Tarea Programada #2

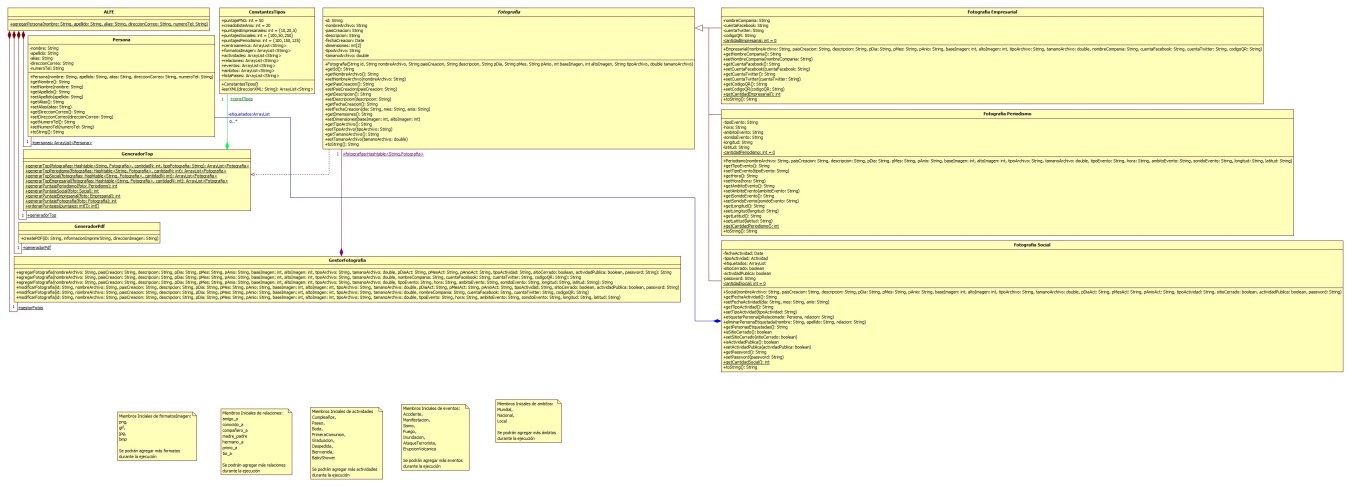
IC-2101 Programación Orientada a Objetos, I Semestre de 2013

Profesor: Luis Javier Chavarría Sánchez

Estudiantes: Jasson González Torrez 201116046, Daniel Sánchez Castillo 201123523

a. Manual de Usuario

b. Diagrama de clase



Para ver correctamente el diagrama, se adjunta este en con el nombre diagrama.jpg en la misma carpeta donde se encuentra este documento.

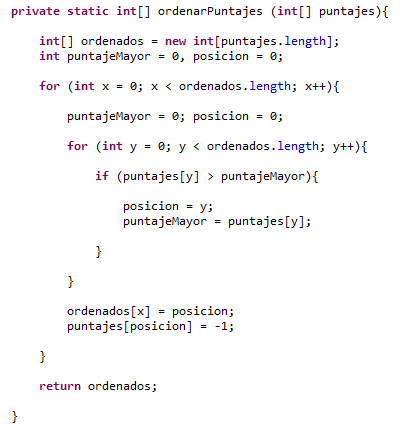
c. Pruebas de funcionalidad

d. Lecciones aprendidas

* Al ser el segundo proyecto que se realizó con el mismo equipo, la división de trabajo fue un poco más sencilla. El uso de diagramas para plasmar las ideas de ambos acerca del contexto dado fue de gran utilidad.
* El conocimiento adquirido hasta este punto del curso y la práctica a través del primer proyecto y los laboratorios ayudaron a mejorar elementos importantes como a modularidad y la claridad a la hora de crear código. A partir de esto se puede dividir más fácilmente el trabajo y montar con más facilidad la interfaz.
* Gracias a que el equipo fue el mismo que para el proyecto anterior, se dividió el trabajo de una manera parecida. Esto ayudó a que cada uno trabajara en una parte de la cual ya tenía conocimientos recientemente adquiridos, lo cual llevó a un ahorro considerable de tiempo.

e. Principios

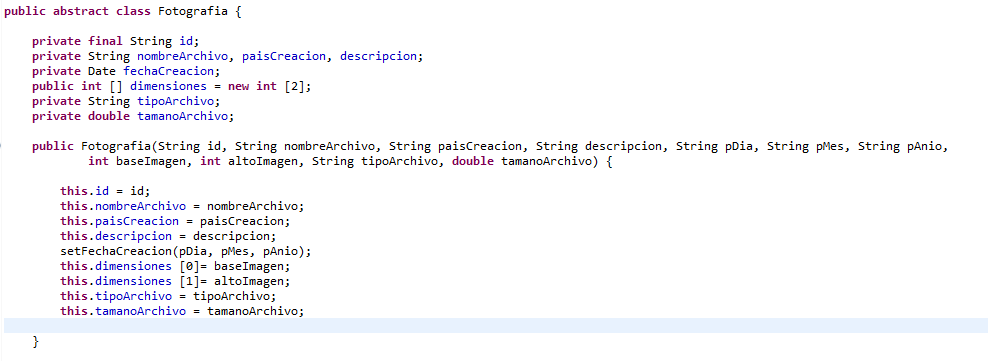
Carácter Personal y Estética:



Una buena forma de conocer un poco del programador de un código es revisar métodos o funciones sencillas, y ver como ataca estos problemas. Hay muchas formas previamente definidas para ordenar un grupo de números, pero cada persona tiene una forma con la cual se siente más cómodo(a) y también existen casos donde simplemente es más sencillo hacer una pequeña función ordenadora antes de buscar métodos complicados que otros han diseñado.

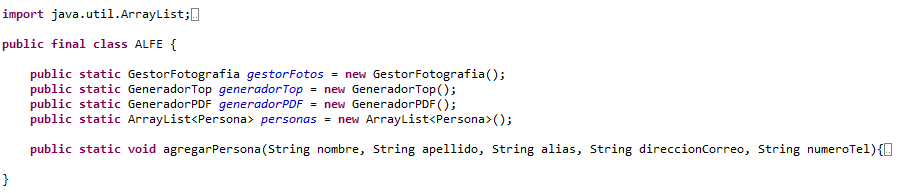
Además, se puede notar la forma ordenada y estética de trabajar, dejando espacios y tabulaciones para facilitar la comprensión hasta del código más simple.

Claridad:



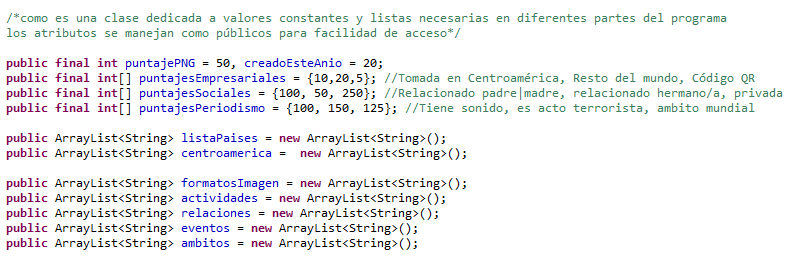
Se utilizan atributos con nombres claros, que no generen dudas sobre la información que va a ser guardada en estos y un constructor ordenado y con parámetros que tratan de evitar la generación de dudas al ser usado.

Distribución:



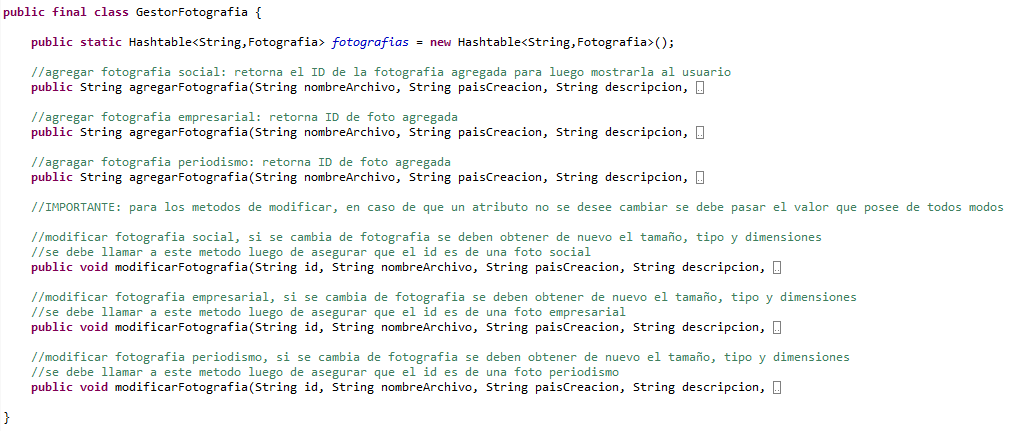
Para facilitar el acceso a las diferentes funcionalidades del programa se creó una clase que se compone de objetos de las clases que realizan los trabajos más importantes y de esta forma el programador de la interfaz solo debe recurrir a una clase para realizar un programa que muestre y reciba todo lo pedido por el cliente.

Implícito:



Se utilizan nombres de variables que eviten confusiones y que revelen automáticamente para lo que serán utilizados pero también en el caso de tener más de un valor (como en este caso los arreglos de puntajes Empresariales, Sociales y de Periodismo) se agrega un comentario que indique que es cada valor, para luego poder modificarlos fácilmente sin tener que buscar los métodos que los necesitan.

Código Auto Documentado y Comentarios



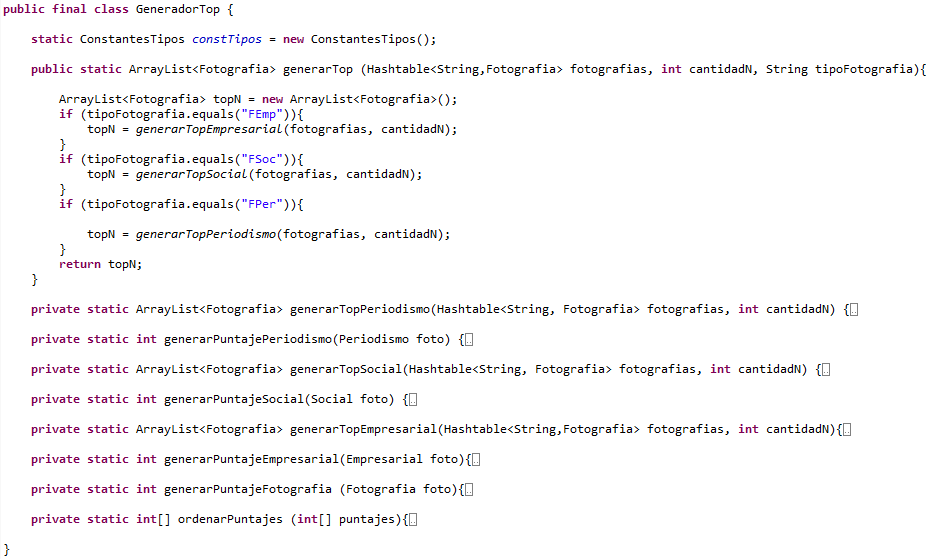
Entre las decisiones tomadas por el equipo se encuentra la definición de los métodos agregarFotografia y modificarFotografia tres veces cada uno debido a la necesidad de trabajar tres diferentes tipos de objetos. Para aumentar la claridad se agregó la información necesaria en un comentario fuera de cada método y además se anotan necesidades a la hora de usar los métodos de modificación para que más adelante sean utilizados correctamente.

El TIEMPO y EL DINERO de otros

Debido al momento del semestre en que este proyecto fue desarrollado, fue complicado para ambos miembros del equipo realizar reuniones para realizar diagramas y comentar ideas. Para evitar retrasos debido a estas dificultades se decidió realizar diagramas individuales para luego en una corta reunión poder unificar las ideas y de estar forma tener una base concreta para empezar la creación del programa. Aun cuando el dinero no es un impedimento tan fuerte como en la realidad, los estudiantes del TEC si tienen complicaciones de tiempo, debido a mal manejo y evaluaciones en fechas cercanas. Por esta razón es importante llegar a soluciones rápidas y así no causar retrasos que luego podrían terminar en malas calificaciones.

e. Olores del software

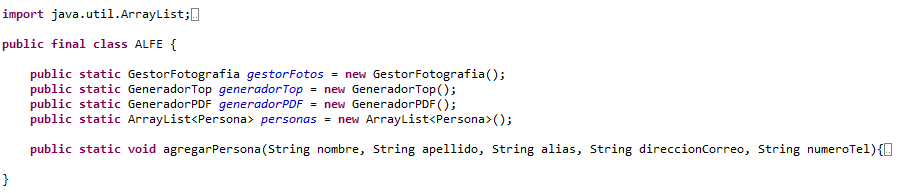
Rigidez y Viscosidad



Para disminuir la rigidez, no solo se dividieron las funcionalidades necesarias en diferentes clases, sino que también dentro de cada clase se trató de modularizar al máximo cada método. Por ejemplo, si se deseara agregar un nuevo tipo de fotografía solo sería necesario agregar el código para obtener su top específico y luego agregar un nuevo if para este nuevo tipo.

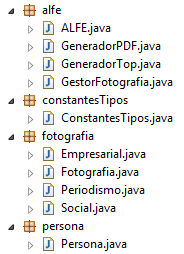
Además, la viscosidad se disminuye ya que si se necesita cambiar la forma de obtener un top o un puntaje solo se debe cambiar el método y no hay que poner a interactuar otros objetos o clases para hacer estos cambios.

Fragilidad



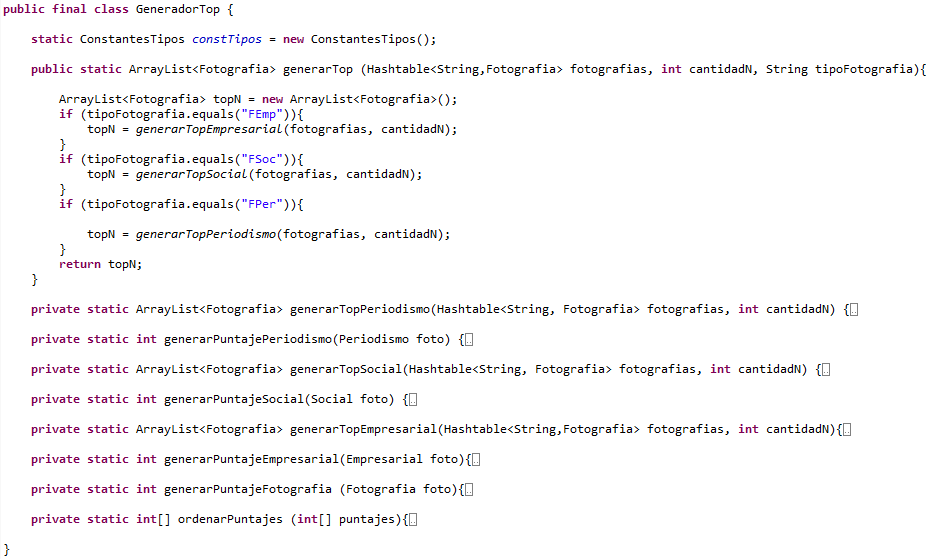
Otra de las razones de la clase ALFE es evitar conexiones entre clases que aumenten la fragilidad del software. Se mantiene centrado el acceso a las funcionalidades y de esta forma si el gestor de fotografías tuviera un fallo, los generadores de PDF o de top no necesariamente notarían este error. Y de la misma forma la interfaz podría continuar funcionando en su mayoría mientras se realizan los arreglos necesarios.

Inmovilidad



Se utiliza una imagen de los paquetes y clases para mostrar que el proyecto permite fácilmente sacar estos módulos y a su vez muestra cuales clases podrían necesitar de otras (como es el caso de Empresarial, Periodismo y Social que necesitan de Fotografia) y de esta forma ayudar a la extracción de código necesario y a la preparación para los errores que podrían generarse.

Complejidad y Repetición innecesaria



En lugar de hacer un método que sacara todo el puntaje y a la vez ordenara las fotografías para cada tipo (lo cual aumentaría la complejidad a la hora de revisar y encontrar errores o de cambiar alguna parte por otra) se dividió al máximo cada método, haciendo uno que obtuviera los puntajes con un generarPuntaje, luego ordenara estos puntajes con ordenarPuntajes y finalmente retornara los n requeridos.

Además, se disminuye la repetición innecesaria con métodos como generarPuntajeFotografia que toma los valores que cualquier fotografía ingresada tiene y obtiene los puntajes requeridos en lugar de repetir estos atributos en cada generarPuntaje específico.

Opacidad

Para evitar la opacidad en todo el código se trató de utilizar nombres de atributos y métodos que ayudaran a entender fácilmente para que serían utilizados o que acción realizarían dentro del programa. Igualmente se trata de tabular la información para reconocer que líneas se encuentran dentro de cada if o cada ciclo utilizado. Esto se complementa con líneas en blanco (ENTER) para dividir textos y evitar acumulaciones masivas de líneas de código y de esta forma ayudar a la lectura posterior del mismo.