Documentación - Proyecto 2

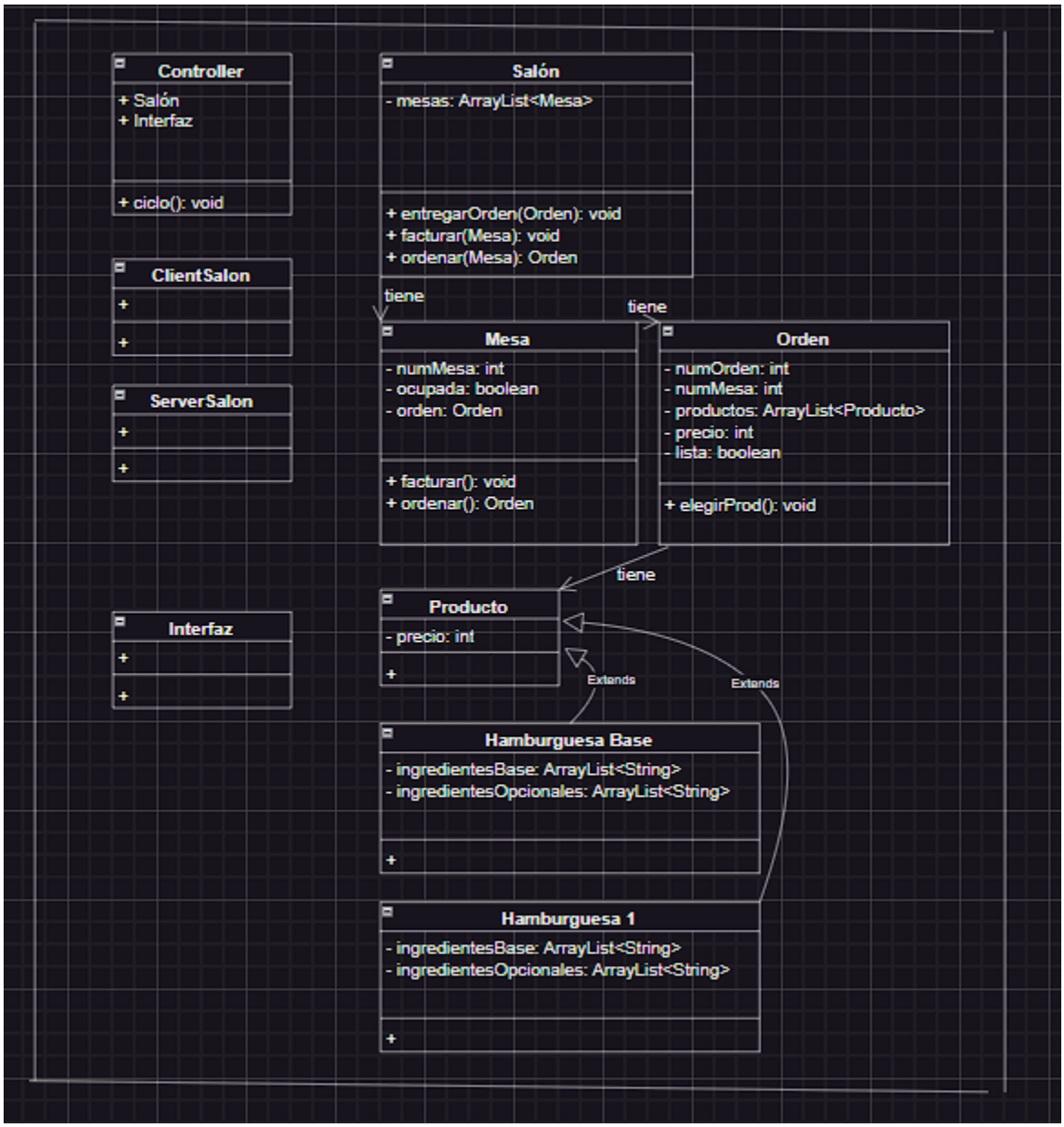
Instituto Tecnológico de Costa Rica - Sede Central Cartago
Programación Orientada a Objetos - Grupo 1
Erick Kauffmann Porcar
Samuel Valverde Arguedas
Prof. Yuen Law Wan
I Semestre 2023

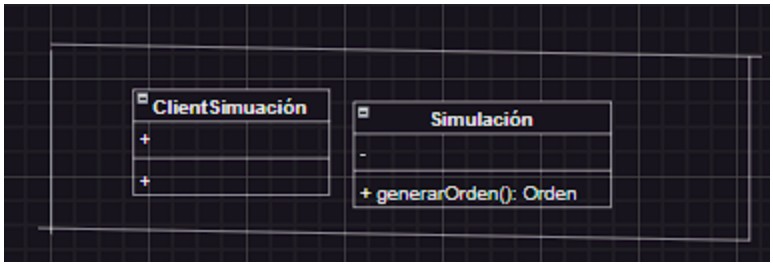
Descripción General

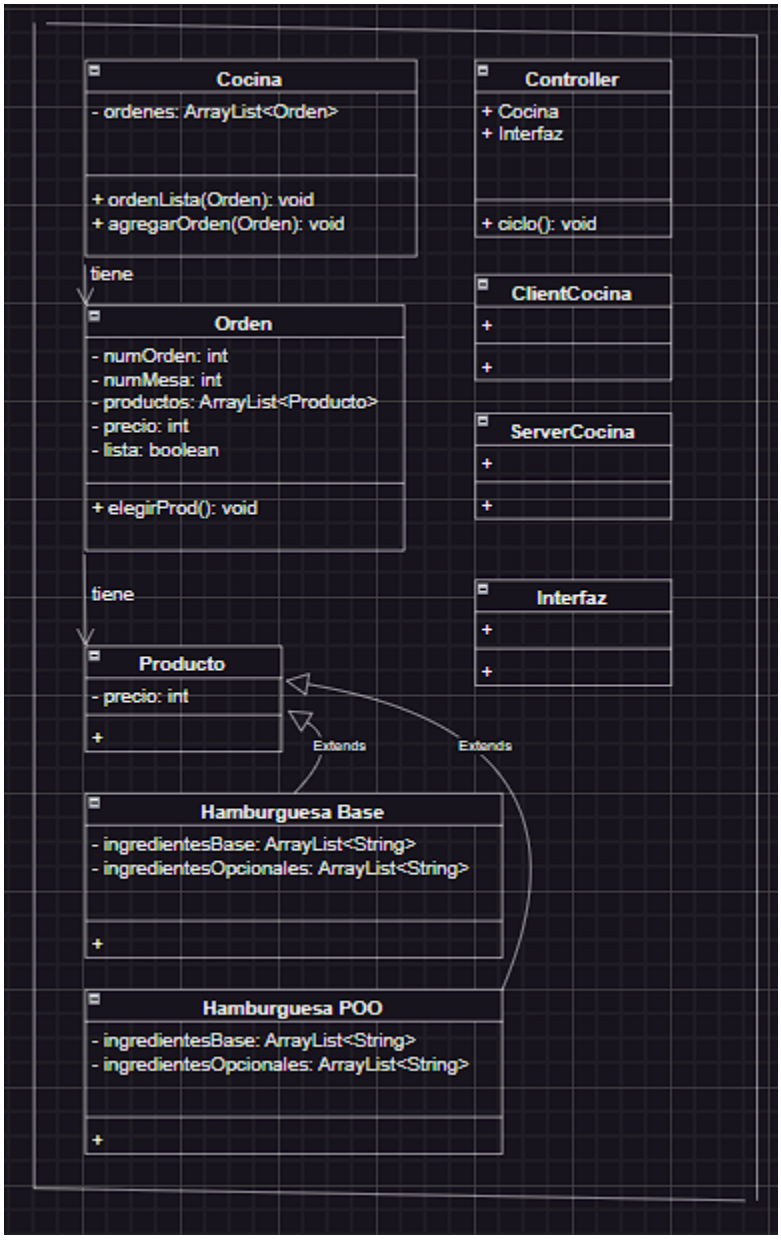
Para la solución de este proyecto se implementaron características base de la programación orientada a objetos: la herencia, el polimorfismo. Además, se implementaron tipos de clases para facilitar y mejorar la calidad del código en general. Con esto se refiere a interfaces y patrones de diseño. Como buena práctica, se programa cuidando el ***clean code***.

Gracias a la solución implementada se pudieron crear 3 módulos que, ejecutados al mismo tiempo, logran simular la cocina de un restaurante, con órdenes y muchas hamburguesas.

Diagrama de Clases




ClientSimuación
+
+
Simulación
-
+ generarOrden(): Orden


Cocina
- ordenes: ArrayList<Orden>
+ ordenLista(Orden): void
+ agregarOrden(Orden): void
Controller
+ Cocina
+ Interfaz
+ ciclo(): void
tiene
Orden
- numOrden: int
- numMesa: int
- productos: ArrayList<Producto>
- precio: int
- lista: boolean
+ elegirProd(): void
ClientCocina
+
+
ServerCocina
+
+
tiene
Interfaz
+
+
Producto
- precio: int
+
Extends
Extends
Hamburguesa Base
- ingredientesBase: ArrayList<String>
- ingredientesOpcionales: ArrayList<String>
+
Hamburguesa POO
- ingredientesBase: ArrayList<String>
- ingredientesOpcionales: ArrayList<String>
+

Explicación de Clases

Módulo Cocina

Controller.java: Se encarga de controlar la cocina y la interfaz.

Interfaz.java: Se encarga de crear la interfaz gráfica de la cocina.

ServerCocina.java, ClientCocina.java: Se encargan de la comunicación con Salón.

Cocina.java: Se encarga de manejar un arraylist de órdenes recibidas del salón y las maneja.

Orden.java: Objeto principal, con hamburguesa, número de mesa y JButton para la interfaz.

Módulo Salón

Controller.java: Se encarga de controlar el salón y la interfaz.

Interfaz.java: Se encarga de crear la interfaz gráfica del salón.

ServerSalon.java, ClientSalon.java: Se encargan de la comunicación con Cocina y Simulacion.

BurgerFactory: Clase principal del patrón Factory. Se encarga de manejar los tipos de hamburguesa y sus constructores.

Burger.java: Clase base de todas las hamburguesas del factory. Asigna los ingredientes extra a cada tipo de hamburguesa de manera aleatoria.

BurgerBasic.java, BurgerTriple.java, BurgerSwiss.java, BurgerPOO.java, BurgerHotChicken.java: Clases de tipo Burger con super() en su constructor más sus especificaciones.

Mesa.java: Objeto con atributo de tipo Orden y número.

Orden.java: Objeto principal, con hamburguesa, número de mesa y más.

Salon.java: Objeto con arraylist de mesas.

Módulo Simulación

Controller.java: Se encarga de controlar la cocina y la interfaz.

Interfaz.java: Se encarga de crear la interfaz gráfica de la cocina.

ServerSimulacion.java, ClientSimulacion.java: Se encargan de la comunicación con Salón.

Simulacion.java: Instancia órdenes aleatorias cada 10 segundos.

Orden.java: Objeto principal de la simulación. Maneja arraylists de tipos de hamburguesa y tipos de ingredientes extra para generar órdenes aleatorias.

Proceso

La implementación de herencia, polimorfismo, patrones de diseño, clean code, hilos, sockets e importaciones entre módulos permitieron que se pudiera desarrollar un proyecto en el que se manejan tres módulos, que trabajando en conjunto, logran simular el programa de un restaurante de hamburguesas. El *Model View Controller*, *Factory Pattern* y *Decorator Pattern* fueron los utilizados para el proyecto.

La implementación de la herencia y el polimorfismo fue clave para la solución de este proyecto siguiendo el modelo planteado, ya que casi todas las clases implementan al menos uno de estos dos. Todas las instancias del juego, los alimentos y los microorganismos, se heredan de clases padre. Además de heredar atributos base, estas clases implementan constructores con diferentes parámetros heredados de sus padres para simplificar el código y evitar la repetición innecesaria entre objetos similares. Esto se puede ver reflejado en el super(specs) que tienen los constructores de las hamburguesas hijas, ya que a todas se les deben asignar sus ingredientes extra y, en vez de asignarse a cada tipo por separado, el constructor de la clase padre puede hacer eso para todas las hamburguesas.

Una desventaja de todo esto es que conlleva un nivel de análisis más elevado y por consiguiente una mayor cantidad de tiempo analizando el problema y las diferentes soluciones así como la mejor forma de programarlo, y una vez completado esto se necesita analizar muy bien cómo se complementan y relacionan los diferentes objetos para poder hacer un uso más eficaz de la herencia y el polimorfismo, así como los patrones de diseño. Sin embargo, la mayor ventaja de estos aspectos es que son muy útiles para evitar la repetición de código innecesario, un estilo de programación sin tanto “if(), else” y una mejor implementación del paradigma de Programación Orientada a Objetos.