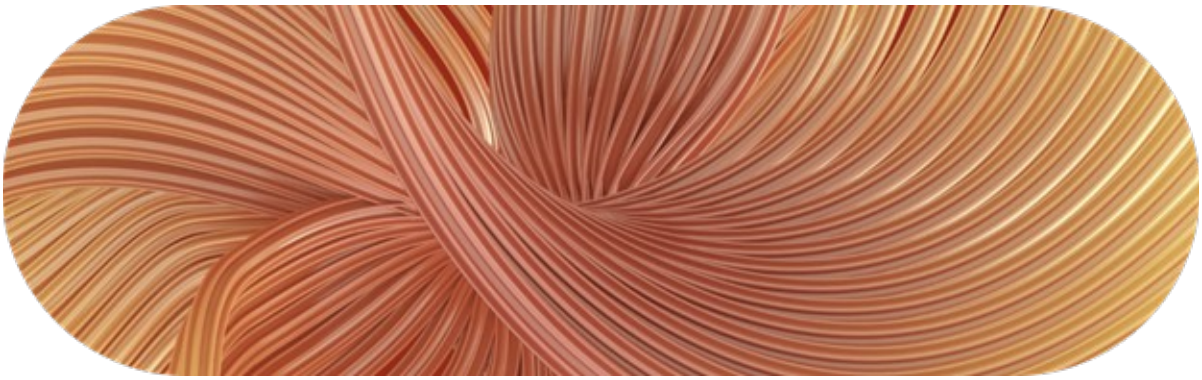
MANUAL TÉCNICO



Francisco Miser Junquera

Marcos Vidal González

David Torres Rial

ÍNDICE

1. Clase AlmacenCentral

1.1 Descripción

Se representa un almacén central con capacidad para almacenar comida animal y vegetal, permitiendo gestionar eficientemente los recursos necesarios para la piscifactoría. Se proporciona funcionalidad para aumentar la capacidad, añadir comida y distribuir los recursos a las piscifactorías registradas.

1.2 Funcionalidades

1. Se permite almacenar comida animal y vegetal en cantidades limitadas por la capacidad máxima del almacén.
2. Se ofrece la posibilidad de ampliar la capacidad máxima del almacén mediante el gasto de monedas.
3. Se distribuye comida animal y vegetal equitativamente entre las piscifactorías disponibles, garantizando el abastecimiento según sus necesidades actuales.
4. Se imprime un informe detallado de la capacidad, cantidades de comida almacenadas y porcentajes de ocupación.

1.3 Constructor

- **public AlmacenCentral():**

  Se define el constructor de la clase AlmacenCentral. Se inicializan los siguientes valores predeterminados:
  Inicialización de atributos:
  capacidadMaxima: Se establece en 200, definiendo la capacidad máxima inicial del almacén.
  cantidadComidaAnimal: Se inicializa en 0, indicando que no hay comida animal almacenada al principio.
  cantidadComidaVegetal: Se inicializa en 0, indicando que no hay comida vegetal almacenada inicialmente.

1.4 Atributos

- **private int capacidadAlmacen:** Se indica la capacidad máxima que el almacén puede contener.
- **private int cantidadComidaAnimal**: Se registra la cantidad actual de comida animal almacenada.
- **private int cantidadComidaVegetal**: Se registra la cantidad actual de comida vegetal almacenada.
- **private final int costoMejora**: Se define el costo fijo en monedas necesario para mejorar la capacidad del almacén. Su valor es de 200.

1.5 Métodos

aumentarCapacidad

- **public void aumentarCapacidad()**

Se incrementa la capacidad del almacén central en 50 unidades si hay monedas suficientes. Si se realiza con éxito, se imprime un mensaje informando el nuevo nivel de capacidad.

añadirComidaAnimal

- **public void añadirComidaAnimal(int cantidad)**

Se añade comida animal al almacén en una cantidad especificada. Si la cantidad es válida y no supera la capacidad total, se actualiza el almacenamiento. En caso contrario, se muestra un mensaje indicando que la cantidad excede la capacidad.

añadirComidaVegetal

- **public void añadirComidaVegetal(int cantidad)**

Se añade comida vegetal al almacén en una cantidad especificada. Si la cantidad es válida y no supera la capacidad total, se actualiza el almacenamiento. En caso contrario, se informa que la capacidad ha sido superada.

distribuirComida

- **public void distribuirComida(List<Piscifactoria> piscifactorias)**

Se distribuye equitativamente la comida disponible entre las piscifactorías registradas. Se revisa cada piscifactoría para determinar sus necesidades de comida y se les asignan recursos según la cantidad disponible y el espacio libre en sus tanques.

2. Clase SistemaMonedas

2.1 Descripción

Se define la clase SistemaMonedas como un sistema de gestión de monedas virtuales dentro de la simulación. Se implementa como un patrón Singleton, asegurando que solo exista una instancia global para administrar la economía del juego. Se permite consultar, aumentar y gastar monedas según sea necesario, aplicando descuentos en ciertas transacciones.

2.2 Funcionalidades

1. Se permite consultar, aumentar y reducir la cantidad de monedas disponibles.
2. Se asegura que las transacciones solo se realicen cuando el saldo sea suficiente y las cantidades sean válidas.
3. Se aplica un descuento automático al calcular costos según la cantidad de recursos gestionados.

2.3 Constructor

- **private SistemaMonedas():**

  Se inicializa el sistema de monedas con un saldo inicial de 100 monedas. El constructor es privado para implementar el patrón Singleton, asegurando que solo una instancia sea creada.
  Inicialización de atributos:
  monedas: Se establece en 100, definiendo el saldo inicial de monedas disponibles en el sistema.

2.4 Atributos

- **private int monedas:** Se registra la cantidad actual de monedas disponibles en el sistema.
- **private static SistemaMonedas instance**: Se almacena la única instancia permitida del sistema de monedas, según el patrón Singleton.

2.5 Métodos

ganarMonedas

- **public boolean ganarMonedas(int cantidad)**

Se incrementa el saldo de monedas en una cantidad específica. Si la cantidad es válida (mayor que 0), se actualiza el saldo y se retorna true. En caso contrario, se retorna false.

gastarMonedas

- **public boolean gastarMonedas(int costo)**

Se reduce el saldo de monedas según un costo especificado. Si el saldo es suficiente y el costo es válido, se actualiza el saldo y se retorna true. Si el costo es mayor que el saldo disponible, se retorna false.

calcularDescuento

- **public int calcularDescuento(int cantidadComida)**

Se calcula el costo total aplicando un descuento de 5 monedas por cada 25 unidades de comida comprada. Si la cantidad de comida es inferior a 25, se retorna el costo completo sin descuento.

3. Clase FileHelper

3.1 Descripción

Se define la clase FileHelper como una utilidad estática para gestionar operaciones relacionadas con archivos y directorios. Se permite crear carpetas, verificar contenido en directorios y listar archivos para su selección en un menú interactivo. La clase también maneja errores y registra incidencias utilizando un sistema de registro de errores (Simulador.logger).

3.2 Funcionalidades

1. Se permite crear directorios de forma segura y verificar su contenido.
2. Se permite listar archivos en un directorio y mostrar un menú interactivo para seleccionar uno.
3. Se registran errores en caso de operaciones fallidas, como rutas inexistentes o errores inesperados.

3.3 Constructor

No se define un constructor.

3.4 Atributos

No se definen atributos.

3.5 Métodos

crearCarpetas

- **public static void crearCarpetas(String[] carpetas)**

Se crea un conjunto de carpetas especificadas en un array de cadenas. Si alguna carpeta ya existe, se omite la creación. Si ocurre un error inesperado, se registra en el sistema de log.

hayContenidoEnDirectorio

- **public static boolean hayContenidoEnDirectorio(String rutaDirectorio)**

Se verifica si un directorio especificado contiene archivos o subdirectorios. Si el directorio no existe, se registra el error correspondiente. Se devuelve:
True si el directorio contiene archivos o subdirectorios.
False si está vacío o no existe.

mostrarMenuConArchivos

- **public static String mostrarMenuConArchivos(String rutaDirectorio)**

Se muestra un menú con los nombres de los archivos en el directorio especificado y permite al usuario seleccionar uno mediante InputHelper. Si no hay archivos, se registra el error correspondiente. Se devuelve:
El nombre del archivo seleccionado sin su extensión o null si no se encuentra ningún archivo o si ocurre un error.

obtenerArchivosEnDirectorio

- **public static String[] obtenerArchivosEnDirectorio(String rutaDirectorio)**

Se devuelve un arreglo con los nombres de los archivos en un directorio especificado. Si el directorio no existe o está vacío, se registra un error y se devuelve un arreglo vacío.
Se devuelve:
Un arreglo de cadenas con los nombres de los archivos encontrados.
Un arreglo vacío si no se encuentra ningún archivo o el directorio es inválido.

4. Clase Logger

4.1 Descripción

Se define la clase Logger como una herramienta para gestionar la escritura de registros en archivos de texto. Se permite registrar mensajes de actividad y errores, asegurando que todos los eventos relevantes del sistema queden registrados en archivos separados. La clase utiliza un patrón Singleton para garantizar que solo exista una instancia activa de Logger.

4.2 Funcionalidades

1. Se registra información general del sistema en un archivo de log.
2. Se registra información de errores en un archivo de errores específico.
3. Se maneja la apertura, escritura y cierre de archivos de manera controlada.

4.3 Constructor

- **private Logger(String logFileName)**

  Se define un constructor privado para la clase Logger, siguiendo el patrón de diseño Singleton, permitiendo la creación de una única instancia de la clase. El constructor se encarga de inicializar los archivos de registro principales, incluyendo el archivo de log general y el archivo de errores, utilizando los nombres de archivo proporcionados. Si ocurre una excepción durante el proceso de inicialización, se registra un mensaje de error en el archivo de errores del simulador.
  Parámetros:
  logFileName: Se recibe una cadena de texto que representa el nombre del archivo de registro principal, al cual se le agrega la extensión .log y se guarda en la carpeta logs/.

  Iniciaciones de atributo:

  logFile: Se crea un archivo de log principal en la ruta logs/logFileName.log.

  logWriter: Se inicializa un objeto BufferedWriter para escribir en el archivo de log principal, permitiendo registrar eventos importantes durante la ejecución del programa.

  errorWriter: Se inicializa un objeto BufferedWriter para registrar errores en el archivo de errores definido en la clase Simulador.

4.4 Atributos

- **private static Logger instance:** Se almacena la única instancia activa de la clase Logger (Singleton).
- **private BufferedWriter logWriter**: Se utiliza para escribir mensajes generales en el archivo de log principal.

- **private BufferedWriter errorWriter:** Se utiliza para escribir mensajes de error en el archivo de log de errores.

4.5 Métodos

log

- **public void log(String message)**

Se escribe un mensaje general en el archivo de log principal, añadiendo una marca de tiempo para indicar el momento en que se registró el mensaje.

logError

- **public void logError(String errorMessage)**

Se escribe un mensaje de error en el archivo de log de errores, añadiendo una marca de tiempo. Si ocurre un error durante la escritura, se registra un mensaje adicional.

close

- **public void close()**

Se cierran los archivos de log y errores de forma segura para liberar recursos del sistema. Si ocurre un error durante el cierre, se registra el error correspondiente.

5. Clase Transcriptor

5.1 Descripción

Se define la clase Transcriptor como una herramienta para registrar transcripciones detalladas de acciones realizadas en el sistema. Utiliza un patrón Singleton para garantizar que solo exista una instancia activa. Los registros se almacenan en un archivo de texto, permitiendo un seguimiento cronológico de las operaciones.

5.2 Funcionalidades

1. Se permite registrar mensajes detallados en un archivo de transcripción.
2. Se maneja la apertura, escritura y cierre de archivos de manera segura.
3. Se gestionan errores durante la creación, escritura o cierre del archivo de transcripción.

5.3 Constructor

- **private Transcriptor(String nombrePartida)**

  Se crea un constructor privado para la clase Transcriptor que se usa para escribir los eventos de una partida en un archivo de texto. El archivo se guarda en la carpeta transcripciones/ con el nombre de la partida y la extensión .tr. Si ocurre un error al intentar crear el archivo, se muestra un mensaje de error en la consola.

  Parámetros:
  nombrePartida: Es el nombre que se usará para crear el archivo de transcripción.

5.4 Atributos

- **private static Transcriptor instancia:** Se almacena la única instancia activa de la clase Transcriptor (Singleton).
- **private BufferedWriter writer:** Se utiliza para escribir mensajes en el archivo de transcripción.

5.5 Métodos

transcribir

- **public void transcribir(String mensaje)**

Se registra un mensaje detallado en el archivo de transcripción, añadiendo una nueva línea tras cada mensaje. Si ocurre un error durante la escritura, se muestra un mensaje de error en la consola.

close

- **public void close()**

Se cierra el archivo de transcripción para liberar recursos del sistema. Si ocurre un error durante el cierre, se muestra un mensaje de error en la consola.

6. Clase Transcriptor

6.1 Descripción

Se define la clase InputHelper para gestionar la entrada de datos desde la consola de manera segura y validada. Se permite leer cadenas, números enteros y seleccionar números dentro de un rango específico, asegurando que las entradas cumplan con los formatos esperados.

6.2 Funcionalidades

1. Se permite ingresar cadenas alfanuméricas, verificando que no estén vacías y que no contengan caracteres especiales.
2. Se permite ingresar números enteros válidos, mostrando mensajes de error si se ingresan valores no numéricos.
3. Se solicita un número dentro de un rango definido por valores mínimo y máximo, con validación automática.
4. Se cierra el Scanner utilizado para liberar los recursos del sistema.

6.3 Constructor

- No se define un constructor.

6.4 Atributos

- **private static final Scanner scanner**: Se utiliza para gestionar las entradas desde la consola de manera eficiente y reutilizable en toda la clase.

6.5 Métodos

readString

- **public static String readString(String prompt)**

Se lee una cadena desde la consola que no esté vacía y no contenga caracteres especiales. Si la entrada no es válida, se muestra un mensaje de error y se solicita una nueva entrada hasta que sea válida, devuelve la cadena inglesada.

readInt

- **public static int readInt(String prompt)**

Se solicita al usuario un número entero desde la consola. Si la entrada no es válida (vacía o no numérica), se muestra un mensaje de error y se solicita un nuevo número, devuelve el número entero ingresado.

solicitarNumero

- **public static int solicitarNumero(int min, int max)**

Se solicita al usuario un número entero dentro de un rango específico definido por valores mínimo y máximo. Si el número ingresado no está dentro del rango, se muestra un mensaje y se vuelve a solicitar.
Se devuelve el número entero válido dentro del rango especificado.

close

- **public static void close()**

Se cierra el Scanner utilizado en la clase para liberar recursos del sistema. Si el Scanner ya ha sido cerrado, se registra un error utilizando el sistema de registro de errores del Simulador.
No devuelve nada

7. Clase MenuHelper

7.1 Descripción

Se define la clase MenuHelper para facilitar la creación y visualización de menús en la consola. Se permite mostrar menús simples con opciones numeradas, incluyendo una opción especial para cancelar la operación.

7.2 Funcionalidades

1. Se muestra una lista de opciones numeradas en la consola para que el usuario seleccione una opción.
2. Se incluye una opción adicional para permitir cancelar operaciones y regresar a un menú anterior.

7.3 Constructor

- No se define un constructor.

7.4 Atributos

- No se definen atributos en esta clase.

7.5 Métodos

mostrarMenuCancelar

- **public static void mostrarMenuCancelar(String[] opciones)**

Se muestra un menú numerado en la consola según un array de opciones proporcionado. Se agrega automáticamente una opción 0. Cancelar, permitiendo al usuario cancelar la operación. No devuelve ningún valor.

mostrarMenu

- **public static void mostrarMenu(String[] opciones)**

Se muestra un menú numerado en la consola según un array de opciones proporcionado. No se incluye una opción de cancelación. No devuelve ningún valor.

8. Clase abstracta Carnivoro

8.1 Descripción

Se define la clase abstracta Carnivoro, que extiende de la clase Pez. Se utiliza para representar peces con una dieta basada en comida animal. Se implementa un método de alimentación que permite consumir comida animal local o, en su defecto, obtenerla del almacén central, asegurando que el pez se mantenga alimentado correctamente.

8.2 Funcionalidades

1. Se permite alimentar a los peces utilizando comida animal disponible localmente o desde el almacén central.
2. Se actualiza el estado de alimentación del pez dependiendo de si logró consumir comida o no.

8.3 Constructor

- **public Carnivoro(boolean sexo, PecesDatos datos)**

  Se define el constructor de la clase Carnivoro, el cual inicializa un pez con características específicas heredadas de su clase base. Se utiliza el método super() para llamar al constructor de la clase padre, permitiendo establecer los atributos iniciales del pez.

  Parámetros:

  sexo: Se define el sexo del pez, donde true representa un sexo y false representa el otro según la implementación específica.

  datos: Se recibe un objeto de tipo PecesDatos, el cual contiene información detallada relacionada con las características y propiedades del pez, como su tamaño, peso, hábitat y requerimientos alimenticios.

8.4 Atributos

- No se definen atributos propios en esta clase

8.5 Métodos

alimentar

- **@Override public int alimentar(int cantidadComidaAnimal, int cantidadComidaVegetal)**

Se implementa el método alimentar, que permite alimentar al pez con comida animal disponible localmente o desde el almacén central. Si hay comida disponible localmente, se consume una unidad y se actualiza el estado del pez a alimentado = true. Si no hay comida

local, se intenta obtener una unidad del almacén central. Si no se encuentra comida, el pez permanece sin alimentar.Devuelve 1 Si el pez consume comida local, y devuelve 0 si consume comida del almacen central o no hay comida disponible.

9. Clase abstracta Carnivoro

9.1 Descripción

Se define la clase abstracta CarnivoroActivo, que extiende de la clase Pez. Se utiliza para representar peces carnívoros con un comportamiento activo, lo que implica que pueden consumir más comida de lo habitual. Se utiliza un generador aleatorio para simular el comportamiento activo, haciendo que el pez pueda requerir una o dos unidades de comida animal según su actividad.

9.2 Funcionalidades

1. Se permite alimentar a los peces según su comportamiento activo, que es determinado aleatoriamente.
2. Se incrementa el consumo de comida animal según el nivel de actividad del pez. Si el pez está activo, consume dos unidades; de lo contrario, consume una.
3. Se actualiza el estado del pez dependiendo de si logró consumir comida o no, asegurando la correcta gestión de recursos.

9.3 Constructor

- **public CarnivoroActivo(boolean sexo, PecesDatos datos)**

  Se define el constructor de la clase CarnivoroActivo, el cual inicializa un pez carnívoro con comportamiento activo, heredando las propiedades de su clase base mediante la invocación del constructor de la superclase usando super(). Este constructor permite establecer los atributos principales del pez desde el momento de su creación.

  Parámetros:

  sexo: Se define el sexo del pez, donde true representa un sexo y false representa el otro según la implementación interna.

  datos: Se recibe un objeto de tipo PecesDatos, que proporciona información esencial sobre el pez, incluyendo características como tamaño, peso, hábitat y comportamientos específicos.

9.4 Atributos

- **Random rand**: Se define un generador de números aleatorios para simular el comportamiento activo del pez.

9.5 Métodos

alimentar

- **@Override public int alimentar(int cantidadComidaAnimal, int cantidadComidaVegetal)**

Se implementa el método alimentar, verificando el nivel de actividad del pez usando un generador aleatorio. Si el pez está activo (probabilidad del 50%), intenta consumir dos unidades de comida animal. Si la comida local no es suficiente, intenta consumir del almacén central. Si no está activo, consume una unidad en condiciones similares. Si no hay comida disponible, permanece sin alimentar. Se devuelve:

2: Si el pez está activo y consume dos unidades de comida animal.
1: Si el pez no está activo y consume una unidad de comida animal.
0: Si no se consume comida por falta de recursos.

10. Clase abstracta Filtrador

10.1 Descripción

Se define la clase abstracta Filtrador, que extiende de la clase Pez. Se utiliza para representar peces que se alimentan exclusivamente de comida vegetal. Se implementa un método de alimentación que permite consumir comida vegetal disponible localmente o desde el almacén central en caso de escasez. El comportamiento de alimentación se controla mediante un generador aleatorio, simulando la búsqueda de alimento en su entorno.

10.2 Funcionalidades

1. Se permite alimentar a los peces utilizando comida vegetal disponible localmente o desde el almacén central.
2. Se utiliza un generador aleatorio para determinar cuándo el pez intenta alimentarse.
3. Se actualiza el estado del pez según su nivel de alimentación, indicando si logró consumir comida vegetal o permaneció sin alimentar.

10.3 Constructor

- **public Filtrador(boolean sexo, PecesDatos datos)**

  Se define el constructor de la clase Filtrador, el cual inicializa un pez con comportamiento filtrador, heredando sus propiedades de la clase base mediante la invocación del constructor de la superclase usando super(). Esto permite establecer atributos esenciales relacionados con el pez desde su creación.

  Parámetros:

  sexo: Se define el sexo del pez, donde true representa un sexo y false representa el otro según la lógica implementada.

  datos: Se recibe un objeto de tipo PecesDatos, que contiene información relevante sobre el pez, como su tamaño, peso, tipo de alimento y características específicas de su comportamiento filtrador.

10.4 Atributos

- **Random rand**: Se define un generador de números aleatorios para simular el comportamiento de alimentación del pez filtrador.

10.5 Métodos

alimentar

- **@Override public int alimentar(int cantidadComidaAnimal, int cantidadComidaVegetal)**

Se implementa el método alimentar, que permite alimentar al pez usando comida vegetal disponible localmente o desde el almacén central. Si hay comida vegetal local, se consume una unidad. Si no hay comida local pero existe comida en el almacén central, se consume una unidad desde allí. Si no se encuentra comida en ninguno de los dos lugares, el pez permanece sin alimentar. El proceso de alimentación ocurre con una probabilidad del 50%. Si no se encuentra comida, el pez permanece sin alimentar. Se devuelve:

1: Si el pez consume comida vegetal local.
0: Si consume comida vegetal del almacén central o no consume comida por falta de recursos.

11. Clase abstracta Omnivoro

11.1 Descripción

Se define la clase abstracta Omnivoro, que extiende de la clase Pez. Se utiliza para representar peces con una dieta mixta, que consumen tanto comida animal como vegetal. Se implementa un método de alimentación que permite consumir cualquiera de los dos tipos de comida, dependiendo de la disponibilidad en el entorno local o en el almacén central. Se utiliza un generador aleatorio para simular la probabilidad de que el pez se alimente.

11.2 Funcionalidades

1. Se permite alimentar al pez con comida animal o vegetal, según la disponibilidad en el entorno local o el almacén central.
2. Se da prioridad a la comida disponible localmente y, en caso de no haber suficiente, se consume comida del almacén central.
3. Se actualiza el estado de alimentación según los recursos consumidos, indicando si el pez logró alimentarse correctamente.

11.3 Constructor

- **public Omnivoro(boolean sexo, PecesDatos datos)**

  Se define el constructor de la clase Omnivoro, el cual inicializa un pez con una dieta omnívora, heredando sus propiedades principales desde la clase base mediante la invocación del constructor de la superclase usando super(). Esto permite establecer atributos esenciales relacionados con el pez en el momento de su creación.

  Parámetros:

  sexo: Se define el sexo del pez, donde true representa un sexo y false representa el otro según la implementación interna.

  datos: Se recibe un objeto de tipo PecesDatos, que contiene información fundamental sobre el pez, incluyendo características como su tamaño, peso, hábitat y sus requerimientos alimenticios específicos.

11.4 Atributos

- **Random rand** Se define un generador de números aleatorios para determinar la probabilidad de que el pez se alimente en cada ciclo.

11.5 Métodos

alimentar

- **@Override public int alimentar(int cantidadComidaAnimal, int cantidadComidaVegetal)**

Se implementa el método alimentar, que permite consumir comida animal o vegetal según la disponibilidad. Con una probabilidad del 75%, el pez intenta alimentarse. Si hay comida disponible localmente, se consume una unidad de cualquiera de los dos tipos. Si no hay comida local, se intenta consumir comida del almacén central, priorizando el tipo de comida con mayor cantidad disponible. Si ninguno de los dos lugares dispone de comida, el pez permanece sin alimentar. Se devuelve:

1: Si el pez consume comida local, ya sea animal o vegetal.
0: Si consume comida del almacén central o no consume comida por falta de recursos.

12. Clase Dorada

12.1 Descripción

Se define la clase Dorada como una subclase de Omnivoro, representando un pez que consume tanto comida animal como vegetal. Se establece como un pez omnívoro con propiedades específicas definidas en la clase AlmacenPropiedades. Se permite crear nuevas instancias y realizar clonaciones según el sexo especificado.

12.2 Funcionalidades

1. Se hereda el comportamiento de alimentación de la clase Omnivoro, permitiendo consumir ambos tipos de comida según la disponibilidad.
2. Se permite crear instancias del pez con un sexo específico y generar copias exactas mediante un método de clonación.

12.3 Constructor

- **public Dorada(boolean sexo)**

  Se define el constructor de la clase Dorada, el cual inicializa un pez de tipo dorada estableciendo sus propiedades específicas mediante la invocación del constructor de la superclase usando super(). Se utiliza una constante predefinida AlmacenPropiedades.DORADA para asociar automáticamente las características específicas del pez dorada desde un almacén de datos centralizado.

  Parámetros:

  sexo: Se define el sexo del pez, (true para macho, false para hembra)

12.4 Atributos

- No se define ningún atributo.

12.5 Métodos

clonar

- **@Override public Dorada clonar(boolean nuevoSexo)**

Se crea una nueva instancia del pez Dorada con el sexo especificado. Este método permite replicar la instancia actual con un sexo diferente.
Devuelve: Una nueva instancia de Dorada con el sexo especificado.

13. Clase abstracta Dorada

13.1 Descripción

Se define la clase SalmonAtlantico como una subclase de Carnivoro, representando un pez que se alimenta exclusivamente de comida animal. Se inicializa con propiedades específicas definidas en la clase AlmacenPropiedades. Se permite crear instancias del pez y realizar clonaciones con diferentes sexos.

13.2 Funcionalidades

1. Se hereda el comportamiento de alimentación de la clase Carnivoro, permitiendo consumir solo comida animal disponible localmente o desde el almacén central.
2. Se permite crear nuevas instancias del pez con un sexo específico y generar copias exactas mediante un método de clonación.

13.3 Constructor

- **public SalmonAtlantico(boolean sexo)**

  Se define el constructor de la clase SalmonAtlantico, el cual inicializa un pez de tipo salmón atlántico estableciendo sus propiedades específicas mediante la invocación del constructor de la superclase usando super(). Se utiliza una constante predefinida AlmacenPropiedades.SALMON_ATLANTICO para asignar automáticamente las características propias de esta especie desde un almacén central de datos.

  Parámetros:

  sexo: Se define el sexo del pez, (true para macho, false para hembra)

13.4 Atributos

- No se define ningún atributo.

13.5 Métodos

clonar

- **@Override public SalmonAtlantico clonar(boolean nuevoSexo)**

Se crea una nueva instancia de SalmonAtlantico con el sexo especificado. Este método permite replicar la instancia actual con un sexo diferente.
Devuelve: Una nueva instancia de SalmonAtlantico con el sexo especificado.

14. Clase TruchaArcoiris

14.1 Descripción

Se define la clase TruchaArcoiris como una subclase de Carnivoro, representando un pez que se alimenta exclusivamente de comida animal. Se configura utilizando propiedades específicas definidas en la clase AlmacenPropiedades. Se permite crear instancias del pez y realizar clonaciones especificando un nuevo sexo.

14.2 Funcionalidades

1. Se hereda el comportamiento de alimentación de la clase Carnivoro, permitiendo consumir comida animal local o desde el almacén central.
2. Se permite crear nuevas instancias del pez con un sexo definido y generar copias exactas mediante un método de clonación.

14.3 Constructor

- **public TruchaArcoiris(boolean sexo)**

  Se define el constructor de la clase TruchaArcoiris, el cual inicializa un pez de tipo trucha arcoíris estableciendo sus propiedades específicas mediante la invocación del constructor de la superclase usando super(). Se utiliza una constante predefinida AlmacenPropiedades.TRUCHA_ARCOIRIS para asignar automáticamente las características propias de esta especie desde un almacén central de datos.

  Parámetros:

  sexo: Se define el sexo del pez, (true para macho, false para hembra)

14.4 Atributos

- No se define ningún atributo.

14.5 Métodos

clonar

- **@Override public TruchaArcoiris clonar(boolean nuevoSexo)**

Se crea una nueva instancia de TruchaArcoiris con el sexo especificado. Este método permite replicar la instancia actual del pez con un sexo diferente.
Devuelve: Una nueva instancia de TruchaArcoiris con el sexo especificado.

15. Clase ArenqueDelAtlantico

15.1 Descripción

Se define la clase ArenqueDelAtlantico como una subclase de Filtrador, representando un pez que se alimenta exclusivamente de comida vegetal. Se configura utilizando propiedades específicas establecidas en la clase AlmacenPropiedades. Se permite crear instancias del pez y realizar clonaciones especificando un nuevo sexo.

15.2 Funcionalidades

1. Se hereda el comportamiento de alimentación de la clase Filtrador, permitiendo consumir comida vegetal disponible localmente o desde el almacén central.
2. Se permite crear nuevas instancias del pez con un sexo específico y generar copias exactas mediante el método de clonación.

15.3 Constructor

- **public ArenqueDelAtlantico(boolean sexo)**

  Se define el constructor de la clase ArenqueDelAtlantico, el cual inicializa un pez de tipo arenque del Atlántico estableciendo sus propiedades específicas mediante la invocación del constructor de la superclase usando super(). Se utiliza una constante predefinida AlmacenPropiedades.ARENQUE_ATLANTICO para asignar automáticamente las características propias de esta especie desde un almacén central de datos.

  Parámetros:

  sexo: Se define el sexo del pez, (true para macho, false para hembra)

15.4 Atributos

- No se define ningún atributo.

15.5 Métodos

clonar

- **@Override public ArenqueDelAtlantico clonar(boolean nuevoSexo)**

Se crea una nueva instancia de ArenqueDelAtlantico con el sexo especificado. Este método permite replicar la instancia actual con un sexo diferente.
Devuelve: Una nueva instancia de ArenqueDelAtlantico con el sexo especificado.

16. Clase Besugo

16.1 Descripción

Se define la clase Besugo como una subclase de Carnivoro, representando un pez que se alimenta exclusivamente de comida animal. Se configura utilizando propiedades específicas establecidas en la clase AlmacenPropiedades. Se permite crear instancias del pez y realizar clonaciones especificando un nuevo sexo.

16.2 Funcionalidades

1. Se hereda el comportamiento de alimentación de la clase Carnivoro, permitiendo consumir comida animal disponible localmente o desde el almacén central.
2. Se permite crear nuevas instancias del pez con un sexo definido y generar copias exactas mediante el método de clonación.

16.3 Constructor

- **public Besugo(boolean sexo)**

  Se define el constructor de la clase Besugo, el cual inicializa un pez de tipo besugo estableciendo sus propiedades específicas mediante la invocación del constructor de la superclase usando super(). Se utiliza una constante predefinida AlmacenPropiedades.BESUGO para asignar automáticamente las características propias de esta especie desde un almacén central de datos.

  Parámetros:

  sexo: Se define el sexo del pez, (true para macho, false para hembra)

16.4 Atributos

- No se define ningún atributo.

16.5 Métodos

clonar

- **@Override public Besugo clonar(boolean nuevoSexo)**

Se crea una nueva instancia de Besugo con el sexo especificado. Este método permite replicar la instancia actual con un sexo diferente.
Devuelve: Una nueva instancia de Besugo con el sexo especificado.

17. Clase Besugo

17.1 Descripción

Se define la clase LenguadoEuropeo como una subclase de Carnivoro, representando un pez que se alimenta exclusivamente de comida animal. Se configura utilizando propiedades específicas establecidas en la clase AlmacenPropiedades. Se permite crear instancias del pez y realizar clonaciones especificando un nuevo sexo.

17.2 Funcionalidades

1. Se hereda el comportamiento de alimentación de la clase Carnivoro, permitiendo consumir comida animal disponible localmente o desde el almacén central.
2. Se permite crear nuevas instancias del pez con un sexo definido y generar copias exactas mediante el método de clonación.

17.3 Constructor

- **public LenguadoEuropeo(boolean sexo)**

  Se define el constructor de la clase LenguadoEuropeo, el cual inicializa un pez de tipo lenguado europeo estableciendo sus propiedades específicas mediante la invocación del constructor de la superclase usando super(). Se utiliza una constante predefinida AlmacenPropiedades.LENGUADO_EUROPEO para asignar automáticamente las características particulares de esta especie desde un almacén central de datos.

  Parámetros:

  sexo: Se define el sexo del pez, (true para macho, false para hembra)

17.4 Atributos

- No se define ningún atributo.

17.5 Métodos

clonar

- **@Override public LenguadoEuropeo clonar(boolean nuevoSexo)**

Se crea una nueva instancia de LenguadoEuropeo con el sexo especificado. Este método permite replicar la instancia actual con un sexo diferente.
Devuelve: Una nueva instancia de LenguadoEuropeo con el sexo especificado.

18. Clase Robalo

18.1 Descripción

Se define la clase Robalo como una subclase de Carnivoro, representando un pez que se alimenta exclusivamente de comida animal. Se configura utilizando propiedades específicas establecidas en la clase AlmacenPropiedades. Se permite crear instancias del pez y realizar clonaciones especificando un nuevo sexo.

18.2 Funcionalidades

1. Se hereda el comportamiento de alimentación de la clase Carnivoro, permitiendo consumir comida animal disponible localmente o desde el almacén central.
2. Se permite crear nuevas instancias del pez con un sexo definido y generar copias exactas mediante el método de clonación.

18.3 Constructor

- **public Robalo(boolean sexo)**

  Se define el constructor de la clase Robalo, el cual inicializa un pez de tipo robalo estableciendo sus propiedades específicas mediante la invocación del constructor de la superclase usando super(). Se utiliza una constante predefinida AlmacenPropiedades.ROBALO para asignar automáticamente las características particulares de esta especie desde un almacén central de datos.

  Parámetros:

  sexo: Se define el sexo del pez, (true para macho, false para hembra)

18.4 Atributos

- No se define ningún atributo.

18.5 Métodos

clonar

- **@Override public Robalo clonar(boolean nuevoSexo)**

Se crea una nueva instancia de Robalo con el sexo especificado. Este método permite replicar la instancia actual con un sexo diferente.
Devuelve: Una nueva instancia de Robalo con el sexo especificado.

19. Clase CarpaPlateada

19.1 Descripción

Se define la clase CarpaPlateada como una subclase de Filtrador, representando un pez que se alimenta exclusivamente de comida vegetal. Se configura utilizando propiedades específicas establecidas en la clase AlmacenPropiedades. Se permite crear instancias del pez y realizar clonaciones especificando un nuevo sexo.

19.2 Funcionalidades

1. Se hereda el comportamiento de alimentación de la clase Filtrador, permitiendo consumir comida vegetal disponible localmente o desde el almacén central.
2. Se permite crear nuevas instancias del pez con un sexo definido y generar copias exactas mediante el método de clonación.

19.3 Constructor

- **public CarpaPlateada(boolean sexo)**

  Se define el constructor de la clase CarpaPlateada, el cual inicializa un pez de tipo carpa plateada estableciendo sus propiedades específicas mediante la invocación del constructor de la superclase usando super(). Se utiliza una constante predefinida AlmacenPropiedades.CARPA_PLATEADA para asignar automáticamente las características particulares de esta especie desde un almacén central de datos.

  Parámetros:

  sexo: Se define el sexo del pez, (true para macho, false para hembra)

19.4 Atributos

- No se define ningún atributo.

19.5 Métodos

clonar

- **@Override public CarpaPlateada clonar(boolean nuevoSexo)**

Se crea una nueva instancia de CarpaPlateada con el sexo especificado. Este método permite replicar la instancia actual con un sexo diferente.
Devuelve: Una nueva instancia de CarpaPlateada con el sexo especificado.

20. Clase CarpaPlateada

20.1 Descripción

Se define la clase Pejerrey como una subclase de Carnivoro, representando un pez que se alimenta exclusivamente de comida animal. Se configura utilizando propiedades específicas establecidas en la clase AlmacenPropiedades. Se permite crear instancias del pez y realizar clonaciones especificando un nuevo sexo.

20.2 Funcionalidades

1. Se hereda el comportamiento de alimentación de la clase Carnivoro, permitiendo consumir comida animal disponible localmente o desde el almacén central.
2. Se permite crear nuevas instancias del pez con un sexo definido y generar copias exactas mediante el método de clonación.

20.3 Constructor

- **public Pejerrey(boolean sexo)**

  Se define el constructor de la clase Pejerrey, el cual inicializa un pez de tipo pejerrey estableciendo sus propiedades específicas mediante la invocación del constructor de la superclase usando super(). Se utiliza una constante predefinida AlmacenPropiedades.PEJERREY para asignar automáticamente las características particulares de esta especie desde un almacén central de datos.

  Parámetros:

  sexo: Se define el sexo del pez, (true para macho, false para hembra)

20.4 Atributos

- No se define ningún atributo.

20.5 Métodos

clonar

- **@Override public Pejerrey clonar(boolean nuevoSexo)**

Se crea una nueva instancia de Pejerrey con el sexo especificado. Este método permite replicar la instancia actual con un sexo diferente.
Devuelve: Una nueva instancia de Pejerrey con el sexo especificado.

21. Clase SalmonChinook

21.1 Descripción

Se define la clase SalmonChinook como una subclase de Carnivoro, representando un pez que se alimenta exclusivamente de comida animal. Se configura utilizando propiedades específicas establecidas en la clase AlmacenPropiedades. Se permite crear instancias del pez y realizar clonaciones especificando un nuevo sexo.

21.2 Funcionalidades

1. Se hereda el comportamiento de alimentación de la clase Carnivoro, permitiendo consumir comida animal disponible localmente o desde el almacén central.
2. Se permite crear nuevas instancias del pez con un sexo definido y generar copias exactas mediante el método de clonación.

21.3 Constructor

- **public SalmonChinook(boolean sexo)**

  Se define el constructor de la clase SalmonChinook, el cual inicializa un pez de tipo salmón Chinook estableciendo sus propiedades específicas mediante la invocación del constructor de la superclase usando super(). Se utiliza una constante predefinida AlmacenPropiedades.SALMON_CHINOOK para asignar automáticamente las características particulares de esta especie desde un almacén central de datos.

  Parámetros:

  sexo: Se define el sexo del pez, (true para macho, false para hembra)

21.4 Atributos

- No se define ningún atributo.

21.5 Métodos

clonar

- **@Override public SalmonChinook clonar(boolean nuevoSexo)**

Se crea una nueva instancia de SalmonChinook con el sexo especificado. Este método permite replicar la instancia actual con un sexo diferente.
Devuelve: Una nueva instancia de SalmonChinook con el sexo especificado.

22. Clase TilapiaDelNilo

22.1 Descripción

Se define la clase TilapiaDelNilo como una subclase de Filtrador, representando un pez que se alimenta exclusivamente de comida vegetal. Se configura utilizando propiedades específicas establecidas en la clase AlmacenPropiedades. Se permite crear instancias del pez y realizar clonaciones especificando un nuevo sexo.

22.2 Funcionalidades

1. Se hereda el comportamiento de alimentación de la clase Filtrador, permitiendo consumir comida vegetal disponible localmente o desde el almacén central.
2. Se permite crear nuevas instancias del pez con un sexo definido y generar copias exactas mediante el método de clonación.

22.3 Constructor

- **public TilapiaDelNilo(boolean sexo)**

  Se define el constructor de la clase TilapiaDelNilo, el cual inicializa un pez de tipo tilapia del Nilo estableciendo sus propiedades específicas mediante la invocación del constructor de la superclase usando super(). Se utiliza una constante predefinida AlmacenPropiedades.TILAPIA_NILO para asignar automáticamente las características particulares de esta especie desde un almacén central de datos.

  Parámetros:

  sexo: Se define el sexo del pez, (true para macho, false para hembra)

22.4 Atributos

- No se define ningún atributo.

22.5 Métodos

clonar

- **@Override public TilapiaDelNilo clonar(boolean nuevoSexo)**

Se crea una nueva instancia de TilapiaDelNilo con el sexo especificado. Este método permite replicar la instancia actual con un sexo diferente.
Devuelve: Una nueva instancia de TilapiaDelNilo con el sexo especificado.

23. Clase Pez

23.1 Descripción

Se define la clase abstracta Pez como la clase base para todos los tipos de peces en la simulación. Se gestiona información esencial como el nombre, edad, sexo, estado de vida, alimentación, fertilidad y madurez. También se permite realizar operaciones como el crecimiento diario, la alimentación y el reinicio del estado del pez. Se implementan métodos para mostrar y devolver información relevante sobre los peces.

23.2 Funcionalidades

1. Se permite establecer y consultar atributos clave como nombre, edad, sexo, fertilidad y estado de vida del pez.
2. Se realiza el crecimiento diario del pez, actualizando su edad, fertilidad y estado de madurez según sus propiedades biológicas.
3. Se permite reiniciar todos los atributos a sus valores iniciales para reutilizar objetos existentes.
4. Se define un método abstracto para crear copias exactas de peces con un nuevo sexo especificado.

23.3 Constructor

- **public Pez(boolean sexo, PecesDatos datos)**

  Se define el constructor principal de la clase Pez, encargado de inicializar un pez con los datos específicos proporcionados mediante un objeto de tipo PecesDatos. Se configuran atributos clave relacionados con su especie, nombre científico, sexo y ciclo de vida utilizando la información contenida en el objeto recibido.

  Parámetros:

  sexo: Se define el sexo del pez, (true para macho, false para hembra)

  datos: Se recibe un objeto de tipo PecesDatos, el cual contiene información detallada sobre la especie del pez, incluyendo su nombre común, nombre científico, ciclo de vida y características específicas.

  Inicialización de atributos:

  nombre: Se establece utilizando el nombre común obtenido de datos.getNombre().

  nombreCientifico: Se asigna utilizando el nombre científico obtenido de datos.getCientifico().

  sexo: Se asigna directamente a partir del parámetro recibido.

datos: Se guarda el objeto PecesDatos completo para acceder a sus propiedades específicas.

ciclo: Se inicializa utilizando datos.getCiclo(), definiendo el ciclo de vida del pez según su especie.

23.4 Atributos

- **private final String nombre:** Se almacena el nombre común del pez.
- **private final String nombreCientifico:** Se registra el nombre científico del pez.
- **private int edad:** Se almacena la edad actual del pez en días.
- **private boolean sexo:** Se define el sexo del pez (true para macho, false para hembra).
- **private boolean fertil:** Se indica si el pez es fértil.
- **private boolean vivo:** Se indica si el pez está vivo.
- **protected boolean alimentado:** Se indica si el pez ha sido alimentado.
- **private boolean maduro:** Se almacena el nombre común del pez.
- **private final String nombre:** Se indica si el pez ha alcanzado la madurez biológica.
- **protected int ciclo**: Se gestiona el ciclo reproductivo del pez.
- **private PecesDatos datos:** Se almacenan propiedades específicas del pez, como su ciclo de reproducción, edad de madurez y más.

23.5 Métodos

showStatus

- **public void showStatus()**

Se muestra el estado actual del pez, incluyendo edad, sexo, estado de vida, fertilidad, madurez y estado de alimentación.

grow

- **public void grow()**

Se realiza el crecimiento diario del pez. Si no está alimentado, existe una probabilidad de muerte. Si está vivo, su edad se incrementa y se verifican las condiciones de fertilidad y madurez según sus propiedades biológicas.

reset

- **public void reset()**

Se restablecen todos los atributos del pez a sus valores iniciales, permitiendo reutilizar la instancia en simulaciones futuras.

clonar

- public abstract Pez clonar(boolean nuevoSexo)

  Se define un método abstracto para crear una copia exacta del pez con el nuevo sexo especificado.
  Se devuelve una nueva instancia de la subclase correspondiente.

alimentar

- public abstract int alimentar(int cantidadComidaAnimal, int cantidadComidaVegetal)

  Se define un método abstracto para alimentar al pez, especificando la cantidad de comida animal y vegetal disponible.
  Se devuelve un int que indica la cantidad de comida consumida.

24. Clase Piscifactoria

24.1 Descripción

Se define la clase abstracta Piscifactoria, que representa una instalación de cría y mantenimiento de peces. Se gestiona una lista de tanques, la cantidad de comida disponible y la capacidad máxima de los tanques. Se incluyen métodos para mostrar el estado general de la piscifactoría, alimentar a los peces y realizar simulaciones diarias para actualizar su ciclo de vida.

24.2 Funcionalidades

1. Se permite agregar y manejar tanques que contienen peces.
2. Se controla la cantidad de comida animal y vegetal almacenada en la piscifactoría.
3. Se realiza la simulación del crecimiento y desarrollo de los peces, gestionando su alimentación, fertilidad y supervivencia.
4. Se permite mostrar información detallada sobre la piscifactoría, los tanques y los peces que contiene.

24.3 Constructor

- **public Piscifactoria(String nombre)**

  Se define el constructor de la clase Piscifactoria, encargado de inicializar una piscifactoría asignándole un nombre específico proporcionado como parámetro. Este nombre permite identificar la piscifactoría dentro del sistema de gestión.

  Párametros:

  nombre: Se recibe una cadena de texto que representa el nombre asignado a la piscifactoría. Este nombre se utiliza para identificar de manera única la piscifactoría en el sistema.

  Inicialización de atributos:

  nombre: Se establece con el valor recibido en el parámetro nombre, permitiendo identificar la piscifactoría en registros y operaciones.

24.4 Atributos

- **protected String nombre:** Se almacena el nombre de la piscifactoría.
- **protected List<Tanque> tanques:** Se guarda la lista de tanques disponibles en la piscifactoría.
- **protected final int numeroMaximoTanques:** Se define el número máximo de tanques permitidos.

- **private int cantidadComidaAnimal:** Se registra la cantidad actual de comida animal disponible.
- **private int cantidadComidaVegetal:** Se almacena la cantidad actual de comida vegetal disponible.
- **protected int capacidadMaximaComida:** Se gestiona la capacidad máxima del almacén de comida.

24.5 Métodos

showStatus

- **public void showStatus()**

Se muestra el estado general de la piscifactoría, incluyendo el número de tanques, la ocupación, el total de peces vivos, alimentados y fértiles, así como la cantidad de comida disponible.

showTankStatus

- **public void showTankStatus()**

Se muestra el estado de cada tanque en la piscifactoría.

showFishStatus

- **public void showFishStatus(int numeroTanque)**

Se muestra información detallada de los peces dentro de un tanque específico.

showCapacity

- **public void showCapacity(int numeroTanque)**

  Se muestra la capacidad actual de un tanque determinado.

showFood

- **public void showFood()**

  Se muestra la cantidad de comida animal y vegetal disponible en la piscifactoría.

nextDay

- **public int[] nextDay()**

  Se simula el avance de un día en la piscifactoría, alimentando a los peces y actualizando su ciclo de vida. Se retorna un array con el total de peces vendidos y monedas ganadas.

upgradeFood

- public abstract void upgradeFood()

  Se define un método abstracto para aumentar la capacidad máxima del almacén de comida.

addTanque

- public abstract void addTanque()

  Se define un método abstracto para añadir un nuevo tanque a la piscifactoría.

añadirComidaAnimal

- public boolean añadirComidaAnimal(int cantidad)

  Se añade una cantidad específica de comida animal, verificando que no se exceda la capacidad máxima. Devuelve:
  True si la comida fue añadida correctamente.
  False si no se pudo añadir la comida.

añadirComidaVegetal

- public boolean añadirComidaVegetal(int cantidad)

  Se añade una cantidad específica de comida vegetal, verificando que no se exceda la capacidad máxima.Devuelve:
  True si la comida fue añadida correctamente.
  False si no se pudo añadir la comida.

alimentarPeces

- private void alimentarPeces(Tanque tanque)

  Se alimenta a los peces de un tanque, gestionando la cantidad de comida disponible y actualizando el estado de los peces.

25. Clase PiscifactoriaDeMar

25.1 Descripción

Se define la clase PiscifactoriaDeMar como una subclase de Piscifactoria, diseñada para gestionar la cría y el mantenimiento de peces de mar. Se establece una capacidad máxima de comida limitada y un sistema de mejora y expansión de tanques basado en monedas virtuales.

25.2 Funcionalidades

1. Se permite mejorar la capacidad de almacenamiento de comida hasta un límite máximo.
2. Se permite añadir nuevos tanques, con un costo creciente basado en la cantidad de tanques ya instalados.
3. Se gestiona el gasto de monedas para realizar mejoras y ampliaciones en la piscifactoría.

25.3 Constructor

- **public PiscifactoriaDeMar(String nombre)**

  Se define el constructor de la clase PiscifactoriaDeMar, encargado de inicializar una piscifactoría especializada en peces de agua salada. Se invoca el constructor de la superclase Piscifactoria para establecer el nombre de la piscifactoría. Además, se inicializa un tanque con capacidad de 100 unidades y se define la capacidad máxima de almacenamiento de comida para la piscifactoría.

  Parámetros:

  nombre: Se recibe una cadena de texto que representa el nombre asignado a la piscifactoría. Este nombre permite identificarla en el sistema de gestión de piscifactorías.

  Inicialización de Atributos:

  super(nombre): Se llama al constructor de la superclase Piscifactoria, asignando el nombre proporcionado a la piscifactoría.

  tanques.add(new Tanque(tanques.size() + 1, 100)): Se crea un nuevo tanque con un identificador único y una capacidad de 100 unidades, añadiéndolo a la lista de tanques de la piscifactoría.

  capacidadMaximaComida = 100: Se establece la capacidad máxima de almacenamiento de comida en 100 unidades, permitiendo gestionar el suministro de alimentos en la piscifactoría.

25.4 Atributos

- **private static final int COSTO_MEJORA = 200:** Define el costo en monedas para mejorar la capacidad de almacenamiento de comida.

- **private static final int INCREMENTO_CAPACIDAD = 100:** Especifica el incremento de capacidad de comida en cada mejora.
- **private static final int CAPACIDAD_MAXIMA_PERMITIDA = 1000:** Establece la capacidad máxima permitida para el almacén de comida de la piscifactoría.

25.5 Métodos

upgradeFood

- **@Override public void upgradeFood()**

Se realiza una mejora en la capacidad máxima del almacén de comida, aumentando su capacidad en INCREMENTO_CAPACIDAD hasta alcanzar CAPACIDAD_MAXIMA_PERMITIDA. Si hay monedas suficientes, se realiza la mejora; de lo contrario, se muestra un mensaje informando la falta de monedas.

addTanque

- **@Override public void addTanque()**

Se agrega un nuevo tanque a la piscifactoría, aumentando su costo según la cantidad de tanques existentes. Si hay suficientes monedas, se instala el tanque; de lo contrario, se muestra un mensaje de error.

26. Clase PiscifactoriaDeRio

26.1 Descripción

Se define la clase PiscifactoriaDeRio como una subclase de Piscifactoria, diseñada para gestionar peces de río. Se configura un límite máximo de capacidad de comida y se permite realizar mejoras mediante monedas virtuales. Se incluye un sistema de expansión de tanques basado en costos crecientes según la cantidad de tanques existentes.

26.2 Funcionalidades

1. Se permite aumentar la capacidad de almacenamiento de comida hasta un límite máximo establecido.
2. Se permite agregar nuevos tanques, con costos crecientes según la cantidad de tanques existentes.
3. Se controla el gasto de monedas para realizar mejoras en la capacidad de comida y añadir tanques.

26.3 Constructor

- **public PiscifactoriaDeRio(String nombre)**

  Se define el constructor de la clase PiscifactoriaDeRio, encargado de inicializar una piscifactoría especializada en peces de agua dulce. Se invoca el constructor de la superclase Piscifactoria para establecer el nombre de la piscifactoría. Además, se crea un tanque con una capacidad inicial de 25 unidades y se establece la capacidad máxima de almacenamiento de comida en 25 unidades.

  Parámetros:

  nombre: Se recibe una cadena de texto que representa el nombre asignado a la piscifactoría, permitiendo identificarla dentro del sistema de gestión de piscifactorías.

  Inicialización de atributos:

  super(nombre): Se llama al constructor de la superclase Piscifactoria, asignando el nombre proporcionado a la piscifactoría.

  tanques.add(new Tanque(tanques.size() + 1, 25)): Se crea un nuevo tanque con un identificador único y una capacidad de 25 unidades, añadiéndolo a la lista de tanques de la piscifactoría.

  capacidadMaximaComida = 25: Se establece la capacidad máxima de almacenamiento de comida en 25 unidades, permitiendo gestionar el suministro de alimentos en la piscifactoría.

26.4 Atributos

- **private static final int COSTO_MEJORA = 50:** Define el costo en monedas para mejorar la capacidad de almacenamiento de comida.
- **private static final int INCREMENTO_CAPACIDAD = 25:** Especifica el incremento de capacidad de comida en cada mejora.
- **private static final int CAPACIDAD_MAXIMA_PERMITIDA = 1000:** Establece la capacidad máxima permitida para el almacén de comida de la piscifactoría.

26.5 Métodos

upgradeFood

- **@Override public void upgradeFood()**

Se mejora la capacidad máxima de almacenamiento de comida en INCREMENTO_CAPACIDAD hasta alcanzar CAPACIDAD_MAXIMA_PERMITIDA. Si hay monedas suficientes, se realiza la mejora; de lo contrario, se muestra un mensaje indicando la falta de monedas.

addTanque

- **@Override public void addTanque()**

Se agrega un nuevo tanque a la piscifactoría, aumentando su costo según la cantidad de tanques existentes. Si hay monedas suficientes, se instala un nuevo tanque; de lo contrario, se muestra un mensaje informando la falta de monedas.

27. Clase Tanque

27.1 Descripción

Se define la clase Tanque, que representa un contenedor para almacenar peces con capacidades de gestión, monitoreo y reproducción. Se permite realizar operaciones de cría, alimentación y venta de peces, además de mostrar información detallada sobre su estado actual.

27.2 Funcionalidades

1. Se permite añadir peces, comprobar compatibilidad y gestionar su crecimiento, reproducción y venta.
2. Se permite mostrar el estado general del tanque y detalles específicos como ocupación, cantidad de peces vivos, alimentados, adultos y fértiles.
3. Se permite simular el crecimiento diario de los peces y registrar estadísticas relacionadas con la reproducción y venta.

27.3 Constructor

- **public Tanque(int numeroTanque, int capacidadMaxima)**

  Se define el constructor de la clase Tanque, encargado de inicializar un tanque de almacenamiento de peces y comida en la piscifactoría. Se asigna un número identificador único al tanque y se establece su capacidad máxima según los valores proporcionados como parámetros.

  numeroTanque: Se recibe un valor entero que representa el número identificador único del tanque dentro de la piscifactoría.

  capacidadMaxima: Se recibe un valor entero que indica la capacidad máxima de almacenamiento del tanque en términos de unidades de peces o comida.

  this.numeroTanque = numeroTanque: Se asigna el número de identificación proporcionado al tanque.

  this.capacidadMaxima = capacidadMaxima: Se establece la capacidad máxima del tanque según el valor recibido.

27.4 Atributos

- **private List<Pez> peces:** Se almacena la lista de peces que habitan el tanque.
- **private final int numeroTanque:** Se define el número identificador del tanque.
- **private final int capacidadMaxima:** Se establece la capacidad máxima de peces que puede contener el tanque.

27.5 Métodos

showStatus

- **public void showStatus()**

Se muestra el estado general del tanque, incluyendo ocupación, cantidad de peces vivos, alimentados, adultos, hembras, machos y fértiles.

showFishStatus

- **public void showFishStatus()**

Se muestra el estado individual de cada pez en el tanque, incluyendo su edad, estado de vida y alimentación.

showCapacity

- **public void showCapacity()**
  Se muestra la capacidad actual del tanque en porcentaje y número de peces almacenados.

nextDay

- **public int[] nextDay()**
  Se simula el avance de un día en el tanque, haciendo crecer a los peces y gestionando su reproducción y ventas. Se retorna un array con el total de peces vendidos y monedas ganadas.

reproduccion

- **public void reproduccion()**
  Se gestiona la reproducción de los peces en el tanque, generando nuevos peces si existen machos y hembras fértiles. Si se supera la capacidad máxima, se interrumpe la reproducción.

addFish

- **public boolean addFish(Pez pez)**
  Se agrega un pez al tanque, verificando compatibilidad de especie y disponibilidad de monedas. Devuelve:

True si el pez se agregó correctamente.

False si el tanque está lleno o la especie es incompatible.

sellFish

- **public int[] sellFish()**

  Se venden los peces maduros del tanque y se actualiza el saldo de monedas. Se registran estadísticas de ventas. Se devuelve:

  Un array con el número total de peces vendidos y monedas ganadas.

28. Clase Simulador

28.1 Descripción

En la clase `Simulador` se administra todas las operaciones relacionadas con la simulación de piscifactorías. Desde la creación y administración de piscifactorías hasta la gestión de peces, recursos y monedas. Además, implementa un sistema de persistencia que permite guardar y cargar el estado de la simulación.

28.2 Funcionalidades

1. Se pueden crear, mejorar y administrar piscifactorías de río y de mar.
2. Compra, venta y alimentación de peces según sus características específicas.
3. Manejo de comida con distribución automática.
4. Seguimiento del estado de la simulación y aplicación de recompensas.
5. Guardado y carga de partidas desde archivos JSON.

28.3 Constructor

- **public Simulador()**

  Se define el constructor de la clase `Simulador`, que inicializa el simulador con valores predeterminados. Esto permite empezar una simulación desde cero con una lista vacía de piscifactorías y el día inicial configurado en cero.

  Inicialización de atributos:

  dia = 0 : Se establece el día actual de la simulación en cero, indicando el inicio del ciclo de gestión.

  piscifactorias = new ArrayList<>() : Se crea una lista vacía para almacenar piscifactorías que se registren durante la simulación.

  nombrePiscifactoria = "" : Se asigna una cadena vacía como nombre inicial para la piscifactoría seleccionada, indicando que aún no se ha elegido ninguna.

28.4 Atributos

- **private int dia:** Se controla el día actual de la simulación.
- **private static List<Piscifactoria> piscifactorias:** Se define el número identificador del tanque.
- **public static String nombreEntidad:** Se almacena el nombre de la entidad o partida.
- **private String nombrePiscifactoria:** Se almacena el nombre de la piscifactoría principal.

- **private final static String[] pecesImplementados**: Se define una lista con los nombres de peces implementados.
- **public static SistemaMonedas monedas**: Se gestiona la cantidad de monedas disponibles.
- **public static Estadisticas estadisticas :** Se registran estadísticas de peces criados, vendidos y ganancias.
- **public static AlmacenCentral almacenCentral**: Se almacena y distribuye comida centralmente.
- **public static Logger logger:** Se registra el historial de eventos.
- **public static Transcriptor transcriptor**: Se transcriben los eventos importantes en un archivo.
- **public final static File errorLog:** Se guarda el archivo de errores `logs/0_errors.log`.

28.5 Métodos

init

- **public void init()**

Inicializa el simulador creando carpetas esenciales, cargando partidas previas o permitiendo crear una nueva, configurando monedas, estadísticas, piscifactorías y el almacén central. Registra todo en los logs y transcripciones, incluyendo monedas iniciales, peces disponibles y estado de los edificios.

menu

- **public void menu()**

Muestra el menú principal con opciones como ver el estado general, gestionar piscifactorías, avanzar días, comprar recursos y salir del simulador. No devuelve nada, solo muestra el menú en consola.

menuPisc

- **public void menuPisc()**

  Presenta una lista de piscifactorías con su nombre, cantidad de peces vivos, peces totales y capacidad total, para que el usuario pueda seleccionar una para gestionar. No devuelve nada, solo muestra opciones en consola.

selectPisc

- public Piscifactoria selectPisc()
  Permite seleccionar una piscifactoría existente mostrando un menú. Si el usuario selecciona una opción válida, devuelve la piscifactoría elegida, o `null` si cancela la selección.

Map.Entry

- public Map.Entry<Piscifactoria, Tanque> selectTank()
  Permite elegir un tanque específico dentro de una piscifactoría seleccionada previamente. Devuelve una pareja `Map.Entry` con la piscifactoría y el tanque elegidos, o `null` si se cancela el proceso.

addFood

- public void addFood()
  Permite añadir comida animal o vegetal a una piscifactoría o al almacén central según el tipo y cantidad seleccionados. Verifica costos y monedas disponibles, actualiza recursos y registra el proceso en los logs. No devuelve nada.

addFish

- public void addFish()
  Añade peces a un tanque seleccionado según el tipo de piscifactoría. Deduce monedas según el costo, verifica si hay espacio suficiente y registra la compra en los registros y estadísticas. No devuelve nada.

sell

- public void sell()
  Vende todos los peces adultos de un tanque seleccionado. Calcula las monedas ganadas y actualiza las estadísticas, devolviendo monedas a la cuenta del jugador. No devuelve ningún valor directamente.

cleanTank

- public void cleanTank()
  Limpia un tanque eliminando todos los peces muertos de su lista interna. Actualiza registros y muestra el resultado en la consola, sin devolver ningún valor.

emptyTank

- public void emptyTank()
  Vacía completamente un tanque seleccionado, eliminando todos los peces vivos y muertos, actualizando la capacidad y registrando la acción en el sistema. No devuelve nada.

addPiscifactoria

- public void addPiscifactoria()
  Permite crear una piscifactoría de río o mar según la selección del usuario. Asigna un nombre, tipo y deduce monedas según el costo correspondiente. Agrega la piscifactoría a la lista del simulador y registra la compra. No devuelve ningún valor.

upgradePiscifactoria

- public void upgradePiscifactoria()
  Mejora una piscifactoría seleccionada incrementando su capacidad de comida o número de tanques. Deduce monedas según el costo, actualiza registros y muestra mensajes de confirmación en la consola. No devuelve nada.

addTanque

- public void addTanque()
  Añade un tanque a una piscifactoría específica verificando monedas y capacidad máxima. Agrega el tanque a la lista interna y registra la compra en los logs. No devuelve ningún valor.

nextDay

- public void nextDay()
  Simula un día en todas las piscifactorías avanzando el ciclo de vida de los peces, vendiendo peces maduros, actualizando monedas y registrando todo el progreso en los registros y la consola. No devuelve ningún valor.

nextDay

- public void nextDay(int dias)
  Simula varios días consecutivos llamando repetidamente a `nextDay()` según el número de días especificado por el usuario. No devuelve nada.

showGeneralStatus

- public void showGeneralStatus()
  Muestra un resumen completo del estado actual del simulador, incluyendo el día actual, monedas disponibles, estado de piscifactorías, peces vivos y almacén central. No devuelve ningún valor, solo muestra información.

showSpecificStatus

- public void showSpecificStatus()
  Muestra el estado detallado de una piscifactoría seleccionada, mostrando información de sus tanques y peces. No devuelve ningún valor, solo muestra el estado en la consola.

showTankStatus

- public void showTankStatus()
  Muestra el estado detallado de un tanque seleccionado, incluyendo información específica de cada pez y su estado vital. No devuelve nada.

showStats

- public void showStats()
  Presenta estadísticas detalladas de peces criados, vendidos y monedas ganadas. Organiza datos por tipo de pez y muestra el resultado en consola sin devolver ningún valor.

recompensas

- public void recompensas()
  Permite usar recompensas disponibles como comida, monedas, tanques y piscifactorías adicionales. Aplica beneficios según las condiciones del juego y registra cada acción en los logs. No devuelve ningún valor.

guardarEstado

- public void guardarEstado()
  Guarda todo el estado actual del simulador en un archivo JSON, incluyendo datos de peces, piscifactorías, monedas y almacenes para su recuperación futura. No devuelve ningún valor.

load

- public void load(String archivoPartida)
  Carga una partida guardada desde un archivo JSON, restaurando todos los datos previos, incluyendo peces, piscifactorías y estadísticas monetarias. No devuelve ningún valor.

gestionarCompraEdificios

- private void gestionarCompraEdificios()
  Gestiona la compra de edificios como almacenes y piscifactorías, mostrando opciones según monedas disponibles. Agrega los edificios comprados a la lista del simulador. No devuelve ningún valor.

gestionarMejoraEdificios

- private void gestionarMejoraEdificios()
  SPermite mejorar edificios existentes, como tanques y almacenes, verificando monedas disponibles y aplicando mejoras si es posible. No devuelve ningún valor.

pecesRandom

- public void pecesRandom()
  Añade cuatro peces aleatorios a una piscifactoría seleccionada, usado como función oculta para pruebas o gestión rápida. No devuelve ningún valor.

distribuirComida

- public static void distribuirComida(int cantidadComida, String tipo)
  Distribuye una cantidad específica de comida entre todas las piscifactorías según su capacidad y tipo de comida seleccionada. No devuelve ningún valor.

addPiscifactoria

- private void addPiscifactoria()
  Permite crear una nueva piscifactoría de tipo río o mar según la selección del usuario y monedas disponibles. Registra la creación en los logs. No devuelve nada.

29. Clase CrearRecompensa

29.1 Descripción

Se implementa una clase encargada de gestionar la creación de recompensas en formato XML para un simulador de piscifactorías, definiendo elementos como alimento, monedas, tanques y almacenes. Se registran y transcriben las recompensas creadas, actualizando los archivos si ya existen.

29.2 Funcionalidades

1. Crear recompensas de alimento (pienso, algas y comida multipropósito).
2. Crear recompensas de monedas.
3. Crear recompensas para tanques, piscifactorías y almacenes.
4. Actualizar archivos XML existentes o crearlos si no están disponibles.
5. Realizar cálculos automáticos para definir cantidades, rarezas y nombres.
6. Registrar y transcribir cada recompensa creada.

29.3 Constructor

- No posee constructor.

29.4 Atributos

- **private static final String REWARDS_DIRECTORY :** Define el directorio donde se almacenan los archivos XML de recompensas.

29.5 Métodos

createPiensoReward

- **public static void createPiensoReward(int type)**

Se crea una recompensa de pienso según el tipo especificado (1-5). Se define su nombre, rareza y cantidad de alimento. Si el archivo XML existe, se actualiza la cantidad; de lo contrario, se genera un archivo nuevo.

createAlgasReward

- **public static void createAlgasReward(int type)**

Se crea una recompensa de algas para alimentar peces filtradores y omnívoros. Se especifica su tipo (1-5), nombre y descripción. Si el archivo XML está presente, se incrementa su

cantidad; de lo contrario, se crea un nuevo archivo.

createComidaReward

- public static void createComidaReward(int type)
  Se genera una recompensa de comida multipropósito según el nivel seleccionado (1-5). Se especifica la cantidad, nombre, descripción y rareza. El archivo se actualiza si ya existe, o se crea uno nuevo.

createMonedasReward

- public static void createMonedasReward(int type)
  Se crea una recompensa de monedas, definiendo la cantidad y la rareza según el nivel (1-5). Si el archivo XML está disponible, se incrementa su valor; si no, se genera uno nuevo.

createTanqueReward

- public static void createTanqueReward(int type, String part)
  Se define una recompensa de tanque de piscifactoría, indicando si es de río o mar según el tipo especificado (1-5) y la parte asignada (A o B). Se actualiza el archivo si existe, o se crea uno nuevo si es necesario.

createPiscifactoriaReward

- public static void createPiscifactoriaReward(int type, String part)
  Se genera una recompensa de piscifactoría, indicando si es de río o mar, junto con la parte correspondiente (A o B). Se actualiza o se crea el archivo según su existencia previa.

createAlmacenReward

- public static void createAlmacenReward(String part)
  Se crea una recompensa de almacén central con la parte especificada (A-D). Se actualiza el archivo XML existente o se crea uno nuevo según sea necesario.

30. Clase UsarRecompensa

30.1 Descripción

Clase encargada de gestionar el uso de recompensas almacenadas en archivos XML dentro del directorio `rewards`. Permite procesar recompensas de comida, monedas, tanques, piscifactorías y almacenes centrales, actualizando los archivos según su cantidad restante y eliminándolos cuando se agotan.

30.2 Funcionalidades

1. Procesa archivos XML para obtener recompensas y reducir cantidades.
2. Añade monedas, comida, tanques, piscifactorías y almacenes según lo definido en el archivo.
3. Verifica la existencia de archivos antes de procesarlos.
4. Documenta eventos ocurridos durante el uso de recompensas.
5. Reduce la cantidad disponible de una recompensa y elimina el archivo si se agota.
6. Devuelve los nombres de las recompensas existentes en el sistema.
7. Verifica si se tienen todas las partes necesarias para construir piscifactorías o almacenes centrales.

30.3 Constructor

- No posee constructor.

30.4 Atributos

- No posee atributos de instancia.

30.5 Métodos

readFood

- **public static void readFood(String fileName)**

Se crea una recompensa de pienso según el tipo especificado (1-5). Se define su nombre, rareza y cantidad de alimento. Si el archivo XML existe, se actualiza la cantidad; de lo contrario, se genera un archivo nuevo.

readCoins

- **public static void readCoins(String fileName)**

Se crea una recompensa de algas para alimentar peces filtradores y omnívoros. Se especifica su tipo (1-5), nombre y descripción. Si el archivo XML está presente, se incrementa su cantidad; de lo contrario, se crea un nuevo archivo.

readTank

- **public static boolean readTank(String fileName)**

  Se genera una recompensa de comida multipropósito según el nivel seleccionado (1-5). Se especifica la cantidad, nombre, descripción y rareza. El archivo se actualiza si ya existe, o se crea uno nuevo.

readPiscifactoria

- **public static Piscifactoria readPiscifactoria(boolean piscifactoriaRio)**

  Se crea una recompensa de monedas, definiendo la cantidad y la rareza según el nivel (1-5). Si el archivo XML está disponible, se incrementa su valor; si no, se genera uno nuevo.

readAlmacen

- **public static boolean readAlmacenCentral()**

  Se define una recompensa de tanque de piscifactoría, indicando si es de río o mar según el tipo especificado (1-5) y la parte asignada (A o B). Se actualiza el archivo si existe, o se crea uno nuevo si es necesario.