

Reserva_Vuelo — TDA de reservas de butacas de un vuelo

Una aerolínea gestiona las reservas de asientos: avión con filas fijas y butacas fijas por fila; cada pasajero (DNI) ocupa a lo sumo una butaca.

Validación: 0 errores, 4 warnings. Los 4 son falsos positivos de V1 sobre operaciones definidas extensionalmente con LHS de variable libre (cantidadButacas, butacaValida, estaLleno, butacasLibres) — el mismo patrón tolerado del resto del workspace.

Diseño algebraico: dos constructores básicos y la funcionalidad pedida como operaciones derivadas.

- **inicializar(filas, butacasPorFila)** — vuelo sin reservas, con las dimensiones del avión fijadas.
- **agregar(r, fila, butaca, dni)** — reserva la butaca (fila, butaca) para el pasajero dni.
- Operaciones derivadas: eliminarReserva, eliminarTodas, estaLleno, pasajeroEn, butacasLibres, listadoPasajeros.

Toda reserva se representa como `agregar(agregar(...agregar(inicializar(F,B), ...)...))`. Filas y butacas se indexan desde 1.

Decisión clave — SIN sobrescritura: la precondition de agregar exige que la butaca esté *libre* y que el pasajero *no tenga ya una reserva* ($\text{not butacaOcupada} \wedge \text{not tieneReserva}$). Así se mantiene la biyección parcial $\text{butaca} \leftrightarrow \text{pasajero}$, y cantidadReservas, butacasLibres y listadoPasajeros se definen por recursión simple — cada agregar aporta una reserva nueva y distinta.

Tipos: Integer en lugar de Natural1. Las filas, butacas, DNIs y cantidades son conceptualmente naturales (≥ 0), pero el analyzer tipa los literales numéricos 0 y 1 como Integer y no los coacciona a Natural: declarar $\text{cantidadReservas} \rightarrow \text{Natural}$ con axioma $\text{base} = 0$ produce un type mismatch. Se usa Integer de forma uniforme; la no-negatividad y los rangos válidos quedan documentados por las precondiciones ($f \geq 1$, butacaValida, etc.). Por eso `IMPORTS Integer, Sequence[Integer]`.

1. Especificación

```
CLASS Reserva_Vuelo
IMPORTS Integer, Sequence[Integer]
BASIC CONSTRUCTORS inicializar, agregar
EFFECTIVE
TYPE Reserva_Vuelo

OPERATIONS

-- — Constructores básicos —————

-- inicializar(filas, butacasPorFila): vuelo sin reservas.
inicializar: Integer * Integer -> Reserva_Vuelo;
```

```

-- agregar(r, fila, butaca, dni): reserva (fila, butaca) para
-- el pasajero dni. Precondición: La butaca es válida y está
-- libre, y el pasajero no tiene todavía una reserva.
agregar: Reserva_Vuelo(r) * Integer(fila) * Integer(butaca)
      * Integer(dni) -> Reserva_Vuelo
  pre: butacaValida(r, fila, butaca)
      and not butacaOcupada(r, fila, butaca)
      and not tieneReserva(r, dni);

-- ——— Dimensiones del avión ———
cantidadFilas:  Reserva_Vuelo -> Integer;
butacasPorFila:  Reserva_Vuelo -> Integer;
cantidadButacas: Reserva_Vuelo -> Integer;  -- filas * butacasPorFila

-- butacaValida(r, f, b): ¿(f, b) cae dentro del avión?
butacaValida: Reserva_Vuelo * Integer * Integer -> Boolean;

-- ——— Consultas sobre el estado de las reservas ———

-- butacaOcupada(r, f, b): ¿la butaca (f, b) está reservada?
butacaOcupada: Reserva_Vuelo * Integer * Integer -> Boolean;

-- tieneReserva(r, dni): ¿el pasajero dni tiene una reserva?
tieneReserva: Reserva_Vuelo * Integer -> Boolean;

-- cantidadReservas(r): cuántas butacas están reservadas.
cantidadReservas: Reserva_Vuelo -> Integer;

-- estaLleno(r): ¿no quedan butacas libres?
estaLleno: Reserva_Vuelo -> Boolean;

-- butacasLibres(r): cantidad de butacas sin reservar.
butacasLibres: Reserva_Vuelo -> Integer;

-- pasajeroEn(r, f, b): DNI del pasajero que ocupa (f, b).
pasajeroEn: Reserva_Vuelo(r) * Integer(f) * Integer(b) -> Integer
  pre: butacaOcupada(r, f, b);

-- ListadoPasajeros(r): Sequence con Los DNIs reservados.
listadoPasajeros: Reserva_Vuelo -> Sequence[Integer];

-- ——— Eliminación de reservas ———

-- eliminarReserva(r, dni): cancela la reserva del pasajero dni.
eliminarReserva: Reserva_Vuelo(r) * Integer(dni) -> Reserva_Vuelo
  pre: tieneReserva(r, dni);

-- eliminarTodas(r): cancela todas las reservas; conserva el
-- avión (es decir, sus dimensiones).
eliminarTodas: Reserva_Vuelo -> Reserva_Vuelo;

```

AXIOMS r: Reserva_Vuelo;
 f, f1, b, b1, cf, bpf, dni, dni1: Integer;

```

-- Dimensiones: fijadas por inicializar, invariantes ante agregar.
cantidadFilas(inicializar(cf, bpf))  = cf;
cantidadFilas(agregar(r, f, b, dni)) = cantidadFilas(r);

butacasPorFila(inicializar(cf, bpf)) = bpf;
butacasPorFila(agregar(r, f, b, dni)) = butacasPorFila(r);

```

```

cantidadButacas(r) = cantidadFilas(r) * butacasPorFila(r);

butacaValida(r, f, b) =
  (f >= 1) and (f <= cantidadFilas(r))
  and (b >= 1) and (b <= butacasPorFila(r));

-- butacaOcupada: ¿La butaca (f, b) está reservada?
butacaOcupada(inicializar(cf, bpf), f, b) = False;
((f1 = f) and (b1 = b)) implies
  butacaOcupada(agregar(r, f1, b1, dni), f, b) = True;
not ((f1 = f) and (b1 = b)) implies
  butacaOcupada(agregar(r, f1, b1, dni), f, b) = butacaOcupada(r, f, b);

-- tieneReserva: ¿el pasajero dni ya tiene una reserva?
tieneReserva(inicializar(cf, bpf), dni) = False;
(dni1 = dni) implies
  tieneReserva(agregar(r, f, b, dni1), dni) = True;
not (dni1 = dni) implies
  tieneReserva(agregar(r, f, b, dni1), dni) = tieneReserva(r, dni);

-- cantidadReservas: número de butacas reservadas.
cantidadReservas(inicializar(cf, bpf)) = 0;
cantidadReservas(agregar(r, f, b, dni)) = 1 + cantidadReservas(r);

-- butacasLibres y estaLleno (derivadas de Las cantidades).
butacasLibres(r) = cantidadButacas(r) - cantidadReservas(r);
estaLleno(r) = (cantidadReservas(r) = cantidadButacas(r));

-- pasajeroEn: DNI del pasajero en la butaca (f, b).
-- La precondition butacaOcupada(r, f, b) excluye inicializar.
((f1 = f) and (b1 = b)) implies
  pasajeroEn(agregar(r, f1, b1, dni), f, b) = dni;
not ((f1 = f) and (b1 = b)) implies
  pasajeroEn(agregar(r, f1, b1, dni), f, b) = pasajeroEn(r, f, b);

-- ListadoPasajeros: Sequence con Los DNIs reservados.
listadoPasajeros(inicializar(cf, bpf)) = createSequence();
listadoPasajeros(agregar(r, f, b, dni)) =
  appendSequence(listadoPasajeros(r), dni);

-- eliminarReserva(r, dni): cancela la reserva del pasajero dni.
-- La precondition tieneReserva(r, dni) excluye inicializar.
(dni1 = dni) implies
  eliminarReserva(agregar(r, f, b, dni1), dni) = r;
not (dni1 = dni) implies
  eliminarReserva(agregar(r, f, b, dni1), dni) =
    agregar(eliminarReserva(r, dni), f, b, dni1);

-- eliminarTodas: descarta todas las reservas, conserva las
-- dimensiones del avión (recursión que reduce a inicializar).
eliminarTodas(inicializar(cf, bpf)) = inicializar(cf, bpf);
eliminarTodas(agregar(r, f, b, dni)) = eliminarTodas(r);

```

END-CLASS

2. Operaciones

Operación	Tipo	Axiomas
-----------	------	---------

inicializar	constructor básico	—
agregar	constructor básico <i>pre: butaca válida</i> <i>∧ libre ∧ pasajero sin reserva</i>	—
cantidadFilas / butacasPorFila	derivada	2 c/u — base sobre inicializar + invariante ante agregar
cantidadButacas	derivada	1 — = filas × butacasPorFila
butacaValida	auxiliar	1 — definición extensional vía las dimensiones
butacaOcupada	derivada	3 — inicializar → False + 2 implies sobre agregar
tieneReserva	derivada	3 — inicializar → False + 2 implies sobre agregar
cantidadReservas	derivada	2 — inicializar → 0, agregar → 1 + ...
estalleno	derivada	1 — = (cantidadReservas = cantidadButacas)
butacasLibres	derivada	1 — = cantidadButacas - cantidadReservas
pasajeroEn	derivada <i>pre: butacaOcupada(r,f,b)</i>	2 — implies sobre agregar según coincida la butaca
listadoPasajeros	derivada	2 — inicializar → createSequence(), agregar → appendSequence(...)
eliminarReserva	derivada <i>pre: tieneReserva(r,dni)</i>	2 — implies sobre agregar según coincida el DNI
eliminarTodas	derivada	2 — recursión que reduce a inicializar

3. Ejemplo: avión 3 filas × 4 butacas

Construimos un vuelo y reservamos tres butacas:

```
r = agregar(agregar(agregar(inicializar(3, 4),
    1, 1, 30111222),    -- pasajero 30111222 en (1,1)
    1, 2, 28999888),    -- pasajero 28999888 en (1,2)
    2, 3, 31555444)    -- pasajero 31555444 en (2,3)
```

	but1	but2	but3	but4
fila 1	30111222	28999888	.	.
fila 2	.	.	31555444	.
fila 3	.	.	.	.

(• = butaca libre)

Evaluación de las operaciones

Expresión	Resultado	Por qué
cantidadButacas(r)	12	3×4 — el avión tiene 12 butacas.
cantidadReservas(r)	3	Tres agregar aplicados.
butacasLibres(r)	9	$12 - 3$.
estaLleno(r)	False	$3 \neq 12$.
butacaOcupada(r, 1, 2)	True	Se reservó (1, 2).
butacaOcupada(r, 3, 4)	False	Ningún agregar tocó (3, 4).
pasajeroEn(r, 2, 3)	31555444	El agregar de (2, 3) coincide.
tieneReserva(r, 28999888)	True	Reservó la butaca (1, 2).
listadoPasajeros(r)	[30111222, 28999888, 31555444]	Un DNI por reserva, en orden de inserción.
eliminarReserva(r, 28999888)	vuelo sin la reserva de (1, 2)	Se descarta el agregar de ese DNI; los otros dos se conservan.
eliminarTodas(r)	inicializar(3, 4)	Recursión que descarta los tres agregar; conserva las dimensiones.

4. Propiedades garantizadas

Sin sobrescritura $\Rightarrow$ conteo único: como agregar exige not butacaOcupada y not tieneReserva, cada butaca y cada DNI aparecen a lo sumo una vez en la cadena de agregar. Por eso cantidadReservas cuenta exactamente las reservas, listadoPasajeros no repite DNIs, y eliminarReserva encuentra y descarta una única reserva por pasajero.

Biyección parcial butaca $\leftrightarrow$ pasajero: la doble precondition de agregar garantiza que ninguna butaca tenga dos pasajeros y ningún pasajero tenga dos butacas. pasajeroEn (de butaca a DNI) y tieneReserva (de DNI a presencia) observan esa relación desde los dos lados.

Parcialidad bien delimitada: pasajeroEn exige butacaOcupada(r, f, b) — no tiene sentido preguntar quién ocupa una butaca libre; por eso no tiene axioma para inicializar. eliminarReserva exige tieneReserva(r, dni) — no se puede cancelar una reserva inexistente. Las preconditiones documentan el dominio exacto de cada operación.

Dimensiones inmutables: cantidadFilas y butacasPorFila quedan fijadas por inicializar y son invariantes ante agregar ($\text{cantidadFilas}(\text{agregar}(r, \dots)) = \text{cantidadFilas}(r)$). El avión nunca cambia de tamaño; incluso eliminarTodas conserva las dimensiones.

Spec validada con el analyzer del workspace: 0 errores, 4 warnings (V1 false positives sobre cantidadButacas, butacaValida, estaLleno y butacasLibres, definidas extensionalmente). Importa Integer y Sequence[Integer]; Boolean se asume importado implícitamente.