

Programación OO Tarea 3

(EN GRUPO DE 2 o 3)

Fecha de entrega en Canvas: 06/06/2025 hasta las 23.59.

Entrega de un enlace de repositorio (nuevo repositorio) en GitHub:

- El readme debe contener los nombres completos de los/las estudiantes
- Diagrama UML (foto o imagen) de la aplicación en el origen del proyecto con clases, métodos, propiedades y relaciones (no es necesario especificar las cardinalidades). Se puede usar software (como <https://online.visual-paradigm.com/diagrams/features/uml-tool/>) o a mano
- Una captura de pantalla de la interfaz en el origen del proyecto (para asegurarme de que se ve igual en mi lado)
- Código del programa y archivos del proyecto
- Tienen que empezar a utilizar los packages de Java (mínimo uno por la parte gráfica y otro por la parte lógica realizado durante la tarea 1)

Esta tarea es un incremento a lo realizado en la tarea 1, que incluyó un Expendedor y un Comprador. El incremento consiste en agregarle la perspectiva gráfica 2D, logrando más parecido con la realidad. La idea es que, usando eventos provocados por el mouse, sea posible mostrar como los objetos de la aplicación se mueven. Para lograr la perspectiva gráfica, se le agregará a nueva clase de objeto con una nueva capacidad que le permitirá auto dibujarse: paintComponent, Para que esta capacidad pueda operar en una ventana gráfica se deben agregar como propiedades un par de enteros para las coordenadas (x, y) de posición del objeto dentro de la ventana, las podrá ser inicializadas en el método constructor y sus parámetros. Debes imaginar que el expendedor tiene un vidrio que permite ver los depósitos en su interior y también las bebidas y monedas que contienen.

Para lograr la tarea 3, usarás como base el proyecto de interfaz GUI visto du en clases y el código de la tarea 2, partiendo de la manera que se te sugiere más adelante.

Para la tarea, el método “main” solo puede tener el código para crear una instancia de Ventana, nada más, todo el resto es creado en una especie de árbol de clases conectadas a partir de Ventana.

Toda la actividad de los objetos será originada por eventos de mouse sobre la clase que extiende JPanel principal. Los eventos serán enviados al comprador o expendedor, que reaccionarán dependiendo de donde sea el click dentro de ellos.

Para lograr que el sistema Comprador-Expendedor sea visible, usaremos una la clase como la siguiente, que define el panel principal al centro de la ventana

```
public class PanelPrincipal extends JPanel { //se ve en el centro de la ventana
    private PanelComprador com;
    private PanelExpendedor exp;
    public PanelPrincipal () {
        exp = new PanelExpendedor (..);
        cf = new PanelComprador(..);
        this.setBackground(Color.white);
    }
    public void paintComponent (Graphics g) { //todo se dibuja a partir de este
método
        super.paintComponent(g); //llama al método paint al que hace override en la
super clase
        //el de la super clase solo pinta el fondo (background)
        com.paintComponent(g); //llama al metodo paintComponent definido en el
PanelComprador
        exp.paintComponent(g); //llama al metodo paintComponent definido en el
PanelExpendedor
    }
}
```

Tanto PanelExpendedor como PanelComprador deben definir sus propios métodos paintComponent

PanelExpendedor y PanelComprador son como vistas/controladores y tienen referencias a clases de la tarea 2 que usarán como modelo.

Los productos y monedas también deben tener su representación gráfica y definir sus métodos paintComponent para dibujarse con un polígono o una imagen, por ejemplo.

Desde el paint del PanelExpendedor se llama a los paintComponent's de los Panel de depósitos (vistas también)

Los paintComponents de los panels de depósitos llaman a los paintComponent's de las vistas de bebidas y monedas;

El enunciado de la tarea 3 se expresa como diferencias con el enunciado de la tarea 2:

- **PanelPrincipal “es un” JPanel:**

- contiene a un PanelExpendedor y sólo un PanelComprador que se reutilizará, pero se modificará para que pueda pasar por una secuencia de estados que se repiten cíclicamente.
- tendrá método paintComponent (extiende JPanel y es el central). Este método ya existe en la super clase por lo tanto le harás override, lo cual implica que **debes llamar al anterior usando super**. Este método paintComponent se ejecuta automáticamente cada vez que ocurre un evento de maximizar la ventana, traer al frente la ventana o si se llama explícitamente al método repaint desde eventos de mouse u otra parte, que lo invoca indirectamente (así ocurre en todas las interfaces gráficas).
- el método constructor de **PanelPrincipal** creará un objeto de clase PanelExpendedor y uno de PanelComprador.
- Las monedas se diferenciarán por color en función de su valor y deben desplegar su número de serie único para cada una, y este número es independiente de su clase. Igual debe suceder con los productos.
- Los click del mouse que se realicen dentro del PanelPrincipal serán enviados al PanelComprador y al PanelExpendedor los que deben poder determinar si el evento ocurrió dentro de ellos y en qué zona, si es que tiene más de una.
- Después de procesados el click de mouse por parte del PanelExpendedor y el PanelComprador, se debe llamar al método repaint para que se ejecute el método paintComponent del PanelPrincipal y actualice todo visualmente.


- **Expendedor:**

- fabricará productos mágicamente en su constructor y las pondrá en depósitos desde el comienzo
- el método comprarProducto tendrá tipo de retorno void, pero en lugar de retornarla la dejará un depósito especial con capacidad de uno solo Producto(sin ArrayList) del cual el comprador debe sacarla de manera similar a getVuelto, en este caso será getProducto. El método getProducto debes imaginarlo como meter la mano al depósito donde cae el producto comprado, para sacarlo.
- las monedas que trae el método comprarProducto (sólo compras exitosas) quedarán almacenadas en un nuevo depósito de monedas sin importar su valor (mezcladas). No se sacarán de ahí.
- Deberá tener también el mismo deposito para monedas de vuelto que en la tarea 2, que puede contener las monedas de diferente valor que se devuelven. **Ahora debería ser posible dar el vuelto en valores distintos de 100.**
- deberá dibujarse usando una vista: incluir un método paintComponent que primero le permite dibujarse como un rectángulo, como mínimo. También debe llamar a los métodos paintComponent de sus componentes vistas de depósitos y estos, de manera análoga, a los productos y monedas que contengan.
- Los panels de Depósitos tendrán posiciones relativas al panel de Expendedor el que les dará sus coordenadas al crearlos desde su constructor.
- Productos y Monedas tendrán también posiciones relativas a los Depósitos y a su posición dentro del ArrayList: 0, 1, n-1, y se deben “ver” los elementos almacenados y sus stocks.
- La vista de los productos dentro de un Depósito puede ser vertical o horizontal, pero debe ser coherente y ordenado.

- El ordenamiento de las monedas puedes definirlo tú, según te funcione mejor.
- Cada vez que se saque o se agregue un producto o moneda desde/a los depósitos, se deberá llamar a reposicionar mediante setXY a todas las vistas de Monedas o Productos almacenadas en ellos.
- Tanto los depósitos como productos deben “verse” en la pantalla dentro del Expendedor y en **posiciones relativas**.
- El depósito de producto único también debe dibujar el producto que contiene, para que se sepa que hay una compra exitosa
- Si se hace click dentro del panel del expendededor, se debe rellenar los depósitos que se encuentre vacío.
- **Comprador:** <este es el que debe diferir más entre las tareas de cada grupo>
 - El comprador es una representación gráfica del usuario de la aplicación, que interactúa con el expendededor haciendo click de mouse en zonas rectangulares que tienen un texto (o button) y un color que las distinga en función de su significado. Un click en una zona puede significar el tipo de producto o de moneda elegidas para la compra. Puedes usar tu creatividad en este aspecto. Las zonas debes diferenciarlas con colores y texto.
 - Se puede dibujarse usando una vista: incluir un método paintComponent que permite se dibuje como un rectángulo, mínimo, y hacer que se dibujen sus componentes internos: depósitos de productos y monedas.
 - Se puede usar JButton, Imagen y otros controles GUI
 - El comprador debe empezar con una suma de dinero suficiente para varias compras y ser capaz de recoger el vuelto a medida que avanza.
 - Hay que tener un área de monedero (infinito eventualmente) para añadir monedas a la interfaz.
- **Productos y Monedas:**
 - Serán similares las mismas de la tarea 2, pero deben tener número de serie y también su método paintComponent en la vista e imprimir su número de serie (gráficamente o en la consola cuando son usadas)
 - Los números de serie se deben inicializar en el constructor
 - Los productos son creados mágicamente en constructor del expendededor
 - Las Monedas pueden ser círculos y los productos pueden ser rectángulos, o imágenes
 - Las monedas deben **ordenarse** cuando se muestran en la interfaz.
 - Las monedas se crean mágicamente al dar el vuelto en el expendededor.
- **Vistas:**
 - Por ahora utilizamos vistas para **separar la lógica propia del código (la tarea 1) de la lógica gráfica**, así que teóricamente podríamos tener varios tipos de vistas (una parte en swing, otra en la web, otra en Android...) utilizando exactamente el mismo modelo lógico.
 - Esto debería permitirle tener que modificar muy poco el código de tarea 2, le recomiendo ponerlo en su propio paquete y usarlo desde la parte gráfica con un import.
 - La parte gráfica (las vistas y otras clases como la JFrame) puede estar en uno o varios paquetes también.
- **Documentación:**
 - Debe documentar las principales clases y métodos de su aplicación usando Javadoc (no es necesario documentar todo en detalle)

Resultados de aprendizaje evaluados:

1. Aplicar el paradigma de la orientación a objetos al desarrollo de aplicaciones software de mediana envergadura, en un entorno de trabajo en equipos.
2. Comparar los beneficios de la orientación a objetos frente a otros paradigmas para el desarrollo de sistemas.
3. Aplicar principios de diseño para el desarrollo de sistemas informáticos de calidad.
4. Construir interfaces gráficas empleando librerías.

Rúbrica de evaluación Tarea 3					 	
Criterios	Calificaciones			Pts		
Ejecución del código KPI: 1.3, 2.1	12 para >8 pts Excelente El código se ejecuta sin problemas y cumple todas las funcionalidades solicitadas en el enunciado.	8 para >4 pts Bien El código se ejecuta sin problema importante pero falta algunas partes menores o hay errores menores.	4 para >0 pts Por mejorar El código tiene problemas importante durante la ejecución o no compila	12 pts		
Calidad de la aplicación KPI: 1.3, 2.1	15 para >10 pts Excelente El código proporcionado utiliza correctamente los conceptos de programación OO y JAVA vistos en clase, tales como: encapsulación, polimorfismo, herencia, abstracción, uso de interfaces, excepciones y enumeraciones. Además, existe una clara separación de responsabilidades y una organización clara.	10 para >5 pts Bien La mayor parte del código hace un buen uso de los conceptos OO y de las capacidades del lenguaje JAVA, pero algunas partes menores del código deben mejorarse en estos aspectos.	5 para >0 pts Por mejorar Gran parte del código proporcionado no utiliza correctamente los conceptos OO, o muestra una falta de dominio de las posibilidades que ofrece el lenguaje JAVA. El código está mal organizado y no respeta la separación de responsabilidades	15 pts		
Documentación KPI: 1.2	6 para >4 pts Excelente Existe documentación en formato javadoc para todas las clases y métodos públicos.	4 para >2 pts Bien Falta informaciones menores en la documentación o problemas menores con el formato.	2 para >0 pts Por mejorar Documentación incompleta o formato incorrecto.	6 pts		
Modelo de clases UML KPI: 1.1	12 para >8 pts Excelente El modelo UML es correcto (contiene todas las clases, métodos, propiedades o relaciones para resolver el problema) y se respeta el formato UML.	8 para >4 pts Bien Falta informaciones menores en el modelo UML y/o hay errores menores.	4 para >0 pts Por mejorar El modelo UML es bastante incompleto o no respeta el estándar UML.	12 pts		
Interfaz gráfica KPI: 1.3, 2.1	15 para >10 pts Excelente La interfaz gráfica respeta el enunciado y es funcional.	10 para >5 pts Bien La interfaz gráfica tiene algunos problemas menores durante la ejecución o en relación con el enunciado.	5 para >0 pts Por mejorar La interfaz gráfica tiene muchos problemas, o problemas significativos o no cumple con el enunciado.	15 pts		
Trabajo en grupo KPI: 5.1	3 para >2 pts Excelente El trabajo está bien distribuido entre los miembros del equipo y esto se refleja tanto en Git.	2 para >1 pts Bien El trabajo está bastante bien distribuido en general pero hay un ligero desequilibrio entre los miembros del grupo.	1 para >0 pts Por mejorar El trabajo está desequilibrado o existe una falta de cooperación por parte de uno o más miembros del grupo. Si no hay una contribución significativa de una o varias personas, se puede aplicar una penalización de puntos en el total de la nota o poner NCR.	3 pts		
Uso de GIT y GitHub KPI: 5.2	6 para >4 pts Excelente GIT se ha utilizado correctamente (los tamaños, las frecuencia, ramas y los mensajes de commits son relevantes).	4 para >2 pts Bien El uso de git es correcto en general, pero algunos mensajes de commits no son claros o las commits deberían ser un poco más frecuentes.	2 para >0 pts Por mejorar e hace un uso moderado de GIT, pero hay problemas recurrentes con los commits en mensajes o frecuencia. Si no se usa git lo suficiente, se puede aplicar una penalización de puntos en el total de la nota o poner NCR (especialmente si se utiliza la interfaz de github en lugar de git directamente)	6 pts		

Puntos totales: 69