	FUNDAMENTOS DE PROGRAMACIÓN **EJERCICIOS PROPUESTOS DEL BLOQUE 1 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: Integer, consultable y modificable
- inscripcion: Boolean, consultable. Toma valor true si la actividad requiere inscripción previa, y false si no la requiere
- 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:

- nombre: String, consultable
- actividades: List<Actividad>, consultable

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

1. La clase Actividad tiene un constructor con un parámetro por cada propìedad básica, en el orden en que están definidas. Escriba las instrucciones que construyen los siguientes objetos:
   a) Un espacio e con el nombre “A3.11” y la capacidad 175.
   b) Una actividad a con la fecha actual, la hora 11:30, el tipo OTRO, el lugar igual al espacio e creado antes, 70 asistentes y sin inscripción previa.

```
Espacio e = new Espacio("A3.11", 175);
Actividad a = new Actividad(LocalDate.now(), LocalTime.of(11, 30),
                            TipoActividad.OTRO, e, 70, false);
```

2. Dados un objeto e de tipo Espacio y un objeto a de tipo Actividad, escriba una expresión lógica que tome valor true si se cumple cada una de las siguientes condiciones, y false en caso contrario:
   a) El espacio donde se celebra la actividad a es igual al espacio e
   b) El nombre del espacio e no está vacío
   c) El número de asistentes a la actividad a es positivo y menor o igual que el máximo de asistentes
   d) La fecha de la actividad a es el 12 de marzo de 2024

```
a) a.getLugar().equals(e)
b) !e.nombre().isEmpty()
c) a.getAsistentes() > 0 && a.getAsistentes() <= a.getMaximoAsistentes()
d) a.getFecha().equals(LocalDate.of(2024, 3, 12))
```

3. El número de asistentes a una actividad debe ser mayor o igual que 0. Indique en qué lugar o lugares de la clase Actividad hay que tener en cuenta esta restricción, y escriba una instrucción para chequearla. Puede utilizar la clase Checkers.

Hay que chequear la restricción en el constructor de la clase y en el método setAsistentes, y se puede hacer con la siguiente instrucción:

```
Checkers.check("El número de asistentes debe ser mayor o igual que 0",
asistentes >= 0);
```

4. Dados un objeto e de tipo Espacio y un objeto a de tipo Actividad, escriba las instrucciones que realizan lo que se pide en cada caso (si algo de lo que se pide no es posible, indíquelo):
   a) Crear un espacio e con el nombre "A3.11" y la capacidad 175
   b) Aumentar un 10% el número máximo de asistentes de la actividad a
   c) Cambiar el lugar de la actividad a por el espacio e
   d) Aumentar en 10 la capacidad del espacio e
   e) Retrasar 30 minutos la hora de comienzo de la actividad a
   f) Mostrar en pantalla la capacidad del espacio donde tiene lugar la actividad a

```
a) Espacio e = new Espacio("A3.11", 175);
b) No se puede modificar el número máximo de asistentes porque es una
   propiedad derivada
c) a.setLugar(e);
d) No es posible cambiar el valor de un atributo de un record porque es un
   tipo inmutable
e) No se puede modificar la hora de comienzo porque la propiedad no es
   modificable
f) System.out.println(a.getLugar().capacidad());
```

5. Escriba en la clase Actividad un método **tipoAcceso** que devuelva una cadena de caracteres con el valor "libre" si la actividad es de tipo CONFERENCIA o TALLER, "restringido" si es de tipo REUNION, y "discrecional" si es de tipo OTRO. Use la estructura switch.

```
public String tipoAcceso() {
   String res = null;
   switch (tipo) {
      case CONFERENCIA:
      case TALLER:
         res = "libre";
         break;
      case REUNION:
         res = "restringido";
         break;
      default:
         res = "discrecional";
   }
   return res;
}
```

6. El tipo Actividad tiene un orden natural dado por la fecha, y a igualdad de fecha por la hora de comienzo. Complete el código de la clase Actividad que se muestra para implementar este orden.

```
public class Actividad                                          {
   . . .
   public int compareTo(              ) {









   }
}
```

```
public class Actividad implements Comparable<Actividad> {
   . . .
   public int compareTo(Actividad a) {
```

```
        int res = fecha.compareTo(a.getFecha());
        if (res == 0) {
            res = horaComienzo.compareTo(a.getHoraComienzo());
        }
        return res;
    }
}
```

7. Escriba los siguientes métodos de la clase Actividad:
   - Un constructor que reciba parámetros para la hora de comienzo, el tipo de actividad y el lugar, y cree una actividad con la fecha de hoy, 0 asistentes y sin inscripción previa
   - El método consultor de la propiedad lugar
   - El método modificador de la propiedad lugar
   - El método consultor de la propiedad capacidadMaxima

```
public Actividad(LocalTime horaComienzo, TipoActividad tipo, Espacio lugar) {
        this.fecha = LocalDate.now();
        this.horaComienzo = horaComienzo;
        this.tipo = tipo;
        this.lugar = lugar;
        this.asistentes = 0;
        this.inscripcion = false;
}
public Espacio getLugar() {
        return lugar;
}
public void setLugar(Espacio lugar) {
        this.lugar = lugar;
}
public Integer getCapacidadMaxima() {
        return lugar.capacidad();
}
```

8. El tipo Actividad tiene un constructor a partir de String que crea un objeto a partir de una cadena con el formato "12/3/2024, 11:30, Otro, A3.11, 175, 70, false". Complete las instrucciones del constructor a partir de String que se indican:

```
public Actividad(String s) {
        String[] sp = s.split(",");
        Checkers.check("Cadena con formato no válido", sp.length == 7);
        ...
        TipoActividad tipo = TipoActividad.valueOf(sp[2].toUpperCase().trim());
        String nombre = sp[3].trim();
        Integer capacidad = Integer.valueOf(sp[4].trim());
        Espacio e = new Espacio(nombre, capacidad);
        Integer asistentes = Integer.valueOf(sp[5].trim());
        ...
}
```

9. El tipo ActividadDePago extiende al tipo Actividad, añadiendo una propiedad **precio** de tipo Double y otra propiedad **recaudación** de tipo Double, que se calcula multiplicando el precio por el número de asistentes.
   a) Escriba un constructor de la clase ActividadDePago que reciba un parámetro por cada propiedad básica del tipo.
   b) Escriba el método consultor de la propiedad derivada recaudación.

```
public ActividadDePago(LocalDate fecha, LocalTime horaComienzo,
```

```
            TipoActividad tipo, Espacio lugar, Integer asistentes,
            Boolean inscripcion, Double precio) {
      super(fecha, horaComienzo, tipo, lugar, asistentes, inscripcion);
      this.precio = precio;
}

public Double getRecaudacion() {
      return precio * getAsistentes();
}
```

10. Sea listaActividades una lista de actividades, conjuntoActividades un conjunto ordenado de actividades, y a una actividad. Indique para cada una de las siguientes instrucciones si es correcta o incorrecta:
    a) `listaActividades.add(0, a);` **correcta**
    b) `Actividad ultima = listaActividades.last();` **incorrecta**
    c) `listaActividades.reverse();` **incorrecta**
    d) `listaActividades.set(0, a);` **correcta**
    e) `Boolean vacio = conjuntoActividades.isEmpty();` **correcta**
    f) `Actividad primera = conjuntoActividades.get(0);` **incorrecta**
    g) `Collections.sort(actividades);` **correcta**
    h) `Integer n = Collections.size(listaActividades);` **incorrecta**

11. Escriba las instrucciones necesarias para realizar las siguientes operaciones:
    a) Cree una lista de actividades con el nombre actividades, copiando el contenido de la lista l
    b) Añada la actividad a en la primera posición de la lista actividades
    c) Obtenga en la variable pos la posición de la última aparición de la actividad a en la lista actividades
    d) Elimine de la lista actividades las actividades que están presentes en el conjunto de actividades otrasActividades
    e) Ordene la lista actividades según su orden natural
    f) Elimine el primer elemento de la lista actividades
    g) Elimine el último elemento de la lista actividades
    h) Invierta las posiciones de los elementos de la lista actividades

```
a) List<Actividad> actividades = new ArrayList<>(1);
b) actividades.add(0, a);
c) Integer pos = actividades.lastIndexOf(a);
d) actividades.removeAll(otrasActividades);
e) Collections.sort(actividades);
f) actividades.remove(0)
g) actividades.remove(actividades.size() – 1);
h) Collections.reverse(actividades);
```

12. Sea el siguiente código:

```
List<Integer> lista = List.of(5, 2, -3, 0, 9, -7, 8, 10, 1, 5, -4, 14, 3);
Integer res = null;
for (Integer n: lista) {
      if (res == null || n > res) {
            res = n;
      }
}
```

    a) ¿Cuál es el valor final de la variable res? **14**
    b) Utilice la clase Collections para realizar la misma operación **res = Collections.max(lista);**

13. Sea espacios una lista de espacios. Escriba un bucle que obtenga la suma de las capacidades de los espacios cuyo nombre comience por la letra ‘A’.

```
Integer suma = 0;
for (Espacio e: espacios) {
      if (e.nombre().startsWith("A")) {
```

```
                suma += e.capacidad();
        }
}
```

14. Escriba un método del tipo Evento que reciba como parámetro una fecha y devuelva una lista con los nombres de los espacios en los que se celebren actividades en esa fecha.

```
public List<String> getNombresEspaciosActividadesFecha(LocalDate f) {
        List<String> res = new LinkedList<>();
        for (Actividad a: actividades) {
                if (a.getFecha().equals(f)) {
                        res.add(a.getLugar().nombre());
                }
        }
        return res;
}
```

15. Escriba un método del tipo Evento que reciba como parámetro un tipo de actividad y devuelva la media de asistentes a las actividades de ese tipo que requieran inscripción previa. Si no hay actividades de ese tipo que requieran inscripción, el valor devuelto será null.

```
public Double getMediaAsistentesActividadesTipoConInscripcion(TipoActividad t) {
        Double media = null;
        Double suma = 0.;
        Integer cont = 0;
        for (Actividad a: actividades) {
                if (a.getInscripcion() && a.getTipo().equals(t)) {
                        suma += a.getAsistentes();
                        cont++;
                }
        }
        if (cont > 0) {
                media = suma / cont;
        }
        return media;
}
```

16. Escriba un método del tipo Evento que reciba como parámetro un numero entero n y devuelva un valor true si todas las actividades tienen un número de asistentes superior al n% de la capacidad del espacio donde se realizan, o false en caso contrario.

```
public Boolean todasPorcentajeAsistentesMayor(Integer n) {
        Boolean res = true;
        for (Actividad a: actividades) {
                if (a.getAsistentes() <= n / 100. * a.getCapacidadMaxima()) {
                        res = false;
                        break;
                }
        }
        return res;
}
```