PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2233 Programación Avanzada (II/2017)

Tarea 6

1. Entrega

- `Tarea`
  - **Fecha/hora:** 20 de noviembre del 2017, 23:59 horas.
  - **Lugar:** GitHub – Carpeta: `Tareas/T06/`
- `README.md`
  - **Fecha/hora:** 21 de noviembre del 2017, 23:59 horas.
  - **Lugar:** GitHub – `Carpeta: Tareas/T06/`

2. Objetivos

- Aplicar conocimientos de creación de *networks*.
- Aplicar conocimientos en manejo de *bytes*.
- Aplicar conocimientos de interfaces gráficas.
- Familiarizarse con el formato de imagen PNG.
- Diseñar e implementar una arquitectura cliente-servidor.
- Comprender el uso y aplicación del archivo `.gitignore`.

3. Introducción

¡Los ayudantes de tareas necesitamos su ayuda! Nuestro jefe, Fernando, está totalmente dedicado a su nuevo pasatiempos: la fotografía. El problema es que se pasa todo el día editando las fotos a mano, y otros días más en imprimirlas y compartirlas con la gente. Debido a esto, no le ha podido dedicar el tiempo suficiente a su jefatura y los ayudantes no podemos seguir trabajando con sólo uno de nuestros dos jefes.

Es por eso que ustedes serán los salvadores del ramo. Deberán construir un programa que permita editar fotos en línea, y mostrar su avance para ir compartiéndolas. De esta forma, Fernando volverá a dedicar el tiempo necesario a sus ayudantías.

4. *Prograshop*

Esta tarea consiste en desarrollar un programa de edición en línea de fotos. Cada usuario entra con una cuenta diferente; ya dentro del programa, puede editar fotos ya cargadas o ingresar nuevas imágenes en una galería pública, donde se pueden hacer comentarios de estas mismas o simplemente contemplarlas. Cada foto puede ser editada por sólo un usuario a la vez, y los cambios realizados por el *editor* se enviarán al servidor en tiempo real. Existen distintos tipos de ediciones, las cuales varían desde agregar figuras una imagen con diversos colores hasta aplicar algún efecto a esta. Las distintas opciones de edición están detalladas en la categoría de ediciones.

4.1. Ingreso (15 %)

4.1.1. Ventana de ingreso

Al comenzar el programa, esta es la primera ventana que se debe mostrar. En esta ventana, el jugador debe ingresar su nombre. No se debe permitir ingresar a usuarios con el mismo nombre. También se debe verificar que el usuario respete las siguientes restricciones:

- El nombre no posee espacios y sólo contiene letras (a–z, mayúsculas o minúsculas) o números.
- Tiene un largo mínimo de **dos** caracteres.

En caso contrario, debe levantarse un *pop-up* avisando que existe un error y qué fue lo que lo causó.

4.1.2. *Dashboard*

Esta es la ventana principal de *Prograshop*, que se debe mostrar únicamente si es que el inicio de sesión fue correcto.

El *dashboard* debe mostrar los siguientes elementos:

- **Galería de fotos:** Se deben mostrar, al menos, 6 imágenes en la pantalla. Estas deben ser escogidas por el servidor de forma aleatoria y se deben ver en su estado actual; es decir, aplicando hasta el último cambio realizado por algún usuario.
- **Editar imagen:** Para editar alguna de las 6 imágenes disponibles en el *dashboard* se debe poder presionar la imagen o algún botón con la instrucción de editar. Luego, esto hará aparecer la *Ventana de edición*.
- **Bloqueo imagen:** Al ser un programa de edición en línea, para evitar que dos personas editen una foto al mismo tiempo, las imágenes deben bloquearse de modificaciones cuando algún usuario la esté editando. Esto debe verse reflejado en el *dashboard* mediante una advertencia en la imagen.
- **Espectador:** Si el usuario selecciona una imagen bloqueada, se debe poder entrar a la ventana de edición en modo espectador (5.4).
- **Usuarios conectados:** Se debe mostrar a todos los usuarios conectados en vivo. Cada vez que alguien nuevo entra, se debe actualizar la lista; cada vez que alguien sale, también.
- **Cerrar sesión:** Debe existir un botón que permita cerrar el *dashboard* y volver a la ventana de ingreso.

5. Edición de imágenes

Todos los archivos dentro de su estructura están compuestos por el ***header*** y el ***body***. El *header* es aquella parte que posee la ***metadata*** del archivo; es decir, la información que describe la estrucura del contenido del *body*. Luego en el *body*, se encuentra toda la información del archivo que se deberá editar, para lograr una satisfactoria edición de imágenes. Para poder hacer la edición de imágenes es necesario manejar sus *bytes* correspondientes. En esta tarea se trabajará exclusivamente con el formato **PNG** (que viene de ***P****ortable* ***N****etwork* ***G****raphics*).

5.1. Manejo de *bytes* (formato PNG)

Este tipo de imágenes, en el *header* de su estructura de *bytes*, comienza siempre con **ocho *bytes*** (conocido como la *firma* de este formato) que son estándar para el formato **PNG**. Esto es importante, ya que, durante el proceso de manejar los *bytes*, esta firma se debe mantener y no modificar. Luego de esto, los *bytes* se ordenan en bloques (o *chunks*).

Los bloques tienen una estructura establecida, que es la siguiente:

Estructura de un chunk o bloque			
Largo de información del bloque	**Tipo de bloque**	**Información**	**CRC**
4 bytes	4 bytes	Largo	4 bytes

Todos los *bytes* dentro de la información de la imagen se encuentran en *big endianness*[1]; es decir, que el dígito más significativo representa el dígito que esta más a la izquierda de un número. Si el número es 2017, el dígito más significativo es 2, mientras que el menos significativo es 7.

- **Largo:** Los primeros 4 *bytes* indican el largo del bloque que posee la sección de información del *chunk*.
- **Tipo:** Los siguientes 4 *bytes* pueden ser decodificados para obtener un *string* de 4 caracteres. Estos indican qué tipo de información posee el bloque, explicados en la siguiente sección.
- **Información:** Esta sección posee la cantidad de *bytes* indicadas en un principio. Esta información varía dependiendo del tipo de bloque en el que se esté trabajando. Los posibles tipos son los siguientes:
  - `IHDR`
    En este bloque se tiene la *metadata* de la imagen.
  - `IDAT`
    Este bloque posee los *bytes* de la imagen comprimidos. En una imagen pueden existir varios bloques `IDAT`. En este caso particular, se debe generar un solo bloque `IDAT`.
  - `IEND`
    Este bloque indica el término del archivo, y en la sección información no posee datos (el largo de la sección es 0).
- **CRC:** Los siguientes 4 *bytes* corresponden a un código CRC (*cyclic redundancy check*), que asegura la integridad de la información. El CRC se calcula con todos los *bytes* del bloque respectivo, sin contar con la sección de largo, a excepción de los bytes correspondientes al largo. Para calcularlo, se recomienda fuertemente usar el método `crc32` de la librería `zlib`. El CRC debe ir siempre presente, incluso si el bloque no tuviese información.

[1] https://es.wikipedia.org/wiki/Endianness

5.2. Convertir imagen a formato PNG (10 %)

A continuación se explicará brevemente cómo pasar una matriz de pixeles RGB, a una imagen en PNG. La matriz de pixeles RGB representa el color del pixel dentro de la matriz, a través de tres valores en una tupla (***r**ed, **g**reen, **b**lue*) que simbolizan la composición del color, en términos de la intensidad de los colores primarios aditivos de la luz. Ante cualquier duda respecto a este proceso, y en particular, cómo construir los bloques, se recomienda revisar la especificación de PNG, ubicada en el siguiente enlace: `https://www.w3.org/TR/PNG/#11Chunks`. En general, para todos los *chunks*, primero se deben formar los *bytes* de la sección *información*.

5.2.1. Creando el IHDR

En el caso del bloque `IHDR`, a continuación se explica el significado de cada elemento que conforma la sección de información:

IHDR	
Cantidad de bytes	**Descripción**
4	Ancho en pixeles
4	Alto en pixeles
1	Profundidad de bits
1	Tipo de colores
1	Tipo de compresión
1	Tipo de filtro
1	Tipo de entrelazado

- **Ancho**: El ancho de la imagen. Debe ser codificado en 4 bytes.
- **Alto**: Igual que el punto anterior, aplicado al alto de la imagen.
- **Profundidad en *bits***: Corresponde a la cantidad de *bits* por canal. Se recomienda usar el valor `16`. Debe ser codificado en 1 *byte*.
- **Tipo de color**: Como cada pixel va a ser representado por un triple RGB, se debe usar el valor `2`. Debe ser codificado en 1 *byte*.
- **Tipo de compresión**: Indica el método usado para comprimir los datos de la imagen. Actualmente sólo se encuentra definido el método con valor `0`; por lo tanto, ese es el que se debe usar. Debe ser codificado en 1 *byte*.
- **Tipo de filtro**: Indica el método de preprocesamiento que se aplicó a la imagen antes de comprimir los datos. Esto se usa usualmente para mejorar el *performance* de la compresión. Por simplicidad, se recomienda omitir el preprocesamiento. En ese caso se debe usar el valor `0`. Debe ser codificado en 1 *byte*.
- **Tipo de entrelazado**: Indica el orden de transmisión de los datos de la imagen. Por simplicidad, se recomienda no usar entrelazado. En ese caso, se debe usar el valor 0. Debe ser codificado en 1 *byte*.

Sólo falta insertar, al principio, el largo de la información y tipo de bloque. Recuerda calcular el CRC y agregarlo al final del bloque. Tenemos nuestro bloque IHDR listo.

Pista: pueden usar los comandos `to_bytes` y `from_bytes` para poder transformar los distintos elementos.

Ejemplo de crear un IHDR: Digamos que se quiere formar una imagen PNG de 3x2 pixeles, sin filtro. Esta imagen tendría, entonces, un ancho de 3 y alto de 2. Luego, la sección de información estará compuesta por la concatenación de los *bytes* al pasar el número 3 a *bytes*, luego el 2, después el 16 por la profundidad, después el 2 por tipo de color, luego un 0 por compresión, luego otro 0 porque no tiene filtro, y finalmente otro 0 porque no tiene entrelazado. La sección de tipo de bloque es obtenida al pasar a *bytes* la palabra IHDR. Con la concatenación de las secciones se calcula el CRC con la librería explicada anteriormente y se le concatena al final del arreglo de *bytes*. Finalmente se genera la sección de largo dada por la longitud de la concatenacion de las secciones de tipo, información y se pasa a *bytes*. Al concatenar las cuatro secciones se tiene listo el IHDR.

5.2.2. Creando el IDAT

La información descomprimida del bloque IDAT corresponde a una representación de la matriz de pixeles de la imagen. Esencialmente, cada *byte* corresponde a un canal del pixel que se está representando. Para ejemplificar:

1. El primer *byte* representa el valor del canal rojo del primer pixel; el segundo *byte*, el canal verde del primer pixel; el tercer *byte*, el canal azul del primer pixel; el cuarto *byte* el canal rojo del segundo pixel, y así sucesivamente.

2. Luego, al principio de cada fila de pixeles es necesario agregar un *byte* adicional que corresponde al filtro aplicado a dicha fila. Como no se aplicó ningún filtro, basta con asignarle el valor `0`. Entonces, si cada pixel estará representado por un triple RGB, y además se debe agregar un *byte* al principio de cada fila para el filtro, el largo de este bloque sin comprimir será $alto \times (ancho \times 3 + 1)$.

3. Finalmente se debe comprimir el bloque de información. Para hacerlo, se recomienda usar el método `compress` de la librería `zlib`. Sólo falta insertar al principio el largo de la información y tipo de bloque. Recuerda calcular el CRC y agregarlo al final del bloque. Tenemos nuestro bloque IDAT listo.

   Ejemplo de crear un IDAT: Siendo la misma imagen del ejemplo de IHDR, su formato descomprimido sería de dos filas, cada una con 7 *bytes*, representados por el RGB de cada uno de los pixeles más el 0 porque no tiene filtro. Es por esto que la sección de información da un largo descomprimido de *14 bytes*.

5.3. Ventana de edición (3 %)

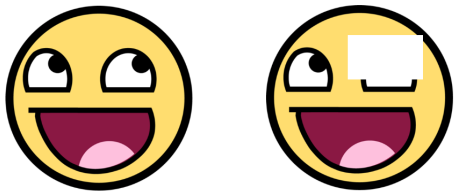
La ventana de edición es donde ocurrirá la magia. Una vez seleccionada la foto a editar, en esta ventana se podrán realizar todos los cambios posibles y/o comentar la foto. En esta ventana, deben estar los siguientes elementos:

- **Botón de descarga:** Genera la imagen en `.png` y la descarga en el computador del usuario.
- **Actualización de cambios:** Luego de alguna modificación, se deben enviar los cambios al servidor, poniéndola a disposición a aquellos que estén en modo espectador y también en el *dashboard* del resto de los usuarios.
- **Comentarios:** Se deben poder ver los comentarios de la imagen respectiva, junto con la opción que permita agregar nuevos comentarios. Dentro de un comentario, se debe poder ver quién lo mandó, la fecha y el mensaje.
- **Salir:** Cierra la ventana de edición y vuelve al *dashboard*.

Además de estos elementos, dentro de la ventana de edición, se le puede hacer las siguientes ediciones a las imágenes. Todas las ediciones deben hacerse trabajando directamnte sobre los pixeles de la imagen.

5.3.1. Recortar (8 %)

Mediante esta opción, se puede seleccionar una parte de la imagen y dejar en blanco. La opción de recortar es mediante un rectángulo cuyo tamaño y posición es definida por el usuario mediante el uso del *mouse*. De forma visual, si se aplica un recorte a una imagen, esta mostrará el siguiente cambio:

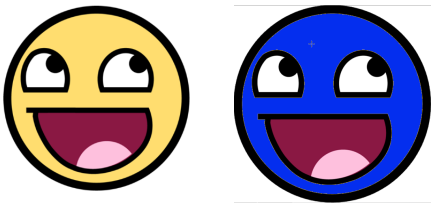


Para poder aplicar este efecto a una imagen PNG, debes encontrar los pixeles que fueron borrados mediante el rectángulo de recorte y cambiar sus colores a blanco `rgb(255, 255, 255)`.

5.3.2. Balde (10 %)

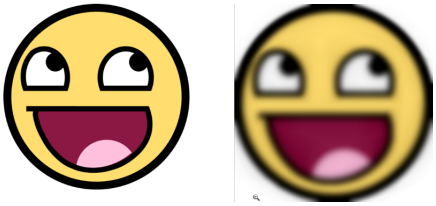
En esta opción, el usuario debe poder escoger un color, un pixel de la imagen y el programa debe ser capaz de propagar el color escogido a todos los pixeles adyacentes directos (arriba, abajo, derecha, izquierda) del mismo color al pixel elegido. En otras palabras, dado un pixel de la imagen, deben revisar todos los que están a su alrededor y si son del mismo color que el pixel seleccionado, deben cambiar su color y seguir con este procedimiento hasta que la propagación no pueda seguir en ninguna dirección.

Una propagación no puede seguir en una dirección cuando el pixel que está en esa dirección ya no es del mismo color o no existe; es decir, cuando llegamos a una orilla o esquina. A continuación, se muestra los efectos de aplica baldes sobre una parte amarilla de la imagen.



5.3.3. *Blurry* (9 %)

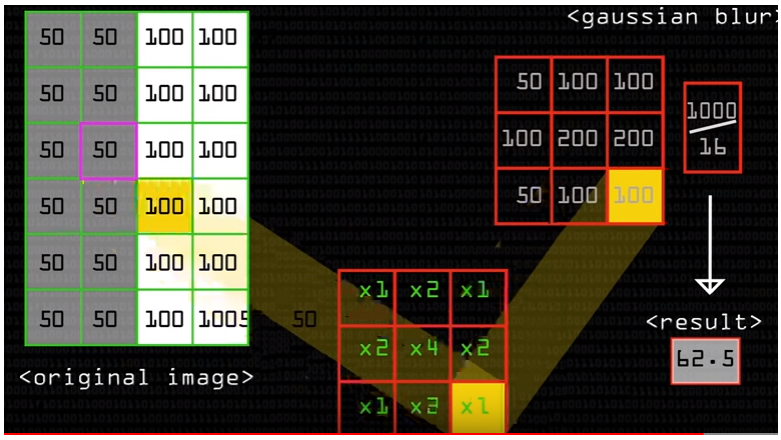
En esta opción, el usuario puede agregar un filtro de *blur* a la imagen, creando un efecto de desenfoque.



Para poder aplicar el efecto a la imagen PNG, en cada valor RGB de cada pixel se debe calcular mediante su ponderación con los pixeles vecinos. La ponderación es la siguiente:

Dado un pixel `(r, g, b)`, los pixeles directos a él (arriba, derecha, izquierda, abajo) se ponderaran por 2, los pixeles diagonales, se ponderaran por 1 y el pixel original se pondera por 4, y todo esto se divide por 16 y se redondea, resultando en el nuevo valor. Esto se tiene que aplicar para los tres valores de la tupla `(r,g,b)`.

Ejemplo:



1. Sea la imagen original la matriz del color rojo perteneciente a la tupla RGB, y se quiere recalular el valor del pixel marcado en magenta de valor 50 en la matriz original.
2. Cada pixel vecino con si mismo pasan por una matriz de ponderadores, siendo multiplicados, como se ve en la imagen.
3. Luego, con los valores ponderados se saca un promedio ponderado, es decir, la suma de todos los valores y se divide en 16 —que es la suma de todos los ponderadores—, formando el nuevo valor R del pixel. Donde el resultado es 62,5.
4. Finalmente, se redondea el número para que sea un entero.
5. Se repite el proceso para hacer el valor de r, de g y de b.

5.4. Espectador (5 %)

Cada imagen sólo puede ser editada por un usuario a la vez. En caso de entrar en modo espectador, se debe mostrar la imagen con el último cambio que fue generado en dicha imagen. Cuando un usuario en edición hace una modificación a la imagen, este cambio se debe hacer visible para todos los usuarios en modo espectador. Por lo tanto, el modo espectador debe ser en tiempo real.

Dentro de este modo, se encuentran disponibles todas las opciones de la ventana de edición, excepto las herramientas de edición.

6. Comentarios (15 %)

Para diferenciarse de su competencia, usted ha decidido incorporarle a un sistemas de comentarios a la interfaz.

Cada comentario debe contener, además, el nombre de usuario que comentó y la fecha y hora en que se creó. Al enviar un comentario, el servidor debe procesar este mensaje para que todos los clientes que estén conectados o se conecten en un futuro a esa imagen lo puedan ver.

Al ingresar a la sección de comentarios, se muestran todos los comentarios que se hayan escrito con la imagen respectiva. En caso de haber una gran cantidad de comentarios, su interfaz debe ser capaz de realizar un *scroll* para poder verlos todos.
Emojis: Su comentario debe soportar el uso de 10 *emojis* especificados en la siguiente tabla:

Imagen	Comando	Notas
	:poop:	Todo con minúscula
	O:)	una “O” en mayúscula
	:D	Una “D” en mayúscula
	;)	
	8)	
	U.U	Dos “U” en mayúsculas
	:(	
	3:)	
	o.o	Dos “o” en minúsculas
	:v	Una “v” en minúscula

El sistema de *emojis* en su comentario reconoce automáticamente los comandos de la tabla y al momento de enviar el mensaje, estos los cambia por el *emoji* adecuado. La única forma de evitar que el comentario cambie algún comando por la imagen es insertar el comando dentro de los símbolos <>, los cuales serán eliminados después de mandar el mensaje.

7. Cliente y servidor (20 %)

Su implementación de *Prograshop* **debe** basarse en la arquitectura *cliente–servidor*, en la que todas las interacciones que se quieran realizar entre usuarios deberán contar con el servidor como mediador de la comunicación. Para lograr esto, tendrán implementar un protocolo eficiente basado en el modelo **TCP/IP**.

Su sistema deberá ser capaz de funcionar entre computadores que estén conectados a una misma red local. Para esto, necesitarán utilizar la librería `socket` y sus métodos. A continuación se muestra como es que deben implementar esta estructura:

7.1. Servidor

El servidor se debe encargar de la lógica del sistema, sus interacciones, el almacenamiento de datos de todos los archivos *subidos*, junto con la información de los usuarios y cualquier otra funcionalidad que consideren pertinente. Por lo mismo, no se debe implementar una interfaz gráfica para este componente, ya que funcionará como parte del *back–end*.
Debe existir sólo una instanciación de este módulo en un único computador para poder ejecutar correctamente el programa. Además, en el módulo en que escriban el código correspondiente al servidor, deben colocar en las primeras dos lineas las variables `HOST` y `PORT` con sus valores respectivos, para que así al momento de corregir, cambiar sus valores sea más fácil. Cabe recalcar que ambas las deben utilizar en el método `bind` del *socket* del servidor para que se realice la conexión.
Ahora, para poder conectarse por red local, deben:

1. Estar conectados a una red.

2. Ajustar la variable `HOST` de su servidor como la IP local de su computador. Para saber cómo se obtiene la de su ordenador, basta con buscar en **Google**. No es un proceso difícil.

7.2. Cliente

Este es el programa con el que interactuarán los usuarios, por lo que deben incluir la interfaz gráfica de *Prograshop*. El cliente tiene que ser capaz de conectarse al servidor para poder funcionar, el cual puede estar ejecutándose en el mismo computador, o en un remoto. Si el cliente fuera ejecutado en dos computadores distintos en la misma red, ambos deberían poder interactuar de la misma forma con el servidor.

Al igual que en el caso del servidor, deberán escribir en las dos primeras líneas `HOST` y `PORT` junto con sus respectivos valores. Ambas también deben ser usadas en el método `connect` del *socket* del cliente.

7.3. Estructura de archivos

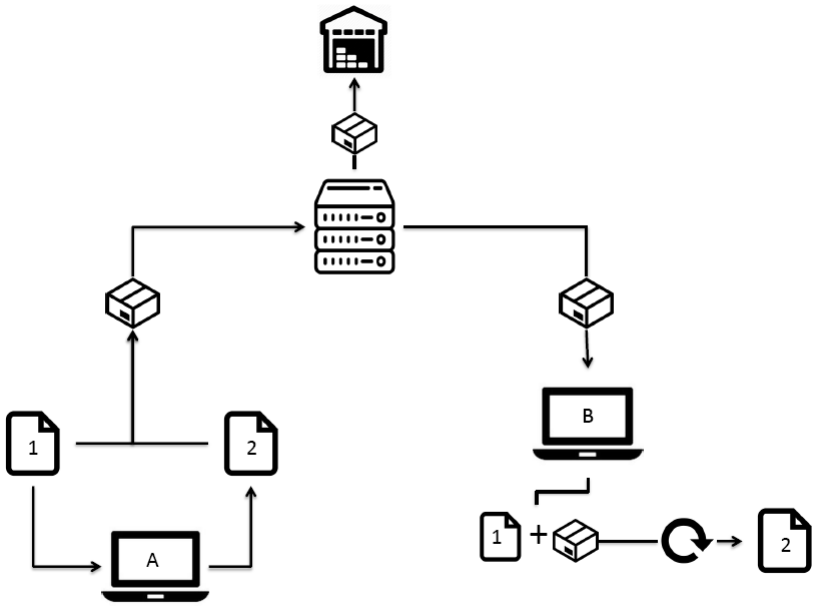
La tarea se debe entregar con la siguiente estructura de carpetas y archivos:

```
T06
|-- client
|   `-- main.py
|   `-- (otros módulos del cliente)
`-- server
    |-- main.py
    |-- (otros módulos del servidor)
    `-- image
        |-- imagen_1.png
        |-- imagen_2.png
        |-- imagen_3.png
        |-- imagen_4.png
        `-- (más_imagenes)
```

Entonces, los programas del cliente y servidor deben estar separados, cada uno en su propia carpeta. Además, como se indica, las imágenes se deben encontrar dentro de la carpeta `image`, que, a su vez, está dentro de la carpeta `server`.

7.4. Envío de archivos

La comunicación entre el cliente y el servidor en tu tarea debe ser a través de paquetes de datos, los cuales deben contener solo los cambios hechos, ahorrando el traspaso de toda la información cada vez que se realice la comunicación.
Como se puede ver en la **figura a continuación**, si se edita, por ejemplo, una imagen, el cliente que realiza tal acción solo debe enviar al servidor la información necesaria para hacer el cambio. Y si, por ejemplo, otro cliente solicita la visión de tal imagen, el servidor sólo debe mandar los componentes necesarios para ver la imagen nueva, sin mandar todo el *data* de esta. En el caso en que este último no posea la ilustración solicitada en el ordenador, sólo ahí debe enviarse el archivo completo.



8. Interfaz gráfica (5 %)

Todas las interacciones que se esperen del cliente deben realizarse a través de interfaces gráficas. La interfaz debe ser amigable y enfocada en la usabilidad; es decir, comportarse de acuerdo a lo que espera el usuario de forma natural, sin eventos inesperados.

9. Restricciones y alcances

- Tu programa debe ser desarrollado en Python v3.6.
- Esta tarea es estrictamente individual, y está regida por el Código de Honor de la Escuela: Clickear para Leer.
- Tu código debe seguir la guía de estilos descrita en el `PEP8`.
- Si no se encuentra especificado en el enunciado, asume que el uso de cualquier librería Python está prohibida. Pregunta en el foro si es que es posible utilizar alguna librería en particular.
- Si no subes el entregable, tendrás **5 décimas menos** en la nota final de la tarea.
- El ayudante puede castigar el puntaje[2] de tu tarea, si le parece adecuado. Se recomienda ordenar el código y ser lo más claro y eficiente posible en la creación algoritmos.
- Debes adjuntar un archivo `README.md` donde comentes sus alcances y el funcionamiento del sistema (*i.e.* manual de usuario) de forma *concisa* y *clara*. **Tendrás hasta 24 horas después de la fecha de entrega** de la tarea para subir el `README.md` a tu repositorio.
- Crea un módulo para cada conjunto de clases. Divídelas por las relaciones y los tipos que poseen en común. **Se descontará hasta un punto si se entrega la tarea en un solo módulo[3].**

[2]Hasta −5 décimas.
[3]No agarres tu código de un solo módulo para dividirlo en dos; separa su código de forma lógica

- Cualquier aspecto no especificado queda a tu criterio, siempre que no pase por sobre otro.

Tareas que no cumplan con las restricciones señaladas en este enunciado tendrán la calificación mínima (1.0).