FUNDAMENTOS DE PROGRAMACIÓN. Curso 2022/23

SEGUNDO EXAMEN PARCIAL. 31 de mayo de 2023. Soluciones

Ejercicio 1

```
public record Etapa(String parada, Integer minutos) {
	public Etapa {
		Checkers.check("Minuto incorrecto", minutos >= 0);
	}

}
```

Ejercicio 2

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

	private LocalDateTime fechaHora;
	private String nombre;
	private Empresa empresa;
	private Integer usuarios;
	private String paradaInicial;
	private List<Etapa> recorrido;

	public Trayecto(LocalDateTime fechaHora, String nombre, Empresa empresa,
			Integer usuarios, String paradaInicial, List<Etapa> recorrido) {
		this.fechaHora = fechaHora;
		this.nombre = nombre;
		this.empresa = empresa;
		this.usuarios = usuarios;
		Checkers.checkNoNull(paradaInicial);
		this.paradaInicial = paradaInicial;
		Checkers.check("Recorridos incorrectos", recorrido.size() >= 1);
		this.recorrido = new ArrayList<>(recorrido);
	}

	public Integer getNumeroParadas() {
		return recorrido.size() + 1;
	}

	public LocalTime getHoraLlegada() {
		return getHoraLlegadaParada(recorrido.size() - 1);
	}

	public LocalTime getHoraLlegadaParada(int ind) {
		Checkers.check("Índice fuera de rango", ind >= 0 && ind < recorrido.size());
		LocalDateTime res = getFechaHora();
		for (int i = 0; i <= ind; i++) {
			Etapa etapa = recorrido.get(i);
			res = res.plusMinutes(etapa.minutos());
		}
		return res.toLocalTime();
	}
```

```
	public List<String> getParadas() {
		List<String> res = new ArrayList<String>();
		res.add(paradaInicial);
		for (Etapa etapa: recorrido) {
			res.add(etapa.parada());
		}
		return res;
	}

	// Alternativa para getHoraLLegada si no se ha hecho getHoraLlegadaParada
	public Integer getTiempoRecorrido() {
		return recorrido.stream().mapToInt(x->x.minutos()).sum();
	}

	public LocalTime getHoraLlegada() {
		return fechaHora.toLocalTime().plusMinutes(getTiempoRecorrido());
	}

	public int compareTo(Trayecto t) {
		int res = getFechaHora().compareTo(t.getFechaHora());
		if (res == 0) {
			res= getNombre().compareTo(t.getNombre());
		}

		return res;
	}

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

Ejercicio 3

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

public static Trayecto parseaTrayecto(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);

	LocalDateTime fechaHora = parseaFechaHora(splits[0].trim());
	String linea = splits[1].trim();
	Empresa empresa = parseaEmpresa (splits[2].trim());
	Integer usuarios = Integer.valueOf(splits[3].trim());
	String paradaInicial = splits[4].trim();
	List<Etapa> recorrido = parseaRecorrido(splits[5].trim());
	return new Trayecto(fechaHora, linea, empresa, usuarios, paradaInicial, recorrido);
}

private static Empresa parseaEmpresa(String cad) {
	return Empresa.valueOf(cad);
}
```

```
private static LocalDateTime parseaFechaHora(String fechaHora) {
	return LocalDateTime.parse(fechaHora, DateTimeFormatter.ofPattern("d/M/y H:m"));
}

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

private static Etapa parseaEtapa(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 == 2);
	String parada = splits[1].trim();
	Integer minutos = Integer.valueOf(splits[0].trim());
	return new Etapa(parada, minutos);
}
```

Ejercicio 4.1

```
public List<String> getParadasEmpresas(Set<Empresa> empresas, Character c) {
	return trayectos.stream()
		.filter(x -> empresas.contains(x.getEmpresa()))
		.flatMap(x -> x.getParadas().stream())
		.filter(x -> c.equals(x.charAt(0)))
		.collect(Collectors.toList());
}
```

Ejercicio 4.2

```
public SortedSet<Trayecto> getTrayectosMasUsuarios(Integer n) {
	Comparator<Trayecto> cmp = Comparator.comparing(Trayecto::getFechaHora)
		.thenComparing(Comparator.naturalOrder());
	return trayectos.stream()
		.sorted(Comparator.comparing(Trayecto::getUsuarios).reversed())
		.limit(n)
		.collect(Collectors.toCollection(() -> new TreeSet<>(cmp)));
}
```

Ejercicio 4.3

```
public String getLineaMasUsuarios(Integer mes) {
	Map<String, Integer> m = trayectos.stream().
		filter(x -> mes.equals(x.getFechaHora().getMonthValue())).
		collect(Collectors.groupingBy(Trayecto::getNombre,
			Collectors.summingInt(Trayecto::getUsuarios)));

	return m.entrySet().stream()
		.max(Comparator.comparing(Map.Entry::getValue))
		.map(Map.Entry::getKey)
		.orElse(null);

}

// Alternativa

public String getLineaMasUsuarios(Integer mes) {
	Map<String, Integer> m = trayectos.stream().
		filter(x -> mes.equals(x.getFechaHora().getMonthValue())).
		collect(Collectors.groupingBy(Trayecto::getNombre,
			Collectors.summingInt(Trayecto::getUsuarios)));

	Map.Entry<String, Integer> maximo = m.entrySet().stream()
		.max(Comparator.comparing(Map.Entry::getValue))
		.orElse(null);

	String res = null;
	if (maximo != null){
		res = maximo.getKey();
	}

	return res;
}
```

Ejercicio 4.4

Método auxiliar de la clase Trayecto:

```
public Integer getTiempoRecorrido() {
	return recorrido.stream()
		.mapToInt(Etapa::minutos)
		.sum();
}

public Map<Empresa, Trayecto> getTrayectoMasTiempoRecorridoPorEmpresa() {
	Comparator<Trayecto> cmp = Comparator.comparing(Trayecto::getTiempoRecorrido);
	return trayectos.stream()
		.collect(Collectors.groupingBy(
			Trayecto::getEmpresa,
			Collectors.collectingAndThen(
				Collectors.maxBy(cmp),
				Optional::get)
		));
}
```

```
// Alternativa

public Map<Empresa, Trayecto> getTrayectoMasTiempoRecorridoPorEmpresa() {
	Comparator<Trayecto> cmp = Comparator.comparing(Trayecto::getTiempoRecorrido);
	return trayectos.stream()
		.collect(Collectors.toMap(
			Trayecto::getEmpresa,
			tray -> tray,
			BinaryOperator.maxBy(cmp)));
}

// Alternativa

public Map<Empresa, Trayecto> getTrayectoMasTiempoRecorridoPorEmpresa() {
	return trayectos.stream()
		.collect(Collectors.groupingBy(
			Trayecto::getEmpresa,
			Collectors.collectingAndThen(
				Collectors.toList(),
				ControlTrayectos::getTrayectoMayorTiempo
			)
		));
}

private static Trayecto getTrayectoMayorTiempo(List<Trayecto> lp) {
	Comparator<Trayecto> cmp =
		Comparator.comparing(Trayecto::getTiempoRecorrido);
	return lp.stream()
		.max(cmp)
		.get();
}

// Alternativa

public Map<Empresa, Trayecto> getTrayectoMasTiempoRecorridoPorEmpresa() {
	Map<Empresa, List<Trayecto>> m = trayectos.stream()
		.collect(Collectors.groupingBy(Trayecto::getEmpresa));
	return m.entrySet().stream()
		.collect(Collectors.toMap(
			Map.Entry::getKey,
			entry -> getTrayectoMayorTiempo(entry.getValue())));
	}
}
```

Ejercicio 4.5

```
public Integer getTotalUsuarios(String parada, LocalTime hora) {
	Integer numUsuarios = 0;
	for (Trayecto t: trayectos) {
		LocalTime horaLlegada = null;
		int ind = t.getParadas().indexOf(parada);

		if (ind == 0) {
			horaLlegada = t.getFechaHora().toLocalTime();
		} else if (ind > 0) {
			horaLlegada = t.getHoraLlegadaParada(ind - 1);
		}

		if (horaLlegada != null && hora.isBefore(horaLlegada)) {
			numUsuarios += t.getUsuarios();
		}
	}
	return numUsuarios;
 }
```