Ejercicio 1

➪ ➪ Ejercicio 1: especificación *importe_alquiler_coche()*

Crea la subcarpeta "**importe_alquiler**", en la que guardarás tu solución para este ejercicio.

Puedes crear uno o varios ficheros. En el caso de que la solución esté dividida en varios ficheros, todos ellos tendrán como nombre "**alquiler-<sufijo>.<extension>**", siendo **<sufijo>** la indicación del paso o pasos seguidos: por ejemplo alquiler-paso1.jpg, alquiler-pasos2-3.jpg,... o simplemente "**alquiler.<extension>**" si todo el ejercicio está resuelto en un fichero. **<extension>** denota el tipo de fichero: jpg, png, ...

En una aplicación de un negocio de alquiler de coches necesitamos una unidad denominada ***importe_alquiler_coche()***. Dicha unidad calcula el importe del alquiler de un determinado tipo de coche durante un cierto número de días, a partir de una fecha concreta, y devuelve el importe de dicho alquiler. Si no es posible realizar los cálculos devuelve una excepción de tipo *ReservaException*. El prototipo del método es el siguiente:

```
public float importe_alquiler_coche (TipoCoche tipo, LocalDate fecha_inicio,
                                     int num_dias) throws ReservaException
```

TipoCoche es un tipo enumerado cuyos posibles valores son: (TURISMO, DEPORTIVO). Asumimos que la fecha de inicio ha sido validada en otra unidad. Nos indican que si la fecha de inicio proporcionada no es posterior a la actual, entonces se lanzará la excepción *ReservaException* con el mensaje "Fecha no correcta". Si el tipo de coche no está disponible durante los días requeridos, o se intenta hacer una reserva de más de 30 días, entonces se lanzará la excepción *ReservaException* con el mensaje "Reserva no posible".

El precio de la reserva por día depende del número de días reservados, según la siguiente tabla:

1 ó 2 días	100 euros/dia
más de 2 días	50 euros/dia

Diseña los casos de prueba teniendo en cuenta la especificación anterior utilizando el método de particiones equivalentes.

TipoCoche : tipoC

T1 : tipoC ∈ {TURISMO, DEPORTIVO}

NT1 : tipoC ∉ {TURISMO, DEPORTIVO}

Fecha_inicio : fechaI

F1: fechaI > fechaActual

NF1: fechaI ≤ fechaActual

numDias : nDias

D1: nDias = {1 ... 30}

ND1: nDias < 1

ND2: nDias > 30

Disponibilidad : d

V1: d = Disponible

NV1: d = No disponible

SalidaEsperada
float o ReservaException

S1: importe * 1 o 2 días

S2: importe * más 2 días

NS1: "Fecha no correcta"

NS2: "Reserva no disponible"

La fecha actual es 25-12-2024

Camino	tipoC	fechaI	nDias	Disponibilidad	Salidas float o ReservaException
T1-F1-D1-V1-S1	TURISMO	31-12-2024	1	Disponible	100 * 1
T1-F1-D1-V1-S2	TURISMO	31-12-2024	5	Disponible	50 * 5
NT1-F1-D1-V1- ??	CARAVANA	31-12-2024	1	Disponible	???
T1-NF1-D1-V1-NS1	TURISMO	24-12-2024	1	Disponible	"Fecha no correcta"
T1-F1-ND1-V1-NS2	TURISMO	31-12-2024	1	Disponible	"Reserva no disponible
T1-F1-ND2-V1-NS2	TURISMO	31-12-2024	1	Disponible	"Reserva no disponible"
T1-F1-D1-NV1-NS2	TURISMO	31-12-2024	1	No disponible	"Reserva no disponible"

Ejercicio 2

Ejercicio 2: especificación *generaEventos()*

Crea la subcarpeta "**generaEventos**", en la que guardarás tu solución para este ejercicio.

Igual que antes, puedes crear uno o varios ficheros. En el caso de que la solución esté dividida en varios ficheros, todos ellos tendrán como nombre "**generaEventos-<sufijo>.<extension>**

En una aplicación de matriculación de una universidad, queremos implementar una unidad denominada ***generaEventos()***, que devuelve una lista de eventos de calendario para todas las sesiones de clase de una asignatura, (asumiremos 1 única clase por semana), o bien una excepción de tipo ***ParseException***, de acuerdo con las siguientes reglas sobre las asignaturas:

- **R1**. Asumimos que ni el nombre de la asignatura ni ninguno de los objetos que representan la fecha (tipo LocalDate) serán null (y además la fecha será una fecha válida).
- **R2.** Si la hora de inicio tiene un formato o valores incorrectos, o el día de la semana no es uno de los valores válidos, se devolverá una instancia de *ParseException*
- **R3.** La fecha de inicio especificada puede ser posterior a la de fin. En tal caso se devolverá una lista de eventos vacía. También se devolverá una lista de eventos vacía si el día de la semana no está incluido en el rango de fechas del curso académico (por ejemplo, si la fecha de inicio de curso fuese miércoles (22/02/23) y la fecha de fin el viernes (24/02/23), y el día de la semana en la que se imparte la asignatura fuese los martes, la salida será una lista vacía)
- **R4**. Si la hora de inicio de la clase es null, se considerará un evento de todo el día y la duración será -1 (la hora de inicio del evento también será null). En caso contrario, la duración contendrá un valor de 120 minutos.

El prototipo del método a probar será el siguiente:

```
List<EventoCalendario> generaEventos(HorarioAsignatura horario) throws ParseException;
```

Los tipos ***HorarioAsignatura*** y ***EventoCalendario*** se definen como:

```
public class HorarioAsignatura {
   String asignatura;
   LocalDate fechaInicioCurso;
   LocalDate fechaFinCurso;
   String horaInicioClase; // Formato "hh:mm"
   int diaSemana;  // Valores válidos:
                   //    1(lunes),2,3,4,5,6(sábado)
}
```

```
public class EventoCalendario {
   String nombreAsig;
   LocalDate fechaDeSesion;
   String horaInicio;
   int duracion;
}
```

Diseña los casos de prueba teniendo en cuenta la especificación anterior utilizando el método de particiones equivalentes.

Asignatura : asignatura

A1 : String

horaInicioClase : hora

H1 : hora tiene formato hh:mm && valores correctos

H2 : hora == null

NH1 : hora en formato incorrecto

NH2 : hora con valores incorrectos

fechaInicio - diaSemana - fechaFinCurso : fd

F1 : diaSemana = {1...6}
fechaInicio < diaSemana < fechaFin

NF1 : diaSemana < 1

NF2 : diaSemana > 6

NF3 : fechaInicio > FechaFin

NF4 : diaSemana < fechaInicio

NF5 : diaSemana > fechaFin

SalidaEsperada

S1 : List<EventoCalendario>

NS1 : List<>()

NS2 : ParseException()

						Salidas				
Camino	asignatura	fechaI	diaSm	fechaF	hora	Duración	fecha	hora	asignatura	List<Eventos> List<>() ParseException
A1-F1-H1-S1	"ppss"	19-02-25	4	22-02-25	9:00	2	20-02-25	9:00	"ppss"	List<Evento>
A1-F1-H2-S1	"ppss"	19-02-25	4	22-02-25	null	-1	20-02-25	—	"ppss"	Evento todo el día
A1-NF1-H1-NS2	"ppss"	19-02-25	0	22-02-25	9:00	—	—	—	—	ParseException
A1-NF2-H1-NS2	"ppss"	19-02-25	7	22-02-25	9:00	—	—	—	—	ParseException
A1-NF3-H1-NS1	"ppss"	22-02-25	4	19-02-25	9:00	—	—	—	—	List<>()
A1-NF4-H1-NS1	"ppss"	19-02-25	1	22-02-25	9:00	—	—	—	—	List<>()
A1-NF5-H1-NS1	"ppss"	17-02-25	5	19-02-25	9:00	—	—	—	—	List<>()
A1-F1-NH1-NS2	"ppss"	19-02-25	4	22-02-25	9,00	—	—	—	—	ParseException
A1-F1-NH2-NS2	"ppss"	19-02-25	4	22-02-25	80:80	—	—	—	—	ParseException

Ejercicio 3

➪ ➪ Ejercicio 3: especificación *agregarGrabacion()*

Crea la subcarpeta "**agregarGrabacion**", en la que guardarás tu solución para este ejercicio.

Puedes crear uno o varios ficheros. En el caso de que la solución esté dividida en varios ficheros, todos ellos tendrán como nombre "**agregarGrabacion-<sufijo>.<extension>**

En una aplicación para gestionar la grabación de programas de TV queremos implementar un método para añadir una nueva grabación a una lista de grabaciones. El prototipo del método a probar es el siguiente :

```
void agregarGrabacion(ArrayList<Grabacion> grabaciones, Grabacion p)
                                                  throws GrabacionException;
```

Cada grabación (clase **Grabacion**) está formada por los campos **canal**, **inicio** y **fin**. Todos de tipo entero, que representan el número de canal de la grabación, y el minuto inicial y final de dicha grabación, respectivamente.

Los canales están numerados desde 1 hasta 20. El sistema de grabación permite grabar en cualquiera de los canales disponibles, pero no puede simultanear dos grabaciones, es decir, las grabaciones no pueden "solaparse" en el tiempo.

En la lista de grabaciones nunca hay elementos que "solapan" entre ellos y en todo momento está ordenada por los valores de inicio de cada elemento de la lista. El sistema aceptará grabaciones con una duración mínima de un minuto.

Asumiremos que los valores de los atributos inicio y fin de la grabación a añadir siempre tendrán valores comprendidos entre 1 y 1440 (que son los minutos de un día). Y también asumiremos que la lista de grabaciones nunca va a tener el valor null.

La nueva grabación que se pasa por parámetro se añadirá a la lista sólo si no solapa con ninguna otra ya existente en dicha lista, en cuyo caso se añadirá en la posición que corresponda para mantener la lista ordenada.

Si la grabación que se quiere añadir solapa con alguna de las de la lista se lanzará la excepción GrabacionException con el mensaje "Solape de grabaciones". Si el canal indicado no existe se lanzará la excepción GrabacionException con el mensaje "No existe el canal", Si el intervalo de grabación a añadir tiene una duración cero, entonces se lanzará la excepción GrabacionException con el mensaje "Error en intervalo de grabación"

Diseña los casos de prueba teniendo en cuenta la especificación anterior utilizando el método de particiones equivalentes.

Salida Esperada

S1: ArrayList<Grabacion>

NS1: GrabacionException "Solape de grabaciones"

NS2: GrabacionException "No existe el canal"

NS3: GrabacionException "Error en intervalo de grabacion"

NS4: ???

Canal : c

C1: c = {1...20}

C2: C_A != C_B

NC1: c < 1

NC2: c > 20

NC3: C_A == C_B

Inicio-Fin : M

M1: (inicioB, fin > 1) && (inicioB, fin < 1440) && inicioB < finB && inicioB > finB

M2: (inicioB, fin > 1) && (inicioB, fin < 1440) && inicioB < finB && finB ≤ inicioA

NM1: 1 < inicioB

NM2: 1440 < inicioB

NM3: 1 > finB

NM4: 1440 < finB

NM5: inicioB > finB

NM6: inicioB == finB

NM7: inicioA == inicioB && finA < finB

NM8: finA == finB && inicioA < inicioB

NM9: inicioA == inicioB && finA == finB

NM10: inicioB > inicioA && finB > inicioA

NM11: inicioB > inicioA && inicioB < finA

NM12: inicioA < inicioB && finA > finB

NM13: inicioB < inicioA && finA < finB

A → Grabacion existente

B → Grabacion Entrada

NS9

inicioA == inicioB
finA == finB

NS10

inicioB < inicioA
finB > inicioA

NS11

inicioB > inicioA
inicioB < finA

NS12

inicioA < inicioB
finA > finB

NS13

inicioB < inicioA
finA < finB

NS7

inicioA == inicioB
finA < finB

NS8

inicioB < inicioA
finA == finB

		Entrada: Grabación			Salidas	
Camino	Lista de Grabaciones	Canal	Inicio	Fin	Lista	GrabacionExcep o void()
C1-M1-S1	{{1,3,5}}	1	11	14	[(1,3,5),(1,6,9)]	—
NC1-M1-NS2	{{1,3,5}}	0	11	14	[(1,3,5)]	"No existe el canal"
NC2-M1-NS2	{{1,3,5}}	30	11	14	[(1,3,5)]	"No existe el canal"
NC3-M1-NS1	{{1,3,5}}	1	11	14	[(1,3,5)]	"Solape de grabaciones"
C1-NM1-NS4	{{1,3,5}}	1	0	14	[(1,3,5)]	???
C1-NM2-NS4	{{1,3,5}}	1	1442	14	[(1,3,5)]	???
C1-NM3-NS4	{{1,3,5}}	1	11	0	[(1,3,5)]	???
C1-NM4-NS4	{{1,3,5}}	1	11	1442	[(1,3,5)]	???
C1-NM5-NS4	{{1,3,5}}	1	8	3	[(1,3,5)]	???
C1-NM6-NS3	{{1,3,5}}	1	7	7	[(1,3,5)]	"Error en intervalo de grabacion"
C1-NM7-NS1	{{1,3,5}}	1	3	14	[(1,3,5)]	"Solape de grabaciones"
C1-NM8-NS1	{{1,3,5}}	1	4	5	[(1,3,5)]	"Solape de grabaciones"
C1-NM9-NS1	{{1,3,5}}	1	3	5	[(1,3,5)]	"Solape de grabaciones"
C1-NM10-NS1	{{1,3,5}}	1	2	4	[(1,3,5)]	"Solape de grabaciones"
C1-NM11-NS1	{{1,3,5}}	1	4	8	[(1,3,5)]	"Solape de grabaciones"
C1-NM12-NS1	{{1,3,7}}	1	4	6	[(1,3,5)]	"Solape de grabaciones"
C1-NM13-NS1	{{1,3,5}}	1	2	8	[(1,3,5)]	"Solape de grabaciones"
C2-M1-S1	{{1,3,5}}	2	3	5	[(1,3,5), (2,3,5)]	—
C1-M2-S1	[(1,11,14)]	1	3	5	[(1,3,5), (1,11,14)]	—