Desarrollo Guía 2

Jullians Mauricio Amado Gutierrez

Tomas Alejandro Santiago Reyes

Juan Esteban Cabal Bautista

Universidad Manuela Beltrán

Taller de programación

Olga Lucia Roa Bohórquez

17 de febrero de 2025

Tabla de contenido

Desarrollo Sesión 1

1. Entrega de Certificado:

Por cada integrante, sobre un curso que se pidió en el transcurso de la semana, para un punto en el planteamiento de esta Guía de laboratorio:

Tomas Santiago:

Jullians Amado:

Juan Cabral:

2. ## Revisar el ejemplo:

   Manejo de archivos aleatorios del libro “Cómo programar en java” de (Deitel, 2018):

   1. Descargue y abra el proyecto de ejemplo.
   2. Observe las librerías que enlaza el proyecto. Verifique las interfaces que implementa la librería y como son utilizadas estas interfaces en el proyecto.
   3. Responda el cuestionario de la guía.

Desarrollo Sesión 2

Se le solicita desarrollar un proyecto que cree el CRUD de una tabla Usuario manejando persistencia en un archivo aleatorio.

1. Cree un nuevo proyecto llamado ProyectoTallerProg
2. Cree la interfaz IGestionDatos, la cual declara los métodos:

    bool AgregarRegistro(Registro reg)
    bool BorrarRegistro(Registro reg)
    bool EditarRegistro(Registro reg)
    Registro buscar(string criterio)

3. Agregue la clase RegistroUsuario la cual implementa IGestionDatos. Implemente las funcionalidades para que esta clase almacene la información de un usuario en un archivo de acceso
aleatorio y desarrolle los métodos necesarios para cumplir con el CRUD de la tabla.
4. En el laboratorio presente el código funcionando.

Desarrollo Trabajo Autónomo

Texto que se usara para hacer el video:

Los archivos en Java se manejan a través de flujos de entrada y salida, utilizando clases del paquete `java.io`, como `FileInputStream`, `FileOutputStream`, `RandomAccessFile`, `FileReader` y `FileWriter`. Los datos en una computadora se reducen a combinaciones de ceros y unos, y Java considera cada archivo como un flujo secuencial de bytes. Para mejorar el rendimiento, se utilizan *buffers*, como `BufferedInputStream` y `BufferedOutputStream`, que almacenan temporalmente datos en memoria antes de su lectura o escritura definitiva, reduciendo la cantidad de operaciones físicas con el dispositivo de almacenamiento.

Las canalizaciones permiten la comunicación entre subprocesos mediante `PipedInputStream` y `PipedOutputStream`. Además, la interfaz `DataInput` permite leer valores primitivos de un flujo, mientras que `DataOutput` los escribe. También existen flujos especializados como `PrintStream` para salida de texto y `ObjectInput` para la lectura de objetos serializados. Estos mecanismos optimizan la manipulación de archivos y la comunicación en aplicaciones Java, garantizando eficiencia y flexibilidad en la gestión de datos.