FUNDAMENTOS DE PROGRAMACIÓN. Curso 2023/24

SEGUNDO EXAMEN PARCIAL. 29 de mayo de 2024. Soluciones

Ejercicio 1

```
public record Aula(String nombre, Integer capacidad) {
	public Aula {
		Checkers.check("El nombre del aula debe comenzar por una letra",
				Character.isLetter(nombre.charAt(0)));
		Checkers.check("La capacidad del aula debe ser mayor que 0",capacidad > 0);
	}
}
```

Ejercicio 2

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

	private String asignatura;
	private Integer curso;
	private LocalDateTime fechaHora;
	private Duration duracion;
	private TipoExamen tipo;
	private Integer asistentes;
	private Boolean inscripcion;
	private List<Aula> aulas;

	public Examen(String asignatura, Integer curso, LocalDateTime fechaHora,
			Duration duracion, TipoExamen tipo, Integer asistentes,
			Boolean inscripcion, List<Aula> aulas) {

		Checkers.check("La duración mínima es de una hora",
				duracion.toHours() >= 1);
		Checkers.check("El número de asistentes debe ser positivo",
				asistentes > 0);

		this.asignatura = asignatura;
		this.curso = curso;
		this.fechaHora = fechaHora;
		this.duracion = duracion;
		this.tipo = tipo;
		this.asistentes = asistentes;
		this.inscripcion = inscripcion;
		this.aulas = aulas;
	}

	public String getAsignatura() {
		return asignatura;
	}

	public Integer getCurso() {
		return curso;
	}

	public LocalDateTime getFechaHora() {
		return fechaHora;
	}

	public Duration getDuracion() {
		return duracion;
	}
```

```
public void setDuracion(Duration duracion) {
	Checkers.check("La duración mínima es de una hora",
			duracion.toHours() >= 1);
	this.duracion = duracion;
}

public Integer getAsistentes() {
	return asistentes;
}

public void setAsistentes(Integer asistentes) {
	Checkers.check("El número de asistentes debe ser positivo",
			asistentes > 0);
	this.asistentes = asistentes;
}

public TipoExamen getTipo() {
	return tipo;
}

public Boolean getInscripcion() {
	return inscripcion;
}

public List<Aula> getAulas() {
	return new ArrayList<>(aulas);
}

public List<Integer> getPuestos() {
	return aulas.stream()
		.map(Aula::capacidad)
		.collect(Collectors.toList());
}

public Integer getCapacidadMaxima() {
	return aulas.stream()
		.mapToInt(Aula::capacidad)
		.sum();
}

public Double getPorcentajeAsistentes() {
	return asistentes * 100. / getCapacidadMaxima();
}

public String toString() {
	return "Examen [asignatura=" + asignatura + ", curso=" + curso
		+ ", fechaHora=" + fechaHora + ", duracion=" + duracion
		+ ", tipo=" + tipo + ", asistentes=" + asistentes
		+ ", inscripcion=" + inscripcion + ", aulas=" + aulas + "]";
}

public int hashCode() {
	return Objects.hash(asignatura, curso, fechaHora);
}
```

```
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Examen other = (Examen) obj;
		return Objects.equals(asignatura, other.asignatura)
				&& Objects.equals(curso, other.curso)
				&& Objects.equals(fechaHora, other.fechaHora);
	}

	public int compareTo(Examen e) {
		int res = fechaHora.compareTo(e.getFechaHora());
		if (res == 0) {
			res = curso.compareTo(e.getCurso());
			if (res == 0) {
				res = asignatura.compareTo(e.getAsignatura());
			}
		}
		return res;
	}

	public Boolean usaAula(String nombreAula) {
		return aulas.stream().anyMatch(a -> a.nombre().equals(nombreAula));
	}
}
```

Ejercicio 3

```
private static Examen parsearExamen(String lineaCSV) {
	String[] trozos = lineaCSV.split(",");
	Checkers.check("Formato de cadena incorrecto", trozos.length == 9);

	String asignatura = trozos[0].trim();
	Integer curso = Integer.valueOf(trozos[1].trim());
	LocalDateTime fechaHora = LocalDateTime.parse(trozos[2].trim() + "-" +
		trozos[3].trim(), DateTimeFormatter.ofPattern("d/M/y-H:m"));
	Duration duracion = Duration.ofMinutes(Integer.valueOf(trozos[4].trim()));
	TipoExamen tipo = TipoExamen.valueOf(trozos[5].trim().toUpperCase());
	Integer asistentes = Integer.valueOf(trozos[6].trim());
	Boolean inscripcion = parseaInscripcion(trozos[7].trim());
	List<Aula> aulas = parseaAulas(trozos[8].trim());

	return new Examen(asignatura, curso, fechaHora, duracion, tipo,
		asistentes, inscripcion, aulas);
}

private static Boolean parseaInscripcion(String cadena) {
	Boolean inscripcion = false;
	if (cadena.toUpperCase().equals("SI")) {
		inscripcion = true;
	}
	return inscripcion;
}
```

```
private static List<Aula> parseaAulas(String cadena) {
	String[] trozos = cadena.split(";");
	List<Aula> aulas = new ArrayList<>();

	for (String trozo: trozos) {
		aulas.add(parseaAula(trozo));
	}
	return aulas;
}

private static Aula parseaAula(String cadena) {
	String[] trozos = cadena.split("-");
	Checkers.check("Formato de cadena incorrecto", trozos.length == 2);
	String nombre = trozos[0].trim();
	Integer capacidad = Integer.valueOf(trozos[1].trim());
	return new Aula(nombre, capacidad);
}
```

Ejercicio 4

Ejercicio 4.1

```
public Map<String, Set<Examen>> getExamenesPorAula() {

	Map<String, Set<Examen>> res = new HashMap<>();

	for (Examen e: examenes) {
		for (Aula aula: e.getAulas()) {
			String key = aula.nombre();
			if (res.containsKey(key)) {
				res.get(key).add(e);
			} else {
				Set<Examen> conjunto = new HashSet<>();
				conjunto.add(e);
				res.put(key, conjunto);
			}
		}
	}
	return res;
}
```

Ejercicio 4.2

```
public Examen getExamenMayorPorcentajeAsistentes(LocalTime t, String nombreAula) {

	return examenes.stream()
		.filter(e -> e.getFechaHora().toLocalTime().isAfter(t) &&
			e.usaAula(nombreAula))
		.max(Comparator.comparing(Examen::getPorcentajeAsistentes))
		.orElse(null);
}
```

FUNDAMENTOS DE PROGRAMACIÓN. Curso 2023/24

SEGUNDO EXAMEN PARCIAL. 29 de mayo de 2024. Soluciones

Ejercicio 4.3

```
public SortedSet<String> getAulasExamenesTipo(TipoExamen tipo) {
	return examenes.stream()
		.filter(e -> e.getTipo().equals(tipo))
		.flatMap(e -> e.getAulas().stream())
		.map(Aula::nombre)
		.distinct()
		.collect(Collectors.toCollection(TreeSet::new));
}
```

Ejercicio 4.4

```
	public String getAulaMasOcupada(LocalDate fecha) {

		Map<String, Set<Examen>> aux = getExamenesPorAula();

		Map<String, Long> aux2 = aux.keySet().stream()
			.collect(Collectors.toMap(
				nombre -> nombre,
				nombre -> sumaDuracionesFecha(fecha, aux.get(nombre)))
			);

		return aux2.keySet().stream()
			.max(Comparator.comparing(nombre -> aux2.get(nombre)))
			.get();
	}

	private Long sumaDuracionesFecha(LocalDate fecha, Set<Examen> l) {
		return l.stream()
			.filter(e -> e.getFechaHora().toLocalDate().equals(fecha))
			.mapToLong(d -> d.getDuracion().toMinutes())
			.sum();
	}
```

Ejercicio 4.5

```
	public List<LocalDate> getFechasConMasAulasDe(Integer umbral) {
		Map<LocalDate, Integer> m = examenes.stream()
			.collect(Collectors.groupingBy(e -> e.getFechaHora().toLocalDate(),
				Collectors.summingInt(e -> e.getAulas().size())));

		return m.keySet().stream()
			.sorted()
			.filter(fecha -> m.get(fecha) > umbral)
			.toList();
	}
```

Test

```
public class TestCalendarioExamenes {

	public static void main(String[] args) {
		CalendarioExamenes c =
				FactoriaExamenes.leerCalendarioExamenes("data/examenes.csv");
```

```
		System.out.println("\nEJERCICIO 4.1==============================");
		testGetExamenesPorAula(c);
		System.out.println("\nEJERCICIO 4.2==============================");
		testGetExamenMayorPorcentajeAsistentes(c, LocalTime.of(8, 30), "F1.30");
		testGetExamenMayorPorcentajeAsistentes(c, LocalTime.of(15, 30), "I2.31");
		System.out.println("\nEJERCICIO 4.3==============================");
		testGetAulasExamenesTipo(c, TipoExamen.PRACTICO);
		testGetAulasExamenesTipo(c, TipoExamen.TEORICO);
		System.out.println("\nEJERCICIO 4.4==============================");
		testGetAulaMayorOcupacion(c, LocalDate.of(2024, 5, 25));
		testGetAulaMayorOcupacion(c, LocalDate.of(2024, 6, 7));
		System.out.println("\nEJERCICIO 4.5==============================");
		testGetFechasConMasAulasDe(c, 5);
		testGetFechasConMasAulasDe(c, 8);
	}

	public static void testGetExamenesPorAula(CalendarioExamenes c) {
		Map<String, Set<Examen>> res = c.getExamenesPorAula();
		System.out.println("Exámenes por aula (solo se muestran asignaturas): ");
		res.keySet().stream()
			.forEach(e -> System.out.println("Aula " + e + ": " +
				res.get(e).stream()
				.map(Examen::getAsignatura).collect(Collectors.toList())));
	}

	public static void testGetExamenMayorPorcentajeAsistentes(CalendarioExamenes c,
			LocalTime hora, String nombreAula) {
		Examen res = c.getExamenMayorPorcentajeAsistentes(hora, nombreAula);
		System.out.println("Examen con mayor % de asistentes realizado en el aula "
			+ nombreAula + " y con hora de comienzo posterior a las " + hora
			+ ": " + res);
	}

	public static void testGetAulasExamenesTipo(CalendarioExamenes c,
			TipoExamen tipo) {
		SortedSet<String> res = c.getAulasExamenesTipo(tipo);
		System.out.println("Aulas utilizadas en exámenes de tipo "
				+ tipo + ": " + res);
	}

	public static void testGetAulaMayorOcupacion(CalendarioExamenes c,
			LocalDate fecha) {
		String res = c.getAulaMasOcupada(fecha);
		System.out.println("Aula con mayor ocupación en la fecha "
				+ fecha + ": " + res);
		res = c.getAulaMasOcupada2(fecha);
		System.out.println("(Alternativa) Aula con mayor ocupación en la fecha "
				+ fecha + ": " + res);
	}

	public static void testGetFechasConMasAulasDe(CalendarioExamenes c,
			Integer umbral) {
		List<LocalDate> res = c.getFechasConMasAulasDe(umbral);
		System.out.println("Fechas con más de " + umbral + " aulas: " + res);
		res = c.getFechasConMasAulasDe2(umbral);
		System.out.println("(Alternativa) Fechas con más de " + umbral + " aulas: "
			+ res);
	}
}
```