Ejercicio 1

```
public record Equipo(String nombre, Integer clasificacion, String estadio) {
	public Equipo {
		Checkers.check("La posición en la clasificación debe estar entre 1 y 20.",
				clasificacion >= 1 && clasificacion <= 20);
		Checkers.check("La propiedad estadio debe comenzar por 'Estadio'",
				estadio.startsWith("Estadio"));
	}
}
```

Ejercicio 2

```
public class Jugador implements Comparable<Jugador> {
	private String nombre;
	private LocalDate fechaNacimiento;
	private Posicion posicion;
	private Integer goles;
	private Integer asistencias;
	private Equipo equipo;
	private List<String> lesiones;

	public Jugador(String nombre, LocalDate fechaNacimiento, Posicion posicion,
			Integer goles, Integer asistencias, Equipo equipo,
			List<String> lesiones) {
		this.nombre = nombre;
		Checkers.check("La fecha de nacimiento no debe ser anterior al 1970-01-01",
				fechaNacimiento.isAfter(LocalDate.of(1970, 1, 1)));
		this.fechaNacimiento = fechaNacimiento;
		this.posicion = posicion;
		Checkers.check("Los goles deben ser mayores o iguales que cero", goles>=0);
		this.goles = goles;
		this.asistencias = asistencias;
		this.equipo = equipo;
		this.lesiones = new ArrayList<>(lesiones);
	}

	public String getNombre() {
		return nombre;
	}

	public LocalDate getFechaNacimiento() {
		return fechaNacimiento;
	}

	public Posicion getPosicion() {
		return posicion;
	}

	public Integer getGoles() {
		return goles;
	}

	public Integer getAsistencias() {
		return asistencias;
	}
```

```
	public Equipo getEquipo() {
		return equipo;
	}

	public List<String> getLesiones() {
		return new ArrayList<String>(lesiones);
	}

	public Integer getEdad() {
		return Period.between(fechaNacimiento, LocalDate.now()).getYears();
	}

	public Boolean esCampeon() {
		return getEquipo().clasificacion().equals(1);
	}

	public Boolean mismaLesionConsecutiva() {
		Boolean res = false;
		List<String> lesiones = getLesiones();
		for (int i = 0; i < lesiones.size() - 1; i++) {
			if (lesiones.get(i).equals(lesiones.get(i + 1))) {
				res = true;
			}
		}
		return res;
	}

	public int hashCode() {
		return Objects.hash(fechaNacimiento, nombre);
	}

	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Jugador other = (Jugador) obj;
		return Objects.equals(fechaNacimiento, other.fechaNacimiento) &&
			Objects.equals(nombre, other.nombre);
	}

	public String toString() {
		return "Jugador [nombre=" + nombre + ", fechaNacimiento=" + fechaNacimiento
			+ ", posicion=" + posición + ", goles=" + goles + ", asistencias="
			+ asistencias + ", equipo=" + equipo + ", lesiones=" + lesiones
			+ "]";
	}

	public int compareTo(Jugador o) {
		int res = this.getFechaNacimiento().compareTo(o.getFechaNacimiento());
		if (res == 0) {
			res = this.getNombre().compareTo(o.getNombre());
		}
		return res;
	}
}
```

Ejercicio 3

```
private static final String DELIMITADOR_PRINCIPAL = ";";
private static final String DELIMITADOR_SECUNDARIO = ",";

public static Jugador parseaJugador(String s) {

	Checkers.checkNoNull(s);
	String[] splits = s.split(DELIMITADOR_PRINCIPAL);
	String msg = String.format("Formato no valido <%s>", s);
	Checkers.check(msg, splits.length == 9);

	String nombre = splits[0].trim();
	LocalDate fechaNacimiento = parseaFecha(splits[1].trim());
	Posicion posicion = Posicion.valueOf(splits[2].trim());
	Integer goles = Integer.valueOf(splits[3].trim());
	Integer asistencias = Integer.valueOf(splits[4].trim());
	List<String> lesiones = parseaLesiones(splits[5].trim());
	Equipo equipo = new Equipo(splits[6].trim(),
		Integer.valueOf(splits[7].trim()), splits[8].trim());

	return new Jugador(nombre, fechaNacimiento, posicion, goles, asistencias,
		equipo, lesiones);
}

private static LocalDate parseaFecha(String fecha) {
	return LocalDate.parse(fecha, DateTimeFormatter.ofPattern("yyyy-MM-dd"));
}

private static List<String> parseaLesiones(String cad) {
	Checkers.checkNoNull(cad);
	String limpia = cad.replace("[", "").replace("]", "").trim();
	List<String> lg = new ArrayList<>();

	if (!limpia.isEmpty()) {
		String [] splits = limpia.split(DELIMITADOR_SECUNDARIO);
		for (String s: splits) {
			lg.add(s);
		}
	}
	return lg;
}
```

Ejercicio 4

Apartado 4.1

```
public Long getNumeroLesionesDistintas(Integer umbralClasificacion) {
	return jugadores.stream()
		.filter(j -> j.getEquipo().clasificacion() < umbralClasificacion)
		.flatMap(j -> j.getLesiones().stream())
		.distinct()
		.count();
}
```

Apartado 4.2

```
public List<String> getNJugadoresMasJovenesPosicionSinLesion(Posicion p, Integer n) {
	return jugadores.stream()
		.filter(j -> j.getPosicion().equals(p) && j.getLesiones().isEmpty())
		.sorted(Comparator.comparing(Jugador::getEdad))
		.map(Jugador::getNombre)
		.limit(n)
		.collect(Collectors.toList());
}
```

Apartado 4.3

```
public Equipo getEquipoMasJugadoresConGolesSuperiorMedia() {
	Double mediaGoles = jugadores.stream()
		.mapToInt(Jugador::getGoles)
		.average()
		.getAsDouble();

	Map<Equipo, Long> m = jugadores.stream()
		.filter(j -> j.getGoles()> mediaGoles)
		.collect(Collectors.groupingBy(
			Jugador::getEquipo,
			Collectors.counting()));

	return m.entrySet().stream()
		.max(Map.Entry.comparingByValue())
		.get().getKey();
}
```

Apartado 4.4

```
public Boolean todosEquiposTienenJugadorLesionado() {
	Map<Equipo, Boolean> aux = jugadores.stream().
		collect(Collectors.groupingBy(
			Jugador::getEquipo,
			Collectors.collectingAndThen(Collectors.toList(),
				l -> hayJugadorLesionado(l))));

	return aux.entrySet().stream()
		.allMatch(e -> e.getValue().equals(true));
}

private Boolean hayJugadorLesionado(List<Jugador> jugadores) {
	return jugadores.stream()
		.anyMatch(j -> !j.getLesiones().isEmpty());
}
```

Apartado 4.5

```
public SortedMap<Equipo, String> getNombreJugadorConMayorSumaGAPorEquipo() {

	Map<Equipo, Jugador> aux = new HashMap<>();
	for (Jugador j: jugadores) {
		Equipo key = j.getEquipo();

		if (aux.containsKey(key)) {
			Jugador maxJugador = aux.get(key);
			Integer maxVal = maxJugador.getGoles() + maxJugador.getAsistencias();

			Integer currVal = j.getGoles() + j.getAsistencias();
			if(currVal > maxVal) {
				aux.put(key, j);
			}
		}
		else {
			aux.put(key, j);
		}
	}

	Comparator<Equipo> cmp = Comparator.comparing(Equipo::clasificacion);
	SortedMap<Equipo, String> res = new TreeMap<>(cmp.reversed());

	for(Entry<Equipo,Jugador> e: aux.entrySet()) {
		res.put(e.getKey(), e.getValue().getNombre());
	}

	return res;

}
```

Test

```
public static void main(String[] args) {
	EstadisticasJugadores jugadores =
		FactoriaJugadores.leeJugadores("data/jugadores.csv");

	System.out.println("EJERCICIO 4.1==========================================");
	testGetNumeroLesionesDistintas(jugadores, 6);
	testGetNumeroLesionesDistintas(jugadores, 2);

	System.out.println("\nEJERCICIO 4.2==========================================");
	testGetNJugadoresMasJovenesPosicionSinLesion(jugadores, Posicion.DELANTERO, 3);
	testGetNJugadoresMasJovenesPosicionSinLesion(jugadores, Posicion.CENTROCAMPISTA,
		3);

	System.out.println("\nEJERCICIO 4.3==========================================");
	testGetEquipoMasJugadoresConGolesSuperiorMedia(jugadores);

	System.out.println("\nEJERCICIO 4.4==========================================");
	testTodosEquiposTienenJugadorLesionado(jugadores);

	System.out.println("\nEJERCICIO 4.5==========================================");
	testGetNombreJugadorConMayorRatioPorEquipo(jugadores);
}
```

```
private static<T> void mostrarColeccion(Collection<T> col) {
	col.stream()
	.map(e -> "\t" + e)
	.forEach(System.out::println);
}

private static void testGetNumeroLesionesDistintas(EstadisticasJugadores jugadores,
		Integer umbralClasificacion) {
	try {
		Long res = jugadores.getNumeroLesionesDistintas(umbralClasificacion);
		String msg = String.format("El número de lesiones distintas de los "
			+ umbralClasificacion + " primeros equipos es: ");
		System.out.println(msg);
		System.out.println(res);
	} catch (Exception e) {
		System.out.println("Capturada excepción inesperada");
		System.out.println(e.getMessage());
	}
}

private static void testGetNJugadoresMasJovenesPosicionSinLesion(EstadisticasJugadores
		jugadores, Posicion p, Integer n) {
	try {
		List<String> res =
			jugadores.getNJugadoresMasJovenesPosicionSinLesion(p,n);
		String msg = String.format(
			"Los %d jugadores más jovenes sin lesión de la posicion %s son: ",
			n, p);
		System.out.println(msg);
		System.out.println(res);
	} catch (Exception e) {
		System.out.println("Capturada excepción inesperada");
		System.out.println(e.getMessage());
	}
}

private static void testGetEquipoMasJugadoresConGolesSuperiorMedia(EstadisticasJugadores
		jugadores) {
	try {
		Equipo res = jugadores.getEquipoMasJugadoresConGolesSuperiorMedia();
		String msg = String.format("El equipo con mayor número de jugadores con
			goles por encima de la media es: ");
		System.out.println(msg);
		System.out.println(res);
	} catch (Exception e) {
		System.out.println("Capturada excepción inesperada");
		System.out.println(e.getMessage());
	}
}

private static void testTodosEquiposTienenJugadorLesionado(EstadisticasJugadores
		jugadores) {
	try {
		Boolean res = jugadores.todosEquiposTienenJugadorLesionado();
		String msg = String.format("¿Todos los equipos tienen al menos un jugador
			que ha sufrido una lesion?: ");
		System.out.println(msg);
		System.out.println(res);
```

```
		} catch (Exception e) {
			System.out.println("Capturada excepción inesperada");
			System.out.println(e.getMessage());
		}
	}

	private static void testGetNombreJugadorConMayorRatioPorEquipo(EstadisticasJugadores
			jugadores) {
		try {
			Map<Equipo, String> res =
				jugadores.getNombreJugadorConMayorSumaGAPorEquipo();
			String msg = String.format("Los nombres de jugadores con mayor ratio
				goles/asistencias por equipo son: ");
			System.out.println(msg);
			mostrarColeccion(res.entrySet());
		} catch (Exception e) {
			System.out.println("Capturada excepción inesperada");
			System.out.println(e.getMessage());
		}
	}
```