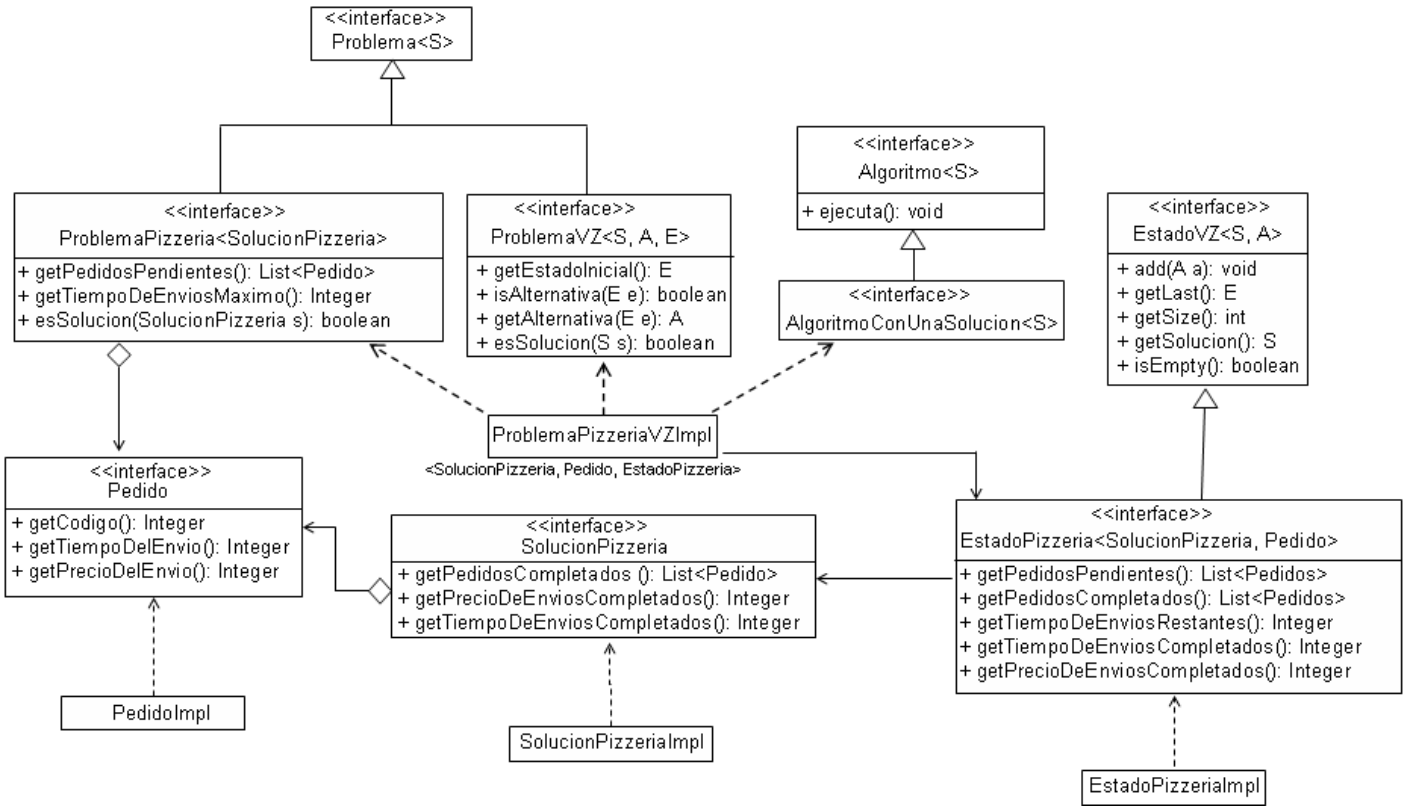
## Ejercicio 3:

Una pizzería tiene que repartir una lista de pedidos por distintas zonas de una ciudad. Cada pedido tiene asociados dos atributos: 1) el precio, que dependerá de los productos incluidos y 2) el tiempo de envío. El tiempo de envío es el tiempo que tarda el repartidor en ir y volver al lugar donde debe dejar el pedido. Nos piden desarrollar un sistema que indique al repartidor la mejor manera de servir los pedidos, para calcular dicha información se ha decidido utilizar la técnica voraz. Los cálculos deberán tener en cuenta que el repartidor dispondrá de un ***tiempo máximo*** para repartir los pedidos, de manera que los pedidos cuyo tiempo de servicio superen el tiempo máximo disponible no serán atendidos. Y considerando que el sistema debe intentar obtener el mayor beneficio posible, se utilizará el ratio ***precio_pedido / tiempo_pedido*** para seleccionar el orden en que se atenderán los pedidos.

Dado el siguiente diagrama de clases:




SE PIDE implementar los métodos TODO de las clases `ProblemaPizzeriaVZImpl` y `EstadoPizzeriaImpl`:

```
public class ProblemaPizzeriaVZImpl implements
    ProblemaPizzeriaVZ, AlgoritmoConUnaSolucion<SolucionPizzeria> {

    List<Pedido> listaDePedidosPendientes;
    Integer tiempoMaximoDeServicio;
    private SolucionPizzeria solucion;

    public void ejecuta() { // TODO }
    public EstadoPizzeria getEstadoInicial() { // TODO }
    public boolean isAlternativa(EstadoPizzeria e) { // TODO }
    public Pedido getAlternativa(EstadoPizzeria e) { // TODO }
}

public class EstadoPizzeriaImpl implements EstadoPizzeria {
   List<Pedido> pedidosCompletados, pedidosPendientes;
   Integer tiempoCompletado, tiempoRestante, precioTotal;

   public EstadoPizzeriaImpl(List<Pedido> lpp, Integer tiempoMax) { // TODO }
   public boolean add(Pedido e) { // TODO  }
   public SolucionPizzeria getSolucion() { // TODO   }
}
```

**Puntuación: 10 puntos** **Tiempo estimado: 50 minutos**

## Solución

```
public class ProblemaPizzeriaVZImpl implements ProblemaPizzeriaVZ,
        AlgoritmoConUnaSolucion<SolucionPizzeria> {
   private List<Pedido> lpp; private Integer tiempoMax;
   private SolucionPizzeria solucion;

    public boolean isAlternativa(EstadoPizzeria e) {
        boolean encontrado = false;
        for(Pedido p: e.getPedidosPedientes()) {
            if (p.getTiempoDelEnvio()+e.getTiempoDeEnviosCompletado() <=
                this.getTiempoDeEnviosMaximo())
                encontrado = true;
            if (encontrado) break;
        } return encontrado;
    }

    public Pedido getAlternativa(EstadoPizzeria e) {
        Pedido a = null;
        for(Pedido p: e.getPedidosPedientes()) {
            if (p.getTiempoDelEnvio()+e.getTiempoDeEnviosCompletado()<=
                this.getTiempoDeEnviosMaximo()) {
                if (a==null) { a= p; }
                else {
                   Double aratio = a.getPrecioDelEnvio().doubleValue()/
                                   a.getTiempoDelEnvio().doubleValue();
                   Double pratio = p.getPrecioDelEnvio().doubleValue()/
                                   p.getTiempoDelEnvio().doubleValue();
                    if (pratio>aratio) a = p;
                }
            }
        } return a;
    }

    public EstadoPizzeria getEstadoInicial() {
        return new EstadoPizzeriaImpl(getPedidosPedientes(), getTiempoDeEnviosMaximo());
    }

    public void ejecuta() {
        Pedido alternativa;
        EstadoPizzeria estado = this.getEstadoInicial();
        while (this.isAlternativa(estado)) {
           alternativa = this.getAlternativa(estado);
           estado.add(alternativa);
        }
        solucion = estado.getSolucion();
    }

}

public class EstadoPizzeriaImpl implements EstadoPizzeria {
    private List<Pedido> lpc,lpp;
    private Integer tiempoCompletado, tiempoRestante, precioTotal;

    public EstadoPizzeriaImpl(List<Pedido> lpp, Integer tiempoMax) {
        this.lpc = new ArrayList<Pedido>();
        this.lpp = lpp; this.tiempoRestante = tiempoMax;
        this.tiempoCompletado = 0; this.precioTotal = 0;
    }

    public boolean add(Pedido e) {
        lpc.add(e); lpp.remove(e);
        tiempoCompletado = tiempoCompletado + e.getTiempoDelEnvio();
        tiempoRestante = tiempoRestante - e.getTiempoDelEnvio();
        precioTotal = precioTotal + e.getPrecioDelEnvio();
        return true;
    }

    public SolucionPizzeria getSolucion() {
        return new SolucionPizzeriaImpl(lpc, tiempoCompletado, precioTotal);
    }
}
```