EVALUACIÓN

SISTEMAS MULTIMEDIA

CURSO 2018-2019

Se pretende realizar una aplicación multimedia que permita gestionar de forma integral diferentes tipos de medios: **gráficos**, **imágenes**, **sonido** y **vídeo**. Sobre cada uno de dichos medios se podrán llevar a cabo diferentes tareas que, en función del medio, abarcarán desde la creación y/o captura hasta la edición, reproducción y procesamiento. Para ello, la aplicación contará con un conjunto de menús y barras de herramientas que permitirán llevar a cabo dichas tareas.

A continuación se detallarán los objetivos y contenidos particulares, así como las acciones a aplicar sobre cada medio. El diseño del interfaz queda abierto a criterio del estudiante, si bien la evaluación tendrá en cuenta la complejidad del diseño, por lo que se recomienda usar el mayor número posibles de elementos Swing.

Las especificaciones indicadas en este documento son los requisitos mínimos, si bien se podrán incorporar todas las alternativas que se consideren oportunas para mejorar la calidad de la aplicación.

Medios e interfaz de usuario

La aplicación permitirá trabajar, de forma integrada, con todos los medios estudiados a lo largo del curso: **gráficos**, **imágenes**, **sonido** y **vídeo**. Para ello, tendrá un escritorio central en el que podrán alojarse ventanas internas de diferentes tipos en función del medio[1]. A continuación se describen las principales características del entorno, si bien se podrán introducir cuantas mejoras se deseen para optimizar la funcionalidad o el aspecto visual.

• Gráficos e imágenes

La aplicación permitirá la gestión conjunta de gráficos e imágenes en un mismo tipo de ventana[2]; así, dada una imagen (ya sea creada nueva, leída o capturada), podremos tanto (1) dibujar sobre ella como (2) procesarla.

La manipulación de estos medios requerirá la incorporación de un tipo específico de ventana interna capaz de mostrar imágenes, tanto las leídas de un fichero como las creadas nuevas por el usuario o generadas al capturar una imagen instantánea a partir de un vídeo o webcam. Cada ventana mostrará una única imagen (existiendo, por tanto, tantas ventanas como imágenes estemos tratando), incluyendo barras de desplazamiento en caso de que la imagen sea mayor que la zona visible. El título de la ventana corresponderá al (1) nombre del fichero, si es una imagen abierta o guardada, (2) "nueva", si ha sido creada por el usuario, o (3) "captura", si es una instantánea captada de un vídeo o de la webcam. Además, deberá indicar (entre corchetes) el espacio de color en el que está la imagen; si se tratase de una imagen asociada a una banda, deberá indicarse a qué banda (número o letra) corresponde. *[Opcional: Al mover el ratón sobre la imagen, indicar en la barra de estado las coordenadas del pixel sobre el que se está situado, así como su valor (color o nivel de gris, según el tipo de imagen)]*

Sobre las imágenes que aparecen en estas ventanas se podrá:

---

[1] El hecho de que haya diferentes tipos de ventanas internas, cada una correspondiente a una clase distinta, hay que tenerlo en cuenta a la hora de usar el método `getSelectedFrame`; por ejemplo, si se va a aplicar una operación sobre una imagen hemos de comprobar que la ventana activa es del tipo adecuado (es decir, que contiene una imagen). Para abordar este punto, se aconseja crear una superclase `VentanaInterna` de la que hereden diferentes subclases correspondientes a diferentes tipos de ventanas.

[2] Véase práctica 8.

- **Dibujar** usando las formas y atributos incluidos en la correspondiente barra de dibujo (véase siguiente sección). Sólo se podrá dibujar sobre el área de la imagen, por lo que estas ventanas tendrán definidas como área de visualización (área "clip") el área rectangular correspondiente a la imagen.
- **Procesar** imágenes, usando para ello las operaciones incluidas en la correspondiente barra de herramientas (véase siguiente sección). En el procesamiento, no es necesario aplicar el efecto sobre las formas dibujadas (solo sobre la imagen).

En posteriores secciones se detallan los requisitos exigidos tanto en la generación de gráficos como en el procesamiento de imágenes.

• Sonido

La aplicación permitirá tanto la **reproducción** como la **grabación** de audio[3]. Para ello contará con una lista de reproducción, que tendrá una lista desplegable asociada, de forma que al abrir un nuevo fichero de audio éste se incluiría en dicha lista de reproducción[4]. Además, la aplicación tendrá:

- Un botón para **reproducir** sonido. Al pulsar el botón, se reproduciría el audio que esté seleccionado en ese momento en la lista de reproducción.
- Un botón para **parar** la reproducción o la grabación.
- Un botón para **grabar** audio, de forma que al pulsarlo se iniciará el proceso de grabación, que terminará cuando se pulse el botón de parada. El sonido se grabará en el fichero indicado por el usuario[5] y se añadirá a la lista de reproducción.

Tanto la lista de reproducción, como los botones de reproducir, parar y grabar, estarán dentro de una barra de herramientas.

• **Vídeo** [Opcional]

La aplicación permitirá tanto la reproducción como la captura de vídeo; para ello, se definirán ventanas internas específicas para cada tarea[6]. Así, la **reproducción de vídeo** requerirá la incorporación de un tipo específico de ventana que dispondrá de una zona de visualización central donde se mostrará el vídeo. La ventana tendrá como título el nombre del fichero que se está reproduciendo, existiendo tantas ventanas como vídeos tengamos abiertos. Además, la aplicación tendrá:

- Un botón para **reproducir** video, que será el mismo que para la reproducción de sonido[7]. Al pulsar el botón, se reproduciría el video de la ventana seleccionada en ese momento (si esta fuese de tipo video).
- Un botón para **parar** la reproducción, que será el mismo que para la reproducción de sonido.

Deberá incluirse, por tanto, una barra común para sonido y video, que incorpore los botones de reproducir/detener y que éstos gestionen la reproducción del medio seleccionado, ya sea sonido o vídeo (en el caso de audio, el seleccionado en la lista de reproducción; en el caso del vídeo, el correspondiente a la ventana interna seleccionada).

---

[3] Véanse prácticas 13 y 14. Nótese que la práctica 13 sólo trabajaba con la *Java Sound API* y, por tanto, con los formatos y códecs soportados por dicha API; para la evaluación, se puede ampliar la funcionalidad con la incorporación de *JMF* o VLCj (práctica 14) y los códec que estos soportan.

[4] Al añadir un nuevo sonido a la lista de reproducción, éste no debe de empezar a reproducirse. Para reproducir el sonido, deberá de pulsarse el botón *play* (*stop* para parar).

[5] A la hora de seleccionar el fichero donde almacenar el sonido grabado, existen dos opciones: que se elija el fichero antes de comenzar la grabación, o que dicho fichero se elija al finalizar el proceso de grabación.

[6] Véase práctica 14.

[7] Nótese que en caso de usar JMF no podrá usarse el panel del control de *Player*, ya que es obligatorio controlar la reproducción desde los botones incorporados en la aplicación para este propósito.

Adicionalmente, la **captura a través de webcam** también requerirá un tipo específico de ventana interna que vaya mostrando la secuencia que esté captando la cámara.

Requisitos de carácter general

La aplicación contará con una barra de herramientas de carácter general que incluirá, al menos, los siguientes botones asociados a las clásicas opciones de archivo (todos ellos tendrán que tener asociado un icono y un "*ToolTipText*"):

- **Nuevo**. Permite crear una nueva imagen (que aparecerá en una nueva ventana). El usuario deberá indicar el tamaño de la imagen (a través de un diálogo que se lance en el momento de la creación o asociado a un menú general de opciones).
- **Abrir**: Abre el dialogo "Abrir fichero" y permite seleccionar un fichero de imagen, sonido o vídeo. Dependiendo del tipo de fichero, éste se mostrará en un tipo de ventana u otra (o en la lista de reproducción, en caso del audio). Los formatos reconocidos serán los estándares manejados por Java.

  Se usará el mismo botón para abrir cualquier medio, teniendo el diálogo abrir asociado filtros para los tipos de ficheros reconocidos. Si se produce algún error al abrir el medio (formato desconocido, etc.) se lanzará un diálogo que informe del problema.

  Asociado al diálogo abrir definiremos filtros para que sólo muestre extensiones correspondientes a ficheros de formatos admitidos.
- **Guardar**: Lanza el dialogo "Guardar fichero" y permite guardar la imagen de la ventana que esté seleccionada, incluyendo las figuras dibujadas (esta opción estará desactivada para el caso del vídeo). Se podrá almacenar en cualquiera de los formatos reconocidos por Java (JPG, PNG, etc.), obteniendo dicho formato a partir de la extensión indicada por el usuario. Asociado al diálogo guardar definiremos filtros para que sólo muestre extensiones correspondientes a formatos admitidos.

En la barra de menú, además del menú **Archivo** (que incluiría las opciones "Nuevo", "Abrir" y "Guardar" indicadas anteriormente), se debe de incluir la opción **Ver** (que permitirá ocultar/visualizar las diferentes barra de herramientas) y **Ayuda** (que tendrá la opción "Acerca de" que lanzará un diálogo con el nombre del programa, versión y autor).

Requisitos de dibujo

La aplicación permitirá dibujar diferentes formas geométricas (líneas, rectángulos, etc.) con diferentes atributos (color, grosor, etc.) sobre una imagen. Para ello, se incorporará una barra de herramientas que dé acceso a todos los elementos necesarios para poder dibujar, incluyendo formas y atributos (los elementos de esta barra tendrán que tener asociado un "*ToolTipText*").

A la hora de dibujar…

El usuario podrá dibujar sobre cualquier imagen utilizando la forma y atributos seleccionados (véanse siguientes subsecciones). Para ello, hay que tener en cuenta los siguientes requisitos de carácter general:

- El lienzo mantendrá todas las figuras que se vayan dibujando.
- Cada figura tendrá sus **propios atributos**[8], independientes del resto de formas (es decir, no compartirán los mismos valores). Cuando se dibuje la forma por primera vez, ésta usará los atributos que estén activos en ese momento (que no tienen por qué coincidir con los de las figuras que ya estén en el lienzo). La

[8] Nótese que, a diferencia de la práctica 7, en la que todas las formas se mostraban con los mismos atributos, en este caso cada forma tendrá asociado un conjunto de atributos propio.

consecución de este requisito **implica obligatoriamente la definición de una jerarquía de clases** asociadas a las formas y sus atributos.

- El usuario podrá **editar las figuras** ya dibujadas. Para ello:

  - ✓ Deberá de existir un panel en el que se **muestre la lista de figuras**[9] que hay en el lienzo activo[10]. Dicha lista tendrá que actualizarse cada vez que cambiemos de ventana[11,12] y, dada una ventana de dibujo, cada vez que añadamos una nueva figura[13].

  - ✓ El usuario podrá seleccionar cualquier figura a través del panel de figuras[14]. La figura seleccionada deberá identificarse mediante un rectángulo (*boundingbox*) discontinuo a su alrededor.

  - ✓ Al seleccionar una figura, deberán de **activarse sus propiedades en la barra de dibujo**, así como deseleccionarse la forma de dibujo que estuviese activa. Por otro lado, si se pulsa el botón de una forma de dibujo (se entiende que para dibujar una nueva figura), automáticamente deberá de deseleccionarse la forma seleccionada (si la hubiese) [15].

  - ✓ Para la figura seleccionada, se podrán **editar sus atributos**, esto es, se podrá modificar cualquiera de sus propiedades (color, trazo, relleno, etc.) sin más que cambiarla en la barra de herramientas (y los cambios se tendrán que ver reflejados en la forma).

  - ✓ Se podrá **cambiar la localización** de la figura seleccionada. Para ello, se incluirá como información en la barra de dibujo la coordenada (x,y) en la que se encuentra localizada la forma[16,17]. El usuario podrá editar esa

---

[9] La lista podrá ser una lista estándar (*JList*) o una lista desplegable (JComboBox), a elección del estudiante. En caso de una lista desplegable, ésta deberá de estar en la barra de herramientas de dibujo; en caso de una lista estándar, podría estar, por ejemplo, en el lateral derecho de un panel divisor (y, en este caso, en el izquierdo podría ubicarse el escritorio).

[10] Se aconseja crear la lista (*JList* o *JComBobox*) estableciendo la clase de los ítems igual a la superclase definida para las figuras (en NetBeans, este parámetros se puede cambiar desde la sección Código de Propiedades). En este caso, recordar que, por defecto, la lista mostrará el *String* devuelto por el método *toString()* del objeto (se aconseja sobrecargarlo en cada clase). Para visualizaciones más avanzadas de los ítems de la lista, seguir las recomendaciones de la práctica 7 para el caso de la lista desplegable de colores.

[11] Se aconseja que la clase "*Lienzo2D*" incluya un método que devuelva la lista de figuras asociada; dicha lista será la que deberá mostrar la lista de figuras.

[12] Recordemos (véase práctica 2) que si queremos añadir elementos a un *JList* (o *JComboBox*) debemos de hacerlo a través de su modelo de datos (accesible a través del método *getModel()*). Para simplificar el proceso, se recomienda asignarle a la lista el modelo por defecto mediante *lista.setModel(new DefaultListModel())*, y añadir elementos mediante *((DefaultListModel)lista.getModel()).addElement(…)*.

[13] Esto implica que, de alguna forma, el "*Lienzo2D*" debe notificar que una nueva figura se ha añadido cada vez que esto ocurra (recordemos que el Lienzo es independiente y su diseño no debe estar condicionado por la ventana en la que se incluya). La forma más adecuada de hacerlo es definiendo un nuevo evento (clase de diseño propio) que se lance cuando se añada una nueva figura a la lista de figuras del lienzo (véase práctica 7, apéndice 2 sobre definición de nuevos eventos). No obstante, existen otras opciones "menos elegantes" también válidas, por lo que se deja al estudiante libertad para afrontar este requisito.

[14] Nótese que en este caso no hay un botón de "edición", como en la práctica 7, que distinga entre un proceso de creación de nuevas figuras y el de edición de las ya existentes. Consecuentemente, tampoco se seleccionan las figuras haciendo clic sobre ellas (como ocurría en la práctica 7).

[15] Cuando se seleccione una forma de dibujo tras un proceso de edición (en el que la barra de dibujo muestra los atributos de la figura seleccionada), se podrá (1) volver a activar los atributos que hubiese antes de la selección o (2) mantener los atributos de la última figura seleccionada y que sean esos los que se usen para la nueva figura; se deja a criterio del estudiante elegir una u otra opción.

[16] Se pueden usar, por ejemplo, campos de texto (uno para la 'x' y otro para la 'y').

[17] Nótese que la localización es una propiedad como cualquier otra (color, grosor, etc.) y la barra de dibujo tendrá que mostrar el valor correspondiente a la figura que se esté pintando o editando. Para simplificar la práctica, no es necesario que el campo se vaya actualizando en una figura de nueva creación mientras esta se

posición y, al hacerlo, la figura deberá de moverse a la localización indicada[18].

[Opcional: adicionalmente al uso del campo de localización, cambiar la localización de una figura mediante "arrastrar y soltar"][19]
[Opcional: redimensionar la figura seleccionada]

• Formas de dibujo

La aplicación deberá permitir dibujar, al menos, las siguientes formas geométricas:

- Línea recta
- Rectángulo
- Rectángulo redondeado *[opcional]*
- Elipse
- Arco *[opcional]*
- Curva con un punto de control *[opcional]*
- Curva con dos puntos de control *[opcional]*
- Trazo libre *[opcional]*
- Polígono *[opcional]*
- Forma personalizadas (área) que defina una nueva figura *[opcional]*
- Texto formateado[20] *[opcional]*

En la barra de herramientas aparecerá un botón (con icono) por cada forma de dibujo disponible[21]. Se usarán botones de dos posiciones agrupados, estando siempre pulsada la forma de dibujo que se va a pintar (salvo que haya una figura del lienzo seleccionada a través del panel de figuras, en cuyo caso ningún botón estará activo).

• Atributos de dibujo

El usuario podrá elegir los atributos con los que se pintarán las formas (el diseño y organización del interfaz queda abierto al criterio del estudiante, si bien la complejidad del mismo se tendrá en cuanta en la evaluación). Al menos, deberán incluirse los siguientes atributos:

- **Color**. El usuario podrá elegir el color del trazo y el de relleno. Deberán aparecer una serie de colores predeterminados y, además, la posibilidad de lanzar un diálogo de selección de colores. En este apartado deberá de haber una referencia al color de trazo y de relleno actuales[22].

---

dibuja, tan solo deberá de hacerlo para las figuras que se seleccionen a través del panel de figuras. Se aconseja que, en el diseño de clases propio para modelar las figuras, se incluya un método "*getLocation*" que devuelva la localización de una figura.

[18] Nótese que en este caso no se mueven las figuras con el "arrastrar y soltar" usado en la práctica 7.

[19] Nótese que sería adicional a la indicada como obligatoria, no sustituta del campo en la barra de dibujo

[20] La escritura del texto se podrá hacer directamente sobre el área de dibujo o bien utilizando un campo de texto o un diálogo previo en el que introducir la cadena. Independientemente de la forma de introducir el texto, éste deberá de aparecer en el punto de la imagen donde se haga el clic y con el formato indicado.

[21] Nótese que, de las formas anteriores, las cinco primeras y la octava corresponden a un dibujo de un solo paso (entendido un paso como una secuencia *pressed-dragged-released*), la sexta de dos pasos, la séptima de tres pasos, y la novena de tantos pasos como lados tenga el polígono (en este caso, se usará el doble clic para indicar el último vértice). Esto implica que, según la forma, habrá que llevar un control del "paso" por el que se va (y la tarea a hacer en dicho paso). En el caso de las formas usadas en las prácticas, todas eran de un solo paso (se gestionaban con una sola secuencia *pressed-dragged-released*); para esta evaluación, nótese que se incluyen nuevas formas de más de un paso.

[22] Nótese que en las prácticas realizadas durante el curso no se distinguía entre color de trazo y color de relleno (se usaba el mismo para ambos); en la evaluación hay que incorporar esta distinción.

- **Trazo**. Se podrán modificar el grosor y el tipo de discontinuidad del trazo. En este último caso, se podrán dibujar, al menos, líneas continuas o líneas punteadas.

  [Opcional: estilos final y de unión de línea, más tipos de discontinuidad]

- **Relleno**. El usuario podrá elegir entre tres opciones a la hora de rellenar: no rellenar, rellenar con un color liso o rellenar con un degradado. En el caso del degradado, éste se aplicará utilizando los dos colores (frente y fondo) seleccionados en ese momento (para la dirección del degradado, se ofrecerá al menos dos posibilidades: horizontal y vertical).

  [Opcional: más direcciones de degradado, relleno mediante imágenes predeterminadas, relleno radial]

- **Alisado de bordes**. El usuario podrá activar/desactivar la mejora en el proceso de renderizado correspondiente al alisado de bordes.

  [Opcional: incluir otras mejoras del renderizado][23]

- **Transparencia**. Se podrá establecer un grado de transparencia asociado a la forma (por ejemplo, mediante un deslizador)[24]

y opcionalmente:

- **Fuente del texto** (en caso de que se incluya el texto como forma). Se podrá establecer la fuente, tamaño y estilo del texto.
- **Reglas en la composición**. Se podrán definir reglas para combinar las nuevas formas con las ya existentes.
- **Transformaciones sobre la forma**. Se podrán aplicar traslaciones, escalados, rotaciones y deformaciones sobre la forma a dibujar.

Cuando se cambie de una ventana interna a otra, los botones de forma y atributos de la barra de herramientas deberán activarse conforme a la forma y atributos de la ventana activa.

Requisitos de procesamiento de imágenes

La aplicación permitirá aplicar un conjunto de operaciones que se podrán llevar a cabo sobre cualquier imagen[25]. Cada una de esta operaciones se incluirán en una barra de herramientas donde cada elemento tendrán que tener asociado un "*ToolTipText*". Al menos, se deberán de poder realizar las siguientes operaciones:

- Duplicar, que creará una nueva "ventana imagen" con una copia de la imagen[26].
- Modificar el brillo mediante un deslizador.
- Filtros de emborronamiento, enfoque y relieve.
- Contraste normal, iluminado y oscurecido.
- Negativo (invertir colores).
- Extracción de bandas
- Conversión a los espacios RGB, YCC y GRAY
- Giro libre mediante deslizador.
- Escalado (aumentar y disminuir).
- Tintado (con el color de frente seleccionado en ese momento)
- Ecualización

[23] Cada mejora podrá ser activada/desactivada por el usuario de forma individual.
[24] Nótese que no es sólo la semitransparencia (como en la práctica 7), sino que el usuario podrá definir el grado de transparencia que vaya desde opaco hasta totalmente transparente.
[25] Véanse prácticas 9-12.
[26] Ha de ser una copia, no una referencia a la original (es decir, no corresponde a una asignación entre variables, sino a la creación de una imagen nueva).

- Sepia[27].
- Umbralización[27] basada en niveles de gris con deslizador para modificar umbral[28].
- Un operador "*LookupOp*" basado en una función definida por el estudiante[29]
- Una nueva operación de diseño propio aplicada componente a componente[27,30,31].
- Una nueva operación de diseño propio aplicada pixel a pixel[27,31,32].

y opcionalmente:

- Suma y resta de imágenes.
- Mezcla de imágenes mediante deslizador[33].
- Detección de fronteras *Sobel.*
- Tintado mediante deslizador (que indique el grado de mezcla).
- Umbralización basada en color [34].
- Mostrar el histograma.
- Cualquier otra operación de diseño propio.

Las operaciones se irán aplicando de forma concatenada, es decir, una operación se aplicará sobre el resultado de operaciones aplicadas anteriormente. En el caso del **brillo**, el deslizador permitirá ir variando el brillo sobre la imagen que haya en ese momento, no sobre el resultado del cambio de brillo (es decir, si deslizamos el brillo a su máximo valor –lo que implicaría ver la imagen en blanco-, si después reducimos dicho valor se tiene que volver a ver la imagen inicial). Una vez que se elija otra operación, el brillo se aplicará de forma definitiva (y se concatenará con el resto de operaciones).

Respecto al **operador basado en una función** definido por el estudiante, deberá indicarse claramente en la documentación qué función se aplica, mostrar una representación gráfica de la misma y explicar qué comportamiento se espera al aplicar esa función. Además, deberá explicarse qué parámetros tiene la función y cómo influyen en el resultado del operador.

En relación a las dos **operaciones de diseño propio**[35], el estudiante deberá indicar claramente en la documentación qué operaciones ha implementado, justificar qué hace cada una de ellas y por qué, así como la correspondiente formulación matemática. Deberá especificarse qué parámetros tiene la operación e incluir en el interfaz los elementos necesarios para su selección y aplicación.

Tanto en el operador basado en una función, como en las dos operaciones de diseño propio, la documentación deberá de incluir ejemplos comentados donde se apliquen los mismos a imágenes reales. Si estas operaciones no están documentadas, o la explicación y justificación de las mismas es insuficientemente, no se dará por válidas.

---

[27] Necesario crear clase propia.

[28] Para la umbralización aplicaremos la basada en niveles de gris. En este caso, el deslizador permitirá ir variando el umbral e ir viendo el resultado parcial (aplicado sobre la imagen inicial). Una vez que se elija otra operación, la umbralización se aplicará de forma definitiva

[29] Ha de ser una función no vista en clase ni disponible en *sm.image* (por lo tanto, no serían válidas las vistas en clase de teoría ni la función seno implementada en prácticas)

[30] La operación se aplicará a cada componente de forma independiente, sin influir en el resultado el resto de componentes del pixel (ni los píxeles del entorno)

[31] Ha de ser una operación nueva no vista en clase, ni desarrollada en ninguno de los guiones de prácticas.

[32] La operación se aplicará pixel a pixel considerando todos los componentes del pixel, es decir, en el resultado influirán todos los componentes (recuérdese, por ejemplo, el caso de la operación "sepia" vista en prácticas).

[33] La mezcla de imágenes usará un deslizador para seleccionar qué proporción de cada imagen es mezclada. Se deja a elección del estudiante la forma en la que se seleccionan las imágenes a mezclar, así como la forma de determinar el fin de la operación (p.e., al elegir otra operación).

[34] En este caso, el usuario podrá elegir entre dos opciones: umbralizar sobre los niveles de gris o umbralizar sobre el espacio de color. En el segundo caso, además del umbral, el usuario deberá elegir el color central de la umbralización que, por defecto, será el color de frente actualmente seleccionado (el deslizador será válido en ambos tipos de umbralizaciones).

[35] Implica crear una clase propia heredando de `BufferedImageOp` en la línea vista en las prácticas 11-12. En este caso, se tratará de operaciones propias y distintas de las implementadas en prácticas.

Requisitos de sonido

La aplicación permitirá tanto reproducir como grabar sonidos, tal y como se indicó anteriormente (sección Medios→Sonido). La reproducción se aplicará sobre los formatos y códecs reconocidos por Java. Respecto a la grabación, se pueden fijar los parámetros de digitalización (codificación, resolución, frecuencia de muestreo, etc.) y formato de fichero[36].

[Opcional: mostrar la evolución de la reproducción en un contador de tiempo o en una barra de progreso]

[Opcional: permitir la selección de los parámetros de digitalización y formato de fichero]

Requisitos de video [Opcional][37]

La aplicación permitirá tanto **reproducir vídeo** como mostrar la secuencia que esté captando la webcam (para ello contará con los correspondientes elementos en la barra de herramientas). Como se indicó anteriormente (sección Medios→Video), estas tareas estarán asociadas a ventanas internas específicas que dispondrán de una zona de visualización central; además, el usuario podrá controlar la reproducción (comenzar, pausar, etc.) mediante los botones situados a tal efecto en la barra de herramientas. La reproducción se aplicará sobre los formatos y códec reconocidos por Java

Adicionalmente, la aplicación permitirá al usuario **capturar imágenes** de la cámara o del vídeo que se esté reproduciendo; concretamente, lo hará de la ventana que esté activa, siempre y cuando sea una ventana de tipo "reproducción de video" o "webcam"[37]. La imagen capturada se mostrará en una nueva ventana interna.

Documentación

La documentación tendrá que seguir las pautas propias de la Ingeniería del Software (requisitos, análisis, diseño y codificación), haciendo especial hincapié en la descripción y justificación de las clases de diseño propio. Para la codificación, es obligatorio documentar el código usando *javadoc* y generar la correspondiente API (especialmente importante en el caso de bibliotecas propias). Indicar las fuentes (bibliográficas o de código) utilizadas en la elaboración de las prácticas.

Una documentación **suficiente** y **adecuada** será **imprescindible** para la corrección de las prácticas, siendo el primer criterio de corte para su evaluación[38].

Javadoc

Todas las clases de diseño propio deberán estar documentadas usando Javadoc[39]. Debe incluirse tanto la descripción de la clase, como la de sus variables y métodos miembro (en éste último caso, incluyendo tanto parámetros como información devuelta).

---

[36] Estos requisitos coindicen con lo desarrollado en la práctica 13.

[37] Estos requisitos coindicen con lo desarrollado en la práctica 14. No obstante, dada la proximidad de la fecha de evaluación y que la correspondiente sesión de prácticas fue la última semana del curso, este requisito se considera opcional para la convocatoria ordinaria de Junio (no así para la extraordinaria de Julio, que será obligatoria). En cualquier caso, puesto que se trata de la práctica desarrollada en clase, se invita al estudiante a que la incorpore como parte de la evaluación final.

[38] La evaluación comienza con la lectura de la documentación; si ésta no explica en detalle lo realizado, justificando el porqué de los diseños y soluciones adoptadas, no procederá la ejecución de la aplicación (y, consecuentemente, implicará el suspenso en la evaluación).

[39] Para quien no conozca su uso, pueden consultarse los siguientes enlaces de la Wikipedia o de Oracle.

A modo de ejemplo, pueden verse las descripciones de las clases relativas a gráficos, imágenes, etc. usadas a lo largo de la asignatura[40].

Entrega

La entrega se realizará a través de la página web de decsai.ugr.es. Se deberá entregar un fichero comprimido (.zip o .rar) que incluya:

- Fichero ejecutable .jar de la aplicación. Este fichero deberá estar en la raíz del fichero comprimido y deberá empaquetar todas las librerías[41].
- Código fuente (proyectos Neatbean completos) incluyendo bibliotecas propias.
- Documentación en PDF (localizable en la raíz del fichero comprimido)
- API generada usando Javadoc (localizable en la raíz del fichero comprimido)

La fecha de entrega será la fijada en el calendario de exámenes, esto es, el día **14 de junio de 2019**.

Defensa

Habrá dos fechas en las que se convocará para defensa/examen:

- El día **14 de junio de** a las **17:00**, correspondiente a la fecha fijada en el calendario de exámenes.
- Días **20-21 de junio** (posterior a la fecha de la entrega).

El estudiante será convocado a la correspondiente defensa, si procede, por correo electrónico (adicionalmente, se publicará la convocatoria en la web de la asignatura). En caso de que sea en la primera defensa, ésta está fijada para el día **14 de junio** a las **17:00**; para el caso de la segunda, ésta se notificará por correo el día y hora de dicha defensa. En ambos casos, y si se trata de defensa/examen escrito, la duración del mismo será de 3h.

La defensa podrá incluir **examen escrito** relativo al contenido de la práctica y los conocimientos adquiridos a lo largo del curso. En este caso, la nota final de la asignatura vendrá determinada por la nota obtenida en el examen escrito.

[40] Recuérdese que, además de la API oficial, desde NetBeans se puede acceder al código fuente de las clases y ver el uso de Javadoc en la documentación de las mismas.
[41] Véase anexo de la práctica 7.

Anexo: Aclaraciones

En este anexo se incluyen algunas de las aclaraciones/sugerencias explicadas en clase a raíz de las consultas realizadas por los estudiantes en relación a este ejercicio de evaluación:

- Es **obligatorio definir clases propias para las distintas formas de dibujo consideradas**. Este punto se indica de forma explícita en el enunciado de la evaluación precisamente para incidir en la importancia de este diseño (y evitar soluciones erróneas que trabajaran solo con el lienzo).

- Considerando el punto anterior, el método *paint* del lienzo tendría que tener la siguiente forma:

```
public void paint(Graphics g){
  super.paint(g);
  Graphics2D g2d = (Graphics2D)g;

  for( XXXX s: vectorFiguras) { //Para cada figura del vector
    // Una única llamada al método que pinta la forma 's'
  }
}
```

  donde "*s*" será una figura de la jerarquía propia definida en el punto anterior (y "*XXXXX*" su superclase). Nótese que en el cuerpo del bucle sólo debe haber una línea de código correspondiente a la llamada al método (externo a la clase del lienzo) que pinta la forma. El único atributo que podría activarse en el código anterior sería "*g2d.clip(areaClip)*" para definir el área de dibujo del lienzo (y, si se desea, la llamada a un método que dibujase un marco alrededor de la imagen y/o la figura seleccionada), si bien el resto de acciones propias de dibujo deben quedar relegadas a la forma.

  Nótese, por tanto, que la implementación realizada en la práctica 7 (donde se activaban los atributos en el método *paint*, ya fuera introduciendo el código directamente en el método, ya fuera llamando a un método tipo *setAtributos*) no sería correcta en esta práctica de evaluación.

  Cualquier solución que no tenga el *paint* anterior **SE CONSIDERARÁ ERRÓNEA** e implicará dar por incorrecto todo el módulo de gráficos. Es preciso insistir mucho en esta parte ya que no hay que pensar que el ejercicio está bien "por el hecho de que pinte": tiene que hacerlo según exige la evaluación (es decir, hay que "contestar a la pregunta del examen").

- Las propiedades que puedan estar asociadas al lienzo (color, grosor, etc.) se usarán **para crear las figuras, nunca para pintarlas**. Esto queda claro en el punto anterior, donde en el método *paint* no se activa ningún atributo relativo a las formas (esta tarea queda relegada a cada una de las formas).

- Por todo lo anterior, el consejo dado en clase varias veces: **es preferible crear una clase Lienzo2D nueva y no partir de la que se ha usado en prácticas** (que no aborda lo exigido en la evaluación).

- **La edición de figuras deberá de hacerse a través de la lista de figuras**, según se indica en los requisitos (no será correcto, por tanto, el uso de un botón de edición). Cualquier otra solución que no se base en una lista de figuras se considerará errónea.

- Para mover una figura será obligatorio poder hacerlo a través del campo de localización de la barra de dibujo, según indica el correspondiente requisito.