FUNDAMENTOS DE PROGRAMACIÓN. Curso 2022/23

SEGUNDO EXAMEN PARCIAL. 30 de mayo de 2023. Soluciones

Ejercicio 1

```
public record Resultado(Integer minuto, Integer golesLocal, Integer golesVisitante) {
	public Resultado {
		Checkers.check("Minuto incorrecto", minuto >= 0);
		Checkers.check("Goles local incorrectos", golesLocal >= 0);
		Checkers.check("Goles visitante incorrectos", golesVisitante >= 0);
	}
}
```

Ejercicio 2

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

	public Partido(LocalDate fecha, String equipoLocal, String equipoVisitante,
			Long espectadores, Competicion competicion,
			List<Resultado> resultadosParciales) {
		this.fecha = fecha;
		this.equipoLocal = equipoLocal;
		this.equipoVisitante = equipoVisitante;
		this.espectadores = espectadores;
		this.competicion = competicion;
		Checkers.check("Resultados parciales incorrectos",
                                                estaOrdenado(resultadosParciales));
		this.resultadosParciales = resultadosParciales;
	}

	private Boolean estaOrdenado(List<Resultado> resultadosParciales) {
		Boolean res = true;
		int indAnt = 0;
		for (int i=1; i < resultadosParciales.size(); i++) {
			Integer minutoAnt = resultadosParciales.get(indAnt).minuto();
			Integer minutoAct = resultadosParciales.get(i).minuto();
			if (! (minutoAnt< minutoAct)) {
				res = false;
				break;
			}
			indAnt = i;
		}
		return res;
	}

	public Integer getGolesLocal() {
		return resultadosParciales.get(resultadosParciales.size() - 1)
			.golesLocal();
	}

	public Integer getGolesVisisitante() {
		return resultadosParciales.get(resultadosParciales.size() - 1)
			.golesVisitante();
	}
```

```
	public List<Integer> getMinutos() {
		return resultadosParciales.stream()
			.map(Resultado::minuto)
			.filter(x -> !x.equals(Integer.valueOf(0)))
			.toList();
	}

	public int compareTo(Partido p) {
		int res = this.getFecha().compareTo(p.getFecha());
		if (res == 0) {
			res = this.getEquipoLocal().compareTo(p.getEquipoLocal());
			if (res == 0) {
				res = this.getEquipoVisitante()
					.compareTo(p.getEquipoVisitante());
			}
		}
		return res;
	}

	// Por claridad de la solución, se omiten en este documento los getters
	// de las propiedades básicas, equals, hashCode y toString
}
```

Ejercicio 3: Factoría (1 pto)

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

public static Partido parsearPartido(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 == 6);
	LocalDate fecha= parseaFecha(splits[0].trim());
	String equipoLocal = splits[1].trim();
	String equipoVisitante = splits[2].trim();
	Long espectadores = Long.valueOf(splits[3].trim());
	Competicion competicion = Competicion.valueOf(splits[4].trim());
	List<Resultado> goles = parseaResultadosParciales(splits[5].trim());
	return new Partido(fecha, equipoLocal, equipoVisitante, espectadores,
		competicion, goles);
}

private static LocalDate parseaFecha(String fecha) {
	return LocalDate.parse(fecha,DateTimeFormatter.ofPattern("d/M/y"));
}

private static List<Resultado> parseaResultadosParciales(String cad) {
	Checkers.checkNoNull(cad);
	String limpia = cad.replace("[", "").replace("]", "").trim();
	String [] splits = limpia.split(DELIMITADOR_SECUNDARIO);
	List<Resultado> lg = new ArrayList<>();
	for (String s: splits) {
		lg.add(parseaResultado(s));
	}
```

```
	return lg;
}

private static Resultado parseaResultado(String cad) {
	 Checkers.checkNoNull(cad);
	 String [] splits = cad.split(DELIMITADOR_RESULTADO);
	 String msg = String.format("Formato resultado no valido <%s>", cad);
	 Checkers.check(msg, splits.length == 3);

	 Integer minuto = Integer.valueOf(splits[0].trim());
	 Integer golesLocal = Integer.valueOf(splits[1].trim());
	 Integer golesVisitantes = Integer.valueOf(splits[2].trim());
	 return new Resultado(minuto, golesLocal, golesVisitantes);
}
```

Ejercicio 4.1

```
public Integer getNumGolesEquipos(Set<String> equipos) {
	Integer numg = 0;
	for (Partido p: partidos) {
		if (equipos.contains(p.getEquipoLocal())) {
			numg += p.getGolesLocal();
		} else if (equipos.contains(p.getEquipoVisitante())) {
			numg += p.getGolesVisisitante();
		}
	}
	return numg;
}
```

Ejercicio 4.2

```
public Integer getNumGolesDespuesMinuto(Integer minutoUmbral) {
	Long res = partidos.stream()
		.flatMap(x -> x.getMinutos().stream())
		.filter(x -> x.compareTo(minutoUmbral) > 0)
		.count();
	return res.intValue();
}
```

Ejercicio 4.3

```
public SortedSet<Partido> getNPartidosMasEspectadores(Integer n) {
	Function<Partido,Integer>fGoles = partido -> partido.getGolesLocal()
		+ partido.getGolesVisisitante();
	Comparator<Partido> cmpGoles = Comparator.comparing(fGoles)
		.thenComparing(Comparator.naturalOrder());

	return partidos.stream()
		.sorted(Comparator.comparing(Partido::getEspectadores).reversed())
		.limit(n)
		.collect(Collectors.toCollection(() -> new TreeSet<>(cmpGoles)));
}
```

Ejercicio 4.4

```
public Competicion getCompeticionMasEspectadores(Integer mes) {
	Map<Competicion, Long> espXCompeticion = partidos.stream()
		.filter(x -> mes.equals(x.getFecha().getMonthValue()))
		.collect(Collectors.groupingBy(
			Partido::getCompeticion,
			Collectors.summingLong(Partido::getEspectadores)));
	return espXCompeticion.entrySet().stream()
		.max(Map.Entry.comparingByValue())
		.map(Map.Entry::getKey)
		.get();
}
```

Ejercicio 4.5

```
public Map<Competicion, Partido> getPartidoGolMasTardioPorCompetition() {
	return partidos.stream()
		.collect(Collectors.groupingBy(
			Partido::getCompeticion,
			Collectors.collectingAndThen(
				Collectors.maxBy(
					Comparator.comparing(Partido::getMinutoUltimoGol)),
					opt -> opt.get())
		));
}

// Solución alternativa

public Map<Competicion, Partido> getPartidoGolMasTardioPorCompeticion() {
	return partidos.stream()
		.collect(Collectors.toMap(
			Partido::getCompeticion,
			p -> p,
			BinaryOperator.maxBy(
				Comparator.comparing(Partido::getUltimoMinuto))));
}

public Integer getMinutoUltimoGol() {
	return resultadosParciales.get(resultadosParciales.size() - 1).minuto();
}

// Solución alternativa

public Map<Competicion, Partido> getPartidoGolMasTardioPorCompeticion() {
	return partidos.stream()
		.collect(Collectors.groupingBy(
			Partido::getCompeticion,
			Collectors.collectingAndThen(
				Collectors.toList(),
				lv -> getPartidoGolMasTardio(lv)
			)
		));
}
```

```
private static Partido getPartidoGolMasTardio(List<Partido> partidos) {
	Comparator<Partido> cmp = Comparator.comparing(Partido::getMinutoUltimoGol);
	return partidos.stream()
		.max(cmp)
		.get();
}

// Solución alternativa

public Map<Competicion, Partido> getPartidoGolMasTardioPorCompeticion() {
	Map<Competicion, List<Partido>> partidoXcompeticion = partidos.stream()
		.collect(Collectors.groupingBy(Partido::getCompeticion));

	return partidoXcompeticion.entrySet().stream()
		.collect(Collectors.toMap(
			x -> x.getKey(),
			x -> getPartidoGolMasTardio(x.getValue())));
}

private static Partido getPartidoGolMasTardio(List<Partido> partidos) {
	Comparator<Partido> cmp = Comparator.comparing(Partido::getMinutoUltimoGol);
	return partidos.stream()
		.max(cmp)
		.get();
}
```