1 **ordenadosPorNacionalidadesDistintas**: devuelve una colección de artículos ordenada por la cantidad de nacionalidades distintas que hay entre sus autores.

```
public List<Articulo> ordenadosPorNacionalidadesDistintas() //ArticulosRecibidos
{
        return listaArticulos.stream().
        sorted(comparing((Articulo a)->a.contarDistintasNacionalidades()).
        reversed()).collect(toList());
}

public int contarDistintasNacionalidades() //Articulo
{
        return (int) listaAutores.stream().
        map(Autor::getNacionalidad).
        distinct().
        count();
}

public String getNacionalida() //Autor
{
        return nacionalidad;
}
```

2 **ordenadosPorNacionalidadesDistintasYEdad**: devuelve una colección de artículos ordenada por la cantidad de nacionalidades distintas que hay entre sus autores y los artículos con la misma cantidad de nacionalidades ordenadas por la media de edad de los autores.

```
public List<Articulo> ordenadosPorNacionalidadesDistiYEdad() //ArticulosRecibidos
{
        return listaArticulos.stream().
        sorted(comparing(Articulo::contarDistintasNacionalidades).
        reversed().
        thenComparing(Articulo::mediaEdadAutores)).
        collect(toList());
}

public double mediaEdadAutores() //Articulo
{
        return listaAutores.stream().
        mapToInt(Autor::getEdad).
        average().
        getAsDouble();
}

public int getEdad() //Autor
{
        return edad;
}
```

3 **obtPorcentajeTemasExperto**: este método calcula el porcentaje de temas del artículo en el que el revisor automático es experto.

```
public float objPorcentajeTemasExperto(Articulo pArticulo) //RevisorAutomatico
{
        return (float)(listaTemas.stream().
        filter(s->pArticulo.incluyeTema(s)).
        count()*1.0) / pArticulo.numTemas();
}

public boolean incluyeTema(String pTema) //Articulo
{
        return listaTemas.stream().
        anyMatch(s->s.equals(pTema));
}
```

4 **obtPosiblesRevisores**: devuelve la lista de revisores automáticos que podrían revisar el artículo clasificados en orden decreciente por el porcentaje de temas del artículo en el que son expertos.

```
public List<RevisorAutomatico> obtPosibleRevisores() //Articulo
{
	RevisoresAutomaticos r = RevisoresAutomaticos.getRevisoresAutomaticos();
	return r.obtPosiblesRevisores(this);

}

public List<RevisorAutomatico> obtPosiblesRevisores(Articulo pArticulo)
{//RevisoresAutomaticos
	return listaRevisores.stream().
	filter(r->r.esAdecuado(pArticulo)).
	sorted(comparing((RevisorAutomatico r)->
	r.objPorcentajeTemasExperto(pArticulo)).reversed()).
	collect(toList());
}

public boolean esAdecuado(Articulo pArticulo) //RevisorAutomatico
{
	return listaTemas.stream().
	anyMatch(s->pArticulo.incluyeTema(s));
}
```

5 **getNumPublicacionesAutorSoloYVarios**: devuelve un mapa con la cantidad de artículos con un único autor (true) y la cantidad de artículos con más de un autor (false).

```
public Map<Boolean, Long> getNumPublicacionesAutorSoloYVarios()
{//ArticulosRecibidos
	return listaArticulos.stream().
	collect(partitioningBy(Articulo::masDeUnAutor,counting()));
}
public boolean masDeUnAutor() //Articulo
{
	return listaAutores.size()>1;
}
```