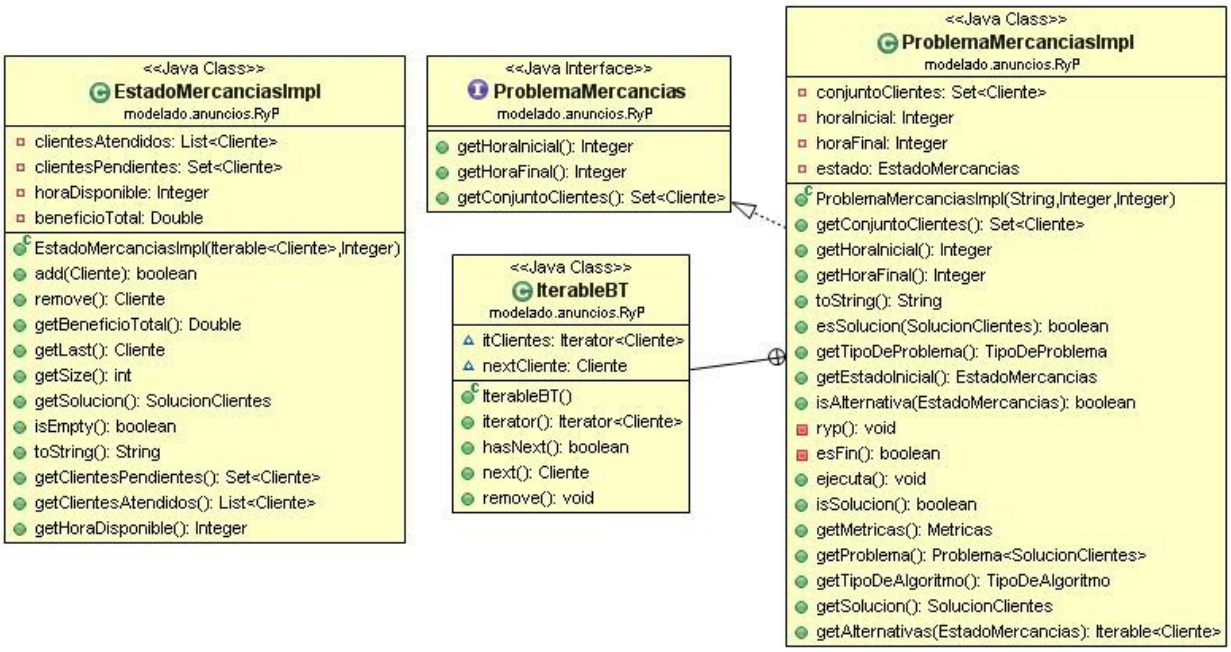
Problema BT: Orden de envío de mercancías

Una empresa de transportes dispone de un único vehículo para el envío de un determinado tipo de mercancía. Diferentes clientes necesitan dicha mercancía para un mismo día. Los envíos se realizan en un orden, y siempre en cada envío el vehículo debe volver a la central. El cliente paga en función de un oferta fija, y la hora del día en que es enviada dicha mercancía. Más concretamente el precio final del pago se calcula como: (oferta * 1000) / horaEnvio + 500.

El objetivo es encontrar, a través de un algoritmo de backtracking, el orden en el que se realizarán las entregas de la mercancía para maximizar los beneficios de la empresa de transportes, aunque para ello se deje a algunos clientes sin suministro. Para modelar el problema, se supondrá que existen *m* clientes (c_1, c_2,..., c_m). El tipo **Cliente** permite almacenar toda la información de un cliente. Los métodos más importantes son:

- Integer getDuracion(): Devuelve el tiempo en horas necesario para realizar la entrega de la mercancía, y retorno a la central después de la entrega.
- Integer getHoraEnvio() : Devuelve la hora de envío prevista para el cliente
- void setHoraEnvío(Integer hora): Establece la hora de envío prevista para el cliente.
- Double getPrecioFinal(): Devuelve el precio final que pagará el cliente por la mercancía (se utilizará la hora de envío almacenada en el cliente para el cálculo).
- Double getPrecioFinal(Integer hora): Devuelve el precio final que pagará el cliente por la mercancía utilizando como hora de envío la recibida como parámetro.

El problema se modelará a través de la clase **ProblemaMercanciasImpl**. Esta clase almacena el conjunto de clientes que esperan el envío en el conjunto ordenado denominado ***conjuntoClientes***. Además se conoce la ***hora inicial*** en la que el vehículo comienza a servir pedidos, y la ***hora final***, a partir de la cual el vehículo deja de servir pedidos. Por ejemplo, suponiendo que el vehículo empieza su recorrido a las 8:00, y puede entregar las mercancías hasta las 18:00, trabajando sin interrupción, se dispondría de 10 horas.



La clase **EstadoMercanciasImpl** permitirá almacenar toda la información del estado del problema (ver ficha adjunta). Los atributos serán:

- SortedSet<Cliente> **clientesPendientes**: Clientes para los que aún está por decidir si se enviará o no mercancía. Cada cliente del conjunto contendrá en el campo correspondiente a la hora de envío el valor -1, que indica que aún no se ha decidido hora de envío.
- List<Cliente> **clientesAtendidos**: Clientes para los que ya se ha decidido enviar la mercancía. Cada cliente de la lista contendrá actualizado el campo correspondiente a la hora de envío que se ha decidido.
- Integer **horaDisponible**: Hora a partir de la cual el vehículo para los repartos esta libre.
- Double **beneficioTotal**: Beneficio acumulado de los repartos a los clientes a los que se ha decidido enviar la mercancía.

SE PIDE:

a) Completar el apartado *add* y *remove* de la ficha proporcionada. Tenga en cuenta que tal cómo se puede ver en la ficha proporcionada, las alternativas posibles son los clientes para los cuáles es posible realizar la entrega de la mercancía antes de la hora final fijada para el problema.

b) Escribir el código completo del método *ryp*() de la clase ProblemaMercanciasImpl.

c) Escribir el código completo de los métodos *add*(Cliente) y *remove*() de la clase EstadoMercanciasImpl.

d) Completar el código de la clase interna IterableBT, que será utilizada en el método *getAlternativas*() para devolver las alternativas para cada estado del problema. Recuerde lo dicho en el apartado a).

e) Escribir el método *getAlternativas*() de la clase ProblemaMercanciasImpl. Tenga en cuenta que tendrá que utilizar la clase interna IterableBT creada en el apartado anterior.

```
class IterableBT implements Iterable<Cliente>, Iterator<Cliente> {

    Iterator<Cliente> itClientes;

    Cliente nextCliente;

    public IterableBT() {

            this.itClientes = Lists.newArrayList(estado.getClientesPendientes()).iterator();

    }

    public Iterator<Cliente> iterator() { return this; }

    public boolean hasNext() { COMPLETAR }

    public Cliente next() { return nextCliente; }

    public void remove() { throw new UnsupportedOperationException...)}

}
```

Tiempo estimado: 60 min. Puntuación: 5 puntos