Enunciado: Hunde la flota

Completa el código que queda por hacer para que la aplicación funcione correctamente. Se debe poder jugar una partida a Hunde la flota y se tienen que poder crear nuevas partidas, ver las soluciones y salir del juego.

Las modificaciones que hay que hacer en el código son:

En la clase `AuxiliarClienteFlota` hay que completar los métodos:

- nuevaPartida
- pruebaCasilla
- getBarco

En la clase `ClienteFlotaSocket` hay que completar el método:

- pintarBarcoHundido

En la clase `HiloServidorFlota` hay que completar:

- case 0, 1, 2, 3 del switch.

A continuación hay algunas explicaciones (por si las necesitas) sobre las clases del juego:

`MyStreamSocket`

Esta clase en Java, llamada `MyStreamSocket`, encapsula la funcionalidad de los sockets para facilitar su uso al enviar y recibir mensajes a través de una red. Un socket es un punto final para enviar o recibir datos a través de una red, y esta clase proporciona una interfaz más simple para trabajar con sockets en comparación con las clases de socket estándar de Java.

1. Constructores:
    - Constructor 1: Toma la dirección IP del host y el número de puerto al que desea conectarse. Internamente, crea un nuevo socket usando esa dirección IP y puerto, y luego llama al método `setStreams()` para inicializar los flujos de entrada y salida para ese socket.
    - Constructor 2: Toma un objeto Socket existente y lo utiliza para establecer los flujos de entrada y salida. Esto es útil cuando se recibe un socket como parámetro y se quiere envolver con esta clase para facilitar las operaciones de entrada/salida.
2. Método `setStreams()`: Este método se encarga de configurar los flujos de entrada (`BufferedReader` para leer datos) y salida (`PrintWriter` para escribir datos) para el socket. Estos flujos simplifican la lectura y escritura de datos en el socket.

3. Método `sendMessage(String message)`: Este método permite enviar un mensaje a través del socket. Toma un mensaje como entrada, lo envía a través del flujo de salida del socket y asegura que los datos se escriban en el flujo del socket llamando a `flush()`. El mensaje se termina con un carácter de nueva línea (`\n`) para indicar el final del mensaje.

4. Método `receiveMessage()`: Este método permite recibir un mensaje desde el socket. Lee una línea completa de datos del flujo de entrada del socket y devuelve el mensaje como una cadena de texto.

5. Método `close()`: Cierra el socket, lo que libera los recursos asociados y termina la conexión con el otro extremo.

`Partida`

La clase `Partida` en el código proporcionado representa una partida del juego de hundir la flota. Aquí hay una explicación detallada de la clase:

1. Constantes:
    - `AGUA`, `TOCADO`, y `HUNDIDO` son constantes enteras utilizadas para representar el estado de una casilla en el tablero.
2. Atributos:
    - `mar`: Una matriz que representa el tablero del juego. Cada elemento de la matriz contiene el estado de una casilla (agua, barco tocado o barco hundido).
    - `numFilas` y `numColumnas`: Representan el número de filas y columnas en el tablero, respectivamente.
    - `barcos`: Un vector dinámico que contiene objetos `Barco` que representan los barcos colocados en el tablero.
    - `numBarcos`: Número total de barcos en la partida.
    - `quedan`: Número de barcos que aún no han sido hundidos.
    - `disparos`: Número de disparos efectuados durante la partida.
3. Constructores:
    - Constructor por Defecto: No hace nada especial.
    - Constructor con Parámetros: Inicializa el tablero con el número especificado de filas, columnas y barcos. Llama a métodos privados para inicializar la matriz del tablero y colocar los barcos en posiciones aleatorias.
4. Método `pruebaCasilla(int f, int c)`:

- Este método se utiliza para disparar a una casilla específica en el tablero.
- Actualiza el estado de la casilla y devuelve el resultado del disparo (agua, tocado, hundido o identidad del barco hundido).

5. Métodos `getBarco(int idBarco)` y `getSolucion()`:
   - `getBarco(int idBarco)`: Devuelve una cadena con los datos de un barco específico, identificado por su índice en el vector de barcos.
   - `getSolucion()`: Devuelve un vector de cadenas con los datos de todos los barcos en el tablero.
6. Métodos Privados:
   - `iniciaMatriz(int nf, int nc)`: Inicializa todas las casillas del tablero como agua.
   - `ponBarcos()`: Coloca los barcos en posiciones aleatorias en el tablero.
   - `ponBarco(int id, int tam)`: Coloca un barco específico en el tablero, garantizando que no se solape con otros barcos y que tenga espacio suficiente alrededor de él.
   - `librePosiciones(int fila, int col, int tam, char ori)`: Comprueba si hay espacio suficiente para colocar un barco de cierto tamaño y orientación a partir de una casilla inicial.

Esta clase encapsula la lógica del juego de hundir la flota, gestionando el estado del tablero y proporcionando métodos para realizar disparos y obtener información sobre los barcos colocados.

Barco

La clase `Barco` representa un barco en el juego de "Hundir la flota". Aquí hay una explicación detallada de la clase:

1. Atributos:
   - `filaInicial` y `columnaInicial`: Representan las coordenadas iniciales del barco en el tablero.
   - `orientacion`: Indica si el barco está orientado horizontal (`'H'`) o verticalmente (`'V'`).
   - `tamanyo`: Número de casillas que ocupa el barco en el tablero.
   - `tocadas`: Número de casillas del barco que han sido tocadas por un disparo.
2. Constructores:

- Constructor por Defecto: No hace nada especial.
- Constructor con Parámetros: Inicializa el barco con las coordenadas iniciales, orientación y tamaño especificados.

3. Método `tocaBarco()`:
   - Este método se utiliza para marcar una nueva casilla del barco como tocada.
   - Incrementa el contador de casillas tocadas y devuelve un valor lógico indicando si todas las casillas del barco han sido tocadas, es decir, si el barco ha sido hundido.
4. Método `toString()`:
   - Sobrescribe el método `toString()` para devolver una cadena que representa el barco.
   - La cadena contiene las coordenadas iniciales del barco, su orientación y su tamaño, separados por el carácter `#`.
5. Métodos Getters:
   - Se proporcionan métodos para acceder a los atributos privados del objeto (`getFilaInicial()`, `getColumnaInicial()`, `getOrientacion()`, `getTamanyo()`, `getTocadas()`).

Esta clase encapsula la información de un barco en el juego, proporcionando métodos para actualizar su estado y obtener detalles sobre su posición y tamaño.

`ServidorFlotaSockets`

La clase `ServidorFlotaSockets` implementa un servidor para el juego "Hundir la flota" utilizando sockets en modo stream para la comunicación entre el servidor y los clientes. A continuación se explica lo que hace la clase:

1. **Establece un Servidor Socket:**
   - La clase crea un `ServerSocket` en el puerto 8000 para escuchar las conexiones entrantes de los clientes.
2. **Espera Conexiones Entrantes:**
   - La ejecución del servidor entra en un bucle infinito (`while (true)`) donde espera constantemente nuevas conexiones de clientes utilizando el método `accept()` del `ServerSocket`. El servidor se mantiene activo y escuchando en este bucle.

3. **Acepta Nuevas Conexiones:**
    - Cuando un cliente se conecta, se acepta su conexión y se crea un objeto `MyStreamSocket` para manejar la comunicación con ese cliente específico.
4. **Crea Hilos para Clientes:**
    - Para manejar la comunicación con cada cliente de forma concurrente, se crea un nuevo hilo (`Thread`) para cada conexión aceptada. Se instancia un objeto de la clase `HiloServidorFlota` (que implementa la interfaz `Runnable` y maneja la lógica específica del juego para un cliente individual) y se inicia el hilo.
5. **Manejo Concurrente de Clientes:**
    - La capacidad para iniciar múltiples hilos permite que el servidor maneje múltiples clientes simultáneamente. Cada cliente tiene su propia instancia de `MyStreamSocket` para la comunicación y se maneja en su propio hilo separado.
6. **Manejo de Excepciones:**
    - Se manejan excepciones generales y se imprimen los detalles en la consola en caso de que ocurra un error durante la ejecución del servidor.

`HiloServidorFlota`

La clase `HiloServidorFlota` implementa la interfaz `Runnable` y representa un hilo del servidor que se encarga de manejar la comunicación con un cliente específico del juego "Hundir la flota". A continuación, se explica la funcionalidad principal del código:

1. **Atributos:**
    - `myDataSocket`: Un objeto de tipo `MyStreamSocket` que representa el socket de comunicación con el cliente.
    - `partida`: Un objeto de tipo `Partida` que representa el estado del juego para el cliente asociado con este hilo.
2. **Constructor:**
    - El constructor toma un objeto `MyStreamSocket` como parámetro y lo asigna al atributo `myDataSocket`.
3. **Método `run()`:**

- Este método se ejecuta cuando se inicia el hilo.
- En un bucle continuo, el hilo espera mensajes del cliente usando el método `receiveMessage()` del objeto `myDataSocket`.
- Dependiendo de la operación especificada por el cliente, el hilo realiza diversas acciones, como crear una nueva partida, probar una casilla, obtener datos de un barco o enviar la solución al cliente.
- La lógica específica para cada operación está marcada con comentarios `//TODO` y debe ser implementada en el código.

4. **Gestión de Operaciones:**
   - El código utiliza un número entero para identificar la operación que el cliente solicita. Dependiendo del número de operación, se espera un cierto formato de mensaje y se realiza la acción correspondiente.
   - Por ejemplo, cuando el cliente solicita la operación 4 (enviar solución), el servidor primero envía el número de barcos y luego envía la información de cada barco al cliente usando el método `sendMessage()` del objeto `myDataSocket`.

`ClienteFlotaSockets`

El fichero **`ClienteFlotaSockets.java`** gestiona el juego "Hundir la flota" en un cliente utilizando una interfaz gráfica (GUI) en Java. Aquí hay una descripción de las principales características y funcionalidades del código:

1. **Clase `ClienteFlotaSockets`:**
   - Representa el cliente del juego "Hundir la flota".
   - Crea una instancia de la clase `AuxiliarClienteFlota` para gestionar la comunicación con el servidor del juego.
   - Implementa la interfaz gráfica del juego mediante la clase interna `GuiTablero`.
   - Contiene clases internas `MenuListener` y `ButtonListener` para manejar los eventos de los elementos del menú y los botones del tablero, respectivamente.
2. **Clase Interna `GuiTablero`:**

- Representa la interfaz gráfica del juego, incluyendo el tablero, los botones y otros componentes visuales.
- Permite mostrar el tablero del juego, manejar las interacciones del usuario y mostrar mensajes de estado.
- Utiliza botones para representar las casillas del tablero, donde el usuario puede hacer clic para disparar.
- Contiene métodos para mostrar la solución, pintar barcos hundidos, cambiar el estado del tablero y liberar recursos.

3. **Clase Interna `MenuListener`:**
   - Implementa `ActionListener` para manejar eventos del menú de opciones del juego.
   - Permite al usuario mostrar la solución, empezar una nueva partida o salir del juego.
4. **Clase Interna `ButtonListener`:**
   - Implementa `ActionListener` para manejar los eventos de los botones del tablero.
   - Permite al usuario interactuar con las casillas del tablero y realiza las acciones correspondientes en función del estado de las casillas.

`AuxiliarClienteFlota`

La clase `AuxiliarClienteFlota` actúa como una interfaz entre el cliente del juego "Hundir la flota" y el servidor del juego. Aquí está una descripción de los métodos y funcionalidades proporcionadas por esta clase:

1. **Constructor `AuxiliarClienteFlota(String hostName, String portNum)`:**
   - Este constructor crea un objeto `AuxiliarClienteFlota` y establece una conexión con el servidor en la dirección `hostName` y el puerto `portNum`.
2. **Método `fin() throws IOException`:**
   - Envía una petición al servidor para indicar el fin de la conexión (`"0"`).
   - Cierra el socket de conexión con el servidor.

3. **Método `nuevaPartida(int nf, int nc, int nb) throws IOException`:**
    - Envía una petición al servidor para crear una nueva partida con el número de filas `nf`, número de columnas `nc` y número de barcos `nb` (`"1#nf#nc#nb"`).
4. **Método `pruebaCasilla(int f, int c) throws IOException`:**
    - Envía una petición al servidor para probar una casilla en la fila `f` y columna `c` (`"2#f#c"`).
    - Recibe el resultado del disparo devuelto por el servidor.
5. **Método `getBarco(int idBarco) throws IOException`:**
    - Envía una petición al servidor para obtener los datos de un barco específico con la identidad `idBarco` (`"3#idBarco"`).
    - Recibe los datos del barco devueltos por el servidor.
6. **Método `getSolucion() throws IOException`:**
    - Envía una petición al servidor para obtener la solución del juego (`"4"`).
    - Recibe el número de barcos y luego recibe los datos de cada barco devueltos por el servidor.