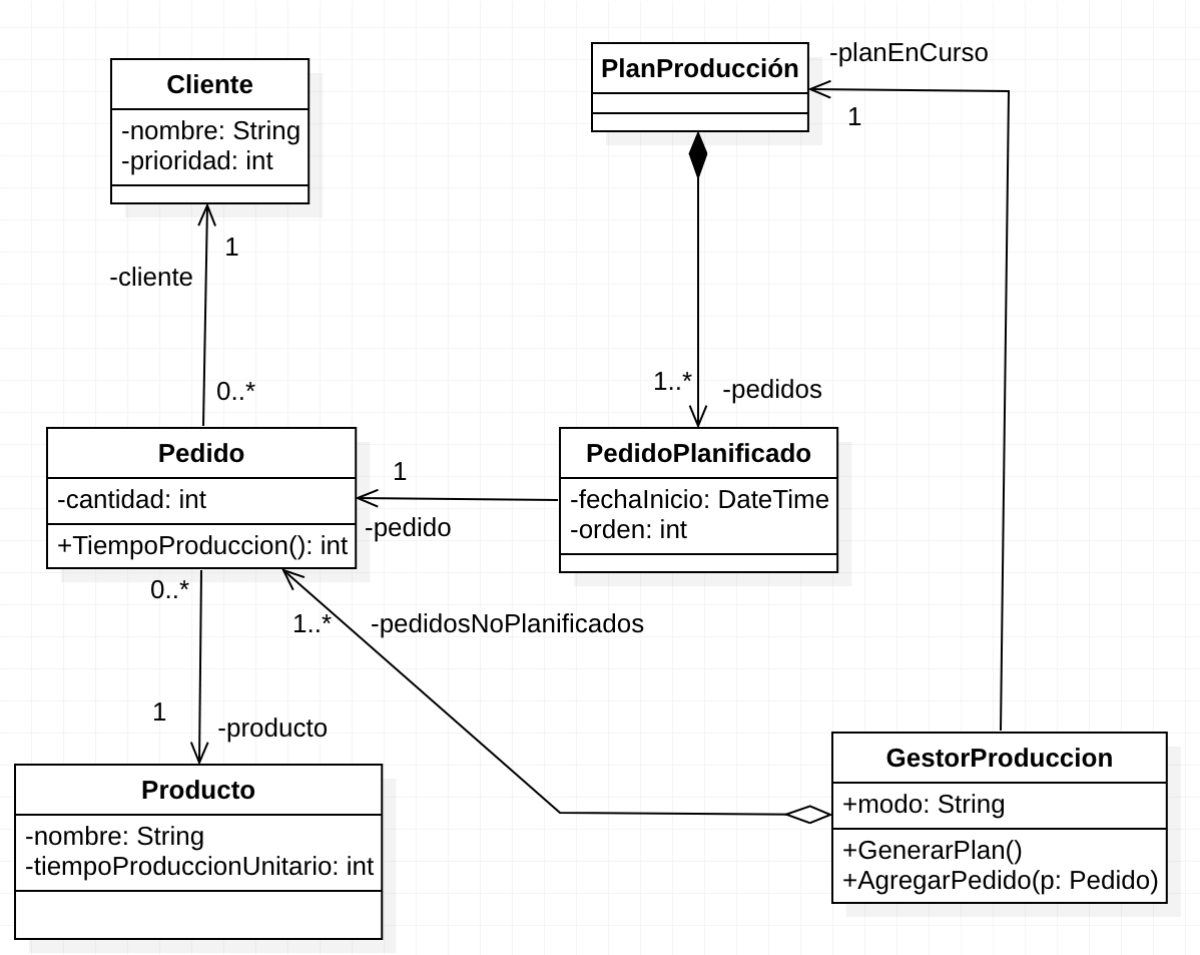
Pregunta 1: Principios, Patrones, GRASP (20 puntos)

El siguiente diagrama presenta el diseño construido para modelar el sistema de gestión de producción de una fábrica. La fábrica produce pedidos para sus clientes. Cada pedido es de un único producto.



La clase GestorProducción almacena los pedidos no planificados, y contiene una referencia al plan de producción en curso. El plan de producción indica que pedidos deben ser producidos y en que orden, permitiendo saber cuando se debe comenzar a producir cada pedido.

a) **(3 puntos)** Se desea que exista una única instancia de la clase GestorProducción. Indique que patrón de diseño utilizaría para resolver el problema, y escriba el código necesario para aplicarlo en este contexto específico.

b) **(5 puntos)** El método **GenerarPlan** en la clase **GestorProducción** toma todos los pedidos sin planificar y genera un nuevo plan de producción. La fábrica tiene dos modos de planificación: **por prioridad**, en la que se producen primero los pedidos de los clientes con mayor prioridad y **por tiempo**, en el que se producen los pedidos que insumen menos tiempo primero. El código actual para generar el plan de producción es el siguiente:

```
public void GenerarPlan() {

    List<Pedido> aPlanificar = new List<Pedido>();
    aPlanificar.AddRange(pedidosNoPlanificados);

    if (modo == "Prioridad") {
        // Ordena los pedidos por prioridad del cliente descendente
        aPlanificar.Sort((pedido1, pedido2) => pedido2.Cliente.Prioridad - pedido1.Cliente.Prioridad);
    } else if (modo == "Tiempo") {
        // Ordena por tiempo de produccion de los pedidos en forma ascendente
        aPlanificar.Sort((pedido1, pedido2) => pedido1.TiempoProduccion() - pedido2.TiempoProduccion());
    }

    PlanProduccion plan = new PlanProduccion();

    int orden = 1;
    foreach (Pedido p in aPlanificar) {
        plan.AgregarPedido(p, orden);
        orden++;
    }

    planEnCurso = plan;
}
```

¿Qué opinión le merece este diseño? Opine en base a los principios de diseño vistos en clase, tomando como premisa que en el futuro pueden aparecer nuevos modos de producción.

Proponga una mejor solución. Construya un diagrama de clases que incluya todos sus cambios. Justifique en función del punto anterior, explicando claramente en que principios o patrones se apoya su solución y que ventajas se obtienen.

c) **(4 puntos)** Realice un diagrama de secuencia para el código original (el que aparece en la letra) del método del punto anterior.

d) **(4 puntos)** En el código de una clase correspondiente a un reporte del plan de producción aparece lo siguiente:

```
public void GenerarReporte(PlanProduccion plan) {

    foreach (PedidoPlanificado p in plan.Pedidos) {
        NuevaLinea();
        GenerarCelda(p.Pedido.Cliente.Nombre);
        GenerarCelda(p.Pedido.Cliente.Prioridad);
        GenerarCelda(p.Pedido.Producto.Nombre);
        GenerarCelda(p.Pedido.Cantidad);
        GenerarCelda(p.Pedido.Cantidad * p.Pedido.Producto.TiempoProduccionUnitario);
    }
}
```

¿Qué puede decir de este código en función de la **Ley de Demeter**?

¿Cómo lo podría mejorar? No es necesario que re-escriba el método completamente, siempre y cuando se entienda la idea de la solución.

e) **(4 puntos)** Siguiendo con los reportes, varios de ellos generarán documentos PDF. Para resolver el problema cuenta con una librearía de terceros que permite generar

documentos PDF, sobre la que no tiene control y que potencialmente necesitará utilizar en varios lugares de su código. ¿Cómo integraría la librería a su solución para minimizar el riesgo de no tener control sobre ella? ¿Hay algún GRASP que lo ayude en esta tarea? Justifique.

Pregunta 2: Clean Code (7 puntos)

a) **(3 puntos)** Compare, las opciones de manejo de excepciones frente al retorno de códigos de error en un método. Indique claramente cual es la opción recomendada y porqué.
b) **(2 puntos)** Según clean code, ¿Cuántos parámetros debería tener un método? ¿Porqué?
c) **(2 punto)** ¿En función de lo anterior, como modificaría el siguiente método para cumplir con lo enunciado?

```
public void Imprimir(string texto, string fuente, int tamano,
    bool italica, bool negrita) {
    //....
}
```

Pregunta 3: TDD (8 puntos)

a) **(4 puntos)** ¿A que refiere el ciclo Red-Green-Refactor en el contexto de TDD?
b) **(4 puntos)** Explique en sus palabras por qué es imprescindible el paso de Refactoring para aplicar TDD correctamente.

Pregunta 4: Práctico (5 puntos)

a) **(2 puntos)** ¿Cuáles son las ventajas de usar Entity Framework como solución de persistencia frente al modelo Conectado y el Desconectado de ADO.NET?
b) **(1 punto)** Explique un motivo por el que usaría Fluent API frente a las anotaciones para configurar el esquema de la base de datos en Entity Framework Code First.
c) **(2 puntos)** ¿Qué características tiene la clase Contexto en Entity Framework y cuáles son sus funciones?