Programación II Tarea 3

(EN GRUPO DE 2 O 3 PERSONAS)

Fecha de entrega en Canvas: domingo 13 de noviembre hasta las 23.59.

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

- 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, propriedades 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 NetBeans

Esta tarea es un incremento a lo realizado en la tarea 2, que incluyó un Expendedor de Bebidas 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 cada clase de objeto una nueva capacidad que le permitirá auto dibujarse: paint, 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 2, 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 tiene 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 de Bebidas 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 Comprador com;
    private Expendedor exp;
    public PanelPrincipal () {
        exp = new Expendedor (..);
        cf = new Comprador(..);
        this.setBackground(Color.white);
    }
    public void paint (Graphics g) {  //todo se dibuja a partir de este método
         super.paint(g);  //llama al método pint al que hace override en la super clase
                          //el de la super clase solo pinta el fondo (background)
         com.paint(g);    //llama al metodo paint definido en el comprador
         exp.paint(g);    //llama al metodo paint definido en el expendedor
    }
}
//tanto expendedor como comprador deben definir sus propios métodos paint
//bebidas y monedas tampien deben definir sus métodos paint
//desde el paint del expendedor se llama a los paint's de los depositos
//los paints de los depósitos llaman a los paint's de bebidas y monedas;
//(al agregar o remover algo en un Deposito, se llama setXY de todo lo que contiene)
```

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

- **PanelPrincipal "es un" JPanel**:
  - contiene a un Expendedor y sólo un Comprador que se reutilizará, pero se modificará para que pueda pasar por una secuencia de estados que se repiten cíclicamente.
  - tendrá método paint (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 paint 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 Expendedor y uno de Comprador.
  - 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 las bebidas.
  - Los click del mouse que se realicen dentro del PanelPrincipal serán enviados al Comprador y al Expendedor 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 Expendedor y el Comprador, se debe llamar al método repaint para que se ejecute el método paint del PanelPrincipal y actualice todo visualmente.

- **Expendedor:**
  - fabricará bebidas mágicamente en su constructor y las pondrá en depósitos desde el comienzo
  - el método comprarBebida tendrá tipo de retorno void, pero en lugar de retornarla la dejará un depósito especial con capacidad de una sola Bebida (sin ArrayList) del cual el comprador debe sacarla de manera similar a getVuelto, en este caso será getBebida. El método getBebida debes imaginarlo como meter la mano al depósito donde cae la bebida comprada, para sacarlo.
  - las monedas que trae el método comprarBebida (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.
  - deberá auto dibujarse: incluir un método paint que primero le permite dibujarse como un rectángulo, como mínimo. También debe llamar a los métodos paint de sus componentes depósitos y estos, de manera análoga, a las Bebidas y monedas que contengan.
  - Los Depósitos tendrán posiciones relativas al Expendedor el que les dará sus coordenadas al crearlos desde su constructor.
  - Bebidas 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.
  - El orden de las Bebidas dentro de un Depósito debe ser vertical y ordenadas de abajo a arriba. La más baja es la de índice 0 del ArrayList.
  - El ordenamiento de las monedas puedes definirlo tú, según te funcione mejor.
  - Cada vez que se saque o se agregue una bebida o moneda desde/a los depósitos, se deberá llamar a reposicionar mediante setXY a todas las Moneda o Bebidas almacenadas en ellos.
  - tanto los depósitos como bebidas deben "verse" en la pantalla dentro del Expendedor y en **posiciones relativas**. (El concepto de relativo se ejemplificará en clases)
  - El depósito de bebida única también debe dibujar la bebida que contiene, para que se sepa que hay una compra exitosa
  - Si se hace click dentro del expendedor, se debe rellenar el depósito de bebidas 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 expendedor 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 bebida 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: incluir un método paint que permite se dibuje como un rectángulo, mínimo, y hacer que se dibujen sus componentes internos: depósitos de bebidas y monedas.
  - Se puede usar JButton y otros controles GUI
- **Las Bebidas y Monedas**:
  - Serán similares las mismas de la tarea 2, pero deben tener número de serie y también su método paint, e imprimir su número de serie
  - ambas deben tener un número de serie entero que sea visible para distinguir los movimientos de cada moneda en particular.
  - Los números de serie se deben inicializar en el constructor

- Sobre lugares creación de monedas y bebidas:
  - Las bebidas sólo son creadas mágicamente en constructor del expendedor
  - Las Monedas pueden ser círculos y las bebidas pueden ser rectángulos
  - Las monedas se crean mágicamente al dar el vuelto en el expendedor y en el comprador al comprar.

Rúbrica de evaluación 3

Criterios	Calificaciones				Pts
Adecuación del código al modelo UML	**6 para >4 pts** **Excelente** El código se corresponde perfectamente con el modelo UML o se justifican las diferencias existentes	**4 para >2 pts** **Bien** Una gran parte del código corresponde al modelo UML	**2 para >0 pts** **Por mejorar** Sólo algunos elementos corresponden al modelo UML	**0 pts** **Falta** El código no coincide con el modelo UML	6 pts
Ejecución del código	**6 para >4 pts** **Excelente** El código se ejecuta sin problemas	**4 para >2 pts** **Bien** El código se ejecuta sin problemas pero hay errores menores	**2 para >0 pts** **Por mejorar** El código tiene problemas durante la ejecución	**0 pts** **Falta** El código no se compila o no se ejecuta	6 pts
Completitud y calidad de la aplicación	**6 para >4 pts** **Excelente** La aplicación contiene todo el código necesario y la implementación es correcta	**4 para >2 pts** **Bien** Faltan algunas partes menores del código o hay problemas menores de implementación	**2 para >0 pts** **Por mejorar** Faltan algunas partes importantes del código o hay problemas importantes con la implementación	**0 pts** **Falta** Falta gran parte del código	6 pts
Trabajo en grupo	**6 para >4 pts** **Excelente** El trabajo está bien repartido entre los dos miembros del equipo	**4 para >2 pts** **Bien** El trabajo podría estar mejor distribuido o falta colaboración en las mismas clases	**2 para >0 pts** **Por mejorar** El trabajo está desequilibrado o no hay colaboración	**0 pts** **Falta** El trabajo está totalmente desequilibrado	6 pts
Uso de GIT	**6 para >4 pts** **Excelente** GIT se ha utilizado correctamente (los tamaños, las frecuencias y los mensajes de los commits son lógicos)	**4 para >2 pts** **Bien** GIT fue bien utilizado (tamaño, la frecuencia y los mensajes de los commits podrían mejorarse)	**2 para >0 pts** **Por mejorar** El tamaño, la frecuencia y los mensajes de los commits podrían mejorarse mucho	**0 pts** **Falta** GIT ha sido poco o nada explotado	6 pts
Puntualidad (dependiendo del retraso, la tarea puede no ser evaluada)	**6 para >4 pts** **Excelente** Entrega el trabajo en la fecha establecida.	**4 para >2 pts** **Bien** Entrega el trabajo fuera de plazo, con justificación válida y oportuna.	**2 para >0 pts** **Por mejorar** Entrega el trabajo fuera de plazo, pero con justificación inoportuna.	**0 pts** **Falta** Entrega el trabajo fuera del plazo y sin justificación	6 pts
Modelo UML	**6 para >4 pts** **Excelente** El modelo UML es correcto (contiene todas las clases, métodos, propiedades o relaciones para resolver el problema)	**4 para >2 pts** **Bien** Una gran parte del modelo UML está presente	**2 para >0 pts** **Por mejorar** Sólo algunos elementos son presente en el modelo UML	**0 pts** **Falta** El modelo UML está ausente o muy incompleto	6 pts
Interfaz gráfica	**6 para >4 pts** **Excelente** La interfaz gráfica respeta el enunciado y es funcional	**4 para >2 pts** **Bien** La interfaz gráfica tiene algunos problemas menores durante la ejecución o en relación con el enunciado	**2 para >0 pts** **Por mejorar** La interfaz gráfica tiene muchos problemas, o problemas significativos o no cumple con el enunciado	**0 pts** **Falta** La interfaz gráfica no funciona o es irrelevante	6 pts
					Puntos totales: 48