Ejercicio 1

```
public record Evaluacion(String hamburgueseria, Integer presentacion, Integer punto,
		Integer ingredientes, Integer pan) {
	public Evaluacion{
		Checkers.check("La puntuación de la presentación es incorrecta",
				presentacion >= 0 && presentacion <= 10);
		Checkers.check("La puntuación del punto de la carne es incorrect0",
				punto >= 0 && punto <= 10);
		Checkers.check("La puntuación de la calidad de los ingredientes es incorrecta",
				ingredientes >= 0 && ingredientes <= 10);
		Checkers.check("La puntuación de la calidad del pan es incorrecta",
				pan >= 0 && pan <= 10);
	}

	public Double puntuacionFinal() {
		Double res = (presentacion() + punto() + ingredientes() + pan()) / 4.0;
		return res;
	}
}
```

Ejercicio 2

```
public class Visita implements Comparable<Visita>{
	private String email;
	private String ciudad;
	private String codigoPostal;
	private LocalDateTime entrada;
	private LocalDateTime salida;
	private Double temperatura;
	private List<Evaluacion> evaluaciones;

	public Visita(String email, String ciudad, String codigoPostal, LocalDateTime entrada,
			LocalDateTime salida, Double temperatura, List<Evaluacion> evaluaciones) {
		Checkers.check("Visita::Momentos de entrada y salida incorrectos",
				entrada.isBefore(salida));
		Checkers.check("Visita::Email incorrecto",
				email.contains("@"));
		Checkers.check("Visita::La lista de evaluaciones no puede estar vacía",
				!evaluaciones.isEmpty());
		this.email = email;
		this.ciudad = ciudad;
		this.codigoPostal = codigoPostal;
		this.entrada = entrada;
		setSalida(salida);
		setTemperatura(temperatura);
		this.evaluaciones = new ArrayList<>(evaluaciones);
	}

	public Double getTemperatura() {
		return temperatura;
	}

	public void setTemperatura(Double temperatura) {
		this.temperatura = temperatura;
	}

	public String getEmail() {
		return email;
	}
```

```
public String getCiudad() {
	return ciudad;
}

public String getCodigoPostal() {
	return codigoPostal;
}

public LocalDateTime getEntrada() {
	return entrada;
}

public void setSalida(LocalDateTime salida) {
	Checkers.check("Visita::El día de entrada y de salida no son el mismo",
			salida.toLocalDate().equals(getEntrada().toLocalDate()));
	Checkers.check("Visita::Momentos de entrada y salida incorrectos",
			getEntrada().isBefore(salida));
	this.salida = salida;
}

public LocalDateTime getSalida() {
	return salida;
}

public List<Evaluacion> getEvaluaciones() {
	return new ArrayList<Evaluacion>(this.evaluaciones);
}

public Integer getNumEvaluaciones(){
	return getEvaluaciones().size();
}

private Double getPuntuacionMediaTotal() {
	Double suma = 0.0;
	for (Evaluacion ev: getEvaluaciones()) {
		suma += ev.puntuacionFinal();
	}
	return suma / getEvaluaciones().size();
}

public Paladar getPaladar() {
	Paladar res = Paladar.MEDIO;
	if (getPuntuacionMediaTotal() >= 9) {
		res = Paladar.BAJO;
	} else if(getPuntuacionMediaTotal() <= 6) {
		res = Paladar.ALTO;
	}
	return res;
}

public Duration getTiempoTranscurrido() {
	return Duration.between(getEntrada(), getSalida());
}

public String toString() {
	return "Visita [email=" + email + ", ciudad=" + ciudad + ", cp=" + codigoPostal + ",
		entrada=" + entrada + ", salida=" + salida + ", temperatura=" + temperatura
		+ ", evaluaciones=" + evaluaciones + "]";
}

public int hashCode() {
	return Objects.hash(email, entrada, salida);
}
```

```
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Visita other = (Visita) obj;
		return Objects.equals(email, other.email) && Objects.equals(entrada, other.entrada)
				&& Objects.equals(salida, other.salida);
	}

	public int compareTo(Visita o) {
		int res = getEntrada().compareTo(o.getEntrada());

		if (res == 0) {
			res = getSalida().compareTo(o.getSalida());
			if (res == 0) {
				res = getEmail().compareTo(o.getEmail());
			}
		}
		return res;
	}
}
```

Ejercicio 3:

```
private static Visita parseaVisita(String lineaCSV) {
	Checkers.checkNoNull(lineaCSV);
	String [] trozos = lineaCSV.split(";");
	Checkers.check("FactoriaVisitas::Formato no válido", trozos.length == 7);
	String email = trozos[0].strip();
	String ciudad = trozos[1].strip();
	String cp = trozos[2].strip();
	LocalDateTime entrada = LocalDateTime.parse(trozos[3].strip(),
		DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm"));
	Double temperatura = Double.valueOf(trozos[4].strip());
	LocalDateTime salida = LocalDateTime.parse(trozos[5].strip(),
		DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm"));
	List<Evaluacion> evaluaciones = parseaEvaluaciones(trozos[6].trim());
	return new Visita(email, ciudad, cp, entrada, salida, temperatura, evaluaciones);
}

private static List<Evaluacion> parseaEvaluaciones(String cad) {
	String clean = cad.replace("[","").replace("]", "");
	String [] trozos = clean.split("-");
	List<Evaluacion> res = new ArrayList<>();
	for (String trozo:trozos) {
		res.add(parseaEvaluacion(trozo.strip()));
	}
	return res;
}

private static Evaluacion parseaEvaluacion(String trozo) {
	String clean = trozo.replace("(","").replace(")", "");
	String [] trozos = clean.split(":");
	Checkers.check("Formato evaluación no válido", trozos.length == 2);

	String nombre = trozos[0].trim();
	String[] puntuacionesStr = trozos[1].split(",");
	Checkers.check("Puntuaciones incorrectas", puntuacionesStr.length == 4);
```

```
        Integer presentacion = Integer.valueOf(puntuacionesStr[0].trim());
        Integer punto = Integer.valueOf(puntuacionesStr[1].trim());
        Integer ingredientes = Integer.valueOf(puntuacionesStr[2].trim());
        Integer pan = Integer.valueOf(puntuacionesStr[3].trim());

        return new Evaluacion(nombre, presentacion, punto, ingredientes, pan);
}
```

Ejercicio 4

Tipo Competición

```
public class Competicion {
        private SortedSet<Visita> visitas;

        private static final Comparator<Visita> CMP_CP =
                Comparator.comparing(Visita::getCodigoPostal)
                .thenComparing(Comparator.naturalOrder());

        public SortedSet<Visita> getVisitas() {
                SortedSet<Visita> ss= new TreeSet<>(CMP_CP);
                ss.addAll(visitas);
                return ss;
        }

        public Competicion(Stream<Visita> stream) {
                this.visitas = stream.collect(Collectors.toCollection(() -> new TreeSet<>(CMP_CP)));
        }
```

Ejercicio 4.1

```
public SortedSet<String> getEmailsOrdenados(Duration d) {

        Double tMed = getVisitas().stream()
                .mapToDouble(Visita::getTemperatura)
                .average()
                .getAsDouble();
        //La media también se puede calcular así:
        //Double tMed = getVisitas().stream()
        //      .collect(Collectors.averagingDouble(Visita::getTemperatura));


        Predicate<Visita> pred = v -> v.getTemperatura().compareTo(tMed) < 0 &&
                v.getTiempoTranscurrido().compareTo(d) > 0;
        //También se pueden definir dos predicados y combinarlos con and:
        //Predicate <Visita> predDuracion = v -> v.getTiempoTranscurrido().compareTo(d) > 0 ;
        //Predicate <Visita> predTemperatura = v -> v.getTemperatura().compareTo(tMed) < 0;
        //Predicate <Visita> pred = predDuracion.and(predTemperatura);

        return getVisitas().stream()
                .filter(pred)
                .map(Visita::getEmail)
                .collect(Collectors.toCollection(TreeSet::new));
}
```

Ejercicio 4.2

```
public Integer getTotalVisitasComilones(){
        Double mediaVisitas = getVisitas().stream()
                .mapToInt(Visita::getNumEvaluaciones)
                .average()
                .getAsDouble();
```

```
                //La media también se puede calcular así
                //Double mediaVisitas = getVisitas().stream()
                //      .collect(Collectors.averagingInt(Visita::getNumEvaluaciones));

        Long res = getVisitas().stream()
                .filter(v -> v.getNumEvaluaciones()>mediaVisitas)
                .count();
        return res.intValue();
}
```

Ejercicio 4.3

```
public String getPeorHamburgueseriaPorCalidadIngredientes() {
        return getMediaIngredientesHamburgueseria().entrySet().stream()
                .min(Comparator.comparing(Map.Entry::getValue))
                .map(Map.Entry::getKey)
                .orElse(null);
}

private Map<String, Double> getMediaIngredientesHamburgueseria(){
        return getVisitas().stream()
                .flatMap(v -> v.getEvaluaciones().stream())
                .collect(Collectors.groupingBy(Evaluacion::hamburgueseria,
                        Collectors.averagingInt(Evaluacion::ingredientes)));
}
```

Ejercicio 4.4

```
public Map<String, String> getTopComilonPorCPEnDia(LocalDate dia) {
        Comparator<Visita> cmp = Comparator.comparing(Visita::getNumEvaluaciones)
                .thenComparing(Visita::getEntrada);

        return getVisitas().stream()
                .filter(v -> v.getEntrada().toLocalDate().equals(dia))
                .collect(Collectors.groupingBy(
                        Visita::getCodigoPostal,
                        Collectors.collectingAndThen(
                                Collectors.maxBy(cmp),
                                o -> o.get().getEmail())));

                //La función para extraer el mail del optional también se puede poner como
                // o -> o.map(Visita::getEmail).get()
}
```

Solución alternativa

```
public Map<String, String> getTopComilonPorCPEnDia2(LocalDate dia) {
        Map<String, List<Visita>> m = getVisitas().stream()
                .filter(v -> v.getEntrada().toLocalDate().equals(dia))
                .collect(Collectors.groupingBy(Visita::getCodigoPostal));

        return  m.entrySet().stream()
                .collect(Collectors.toMap(
                        e -> e.getKey(),
                        e -> obtenerTopComilon(e.getValue())
                ));
        }

private String obtenerTopComilon(List<Visita> visitas) {
        Comparator<Visita> cmp = Comparator.comparing(Visita::getNumEvaluaciones)
                .thenComparing(Visita::getEntrada);
```

```
        return visitas.stream()
                .max(cmp)
                .map(Visita::getEmail)
                .get();
}
```

Ejercicio 4.5

```
public String getHamburgueseriaGanadora() {
        String res = null;
        Double puntuacionMax = null;
        Map<String, List<Double>> mAux = getPuntuacionesHamburgueseria();
        for (Entry<String, List<Double>> e: mAux.entrySet()) {
                Double media = media(e.getValue());
                if (res == null || media > puntuacionMax) {
                        res = e.getKey();
                        puntuacionMax = media;
                }
        }
        return res;
}

public String getHamburgueseriaGanadora2() {
        Map<String, List<Double>> mAux = getPuntuacionesHamburgueseria();
        Entry<String, List<Double>> res = null;
        BinaryOperator<Map.Entry<String, List<Double>>> bo =
                BinaryOperator.maxBy(Comparator.comparing(e -> media(e.getValue())));
        for (Entry<String, List<Double>> e: mAux.entrySet()) {
                if (res == null) {
                        res = e;
                } else {
                        res = bo.apply(res, e);
                }
        }
        return res.getKey();
}

private Map<String, List<Double>> getPuntuacionesHamburgueseria() {
        Map<String, List<Double>> mAux = new HashMap<>();
        for (Visita v: getVisitas()) {
                for (Evaluacion e: v.getEvaluaciones()) {
                        String clave = e.hamburgueseria();
                        if (mAux.containsKey(clave)) {
                                mAux.get(clave).add(e.puntuacionFinal());
                        } else {
                                List<Double> l = new ArrayList<Double>();
                                l.add(e.puntuacionFinal());
                                mAux.put(clave, l);
                        }
                }
        }
        return mAux;
}

private Double media(List<Double> lista) {
        Double res = 0.0;
        if (!lista.isEmpty()) {
                for (Double n: lista) {
                        res += n;
                }
                res = res / lista.size();
        }
        return res;
```

```
}
```

Test

```
public class TestCompeticion {

	public static void main(String[] args) {
		Competicion competicion = FactoriaVisitas.leeCompeticion("./data/evaluaciones.csv");
		System.out.println("FACTORÍA"+" =".repeat(80));
		testCreaCompeticionBurger(competicion);
		System.out.println("\nEJ1"+" =".repeat(80));
		testGetEmailsOrdenados(competicion, Duration.ofMinutes(240));
		testGetEmailsOrdenados (competicion, Duration.ofMinutes(300));
		System.out.println("\nEJ2"+" =".repeat(80));
		testGetTotalVisitasComilones(competicion);
		System.out.println("\nEJ3"+" =".repeat(80));
		testGetPeorHamburgueseriaPorCalidadIngredientes(competicion);
		System.out.println("\nEJ4"+" =".repeat(80));
		testGetTopComilonPorCPEnDia(competicion, LocalDate.of(2024, 6, 2));
		System.out.println("\nEJ5"+" =".repeat(80));
		testGetHamburgueseriaGanadora(competicion);
	}

	private static void testCreaCompeticionBurger(Competicion comp) {
		try {
			SortedSet<Visita> visitas = comp.getVisitas();
			System.out.println(String.format("Leidas %d visitas", visitas.size()));
			System.out.println("Primera visita: "+ visitas.first()+"\n");
			System.out.println("Última visita: "+ visitas.last());

		} catch (Exception e) {
			System.out.println(e.getMessage());
		}
	}

	private static void testGetEmailsOrdenados(Competicion comp, Duration d) {
		try {
			SortedSet<String> res = comp.getEmailsOrdenados(d);
			String msg = String.format(
				"Los emails de las visitas con duración mayor a %d minutos son:\n%s",
				d.toMinutes(), res);
			System.out.println(msg);
			} catch (Exception e) {
				System.out.println(e.getMessage());
			}
	}

	private static void testGetTotalVisitasComilones(Competicion comp) {
		try {
			Integer res = comp.getTotalVisitasComilones();
			String msg = String.format(
				"El total de visitas con un nº de evaluaciones mayor a la media es:\n%d",
				res);
			System.out.println(msg);
			} catch (Exception e) {
				System.out.println(e.getMessage());
			}
	}
```

```
	private static void testGetPeorHamburgueseriaPorCalidadIngredientes(Competicion comp) {
		try {
			String res = comp.getPeorHamburgueseriaPorCalidadIngredientes();
			String msg = String.format(
				"Si se tiene en cuenta solo la calidad de ingredientes" +
				", la peor hamburguesería es:\n%s", res);
			System.out.println(msg);
			} catch (Exception e) {
				System.out.println(e.getMessage());
			}
	}

	private static void testGetTopComilonPorCPEnDia(Competicion comp, LocalDate f) {
		try {
			Map<String, String> res = comp.getTopComilonPorCPEnDia(f);
			String msg = String.format(
				"En el día %s la persona que más comió de cada CP es:\n%s",
				f, res);
			System.out.println(msg);
			} catch (Exception e) {
				System.out.println(e.getMessage());
			}
	}

	private static void testGetHamburgueseriaGanadora(Competicion comp) {
		try {
			String res = comp.getHamburgueseriaGanadora();
			String msg = String.format("La hamburguesería ganadora del campeonato es:\n%s",
				res);
			System.out.println(msg);
			} catch (Exception e) {
				System.out.println(e.getMessage());
			}
	}

}
```