**a) 2,5 puntos**

| ***BT*** | |
|---|---|
| ***Problema envío mercancías*** | |
| *Técnica: Backtracking* | |
| *Tamaño: n = tamaño(clientesPendientes)* | |
| *Propiedades Compartidas* | ***conjuntoClientes**: SortedSet<Cliente>*<br>***horaInicial**: Integer*<br>***horaFinal**: Integer* |
| *Propiedades del Estado* | ***clientesAtendidos**: List<Cliente>*<br>***clientesPendientes**: SortedSet<Cliente>*<br>***horaDisponible**: Integer*<br>***beneficioTotal**: Double* |
| *Solución: SolucionMercancias* | |
| *Alternativas: $A_e = \{c_i \in clientesPendientes \mid horaDisponible + duracion(c_i) <= horaFinal\}$* | |
| *Estado inicial:*<br>*clientesAtendidos = {}*<br>*clientesPendientes = copia(conjuntoClientes)*<br>*horaDisponible = horaInicial*<br>*beneficioTotal = 0* | |
| *Estado final: horaDisponible = horaFinal* $\vee$<br>$(\forall c_i \in clientesPendientes : horaDisponible + duracion(c_i) > horaFinal)$ | |
| ***Add:***<br>*clientesProcesados = clientesAtendidos + $c_i$*<br>*clientesPendientes = clientesPendientes – $c_i$*<br>*horaDisponible = horaDisponible + duracion($c_i$)*<br>*horaEnvio($c_i$) = horaDisponible*<br>*beneficioTotal = beneficioTotal + precioFinal($c_i$)*<br>***Remove:***<br>*clientesProcesados = clientesAtendidos - $c_i$*<br>*clientesPendientes = clientesPendientes + $c_i$*<br>*horaDisponible = horaDisponible - duracion($c_i$)*<br>*beneficioTotal = beneficioTotal - precioFinal($c_i$)*<br>*horaEnvio($c_i$) = -1*<br>*Donde*<br>*- duracion($c_i$) es el tiempo en horas necesarios para el suministro al cliente $c_i$*<br>*- precioFinal($c_i$) precio final que pagaría el cliente $c_i$ al ser suministrado*<br>*- horaEnvio($c_i$) permite recuperar y establecer la hora en la que se enviará la mercancía para el cliente $c_i$* | |

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

**(2,5 puntos en total)**

```
private void ryp() {
      SolucionMercancias solucion = estado.getSolucion();
      if (problema.esSolucion(solucion)) {
            soluciones.add(solucion);
            fin = esFin();
      }
      if (!fin && problema.isAlternativa(estado)) {
            Iterable<Cliente> it=problema.getAlternativas(estado);
            for (Cliente alternativa : it) {
```

```
                estado.add(alternativa);
                ryp();
                estado.remove();
            }
        }
}
```

c) Escribir el código completo de los métodos add(Cliente) y remove() de la clase EstadoMercanciasImpl.
**(2,5 puntos en total)**

```
public boolean add(Cliente c) {
      clientesPendientes.remove(c);
      clientesAtendidos.add(c);
      horaDisponible = horaDisponible + c.getDuracion();
      c.setHoraEnvio(horaDisponible);
      beneficioTotal = beneficioTotal + c.getPrecioFinal();
      return true;
}

public Cliente remove() {
      Cliente c =  clientesAtendidos.remove(
                              clientesAtendidos.size()-1);
      clientesPendientes.add(c);
      horaDisponible = horaDisponible - c.getDuracion();
      beneficioTotal = beneficioTotal - c.getPrecioFinal();
      c.setHoraEnvio(-1);
      return c;
}
```

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.
**(2 puntos en total)**

```
public boolean hasNext() {
      boolean enc = false;
      while(this.itClientes.hasNext() && !enc){
            nextCliente = itClientes.next();
            if (estado.getHoraDisponible() + nextCliente.getDuracion()
                              <= getHoraFinal()){
                  enc = true;
            }
      }
      return enc;
}
```

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

**(0,5 puntos en total)**

```
public Iterable<Cliente> getAlternativas(EstadoMercancias e) {
      return new IterableBT();
}
```