

Resumen Caja Negra

El elemento a probar se considera una caja Negra. Los casos de prueba son independientes de la implementación. Podemos detectar comportamientos NO implementados.

Los métodos basados en Especificación:

1. Analizan la especificación y partitionan el conjunto S.
2. Seleccionan un conjunto de comportamientos bajo algún criterio
3. Obtiene un conjunto de casos de prueba.

- Se trata de un proceso sistemático que identifica, a partir de la especificación disponible, un conjunto de clases de equivalencia para cada una de las entradas y salidas del elemento a probar.
- Cada clase de equivalencia o partición de entrada representa un subconjunto del total de datos posibles de entrada que tienen un mismo comportamiento.
- Cada caso de prueba usará un subconjunto de particiones, no se trata de probar todas las combinaciones posibles

Sistematicidad y Particionamiento:

- Las particiones representan un conjunto de posibles comportamientos del sistema.
- Se deben elegir muestras significativas de cada partición
- Asegurarnos de que cubrimos cada partición

¿Cómo identificamos una partición?

- Las particiones (o clases de equivalencia) se identifican en base a condiciones de entrada/salida de la unidad a probar.
- Una condición de entrada/salida puede aplicarse a una única variable de entrada/salida en una especificación o con un subconjunto de ellas.
- Las variables de entrada/salida no necesariamente se corresponden con parámetros de entrada/salida de la unidad a probar.
- Las particiones deben de ser disjuntas (las particiones no comparten elementos)
- Todos los miembros de una partición de entrada deben de tener su imagen en la misma partición de salida.

P
Particiones Validas e Invalidas

- Las clases de equivalencia pueden clasificarse en validas e invalidas
- Las particiones de entrada invalidas normalmente tienen asociadas clases invalidas
- Solo puede haber una partición invalida de entrada en un caso de prueba.

I

Identificación de clases de equivalencia:

- Si la E/S es un rango de valores válidos, definiremos una clase válida (dentro del rango) y dos inválidas (fuera de cada uno de los extremos del rango).
- Si la E/S especifica un número N de valores válidos, definiremos una clase válida (número de valores entre 1 y N) y dos inválidas (ningún valor, más de N valores).
- Si la E/S especifica un conjunto de valores válidos, definiremos una clase válida (valores pertenecientes al conjunto) y una inválida (valores que no pertenecen al conjunto).
- Si por alguna razón, se piensa que cada uno de los valores de entrada se van a tratar de forma diferente por el programa, entonces definir una clase válida para cada valor de entrada.
- Si la E/S especifica en una situación Debe Ser, definiremos una clase válida y una inválida.
Ej X comienza por un número, Clase válida X empieza por el número, Clase inválida X no empieza por el número.
- Si por alguna razón, se piensa que los elementos de una partición van a ser tratados de forma distinta, subdividir la partición en particiones más pequeñas.

Identificación de los casos de prueba:

Debemos asignar un identificador único por cada partición

- Hasta que todas las clases válidas estén cubiertas (probadas) debemos escribir un nuevo caso de prueba que cubra el número máximo de clases válidas todavía no cubiertas
- Hasta que todas las clases inválidas estén cubiertas (probadas) escribir un nuevo caso de prueba que cubra una y solo una clase inválida, todavía no cubierta.
- Elegir un valor concreto para cada partición

El resultado de este proceso será una tabla con tantas filas como casos de prueba hayamos obtenido

▷ 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 día	100 euros
2 días o más	50 euros/día

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

Entradas identificadas:

- **TipoCoche** es un enumerado por lo que sea única entrada no válida es el null
- **FechaInicio**: debe de ser posterior a la actual
- **Cantidad días reserva**: entradas válidas 1, 2 a 30, y invalidas < 1 null, > 30
- **Disponibilidad coche**: puede estar o no disponible

TipoCoche:

- EV1 : Turismo
- EV2 : Deportivo
- ENV1 : Null

FechaInicio:

- EV3 : Fecha Actual
- EV4 : Fecha Posterior
- ENV2 : Fecha Anterior actual
- ENV3 : Null

Días Reserva:

- EV5 = 1
- EV6 = 2 .. 30
- ENV4 = > 30
- ENV5 = < 1

Disponibilidad

- EV7 : Disponible
- ENV6 : No Disponible

Salidas:

- SV1 : Importe
- SV2 : Importe * N dias
- SNV1 : Fecha no correcta → ENV2
- SNV2 : Reserva no posible → ENV6, ENV4
- SNV3 : ??? → ENV1, ENV3, ENV5

* Asumimos que la fecha actual 17/03/2023

ID	Clase	TipoCoche	FechaInicio	Días Reserva	Disponibilidad	Salida Esperada
C1	EV1-EV3-EV5-EV7-SNV1	Turismo	17/03/2023	1	Disponible	100
C2	EV2-EV4-EV6-EV7-SNV2	Deportivo	18/03/2023	3	Disponible	150
C3	ENV1-EV3-EV5-EV7-SNV3	Null	17/03/2023	1	Disponible	???
C4	EV1-ENV2-EV5-EV7-SNV1	Turismo	16/03/2023	1	Disponible	Fecha No Correcta
C5	EV1-ENV3-EV5-EV7-SNV3	Turismo	Null	1	Disponible	???
C6	EV1-EV3-ENV4-EV7-SNV2	Turismo	Null	32	Disponible	Reserva No Posible
C7	EV1-EV3-ENV5-EV7-SNV3	Turismo	17/03/2023	0	Disponible	???
C8	EV1-EV3-EV5-ENV6-SNV2	Turismo	17/03/2023	1	No Disponible	Reserva No Posible

▷ 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>.<extensión>"

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.

• NombreAsignatura

- EV1: String

• FechaInicio, FechaFin, diaSemana

- EV4: FechaFin > FechaInicio y diaSemana >= 1 y <= 6
- ENV4: FechaInicio > FechaFin
- ENV5: FechaInicio < FechaFin
diaSemana : no dentro
- ENV6 : diaSemana > 6
- ENV7 : diaSemana < 1

• HoraInicio Clase

- EV2 = 00:00 a 24:00
- EV3 = null
- ENV1: formatoIncorrecto
- ENV2: < 00:00
- ENV3: > 24:00

• Salidas

- SV1 : Lista
- SV2 : Lista Vacía
- SW1 : ParseException

ID	Clase	NombreAsignatura	FechaInicio	FechaFin	diaSemana	HoraInicio	Salida Esperada
C-1	EV1-EV4-ENV2	"ppss"	22/04/23	26/04/23	1	22:00	[{"ppss", "22/04/23", "22:00", "10}])
C-2	EV1-ENV1-ENV2	"ppss"	22/04/23	26/04/23	1	null	[{"ppss", "22/04/23", "2200", "-1}])
C-3	EV1-ENV4-EU ENV2	"ppss"	21/04/23	22/04/23	1	22:00	{}
C-4	EV1-ENV5-EU-ENV1	"ppss"	22/04/23	23/04/23	3	22:00	{}
C-5	EV1-ENV6-ENV1	"ppss"	22/04/23	23/04/23	8	22:00	ParseException
C-6	EV1-ENV7-ENV1	"ppss"	22/04/23	26/04/23	1	null	ParseException
C-7	EV1-ENV4-ENV1 ENV1	"ppss"	22/04/23	26/04/23	1	22:03:03	ParseException
C-8	EV1-ENV1-ENV2	"ppss"	22/04/23	26/04/23	1	-00:11	ParseException
C-9	EV1-ENV1-ENV2	"ppss"	22/04/23	26/04/23	1	25:00	ParseException

Entradas:

- Nombre asignatura
- Fecha Inicio Curso
- Fecha Fin Curso
- Dia Semana.
- horaInicio Clase

Agrupación ya que el resultado final depende de las 3

Ejercicio 3: especificación enviarMensaje()

Crea la subcarpeta "enviarMensaje", 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 "enviarMensaje-<sufijo>.<extensión>"

En una aplicación de Juegos a través de internet, tenemos una clase Proxy que se encarga de la comunicación de los juegos entre los clientes y un servidor. Queremos probar el método `Proxy.enviarMensaje()`, cuyo prototipo es el siguiente:

```
void enviaMensaje(Mensaje msg, int idJuego, String urlServidor)
                    throws MensajeException,
                           JuegoInvalidoException;
```

El tipo Mensaje tiene 3 campos: tipo, destinatario y datos.

El tipo puede ser TipoMensaje.DATOS, TipoMensaje.BROADCAST o TipoMensaje.INFO.

Tanto en los mensajes de tipo DATOS como de tipo BROADCAST es obligatorio proporcionar en el campo **datos** un array de bytes diferente de null, con los datos que se le vayan a enviar a uno de los jugadores en el primer caso, o a todos ellos en el segundo. De no ser así se produciría una excepción de tipo **MensajeException**.

De la misma forma, en el caso del tipo DATOS será obligatorio indicar el **destinatario** del mensaje (que es una cadena con el identificador del jugador al que va dirigido), y si no se hace, o el jugador indicado no se encuentra dentro de la partida, también se lanzará un **MensajeException**.

El tipo INFO sólo comunica con el servidor para comprobar el estado del juego (si el juego no está activo en el servidor, se devolverá la excepción de tipo **JuegoInvalidoException**).

El método no envía el mensaje si se produce alguna excepción, o si el tipo de mensaje no es INFO y el juego no está activo. Envíará el mensaje en cualquier otro caso, devolviendo el valor true.

Diseña los casos de prueba para la especificación anterior utilizando el método de particiones equivalentes.

NOTA: Observa que este método envía el mensaje al servidor (en determinados casos). Por lo tanto, "Envía mensaje" es una salida de nuestra SUT. Pero NO es una salida "directa" y (de momento) no sabrás implementar los tests que automatizan la tabla de diseño obtenida (ya que no todavía no sabemos cómo comprobar que efectivamente el mensaje se ha enviado). Lo veremos más adelante.

• TipoMensaje + Datos + Jugador

- EV1: Mensaje: Broadcast
Datos: bytes
Destinatario: null

- ENV1 Mensaje: Broadcast
Datos: null
Destinatario: null

- EV2: Mensaje: Datos
Datos: bytes
Destinatario: Número Real

- ENV2 Mensaje: Datos
Datos: null
Destinatario: número Real

- EV3: Mensaje: INFO
Datos: null
Destinatario: null

- ENV3: Mensaje: INFO
Datos: bytes
Destinatario: null

- ENV4: Mensaje: INFO
Datos: bytes
Destinatario: Real

• IDJuego

- EV5: número
ENV5: juego no existe

Suponemos: Un juego con id=1 activo en juego.com Con 1 jugador de id=1 activo y otro de id=2 inactivo y juego id=2 inactivo

ID	Clase	Tipo Mensaje	Datos	Destinatario	IDJuego	URLServidor	Salida
C1		Broadcast	"bytes"	null	1	juego.com	True
C2		Datos	"bytes"	1	1	juego.com	True
C3		INFO	null	Null	1	juego.com	True
C4	ENV1	Broadcast	null	null	1	juego.com	MensajeException
C5	ENV2	Datos	null	1	1	juego.com	MensajeException
C6	ENV3	Datos	"bytes"	null	1	juego.com	MensajeException
C7	ENV4	INFO	"bytes"	1	1	juego.com	???
C8	ENV7		null		1	juego.com	???

Entradas:

- Mensaje → 3 tipos, null

- Datos : array bytes, si es null MensajeException

Jugador : identificador

- Disponibilidad → Si no esta MensajeException

Juego → ID

- Estado → Si no esta JuegoInvalidoException

URL Servidor:

• URL Servidor :

- EV4 : String empieza https
ENV5: null

- ENV6 : String no empieza https

C9	ENVS	Broadcast "bytes"	null	4	juego.com	JuegoInvalidoException
C10	ENV5	Broadcast "bytes"	null	1	null	???
C11	ENV6	Broadcast "bytes"	null	1	j	???
.						