# Práctica 0: Repaso de Java e introducci´on a Maven
# Cómputo Concurrente 2024-2

Pérez Romero Natalia Abigail

February 2, 2024

## 1 Diagrama UML

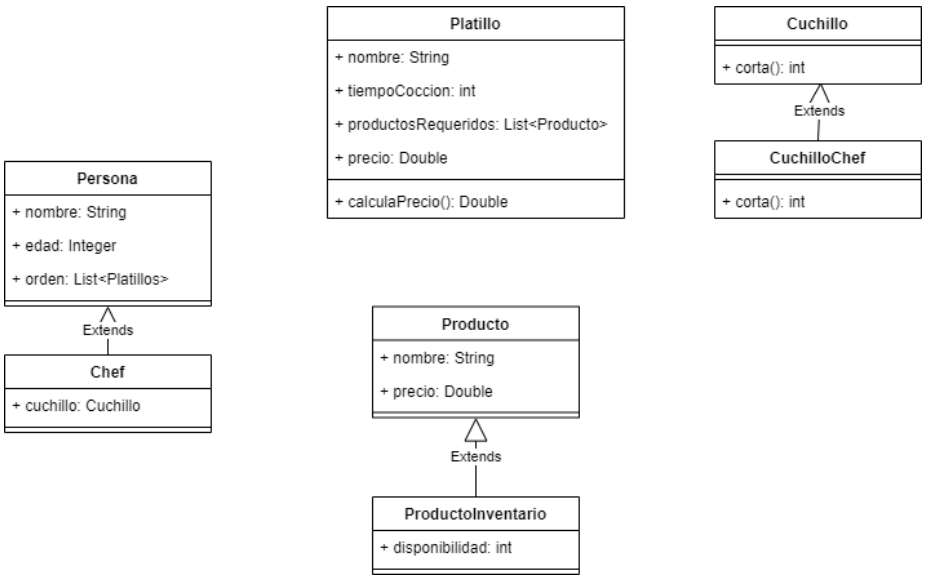


Figure 1: Diagrama UML

## 2 Justificación de Patrones de Diseño

Utilizo el patrón de diseño Strategy, en la clase Persona para permitir almacenar el pedido de una persona, de manera dinámica, es decir, que se pueda cambiar en tiempo de ejecución el algoritmo. Si se ordena un solo platillo, un producto

o varios platillos no seria necesario cambiar la clase, solo se cambia el algoritmo que se va a utilizar.

También en esta clase utilizo el patrón de diseño Adapter, para poder atender el pedido de un solo producto.

```java
/**
 * Ordenar un platillo
 * @param platillo
 */
public void ordenar(Platillo platillo) {
    System.out.println( this.nombre + " ordena " + platillo.getNombre());
    this.orden.add(platillo);
}
/**
 * Ordenar un producto
 * @param producto
 */
public void ordenar(Producto producto) {
    System.out.println(this.nombre + " ordena " + producto.getNombre());
    this.orden.add(new Platillo(List.of(producto), producto.getPrecio(),
    tiempoCoccion:0, producto.getNombre()));
}

/**
 * Ordenar varios platillos
 * @param platillos
 */
public void ordenar(List<Platillo> platillos) {
    System.out.println(this.nombre + " ordena " + platillos);
    for (Platillo platillo : platillos) {
        this.orden.add(platillo);
    }
}
```

Figure 2: Diagrama UML

Luego en las clases Cuchillo-CuchilloChef, Persona-Chef, Producto-ProductoInventario podemos ver el patrón de diseño Decorator, ya que se pueden agregar atributos y métodos a las clases que implentan la interfaz Cuchillo o heredan de la clase Producto, Persona, que les permiten realizar otras operaciones como Chef que es capaz de cortar un producto o Persona que puede hacer un pedido.