Estructuras de datos:

La principal unidad de almacenamiento empleada en nuestro código es el Kakuro, una matriz de Casillas. Cada Casilla consta principalmente de un tipo, que hace referencia a la función que tiene en el juego: blanca o negra.
Las negras pueden ser, además, máximo de fila y/o columna, por lo que hemos añadido un atributo a la clase que indica (si son negras) si son máximo horizontal y/o vertical. Tanto para estas últimas como para las blancas necesitamos almacenar un valor, que tenemos implementado como atributo de la propia clase.

Para almacenar los Kakuros hemos empleado un directorio, manejado en su totalidad por una clase: Biblioteca_Kakuro. Básicamente la clase se encarga de guardar los Kakuros que hemos generado durante la ejecución o que ha introducido el usuario por consola en formato ".txt" (string) con un ID único. En el caso de que sea necesario volver a usar un Kakuro, la clase accede a través del id al mismo, lo convierte de nuevo a Kakuro y se lo envía al que lo solicitó.

Algoritmos:

Todos los métodos a los que haremos referencia en este apartado están explicados con detalle en la sección "Descripción de clases".
Para implementar las principales funcionalidades del programa (generar y resolver) hemos creado 2 algoritmos con estrategias completamente diferentes:

Resolver Kakuro:

Para implementar este algoritmo hemos trabajado con un vector que almacena las posiciones de todas las casillas blancas que hay en el tablero. Esto nos ayudará a la hora de hacer las llamadas recursivas. El método *solve* se encarga de inicializar este vector y de llamar a *solve_Kakuro*. También imprime los mensajes informativos correspondientes.

La función recursiva *solve_Kakuro* se encarga de, para cada casilla blanca almacenada con anterioridad, probar valores entre 1 y 9. El valor se decide teniendo en cuenta el resultado de las llamadas a los métodos takeFila, takeColumna y *checkRun.* Estos se aseguran de que no exista ninguna otra casilla blanca con el mismo valor en la misma fila y columna, y de que la suma de los valores de estas no superen el máximo indicado por una casilla negra al extremo. En el caso de que alguna de las condiciones no se cumpla, procedemos a hacer *backtracking* y regresar a comprobar la casilla blanca anterior con valor + 1.

Como **optimización para la segunda entrega** hemos incorporado un set de enteros *possible* que incorpora los posibles valores que optarían a pertenecer a una casilla.
El contenido de este set (inicializado con todos los valores entre el 1 y el 9) será determinado por 3 funciones:

-takeFila y takeColumna: calculan la diferencia entre el máximo de la línea y la suma del resto de valores introducidos en la misma. Los valores del set que no devuelvan una suma >=0 serán descartados.

-filtrarSumas: para valores máximos de línea determinados, hemos determinado (por lógica) para qué enteros no sería posible construir una suma que cumpla con los requerimientos del Kakuro

Además, habrá 2 funcionalidades distintas para este algoritmo: una para encontrar exclusivamente 1 solución; y otra que utilizaremos para el siguiente algoritmo (Generar Kakuro), que comprobará todas las soluciones posibles. La utilidad la explicaremos a continuación.

Generar Kakuro:

Para generar un Kakuro hemos utilizado una solución iterativa, colocando casillas negras de forma aleatoria y colocando valores random en las casillas blancas, valores que después se sumarán para calcular los máximos de fila y columna correspondiente.

Para comenzar llamaremos a la función *generar* pasándole una dificultad en forma de entero (1 fácil, 2 medio, 3 difícil). Este nos servirá para obtener el número de filas y columnas de nuestro Kakuro (cada nivel de dificultad tiene asociado un rango de valores, el cual se decidirá calculando un número random dentro del mismo), y la proporción entre casillas blancas y negras (la dificultad va asociada a una proporción). A continuación llamaremos a *generarKakuro* pasándole los parámetros calculados. Esta función simplemente calculará el número de casillas negras que se generarán y acudirá en bucle al método generarK hasta que se encuentre una solución que satisfaga los atributos calculados.
Por último, esta función se encargará de, apoyandose en otros métodos con una función concreta:

1. Generar el número de casillas negras solicitado en posiciones random del tablero.
2. Generar valores aleatorios en las casillas blancas (las restantes) satisfaciendo las condiciones impuestas por las normas del juego.
3. Calcular las sumas correspondientes a las líneas creadas por el punto anterior y situarlas en los extremos de las filas y las columnas.

Por último nos quedará almacenar este Kakuro generado en la biblioteca con un ID único, mecanismo que explicamos en “Descripción de clases”.

Como **cambio para la segunda entrega** hemos determinado que un Kakuro generado por este algoritmo no será válido si tiene más de una solución. En el caso de que así sea (determinada por la función validar explicada para el algoritmo de resolver), se repetirá el proceso hasta encontrar uno.