

# Evaluación N°3 — Programación II

---

**Integrantes:**

Joaquín Carrasco Durán

Benjamín Cabrera

Leonardo Chávez

**Profesor:** Guido Mellado

**Asignatura:** Programación II

**Sección:** 2

Noviembre 2025

# Contenidos

Introducción

Arquitectura

Implementación

Implementación

Mejoras Implementadas

Funcionalidades

Implementación



# 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



# 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

- **Módulos de Mejora NUEVOS**

- error\_handler.py: Manejo centralizado de errores
- cache\_manager.py: Sistema de caché con TTL
- utilities.py: Funciones reutilizables
- statistics\_tab.py: Análisis de datos



## Mejoras Implementadas

---

# Manejo Centralizado de Errores

- `error_handler.py`: Validación y logging centralizado
  - Logging configurado con múltiples niveles (DEBUG, INFO, WARNING, ERROR)
  - Validadores personalizados para entrada del usuario
  - Excepciones específicas del dominio
  - Decorador `@manejo_errores` para captura automática

## Beneficios

- Código más robusto y confiable
- Debugging simplificado con logs detallados
- Validación consistente en todo el sistema



# Patrón Template Method - Validadores

- Implementado en `error_handler.py`: Sistema extensible de validación
  - Clase base abstracta `ValidadorTemplate`
  - Flujo común: Preparar → Validar → Registrar
  - Subclases especializadas: Cantidad, Precio, Nombre, Email
  - Fácil de extender sin duplicar código

## Beneficios del Patrón

- Reutilización de lógica común
- Consistencia garantizada
- Bajo acoplamiento entre validadores
- Fácil de testear



# Sistema de Caché con TTL

- `cache_manager.py`: Optimización de operaciones frecuentes
  - Cache thread-safe con Time To Live (TTL)
  - Decorador `@cache_funciones` para almacenamiento automático
  - Estadísticas de uso (hits, misses, tasa de acierto)
  - Limpieza automática de items expirados

## Impacto de Rendimiento

- Sin caché:  $\approx 500ms$  por query
- Con caché:  $\approx 1ms$  (cache hit)
- Tasa de acierto típica: **85-90 %**



# Módulo de Utilidades

- `utilities.py`: Funciones reutilizables y organizadas
  - `UtilFormatter`: Formateo de precios, cantidades
  - `UtilCalculos`: Totales, descuentos, cálculos monetarios
  - `UtilArchivos`: Operaciones de archivos
  - `UtilValidacion`: Validación de entrada

## Ventajas

- Código DRY (Don't Repeat Yourself)
- Mantenibilidad mejorada
- Reutilización a través del proyecto



# Integración con SQLAlchemy

- ORM moderno para persistencia de datos
  - Modelos tipados con definición clara
  - Operaciones CRUD en módulos separados
  - Relaciones tipadas entre entidades
  - Migraciones de esquema de base de datos

## Módulos de CRUD

- cliente\_crud.py
- ingrediente\_crud.py
- menu\_crud.py
- pedido\_crud.py



# Módulo de Estadísticas

- `statistics_tab.py`: Análisis y reportes
  - Análisis de ventas por período
  - Productos más vendidos
  - Ingresos y tendencias
  - Visualización gráfica de datos

## Información Disponible

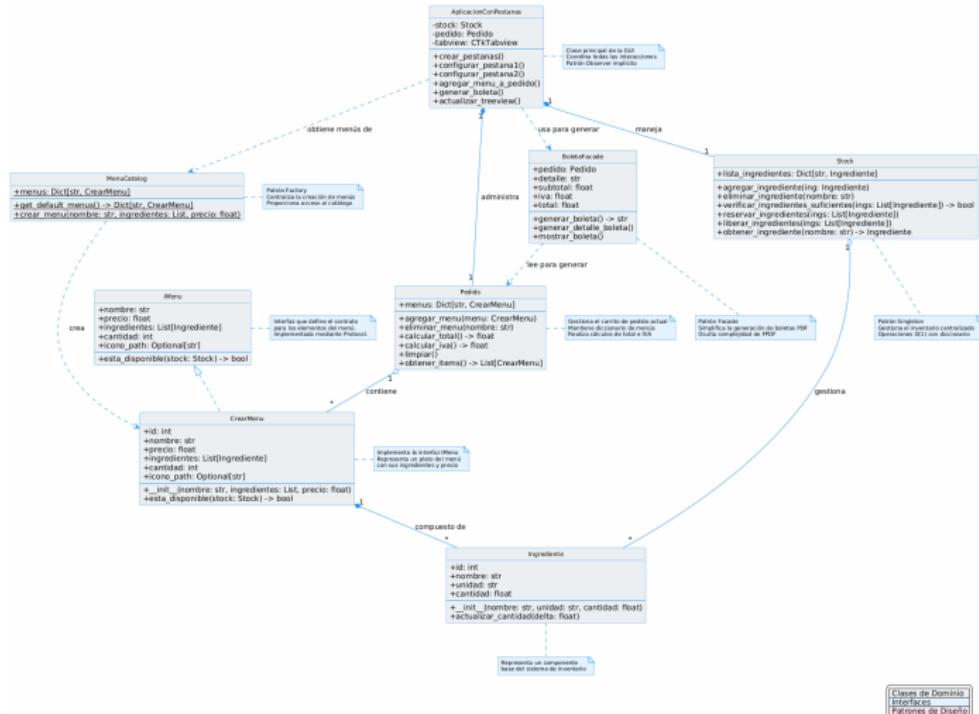
- Estadísticas en tiempo real
- Reportes históricos
- Predicciones de demanda



# Funcionalidades

---

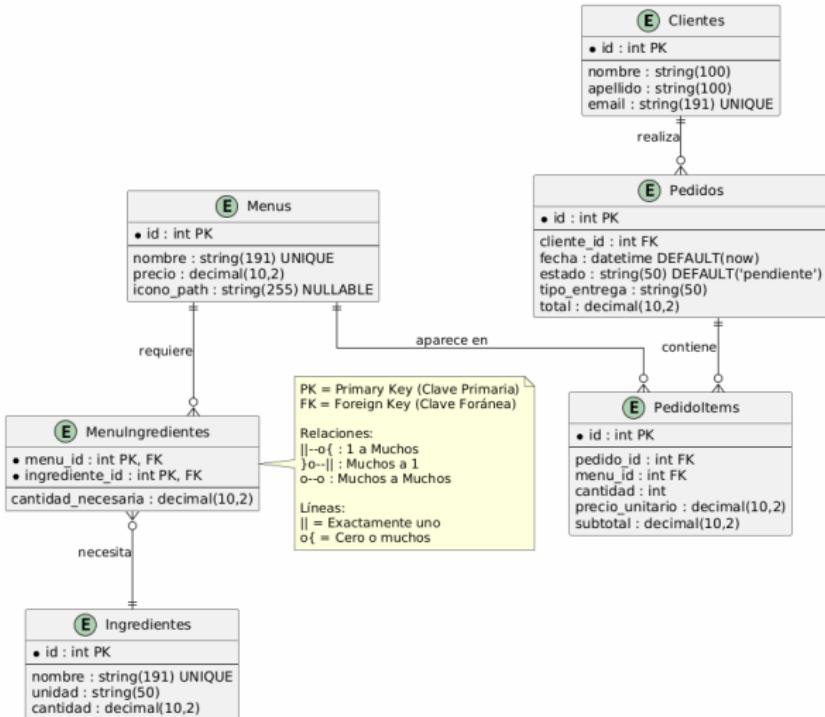
# Diagrama de Clases



Clases de Dominio  
interfaces  
Patrones de Diseño



# Diagrama MER



# 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 >= cantidad
```



# Procesamiento de Pedidos

- Botón “Aregar 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
- Arquitectura escalable y modular

- **Desafíos Superados:**

- Sincronización de GUI con lógica de negocio
- Manejo consistente del estado global
- Validación robusta de datos
- Optimización de performance con caché

- **Mejoras Implementadas:**

- Centralización de errores y logging
- Optimización con sistema de caché TTL
- Integración de ORM (SQLAlchemy)

