	FUNDAMENTOS DE PROGRAMACIÓN **EXAMEN DE TEORÍA. BLOQUE 2 DE JAVA**	**Curso: 2023/24**
APELLIDOS:	**NOMBRE:**	**DNI:**

Sean las siguientes definiciones de tipos:

Tipo **Espacio**
Propiedades:
- nombre: String, consultable
- capacidad: Integer, consultable

Tipo **Actividad**
Propiedades:
- fecha: LocalDate, consultable y modificable
- horaComienzo: LocalTime, consultable
- tipo: del tipo enumerado TipoActividad (REUNION, CONFERENCIA, TALLER, OTRO), consultable
- lugar: Espacio, consultable y modificable
- asistentes: List<String>, consultable y modificable
- inscripcion: Boolean, consultable. Toma valor true si la actividad requiere inscripción previa, y false si no la requiere
- numeroAsistentes: Integer, consultable, derivada. El número de asistentes a la actividad viene dado por el tamaño de la lista de asistentes
- maximoAsistentes: Integer, consultable, derivada. El número máximo de asistentes a la actividad viene dado por la capacidad del espacio donde se realiza

Tipo **Evento**
Propiedades:
- actividades: List<Actividad>, consultable

El tipo Espacio está implementado mediante un record, y los tipos Actividad y Evento mediante una clase.

1. Sea m un Map que relaciona cada espacio con la lista de actividades que se realizan en ese espacio. Escriba las expresiones para realizar las siguientes operaciones sobre el Map m: (1 punto)
   a) Añadir una nueva actividad a al final de la lista de actividades del espacio e
   b) Obtener el conjunto de espacios del Map m
   c) Obtener el número de claves del Map m
   d) Obtener un valor true si el Map m contiene el espacio e
   e) Obtener la lista de actividades que se realizan en el espacio e
   f) Obtener el conjunto de parejas del Map m

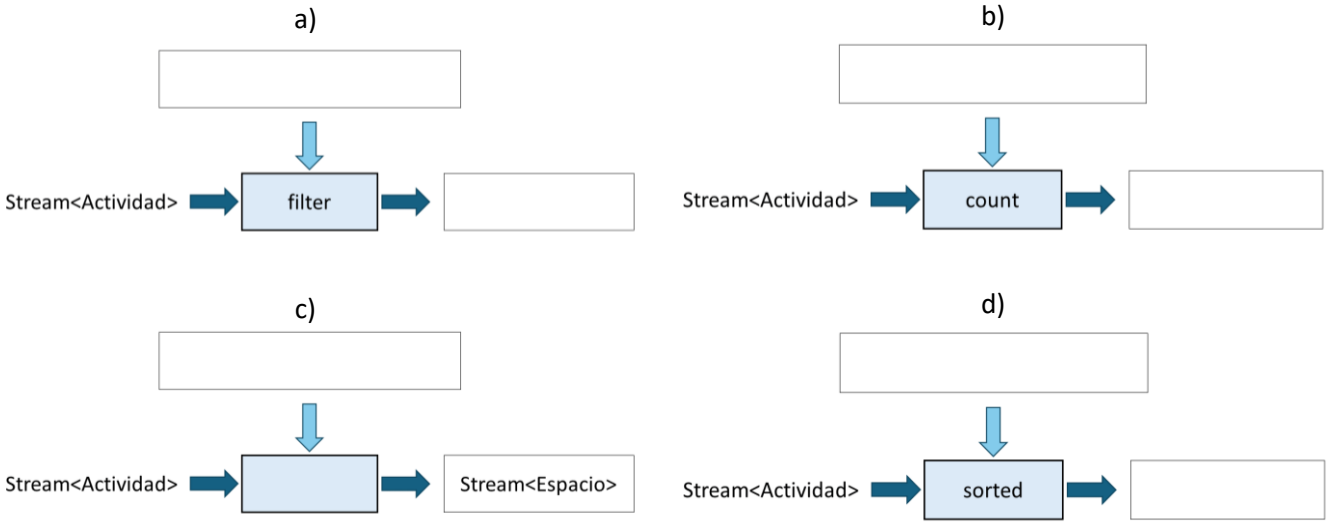
   a) **`m.get(e).add(a)`**
   b) **`m.keySet()`**
   c) **`m.size()`**
   d) **`m.containsKey(e)`**
   e) **`m.get(e)`**
   f) **`m.entrySet()`**

2. Escriba cada método de Stream en la columna que corresponda, según sea el tipo funcional del parámetro que tiene, o no tenga parámetro. (1 punto)

   `max, filter, sorted, anyMatch, map, distinct, findFirst, forEach, sum, flatMap`

Predicate	Function	Comparator	Consumer	No tiene
`filter` **`anyMatch`**	**`map`** **`flatMap`**	**`max`** **`sorted`**	**`forEach`**	**`sum`** **`distinct`** **`findFirst`**

3. Complete los siguientes gráficos añadiendo donde proceda el método de stream, el tipo funcional del parámetro (si lo hay) y el tipo del resultado obtenido al aplicar el método. (1 punto)



a) **Predicate<Actividad>, Stream<Actividad>**
b) **(no hay parámetro), Long**
c) **map, Function<Actividad, Espacio>**
d) **Comparator<Actividad>, Stream<Actividad>**

4. Escriba una expresión lambda y una referencia a método para cada uno de los siguientes predicados y funciones. Cuando no sea posible usar una referencia a método, indíquelo. (1 punto)
   a) Un predicado que tome valor true si la capacidad de un espacio es mayor que un valor n
   b) Un predicado que tome valor true si una actividad requiere inscripción previa
   c) Una función que obtenga la capacidad del espacio donde se realiza una actividad
   d) Una función que obtenga el mes en el que se realiza una actividad
   e) Una función que obtenga el porcentaje de asistentes a una actividad, dado por la relación entre el número de asistentes y la capacidad del espacio donde se realiza

   a) **Expresión lambda: e -> e.capacidad() > n**
      Referencia a método: no es posible
   b) **Expresión lambda: a -> a.getInscripcion()**
      Referencia a método: Actividad::getInscripcion
   c) **Expresión lambda: a -> a.getLugar().capacidad() o a -> a.getMaximoAsistentes()**
      Referencia a método: Actividad::getMaximoAsistentes
   d) **Expresión lambda: a -> a.getFecha().getMonth()**
      Referencia a método: no es posible
   e) **Expresión lambda: a -> a.getNumeroAsistentes() / a.getMaximoAsistentes() * 100**
      Referencia a método: no es posible

5. Dado un Stream<Actividad>, queremos obtener una lista con los nombres de los espacios de las n primeras actividades de tipo CONFERENCIA, ordenadas de la más antigua a la más reciente y sin repetir ninguno. Indique la secuencia de métodos que hay que aplicar en el orden correcto para obtener el resultado deseado. Debe indicar el nombre de cada método y el tipo de su parámetro, si lo tiene. (1 punto)

   filter con Predicate<Actividad>, sorted con Comparator<Actividad>, map con Function<Actividad, String>, distinct, limit con Integer, collect con Collector<String, List<String>>

	FUNDAMENTOS DE PROGRAMACIÓN EXAMEN DE TEORÍA. BLOQUE 2 DE JAVA	Curso: 2023/24
APELLIDOS:	NOMBRE:	DNI:

6. Cree las siguientes variables de tipo Comparator. Puede reutilizar las variables ya creadas si lo desea. Si no se indica otra cosa, se sobreentiende que se usa el orden natural de la propiedad. (1 punto)
   a) Un comparador cmp1 que compare espacios por nombre, y a igualdad de nombre por capacidad
   b) Un comparador cmp2 que compare actividades por número de asistentes de mayor a menor número de asistentes
   c) Un comparador cmp3 que compare actividades por fecha, y a igualdad de fecha por número de asistentes de mayor a menor número de asistentes
   d) Un comparador cmp4 que compare actividades por hora de comienzo, y a igualdad de hora de comienzo por el orden natural del tipo Actividad
   e) Un comparador cmp5 que compare actividades por fecha de más reciente a más antigua, y a igualdad de fecha por tipo

   a)
```
Comparator<Espacio> cmp1 = Comparator.comparing(Espacio::nombre)
      .thenComparing(Comparator.comparing(Espacio::capacidad));
```
   b)
```
Comparator<Actividad> cmp2 =
Comparator.comparing(Actividad::getNumeroAsistentes)
      .reversed();
```
   c)
```
Comparator<Actividad> cmp3 = Comparator.comparing(Actividad::getFecha)
      .thenComparing(cmp2);
Comparator<Actividad> cmp3 =
      Comparator.comparing(Actividad::getFecha).thenComparing(
         Comparator.comparing(Actividad::getNumeroAsistentes).reversed());
```
   d)
```
Comparator<Actividad> cmp4 =
      Comparator.comparing(Actividad::getHoraComienzo)
      .thenComparing(Comparator.naturalOrder());
```
   e)
```
Comparator<Actividad> cmp5 = Comparator.comparing(Actividad::getFecha)
      .reversed()
      .thenComparing(Comparator.comparing(Actividad::getTipo));
```

7. Complete el siguiente método del tipo Evento para que devuelva un valor true si existe alguna actividad del tipo dado como parámetro que tiene el número máximo de asistentes permitido por el espacio donde se realiza. (1 punto)

```
public Boolean existeActividadTipoMaximoAsistentes(TipoActividad tipo) {
      return actividades.stream()



}
```

```
public Boolean existeActividadTipoMaximoAsistentes(TipoActividad tipo) {
      return actividades.stream()
            .filter(a -> a.getTipo().equals(tipo))
            .anyMatch(a ->
                  a.getMaximoAsistentes().equals(a.getNumeroAsistentes()));
}
```

8. Indique el Collector (o Collectors, si hay más de uno) que debemos usar en el groupingBy para acumular los valores en los siguientes Map: (1 punto)
   a) Un Map que relacione cada fecha con la lista de nombres de los espacios donde se realizan las actividades de esa fecha
   b) Un Map que relacione cada tipo con el número de actividades de ese tipo
   c) Un Map que relacione cada lugar con la media de asistentes de las actividades realizadas en ese lugar
   d) Un Map que relacione cada tipo con la actividad con más asistentes de ese tipo
   e) Un Map que relacione cada fecha con el total de asistentes a las actividades realizadas en esa fecha

   a) Collectors.mapping + Collectors.toList
   b) Collectors.counting
   c) Collectors.averagingInt
   d) Collectors.collectingAndThen + Collectors.maxBy
   e) Collectors.summingInt

9. El siguiente método del tipo Evento tiene como objetivo calcular la actividad con más asistentes. Este método puede contener errores. Para cada posible error, indique su causa y explique cómo lo corregiría. (1 punto)

```
public Actividad actividadMasAsistentes() {
      return actividades.stream()
            .max(Actividad::getNumeroAsistentes);
}
```

   El parámetro del método max es de tipo Comparator y no está bien escrito. Además, max devuelve un Optional<Actividad>, por lo que hay que invocar al método get u orElse para obtener la Actividad a partir de ese objeto Optional.

```
public Actividad actividadMasAsistentes() {
      return actividades.stream()
            .max(Comparator.comparing(Actividad::getNumeroAsistentes));
            .orElse(null);
}
```

10. El siguiente método del tipo Evento tiene como objetivo calcular la suma de asistentes de todas las actividades que requieren inscripción. Este método puede contener errores. Para cada posible error, indique su causa y explique cómo lo corregiría. (1 punto)

```
public Integer totalAsistentesInscripcion() {
      return actividades.stream()
            .filter(Actividad::getInscripcion)
            .sum(Actividad::getNumeroAsistentes);
}
```

   El método sum no tiene parámetros. Recibe un stream especializado (IntStream, LongStream o DoubleStream) y suma todos sus elementos. Por tanto, hay que aplicar el método mapToInt para convertir el Stream de actividades filtrado en un IntStream (ya que la propiedad numeroAsistentes es de tipo Integer), y luego aplicar el método map sin parámetros.

```
public Integer totalAsistentesInscripcion() {
      return actividades.stream()
            .filter(Actividad::getInscripcion)
            .mapToInt(Actividad::getNumeroAsistentes)
            .sum();
}
```