Ejercicio 1

```
public record Participante(String id, String nombre, String apellidos, Integer edad, Sexo sexo,
Duration duracion) {

	public Participante {
		Checkers.check("La edad mínima de participación es 17 años", edad >= 17);
	}

	public Categoria getCategoria() {
		Categoria res = null;

		String textoSexo = null;
		if (sexo.equals(Sexo.HOMBRE)) {
			textoSexo = "M";
		} else if (sexo.equals(Sexo.MUJER)) {
			textoSexo = "F";
		}

		String categoria = null;
		if (edad() >= 17 && edad() <= 20) {
			categoria =" JUNIOR";
		} else if (edad() >= 21 && edad() <= 23) {
			categoria = "PROMESA";
		} else if (edad() >= 24 && edad() <= 39) {
			categoria = "SENIOR";
		} else if (edad() >= 40) {
			categoria = "VETERANO";
		}
		if (categoria != null && textoSexo != null) {
			res = Categoria.valueOf(categoria + "_" + textoSexo);
		}
		return res;
	}
}
```

Ejercicio 2

```
public class Carrera implements Comparable<Carrera> {

	private String id;
	private String localidad;
	private LocalDateTime fechahora;
	private Modalidad modalidad;
	private Integer distancia;
	private Integer desnivel;
	private List<Participante> participantes;

	public Carrera(String id, String localidad, LocalDateTime fechaHora,
		Modalidad modalidad, Integer distancia, Integer desnivel) {
		this.id = id;
		this.localidad = localidad;
		this.fechaHora = fechaHora;
		this.modalidad = modalidad;
		setDistancia(distancia);
		setDesnivel(desnivel);
		this.participantes = new ArrayList<>();
	}

	private void setDistancia (Integer distancia) {
		Checkers.check("Distancia mínima superior o igual a 7Km", distancia >= 7000);
		this.distancia = distancia;
```

```
}

private void setDesnivel(Integer desnivel) {
	Checkers.check("Desnivel debe ser >= a cero y menor o igual a 1 Km",
		desnivel >= 0 && desnivel <= 1000);
	this.desnivel = desnivel;
}

public void añadeParticipantes(List<Participante> participantes) {
	for (Participante p: participantes) {
		if (!this.participantes.contains(p)) {
			this.participantes.add(p);
		}
	}
}

public String getId() {
	return id;
}

public String getLocalidad() {
	return localidad;
}

public LocalDateTime getFechahora() {
	return fechahora;
}

public Modalidad getModalidad() {
	return modalidad;
}

public Integer getDistancia() {
	return distancia;
}

public Integer getDesnivel() {
	return desnivel;
}

public List<Participante> getParticipantes() {
	return new ArrayList<>(participantes);
}

public String toString() {
	return "Carrera [id=" + id + ", localidad=" + localidad + ", fechahora="
		+ fechahora + ", tipo=" + modalidad
		+ ", distancia=" + distancia + ", desnivel=" + desnivel + "]";
}

public int hashCode() {
	final int prime = 31;
	int result = 1;
	result = prime * result + ((distancia == null) ? 0 : distancia.hashCode());
	result = prime * result + ((fechahora == null) ? 0 : fechahora.hashCode());
	result = prime * result + ((id == null) ? 0 : id.hashCode());
	result = prime * result + ((localidad == null) ? 0 : localidad.hashCode());
	return result;
}

public boolean equals(Object obj) {
	if (this == obj)
		return true;
```

```
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Carrera other = (Carrera) obj;
		if (distancia == null) {
			if (other.distancia != null)
				return false;
		} else if (!distancia.equals(other.distancia))
			return false;
		if (fechahora == null) {
			if (other.fechahora != null)
				return false;
		} else if (!fechahora.equals(other.fechahora))
			return false;
		if (id == null) {
			if (other.id != null)
				return false;
		} else if (!id.equals(other.id))
			return false;
		if (localidad == null) {
			if (other.localidad != null)
				return false;
		} else if (!localidad.equals(other.localidad))
			return false;
		return true;
	}

	public int compareTo(Carrera c) {
		int res = getFechaHora().compareTo(c.getFechaHora());
		if (res ==0 ) {
			res = getLocalidad().compareTo(c.getLocalidad());
			if (res == 0) {
				res= getDistancia().compareTo(c.getDistancia());
				if (res==0) {
					res= getId().compareTo(c.getId());
				}
			}
		}
		return res;
	}
}
```

Ejercicio 3

```
public class FactoriaCarreras {

	public static Carreras leeCarreras(String nomFichCarreras, String nomFichParticipantes) {
		Carreras res = null;
		try {
			List<Carrera> carreras = Files.lines(Paths.get(nomFichCarreras))
					.skip(1)
					.map(FactoriaCarreras::parseaCarrera)
					.collect(Collectors.toList());
			Map<String, List<Participante>> participantes =
					leeParticipantes(nomFichParticipantes);
			for (Carrera c: carreras) {
				c.añadeParticipantes(participantes.get(c.getId()));
			}
			res=new Carreras(carreras.stream());
		} catch (IOException e) {
			e.printStackTrace();
		}
```

```
        return res;
}
```

Solución alternativa

```
public static Carreras leeCarreras2(String nomFichCarreras, String nomFichParticipantes) {
        Carreras res = null;
        try {
                Map<String, List<Participante>> participantes =
                                                leeParticipantes(nomFichParticipantes);
                Stream<Carrera> carreras = Files.lines(Paths.get(nomFichCarreras))
                                .skip(1)
                                .map(FactoriaCarreras::parseaCarrera)
                                .peek(c ->
                                        c.añadeParticipantes(participantes.get(c.getId()));

                res = new Carreras(carreras);
        } catch (IOException e) {
                e.printStackTrace();
        }
        return res;
}

private static Carrera parseaCarrera(String strCarrera) {
        Checkers.checkNoNull(strCarrera);
        String[] trozos = strCarrera.split(",");
        Checkers.check("Formato no válido <" + strCarrera + ">", trozos.length == 6);
        LocalDateTime fechaHora = LocalDateTime.parse(trozos[2].trim(),
                                DateTimeFormatter.ofPattern("yyyy-M-d H:m"));
        //También es válido el patrón "yyyy-MM-dd H:mm"
        Modalidad modalidad = Modalidad.valueOf(trozos[3].trim());
        Integer distancia = Integer.valueOf(trozos[4].trim());
        Integer desnivel = Integer.valueOf(trozos[5].trim());
        return new Carrera(trozos[0].trim(),
                                trozos[1].trim(),
                                fechaHora,
                                modalidad,
                                distancia,
                                desnivel);
}

private static Map<String, List<Participante>> leeParticipantes(String nombre_fichero) {
        Map<String, List<Participante>> m = new HashMap<>();

        try {
                m = Files.lines(Paths.get(nombre_fichero),StandardCharsets.UTF_8)
                                .skip(1)
                                .map(FactoriaCarreras::parseaParticipante)
                                .collect(Collectors.groupingBy(Participante::id));
        } catch (IOException e) {
                e.printStackTrace();
        }

        return m;
}

public static Participante parseaParticipante(String linea) {
        // 0,Ana,Torres García,19,HOMBRE,1:08:13.736016
        // la duración tiene que tener este formato en el fichero: PT1H8M13.736016S
        Checkers.checkNoNull(linea);
        String[] part = linea.split(",");
        Checkers.check("Formato de línea incorrecto", part.length == 6);
```

```
                Sexo sexo = Sexo.valueOf(part[4].trim());
                Duration dur = Duration.parse(part[5].trim());

                return new Participante(part[0].trim(),
                                        part[1].trim(),
                                        part[2].trim(),
                                        Integer.valueOf(part[3].trim()),
                                        sexo,
                                        dur);
        }
}
```

Ejercicio 4

Tipo Carreras

```
public class Carreras {

        private SortedSet<Carrera> carreras;
        private static final Comparator<Carrera> COMP_FECHA_ORD =
                        Comparator.comparing(Carrera::getFechaHora)
                                        .thenComparing(Comparator.naturalOrder());

        public Carreras(Stream<Carrera> strCarreras) {
                this.carreras = strCarreras.collect(
                                Collectors.toCollection(() -> new TreeSet<>(COMP_FECHA_ORD)));
        }

        public SortedSet<Carrera> getCarreras() {
                SortedSet<Carrera> res = new TreeSet<>(COMP_FECHA_ORD);
                res.addAll(carreras);
                return res;
        }

        // Se admiten también esta soluciones para el constructor y el getter:
        public Carreras(Stream<Carrera> st_carrs) {
                carreras = st_carrs.collect(Collectors.toCollection(TreeSet::new));
        }

        public SortedSet<Carrera> getCarreras(){
                return new TreeSet<>(carreras);
        }

        public Integer getNumCarreras() {
                return carreras.size();
        }

        public List<Participante> participantesUltimaCarrera(){
                return carreras.last().getParticipantes();
        }

        public int hashCode() {
                return Objects.hash(carreras);
        }

        public boolean equals(Object obj) {
                if (this == obj)
                        return true;
                if (!(obj instanceof Carreras))
                        return false;
                Carreras other = (Carreras) obj;
```

```
		return Objects.equals(carreras, other.carreras);
	}

	public String toString() {
		return "Carreras [getCarreras()=" + getCarreras() + "]";
	}
```

Ejercicio 4.1

```
public Carrera carreraMayorDesnivelParticipante(String nombre, String apellidos) {
	return carreras.stream()
			.filter(c -> c.buscaParticipante(nombre,apellidos) != null)
			.max(Comparator.comparing(Carrera::getDesnivel)
					.thenComparing(Comparator.comparing(
						Carrera::getDistancia)
						.reversed()))
			.orElse(null);
}
```

Método buscaParticipante en la clase Carrera:

```
	public Participante buscaParticipante(String nombre, String apellido) {
		return getParticipantes().stream()
				.filter(p -> p.nombre().equals(nombre) &&
					p.apellidos().equals(apellido))
				.findAny() // También es válido findFirst
				.orElse(null);
	}
```

Ejercicio 4.2

```
public Double tiempoMedioCarrera(String idCarr) {
	Carrera carr = buscaCarrera(idCarr);
	return carr.tiempoMedioPorKm();
}

private Carrera buscaCarrera(String id) {
		return carreras.stream()
				.filter(c -> c.getId().equals(id))
				.findFirst()
				.get();
}
```

Método tiempoMedioPorkm en la clase Carrera (dos opciones):

```
	public Double tiempoMedioPorKm() {
		return getParticipantes().stream()
			.collect(Collectors.averagingDouble(
				p -> p.duracion().toMinutes() / (getDistancia() / 1000.0)));
	}

	public Double tiempoMedioPorKm2() {
		return getParticipantes().stream()
				.mapToDouble(
					p -> p.duracion().toMinutes() / (getDistancia() / 1000.0))
				.average()
				.getAsDouble();
	}
```

Ejercicio 4.3

```
public SortedMap<Categoria, String> ganadoresPorCategoria(String idCarrera) {
	Carrera carr = buscaCarrera(idCarrera);

	Function<Optional<Participante>, String> fApeNom =
			opt -> opt.get().apellidos() + " " + opt.get().nombre();

	return carr.getParticipantes().stream()
			.collect(Collectors.groupingBy(
				Participante::getCategoria,
				TreeMap::new,
				Collectors.collectingAndThen(
					Collectors.minBy(
						Comparator.comparing(Participante::duracion)),
						fApeNom)));
}
```

Ejercicio 4.4

```
public Map<String, Integer> posicionesParticipante(String nombre, String apellidos) {

	Sexo sexo = carreras.stream()
			.map(c -> c.buscaParticipante(nombre, apellidos))
			.findAny()
			.get()
			.sexo();

	Map<String, Integer> posiciones = new HashMap<>();
	for (Carrera c:carreras) {
		int pos = c.posicionParticipante(nombre, apellidos, sexo);
		if (pos >= 0) {
			posiciones.put(c.getId(), pos + 1);
		}
	}
	return posiciones;
}
```

Método posicionParticipante en la clase Carrera:

```
	public Integer posicionParticipante(String n, String ap, Sexo sexo) {
		List<Participante> clasificacion = new ArrayList<>();
		for (Participante p: getParticipantes()) {
			if (p.sexo().equals(sexo)) {
				clasificacion.add(p);
			}
		}
		Collections.sort(clasificacion,
				Comparator.comparing(Participante::duracion));

		int pos = -1;
		for (int i = 0; i < clasificacion.size(); i++) {
			Participante p = clasificacion.get(i);
			if (p.nombre().equals(n) && p.apellidos().equals(ap)) {
				pos = i;
			}
		}
		return pos;
	}
```

Ejercicio 4.5

```
public Categoria categoriaMasParticipantes() {
	Map<Categoria,Long> partPorCat = carreras.stream()
			.flatMap(c -> c.getParticipantes().stream())
			.collect(Collectors.groupingBy(
					Participante::getCategoria,
					Collectors.counting()));

	return partPorCat.entrySet().stream()
			.max(Comparator.comparing(e -> e.getValue()))
			.get()
			.getKey();
}
```

Test

```
public class TestCarreras {

  public static void main(String[] args) {
	Carreras c = FactoriaCarreras.leeCarreras("data/carreras.csv",
							"data/participantes.csv");
	testFactoria(c);

	System.out.println("\n----- Carrera con mayor desnivel de participante: --------");
	testCarreraMayorDesnivel(c, "Elena", "Blanco Vázquez");
	testCarreraMayorDesnivel(c, "Alejandro", "Ruiz Blanco");

	System.out.println("\n----- Tiempo medio de carrera: --------");
	testTiempoMedioCarrera(c, "medmar_sev_24");
	testTiempoMedioCarrera(c, "tr_cobre_23");

	System.out.println("\n----- Ganadores por categoría: --------");
	testGanadoresPorCategoria(c, "tr_lima_pedr_24");

	System.out.println("\n----- Posiciones participante: --------");
	testPosicionesParticipante (c, "Sara", "Navarro Díaz");
	testPosicionesParticipante (c, "Luis", "Blanco Pérez");

	System.out.println("\n----- Categoría con mayor número de participantes: --------");
	testCategoriaMasParticipantes(c);
  }

  private static void testCategoriaMasParticipantes(Carreras c) {
	try {
		Categoria res = c.categoriaMasParticipantes();
		String msg = String.format("%s", res);
		System.out.println(msg);
	} catch (Exception e) {
		System.out.println(e.getMessage());
	}
  }

  private static void testPosicionesParticipante(Carreras c, String nombre,
			String apellidos) {
	try {
		Map<String, Integer> res = c.posicionesParticipante(nombre, apellidos);
		String msg = String.format("%s %s: ", nombre, apellidos);
		System.out.println(msg);
		res.entrySet().stream()
			.forEach(System.out::println);
```

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

	private static void testGanadoresPorCategoria(Carreras c, String idCarrera) {
		try {
			Map<Categoria, String> res = c.ganadoresPorCategoria(idCarrera);
			String msg = String.format("%s:", idCarrera);
			System.out.println(msg);
			res.entrySet().stream()
				.forEach(System.out::println);

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

	private static void testTiempoMedioCarrera(Carreras c, String idCarrera) {
		try {
			Double res = c.tiempoMedioCarrera(idCarrera);
			String msg = String.format("%s: %f", idCarrera, res);
			System.out.println(msg);
		} catch (Exception e) {
			System.out.println(e.getMessage());
		}
	}

	private static void testCarreraMayorDesnivel(Carreras c, String nombre, String apellidos) {
		try {
			Carrera res = c.carreraMayorDesnivelParticipante(nombre, apellidos);
			String msg = String.format("%s %s:%s", nombre, apellidos, res);
			System.out.println(msg);
		} catch (Exception e) {
			System.out.println(e.getMessage());
		}
	}

	private static void testFactoria(Carreras c) {
		try {
			System.out.println("Se ha leído información de " + c.getNumeroCarreras() +
					" carreras");

			System.out.println("-----");
			System.out.println("En la última carrera han participado " +
					c.getParticipantesUltimaCarrera().size()
					  + " participantes.");
		} catch (Exception e) {
			System.out.println(e.getMessage());
		}
	}

}
```