



Ejercicio 1. Tenemos un TAD que modela las ventas minoristas de un comercio. Cada venta es individual (una unidad de un producto) y se quieren registrar todas las ventas. El TAD tiene un único observador:

```
Producto ES string
Monto ES  $\mathbb{Z}$ 
Fecha ES  $\mathbb{Z}$  – segundos desde 1/1/1970

TAD Comercio {
  obs ventasPorProducto :  $\text{dicc}\langle \text{Producto}, \text{seq}\langle \text{tupla}\langle \text{Fecha}, \text{Monto} \rangle \rangle \rangle$ 
}
```

ventasPorProducto contiene, para cada producto, una secuencia con todas las ventas que se hicieron de ese producto. Para cada venta, se registra la fecha y el precio. Se puede considerar que todas las fechas son diferentes. Este TAD lo vamos a implementar con la siguiente estructura:

```
Módulo ComercioImpl implementa Comercio <
  var ventas : Vector<tupla<Producto, Fecha, Monto>>
  var totalPorProducto : Diccionario<Producto, Monto>
  var ultimoPrecio : Diccionario<Producto, Monto>
>
```

- **ventas** es una implementación de secuencia con todas las ventas realizadas, indicando producto, fecha y monto.
- **totalPorProducto** asocia cada producto con el dinero total obtenido por todas sus ventas.
- **ultimoPrecio** asocia cada producto con el monto de su última venta registrada.

- Escribir en castellano, con precisión y detalle el invariante de representación y la función de abstracción.
- Escribir ambos en el lenguaje de especificación.

Ejercicio 2. Considere la siguiente especificación de una relación uno/muchos entre alarmas y sensores de una planta industrial: un sensor puede estar asociado a muchas alarmas, y una alarma puede tener muchos sensores asociados.

```
TAD Planta {
  obs alarmas :  $\text{conj}\langle \text{Alarma} \rangle$ 
  obs sensores :  $\text{conj}\langle \text{tupla}\langle \text{Sensor} \rangle \text{Alarma} \rangle$ 
  proc nuevaPlanta() : Planta {
    asegura {  $\text{res.alarmas} = \langle \rangle \wedge \text{res.sensores} = \langle \rangle$  }
  }

  proc agregarAlarma(inout p : Planta, in a : Alarma) {
    requiere {  $p = P_0 \wedge a \notin p.alarmas$  }
    asegura {  $p.alarmas = P_0.alarmas \cup \langle a \rangle \wedge p.sensores = P_0.sensores$  }
  }

  proc agregarSensor(inout p : Planta, in a : Alarma, in s : Sensor) {
    requiere {  $p = P_0 \wedge a \in p.alarmas \wedge \langle s, a \rangle \notin p.sensores$  }
    asegura {  $p.alarmas = P_0.alarmas \wedge p.sensores = P_0.sensores + \langle \langle s, a \rangle \rangle$  }
  }
}
```

Se decidió utilizar la siguiente estructura como representación, que permite consultar fácilmente tanto en una dirección (sensores de una alarma) como en la contraria (alarmas de un sensor).

Módulo PlantaImpl **implementa** Planta <

```
var alarmas: Diccionario<Alarma, Conjunto<Sensor>>
var sensores: Diccionario<Sensor, Conjunto<Alarma>>
```

>

- Escribir en castellano, con precisión y detalle el invariante de representación y la función de abstracción.
- Escribir ambos en el lenguaje de especificación.

Ejercicio 3. Dado el siguiente TAD:

Estudiante ES \mathbb{Z}

TAD Secundario {

obs estudiantes : conj(*Estudiante*)

obs faltas : dicc(*Estudiante*, \mathbb{Z})

obs notas : dicc(*Estudiante*, seq(\mathbb{Z}))

proc NuevoSecundario(in *es* : conj(*Estudiante*)) : Secundario {

requiere { $|es| > 0$ }

asegura {

$res.estudiantes = es \wedge$

$(\forall e : Estudiante) (e \in es \leftrightarrow e \in res.faltas) \wedge_L$

$(\forall e : Estudiante) (res.faltas[e] = 0) \wedge$

$(\forall e : Estudiante) (e \in es \leftrightarrow e \in res.notas) \wedge_L$

$(\forall e : Estudiante) (res.notas[e] = \langle \rangle)$

}

}

proc RegistrarNota(inout *s* : Secundario, in *e* : Estudiante, in *nota* : \mathbb{Z}) {

requiere { $s = S_0 \wedge e \in s.estudiantes \wedge 0 \leq nota \leq 10$ }

asegura {

$s.estudiantes = S_0.estudiantes \wedge$

$s.faltas = S_0.faltas \wedge$

$s.notas = setKey(S_0.notas, e, S_0.notas[e] + \langle nota \rangle)$

}

}

proc RegistrarFalta(inout *s* : Secundario, in *e* : Estudiante) {

requiere { $s = S_0 \wedge e \in s.estudiantes$ }

asegura {

$s.faltas = setKey(S_0.faltas, e, S_0.faltas[e] + 1) \wedge$

$s.alumnos = S_0.alumnos \wedge$

$s.notas = S_0.notas$

}

}

}

Se propone la siguiente estructura de representación:

Módulo SecundarioImpl **implementa** Secundario <

```
var estudiantes: Conjunto<Estudiante>
```

```
var faltas: Diccionario<Estudiante, int>
```

```
var notas: Array<Conj<Estudiante>>
```

```
var notasPorEstudiante: Diccionario<Estudiante, Array<int>>
```

>

Donde:

- En *estudiantes* están todos los estudiantes del colegio secundario
- En *faltas* tenemos para cada estudiante la cantidad de faltas que tiene hasta el momento
- En *notas* tenemos en la posición i -ésima a los estudiantes que tienen nota i
- En *notasPorEstudiante* tenemos para cada estudiante la cantidad de notas que tienen con valor i -ésimo

- a) Escribir en castellano, con precisión y detalle el invariante de representación y la función de abstracción.
- b) Escribir ambos en el lenguaje de especificación.

Ejercicio 4. Dados el siguiente TAD y su implementación

```
Ingrediente, Receta ES  $\mathbb{Z}$ 
TAD Cocina {
  obs recetario : dicc<Receta, conj<Ingrediente>>
  obs stock : dicc<Ingrediente, int>
  proc iniciarActividad() : Cocina {
    asegura { Ambos diccionarios comienzan vacíos }
  }

  proc nuevaReceta(inout c : Cocina, in r : Receta, in is : conj<Ingrediente>) {
    requiere { La receta no está registrada previamente y el conjunto no es vacío }
    asegura { Se registra la receta con los ingredientes requeridos }
  }

  proc comprarIngrediente(inout c : Cocina, in i : Ingrediente) {
    asegura { Se le agrega una existencia al ingrediente comprado }
  }
}
```

Módulo CocinaImpl implementa Cocina <

```
/* Todas las recetas con los ingredientes que son necesarios.
 * Cada ingrediente consume una sola existencia */
var ingredientesPorReceta: Diccionario<Receta, Conjunto<Ingrediente>>

/* Todos los ingredientes que se poseen.
 * Cada ingrediente aparece tantas veces como existencias se tengan */
var existencias: Array<Ingrediente>

/* Cola de prioridad ordenada de acuerdo
 * a la cantidad que se tiene de cada ingrediente */
var ingMasAbundante: ColaDePrioridadMax<Tupla<Ingrediente, int>>

/* Todas las cantidades con el conjunto
 * de los ingredientes que tienen esas existencias */
var abundancias: Diccionario<int, Conjunto<Ingrediente>>

>
```

- a) Indique si cada una de las sentencias propuestas pertenecen al invariante de representación para la estructura propuesta
 - i) Dadas dos recetas distintas en **ingredientesPorReceta** sus conjuntos de ingredientes son disjuntos
 - ii) Todo ingrediente asociado a una receta en **ingredientesPorReceta** debe estar presente en **existencias**
 - iii) Todas las claves registradas en **abundancias** son mayores que cero
 - iv) Dados dos cantidades registradas en **abundancias** sus conjuntos de ingredientes son disjuntos
 - v) Todo ingrediente presente en **existencias** debe ser ingrediente de alguna receta registrada en **ingredientesPorReceta**

- b) Proponga todas las sentencias en castellano que falten para incluir todas las restricciones correspondientes a **ingMasAbundante** en el módulo
- c) Escribir la función de abstracción en lógica de primer orden

Ejercicio 5. Considere el siguiente TAD y una implementación

```

Persona, Mesa ES  $\mathbb{Z}$ 
TAD Elecciones {
  obs padron :  $\text{dicc}\langle \text{Persona}, \text{Mesa} \rangle$ 
  obs votaron :  $\text{conj}\langle \text{Persona} \rangle$ 
  obs votosPorCandidato :  $\text{dicc}\langle \text{Persona}, \mathbb{Z} \rangle$ 
  proc iniciar(in cs :  $\text{conj}\langle \text{Persona} \rangle$ , in vs :  $\text{dicc}\langle \text{Persona}, \text{Mesa} \rangle$ ) : Elecciones
    // Comienzan las elecciones con los candidatos de cs con 0 votos y el conjunto de votaron de vs vacío.
    // Los candidatos pueden ser votantes
  proc votar(inout e : Elecciones, in v : Persona, in m : Mesa, in c : Persona)
    // El votante v está registrado en la mesa y todavía no había votado y el candidato c compite.
    // Se registra el voto a favor del candidato en e.votaron y e.votosPorCandidato
  proc participación(in e : Elecciones) :  $\mathbb{R}$ 
    // Se devuelve el porcentaje de votantes que ya sufragaron
  proc primerLugar(in e : Elecciones) : Persona
    // Se devuelve el candidato que va ganando. En caso de empate se devuelve cualquiera
    // de los que vayan primeros
}
```

Módulo EleccionesImpl implementa Elecciones <

```

/* Todos los candidatos que compiten en la elección */
var candidatos : Conjunto<Persona>

/* Tiene cada mesa asociadas con los votantes registrados en ella */
var votantesPorMesa : Diccionario<Mesa, Conjunto<Persona>>

/* Cola de prioridad que tiene
 * - en la primera componente el candidato
 * - en la segunda la cantidad de votos que tiene hasta el momento, todos > 0.
 * Se ordenan según la cantidad de votos */
var ranking : ColaDePrioridadMax<Tupla<Persona, int>>

/* Todos los votantes que ya emitieron su voto */
var sufragaron : Conjunto<Persona>

/* Todos los votantes registrados en el padrón */
var empadronados : Conjunto<Persona>

>
```

- a) Indique si cada una de las sentencias propuestas pertenecen al invariante de representación para la estructura propuesta
- Todo votante de **sufragaron** pertenece al conjunto asociado a alguna mesa de **votantesPorMesa**
 - Todo votante que pertenece al conjunto asociado a alguna mesa de **votantesPorMesa** también pertenece a **sufragaron**
 - Los conjuntos asociados a todas las mesas registradas en **votantesPorMesa** son disjuntos entre sí
 - Para cada tupla de **ranking** su primer elemento pertenece a **candidatos**
 - Para cada tupla de **ranking** su segundo elemento no es negativo
 - Los conjuntos de votantes **empadronados** y **candidatos** son disjuntos
 - Todo votante de **empadronados** está presente en **sufragaron**
 - La unión de todos los conjuntos de votantes asociados a cada mesa de **votantesPorMesa** es igual a **empadronados**
- b) Escribir la función de abstracción en lógica de primer orden

Ejercicio 6. Considere el siguiente TAD y una implementación

```
Personaje, Clase, Arma ES  $\mathbb{Z}$ 
TAD Legion {
  obs escuadrones : dicc(Personaje, conj(Personaje))
  obs clases : dicc(Personaje, Clase)
  obs equipo : dicc(Personaje, Arma)
  proc crearLegion(in ps : dicc(Personaje, Clase)) : Legion
    // Se crea la legión con los personajes recibidos. Los personajes se reciben sin armas
    // e inicialmente no tenemos escuadrones.
  proc formarEscuadron(inout l : Legion, in p : Personaje, in ps : conj(Personaje))
    // El personaje l se convierte en líder del escuadrón creado con los personajes ps a su cargo.
    // Puede haber como máximo 10 escuadrones por restricción de sistema.
    // Cada personaje puede pertenecer como máximo a un sólo escuadrón.
  proc comprarArma(inout l : Legion, in p : Personaje, in a : Arma)
    // Se equipa el arma a al personaje p. Si ya tenía un arma, se descarta la previa
}
```

Módulo LegionImpl implementa Legion <

```
/* Todos los personajes que lideran un escuadrón */
var lideres : Conjunto<Personaje>

/* Tiene asociada a cada arma con los personajes que la tienen equipada */
var personajesPorArmas : Diccionario<Arma, Conjunto<Personaje>>

/* Relaciona a cada personaje con su clase */
var clasePorPersonaje : Diccionario<Personaje, Clase>

/* Todos los personajes que no pertenecen a ningún escuadrón */
var sinEscuadron : Conjunto<Personaje>

/* Todos los escuadrones formados con todos sus integrantes
 * incluyendo a los líderes */
var escuadrones : Array<Conjunto<Personaje>>(10)

>
```

a) Indique si cada una de las sentencias propuestas pertenecen al invariante de representación para la estructura propuesta

- i) Los conjuntos de personajes asociados a cada arma en **personajesPorArma** son disjuntos entre sí
- ii) Para cada arma de **personajesPorArma**, los personajes asociados no pertenecen a **sinEscuadron**
- iii) En **clasePorPersonaje** no pueden existir dos personajes con la misma clase
- iv) Los conjuntos **lideres** y **sinEscuadron** son disjuntos
- v) Todas las posiciones de **escuadrones** tiene conjuntos no vacíos
- vi) Para toda posición de **escuadrones**, los conjuntos son disjuntos entre sí
- vii) No hay repetidos en **lideres**
- viii) Todos los personajes de **sinEscuadron** son claves de **clasePorPersonaje**
- ix) Todas las armas de **personajePorArmas** tienen asociado un conjunto con un solo elemento
- x) Para cada posición de **escuadrones**, todos los personajes de cada conjunto no pertenecen a **sinEscuadron**

b) Escribir la función de abstracción en lógica de primer orden