FUNDAMENTOS DE PROGRAMACIÓN. Curso 2018/19

SEGUNDO EXAMEN PRÁCTICO. 10 de junio de 2019. Sesión 2. Soluciones

 # EJERCICIO 1 [1 punto]

Comprobar restricciones en el constructor (0,5 puntos)

```
public OfertaEmpleo(String cuerpo, String especialidad, Integer numPlazas,
            LocalDate fechaPublicacionBOJA, Integer numBOJA) {
      Checkers.check("Fecha de publicación incorrecta:" + fechaPublicacionBOJA,
            fechaPublicacionBOJA.getYear() >= 1992 &&
            fechaPublicacionBOJA.compareTo(LocalDate.now()) <= 0);
      Checkers.check("El número de plazas no puede ser negativo", numPlazas >= 0);
      Checkers.check("El número de BOJA debe ser >=1", numBOJA >= 1);

      this.cuerpo = cuerpo;
      this.especialidad = especialidad;
      this.numPlazas = numPlazas;
      this.fechaPublicacionBOJA = fechaPublicacionBOJA;
      this.numBOJA = numBOJA;
}
```

Propiedades derivadas (0,25)

```
public NivelEducativo getNivelEducativo() {
      NivelEducativo res = NivelEducativo.OTROS;
      if (getCuerpo().equals("CUERPO DE MAESTROS")) {
            res = NivelEducativo.PRIMARIA;
      } else if (getCuerpo().equals("PROFESORES DE ENSEÑANZA SECUNDARIA")) {
            res = NivelEducativo.SECUNDARIA;
      } else if (getCuerpo().equals("PROFESORES TÉCNICOS DE FORMACIÓN PROFESIONAL")) {
            res = NivelEducativo.FP;
      }
      return res;
}
```

Método compareTo (0,25 puntos)

```
public int compareTo(OfertaEmpleo o) {
      int res = getFechaPublicacionBOJA().compareTo(o.getFechaPublicacionBOJA());

      if (res == 0) {
            res = getCuerpo().compareTo(o.getCuerpo());
            if (res == 0) {
                  res = getEspecialidad().compareTo(o.getEspecialidad());
                  if (res == 0) {
                        res = getNumPlazas().compareTo(o.getNumPlazas());
                  }
            }
      }
      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 SortedSet<String> getEspecialidadesDeCuerpo(String cuerpo) {
	return ofertasEmpleo.stream()
		.filter(oferta->oferta.getCuerpo().equals(cuerpo))
		.map(OfertaEmpleo::getEspecialidad)
		.collect(Collectors.toCollection(TreeSet::new));
}
```

Apartado b) [1 punto]

```
public Double getMediaPlazasPorOferta(String cuerpo, String especialidad ) {
	Predicate<OfertaEmpleo> pred =
		oferta->oferta.getCuerpo().equals(cuerpo) &&
			oferta.getEspecialidad().equals(especialidad);

	return ofertasEmpleo.stream()
		.filter(pred)
		.mapToInt(OfertaEmpleo::getNumPlazas)
		.average()
		.getAsDouble();
}
```

Apartado c) [1 punto]

```
public List<OfertaEmpleo> getNOfertasMasPlazas(Integer anyo, Integer n) {
	Comparator<OfertaEmpleo> c = Comparator.comparing(OfertaEmpleo::getNumPlazas)
		.reversed();

	return ofertasEmpleo.stream()
		.filter(oferta->oferta.getFechaPublicacionBOJA().getYear() == anyo)
		.sorted(c)
		.limit(n)
		.collect(Collectors.toList());
}
```

Apartado d) [1 punto]

```
public Map<NivelEducativo, Integer> getNumPlazasPorNivelEducativo(Integer anyo) {
	return ofertasEmpleo.stream()
		.filter(oferta -> oferta.getFechaPublicacionBOJA().getYear() == anyo)
		.collect(Collectors.groupingBy(
			OfertaEmpleo::getNivelEducativo,
			Collectors.summingInt(OfertaEmpleo::getNumPlazas)));
}
```

Apartado e) [1,5 puntos]

```
public Integer getAnyoMasOferta (String cuerpo) {
	Map<Integer, Integer> m = ofertasEmpleo.stream()
		.filter(oferta -> oferta.getCuerpo().equals(cuerpo))
		.collect(Collectors.groupingBy(
			oferta -> oferta.getFechaPublicacionBOJA().getYear(),
			Collectors.summingInt(OfertaEmpleo::getNumPlazas)
		));

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

Apartado f) [2 puntos]

```
public Map<NivelEducativo, Double> getPorcentajePlazasPorNivelEducativo(Integer anyo) {
	Map<NivelEducativo, Integer> m = getNumPlazasPorNivelEducativo(anyo);
	Integer totalPlazas = m.values().stream()
		.mapToInt(v -> v)
		.sum();

	return m.entrySet().stream()
		.collect(Collectors.toMap(Map.Entry::getKey,
			entry->entry.getValue() * 100.0 / totalPlazas));

}
```

 # EJERCICIO 3 [1 punto]

```
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 cuerpo = campos[0].trim();
	String especialidad = campos[1].trim();
	Integer numPlazas = Integer.parseInt(campos[2].trim());
	Integer numBOJA = Integer.parseInt(campos[3].trim());
	LocalDate fecha = LocalDate.parse(campos[4].trim(),
		DateTimeFormatter.ofPattern("d/M/yyyy"));

	return new OfertaEmpleo(cuerpo, especialidad, numPlazas, fecha, numBOJA);
}
```