Institución Universitaria

Tercer Evento evaluativo 20%

-YOUR-IF-YOU-PLAY-LTDA

PROFESORA: MORELA DEL SOCORRO MONCADA GONZALEZ

Integrantes:

CUARTAS VILLA CARLOS ANDRES

CESPEDES ZULUAGA CARLOS ANDRES

Documento de Explicación del Proyecto "-YOUR-IF-YOU-PLAY-LTDA"

Introducción:

Este documento describe el desarrollo de un sistema de inventario de muñecos implementado en Java. El sistema permite agregar diferentes tipos de muñecos (bebés y de acción), realizar un seguimiento de su cantidad y precio, y utilizar estructuras de datos como Pilas y Colas para gestionar información específica sobre los muñecos.

Clases Principales:

1. **inventario:**
    - Esta es la clase principal que gestiona el inventario de muñecos.
    - Contiene contadores y acumuladores para realizar un seguimiento de la cantidad y el precio total de los muñecos bebé y de acción.
    - Mantiene un registro de la cantidad de muñecos con más de 12 articulaciones y el total de muñecos.
    - Utiliza una Pila (munecosRecienAgregados) para almacenar los muñecos recién agregados, siguiendo el principio LIFO (Last-In, First-Out).
    - Utiliza una Cola (munecosProximosAgotarse) para gestionar un listado de muñecos que se consideran próximos a agotarse, siguiendo el principio FIFO (First-In, First-Out).
    - Proporciona métodos para agregar muñecos de cada tipo, calcular promedios de precios, contar muñecos con muchas articulaciones, calcular el porcentaje de muñecos de acción y mostrar la información de la Pila y la Cola.
    - La interacción con el usuario se realiza a través de ventanas emergentes de JOptionPane para el menú principal y la entrada de datos.

2. **Pila:**
    - Implementa la estructura de datos Pila.
    - Tiene una capacidad máxima (maxsize), un tamaño actual (size) y un puntero al elemento superior (top).
    - Ofrece operaciones básicas como isEmpty(), isFull(), push() (para agregar un elemento), pop() (para eliminar y obtener el elemento superior) y peek() (para ver el elemento superior sin eliminarlo).
3. **Cola:**
    - Implementa la estructura de datos Cola.
    - Tiene una capacidad máxima (maxsize), un tamaño actual (size), y punteros al frente (frente) y al último (ultimo) elemento.
    - Ofrece operaciones básicas como isEmpty(), isFull(), enqueue() (para agregar un elemento al final), dequeue() (para eliminar y obtener el elemento del frente) y peek() (para ver el elemento del frente sin eliminarlo).
4. **Nodo:**
    - Representa un nodo individual en las estructuras de datos enlazadas Pila y Cola.
    - Contiene un dato (que en este caso será un objeto munecosBebe o munecosAccion) y una referencia al siguiente nodo (siguiente).
5. **munecosBebe:**
    - Representa un muñeco bebé.
    - Tiene atributos como id, nombre, nArticulaciones, precioU (precio unitario) y tipoR (tipo de ropa).
    - Incluye un método ingresarDatos(Scanner scanner) para leer los datos del muñeco desde la consola (aunque en la versión final la entrada se realiza con JOptionPane desde la clase inventario).
    - Tiene constructores para crear instancias con y sin datos iniciales, así como métodos getter para acceder a sus atributos.

6. **munecosAccion:**
    - Representa un muñeco de acción.
    - Hereda de la clase munecosBebe, por lo que tiene todos sus atributos y métodos.
    - Añade un atributo específico: enemigoP (enemigo principal).
    - Sobrescribe el método ingresarDatos(Scanner scanner) para incluir la entrada del enemigo principal (nuevamente, la entrada final se maneja con JOptionPane en inventario).
    - Incluye un constructor que llama al constructor de la clase base y establece el enemigo principal, así como métodos getter y setter para su atributo adicional.

Flujo de Ejecución:

1. Al ejecutar la clase inventario, se muestra un menú principal utilizando JOptionPane con opciones para agregar muñecos bebé, agregar muñecos de acción, mostrar estadísticas, mostrar los muñecos recién agregados (desde la Pila), mostrar los muñecos próximos a agotarse (desde la Cola) y salir.
2. Al agregar un muñeco (de cualquier tipo), se solicitan sus datos a través de ventanas de diálogo de JOptionPane desde la clase inventario.
3. La instancia del muñeco creado se agrega a la lista correspondiente en la clase inventario y también se hace un push en la Pila de muñecos recién agregados.
4. Se simula una verificación de agotamiento (basada en el total de muñecos agregados) y algunos muñecos se encolan en la Cola de próximos a agotarse.
5. Las opciones para mostrar la Pila y la Cola utilizan ventanas de JOptionPane para presentar la información de los muñecos almacenados en estas estructuras.
6. Las estadísticas del inventario (promedios de precios, cantidad de muñecos con muchas articulaciones, porcentaje de muñecos de acción) también se muestran en una ventana de JOptionPane.

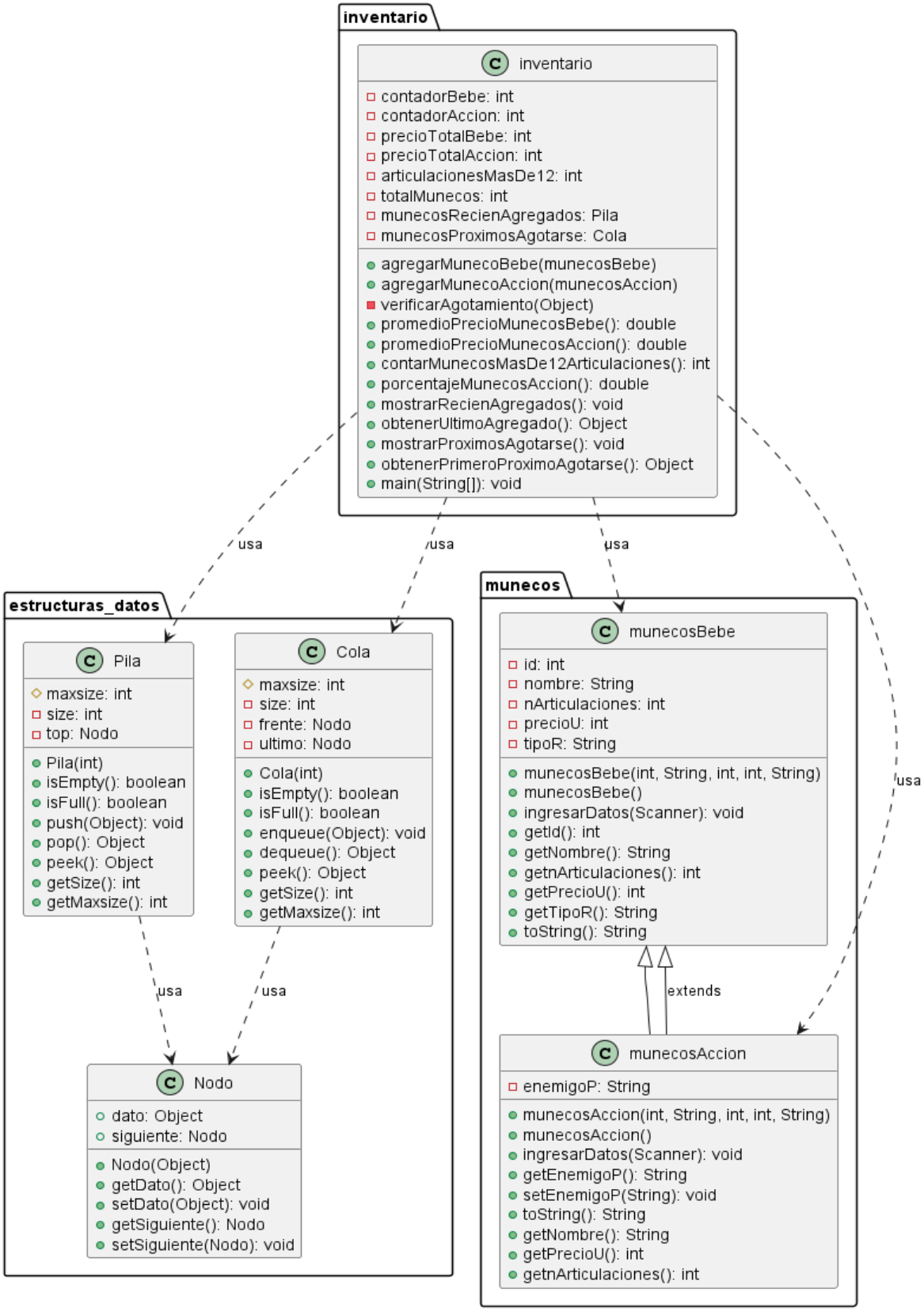
Principios de Programación:

- **Encapsulamiento:** Se aplicó al declarar la mayoría de los atributos como private o protected y proporcionar métodos para acceder y modificar el estado de los objetos.
- **Herencia:** La clase munecosAccion hereda de munecosBebe, lo que permite reutilizar atributos y comportamientos comunes y extender la funcionalidad para los muñecos de acción.
- **Polimorfismo:** Aunque no se explotó extensivamente en este ejemplo sencillo, la capacidad de tratar objetos de diferentes clases de manera uniforme (por ejemplo, al almacenarlos en la Pila y la Cola como Object) es un ejemplo de polimorfismo.
- **Estructuras de Datos:** Se utilizaron las estructuras de datos Pila (LIFO) y Cola (FIFO) para organizar y gestionar los muñecos según diferentes criterios.

Conclusión:

El proyecto " YOUR-IF-YOU-PLAY-LTDA " implementa un sistema básico para gestionar una colección de muñecos, demostrando el uso de clases, herencia, encapsulamiento y estructuras de datos fundamentales como Pilas y Colas. La interfaz de usuario se basa en ventanas emergentes de JOptionPane para facilitar la interacción.

UML


inventario
inventario
contadorBebe: int
contadorAccion: int
precioTotalBebe: int
precioTotalAccion: int
articulacionesMasDe12: int
totalMunecos: int
munecosRecienAgregados: Pila
munecosProximosAgotarse: Cola
agregarMunecoBebe(munecosBebe)
agregarMunecoAccion(munecosAccion)
verificarAgotamiento(Object)
promedioPrecioMunecosBebe(): double
promedioPrecioMunecosAccion(): double
contarMunecosMasDe12Articulaciones(): int
porcentajeMunecosAccion(): double
mostrarRecienAgregados(): void
obtenerUltimoAgregado(): Object
mostrarProximosAgotarse(): void
obtenerPrimeroProximoAgotarse(): Object
main(String[]): void
usa
usa
usa
usa
estructuras_datos
Pila
maxsize: int
size: int
top: Nodo
Pila(int)
isEmpty(): boolean
isFull(): boolean
push(Object): void
pop(): Object
peek(): Object
getSize(): int
getMaxsize(): int
Cola
maxsize: int
size: int
frente: Nodo
ultimo: Nodo
Cola(int)
isEmpty(): boolean
isFull(): boolean
enqueue(Object): void
dequeue(): Object
peek(): Object
getSize(): int
getMaxsize(): int
usa
usa
Nodo
dato: Object
siguiente: Nodo
Nodo(Object)
getDato(): Object
setDato(Object): void
getSiguiente(): Nodo
setSiguiente(Nodo): void
munecos
munecosBebe
id: int
nombre: String
nArticulaciones: int
precioU: int
tipoR: String
munecosBebe(int, String, int, int, String)
munecosBebe()
ingresarDatos(Scanner): void
getId(): int
getNombre(): String
getnArticulaciones(): int
getPrecioU(): int
getTipoR(): String
toString(): String
extends
usa
munecosAccion
enemigoP: String
munecosAccion(int, String, int, int, String)
munecosAccion()
ingresarDatos(Scanner): void
getEnemigoP(): String
setEnemigoP(String): void
toString(): String
getNombre(): String
getPrecioU(): int
getnArticulaciones(): int