EJERCICIO 1 (1,5 puntos)

```
public class PasajeroBarco implements Comparable<PasajeroBarco> {
	private Integer id;
	private Boolean superviviente;
	private Integer clase;
	private String nombre;
	private Sexo sexo;
	private Integer edad;
	private Integer numHermanosOParejaAbordo, numPadresOHijosAbordo;
	private Double precioBillete;
	private String camarote;
	private String ciudadEmbarque;

	public PasajeroBarco(Integer id, Boolean superviviente, Integer clase,
		String nombre, Sexo sexo, Integer edad,
		Integer numHermanosOParejaAbordo, Integer numeroPadresOHijosAbordo,
		Double precioBillete, String cabina, String ciudadEmbarque) {

		Checkers.checkNoNull(id, superviviente, clase, edad,
			numeroPadresOHijosAbordo, numHermanosOParejaAbordo,
			precioBillete);
		Checkers.check("Valor numérico no válido", clase >= 0);
		Checkers.check("Valor numérico no válido", edad >= 0);
		Checkers.check("Valor numérico no válido",
			numHermanosOParejaAbordo >= 0);
		Checkers.check("Valor numérico no válido",
			numeroPadresOHijosAbordo >= 0);
		Checkers.check("Valor numérico no válido", precioBillete >= 0);

		this.id = id;
		this.superviviente = superviviente;
		this.clase = clase;
		this.nombre = nombre;
		this.sexo = sexo;
		this.edad = edad;
		this.numHermanosOParejaAbordo = numHermanosOParejaAbordo;
		this.numPadresOHijosAbordo = numeroPadresOHijosAbordo;
		this.precioBillete = precioBillete;
		this.camarote = cabina;
		this.ciudadEmbarque = ciudadEmbarque;
	}

	public Integer getNumFamiliares() {
		return getNumHermanosOParejaAbordo() + getNumPadresOHijosAbordo();
	}

	public int compareTo(PasajeroBarco o) {
		return getId().compareTo(o.getId());
	}
```

EJERCICIO 2 (8 puntos)

Tipo contenedor, operaciones básicas (0,5 puntos)

```
public class AccidenteNaval {

	private SortedSet<PasajeroBarco> pasajeros;

	public AccidenteNaval() {
		pasajeros = new TreeSet<>();
	}
	public AccidenteNaval(Stream<PasajeroBarco> s) {
		pasajeros = s.collect(Collectors.toCollection(TreeSet::new));
	}

	public String toString() {
		return pasajeros.stream()
				.map(x -> x.toString())
				.collect(Collectors.joining("\n"));
	}
```

a) Método getEdadMediaPorDebajoDe (0,5 puntos)

```
	public Double getEdadMediaPorDebajoDe(Boolean superviviente,
				Integer limite) {
		return pasajeros.stream()
				.filter(x -> x.getEdad() <= limite &&
					x.getSuperviviente().equals(superviviente))
				.collect(Collectors.summarizingDouble(x -> x.getEdad()))
				.getAverage();
	}
```

Alternativa 1:

```
	public Double getEdadMediaPorDebajoDe(Boolean superviviente,
				Integer limite) {
		return pasajeros.stream()
				.filter(x -> x.getEdad() <= limite &&
					x.getSuperviviente().equals(superviviente))
				.mapToInt(PasajeroBarco::getEdad)
				.average()
				.orElse(0);
	}
```

Alternativa 2:

```
	public Double getEdadMediaPorDebajoDe(Boolean superviviente,
				Integer limite) {
		return pasajeros.stream()
				.filter(x -> x.getEdad() <= limite &&
					x.getSuperviviente().equals(superviviente))
				.collect(Collectors.averagingDouble(x -> x.getEdad()))
				.doubleValue();
	}
```

b) Método getCiudadEmbarqueConMayorPrecio (1 punto)

```
public String getCiudadEmbarqueConMayorPrecio() {
	Comparator<? super PasajeroBarco> cmp =
		Comparator.comparing(PasajeroBarco::getPrecioBillete);

	return pasajeros.stream()
		.filter(x -> x.getCiudadEmbarque() != null)
		.max(cmp)
		.get()
		.getCiudadEmbarque();
}
```

c) Método getNumeroVictimasMenoresDePorEdad (1 punto)

```
public Map<Integer, Long> getNumeroVictimasMenoresDePorEdad(Integer edad){
	return pasajeros.stream()
		.filter(x -> x.getEdad() < edad)
		.collect(Collectors.groupingBy(
			PasajeroBarco::getEdad,
			Collectors.counting()));
}
```

d) Método getPorcentajeSupervivientesPorCiudadEmbarque (1,5 puntos)

```
public Map<String, Double> getPorcentajeSupervivientesPorCiudadEmbarque(){
	return pasajeros.stream()
		.filter(x -> x.getCiudadEmbarque() != null)
		.collect(Collectors.groupingBy(
			PasajeroBarco::getCiudadEmbarque,
			Collectors.collectingAndThen(
			   Collectors.mapping(
			      PasajeroBarco::getSuperviviente,
			      Collectors.toList()),
			   l -> 100.0*Collections.frequency(l, true)/l.size())));
}
```

Alternativa 1:

```
public Map<String, Double> getPorcentajeSupervivientesPorCiudadEmbarque(){
	Map<String, List<PasajeroBarco>> pasajerosPorCiudad =
		pasajeros.stream()
		.collect(
			Collectors.groupingBy(PasajeroBarco::getCiudadEmbarque));

	return pasajerosPorCiudad.entrySet().stream()
		.collect(Collectors.toMap(
			e-> e.getKey(),
			e-> porcentajeSupervivientes(e.getValue())));
}
```

Alternativa 2:

```
public Map<String, Double> getPorcentajeSupervivientesPorCiudadEmbarque(){
	Map<String, List<PasajeroBarco>> pasajerosPorCiudad =
		pasajeros.stream()
		.collect(
			Collectors.groupingBy(PasajeroBarco::getCiudadEmbarque));

	Map<String, Double> res = new HashMap<>();
	for (String s: pasajerosPorCiudad.keySet()) {
		res.put(s,porcentajeSupervivientes(pasajerosPorCiudad.get(s)));
	}
	return res;
}

private Double porcentajeSupervivientes(List<PasajeroBarco> pasajeros) {
	Integer totalPasajeros = pasajeros.size();
	Long numSupervivientes = pasajeros.stream()
		.filter(PasajeroBarco::getSuperviviente)
		.count();

	return numSupervivientes * 1.0 / totalPasajeros;
}
```

e) Método getMedianaPrecioTicket (1,5 puntos)

```
public Double getMedianaPrecioTicket(Boolean superviviente) {
	Long cantidad = pasajeros.stream().filter(x ->
		x.getSuperviviente().equals(superviviente)).count();

	return pasajeros.stream()
		.filter(x -> x.getSuperviviente().equals(superviviente))
		.map(PasajeroBarco::getPrecioBillete)
		.sorted()
		.skip(cantidad / 2)
		.findFirst()
		.get();
}
```

Alternativa:

```
public Double getMedianaPrecioTicket(Boolean superviviente) {
	List<Double> preciosOrd = pasajeros.stream()
		.filter(p -> p.getSuperviviente().equals(superviviente)
			&& p.getPrecioTicket() != null)
		.map(PasajeroBarco::getPrecioTicket)
		.sorted(Comparator.naturalOrder())
		.collect(Collectors.toList());

	int ind = preciosOrd.size() / 2;
	return preciosOrd.get(ind);
}
```

f) Método getParejasPorFuncion (2 puntos)

```
public SortedMap<String, SortedSet<PasajeroBarco>> getParejasPorFuncion
            (Function<PasajeroBarco, String> extractorPareja) {

      Comparator<PasajeroBarco> cmp =
            Comparator.comparing(PasajeroBarco::getSexo);

      return new TreeMap<>(pasajeros.stream()
            .collect(
                  Collectors.groupingBy(extractorPareja,
                        Collectors.toCollection(() -> new TreeSet<>(cmp))))
            .entrySet().stream()
            .filter(x -> x.getValue().size() == 2)
            .collect(
                  Collectors.toMap(x -> x.getKey(), x -> x.getValue())));
}
```

Alternativa:

```
public SortedMap<String, SortedSet<PasajeroBarco>> getParejasPorFuncion
            (Function<PasajeroBarco, String> extractorPareja) {

      Comparator<PasajeroBarco> cmp =
            Comparator.comparing(PasajeroBarco::getSexo);
      SortedMap<String,SortedSet<PasajeroBarco>> res = new TreeMap<>();

      for (PasajeroBarco p: pasajeros) {
            String s= extractorPareja.apply(p);
            if (res.containsKey(s)) {
                  res.get(s).add(p);
            } else {
                  SortedSet<PasajeroBarco> sp = new TreeSet<>(cmp);
                  res.put(s, sp);
            }
      }
      return res;
}
```

EJERCICIO 3 (0,5 puntos)

```
	private static PasajeroBarco parsearRegistro(String linea) {
		Checkers.checkNoNull(linea);
		String [] trozos = linea.split(";");
		String msg = String.format("Formato no válido : <%s>", linea);
		Checkers.check(msg, trozos.length == 11);
		Integer id = new Integer(trozos[0].trim());
		Boolean superviviente = parsearSuperviviente(trozos[1].trim());
		Integer clase = parsearEnteroNulo(trozos[2].trim());
		String nombre = trozos[3].trim();
		Sexo sexo = Sexo.valueOf(trozos[4].trim().toUpperCase());
		Integer edad = parsearEnteroNulo(trozos[5].trim());
		Integer hermanos = parsearEnteroNulo(trozos[6].trim());
		Integer padres = parsearEnteroNulo(trozos[7].trim());
		Double precio = new Double(trozos[8].trim());
		String cabina = trozos[9].trim();
		String ciudadEmbarque = trozos[10].trim();
		return new PasajeroBarco(id, superviviente, clase, nombre,
			sexo, edad, hermanos, padres, precio, cabina, ciudadEmbarque);
	}

	private static Integer parsearEnteroNulo(String str) {
		return str.length() == 0 ? null : Integer.valueOf(str);
	}

	private static Boolean parsearSuperviviente(String dato) {
		Integer n = new Integer(dato);
		Boolean res = false;
		if (n > 0) {
			res = true;
		}
		return res;
	}

	private static Boolean parsearSuperviviente(String dato) {
		return dato.equals("1");
	}
```