**Ingeniería de Software III - Primer Parcial**

|  |  |
| --- | --- |
| Nombre: | Daniel Fernando Sánchez Ortiz |

Se trabajará con el proyecto “**partitura2.zip**”. Este proyecto es una adaptación del proyecto Cupi2 de Uniandes, sitio web: <https://cupi2.virtual.uniandes.edu.co/nivel-10/ejemplos/partitura>

1. Se identifica el mal olor “Large Clase” en la clase “Partitura”.

Referencia: <https://refactoring.guru/smells/large-class>. Se le ha pedido que usted:

1. (0.7) Elabore un diagrama de clases que muestre cómo lo solucionaría.

|  |
| --- |
| Incluir aquí el diagrama (como imagen). |

1. (0.6) Elabore un plan de trabajo con tareas concretas que tendría de realizar para cumplir con este cambio (solo tiene que hacer el plan, no hay que hacer el cambio en el código). Para el plan use la siguiente tabla, adicionando las filas que considere necesarias:

|  |  |
| --- | --- |
| **Descripción de la tarea** | **Tiempo estimado (en minutos)** |
| Realizar ingeniería inversa al programa para identificar relaciones | 45 min |
| Identificar patrones de tareas comunes para separar por clases(lo que hice en la gráfica) | 60 min |
| Crear la clase archivo (la de menor dependencia) | 5 min |
| Extraer desde la clase partitura los métodos para trabajar con archivos(abrir y salvar) y eliminarlos de la clase partitura | 5 min |
| Modificar métodos para no recibir el parámetro nombreArchivo sino tomarlo de la misma clase | 15 min |
| Determinar las variables relacionadas a la interfaz en la clase partitura | 10 min |
| Extraer estas para la creación de una nueva clase llamada GraficoPartitura | 10 min |
| Extraer los métodos relacionados a la interfaz gráfica de la clase partitura | 30 min |
| Verificar que la clase partitura solo tenga métodos relacionadas al control de notas luego de las otras extracciones | 15 min |
| Correr pruebas para comprobar que el comportamiento del programa no ha cambiado | 10min |

1. (1.0) Al realizar el análisis de cobertura se encuentra que el método “verificarNota” de la clase “Partitura” no está cubierto por las pruebas.

Texto

Descripción generada automáticamente

Diseñe las pruebas que se requieran para lograr una cobertura de líneas y de decisiones del 100 % en este método (las mínimas requeridas). Use la siguiente tabla, adicionando las líneas que sean necesarias:

|  |  |  |
| --- | --- | --- |
| **Descripción de la prueba** | **Valores de entrada** | **Resultados esperados** |
| posicionActual menor que 0 | PosiciónActual = -20  Nota.size( ) = 10 | false |
| Posición actual mayor que cero con posiciónActual menor que a la de la lista, ademas aux=nota | posicionActual = 12  notas.size( ) = 15  aux = nota | true |
| Posición actual mayor que cero con posiciónActual menor que a la longitud de la lista y aux ¡=nota | posicionActual = 12  notas.size( ) = 15  aux ¡= nota | false |

1. Al realizar un análisis estático se encuentra que en la línea 145, en el método abrir de la clase “Partitura” muestra una violación bloqueante: “Avoid File Stream”.

Referencia: <https://pmd.github.io/latest/pmd_rules_java_performance.html#avoidfilestream>

1. (0.5) Explique con sus propias palabras por qué se presenta esta violación y qué implicaciones tiene.

|  |
| --- |
| R/ Las clases FileInputStream y FileOutputStream contienen un método de finalización que provocará pausas en la recolección de elementos no utilizados. |

1. (0.6) Modifique el programa para quitar esta violación.

|  |
| --- |
| Inserte aquí cómo quedó el método (o métodos, depende del cambio que realice). Puede ser como texto o como imagen (pantallazo).  try{  FileReader file = new FileReader(nombreArchivo)  BufferedReader br = new BufferedReader(file);  } catch(IOException e){  e.printStackTrace();  } finally{  try {  if (objReader! = null)  objReader.close();  } catch (IOException ex) {  ex.printStackTrace();  }  } |

1. (0.5) Identifique otra violación que se encuentre en esta clase, que usted considere una de las más importantes. Explique su respuesta.

|  |
| --- |
| R/    Método largo, este método aparte de abrir el archivo es el encargado de crear las notas a partir de la información del archivo, lo cual bajo mi perspectiva se debería dividir en dos tareas diferentes, es importante en mi opinión realizar esta refactorización ya que el método puede ser difícil de entender a primera vista, además ayuda a que el método abrir pueda ser usado en otros asuntos referentes a la lectura de archivos y no como esta en la imagen que es dependiente de otros métodos de la clase lo cual lo hace poco reutilizable. |

1. (0.5) Considerando las recomendaciones sobre comentarios de código, cambie los comentarios y/o el código del método “cargarImagen” para que sean más claros:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

|  |
| --- |
| Inserte aquí cómo quedó el método. Puede ser como texto o como imagen (pantallazo).  /\*\*  \* carga la imagen con el nombre especificado en el constructor  \* @param imagen Nombre de la imagen a ser cargada  \*  \* Método con propósito de definición de elementos de la interfaz gráfica.  \*  \* Asigna valor a la variable “figuraClaveSol” que será visualizada en la interfaz gráfica  \* Del programa.  \*  \* En caso de que la imagen no exista, la figuraClaveSol será asignada con un fondo de  \* Color plano.  \*  \*/ |

1. (0.6) Identifique una posible refactorización que pueda hacer en el código, en alguna clase del paquete “partitura.**interfaz**”. Debe ser una refactorización diferente a “rename” y “replace magic literal”. Recuerde que no todos los cambios que se hacen en un código son refactorizaciones (revise en el catálogo).

|  |
| --- |
| Inserte aquí el pantallazo donde se observa la parte del código donde se propone hacer la refactorización. |

|  |
| --- |
| Escriba (muy brevemente) en qué consistiría la refactorización. No hay que hacerla, solo explicar.  La refactorización consistiría en eliminar código muerto, como podemos observar en la imagen, la clase partitura tiene dos métodos que no son utilizados en el programa y que tampoco representan una funcionalidad para el programa, la refactorización en este caso sería simplemente eliminar estos segmentos de códigos que consumen memoria pero que no tienen ninguna función dentro del contexto del programa. |