OBJETIVOS

- Crear y utilizar comparadores.
- Realizar tratamientos secuenciales con *streams*.
- Elegir entre distintas clases de implementación para construir objetos.

DEFINICIÓN Y USO DE COMPARADORES

Vamos a escribir varios métodos que realizan operaciones en las cuales es necesario comparar objetos por un orden alternativo al orden natural. En cada caso deberá instanciar los objetos de tipo Comparator que necesite y utilizarlos para conseguir el objetivo del método. Escriba cada método en la clase correspondiente.

Alumno:

```
SortedSet<Asignatura> getAsignaturasOrdenadasPorCurso();
```

Devuelve un `SortedSet` con todas las asignaturas del alumno ordenadas por curso (de mayor a menor) y, a igualdad de curso, por su orden natural.

Centro:

```
SortedSet<Espacio> getEspaciosOrdenadosPorCapacidad();
```

Devuelve un `SortedSet` con los espacios del centro ordenados de mayor a menor capacidad y, a igualdad de capacidad, por su orden natural.

Expediente:

```
List<Nota> getNotasOrdenadasPorAsignatura();
```

Devuelve una lista con las notas del expediente ordenadas por asignatura y, a igualdad de asignatura, por su orden natural.

```
Nota getMejorNota();
```

Devuelve la mejor nota del expediente. Se considera la mejor nota aquella que tenga matrícula de honor. En caso de haber varias con matrícula de honor (o no haber ninguna), se devuelve la que tenga el mayor valor. A su vez, si hay varias con el mismo valor, se devuelve la obtenida en una convocatoria **anterior**, y si aún hay varias, la obtenida en el **menor curso**. Si el expediente no contiene ninguna nota, lance la excepción `NoSuchElementException`.

Grado:

```
SortedSet<Departamento> getDepartamentosOrdenadosPorAsignaturas();
```

Devuelve un `SortedSet` con los departamentos ordenados de mayor a menor número de asignaturas y, a igualdad de número de asignaturas, por su orden natural.

TRATAMIENTOS SECUENCIALES CON *STREAMS*

Redefina el método `getEspacioMayorCapacidad` del tipo **Centro** haciendo uso de *streams*. Hágalo en una nueva clase `CentroImpl2` que extienda a `CentroImpl`. En esta nueva clase sólo debe escribir el método indicado y el constructor por parámetros.

Redefina el método `getCurso` del tipo **Alumno** haciendo uso de *streams*. Hágalo en una nueva clase `AlumnoImpl2` que extienda a `AlumnoImpl`. En esta nueva clase sólo debe escribir el método indicado y los mismos constructores definidos en la clase `AlumnoImpl`.

Utilizando también *streams*, añada el siguiente método al tipo **Departamento** (e impleméntelo en la clase `DepartamentoImpl`):

> `Profesor getProfesorMaximaDedicacionMediaPorAsignatura();`
>
> Devuelve el profesor del departamento que tiene la mayor carga docente media por asignatura. Tenga en cuenta que puede haber profesores sin asignaturas asignadas. Si no hay profesores en el departamento, o los que hay no imparten ninguna asignatura, lance la excepción `NoSuchElementException`.

ELECCIÓN ENTRE DIFERENTES CLASES DE IMPLEMENTACIÓN

Con la creación de las clases `AlumnoImpl2` y `CentroImpl2`, disponemos ahora de dos clases que implementan los tipos **Alumno** y **Centro**, respectivamente. Los métodos de factoría deben permitir el uso de ambas implementaciones. Para ello, añada a la factoría un método `setUsarJava8` que permita elegir cuál de las dos clases se utilizará al construir objetos de cada uno de los tipos. Por defecto se utilizará la implementación mediante Java 8. Tome como modelo lo que se hizo en el boletín anterior con las dos implementaciones del tipo Profesor.

TEST

Añada a las clases de test de los tipos correspondientes casos de prueba para los nuevos métodos. Para probar los métodos redefinidos, utilice los métodos creacionales del tipo cambiando la clase de implementación de la forma indicada.