Evaluación I-2025

Asignatura: Patrones de Software y Programación

Profesor: Daniel San Martín

Nombre Estudiante:

Rut Estudiante:

La evaluación tiene un puntaje total de 90 pts, 60% de exigencia.

Resultados de Aprendizaje

1. Aplicar patrones de diseño para crear un software de acuerdo a estándares de seguridad y desempeño
2. Caracterizar patrones de diseño de acuerdo a su nivel de aplicabilidad a los problemas.

Instrucciones

1. Esta evaluación tiene 3 preguntas. Compruebe que dispone de todas las páginas. Responda a las preguntas en los espacios previstos para ello en las hojas de preguntas. Asegúrese de marcar apropiadamente sus respuestas.
2. Durante la prueba no se puede utilizar: teléfono móvil, calculadora, apuntes u otras fuentes de información.
3. Está prohibido intentar conectarse a internet de cualquier manera. Si es sorprendido obtendrá la calificación mínima. Tampoco puede utilizar dispositivos de almacenamiento externos o cualquier otro dispositivo relojes inteligentes.
4. Lea la prueba completamente DOS veces antes de hacer cualquier pregunta
5. Una prueba respondida correctamente en un 60%, de acuerdo a las ponderaciones asignadas, corresponde a una nota $4,0$.
6. Solamente se pueden realizar preguntas durante los primeros 40 minutos de la prueba. Solo se responderán preguntas respecto a los enunciados a viva voz.
7. La prueba es individual, cualquier sospecha de copia será calificada con la nota mínima y el caso será remitido al comité de ética.
8. En su espacio personal no debe haber nada más que hojas de papel en blanco, lápiz y goma.
9. Los estudiantes quienes se les compruebe falta de honestidad académica o cualquier otro acto contrario a las normas de permanencia universitaria o al espíritu universitario, serán sancionados, según sea la gravedad de la falta, con medidas desde la amonestación verbal hasta la suspensión o pérdida de la condición de estudiante, los estudiantes expulsados no podrán volver a ingresar a ninguna carrera, programa o curso de la institución. El estudiante que incurriere en falta de honestidad, durante la realización de un proceso evaluativo, será calificado con la nota mínima $1,0$.
10. El resto de sus implementos debe guardarlos dentro de su mochila/bolso y ésta debe posicionarse al frente debajo de la pizarra. Si leyó hasta este punto, felicidades, para saber que lo hizo dibuje una estrella al final de esta página.

Enunciado

Usted forma parte del equipo de desarrollo encargado de mantener y evolucionar un sistema de ventas desarrollado con JavaFX. Este sistema permite calcular descuentos a clientes según ciertas condiciones, mostrar productos comprados y registrar información básica del cliente. El sistema funciona correctamente, pero se ha identificado que su diseño presenta problemas de acoplamiento, falta de flexibilidad y uso ineficiente de recursos.

A continuación, se describen tres situaciones específicas que requieren ser abordadas con urgencia para asegurar la escalabilidad del sistema:

1. **Lógica rígida para calcular descuentos** (33%)

   Actualmente, el sistema calcula todos los descuentos directamente dentro de una sola clase (GerenciadorDescuento). Este diseño dificulta modificar las condiciones de descuento, añadir nuevas reglas o realizar pruebas unitarias. Además, todos los descuentos están codificados directamente, y la lógica está altamente acoplada al flujo de cálculo. Su tarea es reorganizar la lógica de cálculo de descuentos de manera que:

   1. Cada tipo de descuento esté separado del resto.

2. Sea posible incorporar nuevos tipos de descuento sin modificar el núcleo del sistema.
3. Se mantenga un bajo nivel de acoplamiento entre el gestor de descuentos y las reglas aplicadas.
4. Estos son los decuentos que actualmente calcula el sistema:
    1. **Descuento por fidelidad**: Cuando el año de registro de un cliente es mayor o igual a 2 años con respecto a la fecha actual.
    2. **Descuento por Cumpleaños**: Cuando la fecha de nacimiento (mes y día) coinciden con la fecha actual.
    3. **Descuento por Categoría**: Cuando por lo menos existe un producto que está siendo comprado en la categoría "casa".
    4. **Descuento por Cantidad de Productos**: Cuando la cantidad de productos comprados es mayor a dos productos.

2. **Acceso no controlado a los datos del cliente** (34%)

   Actualmente, todas las vistas y procesos del sistema acceden directamente al objeto Cliente, obteniendo sus datos desde la base de datos. Sin embargo, hay casos donde el vendedor desea adicionar manualmente la información de clientes por medio de un archivo local de respaldo (clientes.csv) como copia de seguridad. Su tarea es reorganizar el acceso a los datos del cliente de modo que:

   1. El sistema pueda obtener ciertos datos desde archivo si están disponibles y luego desde la base de datos.
   2. Se oculte del resto del sistema cómo se obtienen los datos del cliente.
   3. Se evite el acceso directo e indiscriminado al objeto original Cliente.

3. **Configuración duplicada y difícil de mantener** (33%)

   Actualmente, el sistema utiliza valores fijos distribuidos en distintos lugares del código para datos de configuración como la fecha actual, el formato de fechas, o parámetros que podrían necesitar ser modificados en el futuro (por ejemplo, tipo de moneda, valores por defecto, límites globales, etc.). Esto hace difícil mantener y centralizar estas configuraciones. Su tarea es diseñar una forma centralizada de manejar estas configuraciones generales de la aplicación, de modo que:

   1. Exista una única fuente desde donde se obtengan estos valores.
   2. Pueda ser accedida fácilmente desde cualquier clase sin duplicación de instancias.
   3. Permita modificar un valor de configuración sin tener que recorrer múltiples clases.
   4. Como ejemplo, modifique la línea 71 del main para probar su solución, donde la fecha está hardcodeada.

Evaluación

1. Indique que patrón de diseño se debe implementar para cada caso y así mejorar la mantenibilidad del sistema.
2. Explique brevemente por qué es adecuado la implementación de cada patrón y qué principios de diseño atiende cada uno.
3. Implemente los patrones de cada caso. No es necesario alterar la clase Main para que los cambios se reflejen. Sin embargo, si logra mantener la funcionalidad del sistema (de acuerdo a la interfaz de usuario) tendrá bonificación extra. Además, puede explicar donde se tendría que modificar en el main para que se mantengan las funcionalidades con los cambios realizados.
4. Organize sus implementaciones en paquetes, por ejemplo, *cl.ucn.(nombrePatron)*
5. Para contestar las preguntas, cree un archivo llamado RESPUESTAS.TXT en la raiz del proyecto.

Características del sistema

El sistema incluye cuatro clases modelo: Cliente, Producto, GerenciadorDescuento y Servicio. El proyecto está implementado en JavaFX para IntelliJ, lo que le permite interactuar con una ventana gráfica que contiene controles de interfaz de usuario (UI). Para ejecutar el programa, abra un terminal en su IntelliJ y ejecute el siguiente comando:

```
mvn clean javafx:run
```

Después de algunos segundos aparecerá la pantalla del sistema de cálculo de descuentos. En la parte superior están los datos del cliente. En la parte de al medio, los productos comprados por el cliente y finalmente en la parte inferior, al presionar el botón *calcular descuento* se mostrarán los descuentos aplicados de acuerdo a las reglas presentadas en el enunciado. Para que se muestren los datos del cliente y los productos comprados, debe escoger al cliente

seleccionándolo desde el Combobox, marcado con un rectángulo rojo en la figura (control que se encuentra al lado del label *Cliente:*).

Los datos se encuentran precargados en el sistema en una base de datos SQLite llamada bazar.db, por lo que usted no deberá ingresar nuevos datos. Además, como su tarea es refactorizar el sistema, el comportamiento debe ser el mismo una vez implementado el patrón de diseño.

Aspectos Técnicos

Carga de archivo CSV En el paquete *cl.ucn.utilities* se encuentran dos clases que implementan la lógica de lectura de un archivo csv. Usted solo debe utilizar esas clases para implementar lo solicitado.

Parser de fecha en java Usando la biblioteca joda.time, (la dependencia ya está instalada con maven), a continuación se muestra un ejemplo de obtener la fecha acutal y transformarla en yyyy-mm-dd, como la que utiliza el sistema.

```
import org.joda.time.LocalDate;
import org.joda.time.format.DateTimeFormat;
import org.joda.time.format.DateTimeFormatter;

public class FechaActualJoda {
   public static void main(String[] args) {
      LocalDate fechaActual = LocalDate.now();
      DateTimeFormatter formato = DateTimeFormat.forPattern("yyyy-MM-dd");
      String fechaFormateada = fechaActual.toString(formato);
      System.out.println("Fecha actual: " + fechaFormateada);
   }
}
```

Modificación de interfaz de usuario En la clase Main.java, se encuentra la lógica que implementa la interfaz gráfica. Las líneas de código que son importantes para esta evaluación y que podrían ser afectas a modificación son las siguientes:

1. Desde las línas 89-111, se declaran los controles para etiquetas (labels) y textos de entrada (text inputs) para los descuentos. Si necesitara nuevos controles para mostrar un nuevo descuento por ejemplo, tendría que codificar lo siguiente:

   ```
   Text nuevoDescuentoLbl = new Text("Nuevo Descuento");
   TextField nuevoDescuentoTxt = new TextField();
   nuevoDescuentoTxt.setEditable(false);
   ```

2. Adicionar los controles creados al control *gridPane2* (después de la línea 170 y 177) con la siguiente codificación:

   ```
   gridPane2.add(nuevoDescuentoLbl, 5,0); //adicionar en la linea 171
   gridPane2.add(nuevoDescuentoTxt, 5,1); //adicionar en la linea 178
   ```

Desde las líneas 116-134, se implementa la lógica asociada al botón *Calcular Descuento*. Se describe una breve explicación de cada línea.

1. Creación del botón

   ```
   Button calcularDescuentoBtn = new Button("Calcular Descuento");
   ```

2. Obtención del cliente seleccionado: Se obtiene el cliente desde el ComboBox (desplegable) que ha sido seleccionado.

   ```
   Cliente clienteSeleccionado = comboBox.getValue();
   ```

3. Instanciación de la clase Servicio: Se obtienen los productos asociados a un cliente (rut).

   ```
   Servicio servicio = new Servicio(em);
   List<Producto> productos = servicio.getProductosByRut(clienteSeleccionado.getRut());
   ```

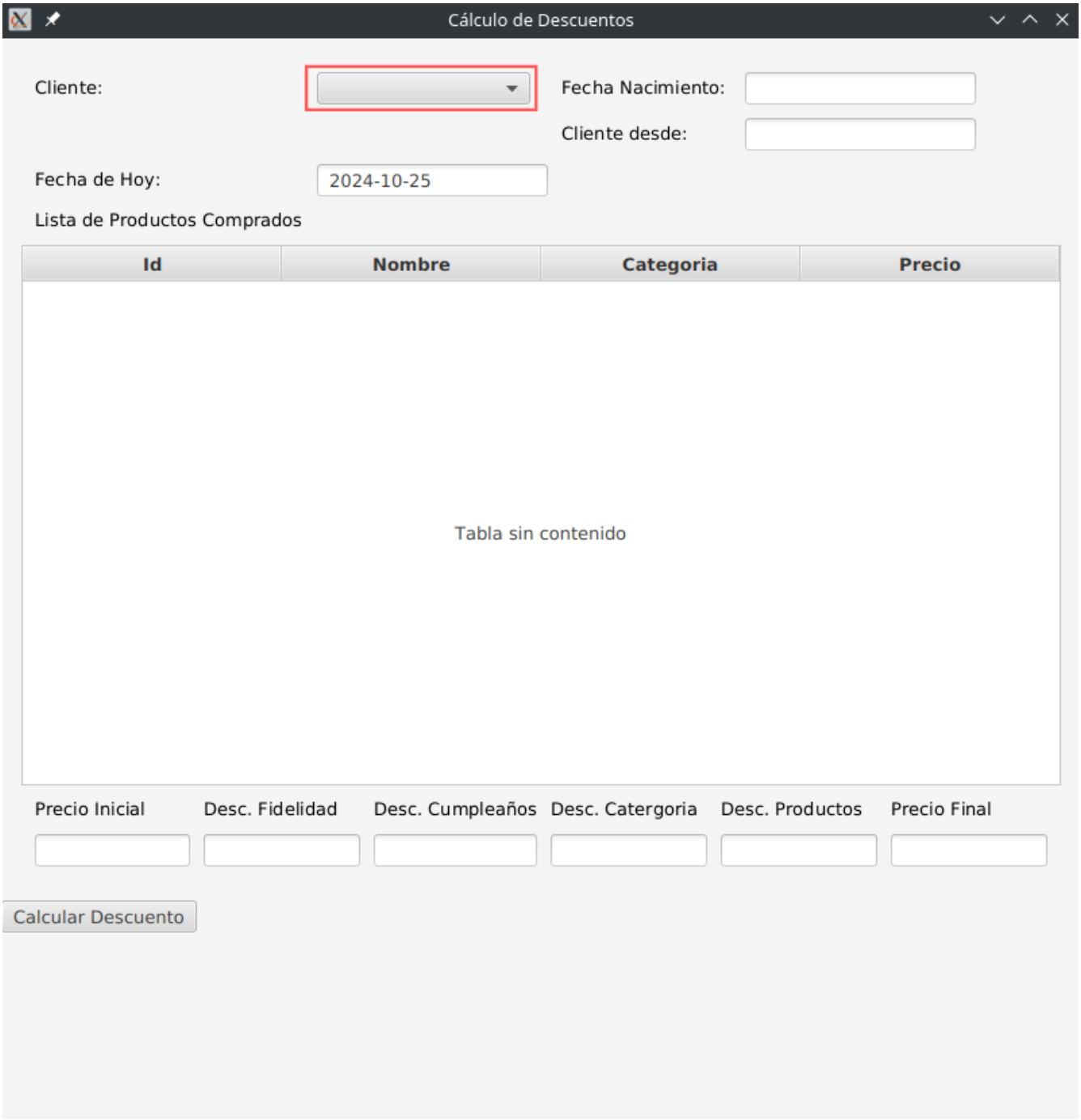

Figure 1: Ventana principal

4. Cálculo de descuentos:

```
GerenciadorDescuento gerenciadorDescuento = new GerenciadorDescuento();
List<Integer> precios = gerenciadorDescuento.calcularPrecioFinal(productos,
clienteSeleccionado.getFechaNacimiento(),
String.valueOf(clienteSeleccionado.getAnhoRegistro()),
fechaActualTxt.getText());
```

5. La instancia *GerenciadorDescuento*, encapsula la lógica para calcular varios descuentos. El método calcularPrecioFinal() recibe la lista de productos y otros parámetros como la fecha de nacimiento del cliente, el año en que se registró y una fecha actual ingresada por el usuario. Este método devuelve una lista de enteros que corresponden a los precios calculados, incluidos los descuentos. A continuación el código que permite la visualización de los descuentos.

```
precioInicialTxt.setText(String.valueOf(precios.get(0)));
descuentoFidelidadTxt.setText(String.valueOf(precios.get(1)));
descuentoCumpleanhosTxt.setText(String.valueOf(precios.get(2)));
descuentoCatergoriaTxt.setText(String.valueOf(precios.get(3)));
descuentoProductosTxt.setText(String.valueOf(precios.get(4)));
precioFinalTxt.setText(String.valueOf(precios.get(5)));
```

Finalmente, se muestran los precios y los diferentes descuentos en varios campos de texto de la interfaz gráfica.

Pauta Evaluación I-2025 - Patrones de Software y Programación

Parte 1: Identificación de patrones (9 pts)

Criterio	Descripción esperada	Puntaje
1.1	Identifica correctamente el patrón Strategy para la lógica de descuentos	3 pts
1.2	Identifica correctamente el patrón Proxy para acceso controlado a datos del cliente	3 pts
1.3	Identifica correctamente el patrón Singleton para configuración centralizada	3 pts

Parte 2: Justificación y principios SOLID (9 pts)

Criterio	Descripción esperada	Puntaje
2.1	Justificación clara de por qué Strategy es adecuado y qué principios SOLID aplica	3 pts
2.2	Justificación clara de por qué Proxy es adecuado y qué principios SOLID aplica	3 pts
2.3	Justificación clara de por qué Singleton es adecuado y qué principios SOLID aplica	3 pts

1. Strategy:
    1. Responsabilidad Única: Separa el algoritmo del contexto.
    2. Abierto/Cerrado: Nuevas estrategias no requieren modificar código existente.
    3. Liskov: Todas las estrategias pueden usarse sin alterar el contexto.
    4. Segregación de Interfaces: Interfaz simple, específica para estrategias.
    5. Inversión de Dependencias: Contexto depende de abstracción, no de implementación.
2. Proxy:
    1. Responsabilidad Única: Separa responsabilidades.
    2. Abierto/Cerrado: Nuevas funcionalidades sin cambiar el objeto real o base.
    3. Liskov: El proxy se puede usar donde se espera el objeto real.
    4. Segregación de Interfaces: Puede presentar una interfaz común para clientes, aunque no siempre se implementa.
    5. Inversión de Dependencias: Cliente depende de la abstracción, no de la implementación concreta.
3. Singleton:
    1. Responsabilidad Única: Mezcla lógica de negocio con control de instancia.
    2. Abierto/Cerrado: Dificil de extender o sustituir ya que es rígido.

3. Liskov: Podría cumplir si usa una interfaz.
4. Segregación de Interfaces: No suele usar interfaces separadas.
5. Inversión de Dependencias: Clases dependientes acceden directamente a una implementación concreta.

Parte 3: Implementación del Patrón Strategy (30 pts)

Criterio	Descripción esperada	Puntaje
3.1	Se define una interfaz `EstrategiaDescuento` con un método común para aplicar descuentos	4 pts
3.2	Se implementan clases concretas para cada tipo de descuento	6 pts
3.3	Se utiliza una lista de estrategias para aplicar descuentos acumulativos	8 pts
3.4	`GerenciadorDescuento` aplica estrategias de forma desacoplada (`List<EstrategiaDescuento>`)	8 pts
3.5	La arquitectura permite añadir nuevas estrategias sin modificar el núcleo del `GerenciadorDescuento`	4 pts

---

Parte 4: Implementación del Patrón Proxy (30 pts)

Criterio	Descripción esperada	Puntaje
4.1	Se define una interfaz común para el acceso a los datos (e.g., `ClienteDataSource`)	4 pts
4.2	Se implementa un proxy (`CargaClienteProxy`) que combina datos desde CSV y base de datos	8 pts
4.3	El proxy crea instancias desde el CSV y las unifica con los datos persistentes	6 pts
4.4	El acceso a los datos del cliente se realiza exclusivamente a través del proxy	8 pts
4.5	El proxy incorpora validaciones o filtros al combinar las fuentes de datos	4 pts

---

Parte 5: Implementación del Patrón Singleton (12 pts)

Criterio	Descripción esperada	Puntaje
5.1	Se implementa correctamente la clase Singleton (`getInstance()`, constructor privado)	4 pts
5.2	Implementación del método que retorna la fecha actual del sistema	2 pts
5.3	Encapsula múltiples valores de configuración	3 pts
5.4	Permite acceso centralizado desde cualquier clase, aunque no se use en `Main.java`	3 pts