Ranking

Esta clase está implementada usando el patrón singleton, ya que no va a haber más de una instancia perteneciente a esta clase.
Como atributos tiene la instancia de ranking y una lista de pares de String y Integers. Uso pares para poder acceder de forma rápida al nombre del jugador y sus puntos. Además, como lista uso ArrayList debido a que puedo insertar elementos en posiciones específicas de la lista.

Jugador

Esta clase es la encargada de la administración de los usuarios. Permite dar de alta un jugador, darlo de baja y modificar su nombre y su contraseña.
Como atributos tiene el nombre del usuario, la contraseña de este, una variable booleana IA que nos dice si el jugador es la máquina o humano, un atributo de tipo entero nFichas que nos indica el número de fichas con las que jugará cada partida ese jugador y que nos sirve para pasárselo a sus subclases CodeMaker y CodeBreaker y el atributo nColores que nos indica el número de colores, de valores que puede coger cada ficha. Esta clase conecta con la clase JugadorPersistencia para la administración de los usuarios de poderlos guardar en ficheros para poderlos mantener una vez el programa termine.

CodeBreaker

La explicación de esta clase la voy a separar en dos, IA y humano:

IA: En este caso se utilizará las variables globales y encontramos 3 matrices de enteros, compatibles que contendrá todas aquellas soluciones que sean compatibles con las pistas que ha ido dando el codemaker, la matriz noUsados que contiene todos los valores posibles que puede introducir como respuesta y la matriz combinaciones que contiene todas las pistas posibles que puede darnos el codemaker.
He usado ArrayList para conseguir las matrices porque nos sugirieron en teoría que no usaramos arrays porque se complicaba un poco y las funciones que nos proporcionan los arraylist como el contains, me eran de gran utilidad.

Humano: En este caso no se utilizarán las variables globales, que ni siquiera las inicializará. Lo que hace aquí es pedir al usuario que introduzca un número determinado de fichas (las que haya escogido al empezar la partida) con unos determinados colores (que también habrá escogido) y una vez recogidos estos valores mirará que sean correctos y los enviará allí donde haya sido llamada.

Codemaker

En la explicación de esta clase también la voy a separar en los dos mismo que en codebreaker, IA y humano, aunque en este caso no me ha hecho falta usar variables globales ya que con locales se podían conseguir todas las funciones:

IA: En el caso de dar el patrón a adivinar por el codebreaker, nos genera un patrón aleatorio de nFichas con nColores. En el caso de dar la pista, llamará a una función que lo que hará será comparar la tirada del codebreaker con la solución que hemos generado.

Humano: En el caso de dar el patrón, pedirá al usuario que lo introduzca por el teclado**,** comprobando que aquello que ha introducido sea correcto dentro de nFichas y nColores.
En el caso de jugar, pedirá al usuario que mediante el teclado de la pista para que el codebreaker pueda seguir intentando encontrar su patrón y comprobará que la pista dada sea de nFichas y esté entre 0 y 2 que son los valores permitidos.

Game

Esta clase es la encargada de gestionar toda la partida en general, de pedir las jugadas a los dos jugadores y de generar el tablero a partir de ellas.
La acción principal se centra en la función *juega*. Esta función recibe como parámetros el objeto Jugador, un String con el identificador de la partida, otro String que representa la dificultad, otro String para el modo (codemaker o codebreaker) y dos ints, que representan el número de elementos que tiene la combinación y el rango de cada uno de los elementos de la combinación. Esta función luego llama a la función SetAtributos, donde se asignan esos atributos a las variables globales de Game. Si se han podido asignar correctamente, entra en el bucle que representa el tablero (una iteración para cada turno).
A continuación se declaran dos ArrayList de Integers: outputM y outputB, donde se almacenan las jugadas de CodeMaker y CodeBreaker, respectivamente, de ese turno. Se ha escogido hacer el ArrayList de Integers ya que es más fácil trabajar con ellos que con CodePegs y KeyPegs y, si el dato que introduce el jugador es incorrecto o es el -1 (guardar partida) o el -2 (salir de la partida) nos ahorramos convertirlo a CodePeg y KeyPeg.
Seguidamente el jugador entra en un bucle de CodeMaker o CodeBreaker, dependiendo de su rol. He escogido hacer un while porque si el jugador introduce -1, en un caso normal, se guardará la partida y se volverá al menú principal. Pero suponiendo el caso de que el guardado falle y que el jugador quiera seguir con la partida para, al menos, poder acabarla, el bucle le permitiría volver a hacer una jugada en ese turno. De esta manera, a no ser que el jugador haga una jugada válida, Game le seguirá pidiendo que haga una jugada.
Si la jugada que introduce es correcta, se convierte a CodePeg (CodeBreaker) o a KeyPeg (CodeMaker) y se procede a generar el output por pantalla de todo el teclado.
Cuando se acaba la partida, se llama a finishGame pasando como atributo un true si el CodeBreaker ha acertado la combinación o false si, por el contrario, se ha alcanzado el máximo de turnos.

En su variante de Persistencia, GamePersistencia, se encuentran las funciones SaveGame, que traduce Game y CodeBreaker a un archivo de bytes mediante serialización, y la función LoadGame, que traduce ese archivo de bits de vuelta a los objetos Game y CodeBreaker.