Evaluación N°2 — Programación II

Integrantes:

Joaquín Carrasco

Benjamin Cabrera

Leonardo Chávez

Profesor: Guido Mellado

Asignatura: Programación II

Sección: 2

14 de octubre de 2025

Universidad Católica de Temuco
Facultad de Ingeniería

Contenidos

Introducción

Descripción del Sistema

- Sistema de gestión para restaurante implementado en Python
- Manejo de inventario, pedidos y generación de boletas
- Interfaz gráfica intuitiva con CustomTkinter
- Implementación de patrones de diseño para una arquitectura robusta

Arquitectura

Patrones de Diseño - Visión General

- Se implementaron cuatro patrones de diseño principales
- Cada patrón resuelve un problema específico
- Los patrones trabajan en conjunto para crear una arquitectura robusta
- Facilitan el mantenimiento y la extensibilidad del sistema

Patrón Facade

¿Qué es Facade?

- Patrón estructural que proporciona una interfaz simplificada
- Oculta la complejidad del sistema
- Reduce el acoplamiento entre componentes

Implementación en el Proyecto

- Clase `BoletaFacade`:
  - Simplifica la generación de boletas
  - Maneja la creación de PDFs
  - Coordina la visualización de documentos

Patrón Observer

¿Qué es Observer?

- Patrón de comportamiento
- Define una dependencia uno-a-muchos
- Notifica automáticamente cambios a los observadores

Implementación en el Proyecto

- Actualización de la interfaz gráfica:
  - Refleja cambios en tiempo real
  - Mantiene sincronizado el estado
  - Separa la lógica de negocio de la presentación

Patrón Singleton

¿Qué es Singleton?

- Patrón creacional
- Garantiza una única instancia
- Proporciona un punto de acceso global

Implementación en el Proyecto

- Clase `Stock`:
  - Control centralizado del inventario
  - Evita inconsistencias en el estado
  - Gestión thread-safe de recursos

Patrón Factory

¿Qué es Factory?

- Patrón creacional
- Delega la creación de objetos
- Permite extensibilidad y flexibilidad

Implementación en el Proyecto

- Clase `Menu_catalog`:
  - Crea elementos del menú dinámicamente
  - Facilita la adición de nuevos productos
  - Mantiene consistencia en la creación

Interacción entre Patrones

- **Facade + Observer**
  - Notificación de cambios en documentos
  - Actualización automática de vistas
- **Singleton + Factory**
  - Validación centralizada de recursos
  - Creación controlada de productos
- **Observer + Singleton**
  - Monitoreo del estado del inventario
  - Actualización en tiempo real del UI

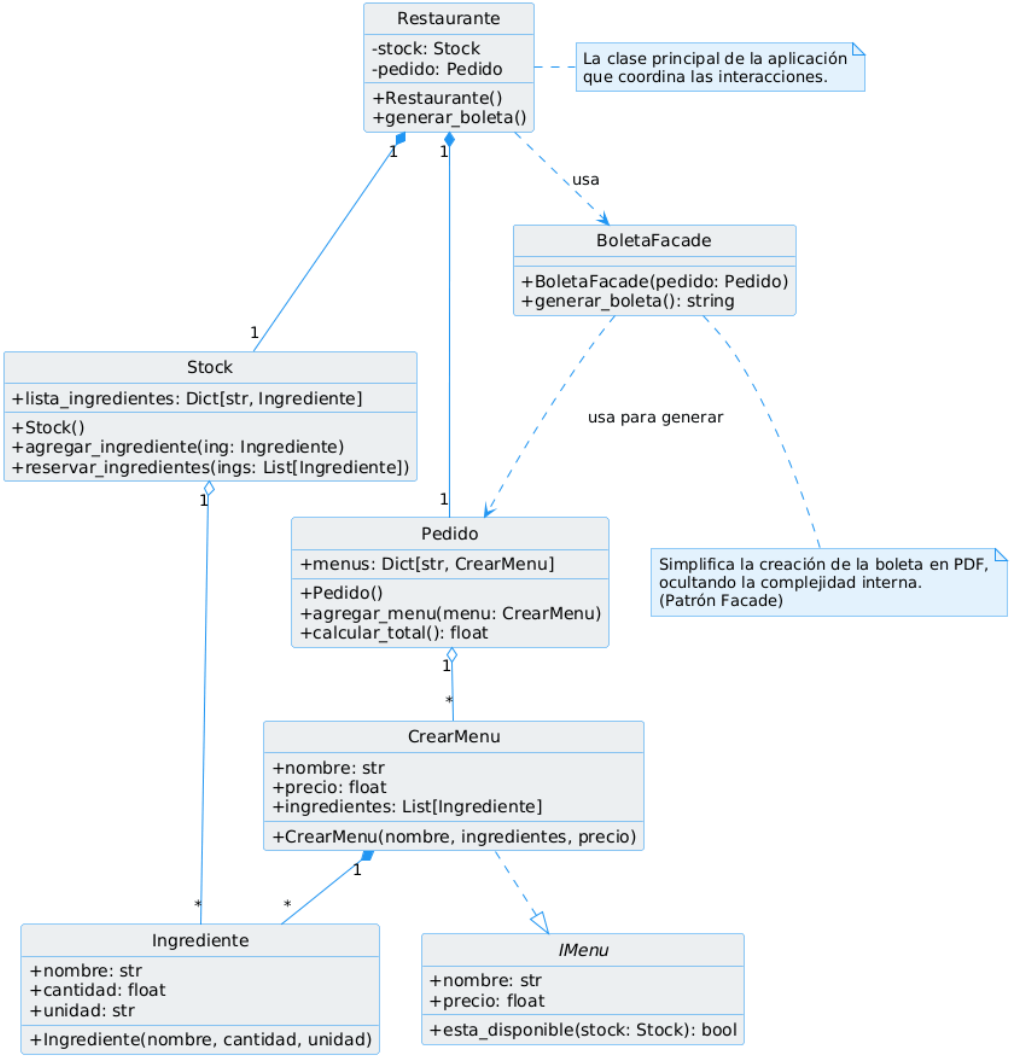
Implementación

Estructura del Proyecto

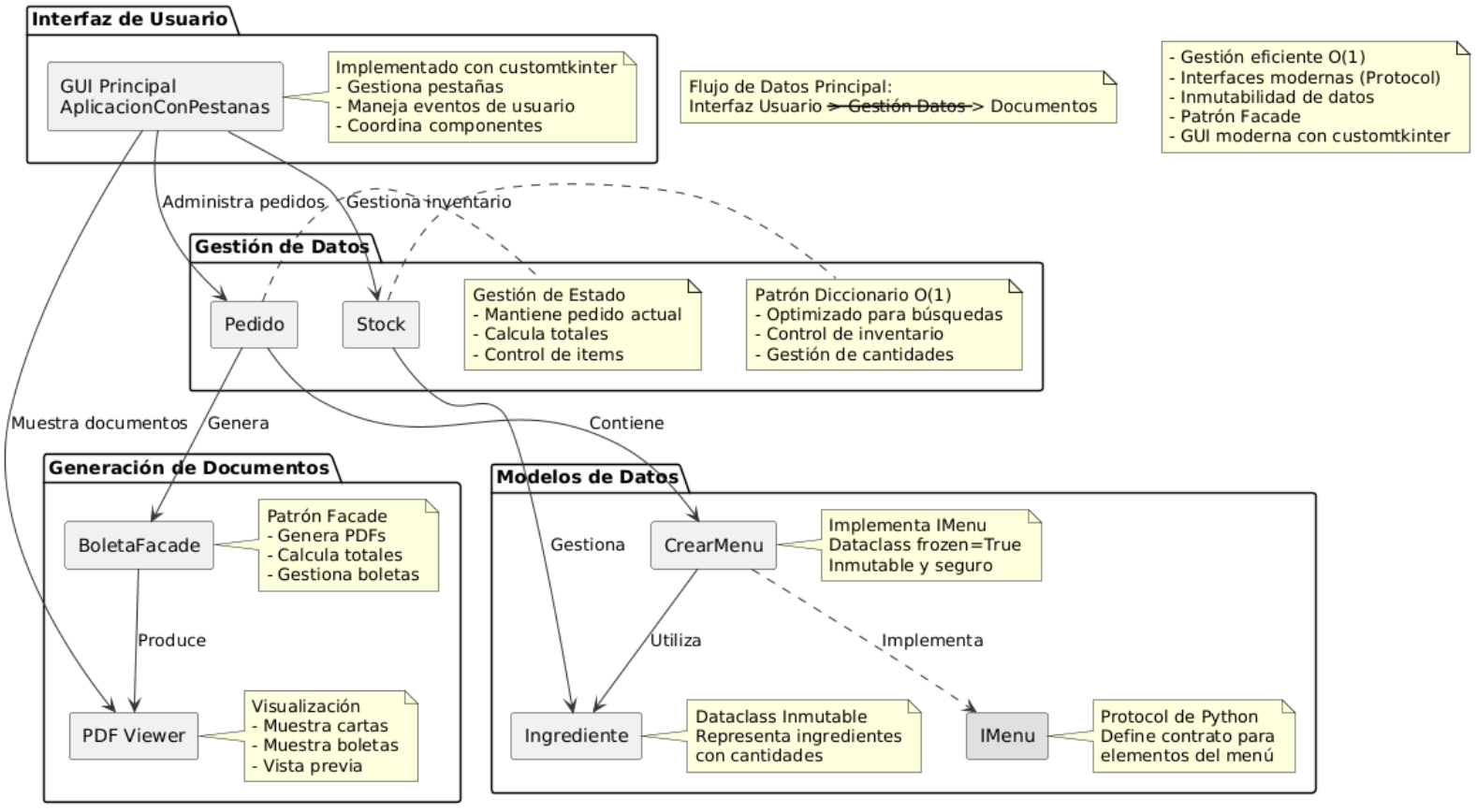
- **Módulos Principales**
  - `Restaurante.py`: Punto de entrada y GUI principal
  - `Stock.py`: Gestión de inventario (Singleton)
  - `Menu_catalog.py`: Catálogo de productos (Factory)
  - `BoletaFacade.py`: Generación de documentos (Facade)
- **Clases de Soporte**
  - `ElementoMenu.py`: Productos del menú
  - `Ingrediente.py`: Componentes base
  - `Pedido.py`: Gestión de órdenes

Funcionalidades

Diagrama de Clases



Explicación del Diagrama de Clases



Implementación

Carga de Ingredientes

- **Botón "Cargar CSV"**:
  - Abre selector de archivo
  - Valida formato CSV
  - Actualiza inventario

Código

```
def cargar_csv(self):
    archivo = filedialog.askopenfilename(
        filetypes=[("CSV files", "*.csv")])
    if archivo:
        self.df_csv = pd.read_csv(archivo)
        self.actualizar_stock()
```

Gestión de Stock

- **Control de Inventario**:
  - Verificación automática
  - Reserva de ingredientes
  - Patrón Singleton

Validación de Stock

```
def verificar_stock(self, nombre, cantidad):
    ingrediente = self.ingredientes.get(nombre)
    return ingrediente and ingrediente.cantidad >= cant
```

Procesamiento de Pedidos

- **Botón "Agregar al Pedido"**:
  - Verifica disponibilidad
  - Reserva ingredientes
  - Actualiza interfaz

Gestión de Pedido

```
def agregar_al_pedido(self, menu_item):
    if self.stock.reservar_ingredientes(menu_item):
        self.pedido.agregar_item(menu_item)
        self.actualizar_vista()
```

Generación de Boleta

- **Botón "Generar Boleta"**:
  - Crea PDF con detalles
  - Usa patrón Facade
  - Muestra vista previa

BoletaFacade

```
def generar_boleta(self):
    facade = BoletaFacade(self.pedido)
    facade.generar_boleta()
    facade.mostrar_boleta()
```

Visualización de PDFs

- **Botón "Mostrar PDF":**
  - Vista previa del documento
  - Navegación entre páginas
  - Zoom y controles interactivos

Visor PDF Personalizado

```
def mostrar_pdf(self, archivo):
    visor = PDFViewer(self.ventana)
    visor.load_pdf(archivo)
    visor.mostrar()
```

Gestión del Menú

- **Botón "Eliminar del Menú"**:
  - Remueve elementos seleccionados
  - Actualiza catálogo en tiempo real
  - Libera ingredientes reservados

Eliminación de Elementos

```
def eliminar_del_menu(self, item_id):
    if item_id in self.pedido.items:
        self.stock.liberar_ingredientes(item_id)
        self.pedido.eliminar_item(item_id)
        self.actualizar_vista()
```

Gestión del Menú - Continuación

- **Botón "Limpiar Pedido"**:
  - Reinicia el pedido actual
  - Libera todos los ingredientes
  - Actualiza la interfaz

Reinicio de Pedido

```
def limpiar_pedido(self):
    for item in self.pedido.items:
        self.stock.liberar_ingredientes(item)
    self.pedido.limpiar()
    self.actualizar_total()
```

Conclusiones

- **Aprendizajes Clave**:
  - Aplicación práctica de patrones de diseño
  - Importancia del desacoplamiento de componentes
  - Manejo efectivo de eventos y estados

- **Desafíos Superados**:
  - Sincronización de GUI con lógica de negocio
  - Manejo consistente del estado global
  - Validación robusta de datos