void** => Ejecutar el menú principal
- **- ImportarImagenes(string path) : void** => Metodo que sirve para importar imágenes. Estas son copiadas desde el path especificado hasta la carpeta Temp (AreaTrabajo). En esta, esta disponible para ser modificada por el programa y si se quiere guardar, se agrega a la biblioteca (carpeta Files). Cuando se realiza esto, se pueden asignar etiquetas a todas las imágenes que se importan.
- **- Buscar() : void** => Ejecutar alguna búsqueda dentro de las imágenes que se han guardado en la biblioteca.
- **- AgregarImagenesAlAreaTrabajo() : void** => Método que sirve para mover imágenes de la biblioteca al AreaTrabajo

- **- Producir() : void** => Ejecutar la producción de imágenes que se han agregado al AreaTrabajo, usando el Productor. En este método se realiza la aplicación de filtros, y los features implementados por el equipo.
- **- Etiquetar() : void** => Agregar alguna etiqueta a una imagen de la biblioteca
- **- ObtenerInfoImagen() : void** => Obtener toda la información posible acerca de una o varias imágenes guardadas en la biblioteca
- **- Valorar() : void** => Valorar una o más imágenes que se encuentran en la biblioteca
- **- ActualizarBiblioteca() : void** => Actualizar la biblioteca actual
- **- AgregarListaInteligente() : void** => Agregar una lista inteligente al programa

ENUM: EFiltro

Enum para **estandarizar la selección de filtros** por parte del usuario. Los filtros posibles son:

1. Windows
2. Mirror
3. OldFilm
4. AjusteAutomatico
5. AgregarTexto
6. BlancoNegro
7. Sepia

ENUM: ESexo

Enum para **estandarizar la selección de sexo** por parte del usuario.

1. Hombre
2. Mujer

ENUM: EColores

Este enum sirve para **estandarizar la entrada de colores a las etiquetas y otros valores del programa**. En vista de que principalmente sirve para los colores de ojos y cabellos no es necesario tener toda la paleta de colores posibles. Los valores que deben aparecer en el son:

1. Rojo
2. Azul
3. Verde

4. Amarillo
5. Naranja
6. Morado
7. Rosa
8. Marrón
9. Blanco
10. Negro
11. Gris
12. Fucsia

ENUM: ENacionalidades

Este enum sirve para **estandarizar la entrada de nacionalidades a las etiquetas de personas**. Los valores posibles son todos los países del mundo.

1. Afganistán
2. Albania
3. Alemania
4. Andorra
5. Angola
6. Antigua y Barbuda
7. Arabia Saudita
8. Argelia
9. Argentina
10. Armenia
11. Australia
12. Austria
13. Azerbaiyán
14. Bahamas
15. Bangladés

16. Barbados
17. Baréin
18. Bélgica
19. Belice
20. Benín
21. Bielorrusia
22. Birmania/Myanmar
23. Bolivia
24. Bosnia y Herzegovina
25. Botsuana
26. Brasil
27. Brunéi
28. Bulgaria
29. Burkina Faso
30. Burundi
31. Bután
32. Cabo Verde
33. Camboya
34. Camerún
35. Canadá
36. Catar
37. Chad
38. Chile
39. China
40. Chipre
41. Ciudad del Vaticano
42. Colombia

43. Comoras
44. Corea del Norte
45. Corea del Sur
46. Costa de Marfil
47. Costa Rica
48. Croacia
49. Cuba
50. Dinamarca
51. Dominica
52. Ecuador
53. Egipto
54. El Salvador
55. Emiratos Árabes Unidos
56. Eritrea
57. Eslovaquia
58. Eslovenia
59. España
60. Estados Unidos
61. Estonia
62. Etiopía
63. Filipinas
64. Finlandia
65. Fiyi
66. Francia
67. Gabón
68. Gambia
69. Georgia

70. Ghana
71. Granada
72. Grecia
73. Guatemala
74. Guyana
75. Guinea
76. Guinea ecuatorial
77. Guinea-Bisáu
78. Haití
79. Honduras
80. Hungría
81. India
82. Indonesia
83. Irak
84. Irán
85. Irlanda
86. Islandia
87. Islas Marshall
88. Islas Salomón
89. Israel
90. Italia
91. Jamaica
92. Japón
93. Jordania
94. Kazajistán
95. Kenia
96. Kirguistán

97. Kiribati
98. Kuwait
99. Laos
100. Lesoto
101. Letonia
102. Líbano
103. Liberia
104. Libia
105. Liechtenstein
106. Lituania
107. Luxemburgo
108. Macedonia del Norte
109. Madagascar
110. Malasia
111. Malaui
112. Maldivas
113. Malí
114. Malta
115. Marruecos
116. Mauricio
117. Mauritania
118. México
119. Micronesia
120. Moldavia
121. Mónaco
122. Mongolia
123. Montenegro

124. Mozambique
125. Namibia
126. Nauru
127. Nepal
128. Nicaragua
129. Níger
130. Nigeria
131. Noruega
132. Nueva Zelanda
133. Omán
134. Países Bajos
135. Pakistán
136. Palaos
137. Panamá
138. Papúa Nueva Guinea
139. Paraguay
140. Perú
141. Polonia
142. Portugal
143. Reino Unido
144. República Centroafricana
145. República Checa
146. República del Congo
147. República Democrática del Congo
148. República Dominicana
149. República Sudafricana
150. Ruanda

151. Rumanía
152. Rusia
153. Samoa
154. San Cristóbal y Nieves
155. San Marino
156. San Vicente y las Granadinas
157. Santa Lucía
158. Santo Tomé y Príncipe
159. Senegal
160. Serbia
161. Seychelles
162. Sierra Leona
163. Singapur
164. Siria
165. Somalia
166. Sri Lanka
167. Suazilandia
168. Sudán
169. Sudán del Sur
170. Suecia
171. Suiza
172. Surinam
173. Tailandia
174. Tanzania
175. Tayikistán
176. Timor Oriental
177. Togo

178. Tonga
179. Trinidad y Tobago
180. Túnez
181. Turkmenistán
182. Turquía
183. Tuvalu
184. Ucrania
185. Uganda
186. Uruguay
187. Uzbekistán
188. Vanuatu
189. Venezuela
190. Vietnam
191. Yemen
192. Yibuti
193. Zambia
194. Zimbabue