Proyecto Piscifactoría

Documentación técnica

Juan Javier Lopez Garrido y
Matías Pérez Bustelo

Índice

Proyecto Piscifactoría

INTRODUCCIÓN

Este proyecto se basa en un programa para manejar una piscifactoría y incluyendo el ciclo de vida de los peces, su alimentación y gestión. A lo largo del módulo se ampliará y modificará cada vez que se vea algo nuevo.

Nosotros para la elaboración del proyecto a la hora de trabajar en conjunto utilizamos el siguiente repositorio: https://github.com/LopezJuanJ/Trabajo-Piscifactoria. Para hacer la siguiente documentación seguiré los paquetes que tengo creados en mi proyecto.

PECES

Aquí se encuentran todas las clases e interfaces que tienen que ver con un pez.

PROPIEDADES

Este es un paquete en el que se guardan los distintos tipos que puede ser un pez; ya sea Activo, carnívoro, Filtrador, voraz…

Activo

La clase Activo es una subclase de la clase Pez y representa un tipo específico de pez, caracterizado por ser activo.

```
public class Activo extends Pez {
    public Activo(boolean sexo, PecesDatos datos) {
        super(sexo, datos);
    }
```

Este constructor crea una instancia de la clase Activo y la inicializa con los valores de sexo y datos proporcionados.

```java
@Override
public void comer(Tanque tanque, Piscifactoria piscifactoria) {
    Random random = new Random();
    int comida = piscifactoria.getComidaActual();
    if (alimentado = false) {
        comida--;
        if (random.nextBoolean()) {
            comida--;
            alimentado = true;
        } else {
            alimentado = true;
        }
    }
}
```

Este método simula el proceso de alimentación del pez activo en el entorno de una piscifactoría. Tiene 50% de comer 2 de alimento ese día.

Carnívoro

La clase Carnívoro es una subclase de la clase Pez y representa un tipo específico de pez

```java
public class Carnivoro extends Pez{
    public Carnivoro(boolean sexo, PecesDatos datos) {
        super(sexo, datos);
        //TODO Auto-generated constructor stub
    }
```

Este constructor crea una instancia de la clase Carnívoro y la inicializa con los valores de sexo y datos proporcionados.

```java
@Override
public void comer(Tanque tanque, Piscifactoria piscifactoria) {
   Random random = new Random();
   int comida = piscifactoria.getComidaActual();
    int pecesMuertos = tanque.getPecesMuertos();
    boolean comerMuerto = random.nextBoolean();
    boolean eliminarPez = random.nextBoolean();
    if (this.alimentado = false){ //Primero comprobar si esta alimentado
        if(pecesMuertos> 0){
        if (comerMuerto = true){//Puede comer un muerto
            alimentado=true;
            if(eliminarPez=true){ //comprobamos si se elimina o no
                pecesMuertos--;
            }
        }else{
            comida--;
        }
    } else{
    }
        }

}
```

El pez puede alimentarse de dos fuentes: peces muertos en el tanque o la comida disponible en la piscifactoría. Se pueden alimentar de peces muertos del tanque. 50% de eliminar al pez muerto tras comer.

CarnívoroActivo

La clase CarnivoroActivo es una subclase de la clase Pez y representa un tipo específico de pez que es carnívoro y activo.

```java
public class CarnivoroActivo extends Pez{
    protected boolean alimentado;

    public CarnivoroActivo(boolean sexo, PecesDatos datos) {
        super(sexo, datos);
        //TODO Auto-generated constructor stub
    }
/**
```

Este constructor crea una instancia de la clase CarnivoroActivo y la inicializa con los valores de sexo y datos proporcionados.

```java
@Override
public void comer(Tanque tanque, Piscifactoria piscifactoria) {
    Random random = new Random();
    int comida = piscifactoria.getComidaActual();
     int pecesMuertos = tanque.getPecesMuertos();
     boolean comerMuerto = random.nextBoolean();
     boolean eliminarPez = random.nextBoolean();
     if (this.alimentado = false){ //Primero comprobar si esta alimentado
         if(pecesMuertos> 0){
         if (comerMuerto = true){//Puede comer un muerto
             alimentado=true;
             if(eliminarPez=true){ //comprobamos si se elimina o no
                 pecesMuertos--;
             }
         }else{
            if(random.nextBoolean()){
                comida-=2;
            }else{
                comida--;
            }
         }
     } else{
     }
         }

}
```

Este método simula el proceso de alimentación de un pez carnívoro activo en el entorno de una piscifactoria. El pez puede alimentarse de peces muertos del tanque.y 50% de eliminar al pez muerto tras comer pero tambien puede tener un 50% de probablidades de comer 2 de alimento ese día.

CarnívoroVoraz

La clase CarnivoroVoraz es una subclase de la clase Pez y representa un tipo específico de pez que es carnívoro y Voraz.

```java
public class CarnivoroVoraz extends Pez{
    protected boolean alimentado;


    public CarnivoroVoraz(boolean sexo, PecesDatos datos) {
        super(sexo, datos);
        //TODO Auto-generated constructor stub
    }
```

Este constructor crea una instancia de la clase CarnivoroVoraz y la inicializa con los valores de sexo y datos proporcionados.

```java
@Override
public void comer(Tanque tanque, Piscifactoria piscifactoria) {
    Random random = new Random();
    int comida = piscifactoria.getComidaActual();
     int pecesMuertos = tanque.getPecesMuertos();
     boolean comerMuerto = random.nextBoolean();
     boolean eliminarPez = random.nextBoolean();
     if (this.alimentado = false){ //Primero comprobar si esta alimentado
         if(pecesMuertos> 0){
         if (comerMuerto = true){//Puede comer un muerto
             alimentado=true;
             if(eliminarPez=true){ //comprobamos si se elimina o no
                 pecesMuertos--;
             }
         }else{
             comida-=2;
         }
     } else{
     }
         }

}
```

Este método simula el proceso de alimentación de un pez carnívoro voraz en el entorno de una piscifactoría. El pez puede alimentarse de peces muertos del tanque. 50% de eliminar al pez muerto tras comer y puede alimentarse doble de comida normal.

Filtrador

La clase Filtrador es una subclase de la clase Pez y representa un tipo específico de pez, caracterizado por tener la capacidad de alimentarse filtrando.

```
public class Activo extends Pez {
    public Activo(boolean sexo, PecesDatos datos) {
        super(sexo, datos);
    }
```

Este constructor crea una instancia de la clase Filtrador y la inicializa con los valores de sexo y datos proporcionados.

```
@Override
public void comer(Tanque tanque, Piscifactoria piscifactoria) {
   Random random = new Random();
   int comida = piscifactoria.getComidaActual();
   boolean comer = random.nextBoolean();
   if(alimentado = false){
    if(comer=true){
        comida--;
        alimentado=true;
    } else{

    }

   }
}
```

Este método simula el proceso de alimentación del pez Filtrador en el entorno de una piscifactoría. Tiene un 50% de probabilidad de no consumir comida.

Voraz

La clase Voraz es una subclase de la clase Pez y representa un tipo específico de pez, caracterizado por tener la capacidad de alimentarse filtrando.

```
public class Activo extends Pez {
    public Activo(boolean sexo, PecesDatos datos) {
        super(sexo, datos);
    }
```

Este constructor crea una instancia de la clase Voraz y la inicializa con los valores de sexo y datos proporcionados.

```
@Override
public void comer(Tanque tanque, Piscifactoria piscifactoria) {
    int comida = piscifactoria.getComidaActual();
    if(alimentado=false){
        comida-=2;
        alimentado=true;
    }


}
```

Este método simula el proceso de alimentación del pez Voraz en el entorno de una piscifactoría. Come siempre doble de alimento.

PEZ ESPECÍFICO

Todos los distintos peces van a tener lo mismo pero cambiando su nombre.

```
public class Besugo extends Carnivoro implements IMar {

  public Besugo(boolean sexo) {
    super(sexo, AlmacenPropiedades.BESUGO);
  }

}
```

Este constructor crea una instancia de la clase Besugo y la inicializa con el valor de sexo proporcionado. Además, invoca el constructor de la clase base Carnívoro para inicializar los atributos heredados y utiliza la información de propiedades almacenada en AlmacenPropiedades.BESUGO.

La clase Besugo hereda el método comer de la clase Carnívoro y cualquier otro método que esté definido en la clase base o la interfaz implementada (I mar).

INTERFACES

```java
package Peces;

public interface IRio {


}
```

La interfaz IRio hace referencia a los peces de río para poder implementar algunos métodos en los que se necesita diferenciar si un pez es de mar o de río.

```java
package Peces;

public interface IMar {


}
```

La interfaz IMar hace referencia a los peces de mar para poder implementar algunos métodos en los que se necesita diferenciar si un pez es de mar o de río.

PEZ

La clase Pez es una clase abstracta que sirve como base para representar a los diferentes tipos de peces en una piscifactoría.

```java
*/
public Pez (boolean sexo, PecesDatos datos) {
    this.edad = 0;
    this.sexo = sexo;
    this.datos = datos;

}
```

Este constructor crea una instancia de la clase Pez y la inicializa con los valores de sexo y datos proporcionados.

```java
public abstract void comer(Tanque<? extends Pez> tanque, Piscifactoria piscifactoria);
```

Este método abstracto establece la lógica de alimentación específica para cada tipo de pez. Debe ser implementado por las subclases para determinar cómo un pez particular se alimenta.

```java
public void showStatus() {
    System.out.println("---------------" + this.datos.getNombre() + "---------------");
    System.out.println("Edad " + edad + "dias");
    System.out.println("Sexo: " + sexo);
    if(alimentado = true){
        System.out.println(x:"Alimentado: Si");
    } else {
        System.out.println(x:"Alimentado: No");
    }

    if (fertilidad = true) {
        System.out.println(x:"Fertilidad: Si");
    } else {
        System.out.println(x:"Fertilidad: No");
    };

    if(vida = true){
        System.out.println(x:"Vivo: Si");
    } else {
        System.out.println(x:"Vivo: No");
    }

    if (edad > 5){
        System.out.println(x:"Adulto: Si");
    }else{
        System.out.println(x:"Adulto: No");
    }
}
```

Este método muestra en la consola el estado actual del pez, incluyendo su edad, sexo, estado de alimentación, fertilidad, estado de vida y si es adulto o no.

```java
public boolean verificarMadurez(){
    if (this.edad >= datos.getMadurez()){
        return true;
    }else{
        return false;
    }
}
```

Este método verifica si el pez ha alcanzado la madurez en función de su edad. Devuelve **true** si es maduro y **false** en caso contrario.

```java
public void comprobarComida(int comida){
        if (comida == 0){
        alimentado=false;
    }else{
        alimentado=true;
    }

}
```

Este método verifica si el pez ha sido alimentado con base en la cantidad de comida disponible.

```
public void grow(Tanque<? extends Pez> tanque, Piscifactoria piscifactoria ) {
    Random random = new Random();
    if(this.vida = true){
        comer(tanque, piscifactoria);
        if(!this.alimentado && !random.nextBoolean()){
            this.vida = false;
        }else{
            this.edad++;

        }
       if (verificarMadurez()){
            this.ciclo--;
       }
       if( this.ciclo == 0){
        this.fertilidad=true;
       } else{
        this.fertilidad=false;
    }
    }
}
```

Este método simula el crecimiento del pez. El pez se alimenta, aumenta su edad y se verifica su fertilidad y madurez.

```
public void reset() {
    this.edad = 0;
    this.fertilidad = false;
    this.vida = true;
    this.alimentado = false;

}
```

Este método restablece las propiedades del pez a sus valores iniciales.

MONEDERO

La clase Monedero representa un objeto que almacena la cantidad de monedas disponibles en una aplicación. Esta clase utiliza el patrón Singleton para garantizar que solo exista una única instancia del monedero en toda la aplicación.

```
private Monedero() {
    this.monedas = 100;
}
```

El constructor privado crea una instancia del monedero y lo inicializa con 100 monedas por defecto. Este constructor se usa en el patrón Singleton para garantizar que solo exista una instancia de Monedero en la aplicación.

```
public void vender(int cantidad) {
    this.monedas += cantidad;
}
```

Este método aumenta la cantidad de monedas en el monedero al realizar una venta.

```
public void comprar(int cantidad) {
    this.monedas -= cantidad;
}
```

Este método disminuye la cantidad de monedas en el monedero al realizar una compra.

```
public void saberMonedas() {
    System.out.println("Actualmente tienes: " + this.monedas + " monedas.");
}
```

Este método imprime en la consola la cantidad actual de monedas en el monedero.

ALMACÉN GENERAL

La clase Almacén representa un almacén de comida con una cantidad de comida disponible. Esta clase utiliza el patrón Singleton para garantizar que solo exista una instancia de este almacén central en toda la aplicación.

```
public AlmacenCentral() {
    this.comida = 200;
    this.comidaMax= 200;
}
```

Este constructor inicializa la cantidad de comida en 200 unidades y establece la capacidad máxima en 200 unidades.

```
public static AlmacenCentral getInstance() {
    if (instance == null ) {
        instance = new AlmacenCentral();
    }
    return instance;

}
```

Este método estático se utiliza para obtener la única instancia del almacén central. Si no la hay la crea.

```
public void upgradeAlmacen(){
    Monedero.getInstance().comprar(cantidad:100);
    this.comidaMax += 50;
}
```

Este método permite mejorar la capacidad de almacenamiento del almacén central. Una mejora de almacenamiento tiene un costo de 100 monedas y luego, aumenta la capacidad máxima del almacén en 50 unidades.

```java
public void restarComida(int cantidad){
    if(comida > 0){
        comida-=cantidad;
    }else{

        System.out.println(x:"Sin comida en el Almacen");
    }
}
```

Este método permite restar una cantidad específica de comida del almacén

TANQUE

La clase Tanque representa un tanque que puede alojar una colección de peces. El tanque tiene una capacidad máxima para peces dependiendo del tipo, y se utiliza para mantener y gestionar la población de peces en una piscifactoría.(class Tanque<T extends Pez>)

```java
public void showStatus() {
    System.out.println("============== Tanque " + this.nombre + "==============");

            System.out.println("Ocupacion: " + this.getPeces().size() + "/" + capacidadMax + "("+ (this.getPeces().size() / capacidadMax) * 100 + "%)");
            System.out.println("Peces vivos: " + this.getPecesVivos() + "/" + this.getPeces().size() + "("+ porcentaje(this.getCantPecesVivos(),this.getPeces().size()) + "%)");
            System.out.println("Peces alimentados: " + this.getPecesAlimentados() + "/" + this.getPecesVivos() + "("+ porcentaje(this.getPecesAlimentados(), this.getPecesVivos()) + "%)");
            System.out.println("Peces adultos: " + this.getPecesMaduros() + "/" + this.getPecesVivos() + "("+ porcentaje(this.getPecesMaduros(), this.getPecesVivos()) + "%)");
            System.out.println("Hembras/Machos: " + this.getMachos() + "/" + this.getHembras());
            System.out.println("Fertiles: " + this.getPecesFertiles() + "/" + this.getPecesVivos() + "("+ porcentaje(this.getPecesFertiles(),this.getPecesVivos()) + "%)");
}
```

Este método muestra información sobre el estado del tanque, incluyendo la ocupación, la cantidad de peces vivos, la cantidad de peces alimentados, la cantidad de peces maduros, la cantidad de machos y hembras, y la cantidad de peces fértiles.

```java
public void showFishStatus() {
    for (T pez : peces) {
        pez.showStatus();
    }

}
```

Muestra la información detallada de cada pez.

```java
public void nextDay(Tanque<? extends Pez> tanque, Piscifactoria piscifactoria) {
    for (T pez : peces) {
        pez.grow(tanque, piscifactoria);
    }
}
```

Este método simula el avance de un día en el tanque, actualizando el estado de todos los peces presentes en el tanque llamando al método que hace crecer a cada pez.

```java
public void reproduccion(boolean fertilidad) {
    boolean machoFertil = false;
    boolean hembraFertil = false;

    for (T pez : peces) {
        if (pez.isSexo() && pez.fertilidad) {
            machoFertil = true;
        } else if (!pez.isSexo() && pez.fertilidad) {
            hembraFertil = true;
            int cantHuevos= pez.getDatos().getHuevos();
            if(cantHuevos%2==0){
                int machos = cantHuevos/2;
                for (int i=0; i <= machos; i++){
                    // Pez pezChico = new Pez(true, pez.getDatos());
                }
            }
        }

        if (machoFertil && hembraFertil) {
            System.out.println(x:"Los peces se pueden reproducir");
            break;
        }
    }

    if (!machoFertil || !hembraFertil) {
        System.out.println(x:"Los peces no se pueden reproducir");
    }
}
```

Este método verifica si hay al menos un pez macho y una pez hembra fértiles en el tanque y muestra un mensaje indicando si los peces pueden reproducirse o no.

```
public void showCapacity(String nombrePisc){
    System.out.println("Tanque " + this.nombre + " de la piscifactoria " + nombrePisc + " al " + (this.getPeces().size()/this.capacidadMax)*100 + "% de capacidad" + this.getPeces().size() + "/" + this.capacidadMax );
}
```

Este método muestra información sobre la capacidad actual del tanque y el porcentaje de ocupación en relación con su capacidad máxima.

```
public String getTipoPezTank(){
        if(peces.isEmpty()){
            return "NO hay peces en el tanque";
        }
    return  peces.get(index:0).getDatos().getNombre();
}
```

Este método obtiene el tipo de pez que está en el tanque.

```
public int getMachos() {
    int getMachos = 0;
    for (T pez : peces) {
        if (pez.isSexo() && pez.isVida()) {
            getMachos++;
        }
    }
    return getMachos;
}
```

Este método obtiene la cantidad de machos vivos.

```
public int getHembras() {
    int getHembras = 0;
    for (T pez : peces) {
        if (!pez.isSexo() && pez.isVida()) {
            getHembras++;
        }
    }
    return getHembras;
}
```

Este método obtiene la cantidad de hembras vivos.

```java
public int getPecesFertiles() {
    int pecesFertiles = 0;
    for (T pez : peces) {
        if (pez.isFertilidad() == true) {
            pecesFertiles++;
        }
    }
    return pecesFertiles;
}
```

Este método obtiene la cantidad de peces fértiles.

```java
public int getPecesMuertos() {
    int pecesMuertos = 0;
    for (T pez : peces) {
        if (!pez.isVida()) {
            pecesMuertos++;
        }
    }
    return pecesMuertos;
}
/**
 * Obtiene la cantidad de peces
 * @return La cantidad de peces
 */
public int getPecesVivos() {
    int pecesVivos = 0;
    for (T pez : peces) {
        if (pez.isVida()) {
            pecesVivos++;
        }
    }
    return pecesVivos;
}
```

Los siguientes métodos obtienen la cantidad de peces vivos(getPecesVivos) o muertos(getPecesMuertos).

```java
public int getPecesAlimentados() {
    int getPecesAlimentados = 0;
    for (T pez : peces) {
        if (pez.isAlimentado()) {
            getPecesAlimentados++;
        }
    }
    return getPecesAlimentados;
}
```

Este método obtiene todos los peces de un tanque que ya comieron.

```java
public int getPecesMaduros() {
    int getPecesMaduros = 0;
    for (T pez : peces) {
        if (pez.verificarMadurez()) {
            getPecesMaduros++;
        }
    }
    return getPecesMaduros;
}
```

Este método obtiene todos los peces de un tanque que ya son maduros.

PISCIFACTORÍA

Esta clase se utiliza para gestionar varios aspectos relacionados con la piscifactoría, como la cantidad de comida, los tanques, la capacidad de los tanques, los peces en cada tanque y más.

```java
public void showStatus() {
    System.out.println("==============" + this.nombre + "==============");
    System.out.println("Tanques: " + getCantTanques());
    System.out.println("Ocupación: " + this.getPecesTotales() +"/"+this.getEspacioTotal()+ " (" +porcentaje(this.getPecesTotales(),this.getEspacioTotal()) + "%)");
    System.out.println("Peces vivos: " + this.getVivosTotales()+"/"+getPecesTotales() + " (" + porcentaje(this.getVivosTotales(),getPecesTotales()) +"%)");
    System.out.println("Peces alimentados:" + this.getPecesAlimentadossTotales()+"/"+this.getVivosTotales() + " (" + porcentaje(this.getPecesAlimentadossTotales(),this.getVivosTotales())+"%)");
    System.out.println(x:"Hembras / Machos: H/M");
    System.out.println(x:"Fértiles: fértiles / vivos");
    System.out.println(x:"Almacén de comida: actual / max (x%)");
}
```

Este método muestra información detallada sobre el estado general de la piscifactoría, incluyendo la cantidad de tanques, la ocupación, la cantidad de peces vivos, la cantidad de peces alimentados y si es hembra o macho si es fértil y esta vivo.

```java
public double porcentaje(int numero1, int numero2){
  if (numero2 == 0) {
      return 0.0;
  }
  double porcentaje = (double) numero1 / numero2 * 100;
  porcentaje = Math.round(porcentaje * 10) / 10.0;
  return porcentaje;
}
```

Este método calcula el porcentaje en base a dos números enteros

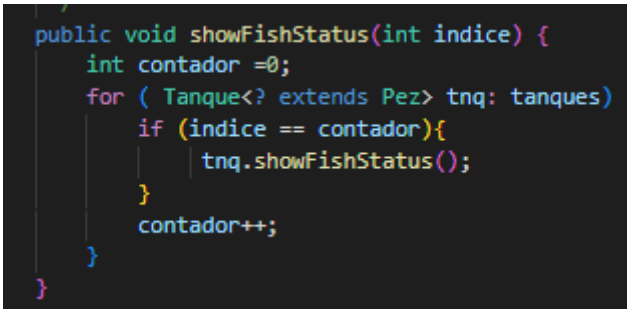
```java
public void showTankStatus() {
    int indice =1;
    for ( Tanque<? extends Pez> tnq: tanques) {
        System.out.println(indice);
        tnq.showStatus();
        System.out.println("Tipo: " + tnq.getTipoPezTank());
        indice++;
    }

}
```

Este método muestra un menú que presenta la información de cada tanque en la piscifactoría, incluyendo estadísticas de ocupación y estado de los peces en cada tanque.

```java
public void addTank(int tipo){
    System.out.print(s:"Dime Nombre del tanque");
    String nom = newScan.nextLine();
    Tanque tanque = new Tanque<Pez>(nom, tipo);
    this.tanques.add(tanque);
    if (tipo==25){
        Monedero.getInstance().comprar(cantidad:150);
    }else{
        Monedero.getInstance().comprar(cantidad:600);
    }
}
```

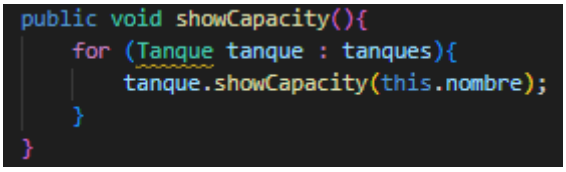
Este método permite agregar un nuevo tanque a la piscifactoría.

También realiza una compra y actualiza el monedero basado en el tipo de tanque agregado.



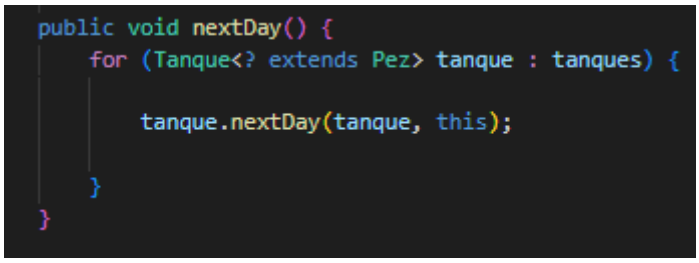
```
public void showFishStatus(int indice) {
    int contador =0;
    for ( Tanque<? extends Pez> tnq: tanques)
        if (indice == contador){
            tnq.showFishStatus();
        }
        contador++;
    }
}
```

Muestra la información sobre los peces de un determinado tanque



```
public void showCapacity(){
    for (Tanque tanque : tanques){
        tanque.showCapacity(this.nombre);
    }
}
```

Este método muestra la ocupación de todos los tanques en la piscifactoría, mostrando un porcentaje.



```
public void nextDay() {
    for (Tanque<? extends Pez> tanque : tanques) {

        tanque.nextDay(tanque, this);

    }
}
```

Este método avanza un día en la piscifactoría y llama al nextDay de cada tanque.

```java
public void sellFish() {
    for ( Tanque<? extends Pez> tanque : tanques) {
        for (Pez pez : tanque.getPeces()){
            if(pez.isVida() == true && pez.verificarMadurez()){
                PecesDatos datos = pez.getDatos();
                int precio = datos.getMonedas();
                Monedero.getInstance().vender(precio);

            }
            }
        tanque.limpiarPeces();
    }
}
```

Este método vende peces que son adultos y fértiles, obteniendo monedas y actualizando el monedero. Se ejecuta en todos los tanques y elimina los peces vendidos.

```java
public void upgradeFood() {
    Monedero monedas = Monedero.getInstance();
    int cantAument = 0;
   if (this instanceof IRio) {
        if(!(this.comidaMaxima==250)){
            if(monedas.getMonedas()>=100){
                monedas.comprar(cantidad:100);
                setComidaMaxima(25+this.comidaMaxima);
                cantAument = 25;
                System.out.println("Almacen de la comida de la piscifactoria "+ this.nombre + " mejorado. Su cacidad en " + cantAument + " hasta un total de " + this.capacidadMaxima);
            } else{
                System.out.println(x:"Dinero Insuficiente");
            }
        }
    } else {
        if(!(this.comidaMaxima==1000)){
            if(monedas.getMonedas()>=200){
                monedas.comprar(cantidad:200);
                setComidaMaxima(100+this.comidaMaxima);
                cantAument = 100;
                System.out.println("Almacen de la comida de la piscifactoria "+ this.nombre + " mejorado. Su cacidad en " + cantAument + " hasta un total de " + this.capacidadMaxima);
            } else{
                System.out.println(x:"Dinero Insuficiente");
            }
        }
    }

}
```

Este método se utiliza para aumentar la capacidad de almacenamiento de comida. El costo de la mejora varía según el tipo de piscifactoría.

```java
public  void deleteEmptyTank(int indice){

    if (tanques.get(indice).peces.isEmpty()) {
        tanques.remove(indice);
    } else {
        System.out.println(x:"El tanque tiene peces, no puede ser eliminado.");
    }
}
```

Este método elimina un tanque vacío en la piscifactoría, siempre que no haya peces presentes en él.

PiscMar

La clase PiscMar es una subclase de la clase Piscifactoría y representa una piscifactoría de mar. El Constructor de la clase que inicializa una nueva piscifactoría de mar con un nombre dado. Crea un tanque inicial y establece la cantidad de comida disponible, su capacidad máxima y la capacidad máxima de los tanques.

```java
package Piscifactoria;

import Peces.Pez;
import Tanque.Tanque;

public class PiscMar extends Piscifactoria {

    public PiscMar(String nombre) {
        super(nombre);
        comidaActual = 100;
        comidaMaxima=100;
        capacidadMaxima=100;
        Tanque tanque = new Tanque<Pez>(nombre:"Primer Tanque", capacidadMaxima);
        this.tanques.add(tanque);
    }

}
```

PiscRio

La clase PiscRio es otra subclase de la clase Piscifactoria que representa una piscifactoría especializada en la cría y gestión de peces en entornos de agua dulce, como ríos. El Constructor de la clase que inicializa una nueva piscifactoría de rio con un nombre dado. Crea un tanque inicial y establece la cantidad de comida disponible, su capacidad máxima y la capacidad máxima de los tanques.

```java
package Piscifactoria;

import Peces.Pez;
import Tanque.Tanque;

public class PiscMar extends Piscifactoria {

    public PiscMar(String nombre) {
        super(nombre);
        comidaActual = 100;
        comidaMaxima=100;
        capacidadMaxima=100;
        Tanque tanque = new Tanque<Pez>(nombre:"Primer Tanque", capacidadMaxima);
        this.tanques.add(tanque);
    }

}
```

Simulador

Esta clase se utiliza para gestionar varios aspectos relacionados con la piscifactoría, como la cantidad de comida, los tanques, la capacidad de los tanques, los peces en cada tanque y más.

```
public void init() {
  Scanner scanner = new Scanner(System.in);

  Monedero monedero = Monedero.getInstance();
  System.out.print(s:"Escribe el nombre de la entidad/empresa/partida: ");
  this.nombreEmpresa = scanner.nextLine();

  System.out.print(s:"Por favor, ingrese el nombre de la piscifactoría:");
  String nombrePiscifactoria = scanner.nextLine();

  PiscRio inicial = new PiscRio(nombrePiscifactoria);
  piscifactorias.add(inicial);
}
```

Metodo para inicializar la partida ,el cual pide al usuario , nombre para la empresa y nombre de la piscifactoria iniciales

```java
public void menu() {
    int seleccion;
    do {
        Scanner scanner = new Scanner(System.in);
        System.out.println(x:"=========Menu=========");
        System.out.println(x:"1. Estado general");
        System.out.println(x:"2. Estado piscifactoria");
        System.out.println(x:"3. Estado tanques");
        System.out.println(x:"4. Informes");
        System.out.println(x:"5. Ictiopedia");
        System.out.println(x:"6. Pasar dia");
        System.out.println(x:"7. Comprar comida");
        System.out.println(x:"8. Comprar peces");
        System.out.println(x:"9. Vender peces");
        System.out.println(x:"10. Limpiar tanques");
        System.out.println(x:"11. Vaciar tanque");
        System.out.println(x:"12. Mejorar");
        System.out.println(x:"13. Pasar varios dias");
        System.out.println(x:"14. Salir");
        System.out.print(s:"Selecciona una opcion: ");
        seleccion = newScan.nextInt();
        switch (seleccion) {
            case 1:
                this.showGeneralStatus();
                break;
            case 2:
                this.showSpecificStatus();
                break;
            case 3:
                this.showTankStatus();
                break;
            case 4:
              showStats();
                break;
            case 5:
                this.showIctio();
                break;
            case 6:
                this.nextDay();
                break;
            case 7:
                this.AddComidaFin()
                break;
            case 8:
                this.addFish();
                break;
            case 9:
                this.sell();
                break;
            case 10:
                this.cleanTank();
                break;
            case 11:
                this.emptyTank();
                break;
            case 12:
                this.upgrade();
                break;
            case 13:
              System.out.print(s:"Cuantos dias avance:");
              int valor = newScan.nextInt();
              for (int i = 0; i <= valor; i++) {
                  nextDay();
              }
                break;

            case 98:
                addFish98();
                break;
            case 99:
              System.out.println(x:"1000 monedas mas");
                Monedero.getInstance().setMonedas(Monedero.getInstance().getMonedas()+1000);
                break;
        }
    } while (seleccion != 14);
```

Menú para seleccionar las opciones a realizar para la gestión de la piscifactoría, según los numero que pulse el usuario haremos distintas cosas

```java
public void menuPisc() {
  System.out.println(x:"Seleccione una opcion: ");
  System.out.println(x:"------------------------- Piscifactorías -------------------------");
  System.out.println(x:"[Peces vivos / Peces totales / Espacio total]");
  getDatosPisc();
  System.out.println(x:"0-Cancelar");

}
```

Menú que nos muestra las piscifactorías del sistema y y el formato que tendrán los datos así mostrados.

```java
public Pez menuSelPez(Piscifactoria pisc){
  int sel;
  if (pisc instanceof IMar){
    System.out.println(x:"1.Tilapia del Nilo");
    System.out.println(x:"2.Lucio del norte");
    System.out.println(x:"3.Corvina");
    System.out.println(x:"4.Salmon Chinook ");
    System.out.println(x:"5.Pejerrey");
  } else {
    System.out.println(x:"6.Lenguado Europeo");
    System.out.println(x:"7.Caballa");
    System.out.println(x:"8.Robalo");
    System.out.println(x:"9.Lubina Europea");
    System.out.println(x:"10.Besugo");
  }
  System.out.println(x:"11.Lubina Rayada");
  System.out.println(x:"12. Salmon atlantico");


  System.out.print(s:"Selecciona un Pez para comprar: ");
  sel = newScan.nextInt();
  switch (sel){
    case 1:
      return new TilapiaNi();
    case 2:
      return  new LucioNor();

    case 3:
      return new Corvina();

    case 4:
      return new SalmonCh();

    case 5:
      return new Pejerrey();

    case 6:
      return new LenguadoEu();

    case 7:
      return new Caballa();

    case 8:
      return new Robalo();
    case 9:
      return new LubinaEu();

    case 10:
      return new Besugo();

    case 11:
      return new LubinaRa();

    case 12:
      return new SalmonAt();

  }
```

Menú para seleccionar los peces a añadir en un tanque , y por supuesto realiza una compra de los mismos.

```java
public Pez menuSelPez98(Piscifactoria pisc){
  Random ran = new Random();
  int sel;
  if (pisc instanceof IMar){
    System.out.println(x:"1.Tilapia del Nilo");
    System.out.println(x:"2.Lucio del norte");
    System.out.println(x:"3.Corvina");
    System.out.println(x:"4.Salmon Chinook ");
    System.out.println(x:"5.Pejerrey");
  } else {
    System.out.println(x:"6.Lenguado Europeo");
    System.out.println(x:"7.Caballa");
    System.out.println(x:"8.Robalo");
    System.out.println(x:"9.Lubina Europea");
    System.out.println(x:"10.Besugo");
  }
  System.out.println(x:"11.Lubina Rayada");
  System.out.println(x:"12. Salmon atlantico");

  if(pisc instanceof IMar){
    sel= ran.nextInt(1,5);
  }else {
    sel=ran.nextInt(6,7);
  }
  switch (sel){
    case 1:
      return new TilapiaNi();
    case 2:
      return  new LucioNor();

    case 3:
      return new Corvina();

    case 4:
      return new SalmonCh();

    case 5:
      return new Pejerrey();

    case 6:
      return new LenguadoEu();

    case 7:
      return new Caballa();

    case 8:
      return new Robalo();
    case 9:
      return new LubinaEu();

    case 10:
      return new Besugo();

    case 11:
      return new LubinaRa();

    case 12:
      return new SalmonAt();

  }

  return null;
}
```

Menú que hace lo mismo que el anterior pero calcula un random y compra el pez random

```java
/**
 * Obtiene los datos de las piscifactorías y los muestra en el menú.
 */
public void getDatosPisc() {
  int indice = 1;
  for (Piscifactoria piscifactoria : piscifactorias) {
    System.out.println(indice + ".- " + piscifactoria.getNombre() + "[" + piscifactoria.getVivosTotales() + "/"
        + piscifactoria.getPecesTotales() + "/" + piscifactoria.getEspacioTotal() + "]");
    System.out.println("Comida de la piscifactoria "+ piscifactoria.getNombre()+ ": " + piscifactoria.getComidaActual());
    indice++;

  }
}
```

Es el método que se llama para ver los datos de cada piscifactoría y muestra los peces vivos/total/espacio total

```java
public int selectPisc() {
  Scanner scanner = new Scanner(System.in);
  menuPisc();
  System.out.print("Seleccione una piscifactoría (0-" + (piscifactorias.size()) + "): ");
  int selPisc = scanner.nextInt();
  if (selPisc > 0) {
    return selPisc;
  } else {
    return 0;
  }
}
```

Método para seleccionar una piscifactoría en concreto

```java
public int selectTank() {
  int valor = selectPisc() - 1;
  Scanner scanner = new Scanner(System.in);
  int indice = 0;

  for (Piscifactoria piscifactoria : piscifactorias) {
    if (indice == valor) {
      System.out.println();
      piscifactoria.showTankStatus();
    }
    indice++;
  }
  System.out.print(s:"Selecciona el indice de un tanque: ");
  int seleccion = scanner.nextInt();
  return seleccion;
}
```

Método Para Seleccionar un Tanque en concreto

```
public void showGeneralStatus() {
    getDatosPisc();
  System.out.println("Dias: " + this.dias);
  System.out.println("Monedas: " + Monedero.getInstance().getMonedas());

  if (!this.creado) {
    System.out.println(x:"No dispone de Almacen Central");
  } else {
    System.out.println("Almacen con " + AlmacenCentral.getInstance().getComida() + "/" + AlmacenCentral.getInstance().getComidaMax()
        + " de comida, esta al " + (AlmacenCentral.getInstance().getComida() / AlmacenCentral.getInstance().getComidaMax())* 100 + "%");
  }
}
```

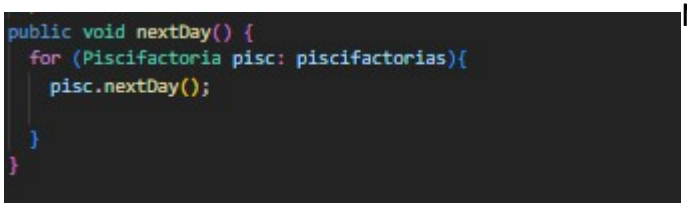
Este método nos muestra los datos de cada piscifactoría junto las monedas globales y los días que lleva el juego

```
public void showTankStatus() {
  int tanqueMostrar = selectTank() - 1;
  int indice = 0;
  for (Piscifactoria piscifactoria : piscifactorias) {
    if (indice == tanqueMostrar) {
      piscifactoria.showFishStatus(tanqueMostrar);
    }
  }
}
```

Método para ver la información de un tanque el cual es seleccionado por el método selectTank

```java
public void showIctio() {
  int sel;
  System.out.println(x:"1.Tilapia del Nilo");
  System.out.println(x:"2.Lucio del norte");
  System.out.println(x:"3.Corvina");
  System.out.println(x:"4.Salmon Chinook ");
  System.out.println(x:"5.Pejerrey");
  System.out.println(x:"6.Lenguado Europeo");
  System.out.println(x:"7.Caballa");
  System.out.println(x:"8.Robalo");
  System.out.println(x:"9.Lubimna Rayada");
  System.out.println(x:"10.Lubina Europea");
  System.out.println(x:"11. Salmon atlantico");
  System.out.println(x:"12.Besugo");
  System.out.print(s:"Selecciona un Pez: ");
  sel = newScan.nextInt();
  newScan.nextLine();
  switch (sel) {
    case 1:
      TilapiaNi.showIctio();
      break;
    case 2:
      LucioNor.showIctio();
      break;
    case 3:
      Corvina.showIctio();
      break;
    case 4:
      SalmonCh.showIctio();
      break;
    case 5:
      Pejerrey.showIctio();
      break;
    case 6:
      LenguadoEu.showIctio();
      break;
    case 7:
      Caballa.showIctio();
      break;
    case 8:
      Robalo.showIctio();
      break;
    case 9:
      LubinaRa.showIctio();
      break;
    case 10:
      LubinaEu.showIctio();
      break;
    case 11:
      SalmonAt.showIctio();
      break;
    case 12:
      Besugo.showIctio();
      break;


    default:
      break;
  }
}
```

Método para ver los datos de cada pez , los cuales tenemos en nuestro juego



```
public void nextDay() {
  for (Piscifactoria pisc: piscifactorias){
    pisc.nextDay();

  }
}
```

Método para avanzar un día

Metodo para gestionar la distribution de comida

```java
public void addFood(Piscifactoria piscifactoria) {
  System.out.println(x:"----------------Comprar Comida----------------");
  System.out.println(x:"1.5 de comida");
  System.out.println(x:"2.10 de comida");
  System.out.println(x:"3.25 de comida");
  System.out.println(x:"4. Llenar");
  System.out.print(s:"Seleccione una opcion: ");
  int seleccion = newScan.nextInt();
  if (AlmacenCentral.getInstance() == null) {
    if (seleccion == 1) {
      if (piscifactoria.getComidaMaxima() - piscifactoria.getComidaActual() >= 5) {
        piscifactoria.setComidaActual(piscifactoria.getComidaActual() + 5);
        Monedero.getInstance().comprar(cantidad:5);
      } else {
        System.out.println(x:"No tienes suficiente espacio");
      }
    } else if (seleccion == 2) {
      if (piscifactoria.getComidaMaxima() - piscifactoria.getComidaActual() >= 10) {
        piscifactoria.setComidaActual(piscifactoria.getComidaActual() + 10);
        Monedero.getInstance().comprar(cantidad:10);
      }else{
        System.out.println(x:"No tienes suficiente espacio");
      }
    } else if (seleccion == 3) {
      if (piscifactoria.getComidaMaxima() - piscifactoria.getComidaActual() >= 25) {
        piscifactoria.setComidaActual(piscifactoria.getComidaActual() + 25);
        Monedero.getInstance().comprar(cantidad:20);
      }else{
        System.out.println(x:"No tienes suficiente espacio");
      }
    } else if (seleccion == 4) {
      int valorAAumentar = piscifactoria.getComidaMaxima() - piscifactoria.getComidaActual();
      if(valorAAumentar == 0){
        System.out.println(x:"Almacen lleno");
      }else {
        int cantVecesCompletas = valorAAumentar / 25;
        int cantidadIncompleta = valorAAumentar % 25;
        int totalDinero = 20 * cantVecesCompletas + cantidadIncompleta;
        int totalComida = 25 * cantVecesCompletas + cantidadIncompleta;
        piscifactoria.setComidaActual(piscifactoria.getComidaActual() + totalComida);
        Monedero.getInstance().comprar(totalDinero);
      }
    }
```

```java
//En caso de Almacen
} else{
  if(seleccion==1) {
    if (AlmacenCentral.getInstance().getComidaMax() - AlmacenCentral.getInstance().getComida() >= 5) {
      AlmacenCentral.getInstance().setComida(AlmacenCentral.getInstance().getComida() + 5);
      Monedero.getInstance().comprar(cantidad:5);
    }else{
      System.out.println(x:"No tienes suficiente espacio");
    }
  } else if (seleccion == 2) {
    if (AlmacenCentral.getInstance().getComidaMax() - AlmacenCentral.getInstance().getComida() >= 10) {
      AlmacenCentral.getInstance().setComida(AlmacenCentral.getInstance().getComida() + 10);
      Monedero.getInstance().comprar(cantidad:10);
    }else{
      System.out.println(x:"No tienes suficiente espacio");
    }
  } else if (seleccion==3) {
    if (AlmacenCentral.getInstance().getComidaMax() - AlmacenCentral.getInstance().getComida() >= 25) {
      AlmacenCentral.getInstance().setComida(AlmacenCentral.getInstance().getComida() + 25);
      Monedero.getInstance().comprar(cantidad:20);
    }else{
      System.out.println(x:"No tienes suficiente espacio");
    }
  } else if (seleccion ==4) {
    int valorAAumentar = AlmacenCentral.getInstance().getComidaMax() - AlmacenCentral.getInstance().getComida();
    if(valorAAumentar == 0){
      System.out.println(x:"Almacen lleno");
    }else {
      int cantVecesCompletas = valorAAumentar / 25;
      int cantidadIncompleta = valorAAumentar % 25;
      int totalDinero = 20 * cantVecesCompletas + cantidadIncompleta;
      int totalComida = 25 * cantVecesCompletas + cantidadIncompleta;
      AlmacenCentral.getInstance().setComida(AlmacenCentral.getInstance().getComida() + totalComida);
      Monedero.getInstance().comprar(totalDinero);
    }
  }
for (Piscifactoria pisc : piscifactorias) {
  // Verificar si hay comida en el almacén y si la piscifactoria tiene espacio
  while (AlmacenCentral.getInstance().getComida() > 0 && pisc.getComidaMaxima() > pisc.getComidaActual()) {
    AlmacenCentral almacen = AlmacenCentral.getInstance();

    // Verificar si la piscifactoria puede aceptar al menos una unidad de comida
    if (pisc.getComidaMaxima() - pisc.getComidaActual() > 0) {
      almacen.setComida(almacen.getComida() - 1);
      pisc.setComidaActual(pisc.getComidaActual() + 1);
    } else {
      break;
    }
  }
}
```

```java
public void AddComidaFin(){
  int valor = selectPisc() - 1;
  if(valor<1 || valor>piscifactorias.size()){
    addFood(piscifactorias.get(valor));
  }
}
```

Método para invocar toda la gestión de comida.

```java
public void addFish() {
  int valor = selectPisc() - 1;
  if(valor<1 || valor>piscifactorias.size()){
    Piscifactoria pisc = piscifactorias.get(valor);
    pisc.anadirPez(menuSelPez(pisc));
  }
}
public void addFish98() {
  int valor = selectPisc() - 1;
  if(valor<1 || valor>piscifactorias.size()){
    Piscifactoria pisc = piscifactorias.get(valor);
    for (int i = 0; i < 4; i++) {
      pisc.anadirPez(menuSelPez98(pisc));
    }
  }
}
```

Metodo para añadir peces y metodo para añadir peces en la opcion 98

```java
public void sell() {
  for (Piscifactoria piscifactoria : piscifactorias) {
      piscifactoria.sellFish();

  }
}
```

Método para vender Peces

```java
public void cleanTank() {
  int indice = 0;
  int valor = selectPisc() - 1;

  for (Piscifactoria piscifactoria : piscifactorias) {
    if (indice == valor) {
      for (Tanque tanque : tanques) {
        for (Pez pez : peces)
          if (pez.isVida() == false) {
            tanque.limpiarPeces();
          }
      }
    }
    indice++;
  }
}
```

Método para limpiar los peces muertos de la piscifactoria

```
public void emptyTank() {
  int indice = 0;
  int valor = selectPisc() - 1;

  for (Piscifactoria piscifactoria : piscifactorias) {
    if (indice == valor) {
      for (Tanque tanque : tanques) {
        for (Pez pez : peces)
          tanque.limpiarPeces();
      }
    }
    indice++;
  }
}
```

Método para quitar todos los peces de una piscifactoria

```
  break;

case 2:
  System.out.println(x:"a. Piscifactoria");
  System.out.println(x:"b. Almacén Central");
  System.out.print(s:"Seleccione una opción: ");
  val3 = newScan.nextLine().trim();

  switch (val3) {
    case "a":
      System.out.println(x:"i. Comprar tanque.");
      System.out.println(x:"ii. Aumentar almacén de comida.");
      System.out.print(s:"Seleccione una opción: ");
      String val4 = newScan.nextLine().trim();

      switch (val4) {
        case "i":

          int valor = selectPisc() - 1;

          if(valor<1 || valor>this.piscifactorias.size()){
            this.piscifactorias.get(valor).addTank(this.piscifactorias.get(valor).obtenerCapacidadMaximaTanq());
          }
          break;

        case "ii":
          int valor2 = selectPisc() - 1;
          if(valor2<1 || valor2>this.piscifactorias.size()){
            this.piscifactorias.get(valor2).upgradeFood();
          }
          break;

        default:
          System.out.println(x:"Opción no válida");
          break;
      }
      break;

    case "b":
      System.out.println(x:"i. Aumentar Capacidad");
      System.out.print(s:"Seleccione una opción: ");
      String val5 = newScan.nextLine().trim();
      switch (val5) {
        case "i":
          if(!this.creado){
            System.out.println(x:"Sin Almacen");
          }else{
            AlmacenCentral.getInstance().upgradeAlmacen();
          }
          break;
        default:
          break;
      }
      break;

    default:
      System.out.println(x:"Opción no válida");
      break;
  }
  break;

default:
  System.out.println(x:"Opción no válida");
  break;
}
```

```
            break;

          case "b":
            if(!this.creado){
              if (Monedero.getInstance().getMonedas() >= 2000) {
              AlmacenCentral.getInstance();
              Monedero.getInstance().comprar(cantidad:2000);
              this.creado=true;
              }
            }
            break;

          default:
            System.out.println(x:"Opción no válida");
            break;
        }
```

Metodo para mejorar la infraestructura de nuestra piscifactoria