UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Trabajo teórico 2

Grupo Trabajo	Grupo 3b1
Componentes	1. Chorouk chemmari 2. Imane El Harradji Aoury 3. Adrián Díaz-Plaza Remesal 4. Arturo González Rojas 5. Sergey Ghukasyan Poghosyan 6. Victor Cerrillo Arévalo

Problema 1

Realizado por Víctor Cerrillo Arévalo y Adrián Díaz-Plaza Remesal

1. Escribir, al menos el pseudocódigo correspondiente al método o a los métodos identificados.

```java
public class Fecha {
    private int dia;
    private int mes;
    private int anyo;


    public Fecha (String strFecha) throws FormatoNoValido, FechaNoValida{
        parsearFecha(strFecha);
        comprobarFecha();
    }

    private void parsearFecha (String strFecha) throws FormatoNoValido{

        String fecha [] = {};
        fecha = strFecha.split("/");
        if (fecha.length != 3){
           throw new FormatoNoValido (mensaje: "Error: La fecha debe tener el formato 'dd/mm/yyyy'.");
        }

        try{
            dia = Integer.parseInt(fecha[0]);
            mes = Integer.parseInt(fecha[1]);
            anyo = Integer.parseInt(fecha[2]);
        }catch (NumberFormatException e){
            throw new FormatoNoValido (mensaje: "Error: La fecha solo debe contener números.");
        }

    }

    private void comprobarFecha() throws FechaNoValida{
        if (dia < Constantes.MIN_VALUE || mes < Constantes.MIN_VALUE || anyo < Constantes.MIN_VALUE)
            throw new FechaNoValida("Error: Los valores de día, mes y año deben ser mayor a "+Constantes.MIN_VALUE);
        if (mes > Constantes.MAX_MES_VALUE)
            throw new FechaNoValida("Error: El mes " + mes + " no es válido. Debe ser menor a: " + Constantes.MAX_MES_VALUE + ".");
        int maxDiaMes = Constantes.MAX_DIA_VALUE[mes];
        if (mes == 2 && !esBisiesto()) {
            maxDiaMes = 28;
        }
        if (dia > maxDiaMes)
            throw new FechaNoValida("Error: El día " + dia + " no es válido para el mes " + mes + ". Este mes solo tiene " + maxDiaMes + " días.");
    }
```

```java
    public boolean esBisiesto(){
        boolean bisiesto = (anyo % 4 == 0 && anyo % 100 != 0) || (anyo % 400 == 0);
        return bisiesto;
    }
}
```

Constantes:

MIN_VALUE ← 1

MAX_MES_VALUE ← 12

MAX_DIA_VALUE ← array con el número máximo de días por mes

2. Identificar las unidades de prueba.

La unidad de prueba es la clase **Fecha**.

La clase Fecha concentra toda la logica de validación y cálculo del problema. Aunque la clase contiene varios métodos, son privados o no tienen punto de entrada independiente, por lo que su comportamiento solo puede observarse a través del constructor. Por ese motivo, las pruebas se realizarán a nivel de clase, no de método.

3. Identificar las variables que se deben tener en cuenta para probar los métodos de interés incluidos en las unidades de prueba.

Las variables relevantes para definir los casos de prueba son la variable de entrada directa al sistema, **strFecha**, así como los enteros **dia**, **mes** y **anyo** derivadas del String. Estas variables influyen directamente en las validaciones realizadas en el constructor de la clase Fecha y en el resultado del método esBisiesto().

En resumen:

- strFecha: String
- dia: int
- mes: int
- anyo: int

4. Identificar los valores de pruebas para cada una de las variables anteriores que conformarán los correspondientes casos de prueba usando las tres técnicas vistas en teoría (clases de equivalencia, teoría de los valores límites y conjetura de errores), especificando para cada valor obtenido cuál es la que ha sido usada.

Valores límite: Conjetura de errores:

Parámetros	Clases de equivalencia	Valores seleccionados	Valores totales
strFecha	Formato correcto Formato incorrecto	Formato correcto “a" “a/a/a” “” String con más de 255 caracteres	5
dia	(-Inf, 1) ∪ [1, 31] ∪ (31, +Inf)	-5, 0, 1,15, 31, 32, 40	7
mes	(-Inf, 1) ∪ [1, 12] ∪ (12, +Inf)	-3, 0, 1, 5, 12, 13, 20	7
anyo	(-Inf, 1) [1, +inf) y no bisiesto [1, +inf) y bisiesto	0, 1, 1900, 2024	4

Los valores de prueba se han seleccionado aplicando clases de equivalencia, valores límite y conjetura de errores. Para cada parámetro se han definido valores válidos e inválidos representativos, incluyendo limites inferiores y superiores, así como casos típicos de error en el parámetro strFecha. En el caso de anyo, se han considerado también años bisiestos y no bisiestos.

El dominio del parámetro **dia** varía en función del mes seleccionado. Para facilitar la generación de los casos de prueba para each-use y pair-wise, se han elegido únicamente meses de 31 días.

5. Calcular el número máximo posible de casos de pruebas que se podrían generar a partir de los valores de pruebas (combinatoria).

nº valores dia * nº valores mes * nº valores anyo = 7 * 7 * 5 = 245 casos de prueba

El parámetro strFecha no forma parte de la combinatoria, ya que, en caso de formato incorrecto, se lanza una excepción y no se llegan a asignar valores a los parámetros dia, mes y año.
Por tanto, los casos de prueba correspondientes a formatos incorrectos de strFecha son independientes. Al añadir los casos de prueba definidos para este parámetro, el número total de casos de prueba asciende a 249.

6. Defina un conjunto de casos de pruebas para cumplir con each use (cada valor una vez).

El conjunto de casos de prueba se ha diseñado para cumplir el criterio each-use, garantizando que cada valor aparece al menos una vez. Se han combinado los valores de forma estratégica con el objetivo de obtener la mayor cobertura posible y ver los diferentes comportamientos de la clase Fecha.

Caso	strFecha	dia	mes	anyo	Resultado esperado
1	“-5/1/2024”	-5	1	2024	FechaNoValida
2	“0/0/1”	0	0	1	FechaNoValida
3	“1/-3/1900”	1	-3	1900	FechaNoValida
4	“15/5/0”	15	5	2024	Fecha válida (esBisiesto = True)
5	“31/12/1900”	31	12	1900	Fecha válida (esBisiesto = False)
6	“32/13/2024”	32	13	2024	FechaNoValida
7	“40/20/2024”	40	20	0	FechaNoValida
8	“a”	-	-	-	FormatoNoValido
9	String > 255 caracteres	-	-	-	FormatoNoValido
10	“”	-	-	-	FormatoNoValido
11	“a/a/a”	-	-	-	FormatoNoValido

7. Defina conjuntos de pruebas para alcanzar cobertura pairwaise usando el algoritmo explicado en clase. Se pueden comprobar los resultados con el programa PICT5.

Se han generado los casos de prueba que cubren todas las combinaciones posibles de pares de valores entre los parámetros definidos. De esta manera, podemos reducir la combinatoria completa manteniendo una buena cobertura.

Únicamente se ha hecho pairwise con las variables dia, mes y anyo, ya que para que puedan recibir un valor, el formato de strFecha deberá ser correcto.

El programa utilizado para generar los casos de prueba de pairwise es el siguiente: https://github.com/microsoft/pict

Caso	strFecha	dia	mes	anyo	Resultado esperado
1	“32/5/1”	32	5	1	FechaNoValida
2	“40/0/2024”	40	0	2024	FechaNoValida
3	“0/20/1”	0	20	1	FechaNoValida
4	“0/5/2024”	0	5	2024	FechaNoValida
5	“40/-3/0”	40	-3	0	FechaNoValida
6	“15/5/0”	15	5	0	FechaNoValida
7	“31/-3/1”	31	-3	1	FechaNoValida
8	“1/-3/1900”	1	-3	1900	FechaNoValida
9	“40/20/1900”	40	20	1900	FechaNoValida
10	“40/1/1”	40	1	1	FechaNoValida
11	“-5/1/1900”	-5	1	1900	FechaNoValida
12	“-5/12/1”	-5	12	1	FechaNoValida
13	“32/13/2024”	32	13	2024	FechaNoValida
14	“32/-3/0”	32	-3	0	FechaNoValida
15	“40/13/1900”	40	13	1900	FechaNoValida
16	“15/12/2024”	15	12	2024	Fecha válida (esBisiesto = True)
17	“32/1/1900”	32	1	1900	FechaNoValida
18	“0/12/1900”	0	12	1900	FechaNoValida
19	“31/12/0”	31	12	0	FechaNoValida
20	“1/13/0”	1	13	0	FechaNoValida
21	“32/12/0”	32	12	0	FechaNoValida
22	“-5/0/0”	-5	0	0	FechaNoValida
23	“15/20/0”	15	20	0	FechaNoValida
24	“40/5/1900”	40	5	1900	FechaNoValida
25	“3/1/2024”	31	1	2024	Fecha válida (esBisiesto = True)
26	“0/13/0”	0	13	0	FechaNoValida
27	“15/13/1”	15	13	1	FechaNoValida
28	“15/-3/2024”	15	-3	2024	FechaNoValida
29	“15/1/0”	15	1	0	FechaNoValida
31	“1/5/2024”	1	5	2024	Fecha válida (esBisiesto = True)
31	“-5/13/2024”	-5	13	2024	FechaNoValida
32	“31/0/1900”	31	0	1900	FechaNoValida
33	“40/12/1900”	40	12	1900	FechaNoValida
34	“0/0/1”	0	0	1	FechaNoValida

35	“32/20/2024”	32	20	2024	FechaNoValida
36	“31/13/1”	31	13	1	FechaNoValida
37	“1/20/1”	1	20	1	FechaNoValida
38	“15/0/1900”	15	0	1900	FechaNoValida
39	“-5/5/0”	-5	5	0	FechaNoValida
40	“0/1/2024”	0	1	2024	FechaNoValida
41	“-5/20/2024”	-5	20	2024	FechaNoValida
42	“-5/-3/0”	-5	-3	0	FechaNoValida
43	“0/-3/0”	0	-3	0	FechaNoValida
44	“1/0/1”	1	0	1	FechaNoValida
45	“31/5/1900”	31	5	1900	Fecha válida (esBisiesto = False)
46	“31/20/2024”	31	20	2024	FechaNoValida
47	“1/12/0”	1	12	0	FechaNoValida
48	“32/0/0”	32	0	0	FechaNoValida
49	“1/1/1”	1	1	1	Fecha válida (esBisiesto = False)

8. Para los trozos de código que incluyan decisiones, proponga conjunto de casos de prueba para alcanzar cobertura de decisiones

Se han identificado todas las decisiones presentes en el código y se han definido los casos de prueba que permiten que cada decisión tome tanto el valor verdadero como el falso al menos una vez.

D1: fecha.length != 3

D2: dia < Constantes.MIN_VALUE || mes < Constantes.MIN_VALUE || anyo < Constantes.MIN_VALUE

D3: mes > Constantes.MAX_MES_VALUE

D4: mes == 2 && !esBisiesto()

D5: dia > maxDiaMes

D6: (anyo % 4 == 0 && anyo % 100 != 0) || (anyo % 400 == 0)

Casos	strFecha	dia	mes	anyo	D1	D2	D3	D4	D5	D6	Resultado esperado
1	“1/1/2024”	1	1	2024	F	F	F	F	F	T	Fecha Valida (esBisiesto = true)
2	“1/1/2023”	1	1	1900	F	F	F	F	F	F	Fecha Valida (esBisiesto = false)
3	“0/1/2024”	0	1	2024	F	T	-	-	-	-	FechaNoValida
4	“1/13/2024”	1	13	2024	F	F	T	-	-	-	FechaNoValida

5	“29/2/2023”	29	2	2023	F	F	F	T	T	F	FechaNoValida
6	“1/1”	-	-	-	T	-	-	-	-	-	FormatoNoValida

Debido a que la mayoría de las decisiones decisiones de la clase Fecha provocan el lanzamiento de una excepción cuando se evalúan a True, el flujo de ejecución se interrumpe y el resto de decisiones no llegan a evaluarse. Por este motivo, ha sido necesario diseñar casos de prueba adicionales para cumplir con el criterio de cobertura de decisiones.

9. Para los trozos de código que incluyan decisiones, proponga conjunto de casos de prueba para alcanzar cobertura MC/DC.

Para las decisiones con múltiples condiciones, se aplica MC/DC, construyendo las tablas de verdad y seleccionando casos de prueba que permiten demostrar la influencia independiente de cada condición sobre el resultado final de la decisión.

En todas las decisiones, se han elegido casos de prueba que, manteniendo constantes el resto de las condiciones, la variación de una única condición provoca un cambio en la decisión.

dia < Constantes.MIN_VALUE || mes < Constantes.MIN_VALUE || anyo < Constantes.MIN_VALUE

A or B or C				
Condiciones			Decisión	Condicion dominante
A	B	C	A or B or C	
True	True	True	True	-
True	True	False	True	-
True	False	True	True	-
True	False	False	True	A
False	True	True	True	-
False	True	False	True	B
False	False	True	True	C
False	False	False	False	A, B, C

mes == 2 && !esBisiesto()

A and B			
Condiciones		Decisión	Condicion dominante
A	B	A and B	
True	True	True	A, B
True	False	False	B
False	True	False	A
False	False	False	A, B

(anyo % 4 == 0 && anyo % 100 != 0) || (anyo % 400 == 0)

(A and B) or C				
Condiciones			Decisión	Condicion dominante
A	B	C	(A and B) or C	
True	True	True	True	-
True	True	False	True	A, B
True	False	True	True	C
True	False	False	False	B,C
False	True	True	True	C
False	True	False	False	A, C
False	False	True	True	C
False	False	False	False	C

10. Comente los resultados del número de los casos de pruebas conseguidos en los apartados 4, 5 y 6 ¿qué podría decirse de la cobertura alcanzada? ¿en qué medida la implementación del programa influye en el diseño e implementación de los casos de pruebas?

La cobertura alcanzada utilizando únicamente each-use es limitada, ya que no asegura la cobertura de las interacciones entre parámetros ni de todas las decisiones del programa.

La implementación del programa influye directamente en el diseño y número de casos de prueba, ya que la existencia de múltiples decisiones encadenadas que pueden lanzar excepciones provoca que se interrumpa la ejecución, impidiendo que el resto de las decisiones se evalúen, teniendo que diseñar casos de prueba específicos para cada decisión. Además, las decisiones con múltiples condiciones son difíciles de cubrir adecuadamente sin recurrir a criterios más exigentes como MC/DC.

Por último, el uso de métodos privados dificulta el diseño e implementación de pruebas, ya que limita los posibles puntos de acceso a la unidad de pruebas. En el caso de la clase Fecha, hay un único punto de entrada para la ejecución de pruebas, el parámetro strFecha del constructor, a partir del cual deben validarse indirectamente las variables dia, mes y anyo, que son las que determinan gran parte del comportamiento del sistema.

En resumen, para facilitar el diseño de casos de prueba, implementación del programa debería evitar o reducir al máximo, siempre que sea posible, las decisiones encadenadas con retorno, las decisiones con múltiples condiciones y los métodos privados.

Problema 2

Realizado por Imane El Harradji Aoury y Chorouk Chemmari

1. Escribir, al menos el pseudocódigo correspondiente al método o a los métodos identificados:

```java
public class TarifaAerea {

    public static String calcularTarifa(
            int edad, int vuelosAnuales, boolean estudiante, boolean
viveConPadres,
            String clase, String destino, double ingresos, boolean conNinos)
    {

        String tarifa = "Sin tarifa";

        // Regla 1: Pajarillo
        if (edad < 18 && vuelosAnuales >= 6) {
            tarifa = "Pajarillo (10%)";
        }

        // Regla 2: Gorrión
        else if (edad >= 18 && edad <= 25 && estudiante &&
                 clase.equalsIgnoreCase("turista")) {
            tarifa = "Gorrión (15%)";
        }

        // Regla 3: Viaja ahora / Saltar del Nido
        else if (edad >= 18 && edad <= 25 && !estudiante) {
            if (viveConPadres && vuelosAnuales >= 3 &&
clase.equalsIgnoreCase("turista")) {
                tarifa = "Viaja ahora que puedes (5%)";
            }
            else if (!viveConPadres) {
                tarifa = "Atreviéndose a saltar del Nido (25%)";
            }
        }

        // Regla 4: Conoce Europa
        else if (edad > 25 && ingresos > 20000 && ingresos < 35000 &&
                 vuelosAnuales >= 6 && clase.equalsIgnoreCase("turista") &&
                 destino.equalsIgnoreCase("Europa")) {

            if (conNinos) {
                tarifa = "Conoce Europa con tus peques (10%)";
            } else {
                tarifa = "Conoce Europa (15%)";
            }
        }

        // Regla 5: Conoce el Mundo
        else if (edad > 25 && ingresos > 35000 && vuelosAnuales >= 6 &&
```

```
                clase.equalsIgnoreCase("business") &&
                (destino.equalsIgnoreCase("Asia") ||
destino.equalsIgnoreCase("America"))) {

            if (conNinos) {
                tarifa = "Conoce el Mundo con tus peques (10%)";
            } else {
                tarifa = "Conoce el Mundo (20%)";
            }
        }

        return tarifa;
    }

}
```

2. Identificar las unidades de prueba.

Unidad de prueba 1 — Tarifa "Pajarillo"

Condición principal:

- edad < 18
- vuelos ≥ 6

Unidad de prueba 2 — Tarifa "Gorrión"

Condiciones:

- 18 ≤ edad ≤ 25
- estudiante = true
- clase = "turista"

Unidad de prueba 3 — Tarifa "Viaja ahora que puedes" / "Atreviéndose a saltar del Nido"

Condiciones comunes:

- 18 ≤ edad ≤ 25
- estudiante = false

Casos dentro de la unidad:

— "Viaja ahora que puedes"

- viveConPadres = true
- vuelos ≥ 3
- clase = "turista"

— "Atreviéndose a saltar del Nido"

- viveConPadres = false

Unidad de prueba 4 — Tarifas Europa

Condiciones:

- edad > 25
- ingresos 20.000–35.000
- vuelos ≥ 6
- clase = “turista”
- destino = “Europa”

Casos:

- con niños → "Conoce Europa con tus peques"
- sin niños → "Conoce Europa"

Unidad de prueba 5 — Tarifas Mundo

Condiciones:

- edad > 25
- ingresos > 35.000
- vuelos ≥ 6
- clase = “business”
- destino = “Asia” o “America”

Casos:

- con niños → "Conoce el Mundo con tus peques"
- sin niños → "Conoce el Mundo"

3. Identificar las variables que se deben tener en cuenta para probar los métodos de interés incluidos en las unidades de prueba.

- Edad: int
- Vuelos: int
- Estudiante: boolean
- viveConPadres: boolean
- clase: String
- destino: String
- ingresos: double
- conNinos: boolean

4. Identificar los valores de pruebas para cada una de las variables anteriores que conformarán los correspondientes casos de prueba usando las tres técnicas vistas en teoría (clases de equivalencia, teoría de los valores límites y conjetura de errores), especificando para cada valor obtenido cuál es la que ha sido usada.

Parametros	Clases de equivalencia	Valores de prueba	Conjetura de errores	Valores totales
edad	(-00, 0], [1,17],[18,25],[25,+00)	0,17,18,25,26	0,-1,2x10^	10

			6, “e”, “.”	
Vuelos	(-00,2],[3,6),[6,+00)	2,3,5,6	-1, “a”, “.”	7
Estudiante	True, false	True, false	“ “	3
viveConPadres	True, false	True, false	“ “	3
Clase	Turista, business	Turista, business	200, “ “	4
Destino	Europa, Asia, America	Europa, Asia, America	-1000, “ “	5
ingresos	(-00,20000),[20000,35000),[35000,+00)	19999,20000,34999,35000, 50000	“o”	6
conNinos	True, false	True, false	{“ “}	3

5. Calcular el número máximo posible de casos de pruebas que se podrían generar a partir de los valores de pruebas (combinatoria).

c) Cobertura “cada uso” (each-use)

Para esta técnica se toma **el máximo número de valores de una variable**, ya que al menos una vez se debe usar cada valor.

La variable con mayor número de valores es:

- **Edad**, con **10 valores**

EACH–USE=10 casos de prueba

c) Cobertura por pares (pair-wise)

Para esta técnica se toman **las dos variables con más valores** y se multiplican.

Nuestras dos variables con más valores son:

- Edad → **10 valores**
- Ingresos → **7 valores**

PAIR–WISE=10×7=70 casos de prueba

c) Cobertura total combinatoria (n-wise, combinación completa)

Aquí se multiplican **todas las cantidades**, generando todas las combinaciones posibles:

N-WISE= 10*7*3*3*4*5*6*3 = 226 800 casos de prueba

6. Defina un conjunto de casos de pruebas para cumplir con each use (cada valor una vez)

Caso	edad	vuelos	estudiante	viveConPadres	clase	destino	ingresos	conNinos
1	17	6	true	true	turista	Europa	19999	false
2	18	3	true	true	turista	Europa	20000	false

3	25	5	false	true	turista	Europa	34999	false
4	26	2	false	false	business	Asia	50000	true
5	0	2	false	true	turista	America	19999	false
6	"" *(vacío)*	-1	*null*	*null*	" "	" "	-1000	" "
7	-1	1000	false	false	200	Africa	1000000	false
8	20	5	true	true	Turista *(case)*	EUROPA *(mayús)*	35000	false
9	30	3	false	false	business	Asia	35000	true
10	2000000	6	false	true	turista	Europa	50000	true

7. Defina conjuntos de pruebas para alcanzar cobertura pairwaise usando el algoritmo explicado en clase. Se pueden comprobar los resultados con el programa PICT5

Tenemos que comprobar los resultados de los 70 casos de prueba, y para ello, tal como se recomienda, hemos utilizado el algoritmo PICT5. El programa https://pragmatic-qa.com/pairwise-testing-with-pict/ no se encuentra disponible, por lo que hemos usado la herramienta online ***Pairwise PICT Online Tool*** (disponible en https://pairwise.yuuniworks.com/), que implementa el mismo algoritmo. A continuación, mostraremos la tabla con los resultados:

Edad	Vuelos	Estudiante	ViveConPadres	Clase	Destino	Ingresos	ConNinos
1	-1	false	""	turista	""	19999	true
17	3	false	false	""	Europa	20000	false
"e"	6	true	true	200	Asia	50000	""
25	"a"	true	""	business	-1000	35000	false
0	-1	""	false	business	America	"o"	""
26	"a"	""	true	200	Europa	34999	true
"e"	""	""	false	turista	-1000	20000	true
17	-1	""	true	200	""	50000	false
17	6	true	""	""	America	"o"	true
0	5	false	true	""	Asia	19999	false
0	2	""	""	turista	Asia	35000	true

-1	3	false	""	business	""	34999	""
26	""	false	false	200	America	19999	""
18	""	true	true	turista	Europa	"o"	false
2000000	2	false	true	""	-1000	20000	""
18	2	false	false	business	America	50000	""
25	2	true	false	turista	Europa	34999	""
0	""	true	false	""	""	34999	false
26	5	true	""	business	-1000	50000	true
17	5	false	false	business	Europa	35000	""
0	"a"	true	""	200	America	20000	""
""	-1	true	true	""	America	35000	false
17	"a"	true	false	business	Asia	19999	false
2000000	-1	true	true	business	Asia	34999	true
2000000	3	""	false	200	-1000	"o"	true
18	-1	""	""	""	-1000	20000	true
2000000	"a"	false	""	turista	Europa	50000	false
-1	6	""	false	turista	Europa	20000	false
25	""	false	""	""	Asia	50000	true
-1	2	true	true	200	-1000	19999	false
25	5	""	true	200	""	"o"	false
17	2	false	true	turista	-1000	"o"	""
1	3	true	true	200	America	35000	false
-1	5	""	false	turista	America	20000	true
0	3	false	false	turista	Asia	50000	true
26	2	""	false	business	""	20000	false
1	""	""	false	business	-1000	34999	""
-1	""	false	true	""	""	35000	false
""	6	false	false	business	Europa	19999	true
25	6	false	""	business	""	20000	true

18	"a"	true	true	""	""	"o"	true
"e"	-1	false	""	business	Europa	34999	false
""	5	""	""	200	America	34999	""
18	6	""	false	200	-1000	35000	false
2000000	""	""	true	""	America	19999	false
26	-1	false	false	turista	Asia	"o"	true
-1	"a"	""	true	200	Asia	"o"	false
25	3	true	true	business	America	19999	false
26	6	false	true	""	-1000	35000	false
-1	-1	""	true	turista	Europa	50000	true
2000000	5	""	false	200	""	35000	true
0	6	""	""	200	-1000	34999	""
1	5	false	""	""	Asia	20000	""
"e"	2	true	true	""	""	"o"	false
17	""	false	false	200	""	34999	false
18	5	true	false	business	Asia	19999	true
18	3	true	false	turista	Asia	34999	""
""	2	true	""	turista	-1000	20000	false
25	-1	false	""	business	Asia	34999	true
""	3	true	false	""	Asia	50000	""
2000000	6	false	""	turista	Asia	35000	""
""	""	false	true	turista	""	"o"	false
"e"	5	""	false	200	America	35000	false
"e"	"a"	""	false	200	Asia	19999	true
1	"a"	true	""	business	Europa	"o"	true
"e"	3	""	""	turista	Asia	"o"	""
""	"a"	false	""	200	""	19999	""
1	6	false	true	business	Europa	50000	false
1	2	""	false	""	Europa	34999	false
26	3	true	""	200	Asia	19999	true

8. Para los trozos de código que incluyan decisiones, proponga conjunto de casos de prueba para alcanzar cobertura de decisiones.

Se analizan todos los fragmentos del código que contienen decisiones (if / else if).
Para cada decisión se proponen casos de prueba que garanticen que la condición global se evalúa al menos una vez a verdadero y una vez a falso.

Decisión 1 – Tarifa *Pajarillo:*

```
if (edad < 18 && vuelosAnuales >= 6)
```

Variables:

- A: edad < 18
- B: vuelosAnuales ≥ 6

A	B	A∧ B	Edad	Vuelos
0	0	0	20	3
0	1	0	20	6
1	0	0	17	3
1	1	1	17	6

Decisión 2 – Tarifa *Gorrión:*

```
else if (edad >= 18 && edad <= 25 && estudiante && clase == "turista")
```

Variables:

- A: edad ≥ 18
- B: edad ≤ 25
- C: estudiante
- D: clase = turista

A	B	C	D	Resuelto	valores
1	1	1	1	TRUE	edad=20, estudiante=true, turista
0	1	1	1	FALSE	edad=17
1	0	1	1	FALSE	edad=26
1	1	0	1	FALSE	estudiante=false
1	1	1	0	FALSE	clase=business

Decisión 3 – Tarifa *Viaja ahora que puedes:*

Variables:

- A: viveConPadres
- B: vuelos ≥ 3
- C: clase = turista

A	B	C	Resultado	Valores

1	1	1	TRUE	true, 5, turista
0	1	1	FALSE	false, 5
1	0	1	FALSE	true, 2
1	1	0	FALSE	business

Decisión 4 – Tarifa *Atreviéndose a saltar del Nido:*

else if (!viveConPadres)

viveConPadres	Resultados
true	FALSE
False	TRUE

Decisión 5 – Tarifas *Conoce Europa:*

```
else if (edad > 25 && ingresos > 20000 && ingresos < 35000 &&
      vuelosAnuales >= 6 && clase == "turista" &&
      destino == "Europa")
```

Caso TRUE:

edad=30, ingresos=30000, vuelos=6, clase=turista, destino=Europa

Caso FALSE:

ingresos=18000 (resto de condiciones verdaderas)

```
else if (edad > 25 && ingresos > 35000 && vuelosAnuales >= 6 &&
clase == "business" &&
(destino == "Asia" || destino == "America"))
```

Caso TRUE:

edad=40, ingresos=50000, vuelos=6, business, Asia

Caso FALSE:

destino=Europa

9. Para los trozos de código que incluyan decisiones, proponga conjunto de casos de prueba para alcanzar cobertura MC/DC.

La cobertura MC/DC garantiza que cada condición básica influye de manera independiente en el resultado de la decisión.

MC/DC – Regla *Pajarillo:*

edad < 18 && vuelosAnuales >= 6

Caso	Edad	Vuelos	Resultados	Condición dominante
1	17	6	TRUE	A Y B

2	18	6	FALSE	A
3	17	5	FALSE	B

MC/DC – Regla *Gorrión:*

edad >= 18 && edad <= 25 && estudiante && clase == "turista"

Caso	Edad	Estudiante	Clase	Resultados	dominante
1	20	True	turista	True	todas
2	17	True	Turista	False	edad≥18
3	26	True	Turista	False	Edad≤25
4	20	False	Turista	False	Estudiante
5	20	true	Business	false	Clase

MC/DC – Decisiones internas por hijos

if (conNinos)

conNinos	resultados
True	Tarifa con peques
False	Tarifa estándar

Conclusión MC/DC:

Con los casos propuestos:

- Todas las condiciones booleanas se evalúan a TRUE y FALSE
- Cada condición se demuestra independiente
- Se alcanza cobertura MC/DC completa con un número reducido de pruebas

10. Comente los resultados del número de los casos de pruebas conseguidos en los apartados 4, 5 y 6 ¿qué podría decirse de la cobertura alcanzada? ¿en qué medida la implementación del programa influye en el diseño e implementación de los casos de pruebas?

Apartado4-combinatoria total (N-wise = 226.800 casos)

Este enfoque proporciona cobertura completa, ya que prueba todas las combinaciones posibles entre las variables. No obstante, el número de casos es inviable en un entorno real, tanto por coste como por tiempo de ejecución.

Apartado 5 – Each-use (10 casos)

La técnica each-use garantiza que cada valor de cada variable aparece al menos una vez. Ofrece una cobertura mínima, adecuada como primer filtrado, pero insuficiente para detectar errores derivados de interacciones entre variables

Apartado 6 – Pair-wise (70 casos)

La cobertura pair-wise supone el mejor equilibrio entre esfuerzo y efectividad, asegurando que todas las combinaciones de pares de variables se prueban al menos una vez. Este enfoque detecta la mayoría de defectos reales con un coste asumible.

Influencia de la implementación en el diseño e implementación de los casos de prueba
La estructura del programa, basada en una cadena de sentencias if–else if, con condiciones compuestas y reglas mutuamente excluyentes, condiciona directamente el diseño de los casos de prueba. En particular, influye en:

- El número de decisiones que deben ser cubiertas durante el proceso de testing.
- La complejidad necesaria para alcanzar coberturas avanzadas, como la cobertura MC/DC, al existir múltiples condiciones lógicas dentro de una misma decisión.
- La necesidad de diseñar cuidadosamente los casos de prueba para evitar solapamientos entre tarifas y asegurar que cada regla de negocio se evalúe de forma independiente.

Una implementación clara, bien estructurada y sin ambigüedades facilita significativamente el diseño de los casos de prueba, mejora la trazabilidad entre requisitos y pruebas, y permite alcanzar altos niveles de cobertura con un número razonable de casos, reduciendo el esfuerzo y el coste del proceso de verificación.

Problema 3

Realizado por Arturo González Rojas y Sergey Ghukasyan Poghosyan

1. **Escribir, al menos el pseudocódigo correspondiente al método o a los métodos identificados.**

```java
package recommender;

public class RecomendadorActividades {

    public String recomendar(Clima clima, EstadoSalud estado, Espacio espacio) {

        if (clima == null || estado == null || espacio == null) {
            throw new IllegalArgumentException(s: "Los parámetros no pueden ser nulos");
        }

        if (!estado.isSano() || estado.isRiesgo()) {
            return "No se recomienda realizar ninguna actividad.";
        }

        int temperatura = clima.getTemperatura();
        int humedad = clima.getHumedad();
        boolean lluvia = clima.hayPrecipitacion();

        if (temperatura < 0) {
            if (lluvia) {
                return "Quédese en casa.";
            }
            if (humedad < 15 && espacio.plazasDisponibles() > 0) {
                return "Actividades de esquí.";
            }
            if (espacio.plazasDisponibles() <= 0) {
                return "Aforo completo, no se puede realizar esquí.";
            }
            return "No hay recomendación disponible.";
        }

        if (temperatura <= 15 && !lluvia) {
            return "Senderismo o escalada.";
        }

        if (temperatura >= 30) {
            if (espacio.plazasDisponibles() > 0) {
                return "Ir a la playa o piscina.";
            }
            return "La piscina está en aforo completo.";
        }

        if (!lluvia && humedad <= 60 && temperatura > 25) {
            return "Actividades culturales o gastronómicas.";
        }

        if (!lluvia && humedad <= 60) {
            return "Actividades de catálogo primavera/verano/otoño.";
        }


        return "No hay recomendación disponible.";
    }
}
```

2. Identificar las unidades de prueba.

La unidad de prueba principal es el método RecomendadorActividades.recomendar(...). Las clases que dependen de él (objetos Clima, EstadoSalud, Espacio) son las que van a ser usadas como prueba para verificar la lógica de la clase recomendador.

3. Identificar las variables que se deben tener en cuenta para probar los métodos de interés incluidos en las unidades de prueba.

Las variables independientes del sistema son:

- **Temperatura (double) medida en °C.**
- **Humedad (int) medida en %.**
- **Precipitación (boolean).**
- **Estado de Salud (boolean).**
- **Aforo (int) tanto capacidad como ocupación.**

4. Identificar los valores de pruebas para cada una de las variables anteriores que conformarán los correspondientes casos de prueba usando las tres técnicas vistas en teoría (clases de equivalencia, teoría de los valores límites y conjetura de errores), especificando para cada valor obtenido cuál es la que ha sido usada.

Parámetros	Clases de equivalencia	Valores seleccionados (Límite/Clase)	Conjetura de errores	Valores totales
Temperatura	(-Inf, 0), [0, 15], (15, 25], (25, 30), [30, +Inf)	-1, 0, 15, 20, 26, 30	-274, String	6
Humedad	[0, 15), [15, 60], (60, 100]	14, 15, 60, 61	-1, 101	4
Lluvia	True, False	True, False	null	2

Parámetros	Clases de equivalencia	Valores seleccionados (Límite/Clase)	Conjetura de errores	Valores totales
Salud	Sano/Sin Riesgo, No Sano o Riesgo	{T, F}, {F, T}	null	2
Plazas	<= 0, > 0	0, 1	-1	2
Argumentos	Objetos válidos	Instancias válidas	null	2

5. Calcular el número máximo posible de casos de pruebas que se podrían generar a partir de los valores de pruebas (combinatoria).

• Combinatoria **Total (N-Wise):** Multiplicando los rangos teóricos: 6 (temp) * 4 (hum) * 2 (lluvia) * 2 (salud) * 2 (plazas) = **192 combinaciones teóricas** (sin contar argumentos nulos).

• Cobertura **"Cada uso" (Each-Use):** Determinada por la variable con mayor número de clases de equivalencia: **Temperatura (6 valores)**. Se requieren al menos 6 casos base.

• Cobertura **por pares (Pair-Wise):** Combinando las dos variables con más valores (Temp y Humedad): 6 * 4 = **24 casos de prueba** base (aproximación inicial).

6. Defina un conjunto de casos de pruebas para cumplir con each use (cada valor una vez).

Caso	Temp	Humedad	Lluvia	Salud (Sano/Riesgo)	Plazas	Resultado Esperado
1	-1	14	False	T / F	1	Actividades de esquí
2	0	15	True	T / F	0	Quédese en casa / No hay recomendación
3	15	60	False	T / F	1	Senderismo o escalada
4	26	61	False	F / T	1	No se recomienda (Salud)
5	30	14	False	T / F	1	Playa o piscina
6	20	61	True	T / F	0	No hay recomendación

7. Defina conjuntos de pruebas para alcanzar cobertura pairwaise usando el algoritmo explicado en clase. Se pueden comprobar los resultados con el programa PICT5.

Para comprobar todos los casos de prueba siguiendo el algoritmo PICT5 se ha usado la extensión de VSCode
https://marketplace.visualstudio.com/items?itemName=EXCEEDSYSTEM.vscode-pict

1. Cada rango de temperatura se ha cruzado con cada estado de lluvia y plazas.
2. Los casos de salud están distribuidos para probar que bloquean las actividades del clima.

ID Caso	Temp (°C)	Humedad (%)	Lluvia	Salud	Plazas (Aforo)	Resultado Esperado
1	-1	14	False	Sano	1	Actividades de esquí
2	-1	70	True	Sano	0	Quédese en casa
3	-1	50	True	Riesgo	1	No se recomienda (Salud)
4	-1	14	False	Sano	0	Aforo completo, no esquí
5	10	14	True	Sano	1	No hay recomendación
6	10	50	False	Sano	0	Senderismo o escalada
7	10	70	False	Riesgo	1	No se recomienda (Salud)
8	10	14	True	Riesgo	0	No se recomienda (Salud)
9	20	50	False	Sano	1	Catálogo primavera/verano
10	20	70	True	Sano	0	No hay recomendación
11	20	14	True	Riesgo	1	No se recomienda (Salud)
12	20	14	False	Sano	1	Catálogo primavera/verano
13	26	50	False	Sano	1	Actividades culturales/gastronómicas
14	26	14	False	Riesgo	0	No se recomienda (Salud)
15	26	70	True	Sano	1	No hay recomendación
16	26	50	True	Sano	0	No hay recomendación
17	30	14	False	Sano	1	Ir a la playa o piscina

ID Caso	Temp (°C)	Humedad (%)	Lluvia	Salud	Plazas (Aforo)	Resultado Esperado
18	30	50	True	Sano	0	La piscina está en aforo completo
19	30	70	False	Riesgo	1	No se recomienda (Salud)
20	30	14	True	Sano	0	La piscina está en aforo completo
21	-1	50	False	Riesgo	0	No se recomienda (Salud)
22	30	70	True	Sano	1	Ir a la playa o piscina (Ignora lluvia)

8. Para los trozos de código que incluyan decisiones, proponga conjunto de casos de prueba para alcanzar cobertura de decisiones.

Debido a que muchas decisiones son anidadas, es necesario diseñar casos que fuercen la evaluación False de las condiciones superiores para llegar a las inferiores.

Caso	Valores Entrada	Decisión Objetivo	Evaluación (T/F)	Resultado
1	clima=null	D1	True	Excepción
2	Sano=false	D2	True	No actividad
3	Sano=true, Temp=-5	D3	True	Entra bloque frío

Caso	Valores Entrada	Decisión Objetivo	Evaluación (T/F)	Resultado
4	Temp=-5, Lluvia=true	D4	True	Quédese en casa
5	Temp=-5, Lluvia=false, Hum=10, Plazas=5	D5	True	Esquí
6	Temp=-5, Lluvia=false, Hum=20	D5	False	Evalúa siguiente
7	Temp=-5, ..., Plazas=0	D6	True	Aforo completo
8	Temp=10	D7	True	Entra bloque moderado
9	Temp=10, Lluvia=false	D8	True	Senderismo
10	Temp=35	D9	True	Entra bloque calor
11	Temp=35, Plazas=5	D10	True	Playa

9. Para los trozos de código que incluyan decisiones, proponga conjunto de casos de prueba para alcanzar cobertura MC/DC.

Para seleccionar los casos de prueba para alcanzar cobertura MC/DC, se hace una tabla de verdad y se escogen los casos donde la variación de una sola constante y manteniendo las demás constantes fijas, el resultado cambia.

Caso	A (!Lluvia)	B (Hum<=60)	C (Temp>25)	Resultado (A y B y C)	Independencia Demostrada
1	**True**	**True**	**True**	**True**	Base (Cultural)
2	**False**	True	True	**False**	A (vs 1)
3	True	**False**	True	**False**	B (vs 1)
4	True	True	**False**	**False**	C (vs 1)

- El caso 1 (T,T,T) es el caso base donde se cumple la condición.

- El caso 2 varía solo A (Lluvia=True), cambiando el resultado a False. → A es independiente.

- El caso 3 varía solo B (Humedad=70), cambiando el resultado a False. → B es independiente.

- El caso 4 varía solo C (Temp=20), cambiando el resultado a False. → C es independiente.

10. Comente los resultados del número de los casos de pruebas conseguidos en los apartados 4, 5 y 6 ¿qué podría decirse de la cobertura alcanzada? ¿en qué medida la implementación del programa influye en el diseño e implementación de los casos de pruebas?

La cobertura **Each-Use** no basta en este caso porque no explora las combinaciones específicas de cada bloque.

La aplicación de **MC/DC** en condiciones con varias variables como la de actividades culturales o gastronómicas es necesario para saber que las variables como la humedad o la temperatura afectan al resultado y no sean ocultadas por otras.

La cobertura **Pair-wise** garantiza que todas las combinaciones posibles de dos variables se prueben al menos una vez. Es bastante efectivo en este caso.

La cobertura **N-wise** implica probar todas las combinaciones posibles para asegurarse 0 errores. No creemos que sea necesario para este sistema ni el más eficiente.

Existe un tipo de prioridad en las variables. Por ejemplo, la variable de salud restringe a las otras variables si el estado de salud se encuentra en riesgo, las demás no importan. Además, algunos casos tienen hasta 3 condiciones. Por eso, hay que asegurarse que las 3 funcionan por separado y son necesarios para el caso, y no sean solo añadidos a una sola condición que ya cumpla el caso.