Manual Técnico: Manejo de Excepciones

Proyecto: Liminalis

Tabla de Contenidos

Introducción

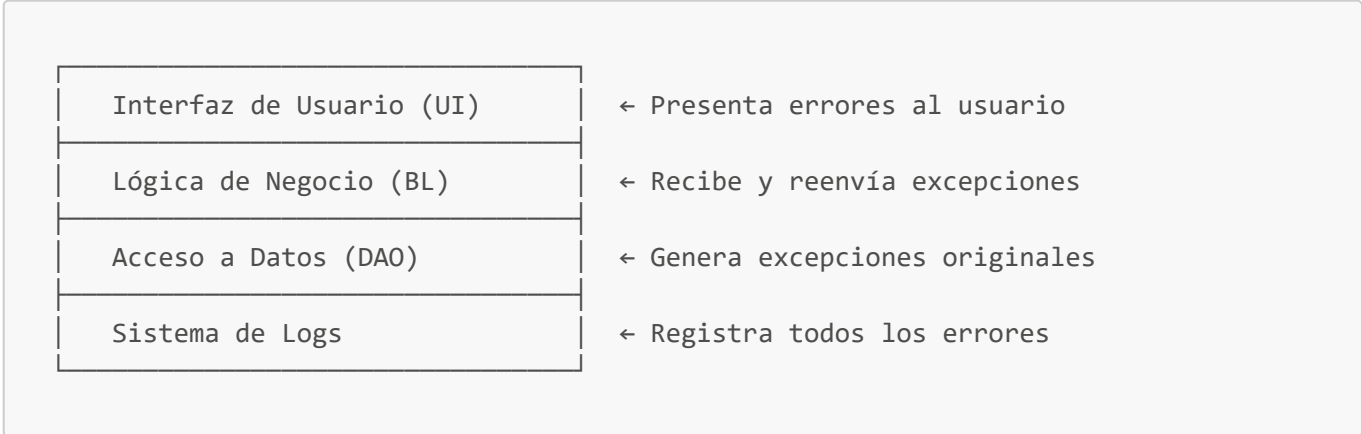
El sistema de manejo de excepciones de Liminalis está diseñado para:

- **Capturar errores** en todas las capas de la aplicación (Data Access, Business Logic, UI)
- **Registrar eventos** en archivos de log con detalles técnicos
- **Presentar mensajes claros** al usuario final en la interfaz gráfica
- **Mantener trazabilidad** del error indicando clase y método donde ocurrió

Este manual técnico describe cómo funciona el sistema completo, desde la generación del error hasta su presentación al usuario.

Arquitectura de Manejo de Excepciones

Estructura de Capas



Características Principales

Componente	Función
`AppException`	Clase personalizada que extiende `Exception`
`AppMSG`	Clase de utilidad para mostrar mensajes a usuarios
`AppConfig`	Configuración centralizada, incluyendo rutas de logs
Logs	Archivo de registro de errores en `%APPDATA%\Liminalis\Logs\`

Clase Base: AppException

¿Qué es AppException?

`AppException` es una clase personalizada que extiende la clase estándar `Exception` de Java. Su propósito es proporcionar un manejo consistente de excepciones en todo el proyecto.

Ubicación: `src/Infrastructure/AppException.java`

Constructores

Constructor 1: Excepción Simple

```
public AppException(String showMsg)
```

Parámetros:

- `showMsg` (String): Mensaje que será mostrado al usuario

Descripción: Se usa cuando ocurre un error sin una excepción subyacente.

Ejemplo:

```
if (nombre == null || nombre.isEmpty()) {
    throw new AppException("El nombre del jugador no puede estar vacío");
}
```

Constructor 2: Excepción con Contexto Técnico

```
public AppException(String showMsg, Exception e, Class<?> clase, String metodo)
```

Parámetros:

- `showMsg` (String): Mensaje amigable para mostrar al usuario
- `e` (Exception): La excepción original que causó el error

- `clase` (Class<?>): La clase donde ocurrió el error
- `metodo` (String): El método donde ocurrió el error

Descripción: Se usa cuando se captura una excepción técnica y se necesita proporcionar contexto.

Ejemplo:

```
try {
    // Código que falla
    Connection conn = openConnection();
    // ... operación de base de datos
} catch (SQLException e) {
    throw new AppException(
        "No se pudo crear el jugador: " + entity.getName(),
        e,                          // Excepción original
        UserPlayerDAO.class,        // Clase donde ocurrió
        "create"                    // Nombre del método
    );
}
```

Métodos Internos

saveLogFile()

```
private void saveLogFile(String logMsg, Class<?> clase, String metodo)
```

Este método es llamado automáticamente por los constructores y realiza:

1. **Genera un timestamp** con formato `yyyy-MM-dd HH:mm:ss`
2. **Extrae el nombre simple** de la clase (ej: `UserPlayerDAO`)
3. **Formatea el mensaje** con un patrón visual consistente
4. **Imprime en consola** (en rojo si hay error)
5. **Guarda en archivo de log** en la ruta configurada

Formato del log:

```
┌─🔥─ SHOW ⚠ <mensaje_usuario>
└──── LOG  ⚠ <timestamp> | <Clase>.<Método> | <mensaje_técnico>
```

Ejemplo de salida:

```
┌─🔥─ SHOW ⚠ No se pudo crear el jugador: Juan
└──── LOG  ⚠ 2026-01-27 14:35:22 | UserPlayerDAO.create | Duplicate entry 'Juan'
```

Sistema de Mensajes: AppMSG

¿Qué es AppMSG?

`AppMSG` es una clase de utilidad que encapsula la presentación de mensajes al usuario mediante cuadros de diálogo de Swing.

Ubicación: `src/Infrastructure/AppMSG.java`

Métodos Disponibles

1. show() - Mensaje Informativo

```
public static final void show(String msg)
```

Uso: Para mostrar mensajes de información al usuario.

```
AppMSG.show("Jugador creado correctamente");
```

Resultado: Cuadro de diálogo con ícono de información.

---

2. showError() - Mensaje de Error

```
public static final void showError(String msg)
```

Uso: Para mostrar mensajes de error al usuario.

```
AppMSG.showError("Error al crear el jugador");
```

Resultado: Cuadro de diálogo con ícono de error.

---

3. showConfirmYesNo() - Confirmación

```
public static final boolean showConfirmYesNo(String msg)
```

Uso: Para obtener confirmación del usuario (Sí/No).

Retorna: `true` si el usuario hace clic en Sí, `false` si en No.

```
if (AppMSG.showConfirmYesNo("¿Estás seguro de continuar?")) {
    // Lógica si el usuario confirma
}
```

Flujo de Burbujeo de Excepciones

¿Qué es el Burbujeo?

El "burbujeo" es el proceso por el cual una excepción se propaga desde la capa más baja (Data Access) hacia la más alta (User Interface), donde finalmente se presenta al usuario.

Diagrama del Flujo

```
┌─────────────────────────────────────────────────────────┐
│ 1. ERROR EN LA BASE DE DATOS (DAO)                      │
│    SQLException → Se captura en UserPlayerDAO.create()  │
├─────────────────────────────────────────────────────────┤
│ 2. SE LANZA APPEXCEPTION                                │
│    throw new AppException(                              │
│        "No se pudo crear el jugador",                   │
│        e,  // SQLException original                     │
│        UserPlayerDAO.class,                             │
│        "create"                                         │
│    )                                                    │
├─────────────────────────────────────────────────────────┤
│ 3. ASCIENDE A LA CAPA DE NEGOCIO (BL)                   │
│    UserPlayerBL.create() recibe AppException            │
│    catch (AppException e) { throw e; } // La reenvía    │
├─────────────────────────────────────────────────────────┤
│ 4. LLEGA A LA INTERFAZ DE USUARIO (UI)                  │
│    CreatePlayer.java captura AppException               │
│    Se muestra al usuario: appEx.getMessage()            │
├─────────────────────────────────────────────────────────┤
│ 5. SE REGISTRA EN LOG                                   │
│    AppException guarda automáticamente en archivo       │
│    (Este proceso ocurre en el paso 2)                   │
└─────────────────────────────────────────────────────────┘
```

Sistema de Logs

Ubicación de Archivos de Log

Los archivos de log se guardan en:

```
%APPDATA%\Liminalis\Logs\
```

Ejemplo en Windows:

```
C:\Users\<NombreUsuario>\AppData\Roaming\Liminalis\Logs\errors.log
```

Configuración

La ruta se define en `src/app.properties`:

```
df.logFile=Logs/errors.log
```

Y se gestiona desde `AppConfig.getLogFile()`.

Contenido del Log

Cada entrada de log contiene:

1. **Mensaje mostrado al usuario** (SHOW)
2. **Timestamp** con fecha y hora exacta
3. **Nombre de la clase** donde ocurrió
4. **Nombre del método** donde ocurrió
5. **Mensaje técnico detallado** (del error original)

Ejemplo de Archivo de Log

```
┌─🔥─ SHOW ⚠ No se pudo leer los jugadores
└──── LOG  ⚠ 2026-01-27 14:23:15 | UserPlayerDAO.readAllStatus | Connection
refused to host.

┌─🔥─ SHOW ⚠ No se pudo crear el jugador: Juan
└──── LOG  ⚠ 2026-01-27 14:25:30 | UserPlayerDAO.create | Duplicate entry 'Juan'
for key 'UserPlayer.Name'

┌─🔥─ SHOW ⚠ No se pudo verificar la existencia del jugador: Juan
└──── LOG  ⚠ 2026-01-27 14:27:45 | UserPlayerBL.exists | java.sql.SQLException
```

Cómo Declarar y Lanzar Excepciones

Paso 1: Declarar que el Método Lanza AppException

En la firma del método, añade `throws AppException`:

```
public boolean create(UserPlayerDTO entity) throws AppException {
    // Implementación del método
```

```
}
```

Paso 2: Capturar Excepciones Técnicas

Usa un bloque `try-catch` para capturar excepciones técnicas:

```
try {
    Connection conn = openConnection();
    PreparedStatement pstmt = conn.prepareStatement(query);
    pstmt.setInt(1, entity.getIdUserType());
    pstmt.executeUpdate();
} catch (Exception e) {
    // Captura cualquier excepción (SQLException, etc.)
    throw new AppException(
        "No se pudo crear el jugador: " + entity.getName(),
        e,
        getClass(),
        "create"
    );
}
```

Paso 3: Lanzar AppException Directamente

Para validaciones o errores lógicos sin excepción subyacente:

```
public int getIdByUsername(String username) throws AppException {
    try {
        UserPlayerDAO userPlayerDAO = new UserPlayerDAO();
        UserPlayerDTO player = userPlayerDAO.readByName(username);
        if (player != null) {
            return player.getIdPlayer();
        }
        // Lanzar directamente sin excepción subyacente
        throw new AppException(
            "Jugador no encontrado: " + username,
            null,  // Sin excepción subyacente
            UserPlayerBL.class,
            "getIdByUsername"
        );
    } catch (AppException e) {
        throw e;
    } catch (Exception e) {
        throw new AppException(
            "Error al obtener el ID del jugador: " + username,
            e,
            UserPlayerBL.class,
            "getIdByUsername"
        );
```

```
    }
}
```

Paso 4: Propagar o Reenviar Excepciones en BL

En la capa de Lógica de Negocio, típicamente reenvías las excepciones:

```
public static Boolean create(String username, int score) throws AppException {
    try {
        UserPlayerDAO userPlayerDAO = new UserPlayerDAO();
        UserPlayerDTO userPlayerDTO = new UserPlayerDTO();
        userPlayerDTO.setName(username);
        userPlayerDTO.setScore(score);
        return userPlayerDAO.create(userPlayerDTO);
    } catch (AppException e) {
        throw e;  // Reenvía la excepción tal como está
    } catch (Exception e) {
        // Captura cualquier otra excepción inesperada
        throw new AppException(
            "No se pudo crear el jugador: " + username,
            e,
            UserPlayerBL.class,
            "create"
        );
    }
}
```

Captura y Presentación de Excepciones

En la Interfaz de Usuario

En la capa de User Interface (screens), las excepciones se capturan y presentan al usuario:

```
JButton create = StyleConfig.createButton("Jugar", StyleConfig.buttonPrimary(),
200, 50);
create.addActionListener(e -> {
    try {
        // Validación simple
        if (playerNameField.getText().trim().isEmpty()) {
            JOptionPane.showMessageDialog(
                namePanel,
                "Nombre no Válido",
                "Error",
                JOptionPane.ERROR_MESSAGE
            );
            MainFrame.setContentPane(CreatePlayer.createPlayerPanel());
        } else {
            try {
```

```
                // Llamada a la lógica de negocio
                UserPlayerBL playerBL = new UserPlayerBL();
                playerBL.create(playerNameField.getText().trim(), 0);

                // Si todo va bien, avanza a la siguiente pantalla
                MainFrame.setContentPane(GameScreen.game());
            } catch (AppException appEx) {
                // La excepción es reenvía con más contexto si es necesario
                throw new AppException(
                    "Error al crear jugador: " + appEx.getMessage(),
                    appEx,
                    CreatePlayer.class,
                    "createPlayerPanel"
                );
            }
        }
    } catch (AppException appEx) {
        // PRESENTACIÓN AL USUARIO
        JOptionPane.showMessageDialog(
            namePanel,
            appEx.getMessage(),  // Mensaje amigable
            "Error",
            JOptionPane.ERROR_MESSAGE
        );
        appEx.printStackTrace();
    }
});
```

Componentes Usados

- **JOptionPane.showMessageDialog()**: Muestra un cuadro de diálogo
- **appEx.getMessage()**: Obtiene el mensaje que se mostró al usuario
- **appEx.printStackTrace()**: Imprime el seguimiento de pila en la consola

---

Ejemplo Práctico Completo

Escenario: Crear un Nuevo Jugador

Este ejemplo muestra todo el ciclo completo de manejo de excepciones.

1. El Usuario Hace Clic en "Crear Jugador"

Archivo: `CreatePlayer.java` (Interfaz)

```
JButton create = StyleConfig.createButton("Jugar", ...);
create.addActionListener(e -> {
    try {
        // Llamada a la lógica de negocio
        UserPlayerBL playerBL = new UserPlayerBL();
        playerBL.create("Juan", 0);  // ← Punto de inicio
```

```
            MainFrame.setContentPane(GameScreen.game());
        } catch (AppException appEx) {
            // Mostrar error al usuario
            JOptionPane.showMessageDialog(
                namePanel,
                appEx.getMessage(),
                "Error",
                JOptionPane.ERROR_MESSAGE
            );
        }
    });
```

---

2. La Lógica de Negocio Procesa la Solicitud

Archivo: `UserPlayerBL.java` (Business Logic)

```
public static Boolean create(String username, int score) throws AppException {
    try {
        UserPlayerDAO userPlayerDAO = new UserPlayerDAO();
        if (exists(username)) {
            int id = getIdByUsername(username);
            return update(username, score, id);
        } else {
            UserPlayerDTO userPlayerDTO = new UserPlayerDTO();
            userPlayerDTO.setIdUserType(1);
            userPlayerDTO.setName(username);
            userPlayerDTO.setScore(score);
            return userPlayerDAO.create(userPlayerDTO);  // ← Llamada a DAO
        }
    } catch (AppException e) {
        throw e;  // Reenvía la excepción
    } catch (Exception e) {
        throw new AppException(
            "No se pudo crear el jugador: " + username,
            e,
            UserPlayerBL.class,
            "create"
        );
    }
}
```

---

3. El DAO Intenta Crear el Registro

Archivo: `UserPlayerDAO.java` (Data Access)

```
@Override
public boolean create(UserPlayerDTO entity) throws AppException {
```

```
    DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
    LocalDateTime now = LocalDateTime.now();
    String query = "INSERT INTO UserPlayer (IdUserType, Name, Score, CreationDate,
ModificateDate) "
            + "VALUES (?, ?, ?, ?, ?);";
    try {
        Connection conn = openConnection();
        PreparedStatement pstmt = conn.prepareStatement(query);
        pstmt.setInt(1, entity.getIdUserType());
        pstmt.setString(2, entity.getName());
        pstmt.setInt(3, entity.getScore());
        pstmt.setString(4, dtf.format(now).toString());
        pstmt.setString(5, dtf.format(now).toString());
        pstmt.executeUpdate();  // ← AQUÍ FALLA SI JUAN YA EXISTE
        return true;
    } catch (Exception e) {  // Captura SQLException
        throw new AppException(
            "No se pudo crear el jugador: " + entity.getName(),
            e,  // La excepción técnica (SQLException)
            getClass(),
            "create"
        );
    }
}
```

---

4. Excepción Ocurre y Se Genera AppException

Lo que sucede internamente:

```
SQLException original:
┗ "Duplicate entry 'Juan' for key 'UserPlayer.Name'"

Se convierte en:
┗ AppException con mensaje para usuario:
   "No se pudo crear el jugador: Juan"

Y se registra automáticamente en log:
┌─🔥─ SHOW ⚠ No se pudo crear el jugador: Juan
└──── LOG  ⚠ 2026-01-27 14:35:22 | UserPlayerDAO.create | Duplicate entry 'Juan'
for key 'UserPlayer.Name'
```

---

5. La Excepción Burbujea Hacia Arriba

Flujo:

1. DAO lanza AppException → UserPlayerBL.create()
2. BL recibe y reenvía → CreatePlayer (interfaz)

3. Interfaz captura y muestra al usuario

---

6. El Usuario Ve el Mensaje

Cuadro de diálogo mostrado:

```
┌──────────────────────────────────────┐
│           Error                      │
├──────────────────────────────────────┤
│                                      │
│  No se pudo crear el jugador: Juan   │
│                                      │
│             [ OK ]                   │
└──────────────────────────────────────┘
```

---

7. Registro en Archivo de Log

Archivo: `%APPDATA%\Liminalis\Logs\errors.log`

```
┌─🔥─ SHOW ⚠ No se pudo crear el jugador: Juan
└──── LOG  ⚠ 2026-01-27 14:35:22 | UserPlayerDAO.create | Duplicate entry 'Juan'
for key 'UserPlayer.Name'
```

Resumen del Ejemplo

Paso	Componente	Acción	Resultado
1	CreatePlayer.java	Usuario hace clic en botón	Llamada a BL
2	UserPlayerBL.java	Procesa solicitud	Llamada a DAO
3	UserPlayerDAO.java	Intenta INSERT	SQLException
4	UserPlayerDAO.java	Captura error	Crea AppException
5	UserPlayerDAO.java	Lanza excepción	Se registra en log
6	UserPlayerBL.java	Recibe excepción	La reenvía
7	CreatePlayer.java	Recibe excepción	Muestra al usuario

Mejores Prácticas

1. Siempre Usar Mensajes Descriptivos

☑ **CORRECTO:**

```
throw new AppException(
    "No se pudo actualizar el puntaje del jugador: " + username,
    e,
    UserPlayerBL.class,
    "update"
);
```

❌ **INCORRECTO:**

```
throw new AppException("Error", e, UserPlayerBL.class, "update");
```

---

2. Incluir el Contexto de Ejecución

Siempre proporciona la clase y el método:

☑ **CORRECTO:**

```
throw new AppException(msg, e, UserPlayerDAO.class, "readByName");
```

❌ **INCORRECTO:**

```
throw new AppException(msg, e, null, null);
```

---

3. Distinguir Entre Reenvío y Captura

Reenvío (en BL):

```
catch (AppException e) {
    throw e;  // Simplemente reenvía
}
```

Captura con Envoltura (en UI):

```
catch (AppException appEx) {
    throw new AppException(
        "Error al procesar: " + appEx.getMessage(),
        appEx,
        CreatePlayer.class,
        "createPlayerPanel"
```

```
    );
}
```

---

4. Usar null Solo para Excepciones Lógicas

Si no hay excepción técnica subyacente (solo una validación lógica):

```
throw new AppException(
    "Jugador no encontrado",
    null,  // No hay excepción subyacente
    UserPlayerBL.class,
    "getIdByUsername"
);
```

---

5. Capturar Excepciones Técnicas Antes de AppException

```
try {
    // Código que puede fallar
} catch (SQLException e) {
    // Captura excepción específica primero
    throw new AppException(..., e, ...);
} catch (IOException e) {
    // Captura otra excepción específica
    throw new AppException(..., e, ...);
} catch (Exception e) {
    // Captura genérica al final
    throw new AppException(..., e, ...);
}
```

---

6. Mantener Consistencia en Mensajes

Usar un patrón consistente en los mensajes:

- `"No se pudo <acción>: <identificador>"`
- `"Error al <acción>: <identificador>"`

☑ Consistentes:

```
"No se pudo crear el jugador: Juan"
"No se pudo actualizar el puntaje: 42"
"No se pudo leer los jugadores"
```

---

7. Revisar Regularmente los Logs

Los administradores deben revisar periódicamente:

```
%APPDATA%\Liminalis\Logs\errors.log
```

Para identificar patrones de errores y problemas de producción.

Checklist de Implementación

Cuando implementes manejo de excepciones en un nuevo método:

- [ ] Declarar `throws AppException` en la firma del método
- [ ] Envolver código que falla en `try-catch`
- [ ] Capturar excepciones técnicas específicas (SQLException, IOException, etc.)
- [ ] Lanzar `AppException` con mensaje descriptivo, excepción original, clase y método
- [ ] En BL: Reenviar `AppException` tal como está
- [ ] En DAO: Envolver cualquier excepción en `AppException`
- [ ] En UI: Capturar `AppException` y mostrar al usuario
- [ ] Probar que el log se crea correctamente

Conclusión

El sistema de excepciones de Liminalis proporciona:

1. **Captura centralizada** de todos los errores mediante `AppException`
2. **Burbujeo controlado** desde DAO → BL → UI
3. **Registro automático** en archivos de log para auditoría
4. **Mensajes claros** al usuario sin exponer detalles técnicos
5. **Trazabilidad completa** del error con clase, método y timestamp

Este enfoque garantiza que ningún error pase desapercibido y que tanto los usuarios como los desarrolladores reciban la información que necesitan para resolver problemas.

Última actualización: 27 de enero de 2026
Versión: 1.0
Autor: Documentación Técnica - Proyecto Liminalis