1. ¿Qué hace el constructor de InputStreamReader?

a. Convierte un flujo de caracteres en bytes.
b. Envuelve un InputStream para convertir bytes en caracteres. **InputStreamWriter de byte a caracteres, es para binarios!!**
c. Bufferiza la salida de datos.
d. Permite la lectura secuencial de objetos.

---

2. ¿Cuál es la principal ventaja de utilizar un BufferedReader sobre un InputStreamReader solo?

a. Mejora la legibilidad del código.
b. Permite leer líneas completas de texto de forma eficiente.***
c. Transforma automáticamente los datos en mayúsculas.
d. Aumenta el tamaño de la memoria asignada.

---

3. ¿La clase OutputStreamWriter se utiliza para...?

a. Convertir caracteres a bytes para escribir en un flujo. **De caracteres a bytes, es para binarios!!**
b. Leer datos de un flujo de bytes sin conversión.
c. Bufferizar datos de entrada.
d. Imprimir datos en la consola.

---

4. ¿Qué método de PrintWriter permite imprimir datos con un salto de línea?

a. Print()
b. Println()***
c. WriteLine()
d. AppendLine()

---

5. ¿Qué sucede al llamar al método readLine() de un BufferedReader al finalizar el flujo?

a. Retorna una cadena vacía.
b. Lanza una excepción.
c. Retorna null. **Termina el flujo y como no hay mas lineas pues devuelve null**
d. Reinicia la lectura del archivo.

---

6. ¿Cuál es una consecuencia de no cerrar los flujos de entrada/salida en Java?

a. Se produce una pérdida de precisión en los datos.
b. Se liberan automáticamente sin consecuencias.

c. Puede ocasionar fugas de recursos y agotar los descriptores de archivo.***

d. Se mejora el rendimiento del sistema.

---

7. ¿Cuál de las siguientes opciones no es una función de BufferedReader?

a. Leer una línea completa con readLine().

b. Leer un único carácter con read().***

c. Escribir datos en un flujo. **El mismo metodo lo dice reader por tanto no puede escribir**

d. Mejorar el rendimiento de lectura mediante buffer.

---

8. ¿Qué método se utiliza para establecer la operación de cierre de una ventana de Swing?

a. setCloseOperation()

b. setDefaultCloseOperation()***

c. setExitOnClose()

d. closeOnExit()

---

9. ¿Qué componentes de Swing se utilizan para agrupar otros componentes y organizar su disposición?

a. JFrame

b. JPanel***

c. JButton

d. JLabel

---

10. ¿Cuál de las siguientes afirmaciones sobre un JMenuBar es correcta?

a. Solo se puede usar con JDialog.

b. Es el contenedor principal para menús y se añade a un JFrame mediante setJMenuBar().***

c. Se utiliza para mostrar barra de herramientas.

d. No es compatible con Swing.

---

11. ¿Qué clase se utiliza para mostrar cuadros de diálogo informativos en Swing?

a. JOptionPane***

b. JDialog

c. JWindow

d. JFrame

---

12. Si necesitas actualizar un componente en Swing desde un hilo en segundo plano, ¿cuál es la técnica recomendada?

a. Llamar directamente al método setText() desde cualquier hilo.
b. Utilizar SwingUtilities.invokeLater() para ejecutar la actualización.***
c. Utilizar Thread.sleep() para sincronizar la actualización.
d. No es posible actualizar con hilos.

---

13. En una aplicación Swing que realice operaciones de Entrada/Salida, ¿cuál es una buena práctica respecto a la gestión de excepciones?

a. Ignorar las excepciones para no interrumpir la interfaz.
b. Capturarlas y mostrarlas mediante cuadros de diálogo para informar al usuario.***
c. Imprimir la excepción en consola y continuar sin más acciones.
d. Reiniciar la aplicación automáticamente.

---

14. ¿Por qué es importante manejar adecuadamente las excepciones de entrada/salida?

a. Para que la aplicación se compile correctamente.
b. Para informar al usuario y evitar la pérdida inesperada de datos o recursos.***
c. Porque Java lo requiere para la conexión a Internet.
d. Porque aumenta la velocidad del programa.

---

15. ¿Qué componente de Swing se utiliza para mostrar contenido que puede desplazarse cuando es demasiado grande para el contenedor?

a. JPanel
b. JScrollPane***
c. JLayeredPane
d. JViewport

---

16. Una buena práctica en el manejo de excepciones durante operaciones de entrada/salida es:

a. Capturarlas y, a su vez, informar al usuario y hacer análisis.***
b. Ignorarlas para no interrumpir la ejecución.
c. Reintentar la operación sin límites.
d. Reiniciar la aplicación en caso de error.

---

17. Cuando se diseñan interfaces con Swing, ¿cuál es una buena práctica para asegurar que la interfaz sea escalable y adaptable a diferentes resoluciones?

a. Fijar tamaños absolutos a todos los componentes.
b. Usar layoutManagers y evitar posiciones y tamaños fijos.***
c. Diseñar la interfaz en un único tamaño y luego ampliarla.
d. Evitar el uso de imágenes o iconos.

---

18. Si se considera que el archivo `C:\empty.txt` existe y está vacío, ¿cuál es la salida del siguiente método?

```
java
Copiar código
void test() throws IOException {
    BufferedReader reader = new BufferedReader(new FileReader("C:/empty.txt"));
    System.out.println(reader.readLine());
    reader.close();
}
```

a. Imprime una cadena vacía “”.
b. Imprime null. **Lega al final del archivo y devuelve null, no puede devolver cadena vacia porque no hay nada**
c. Se lanza una FileNotFoundException.
d. Error de compilación.

---

19. ¿Qué limitación tiene el uso de BufferedReader en este contexto?

```
java
Copiar código
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String input = br.readLine();
System.out.println("Ingresaste: " + input);
br.close();
```

a. No permite leer más de una línea de entrada.
b. Puede bloquear la ejecución del programa hasta que el usuario ingrese datos. **Se queda bloqueado porque el buffer engloba un System.in lo que indica que queda esperando una entrada del usuario**
c. Convierte automáticamente el texto ingresado a minúsculas.
d. No es compatible con la entrada desde la consola (System.in).

---

20. ¿Qué condición verifica este código?

```
java
Copiar código
BufferedReader br = new BufferedReader(new FileReader("vacio.txt"));
String data = br.readLine();
br.close();

if (data == null) {
```

```
    System.out.println("El archivo está vacío o se ha alcanzado el final");
}
```

a. Que el archivo `vacio.txt` contenga al menos una línea de texto.
b. Que el método `readLine()` devuelva una cadena vacía (“”) cuando no hay datos.
c. Que se haya alcanzado el final del flujo, lo que indica que el archivo está vacío o no contiene más líneas.***
d. Que se haya producido un error en la lectura del archivo.

📁 Clase File
● Sirve para representar archivos y directorios.
● Métodos clave:
○ exists(): comprueba si el archivo existe.
○ createNewFile(), delete(), mkdir(), etc.

🔁 2. Streams (Flujos)
● Un stream es una secuencia de datos desde/hacia una fuente/destino (archivo, teclado...).
● Necesitan abrirse (al crear el objeto) y cerrarse con close().
Tipos según datos:
● Binarios (bytes):
○ Entrada: FileInputStream **LEER ARCHIVOS BYTE POR BYTE**
○ Salida: FileOutputStream **ESCRIBIR EN ARCHIVOS BYTE POR BYTE**
● Texto (caracteres):
○ Entrada: FileReader, BufferedReader **LEER ARCHIVOS CHAR A CHAR // LEER ARCHIVOS POR LINEAS**
○ Salida: FileWriter, PrintWriter **ESCRIBIR EN ARCHIVOS CHAR A CHAR // ESCRIBIR EN ARCHIVOS LINEA POR LINEA**
Tipos según dirección:
● Entrada (Input): datos entran al programa.
● Salida (Output): datos salen del programa.
Streams comunes:
● System.in: entrada estándar (teclado)
● System.out: salida estándar (pantalla)
● System.err: salida de error

📥 3. Lectura de archivos
● Texto: con FileReader + BufferedReader
BufferedReader br = new BufferedReader(new FileReader("archivo.txt"));
String linea = br.readLine();

📤 4. Escritura de archivos
Se usa FileWriter y PrintWriter.
PrintWriter pw = new PrintWriter(new FileWriter("archivo.txt"));
pw.println("Hola mundo");
pw.close();

Métodos comunes de `BufferedReader`:

`read()`: Lee un carácter.

`readLine()`: Lee una línea completa.

`close()`: Cierra el flujo y libera recursos.

Métodos comunes de `PrintWriter`:

`print()`: Escribe el texto o valor dado, pero **sin agregar un salto de línea** al final.

`println()`: Escribe el texto o valor dado **y agrega un salto de línea** al final. Este método es útil para imprimir líneas completas.
`write()`: Escribe un único carácter o una cadena de caracteres (sin agregar salto de línea). Es una forma más detallada de escribir en el archivo.
`close()`: Cierra el `PrintWriter` y libera los recursos asociados