FUNDAMENTOS DE PROGRAMACIÓN. Curso 2018/19

SEGUNDO EXAMEN PRÁCTICO. 10 de junio de 2019. Sesión 1. Soluciones

 # EJERCICIO 1 [1 punto]

Comprobar restricciones en el constructor (0,5 puntos)

```
	public OfertaEmpleo(String especialidad, Integer numPlazas,
			Integer numPlazasDiscapacidad, LocalDate fechaPublicacionBOJA,
			TipoAcceso tipoAcceso) {

		Checkers.check(
			"Fecha de publicación incorrecta:" + fechaPublicacionBOJA,
			fechaPublicacionBOJA.getYear() >= 1990 &&
			fechaPublicacionBOJA.compareTo(LocalDate.now()) <= 0);
		Checkers.check(
			"El número de plazas no puede ser negativo", numPlazas >= 0);
		Checkers.check(
			"El número de plazas para discapacitados no puede ser negativo",
			numPlazasDiscapacidad >= 0);

		this.especialidad = especialidad;
		this.numPlazas = numPlazas;
		this.numPlazasDiscapacidad = numPlazasDiscapacidad;
		this.fechaPublicacionBOJA = fechaPublicacionBOJA;
		this.tipoAcceso = tipoAcceso;
	}
```

Propiedades derivadas (0,25 puntos)

```
	public Integer getNumPlazasTotales() {
		return getNumPlazas() + getNumPlazasDiscapacidad();
	}

	public Double getPorcentajePlazasDiscapacidad() {
		return getNumPlazasDiscapacidad()*100./getNumPlazasTotales();
	}
```

Método compareTo (0,25 puntos)

```
	public int compareTo(OfertaEmpleo o) {
		int res = getFechaPublicacionBOJA().compareTo(o.getFechaPublicacionBOJA());
		if (res == 0) {
			res = getEspecialidad().compareTo(o.getEspecialidad());
			if (res == 0) {
				res = getTipoAcceso().compareTo(o.getTipoAcceso());
				if (res == 0) {
					res =
					  getNumPlazasTotales().compareTo(o.getNumPlazasTotales());
				}
			}
		}
		return res;
	}
```

 # EJERCICIO 2 [8 puntos]

Tipo Contenedor [0,5 puntos]

```
public class OfertasEmpleo {

	private List<OfertaEmpleo> ofertasEmpleo;

	public OfertasEmpleo() {
		this.ofertasEmpleo = new ArrayList<OfertaEmpleo>();
	}

	public OfertasEmpleo (Stream<OfertaEmpleo> s) {
		this.ofertasEmpleo = s.collect(Collectors.toList());
	}

	public List<OfertaEmpleo> getOfertasEmpleo() {
		return new ArrayList<>(ofertasEmpleo);
	}

	public Integer getNumOfertasEmpleo() {
		return ofertasEmpleo.size();
	}
```

Apartado a) [1 punto]

```
public Boolean hayAlgunaOfertaConPorcentajePlazasDiscapacidadMayorA(Double umbral) {
	Checkers.check("El umbral debe estar entre 0 y 100", umbral >= 0 && umbral <= 100);
	Predicate<OfertaEmpleo> pred =
		oferta -> oferta.getPorcentajePlazasDiscapacidad() >= umbral;
	return ofertasEmpleo.stream()
		.anyMatch(pred);
}
```

Apartado b) [1 punto]

```
public Integer getNumeroEspecialidadesFacultativosEspecialistas() {
	Long res = ofertasEmpleo.stream()
		.filter(oferta -> oferta.getEspecialidad().startsWith("FEA"))
		.map(OfertaEmpleo::getEspecialidad)
		.distinct()
		.count();
	return res.intValue();
}
```

Apartado c) [1 punto]

```
public Integer getTotalPlazas(String especialidad, Integer anyo) {
	return ofertasEmpleo.stream()
		.filter(oferta->oferta.getEspecialidad().equals(especialidad) &&
			oferta.getFechaPublicacionBOJA().getYear() == anyo)
		.mapToInt(OfertaEmpleo::getNumPlazasTotales)
		.sum();
}
```

```
//solución alternativa
public Integer getTotalPlazas(String especialidad, Integer anyo) {
	return ofertasEmpleo.stream()
		.filter(oferta -> oferta.getEspecialidad().equals(especialidad) &&
			oferta.getFechaPublicacionBOJA().getYear() == anyo)
		.map(OfertaEmpleo::getNumPlazasTotales)
		.reduce(Integer::sum); //También se puede usar la lambda
}
```

Apartado d) [1 punto]

```
public SortedMap<Integer, Integer> getTotalPlazasTipo(TipoAcceso tipo) {
	Predicate<OfertaEmpleo> pred =  oferta -> oferta.getTipoAcceso().equals(tipo);
	return ofertasEmpleo.stream()
		.filter(pred)
		.collect(Collectors.groupingBy(
			oferta -> oferta.getFechaPublicacionBOJA().getYear(),
			TreeMap::new,
			Collectors.summingInt(OfertaEmpleo::getNumPlazasTotales)));

}
```

Apartado e) [1,5 puntos]

```
public String getEspecialidadMasOfertada() {
	Map<String, Integer> m = getTotalPlazasPorEspecialidad();

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

//Método auxiliar, se puede poner también en el mismo método
private Map<String, Integer> getTotalPlazasPorEspecialidad() {
	return ofertasEmpleo.stream()
		.collect(Collectors.groupingBy(
			OfertaEmpleo::getEspecialidad,
			Collectors.summingInt(OfertaEmpleo::getNumPlazasTotales)));
}
```

Apartado f) [2 puntos]

```
//Método principal
public Map<Integer, List<String>> getNEspecialidadesMasPlazasPorAnyo(Integer n){
	Map<Integer, List<OfertaEmpleo>> mpaux= obtenerOfertasPorAnyo();

	return mpaux.entrySet().stream()
		.collect(Collectors.toMap(
			Map.Entry::getKey,
			entry -> obtenerNOfertasMayorNumPlazasTotales(entry.getValue(), n)));

}
```

```java
//Solución alternativa para el método principal con collectingAndThen
public Map<Integer, List<String>> getNEspecialidadesMasPlazasPorAnyo(Integer n) {
	return ofertasEmpleo.stream()
		.collect(Collectors.groupingBy(
			of->of.getFechaPublicacionBOJA().getYear(),
			Collectors.collectingAndThen(
				Collectors.toList(),
				l -> obtenerNOfertasMayorNumPlazasTotales(l, n)
		)));
}

//Método auxiliar 1
private Map<Integer, List<OfertaEmpleo>> obtenerOfertasPorAnyo() {
	return ofertasEmpleo.stream()
		.collect(Collectors.groupingBy(
			oferta -> oferta.getFechaPublicacionBOJA().getYear()
		));
}

//Método auxiliar 2
private List<String> obtenerNOfertasMayorNumPlazasTotales(List<OfertaEmpleo> lista,
		Integer n) {
	return lista.stream()
		.sorted(Comparator.comparing(OfertaEmpleo::getNumPlazasTotales).reversed())
		.limit(n)
		.map(OfertaEmpleo::getEspecialidad).
		.collect(Collectors.toList());
}
```

 # EJERCICIO 3 [1 punto]

```java
private static OfertaEmpleo parsearOferta(String lineaCSV) {

	String[] campos = lineaCSV.split(";");
	String msg = String.format(
		"Formato no válido. Se esperan 5 campos: %d - <%s>", campos.length, lineaCSV);

	Checkers.check(msg, campos.length == 5);
	String especialidad = campos[0].trim();
	Integer numPlazas = Integer.parseInt(campos[1].trim());
	Integer numPlazasDiscapacidad = Integer.parseInt(campos[2].trim());
	LocalDate fecha = LocalDate.parse(campos[3].trim(),
		DateTimeFormatter.ofPattern("d/M/yyyy"));
	TipoAcceso acceso = TipoAcceso.valueOf(campos[4].trim());

	return new OfertaEmpleo(especialidad, numPlazas, numPlazasDiscapacidad,
		fecha, acceso);
}
```