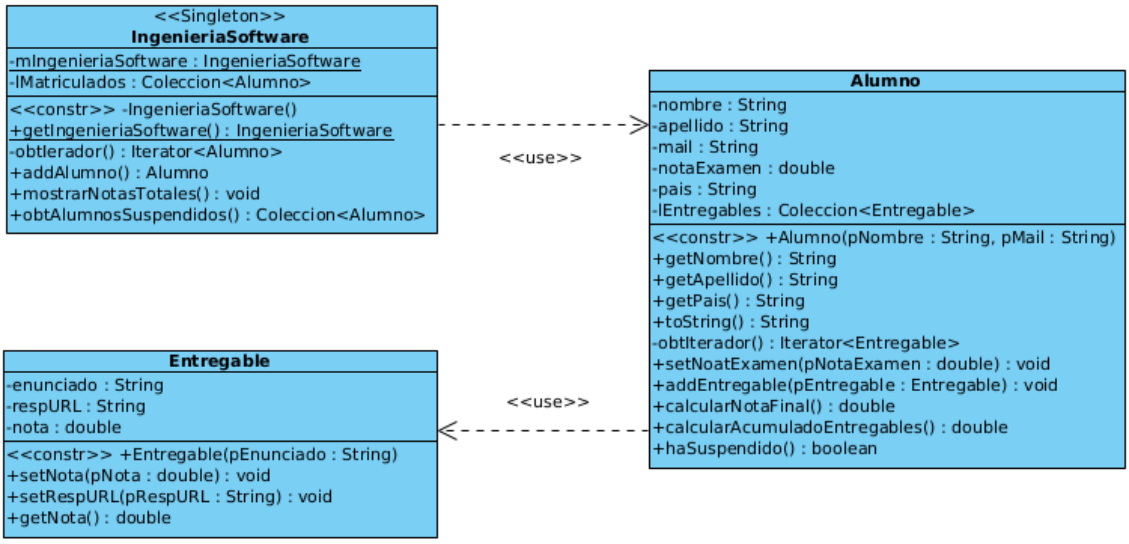
Java8

En este laboratorio se continuará con el ejercicio de repaso: *Ingeniería del Software*.



Implementa los métodos que se describen en los siguientes ejercicios utilizando las operaciones de agregación y las nuevas características incorporadas a partir de Java 8. Para ello, consultad la documentación de las interfaces Stream, IntStream, DoubleStream, Collectors, Comparator y Predicate, DoubleSummaryStatistics.

Ejercicio 1: Implementar el método `mostrarNotasTotalesJava8` que imprime las notas totales de los alumnos de la asignatura.

```
public void mostrarNotasTotalesJava8()
{
	lMatriculados.stream()
	.mapToDouble(Alumno::calcularNotaFinalJava8)
	.forEach(System.out::println);
}
```

Además, es recomendable implementar en la clase `Alumno` el método `calcularNotaFinalJava8` que se encargará de calcular la nota final del alumno.

```
public double calcularNotaFinalJava8()
{
	return calcularNotaEntregablesJava8()*0.6 + notaExamen*0.4;
}

public double calcularAcumuladoEntregablesJava8()
{
	return lEntregables.stream()
	.mapToDouble(Entregable::getNota)
	.average()
	.orElse(0.0);
}
```

Ayuda:

- *En el método `mostrarNotasTotalesJava8` primero se deben obtener las notas de los alumnos y para ello se debe usar la operación intermedia **mapToDouble**, a continuación se deben imprimir esos valores mediante la operación final **forEach**.*
- *En el método `calcularNotaEntregablesJava8` primero se deben obtener la notas de los entregables del alumno y para ello se debe usar la operación intermedia **mapToDouble,** a continuación se debe calcular la nota media de los entregables mediante la operación final **average**. Como average devuelve un Optional se debe indicar cual es el valor a devolver mediante **orElse**.*

Ejercicio 2: Implementar el método `obtenerAlumnosSuspendidosJava8` que devuelve la lista de los alumnos que han suspendido la asignatura (la nota final es inferior a 5.0).

```
public List<Alumno> obtenerAlumnosSuspendidosJava8()
{
	return lMatriculados.stream()
	.filter(p -> p.calcularNotaFinalJava8()<5.0)
	.collect(Collectors.toList());

	return lMatriculados.stream()
	.filter(Alumno::haSuspendido)
	.collect(Collectors.toList());
}
```

*Ayuda: primero se deben filtrar los alumnos que han suspendido mediante la operación intermedia **filter** y a continuación esos alumnos se deben guardar en una lista mediante la operación final **collect**.*

Ejercicio 3: Implementar el método `obtenerAlumnosAprobadosOrdenadosNombre` que devuelve la lista de alumnos que han aprobado la asignatura ordenados por el nombre.

```
public List<Alumno> obtenerAlumnosAprobadosOrdenadosNombre ()
{
	return lMatriculados.stream()
	.filter(p -> p.calcularNotaFinalJava8()>=5.0)
	.sorted(Comparator.comparing(Alumno::getNombre))
	.collect(Collectors.toList());
}
```

*Ayuda: primero se deben filtrar los alumnos que han suspendido mediante la operación intermedia **filter**, a continuación se deben ordenar los alumnos usando la operación intermedia **sorted** y finalmente se deben guardar los alumnos ordenados en una lista mediante la operación final **collect**.*
*Para el Comparator que se usará en la operación intermedia **sorted** se recomienda utilizar el método **comparing** de la interfaz Comparator.*

Ejercicio 4: Implementar el método `obtenerAlumnosAprobadosOrdenadosNombreApellido` que devuelve la lista de alumnos que han aprobado la asignatura ordenados por el nombre y los alumnos con el mismo nombre ordenados por su apellido.

```
public List<Alumno> obtenerAlumnosAprobadosOrdenadosNombreApellido()
{
	return lMatriculados.stream()
	.filter(p -> p.calcularNotaFinalJava8()>=5.0)
	.sorted(Comparator.comparing(Alumno::getNombre)
	.thenComparing(Alumno::getApellido))
	.collect(Collectors.toList());
```

```
}
```

__Ayuda:__ Es parecido al ejercicio anterior pero para el Comparator que se usará en la operación intermedia __sorted__ se recomienda utilizar los métodos __comparing__ y __thenComparing__ de la interfaz Comparator.

Ejercicio5: Implementar el método `obtenerPorcentajeAprobados` que devuelve el porcentaje de aprobados de la asignatura.

```
public double obtenerPorcentajeAprobados()
{
        double ikasleKop = (double) lMatriculados.size();
        return lMatriculados.stream()
        .filter(a->!a.haSuspendido())
        .count() / ikasleKop;
}
```

__Ayuda:__ se debe usar la operación terminal __count.__

Ejercicio 6: Implementar el método `obtenerPaisesRepresentados` que devuelve una lista con los países representados en clase (en la lista no puede haber países repetidos).

```
public List<String> obtenerPaisesRepresentados()
{
        return lMatriculados.stream()
        .map(Alumno::getPais)
        .distinct()
        .collect(Collectors.toList());
}
```

__Ayuda:__ se debe usar la operación intermedia __distinct__.

Ejercicio 7: Implementar el método `obtenerAlumnosTodosEntregablesAprobados` que devuelve la lista de alumnos que han aprobado todas los entregables.

```
public List<Alumno> obtenerAlumnosTodosEntregablesAprobados()
{
        return lMatriculados.stream()
        .filter(Alumno::todosEntregablesAprobados)
        .collect(Collectors.toList());
}
```

Además, es recomendable implementar en la clase `Alumno` el método `todosEntregablesAprobados` que se encargará de comprobar si el alumno tiene aprobados todos los entregables.

```
public boolean todosEntregablesAprobados()
{
        return lEntregables.stream()
        .allMatch(e -> e.getNota()>5);
}
```

__Ayuda:__ En el método `todosEntregablesAprobados` se debe usar la operación final __allMatch__.

Ejercicio 8: Implementar el método `obtenerAlumnosQueSuperanNotaEnEntregable` que dada una nota devuelve la lista de alumnos que hayan superado esa nota en algún entregable.

```
public List<Alumno> obtenerAlumnosQueSuperanNotaEnEntregable()
{
	return lMatriculados.stream()
	.filter(a -> a.superaNotaEnEntregable(nota))
	.collect(Collectors.toList());
}
```

Además, es recomendable implementar en la clase `Alumno` el método `todosEntregablesAprobados` que se encargará de comprobar si el alumno tiene aprobados todos los entregables.

```
public boolean superaNotaEnEntregable()
{
	return lEntregables.stream()
	.anyMatch(e->e.getNota()>nota);
}
```

Ayuda: *En el método `superaNotaEnEntregable` se debe usar la operación final* ***anyMatch****.*

Ejercicio 9: Implementar el método `obtenerAprobadosSuspendidos` que devuelve un mapa con la lista de alumnos que han aprobado y la lista de alumnos que no han aprobado.

```
public Map<Boolean,List<Alumno>> obtenerAprobadosSuspendidos()
{
	return lMatriculados.stream()
	.collect(Collectors.partitioningBy(Alumno::haSuspendido));
}
```

Ayuda: *Se debe usar el método* ***partiotioningBy*** *de la clase Collectors.*

Ejercicio 10: Implementar el método `obtenerNotaMediaPorPaises` que devuelve un mapa que indica para cada país la nota media de sus estudiantes.

```
public Map<String,Double> obtenerNotaMediaPorPaises()
{
	return lMatriculados.stream()
	.collect(groupingBy(Alumno::getPais,
	averagingDouble(Alumno::calcularNotaFinalJava8)));
}
```

Ayuda: *Se debe usar los métodos* ***groupingBy*** *y* ***averagingDouble*** *de la clase Collectors.*

Ejercicio 11: Implementar el método `imprimirEstadisticasAlumnos` que imprime las estadísticas de los alumnos de la asignatura: nota máxima, mínima y media.

```
public void imprimirEstadisticasAlumnos()
{
	DoubleSummaryStatistics st = lMatriculados.stream()
	.mapToDouble(Alumno::calcularNotaFinalJava8).summaryStatistics();
	System.out.println("Estadisticas:\n Max:" + st.getMax() + "\n Min: " +
	st.getMin() + "\n Media: " + st.getAverage()  );
}
```

Ayuda: *Se debe usar la operación terminal* ***summaryStatistics****.*

Ejercicio 12.0: Implementar el método `obtenerAlumnosPorPais` que devuelve un mapa con la lista de alumnos de cada país.

```
public Map<String,List<Alumno>> obtenerAlumnosPorPais()
{
	return lMatriculados.stream()
	.collect(groupingBy(Alumno::getPais));
}
```

*Ayuda: Se debe usar el método **groupingBy** de la clase Collectors.*

Ejercicio 12.1: Implementar el método `obtenerNotasMediasPorPais` que devuelve un mapa con la nota media de los alumnos de cada país.

```
public Map<String,Double> obtenerNotasMediasPorPais ()
{
	return lMatriculados.stream()
	.collect(groupingBy(Alumno::getPais,
	averagingDouble(Alumno::calcularNotaEntregablesJava8)));
}
```

*Ayuda: Se debe usar los métodos **groupingBy** y **averagingDouble** de la clase Collectors.*

Ejercicio 12.2: Implementar el método `obtenerAlumnoConNotaMaxPorPais` que devuelve un mapa con el alumno que ha obtenido la nota más alta de cada país.

```
public Map<String,Alumno> obtenerAlumnoConNotaMaxPorPais()
{
	return lMatriculados.stream()
	.collect(groupingBy(Alumno::getPais,
	collectingAndThen(maxBy(comparing(
	Alumno::calcularNotaFinalJava8)),Optional::get)));
}
```

*Ayuda: Se debe usar los métodos **groupingBy, collectingAndThen** y **maxBy** de la clase Collectors.*

Ejercicio 12.3: Implementar el método `obtenerNotaMaxPorPais` que devuelve un mapa con la nota más alta de cada país.

```
public Map<String,Double> obtenerNotaMaxPorPais()
{
	return lMatriculados.stream()
	.collect(groupingBy(Alumno::getPais,
	collectingAndThen(maxBy(comparing(
	Alumno::calcularNotaFinalJava8)),
	p->p.get().calcularNotaFinalJava8())));
}
```

*Ayuda: Se debe usar los métodos **groupingBy, collectingAndThen** y **maxBy** de la clase Collectors.*