TEMA 6: **DISEÑO DE CLASES**

PROBLEMA 1. **THE LEGEND OF ZELDA**[1]

Actividad 1: Acoplamiento, cohesión y diseño dirigido por responsabilidades

Los proyectos *zelda_mal, zelda_mejorado_nohash* y *zelda_mejorado* implementan una parte del juego "The Legend of Zelda" donde Link se mueve por el Castillo de Hyrule en busca de la sala del tesoro. Como no tenemos conocimientos suficientes para generar una interfaz gráfica, el juego está basado en interfaz textual. Las clases `Command`, `CommandWords` y `Parser` se utilizan para interpretar los comandos que puede introducir el usuario en la interfaz textual. Nosotros trabajaremos con las clases `Juego` y `Habitacion`.

En las clases `Juego` y `Habitacion`, el proyecto *zelda_mal* contiene ejemplos de decisiones de diseño mal tomadas que dan lugar a alto acoplamiento y baja cohesión, características no deseables en un diseño de clases. Por su parte, los proyectos *zelda_mejorado_nohash* y *zelda_mejorado* contienen ejemplos de decisiones de diseño bien tomadas que dan lugar a bajo acoplamiento y alta cohesión.

Realiza las siguientes actividades para comprender el funcionamiento del juego y ver cómo impactan las decisiones de diseño en el mantenimieto y modificación de la aplicación.

- Ejecuta mediante BlueJ el proyecto *zelda_mal* y responde a las siguientes preguntas. ¿Qué tres comandos acepta el juego? ¿Qué función desempeña cada comando? ¿Cuántas habitaciones hay en el Castillo de Hyrule? Dibuja un mapa de las habitaciones del castillo.

- En *zelda_mal*, las líneas de la clase `Juego` 80 – 90 se encuentran repetidas en otro lugar. ¿Dónde? ¿Cómo se evita la duplicación de estas líneas de código en *zelda_mejorado_nohash* y *zelda_mejorado*? ¿Cómo se mejora la cohesión de los métodos involucrados? (Para visualizar los números de línea de una clase ve a Options + Preferences y marca la casilla Display line numbers)

[1]Problema basado en el ejercicio "World of Zulu" del libro: Barnes David J., Kölling Michael, "*Programación orientada a objetos con Java: una introducción práctica usando BlueJ*" , Ed. 3ª Pearson Education

- La definición de los campos en *zelda_mal* son un ejemplo evidente de acoplamiento alto entre la clase `Juego` y la clase `Habitacion`. ¿Mediante qué mecanismo se soluciona en *zelda_mejorado_nohash*?

- Analiza en *zelda_mal* las líneas de la clase `Juego` 45 – 50 y en *zelda_mejorado_nohash* o *zelda_mejorado* las líneas de la clase `Juego` 45 – 57. ¿Cuál de las dos maneras de definir las salidas te parece más acertada? ¿Por qué?.

- Añade las salidas “arriba” y “abajo” y las habitaciones “Salón de la Princesa” y “Mazmorras” de modo que se pueda llegar al “Salón de la Princesa” desde la “Sala del Tesoro” utilizando la salida “arriba” y a las “Mazmorras” desde “Ala este” y “Ala oeste” utilizando la salida “abajo”. ¿Qué partes del código de *zelda_mal*, *zelda_mejorado_nohash* y *zelda_mejorado* has tenido que modificar? ¿Cuál de los tres proyectos ha resultado más fácil de modificar?

Nota: el proyecto *zelda_mejorado* en lugar de utilizar variables independientes para almacenar las salidas como hacían *zelda_mal* y *zelda_mejorado_nohash*, usa una estructura denominada `HashMap`. Esta estructura permite almacenar una correspondencia entre el nombre de la dirección a la que se quiere llegar y la habitación a la que se llega mediante dicha dirección. La utilización de esta estructura minimiza el número de cambios a realizar cuando se desea introducir nuevas salidas a las habitaciones.