**PARTE 2 - Equipos: Ejercicios de seudocódigo**

**Duración: 60 minutos**

**EJERCICIO 1**

La información de todos los alumnos de una facultad se han almacenado en una lista. Cada alumno está representado mediante un identificador único (numérico, sin código de verificación), su nombre, apellido y carrera en la que está matriculado (estos tres son alfanuméricos). Además, se tiene una lista con la información de los cursos de la facultad.

Una de las tareas típicas que se suele realizar es consultar los alumnos de una cierta asignatura. Para evitar realizar una búsqueda extensa sobre todos los alumnos de la facultad, se decide generar otras estructuras que permitan hacerlo en forma más eficiente.

Se desea entonces, dado la lista de todos los alumnos de la facultad (con los datos indicados más arriba, habiendo sido insertados de acuerdo a su número de CI), y dado el nombre de una cierta carrera, crear un nuevo árbol que indexe la información de los alumnos correspondientes, por Nombre y Apellido (se desea poder imprimir la lista de alumnos de la carrera ordenada por Nombre y Apellido en la forma más eficiente).

**Tipo Alumno**

Identificador : numérico

Nombre: alfanumérico

Apellido: alfanumérico

CursoIdentificador: numérico

**Tipo Curso**

Identificador: numérico

Nombre: alfanumérico

**Tipo TListaAlumnos <es un> TLista <Alumno>**

**Tipo TListaCursos <es un> TLista <Curso>**

**Tipo Facultad**

Alumnos: TLstaAlumnos

Cursos: TListaCursos

**de Tipo Facultad armarIndiceCurso(de tipo string CursoNombre): devuelve TArbolBB<Alumno>**

**CONSIDERACIONES IMPORTANTES**

* **La estructura resultante de invocar el método armarIndiceCurso debe estar optimizada para búsquedas de alumnos por nombre. Indique qué consideración debe tenerse en cuenta para cumplir con esto.**
* Deben respetarse estrictamente las firmas indicadas
* No es necesario desarrollar los métodos de los TDAs utilizados si estos corresponden a las variantes vistas en clase.
* Se deben cumplir todos los pasos estándar de desarrollo de algoritmos, Lenguaje Natural, Pre y Post Condiciones, Pseudocódigo y Análisis del Orden del Tiempo de Ejecución

**EJERCICIO 2**

Una institución educativa ofrece distintos cursos, los que suelen tener diferentes reglas de previaturas. El Coordinador desea contar con funcionalidades que le permitan estimar las cantidades de alumnos que se pueden inscribir a un cierto curso.

Por ejemplo, el curso “INTEGRADOR 101” puede nutrirse de alumnos que hayan cursado “BASICO-ING 1” o “BASICO-EMP 1” indistintamente (o ambos).

Otra situación típica es, por ejemplo, en el curso “EXIGENTE 102”, que solo puede ser cursado por aquellos alumnos que hayan aprobado AMBOS cursos “BASICO-ING 1” o “BASICO-EMP 1”.

El líder de equipo de desarrollo entiende que estas funcionalidades pueden obtenerse a partir de la abstracción “CONJUNTO” y sus correspondientes operaciones típicas: Unión e Intersección.

Desarrolla los algoritmos para implementar las operaciones de Unión e Intersección sobre el TDA, utilizado para representar un CONJUNTO.

* Conjunto <es un TDA>
* Conjunto.Union (Conjunto otroConjunto) : devuelve Conjunto
* Conjunto.Intersección (Conjunto otroConjunto) : devuelve Conjunto

CONSIDERACIONES IMPORTANTES

* Deben desarrollarse en detalle todos los métodos que se invoquen.
* Se tendrá en cuenta la eficiencia de los algoritmos desarrollados, calificando mejor aquellos que tengan mejor tiempo de ejecución o mejor performance general.