FUNDAMENTOS DE PROGRAMACIÓN. Curso 2022/23

SEGUNDA CONVOCATORIA. 14 de julio de 2023. Segundo cuatrimestre. Soluciones

Ejercicio 1

```
public record Asignatura(String id, Integer creditos, Integer curso) {

	public Asignatura {
		Checkers.check("Los cursos deben estar en el rango [1..4]", curso >= 1
			&& curso <= 4);
	}
}
```

Ejercicio 2

```
public class Estudiante implements Comparable<Estudiante> {
	private String nombre;
	private LocalDate fechaNacimiento;
	private Boolean esRepetidor;
	private Double notaMedia;
	private Set<Asignatura> asignaturas;

	public Estudiante(String nombre, LocalDate fechaNacimiento, Boolean esRepetidor,
			Double notaMedia, Set<Asignatura> asignaturas) {
		Checkers.check("La nota media no es negativa ni supera el 10",
			notaMedia >= 0.0 && notaMedia <= 10.0);
		Checkers.check(
			"El número de asignaturas cursadas debe variar en el rango [0..10]",
			asignaturas.size() >= 0 && asignaturas.size() <= 10);

		this.nombre = nombre;
		this.fechaNacimiento = fechaNacimiento;
		this.esRepetidor = esRepetidor;
		this.notaMedia = notaMedia;
		this.asignaturas = new HashSet<>(asignaturas);
	}

	public String getNombre() {
		return nombre;
	}

	public LocalDate getFechaNacimiento() {
		return fechaNacimiento;
	}

	public Boolean esRepetidor() {
		return esRepetidor;
	}

	public Double getNotaMedia() {
		return notaMedia;
	}

	public Integer getEdad() {
		LocalDate hoy = LocalDate.now();
		return this.fechaNacimiento.until(hoy).getYears();
	}
```

```
	public Set<Asignatura> getAsignaturas() {
		return new HashSet<>(asignaturas);
	}

	// Se omiten los métodos equals, hashCode y toString

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

Ejercicio 3

```
public class FactoriaGrupo {

	private static final Charset ENCODING = StandardCharsets.ISO_8859_1;
	private static final String SEPARADOR_PRINCIPAL=";";
	private static final String SEPARADOR_SECUNDARIO=",";

	// Se omite el método leeEstudiante, que se proporciona en el examen

	private static Estudiante parsearEstudiante(String linea) {
		String[]campos = linea.split(SEPARADOR_PRINCIPAL);
		Checkers.check("Formato no válido", campos.length == 5);

		String nombre = campos[0].trim();
		LocalDate fechaNacimiento = parsearFecha(campos[1].trim());
		Boolean esRepetidor = parsearLogico(campos[2].trim());
		Double notaMedia = Double.valueOf(campos[3].trim());
		Set<Asignatura>asignaturas = parsearAsignaturas(campos[4].trim());
		return new Estudiante(nombre, fechaNacimiento,
			esRepetidor, notaMedia, asignaturas);
	}

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

	private static Boolean parsearLogico(String textoLogico) {
		Boolean res = null;
		String t = textoLogico.toLowerCase();
		if (t.equals("si")) {
			res = true;
		} else if (t.equals("no")) {
			res = false;
		}
		return res;
	}
```

```
	private static Set<Asignatura> parsearAsignaturas(String textoAsignaturas) {
		String texto = textoAsignaturas.replace("[", "").replace("]", "");
		Set<Asignatura> r = new HashSet<>();
		if (!texto.equals("")) {
			String[] campos = texto.split(SEPARADOR_SECUNDARIO);
			for (String campo: campos) {
				r.add(parsearAsignatura(campo));
			}
		}
		return r;
	}

	private static Asignatura parsearAsignatura(String textoAsignatura) {
		String[] datos = textoAsignatura.split("-");
		Checkers.check("Formato asignatura no válido", datos.length == 3);
		String id = datos[0].trim();
		Integer creditos = Integer.valueOf(datos[1].trim());
		Integer curso = Integer.valueOf(datos[2].trim());
		return new Asignatura(id, creditos, curso);
	}
}
```

Ejercicio 4.1

```
public SortedSet<Estudiante> estudiantesQueCursanNAsignaturasAlMenos(Integer curso,
		Integer numAsignaturas) {
	return estudiantes.stream()
		.filter(e -> e.getAsignaturas().size() >= numAsignaturas &&
			tieneAsignaturaCurso(e, curso))
		.collect(Collectors.toCollection(TreeSet::new));
}

private Boolean tieneAsignaturaCurso(Estudiante e, Integer curso) {
	return e.getAsignaturas().stream()
		.anyMatch(asig -> asig.curso().equals(curso));
}
```

Ejercicio 4.2

```
public Double edadMediaEstudiantesRepetidores(Set<String> idsAsignaturas) {
	Double totalEdad = 0.0;
	Integer numEstudiantes = 0;

	for (Estudiante e: estudiantes) {
		if (e.esRepetidor() && cursaAsignaturas(e, idsAsignaturas)) {
			totalEdad += e.getEdad();
			numEstudiantes += 1;
		}
	}

	Double media = null;
	if (numEstudiantes > 0) {
		media = totalEdad / numEstudiantes;
	}
	return media;
}
```

```
private Boolean cursaAsignaturas(Estudiante e, Set<String> idsAsignaturas) {
	Set<String> asignaturasCursadas = new HashSet<>();
	for (Asignatura a: e.getAsignaturas()) {
		asignaturasCursadas.add(a.id());
	}
	return asignaturasCursadas.containsAll(idsAsignaturas);
}
```

Ejercicio 4.3

```
public Map<String, Map<Integer, List<Asignatura>>>
		asociarEstudiantesConAsignaturasPorCurso() {
	return estudiantes.stream()
		.collect(Collectors.toMap(Estudiante::getNombre,
			e -> agruparAsignaturasPorCurso(e)));
}

private Map<Integer, List<Asignatura>> agruparAsignaturasPorCurso(Estudiante e) {
	return e.getAsignaturas().stream()
		.collect(Collectors.groupingBy(Asignatura::curso));
}
```

Ejercicio 4.4

```
public String estudianteRepetidorConMasCreditos() {
	Comparator<Estudiante> c = Comparator.comparing(e -> totalCreditosCursados(e));
	return estudiantes.stream()
		.filter(Estudiante::esRepetidor)
		.max(c)
		.map(Estudiante::getNombre)
		.orElse(null);
}

private Integer totalCreditosCursados(Estudiante e) {
	return e.getAsignaturas().stream()
		.mapToInt(Asignatura::creditos)
		.sum();
}
```

Ejercicio 4.5

```
public Map<Integer, List<String>> asignaturasPorCursoOrdenadas() {
	return estudiantes.stream()
		.flatMap(estudiante -> estudiante.getAsignaturas().stream())
		.collect(Collectors.groupingBy(
			Asignatura::curso,
			Collectors.collectingAndThen(
				Collectors.toSet(),
				lista -> ordenarAsignaturas(lista))
		));
}
```

```
private List<String> ordenarAsignaturas(Collection<Asignatura> asignaturas) {
	Comparator<Asignatura> c = Comparator.comparing(Asignatura::creditos)
		.thenComparing(Asignatura::id);
	return asignaturas.stream()
		.sorted(c)
		.map(Asignatura::id)
		.collect(Collectors.toList());
}
```

Ejercicio 5

```
public class TestGrupo {

	public static void main(String[] args) {
		Grupo clase = FactoriaGrupo.leerEstudiantes("data/estudiantes.csv");
		System.out.println("TEST DE LECTURA:");
		System.out.println(clase);
		testEstudiantesQueCursanNAsignaturasAlMenos (clase, 3, 5);
		testEdadMediaEstudiantesRepetidores(clase, Set.of("FP", "ADDA"));
		testAsociarEstudiantesConAsignaturasPorCurso(clase);
		testEstudianteRepetidorConMasCreditos(clase);
		testAsignaturasPorCursoOrdenadas(clase);
	}

	private static void testAsignaturasPorCursoOrdenadas(Grupo clase) {
		try {
			String msg = String.format("EJ. 4.5: Asignaturas por curso ordenadas
			por créditos y por id.");
			System.out.println(msg);
			Map<Integer, List<String>> res =
				clase.asignaturasPorCursoOrdenadas();
			res.entrySet().stream()
				.forEach(e -> System.out.println("\t" + e.getKey() + "->"
					+ e.getValue()));
		} catch(Exception e) {
			e.printStackTrace();
			System.out.println("Excepción capturada " + e.getMessage());
		}
	}

	private static void testEstudianteRepetidorConMasCreditos(Grupo clase) {
		try {
			String msg = String.format("EJ. 4.4: Nombre del estudiante repetidor
			con mayor número de créditos cursados:");
			System.out.println(msg);
			String res = clase.estudianteRepetidorConMasCreditos();
			System.out.println("\t" + res);
		} catch(Exception e) {
			e.printStackTrace();
			System.out.println("Excepción capturada " + e.getMessage());
		}
	}
```

```
	private static void testAsociarEstudiantesConAsignaturasPorCurso(Grupo clase) {
		try {
			String msg = String.format("EJ. 4.3: Asociar el nombre de cada
			estudiante con las asignaturas que cursa agrupadas por curso.");
			System.out.println(msg);
			Map<String, Map<Integer, List<Asignatura>>> res =
				clase.asociarEstudiantesConAsignaturasPorCurso();
			res.entrySet().stream()
				.forEach(e -> System.out.println("\t" + e.getKey() + "->"
					+ e.getValue()));
		} catch(Exception e) {
			e.printStackTrace();
			System.out.println("Excepción capturada " + e.getMessage());
		}
	}

	private static void testEdadMediaEstudiantesRepetidores(Grupo clase,
			Set<String> idsAsignaturas) {
		try {
			String msg = String.format("EJ. 4.2: Edad media de los estudiantes
			repetidores que cursan %s:", idsAsignaturas);
			System.out.println(msg);
			Double res = clase.edadMediaEstudiantesRepetidores(idsAsignaturas);
			System.out.println("\t" + res);
		} catch(Exception e) {
			e.printStackTrace();
			System.out.println("Excepción capturada " + e.getMessage());
		}
	}

	private static void testEstudiantesQueCursanNAsignaturasAlMenos(Grupo clase,
			Integer curso, Integer numAsignaturas) {
		try {
			String msg = String.format("EJ.4.1: Estudiantes que cursan al menos
			%d asignaturas con alguna de estas pertenecientes al curso %d:",
			numAsignaturas, curso);
			System.out.println(msg);
			Set<Estudiante> res =
				clase.estudiantesQueCursanNAsignaturasAlMenos(curso,
					numAsignaturas);
			res.stream()
				.forEach(e -> System.out.println("\t" + e));
		} catch(Exception e) {
			e.printStackTrace();
			System.out.println("Excepción capturada " + e.getMessage());
		}
	}
}
```