Manual Técnico

Proyecto Invernadero II – Programación

IES Armando Cotarelo Valledor

1º Desenvolvemento de aplicacións web

Christian Alvarado

Índice

1. Introducción

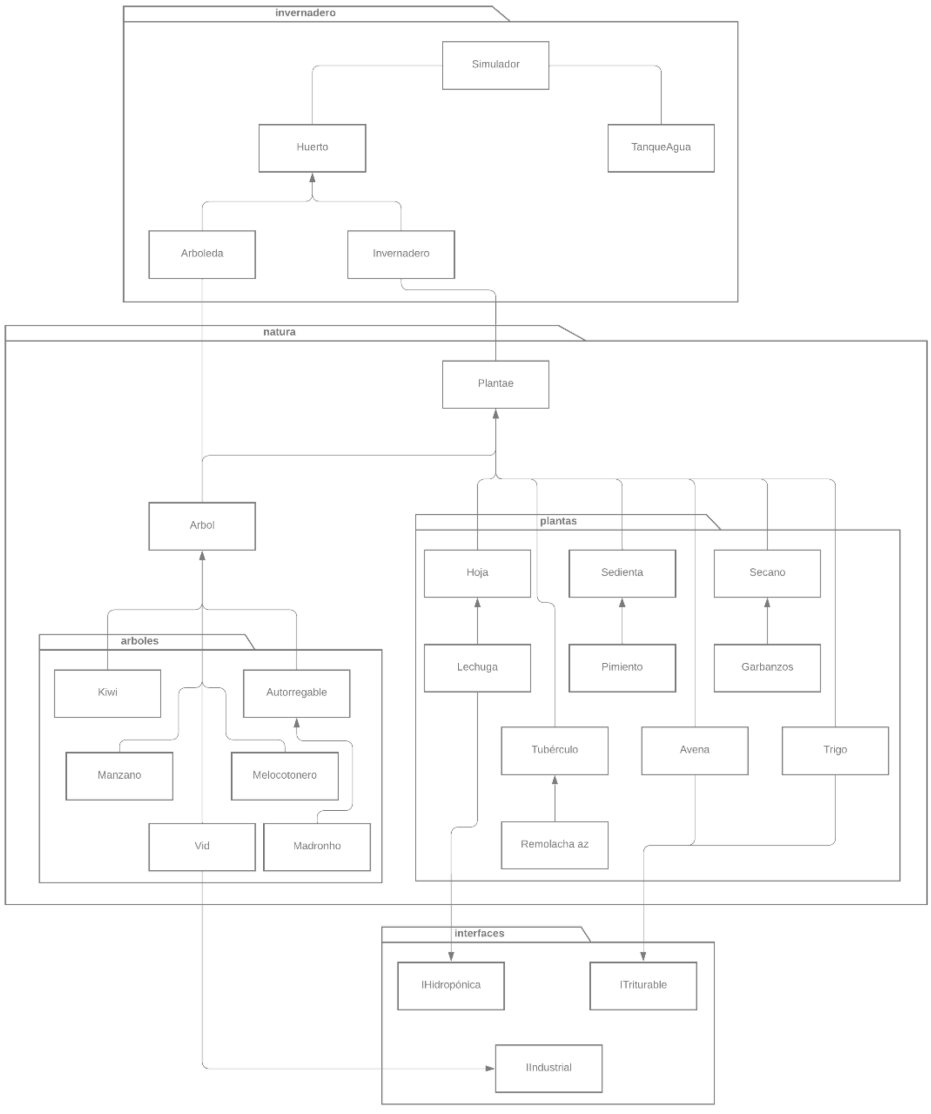
El propósito de este trabajo es crear una aplicación que simule a un conjunto de invernaderos, con la posibilidad de crear y gestionar plantas en cada uno de ellos.

Esta es la segunda versión del trabajo, añadiéndose estas principales mejoras:

- Se añade una serie de cultivos preestablecidos.
- Se añade un nuevo edificio de plantación, la arboleda.
- Se añade un nuevo edificio de mejora, el molino.
- Se añade un nuevo tipo de cultivo, los árboles frutales.
- Se añade un sistema de monedas, costes y ganancias.
- Se mejora el sistema de información, con un informe desglosado de cosecha por planta y un glosario de plantas.
- Se añade una nueva mejora al tanque de agua, los aspersores, que permite regar de forma automática.

2. Esquema general

El siguiente diagrama representa a las clases del programa:



Cada uno con sus características y operaciones, que se explicarán a lo largo de este documento.

3. Paquetes

Las clases se han dividido en dos primeros paquetes, permitiendo su ampliación a futuros cambios o versiones.

3.1. Paquete Invernadero

Contiene Arboleda, Huerto, Invernadero, Simulador y Tanque de Agua.

3.1.1. Huerto

Clase abstracta que representa un edificio, con capacidad para plantas (Clase Plantae) y/o árboles (Clase Arbol) y su gestión sobre esas clases.

3.1.1.1. Atributos

`protected final String nombreEdif;` – El nombre del
`protected Plantae[] plantas;` – El Array de plantas/árboles
`protected int nivel;` – Nivel del edificio (Máximo 10)
`private final String tipo;` – El tipo del edificio (Arboleda o Invernadero)
`protected final String natura;` – El tipo de natura que almacenará (Plantas o árboles)

3.1.1.2. Detalles del Constructor

```
protected Huerto(String nombreEdif, String tipo, String natura) {
        this.nombreEdif = nombreEdif;
        this.natura = natura;
        this.plantas = new Plantae[10];
        this.nivel = 1;
        this.tipo = tipo;
}
```

Constructor exclusivo para herencia parametrizado. Se le asigna el nombre del edificio, el tipo de edificio, la natura, y el Array de Plantae inicializa con diez espacios.

Parámetros:

`nombreEdif` – El nombre del invernadero.
`tipo` – El tipo de edificio
`natura` – El tipo de natura

3.1.1.3. Método getName

```
public String getName() {
        return this.nombreEdif;
}
```

Método para obtener el nombre del edificio.

3.1.1.4. Método getNum

```
public int getNum() {
        int naturaPlantada = 0;
        for (Plantae planta : plantas) {
```

```
            if (planta != null) {
                naturaPlantada++;
            }
        }
        return naturaPlantada;
}
```

Verifica que las posiciones del Array no sean nulas antes de aumentar el contador de plantas plantadas.

3.1.1.5. Método getTiles

```
public int getTiles() {
        return this.plantas.length;
}
```

Método que obtiene el número de espacios totales del Array de plantas.

3.1.1.6. Método getAlive

```
public int getAlive() {
        int naturaViva = 0;
        for (Plantae planta : plantas) {
            if (planta != null && !planta.isDead()) {
                naturaViva++;
            }
        }
        return naturaViva;
}
```

El número de plantas/árboles vivos. Recorre el Array verificando que la posición no sea nula y llamando al método `isDead` de Planta negado para comprobar que esté viva. Si está viva, aumenta el contador en +1.

3.1.1.7. Método getMature

```
public int getMature() {
        int naturaMad = 0;
        for (Plantae planta : plantas) {
            if (planta != null && planta.isMature()) {
                naturaMad++;
            }
        }
        return naturaMad;
}
```

Obtiene el número de plantas maduras. Se recorre el Array de plantas comprobando que no esté vacío y llamando al método `isMature` de la clase Planta. Si cumple con la condición, suma el tamaño del contador.

3.1.1.8. Método getWatered

```
public int getWatered() {
        int naturaRegada = 0;
        for (Plantae planta : plantas) {
            if (planta != null && planta.isWatered()) {
                naturaRegada++;
            }
        }
        return naturaRegada;
}
```

Este método devuelve el número de plantas/árboles regados; es posible porque se verifica mediante un bucle, que no sea null y además llamando al método `isWatered` de la clase Planta, que este devuelve true o false si está regada o no.

3.1.1.9. Método getLevel

```
public int getLevel() {
        return this.nivel;
}
```

3.1.1.10. Método showStatus

```
public abstract void showStatus();
```

Método abstracto, al implementarlo, deberá mostrar por pantalla toda la información del edificio.

3.1.1.11. Método showTileStatus

```
public void showTileStatus() {
        int contador = 0;
        for (Plantae planta : plantas) {
            if (planta != null) {
                System.out.println("\n|" + (++contador) + "|");
                planta.showStatus();
            }
        }
        if (contador == 0) {
            System.out.println("\nNo hay " + this.natura + " en " +
this.nombreEdif);
        }
}
```

Método que muestra el estado de cada una de las plantas o árboles. Mediante un bucle se recorre el Array de plantas, y con un condicionante, de que la posición no sea nula.

Se imprime el estado de cada una de las plantas llamando al método `showStatus` de la clase Plantae. Además, se ha implementado un contador para cada planta/árbol, que aumenta cada vez que una planta no sea nula.

3.1.1.12. Método showCapacity

```
public void showCapacity() {

        System.out.println(...);
}
```

Muestra la ocupación del edificio. El porcentaje es calculado obteniendo el número de plantas que hay multiplicado por 100 y luego esto es dividido entre el tamaño de Array; que es el número de espacios disponibles. Se concatena un "%".

3.1.1.13. Método waterCrops

```
public int waterCrops() {
        int contador = 0;
        for (Plantae planta : plantas) {
            if (planta != null && planta.water()) {
                contador++;
            }
        }
        return contador;
}
```

Este método riega todas las plantas y obtiene el número de plantas regadas.

3.1.1.14. Método waterCrops (Parametrizado)

```
public int waterCrops(TanqueAgua tanqueAgua) {
        int aguaDisponible = tanqueAgua.getWater();
        int plantasRegadas = 0;
        int aguaGastada = 0;

        for (Plantae planta : plantas) {
            if (planta != null && !planta.isWatered()) {
                if (aguaDisponible > 0) {
                    if (planta instanceof Sedienta && aguaDisponible > 1
&& planta.water()) {
                        aguaDisponible -= 2;
                        aguaGastada += 2;
                        plantasRegadas++;
                    } else if (aguaDisponible > 0 && planta.water()) {
                        if (planta instanceof Secano) {
                            Random rand = new Random();
                            if (rand.nextInt(0,4) > 0) {
                                aguaDisponible--;
                                aguaGastada++;
                            }
                        } else {
                            aguaDisponible--;
                            aguaGastada++;
                        }
                        plantasRegadas++;
```

```
                    }

                }
            }
        }
        tanqueAgua.restWater(aguaGastada);
        return plantasRegadas;
}
```

Método que riega las plantas si una planta no está previamente regada, y según la disponibilidad del agua de un tanque.

Se comprueba si la planta es de tipo Sedienta (consume dos de agua) o si es Secano (25% de no consumir agua).

Parámetros:

`tanqueAgua` – Tanque con cierto número de agua.

3.1.1.15. Método growCrops

```
public void growCrops() {
        for (Plantae planta : plantas) {
            if (planta != null) {
                planta.grow();
            }
        }
 }
```

Este método hace crecer a cada una de las plantas/árboles, invocando al método `grow` de la clase Plantae. La planta aumenta de día sólo si no está muerta.

3.1.1.16. Método upgrade

```
public void upgrade() {
        if (this.nivel <= 10) {
            Plantae[] newArray = Arrays.copyOf(this.plantas,
this.plantas.length + 10);
            this.plantas = newArray;
            System.out.println("\n" + this.tipo + " " + this.nombreEdif
                    + " mejorado. Su capacidad ha aumentado en 10 hasta
un total de " + this.plantas.length + ".");
            this.nivel++;
        } else {
            System.out.println("\nNo es posible mejorar este/esta " +
this.natura + ". Ha llegado a su nivel máximo.");
        }
}
```

Se mejora el edificio, aumentando en 10 la capacidad del Array de plantas. Se ha utilizado un método de la clase Arrays, `copyOf`. El nuevo Array es igualado con el anterior, además de añadir diez posiciones, y esto sin afectar al contenido incial del Array. Posteriormente, se

muestra un mensaje por pantalla de la mejora. Si el edificio llega al nivel máximo, no permitirá aumentar más el nivel.

3.1.1.17. Método addPlanta

```
public void addPlanta(Plantae planta) {
        for (int i = 0; i < this.plantas.length; i++) {
            if (plantas[i] == null) {
                plantas[i] = planta;
                break;
            }
        }
}
```

Este método añade una planta dada al Array de plantas.

Parámetros:

`planta` – Objeto de la clase Plantae.

3.1.1.18. Método harvest

```
public int[] harvest() {
        int contadorProductos = 0;
        int productosRecolectados = 0;
        int naturaRecolectada = 0;
        int naturaMuerta = 0;
        int monedasNaturaTriturables = 0;
        for (Plantae planta : this.plantas) {
            if (planta != null && !planta.isDead() && planta.isMature())
{
                contadorProductos = planta.harvest();
                Simulador.stats.registrarCosecha(planta.getName(),
contadorProductos);
                Simulador.monedas.sumarMonedas(contadorProductos *
planta.getGain());
                if (planta instanceof ITriturable) {
                    monedasNaturaTriturables += planta.getGain();
                }
                productosRecolectados += contadorProductos;
                if (planta.getCiclo() == -1) {
                    naturaMuerta++;
                }
                naturaRecolectada++;
            }
        }
        return new int[] { productosRecolectados, naturaRecolectada,
monedasNaturaTriturables, naturaMuerta };
}
```

Método que recolecta los productos de las plantas que estén maduras y las planta que han sido recolectadas en el proceso. Si una planta es Triturable, las monedas de esa planta se

almacenan además en otra variable. Si la planta no tiene un ciclo de días luego de ser recolectada, pasará a contar como natura muerta.

Devuelve un Array con los productos cosechados en la posición 0 y las plantas recolectadas en la posición 1, las monedas de la natura que es triturable en la posición 2, y la natura muerta en la posición 3.

3.1.1.19. Método plow

```
public abstract int plow();
```

Este método abstracto, que es implementado de forma diferente según el tipo de edificio; ese elimina todas las plantas muertas del Array de plantas. Devuelve el número de plantas que han sido eliminadas.

3.1.1.20. Método unroot

```
public abstract void unroot();
```

Método abstracto que elimina todas las plantas del Array de plantas.

3.1.1.21. Método toString

```
public String toString() {
        return this.tipo + " Nv. " + this.nivel + " - " +
this.nombreEdif + "[" + this.getAlive() + "/" + this.getNum()
                + "/" + this.plantas.length + "]";
}
```

3.1.2. Arboleda

Clase que extiende de Huerto; representa el terreno en el cual se plantan árboles.

3.1.2.1. Atributos

Hereda todos los atributos de la clase Huerto y se agrega un atributo adicional.

`private boolean riegoPorGoteo;` – Sistema de riego por goteo, es único por arboleda

3.1.2.2. Detalles del constructor

```
public Arboleda(String nombreArboleda) {
        super(nombreArboleda, "Arboleda", "Árboles");
        this.riegoPorGoteo = false;
}
```

Constructor parametrizado. Con super obtiene el constructor de la Clase Huerto.

Parámetros:

`nombreArboleda` – El nombre de la arboleda

3.1.2.3. Método plow

```
public int plow() {
        int plantasEliminadas = 0;
```

```
        for (int i = 0; i < plantas.length; i++) {
            if (super.plantas[i] != null && plantas[i].isDead()) {
                if (((Arbol) super.plantas[i]).getGrowed()) {
                    int precioMedio = super.plantas[i].getPrice() / 2;
                    Simulador.monedas.sumarMonedas(precioMedio);
                }
                plantasEliminadas++;
                plantas[i] = null;
            }
        }
        return plantasEliminadas;
}
```

3.1.2.4. Método unroot

```
public void unroot() {
        for (int i = 0; i < plantas.length; i++) {
            if (plantas[i] != null) {
                int precioMedio = super.plantas[i].getPrice() / 2;
                if (Simulador.monedas.comprobarMonedas(precioMedio)) {
                    Simulador.monedas.restMonedas(precioMedio);
                    plantas[i] = null;
                } else {
                    System.out.println("No se ha podido eliminar el
árbol " + super.plantas[i].getName()
                            + ". No hay monedas suficientes.");
                }
            }
        }
}
```

3.1.2.5. Método isRiegoPorGoteo

```
public boolean isRiegoPorGoteo() {
        return this.riegoPorGoteo;
}
```

Método que devuelve si el sistema de riego por goteo está disponible.

3.1.2.6. Método setRiegoPorGoteo

```
public void setRiegoPorGoteo() {
        this.riegoPorGoteo = true;
}
```

3.1.2.7. Método waterCrops

```
@Override
    public int waterCrops(TanqueAgua tanqueAgua) {
        Random rand = new Random();
        int aguaDisponible = tanqueAgua.getWater();
        int arbolesRegados = 0;
```

```
        int aguaGastada = 0;

        for (Plantae planta : plantas) {
            if (planta != null && !planta.isWatered()) {
                if (aguaDisponible > 0 && planta.water()) {
                    if (this.riegoPorGoteo) {
                        // si es true se gasta agua, si es false, se
ahorra agua
                        if (rand.nextBoolean()) {
                            aguaDisponible--;
                            aguaGastada++;
                        }
                    } else if (!(planta instanceof Autorregable)) {
                        aguaDisponible--;
                        aguaGastada++;
                    }
                    arbolesRegados++;
                }
            }
        }
        tanqueAgua.restWater(aguaGastada);
        return arbolesRegados;
}
```

Método que riega los árboles si un árbol no está previamente regado, y según la disponibilidad del agua del tanque. Si se dispone del sistema de riego por goteo, un 50% de probabilidad de no consumir agua.

3.1.2.8. Método showStatus

```
@Override
    public void showStatus() {
       System.out.println(...);

            ...

}
```

3.1.3. Invernadero

Clase que extiende de Huerto; representa a un invernadero, con capacidad para plantas (Clase Plantae) y gestionar la clase.

3.1.3.1. Atributos

Todos los de Huerto.

3.1.3.2. Detalles del Constructor

```
public Invernadero(String nombreInv) {
    super(nombreInv, "Invernadero", "Plantas");
}
```

Con super llama al constructo de la clase Huerto y pasa por parámetros el nombre del invernadero.

`nombreInv` – El nombre de la arboleda

3.1.3.3. Método plow

```
@Override
public int plow() {
    int plantasEliminadas = 0;
    for (int i = 0; i < plantas.length; i++) {
        if (plantas[i] != null && plantas[i].isDead()) {
            plantasEliminadas++;
            plantas[i] = null;
        }
    }
    return plantasEliminadas;
}
```

3.1.3.4. Método unroot

```
@Override
public void unroot() {
    for (int i = 0; i < plantas.length; i++) {
        if (plantas[i] != null) {
            plantas[i] = null;
        }
    }
}
```

3.1.3.5. Método showStatus

```
@Override
public void showStatus() {
    System.out.println(...);
      ...
}
```

3.1.4. Simulador

Clase que contiene la lógica general del proyecto.

3.1.4.1. Atributos

```
private int numDias;
private Invernadero[] invernaderos;
private Arboleda[] arboledas;
private TanqueAgua;
private String nombre;
private Scanner sc;
public static SistemaMonedas monedas;
```

```
private boolean molino;
private boolean aspersor;
public static Estadisticas stats;
private final Plantae[] plantas = { ... };
```

numDias – Contador para los días que han pasado.
invernaderos – Array de los invernaderos. Empieza con cinco espacios.
arboledas – Array de las arboledas. Empieza con cinco espacios.
tanqueAgua – Representa al tanque de agua.
nombre – Nombre de la entidad/empresa/partida.
monedas – Monedas del Simulador.
molino – Si se dispone de este edificio. Es una compra única.
aspersor – Mejora única que riega de manera automática al pasar de día.
stats – Estadísticas del Simulador (natura recolectada; productos cosechados; etc.…)
plantas – Array con las plantas y árboles disponibles en el Simulador.
sc – Objeto de la clase Scanner. Obtiene las interacciones por teclado del usuario.

3.1.4.2. Detalles del Constructor

```
public Simulador();
```

Constructor por defecto, no declarado y sin parámetros.

3.1.4.3. Método init

```
public void init() {
    this.numDias = 0;
    this.invernaderos = new Invernadero[5];
    this.arboledas = new Arboleda[5];
    this.tanqueAgua = new TanqueAgua();
    this.sc = new Scanner(System.in);
    monedas = new SistemaMonedas();
    this.molino = false;
    this.aspersor = false;
    stats = new Estadisticas(new String[] { ... });
    System.out.print("Introduce el nombre de ... ");
    this.nombre = sc.nextLine();
    System.out.print("Introduce el nombre del primer invernadero: ");
    Invernadero inv1 = new Invernadero(sc.nextLine());
    this.invernaderos[0] = inv1;
    System.out.println("\nBienvenido, " + this.nombre + ".");
}
```

Método que inicializa todo el programa solicitando el nombre de la sesión y el nombre del primer invernadero al usuario.

3.1.4.4. Método menu

```
public void menu() {
    System.out.println("========= Menú principal =========");
    System.out.println("1.- Estado general");
```

```
    System.out.println("2.- Estado cultivos");
    System.out.println("3.- Informes");
    System.out.println("4.- Naturpedia");
    System.out.println("5.- Pasar día");
    System.out.println("6.- Recolectar agua");
    System.out.println("7.- Regar");
    System.out.println("8.- Plantar");
    System.out.println("9.- Cosechar");
    System.out.println("10.- Desbrozar");
    System.out.println("11.- Arrancar");
    System.out.println("12.- Mejorar");
    System.out.println("13.- Pasar varios días");
    System.out.println("14.- Salir");
    System.out.print("\nSeleccione una opción: ");
}
```

Este método representa un menú de opciones, del cual el usuario podrá elegir.

3.1.4.5. Método menuInv

```
public int menuInv() {
    int contador = 1;
    System.out.println();
    System.out.println(" ... ");
    System.out.println(" ... ");
    System.out.println("0.- Cancelar");
    for (Invernadero invernadero : this.invernaderos) {
        if (invernadero != null) {
            System.out.println( ... );
        }
    }
    return contador;
}
```

Este método lista la información general de los invernaderos. Un contador se irá incrementando cuando exista un invernadero. De momento no se eliminarán invernaderos, por lo que usar el contador (-1) obtendrá el índice del invernadero en el Array. Se implementa esto para ahorrar un bucle adicional. Devuelve el número de invernaderos que no son nulos.

3.1.4.6. Método selectInv

```
public int selectInv() {
    return this.select(this.menuInv());
}
```

Este método muestra el menú de invernaderos y devuelve un número entero que representa la selección del usuario. Se gestiona la posibilidad de errores mediante un método privado genérico.

La opción 0 es usada en caso que el usuario se rectifique en usar el método complementario. (Esto es gestionado en los métodos que usen `selectInv`). Devuelve un número introducido por teclado.

3.1.4.7. Método menuArb

```
public int menuArb() {
    int contador = 1;
    System.out.println();
    System.out.println(" ... ");
    System.out.println(" ... ");
    System.out.println("0.- Cancelar");
    for (Arboleda arboleda : this.arboledas) {
        if (arboleda != null) {
            System.out.println( ... );
        }
    }
    return contador;
}
```

Método que lista los árboles. Tiene el mismo funcionamiento que `menuInv`.

3.1.4.8. Método selectArb

```
public int selectArb() {
    return this.select(this.menuArb());
}
```

Método que permite seleccionar una arboleda.

3.1.4.9. Método showGeneralStatus

```
public void showGeneralStatus() {
    System.out.println();
    System.out.println("========== Invernaderos ==========");
    for (Invernadero invernadero : this.invernaderos) {
        if (invernadero != null) {
            invernadero.showStatus();
        }
    }
    System.out.println("\n========== Arboledas ==========");
    if (this.invArbExisten(2) > 0) {
        for (Arboleda arboleda : this.arboledas) {
            if (arboleda != null) {
                arboleda.showStatus();
            }
        }
    } else {
        System.out.println("Aún no se ha adquirido ninguna arboleda.");
    }

    System.out.println("--------------------------------");
```

```
    this.tanqueAgua.showStatus();
    System.out.println("Día actual: " + this.numDias);
    System.out.println("Molino: " ... );
    System.out.println("Aspersor: " ... );
    System.out.println("Monedas: " + monedas.toString());
}
```

Método que muestra el estado de todos los invernaderos, el tanque de agua el día actual, el número de plantas recolectadas y el número de productos cosechados.

3.1.4.10. Método showSpecificStatus

```
public void showSpecificStatus() {
    int contInv = 0;
    int contArb = 0;
    int select;
    System.out.println("\n====== Estado plantas/árboles ======");
    System.out.println("0.- Cancelar");
    for (Invernadero inv : this.invernaderos) {
        if (inv != null) {
            System.out.println((++contInv) + ".- [I] " + inv.getName());
        }
    }
    if (this.invArbExisten(2) > 0) {
        contArb = contInv;
        for (Arboleda arb : this.arboledas) {
            if (arb != null) {
                System.out.println( ... );
            }
        }
    } else {
        System.out.println("[No hay arboledas disponibles]");
    }
    if (contArb == 0) {
        select = this.select(contInv);
    } else {
        select = this.select(contArb);
    }
    if (select != 0) {
        if (select <= contInv) {
            this.invernaderos[select - 1].showTileStatus();
        } else {
            this.arboledas[(select - 1) - contInv].showTileStatus();
        }
    }
}
```

Método que muestra el estado de todas las plantas/árboles de un invernadero/arboleda seleccionada. Si no hay arboledas, se muestra un mensaje indicándolo.

3.1.4.11. Método showStats

```
public void showStats() {
    System.out.println("\n----- Informe -----");
    stats.mostrar();
}
```

Este método muestra un desglose de las plantas y árboles del sistema con información general de los productos cosechados y la harina generada.

3.1.4.12. Método showNatura

```
public void showNatura() {
    int contador = 1;

    System.out.println("\n --------- NaturPedia ---------");
    System.out.println("0.- Cancelar");
    System.out.println("1.- Consultar plantas");
    System.out.println("2.- Consultar árboles");
    int select = this.select(2);

    switch (select) {
        case 1:
            System.out.println(" ... ");
            System.out.println("0.- Cancelar");
            for (Plantae plant : this.plantas) {
                if (!(plant instanceof Arbol)) {
                    System.out.println( ... );
                }
            }
            select = this.select(contador);
            this.plantas[select - 1].showInfoNatura();
            break;
        case 2:
            System.out.println( ... );
            System.out.println("0.- Cancelar");
            for (Plantae plant : this.plantas) {
                if (plant instanceof Arbol) {
                    System.out.println( ... );
                }
            }
            select = this.select(contador);
            this.plantas[(this.contadorPlantas()) + (select -
1)].showInfoNatura();
            break;
        default:
            break;
    }
}
```

Método que permite consultar la información de cada planta y árbol. Primero, muestra un menú para consultar árboles o plantas. Luego, una lista con las plantas o árboles disponibles. Por último, muestra la información de la planta o árbol.

3.1.4.13. Método nextDay

```
public void nextDay() {
    for (Invernadero invernadero : this.invernaderos) {
        if (invernadero != null) {
            invernadero.growCrops();
        }
    }
    for (Arboleda arboleda : this.arboledas) {
        if (arboleda != null) {
            arboleda.growCrops();
        }
    }
    if (this.aspersor) {
        this.waterInv();
        this.waterArb();
    }
    this.numDias++;
    System.out.println("\n¡Se ha aumentado un día!");
}
```

Aumenta en un día a todas las plantas vivas de los invernaderos y arboledas, al igual que el día del simulador. Si se ha comprado el aspersor, regará de manera consiguiente los invernaderos y las arboledas.

3.1.4.14. Método addWater

```
public void addWater() {
    int select = -1;
    int agua = -1;

    System.out.println("\n------------- Recolectar agua -------------");
    System.out.println("0.- Cancelar");
    System.out.println("1.- Añadir 1 unidad de agua");
    System.out.println("2.- Añadir 5 unidades de agua");
    System.out.println("3.- Añadir 10 unidades de agua");
    System.out.println("4.- Llenar el tanque de agua");

    select = this.select(4);
    if (select != 0) {
        switch (select) {
            case 1:
                if (Simulador.monedas.comprobarMonedas(1) &&
this.tanqueAgua.addWater(1)) {
                    Simulador.monedas.restMonedas(agua = 1);
                }
                break;
```

```
            case 2:
                if (Simulador.monedas.comprobarMonedas(5) &&
this.tanqueAgua.addWater(5)) {
                    Simulador.monedas.restMonedas(agua = 5);
                }
                break;
            case 3:
                if (Simulador.monedas.comprobarMonedas(10) &&
this.tanqueAgua.addWater(10)) {
                    Simulador.monedas.restMonedas(agua = 10);
                }
                break;
            case 4:
                int dif = this.tanqueAgua.getMax() -
this.tanqueAgua.getWater();
                int descuento = dif / 10;
                if (Simulador.monedas.comprobarMonedas(dif - descuento))
{
                    Simulador.monedas.restMonedas(dif - descuento);
                    this.tanqueAgua.addWater(dif);
                    agua = dif;
                }
                break;
        }
        if (agua != -1) {
            System.out.println("\nAñadidos " + agua + " de agua");
        }
        this.tanqueAgua.showStatus();
    }
}
```

Este método añade cierta cantidad de agua al tanque según sea elegido por el usuario (mediante un menú); luego muestra el estado del tanque de agua.

Cada unidad de agua cuesta 1 moneda, excepto si se llena el tanque (`case 4`), el cual tiene 1 moneda de ahorro por cada 10 de agua.

Para llenar el tanque de agua, se ha hecho un sencillo cálculo. Se obtiene la diferencia del agua máxima del tanque con el agua disponible.

3.1.4.15. Método waterCrops

```
public void waterCrops() {
    int aguaDisponible = this.tanqueAgua.getWater();
    if (aguaDisponible > 0) {
        int select;
        System.out.println("0.- Cancelar");
        System.out.println("1.- Regar invernaderos");
        if (this.invArbExisten(2) > 0) {
            System.out.println("2.- Regar arboledas");
            select = select(2);
        } else {
            select = this.select(1);
        }

        switch (select) {
            case 1:
                this.waterInv();
                break;
            case 2:
                this.waterArb();
                break;
            default:
                break;
        }
    } else {
        System.out.println(" ... ");
    }
}
```

Método riega las plantas o árboles de un huerto según los recursos del tanque y la selección del usuario. Muestra por pantalla las plantas o árboles que han sido regadas. Si no hay arboledas, no se muestra la opción para regarlas.

3.1.4.16. Método plant

```
public void plant() {
    int select = -1;
    int selectHuerto = -1;
    int contador = 1;
    /** Vefifica si hay al menos una arboleda disponible. **/
    boolean isArb = this.invArbExisten(2) != 0;

    System.out.println("\n========= Nueva plantación =========");
    System.out.println("0.- Cancelar");
    System.out.println("1.- Seleccionar una planta");
    if (isArb) {
        System.out.println("2.- Seleccionar un árbol");
    }

    if (isArb) {
```

```
    select = this.select(2);
} else {
    select = this.select(1);
}

if (select != 0) {
    switch (select) {
        case 1:
            System.out.println( ... );
            System.out.println("0.- Cancelar");
            for (Plantae plant : this.plantas) {
                if (!(plant instanceof Arbol)) {
                    System.out.println( ... );
                }
            }
            select = this.select(contador);
            if (select != 0) {
                if ( comprobar monedas ... ){
                    selectHuerto = this.selectInv();
                    final Invernadero inv = ...;
                    if ((inv.getTiles() - inv.getNum()) > 0) {
                        ...
                        System.out.println( ... );
                        inv.showCapacity();
                    } else {
                        System.out.println( ... );
                    }
                }

            }
            break;
        case 2:
            System.out.println( ... );
            System.out.println("0.- Cancelar");
            for (Plantae plant : this.plantas) {
                if (plant instanceof Arbol) {
                    System.out.println( ... );
                }
            }
            select = this.select(contador);
            if (select != 0) {
                if ( comprobar monedas ... ) {
                    ...
                }
            }
            break;
        default:
            break;
    }
```

```
    }
}
```

Método que permite seleccionar si agregar una planta o un árbol a uno de los edificios de plantación. La opción de árboles sólo es mostrada si existe, al menos, una arboleda en el sistema.

Agrega una planta o árbol, listando los disponibles, a un invernadero/arboleda específica sólo si hay espacio en dicho invernadero/arboleda. Luego es mostrado un mensaje con el estado del invernadero seleccionado.

3.1.4.17. Método harvest

```
public void harvest() {
    int select;
    System.out.println("\n-------- Recolectar productos --------");
    System.out.println("0.- Cancelar");
    System.out.println("1.- Cosechar plantas");
    if (this.invArbExisten(2) > 0) {
        System.out.println("2.- Cosechar árboles");
        select = this.select(2);
    } else {
        select = this.select(1);
    }

    if (select != 0) {
        final int monedasActuales = Simulador.monedas.getMonedas();
        int productosRecolectados = 0;
        int naturaRecolectada = 0;
        int naturaMuerta = 0;
        int monedasNaturaTriturables = 0;
        int[] harvest;

        switch (select) {
            case 1:
                if (naturaMadura(1) > 0) {
                    for (Invernadero invernadero : invernaderos) {
                        if (invernadero != null) {
                            harvest = invernadero.harvest();
                            productosRecolectados += harvest[0];
                            naturaRecolectada += harvest[1];
                            monedasNaturaTriturables += harvest[2];
                            naturaMuerta += harvest[3];
                            if (this.molino && monedasNaturaTriturables
> 0) {
                                Simulador.stats.registrarHarina(monedasN
aturaTriturables);
                                Simulador.monedas.restMonedas(monedasNat
uraTriturables);
```

```
                                Simulador.monedas.sumarMonedas((int)
(monedasNaturaTriturables * 1.20));
                            }
                        }
                    }

                    System.out.println("\n------------- Cosecha de
productos -------------");
                    System.out.println(naturaRecolectada + " plantas han
producido " + productosRecolectados + " productos.");
                    System.out.println("Han muerto " + naturaMuerta + "
plantas en el proceso.");
                    System.out.println(
                            "Se han ganado " +
(Simulador.monedas.getMonedas() - monedasActuales) + " monedas.");
                } else {
                    System.out.println("\nNo hay plantas maduras para
recolectar.");
                }
                break;
            case 2:
                if (naturaMadura(2) > 0) {
                    for (Arboleda arb : this.arboledas) {
                        if (arb != null) {
                            harvest = arb.harvest();
                            productosRecolectados += harvest[0];
                            naturaRecolectada += harvest[1];
                            naturaMuerta += harvest[3];
                            ...
                        }
                    }

                    System.out.println( ... );
                        ...
                } else {
                    System.out.println("\nNo hay árboles maduros para
recolectar.");
                }
                break;
            default:
                break;

        }
    }
}
```

Método que permite recolectar todas las plantas de los invernaderos o árboles de las arboledas que estén maduros y obtiene los productos de cada uno, luego es mostrado mensaje de las plantas/árboles afectados y los productos obtenidos.

Se comprueban que el número de plantas maduras (o árboles maduros, si es el caso) sea mayor que 0 y si es correcto, por cada invernadero o arboleda es llamado al método `harvest` que devuelve un Array de cuatro índices donde: 0 son los productos recolectados, 1 las plantas o árboles recolectados, 2 si la planta/árbol es de tipo triturable se almacenan, y 3 las plantas o árboles que han muerto tras recolectarlos. Luego si son Triturables y se ha comprado el molino, son agregadas a las estadísticas de la harina y aumentando el 20% de ganancias en el conjunto del edificio. Finalmente se muestra un mensaje resumido sobre la cosecha de productos.

3.1.4.18. Método plow

```
public void plow() {
    int select;
    int naturaEliminada = 0;
    System.out.println("\n-------- Eliminar plantas/árboles muertos ----
----");
    System.out.println("0.- Cancelar");
    System.out.println("1.- Desbrozar plantas");
    if (this.invArbExisten(2) > 0) {

        System.out.println("2.- Desbrozar árboles");
        select = this.select(2);
    } else {
        select = this.select(1);
    }

    System.out.println("\n------------ Eliminación de plantas muertas -
-----------");
    switch (select) {
        case 1:
            if (this.naturaViva(1) < this.naturaTotal(1)) {

                for (Invernadero inv : this.invernaderos) {
                    if (inv != null) {
                        naturaEliminada += inv.plow();
                    }
                }
                System.out.println("Se han eliminado un total de " +
naturaEliminada
                        + " plantas muertas de los invernaderos.");
            } else {
                System.out.println("\nNo hay ninguna planta muerta para
eliminar.");
            }
            break;
        case 2:
            if (this.naturaViva(2) < this.naturaTotal(2)) {

                for (Arboleda arb : this.arboledas) {
                    if (arb != null) {
                        naturaEliminada += arb.plow();
```

```
                    }
                }
                System.out.println(
                        "Se han eliminado un total de " +
naturaEliminada + " árboles muertos de las arboledas.");
            } else {
                System.out.println("\nNo hay ningún árbol muerto para
eliminar.");
            }
            break;
        default:
            break;
    }
}
```

Método que elimina las plantas/árboles muertos de un invernadero/arboleda en concreto. El método `plow` de la clase Huerto, devuelve el número de plantas/árboles que han sido eliminados, recorriendo el Array de plantas, comprobando primero que no sea nulo y que la planta esté muerta. Es sumando a `naturaEliminada` por cada uno de los invernaderos o arboledas.

3.1.4.19. Método unroot

```
public void unroot() {
    int selectInvArb;
    int select;
    System.out.println("\n-------- Arrancar plantas/árboles --------");
    System.out.println("0.- Cancelar");
    System.out.println("1.- Arrancar plantas");
    if (this.invArbExisten(2) > 0) {

        System.out.println("2.- Arrancar árboles");
        selectInvArb = this.select(2);
    } else {
        selectInvArb = this.select(1);
    }

    switch (selectInvArb) {
        case 1:
            select = selectInv();
            if (select > 0) {
                this.invernaderos[selectInvArb - 1].unroot();
                System.out.println("\n------------ Eliminación de
plantas -------------");
                System.out.println(
                        "\nSe han eliminado todas las plantas de " +
invernaderos[selectInvArb - 1].getName());
            }
            break;
        case 2:
```

```
                select = selectArb();
                if (select > 0) {
                    this.arboledas[selectInvArb - 1].unroot();
                    System.out.println( ... );
                    System.out.println( se han eliminado ... );
                }
                break;
            default:
                break;
        }
    }
```

Método que elimina las plantas/árboles de un edificio en concreto. En el caso de los árboles, tiene un coste de la mitad de su precio. Si la selección del usuario es 0, no se ejecutará la estructura interna del método.

3.1.4.20. Método upgrade

```
public void upgrade() {
    int seleccion;
    int select;
    int contadorHuerto;
    int costoInv;
    int costoArb;

    System.out.println("\n------------- Mejoras -------------");
    System.out.println("1.- Comprar edificio");
    System.out.println("2.- Mejorar edificio");
    System.out.println("3.- Mejorar el tanque de agua");
    System.out.println("4.- Cancelar");

    seleccion = this.select(1, 4);

    switch (seleccion) {
        case 1:
            System.out.println("\n--- Comprar edificio ---");
            System.out.println("0.- Cancelar");

            ...

            if (!this.molino) {
                System.out.println("3.- Molino [3000 monedas]");
                select = select(3);
            } else {
                select = select(2);
            }

            switch (select) {
                case 1:
                    if (Simulador.monedas.comprobarMonedas(costoInv)) {
```

```
                System.out.print( ... );
                sc.nextLine();
                Invernadero nuevoInv = ...;

                if (this.invArbNulos(1) == 0) {
                    this.upgradeInvArb(1);
                }

                for ( ... ){
                    ...
                }
                System.out.println( ... );
                Simulador.monedas.restMonedas(costoInv);
            } else {
                System.out.println( ... );
            }
            break;
        case 2:
            if (Simulador.monedas.comprobarMonedas(costoArb)) {

                System.out.print( ... );
                sc.nextLine();
                Arboleda nuevaArb = new Arboleda(sc.nextLine());

                if (this.invArbNulos(2) == 0) {
                    this.upgradeInvArb(2);
                }

                for ( ... ) {
                    ...
                }
                System.out.println( ... );
                Simulador.monedas.restMonedas(costoArb);
            } else {
                System.out.println( ... );
            }
            break;
        case 3:
            if (Simulador.monedas.comprobarMonedas(3000)) {
                Simulador.monedas.restMonedas(3000);
                this.molino = true;
            } else {
                System.out.println( ... );
            }
            break;
        default:
            break;
    }
```

```
            break;
        case 2:
            System.out.println("\n--- Mejorar edificio ---");
            select = ...;

            switch (select) {
                case 1:
                    System.out.println("\n- Mejorar invernadero -");
                    System.out.println("0.- Cancelar");
                    System.out.println("1.- Aumentar tamaño [150
monedas/nivel]");
                    select = this.select(1);

                    if (select != 0) {
                        seleccion = this.selectInv();
                        if (seleccion != 0) {
                            if ( inv < 10 ...) {
                                costoInv = this.invernaderos[seleccion -
1].getLevel() * 150;
                                if ( coste ... ) {
                                    this.invernaderos[seleccion -
1].upgrade();
                                    Simulador.monedas.restMonedas(costoI
nv);
                                } else {
                                    System.out.println( ... );
                                }
                            } else {
                                System.out.println( nivel máx. ...);
                            }
                        }
                    }

                    break;
                case 2:
                    System.out.println("\n- Mejorar arboleda -");
                    System.out.println("0.- Cancelar");
                    System.out.println("1.- Aumentar tamaño [150
monedas/nivel]");
                    System.out.println("2.- Añadir sistema de riego por
goteo [500 monedas]");
                    select = this.select(2);

                    switch (select) {
                        case 1:
                            seleccion = this.selectArb();
                            if (seleccion != 0) {
                                if (this.arboledas[seleccion -
1].getLevel() < 10) {
```

```
                                    costoArb = this.arboledas[seleccion
- 1].getLevel() * 150;
                                    if ( coste ... ) {
                                        this.arboledas[seleccion -
1].upgrade();
                                        Simulador.monedas.restMonedas(co
stoArb);
                                    } else {
                                        System.out.println( ... );
                                    }
                                } else {
                                    System.out.println( nivel máx. ...);
                                }
                            }
                            break;
                        case 2:
                            seleccion = this.selectArb();
                            if (seleccion != 0) {
                                costoArb = 500;
                                if (arb sin sistema rpg ...) {
                                    if ( coste ... ) {
                                        this.arboledas[seleccion -
1].setRiegoPorGoteo();
                                        Simulador.monedas.restMonedas(co
stoArb);
                                    } else {
                                        System.out.println( ... );
                                    }
                                } else {
                                    System.out.println( ... );
                                }
                            }
                            break;
                        default:
                            break;
                    }
                    break;
                default:
                    break;
            }
            break;
        case 3:
            System.out.println("\n--- Mejorar el tanque de agua ---");
            System.out.println("0.- Cancelar");
            System.out.println("1.- Aumentar tamaño [100 monedas]");
            if (!this.aspersor) {
                System.out.println("2.- Añadir aspersores [1000
monedas]");
            select = ...;
```

```
            int costo;

            switch (select) {
                case 1:
                    costo = 100;
                    if ( coste ... ){
                        this.tanqueAgua.upgrade();
                        System.out.println( ... );
                        Simulador.monedas.restMonedas(costo);
                    } else {
                        System.out.println( monedas insuf ... );
                    }
                    break;
                case 2:
                        costo = 1000;
                        if (coste ... ) {
                            this.aspersor = true;
                            System.out.println("\n¡Aspersor comprado!");
                            Simulador.monedas.restMonedas(costo);
                        } else {
                            System.out.println( monedas insuf ... );
                        }
                    break;
                default:
                    break;
            }
            break;
        default:
            break;
    }
}
```

Este método permite ampliar las condiciones del Simulador. Algunas opciones no estarán disponibles si antes no se ha adquirido el objeto indispensable para su mejora. Por ejemplo, no se podrá mejorar una arboleda sin antes haber adquirido una previamente.

3.1.4.21. Método forwardDays

```
public void forwardDays() {
    int select;
    System.out.println("\nIntroduzca cuántos días avanzar en el
Simulador [Entre 1 y 5 días; 0 para cancelar]");
    select = this.select(5);
    if (select != 0) {
        for (int i = 1; i <= select; i++) {
            this.nextDay();
        }
        System.out.println("\n|- Han pasado un total de " + select + "
días -|");
```

```
    }
}
```

Método que permite adelantar entre 1 a 5 días el estado del Simulador.

3.1.4.22. Métodos privados

Se han incluido unos métodos adicionales con funcionalidad específica de otros métodos principales de esta clase.

3.1.4.22.1. Método privado selectInvArb

```
private int selectInvArb() {
    System.out.println("0.- Cancelar");
    System.out.println("1.- Seleccionar un invernadero");
    System.out.println("2.- Seleccionar una arboleda");
    return this.select(2);
}
```

Método privado que permite seleccionar un tipo de edificio del simulador.

3.1.4.22.2. Método privado select (Un parámetro)

```
private int select(int rangoMax) {
    return this.select(0, rangoMax);
}
```

Método que por defecto asigna el rango mínimo en 0. Parámetro: `rangoMax` el rango máximo a evaluar.
Devuelve la selección correcta del usuario.

3.1.4.22.3. Método privado select (Dos parámetros)

```
private int select(int rangoMin, int rangoMax) {
    boolean flag = false;
    int select = -1;
    do {
        try {
            System.out.print("\n· Selecciona una opción: ");
            select = sc.nextInt();
            if (select >= rangoMin && select <= rangoMax) {
                if (select == 0) {
                    System.out.println( ... );
                }
                flag = !flag;
            } else {
                System.out.println("Debe elegir una opción válida.");
            }
        } catch (InputMismatchException e) {
            System.out.println("Debe elegir una opción válida.");
            sc.nextInt();
        }
    } while (!flag);
    return select;
}
```

Método privado que facilita la obtención de un número válido introducido por el usuario. Por defecto se comprueba que sea un valor positivo. Parámetros: `rangoMin` número mínimo que puede tener el valor introducido por teclado; `rangoMax` número máximo que puede tener el valor introducido por teclado. Devuelve el número introducido.

3.1.4.22.4. Método privado waterInv

```
private void waterInv() {
    if (this.naturaTotal(1) > 0) {
        int plantasRegadas = 0;
        for (Invernadero inv : this.invernaderos) {
            if (inv != null && tanqueAgua.getWater() > 0) {
                plantasRegadas += inv.waterCrops(this.tanqueAgua);
            }
        }
        System.out.println("\nSe han regado [" + plantasRegadas + "/" +
this.naturaViva(1)
                + "] plantas de los invernaderos");
    } else {
        System.out.println("\nNo hay ninguna planta para regar");
    }
}
```

Método para regar plantas de los invernaderos.

3.1.4.22.5. Método privado waterArb

```
private void waterArb() {
    if (this.naturaTotal(2) > 0) {
        int arbolesRegados = 0;
        for (Arboleda arb : this.arboledas) {
            if (arb != null && tanqueAgua.getWater() > 0) {
                arbolesRegados += arb.waterCrops(this.tanqueAgua);
            }
        }
        System.out.println("\nSe han regado [" + arbolesRegados + "/" +
this.naturaViva(2)
                + "] árboles de las arboledas");
    } else {
        System.out.println("\nNo hay ningún árbol para regar");
    }
}
```

Método que riega los árboles de las arboledas.

3.1.4.22.6. Método privado contadorPlantas

```
private int contadorPlantas() {
    int contador = 0;
    for (Plantae plant : this.plantas) {
        if (!(plant instanceof Arbol)) {
            contador++;
        }
```

```
    }
    return contador;
}
```

Método que devuelve el número de plantas (no árboles) que están implementadas en el Simulador (en el Array de Simulador).

3.1.4.22.7. Método privado invArbNulos

```
private int invArbNulos(int select) {
    int contador = 0;
    switch (select) {
        case 1:
            for (Invernadero inv : this.invernaderos) {
                if (inv == null) {
                    contador++;
                }
            }
            break;
        case 2:
            for (Arboleda arb : this.arboledas) {
                if (arb == null) {
                    contador++;
                }
            }
            break;
        default:
            break;
    }
    return contador;
}
```

Método que devuelve el número de espacios vacíos en los Arrays de invernaderos o arboledas. Parámetro: `select` 1 para devolver número de espacios en los invernaderos, 2 para las arboledas. Devuelve el número de espacios vacíos (nulos) en el edificio de plantación.

3.1.4.22.8. Método privado invArbExisten

```
private int invArbExisten(int select) {
    int contador = 0;
    switch (select) {
        case 1:
            for (Invernadero inv : this.invernaderos) {
                if (inv != null) {
                    contador++;
                }
            }
            break;
        case 2:
            for (Arboleda arb : this.arboledas) {
                if (arb != null) {
                    contador++;
```

```
                }
            }
            break;
        default:
            break;
    }
    return contador;
}
```

Método que devuelve el número de invernaderos o arboledas que hay en el Simulador.

3.1.4.22.9. Método privado upgradeInvArb

```
private void upgradeInvArb(int select) {
    switch (select) {
        case 1:
            Invernadero[] invs = Arrays.copyOf(this.invernaderos,
this.invernaderos.length + 5);
            this.invernaderos = invs;
            break;
        case 2:
            Arboleda[] arbs = Arrays.copyOf(this.arboledas,
this.arboledas.length + 5);
            this.arboledas = arbs;
            break;
        default:
            break;
    }
}
```

Método que aumenta el tamaño del Array de invernaderos o arboledas en cinco posiciones más (1 para el Array de invernaderos, 2 para el de arboledas). El Array es copiado en uno nuevo con cinco posiciones extra, y luego es reemplazado.

3.1.4.22.10. Método privado naturaViva

```
private int naturaViva(int select) {
    int naturaViva = 0;
    switch (select) {
        case 1:
            for (Invernadero invernadero : this.invernaderos) {
                if (invernadero != null) {
                    naturaViva += invernadero.getAlive();
                }
            }
            break;
        case 2:
            for (Arboleda arb : this.arboledas) {
                if (arb != null) {
                    naturaViva += arb.getAlive();
                }
            }
```

```
            break;
        default:
            break;
    }
    return naturaViva;
}
```

Método que devuelve el número de plantas vivas o el número de árboles vivos de todos los invernaderos o arboledas. El número pasado por parámetro elige los invernaderos o las arboledas (1, 2 respectivamente).

3.1.4.22.11. Método privado naturaTotal

```
private int naturaTotal(int select) {
    int naturaTotal = 0;
    switch (select) {
        case 1:
            for (Invernadero invernadero : invernaderos) {
                if (invernadero != null) {
                    naturaTotal += invernadero.getNum();
                }
            }
            break;
        case 2:
            for (Arboleda arb : this.arboledas) {
                if (arb != null) {
                    naturaTotal += arb.getNum();
                }
            }
            break;
        default:
            break;
    }
    return naturaTotal;
}
```

Método que obtiene el número de plantas/árboles totales de los invernaderos/arboledas.

3.1.4.22.12. Método privado naturaMadura

```
private int naturaMadura(int select) {
    int naturaMadura = 0;
    switch (select) {
        case 1:
            for (Invernadero inv : this.invernaderos) {
                if (inv != null) {
                    naturaMadura += inv.getMature();
                }
            }
            break;
        case 2:
            for (Arboleda arb : this.arboledas) {
                if (arb != null) {
```

```
                    naturaMadura += arb.getMature();
                }
            }
            break;
        default:
            break;
    }
    return naturaMadura;
}
```

Método que obtiene el número de plantas/árboles que sólo son maduros de los invernaderos/arboledas.

3.1.4.22.13. Método privado addNatura

```
public void addNatura() {
    int select;
    int contador;
    int planta;
    select = this.selectInvArb();

    switch (select) {
        case 1:
            select = this.selectInv();
            if (select != 0) {
                contador = 4;
                Random rand = new Random();
                final Invernadero inv = this.invernaderos[select - 1];
                while (contador >= 1) {
                    planta = rand.nextInt(0, (this.contadorPlantas() +
1));
                    if ((inv.getTiles() - inv.getNum()) > 0) {
                        this.invernaderos[select -
1].addPlanta(this.plantas[planta]);
                    }
                    contador--;
                }
                this.invernaderos[select - 1].showCapacity();
            }
            break;
        case 2:
            select = this.selectArb();
            if (select != 0) {
                contador = 5;
                Random rand = new Random();
                final Arboleda arb = this.arboledas[select - 1];
                while (contador >= 1) {
                    planta = rand.nextInt((this.contadorPlantas()),
(this.plantas.length) + 1);
                    if ((arb.getTiles() - arb.getNum()) > 0) {
```

```
                        this.arboledas[select -
1].addPlanta(this.plantas[planta]);
                    }
                    contador--;
                }
                this.arboledas[select - 1].showCapacity();
            }
            break;
        default:
            break;
    }
}
```

Método oculto que añade 4 plantas o 5 árboles aleatorios; según se escoja la opción. Este método no resta monedas por la natura nueva.

3.1.4.23. Método main

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int select = -1;
    Simulador simulador = new Simulador();

    simulador.init();
    simulador.showGeneralStatus();

    do {
        System.out.println();
        simulador.menu();
        try {
            select = sc.nextInt();
            switch (select) {
                case 1:
                    simulador.showGeneralStatus();
                    break;
                case 2:
                    simulador.showSpecificStatus();
                    break;
                case 3:
                    simulador.showStats();
                    break;
                case 4:
                    simulador.showNatura();
                    break;
                case 5:
                    simulador.nextDay();
                    simulador.showGeneralStatus();
                    break;
                case 6:
                    simulador.addWater();
                    break;
```

```
                case 7:
                    simulador.waterCrops();
                    break;
                case 8:
                    simulador.plant();
                    break;
                case 9:
                    simulador.harvest();
                    break;
                case 10:
                    simulador.plow();
                    break;
                case 11:
                    simulador.unroot();
                    break;
                case 12:
                    simulador.upgrade();
                    break;
                case 13:
                    simulador.forwardDays();
                    break;
                case 14:
                    System.out.println("¡Hasta luego!");
                    sc.close();
                    break;
                case 98:
                    System.out.println("\nOpción oculta: Añadir natura
aleatoria.");
                    simulador.addNatura();
                    break;
                case 99:
                    System.out.println("\nOpción oculta: Se han agregado
1000 monedas al Simulador.");
                    Simulador.monedas.sumarMonedas(1000);
                    break;
                default:
                    System.out.println("Debe introducir un número en el
rango de 1 a 14");
                    break;
            }
        } catch (InputMismatchException e) {
            System.err.println("Debe introducir un número en el rango de
1 a 14");
            sc.next();
        }
    } while (select != 14);
}
```

Este método simula un menú interactivo de 14 opciones disponibles (2 más ocultas) con gestión de errores en caso de valores erróneos introducidos por teclado.

Mientras no se seleccione la opción indicada para finalizar, el programa seguirá en funcionamiento.

Parámetros:

`args` – Argumentos.

3.1.5. Tanque de Agua

Clase que representa a un tanque de agua, con una capacidad máxima de 20 unidades, con la posibilidad de ser aumentado.

3.1.5.1. Atributos

```
private int aguaDisponible;
private int maxCapacidad;
```

`aguaDisponible` – Representa la cantidad de agua que hay en el tanque.
`maxCapacidad` – Es el número máximo de agua que se puede almacenar. (Se puede ampliar).

3.1.5.2. Detalles del Constructor

```
public TanqueAgua() {
    this.aguaDisponible = 10;
    this.maxCapacidad = 20;
}
```

Constructor sin parámetros. Inicializa el agua con 10 unidades y la capacidad máxima a 20 unidades.

3.1.5.3. Método getWater

```
public int getWater() {
    return this.aguaDisponible;
}
```

Método que obtiene el agua disponible en el tanque.

3.1.5.4. Método getMax

```
public int getMax() {
    return this.maxCapacidad;
}
```

Método que obtiene la capacidad máxima de almacenaje de agua.

3.1.5.5. Método showStatus

```
public void showStatus() {
    System.out.println(this.toString());
}
```

Método que muestra la capacidad del tanque. El porcentaje es calculado multiplicando el agua disponible por 100 y luego es dividido entre la capacidad máxima.

3.1.5.6. Método toString

```
public String toString() {
    return "Tanque de agua al " + ((aguaDisponible * 100) /
maxCapacidad) + "% de su capacidad. [" + aguaDisponible + "/" +
maxCapacidad +"]";
}
```

3.1.5.7. Método addWater

```
public boolean addWater(int cantidadAgua) {
    if ((cantidadAgua + aguaDisponible) > maxCapacidad) {
        System.out.println( ... );
        return false;
    } else {
        aguaDisponible += cantidadAgua;
        return true;
    }
}
```

Método que añade agua al tanque. No permitirá aumentar si la cantidad de agua introducida con la suma del agua disponible, es superior a la máxima capacidad.

Parámetros:

`cantidadAgua` – Cantidad de agua que se quiere introducir al tanque.

3.1.5.8. Método upgrade

```
public void upgrade() {
    maxCapacidad += 5;
}
```

Método que aumenta en 5 unidades la capacidad máxima del tanque

3.1.5.9. Método restWater

```
public void restWater(int restar) {
    this.aguaDisponible = this.aguaDisponible - restar;
}
```

Método que resta ciertas unidades de agua del tanque.

Parámetros:

`restar` – Cantidad de agua a restar del tanque.

3.2. Paquete Natura

Conjunto de plantas y árboles.

3.2.1. Plantae

Clase abstracta que representa la base de una planta.

3.2.1.1. Atributos

```
protected int cicloDias;
protected int numDias;
protected boolean estaRegada;
protected boolean estaViva;
protected boolean estaMadura;
protected final CultivoDatos data;
```

`cicloDias` – Los días restantes para que vuelva a estar madura, en su caso.
`numDias` – Número de días de la planta. (Representa su edad).
`estaRegada` – Si está regada o no.
`estaViva` – Si está viva o no.
`estaMadura` – Si está madura o no.
`estaRecolectada` – Si está recolectada.
`data` – Los datos generales de la planta.

3.2.1.2. Detalles del Constructor

```
protected Plantae(CultivoDatos data) {
    this.data = data;
    this.numDias = 0;
    this.estaRegada = false;
    this.estaViva = true;
    this.estaMadura = false;
    this.cicloDias = this.data.getMadura() + this.data.getCiclo();
}
```

Constructor exclusivo parametrizado. La edad de la planta inicializa en 0 días, no está regada ni madura.

Parámetros:

`data` – Los datos de la planta.

3.2.1.3. Método getName

```
public String getName() {
    return this.data.getNombre();
}
```

Obtiene el nombre de la planta.

3.2.1.4. Método getScientificName

```
public String getScientificName() {
    return this.data.getCientifico();
}
```

Obtiene el nombre científico de la planta.

3.2.1.5. Método isMature

```
public boolean isMature() {
    return this.estaMadura;
}
```

Obtiene true o false si está madura.

3.2.1.6. Método isDead

```
public boolean isDead() {
    return !this.estaViva;
}
```

Este método indica si la planta está muerta.

3.2.1.7. Método isWatered

```
public boolean isWatered() {
    return this.estaRegada;
}
```

Obtiene si la planta está regada o no.

3.2.1.8. Método getPrice

```
public int getPrice() {
    return this.data.getCoste();
}
```

Método que obtiene el precio de la planta.

3.2.1.9. Método getGain

```
public int getGain() {
    return this.data.getMonedas();
}
```

Método que obtiene la cantidad de monedas que se generan por producto cosechado.

3.2.1.10. Método getProduct

```
protected String getProduct() {
    return this.data.getProducto();
}
```

Método que devuelve el nombre del producto de la planta.

3.2.1.11. Método showStatus

```
public void showStatus() {
    System.out.println( ... );
}
```

Método que muestra por pantalla el estado de la planta.

3.2.1.12. Método water

```
public boolean water() {
    if (this.estaViva) {
        return this.estaRegada = true;
    } else {
        return false;
    }
}
```

Método que cambia el valor de `estaRegada` a `true`; si `estaViva` es `true`.

3.2.1.13. Método grow

```
public void grow() {
    if (this.estaViva) {
        if (this.estaRegada) {
            this.numDias++;
            if (this.numDias == this.data.getMadura()) {
                this.estaMadura = true;
            }
            if (this.numDias >= this.data.getMadura()) {
                if (this.data.getCiclo() != -1) {
                    if (this.numDias == this.cicloDias) {
                        this.estaMadura = true;
                        this.cicloDias += this.data.getCiclo();
                    }
                }
                if (this.numDias == this.data.getMuerte()) {
                    this.estaViva = false;
                    this.estaMadura = false;
                }
            }
            this.estaRegada = false;
        } else {
            Random rand = new Random();
            this.estaViva = rand.nextBoolean();
        }
    }
}
```

Aumenta de día a la planta/árbol. Estará madura cuando llegue su edad de maduración o llegue al día del ciclo de maduración. Tras pasar de día, vuelve a no estar regada.

3.2.1.14. Método harvest

```
public int harvest() {
    if (this.estaMadura) {
        Random rand = new Random();
        if (this.data.getCiclo() == -1) {
            this.estaViva = false;
        }
```

```
            this.estaMadura = false;
            return rand.nextInt(this.data.getProdMin(),
this.data.getProdMax() + 1);
        } else {
            return -1;
        }
}
```

Método que devuelve el fruto de la planta en un valor aleatorio entre su producción mínima y su máxima; cuando esta está madura (`true`).

Su estado de viva cambia a `false` si no dispone de un ciclo de recolección.

Devuelve los frutos generados, y si no está madura, -1.

3.2.1.15. Método replant

```
protected void replant() {
    this.numDias = 0;
    this.estaRegada = false;
    this.estaViva = true;
    this.estaMadura = false;
    System.out.println("\n" + this.data.getNombre() + " se ha
replantado.");
}
```

Reinicia los atributos de la planta a su estado inicial.

3.2.1.16. Método showInfoNatura

```
public void showInfoNatura() {
    System.out.println( ... );
      ...
}
```

Método que muestra la información de la planta/árbol. Si tiene un ciclo de maduración, se mostrará ello.

3.2.1.17. Método toString

```
public String toString() {
    return this.data.getNombre() + " - Viva: " + ((this.estaViva) ? "Si"
: "No") +
            " | Madura: " + ((this.estaMadura) ? "Si" : "No") +
            " | Regada: " + ((this.estaRegada) ? "Si " : "No");
}
```

3.2.1.18. Método getCiclo

```
public int getCiclo() {
    return this.data.getCiclo();
}
```

El número de días para que esté madura de nuevo, o -1 si muere al recolectarse.

3.2.2. Arbol

Clase que extiende de Plantae, representa a un árbol que puede ser plantado en una Arboleda.

3.2.2.1. Atributos

Todos los de la clase Plantae y además, uno que verifica si ha llegado por primera vez a la madurez.

```
private boolean growed;
```

3.2.2.2. Detalles del Constructor

```
public Arbol(CultivoDatos arbol) {
    super(arbol);
    this.growed = false;
}
```

Constructor básico.

`arbol` – Los datos del árbol.

3.2.2.3. Método getGrowed

```
public boolean getGrowed() {
    return growed;
}
```

Método que devuelve si el árbol ha llegado a la madurez por primera vez.

3.2.2.4. Método getProduct

```
public String getProduct() {
    return super.data.getProducto();
}
```

Devuelve el nombre del producto que genera el árbol.

3.2.2.5. Método grow

```
@Override
public void grow() {
    if (super.estaViva) {
        if (super.estaRegada) {
            super.numDias++;
            if (this.numDias == super.data.getMadura()) {
                this.growed = true;
                super.estaMadura = true;
            }
            if (this.growed) {
                if (super.data.getCiclo() != -1) {
                    if (super.numDias == super.cicloDias) {
                        super.estaMadura = true;
                        super.cicloDias += super.data.getCiclo();
                    }
                }
```

```
            }
            super.estaRegada = false;
        }

        if (super.numDias >= super.data.getMuerte()) {
            Random rand = new Random();
            if (rand.nextInt(100) + 1 <= 5) {
                super.estaViva = false;
                super.estaMadura = false;
            }
        }
    }
}
```

Es sobrescrito sobre el de Plantae para adecuarse a las condiciones de un árbol. Cuando llegue a su edad de muerte, tendrá una probabilidad del 5% de morir cada día. Los árboles no se mueren cuando no se riegan.

3.2.2.6. Método replant

```
@Override
public void replant() {
    super.replant();
    this.growed = false;
}
```

Se reinician los valores de la planta.

3.2.3. Autorregable

Clase que representa a un árbol autorregable. Extiende de la Clase Arbol.

```
protected Autorregable(CultivoDatos data) {
    super(data);
}
```

Constructor básico; es pasado por parámetro los datos del árbol.

3.2.3.1. Método grow

```
@Override
public void grow() {
    if (super.estaViva) {
        super.numDias++;
        if (super.numDias == super.data.getMadura()) {
            super.estaMadura = true;
        }
        if (super.numDias >= super.data.getMadura()) {
            if (super.data.getCiclo() != -1) {
                if (super.numDias == super.cicloDias) {
                    super.estaMadura = true;
                    super.cicloDias += super.data.getCiclo();
                }
```

```
            }
            if (super.numDias == super.data.getMuerte()) {
                super.estaViva = false;
                super.estaMadura = false;
            }
        }
        super.estaRegada = false;
    }
}
```

Método sobrescrito que suprime la condición de que debe estar regada.

3.2.4. Hoja

Clase que representa a una planta de tipo Hoja. Si no es regada, muere.

3.2.4.1. Método grow

```
@Override
public void grow() {
    if (super.estaViva) {
        if (super.estaRegada) {
            super.numDias++;
            if (super.numDias == super.data.getMadura()) {
                super.estaMadura = true;
            }
            if (super.numDias >= super.data.getMadura()) {
                if (super.data.getCiclo() != -1) {
                    if (super.numDias == super.cicloDias) {
                        super.estaMadura = true;
                        super.cicloDias += super.data.getCiclo();
                    }
                }
                if (super.numDias == super.data.getMuerte()) {
                    super.estaViva = false;
                    super.estaMadura = false;
                }
            }
            super.estaRegada = false;
        } else {
            super.estaViva = false;
        }
    }
}
```

Si la planta no es regada y se llama a este método, morirá.

3.2.5. Sedienta

Clase que representa a una planta sedienta. Extiende de Plantae. Consume dos de agua al regarse.

3.2.5.1. Atributos

```
@Override
public void grow() {
    if (super.estaViva) {
        if (super.estaRegada) {
            super.numDias++;
            if (super.numDias == super.data.getMadura()) {
                super.estaMadura = true;
            }
            if (super.numDias >= super.data.getMadura()) {
                if (super.data.getCiclo() != -1) {
                    if (super.numDias == super.cicloDias) {
                        super.estaMadura = true;
                        super.cicloDias += super.data.getCiclo();
                    }
                }
                if (super.numDias == super.data.getMuerte()) {
                    super.estaViva = false;
                    super.estaMadura = false;
                }
            }
            super.estaRegada = false;
        } else {
            /* 75% de probabilidad de morir si no es regada */
            Random rand = new Random();
            if (rand.nextInt(0, 4) > 0) {
                super.estaViva = false;
            }
        }
    }
}
```

Tiene una probabilidad aumentada de morir si no se ha regado.

3.2.6. Tubérculo

Clase que extiende de Plantae. Tiene un 50% de probabilidad de replantarse al ser recolectada.

3.2.6.1. Método harvest

```
@Override
public int harvest() {
    if (this.estaMadura) {
        Random rand = new Random();
        if (this.data.getCiclo() == -1) {
            this.estaViva = false;
        }
        this.estaMadura = false;
        /* 50% de probabilidad de replantarse */
        if (rand.nextBoolean()) {
            super.replant();
```

```
        }
        return rand.nextInt(this.data.getProdMin(),
this.data.getProdMax() + 1);
    } else {
        return -1;
    }
}
```

3.2.7. Subpaquete Arboles

Paquete con los árboles implementados en esta versión del Simulador.

3.2.7.1. Kiwi

Clase que representa a un árbol de tipo frutal.

```
public Kiwi() {
    super(AlmacenPropiedades.KIWI);
}
```

```
==========Kiwi==========
Precio: 400
Producto: Kiwi
Número de productos: 5-20
Monedas por producto: 5/producto
Maduración: 20 días
Tiempo de vida: 150 días
Ciclo: 10 días
```

3.2.7.2. Madroño

Clase que representa a un árbol autorregable de tipo frutal.

```
public Madronho() {
    super(AlmacenPropiedades.MADRONHO);
}
```

```
==========Madroño==========
Precio: 900
Producto: Madroño
Número de productos: 5-20
Monedas por producto: 15/producto
Maduración: 20 días
Tiempo de vida: 80 días
Ciclo: 10 días
```

3.2.7.3. Manzano

Clase que representa a un árbol de tipo frutal.

```
public Manzano() {
    super(AlmacenPropiedades.MANZANO);
}
```

```
==========Manzano==========
Precio: 400
Producto: Manzana
Número de productos: 5-20
Monedas por producto: 8/producto
Maduración: 50 días
Tiempo de vida: 200 días
Ciclo: 10 días
```

3.2.7.4. Melocotonero

Clase que representa a un árbol de tipo frutal.

```
public Melocotonero() {
    super(AlmacenPropiedades.MELOCOTONERO);
}
```

```
==========Melocotonero==========
Precio: 700
Producto: Melocotón
Número de productos: 5-20
Monedas por producto: 8/producto
Maduración: 30 días
Tiempo de vida: 150 días
Ciclo: 10 días
```

3.2.7.5. Vid

Clase que representa a un árbol industrial de tipo frutal.

```
public Vid() {
    super(AlmacenPropiedades.VID);
}
```

```
==========Vid==========
Precio: 900
Producto: Uva
Número de productos: 5-10
Monedas por producto: 5/producto
Maduración: 50 días
Tiempo de vida: 500 días
Ciclo: 10 días
```

3.2.8. Subpaquete Plantas

Paquete de las plantas implementadas en esta versión del Simulador.

3.2.8.1. Avena

Clase que representa a una planta triturable de tipo cereal.

```
public Avena() {
    super(AlmacenPropiedades.AVENA);
}
```

```
==========Avena==========
Precio: 15
Producto: Avena
Número de productos: 0-6
Monedas por producto: 8/producto
Maduración: 8 días
Tiempo de vida: 18 días
```

3.2.8.2. Garbanzos

Clase que representa a una planta secano de tipo legumbre.

```
public Garbanzos() {
    super(AlmacenPropiedades.GARBANZOS);
}
```

```
==========Garbanzos==========
Precio: 20
Producto: Garbanzos
Número de productos: 10-15
Monedas por producto: 3/producto
Maduración: 4 días
Tiempo de vida: 7 días
```

3.2.8.3. Lechuga

Clase que representa a una planta hidropónica de tipo hoja.

```
public Lechuga() {
    super(AlmacenPropiedades.LECHUGA);
}
```

```
==========Lechuga==========
Precio: 30
Producto: Lechuga
Número de productos: 1-1
Monedas por producto: 40/producto
Maduración: 2 días
Tiempo de vida: 4 días
```

3.2.8.4. Pimiento

Clase que representa a una planta sedienta de tipo hortaliza.

```
public Pimiento() {
    super(AlmacenPropiedades.PIMIENTO);
}
```

```
==========Pimiento==========
Precio: 120
Producto: Pimiento
Número de productos: 5-10
Monedas por producto: 4/producto
Maduración: 6 días
Tiempo de vida: 33 días
Ciclo: 5 días
```

3.2.8.5. Remolacha azucarera

Clase que representa a una planta de tipo tubérculo.

```
public RemolachaAz() {
    super(AlmacenPropiedades.REMOLACHA_AZ);
}
```

```
==========Remolacha az.==========
Precio: 30
Producto: Remolacha az.
Número de productos: 1-2
Monedas por producto: 40/producto
Maduración: 8 días
Tiempo de vida: 12 días
```

3.2.8.6. Trigo

Clase que representa a una planta triturable de tipo cereal.

```
public Trigo() {
    super(AlmacenPropiedades.TRIGO);
}
```

```
==========Trigo==========
Precio: 10
Producto: Trigo
Número de productos: 2-3
Monedas por producto: 5/producto
Maduración: 3 días
Tiempo de vida: 5 días
```

3.2.8.7. Secano

Clase que representa a una planta secano. (Podría ser perfectamente una interfaz, la verdad)

```
protected Secano(CultivoDatos data) {
    super(data);
}
```

3.3. Paquete de interfaces

Una serie de interfaces bandera son usadas en diferentes métodos para aplicar una serie de características especiales en algunas plantas y/o árboles.

- **IHidropónica**: Permite plantación hidropónica.
- **IIndustrial**: Tiene usos industriales.
- **ISumergida**: En las plantas, consume 5 de agua al ser regada y un 75% de morir si no se ha regado antes de pasar un día.
- **ITriturable**: Permite ser procesada en un molino, lo que genera harina y ganancias del 20% en el momento de su recolección.

3.4. Paquete Monedas

3.4.1. Sistema de monedas

Es un sistema económico en el Simulador. Esto permite ganar mediante acciones o gastar en otras.

3.4.1.1. Atributos

`private int monedas;` - Monedas del sistema

3.4.1.2. Detalles del Constructor

```
public SistemaMonedas() {
    this.monedas = 100;
}
```

Constructor sin parámetros. El sistema se inicia con 100 monedas.

3.4.1.3. Método getMonedas

```
public int getMonedas() {
    return this.monedas;
}
```

Método que devuelve el número de monedas que se disponen.

3.4.1.4. Método comprobarMonedas

```
public boolean comprobarMonedas(int monedas) {
    if (monedas <= this.monedas) {
        return true;
    } else {
        return false;
    }
}
```

Este método comprueba si se pueden restar las monedas pasadas por parámetro. Es útil para métodos en los que no se restan directamente las monedas, si existen una serie de pasos o comprobaciones adicionales antes.

3.4.1.5. Método restMonedas

```
public void restMonedas(int monedas) {
    System.out.println("\n[-" + monedas + " monedas]");
    this.monedas -= monedas;
}
```

Método que resta las monedas pasadas por parámetro. Lo ideal es que primero se llame al método comprobar monedas para confirmar la ejecución de este método. Se muestra un mensaje intrusivo de las monedas restadas.

3.4.1.6. Método toString

```
@Override
public String toString() {
    return this.monedas + " monedas disponibles";
}
```

3.4.1.7. Método sumarMonedas

```
public void sumarMonedas(int monedas) {
    System.out.println("\n[+" + monedas + " monedas]");
    this.monedas += monedas;
}
```

Agrega monedas al sistema pasadas por parámetro. Además también se muestra un mensaje de la adición de las mismas.

4. Librerías adicionales utilizadas

- Class Scanner. https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html
- Class Arrays. https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html
- Class Random. https://docs.oracle.com/javase/8/docs/api/java/util/Random.html
- Class InputMismatchException. https://docs.oracle.com/javase/7/docs/api/java/util/InputMismatchException.html