PEC2 – Segunda Prueba de Evaluación Continuada

Presentación

A continuación, os presentamos el enunciado de la segunda prueba de evaluación continuada del curso. Tenga en cuenta que la PEC debe resolverse individualmente.

Competencias

Las competencias que trabajaréis a la PEC son las siguientes:

Específicas

- Capacidad para analizar un problema en el nivel de abstracción adecuado a cada situación y aplicar las habilidades y conocimientos adquiridos para resolverlo.
- Capacidad para diseñar y construir aplicaciones informáticas mediante técnicas de desarrollo, integración y reutilización.
- Capacidad para proponer y evaluar diferentes alternativas tecnológicas para resolver un problema concreto.

Transversales

- Uso y aplicación de las TIC en el ámbito académico y profesional.
- Capacidad para innovar y generar nuevas ideas.

Objetivos

Los objetivos que se persiguen en el desarrollo de la PAC son los siguientes:

- Entender el concepto de TAD y saber hacer la especificación.
- Conocer la biblioteca de TADs de la asignatura y saber utilizarlos para diseñar e implementar nuevas estructuras de datos.
- Saber calcular la eficiencia espacial y temporal de una estructura de datos y los algoritmos asociados para comparar diferentes alternativas y poder elegir la mejor en términos de eficiencia (temporal o espacial).
- Ser capaz de identificar la estructura de datos utilizada en un programa y entender su funcionamiento.
- Entender el funcionamiento de los contenedores secuenciales y los árboles presentados en la asignatura. Saber cuándo y cómo utilizar estos contenedores.

Descripción de la PEC a realizar

La PEC consta de 4 ejercicios, algunos más teóricos y otros más prácticos, en los que practicareis los conocimientos adquiridos en el estudio de los tres primeros módulos de la asignatura.

Recursos

Los recursos necesarios para desarrollar la PEC son los siguientes:

Básicos (material didáctico de la asignatura)

- Módulo 4
- Módulo 5
- Módulo 6
- Módulo 7

Complementarios

No se requieren materiales complementarios.

Criterios de valoración

La puntuación global de la PAC es la suma de las puntuaciones individuales obtenidas en cada uno de los ejercicios que la componen. La puntuación individual de cada ejercicio se especifica en cada uno de ellos.

Formato y fecha de entrega

Fecha de publicación

05 de diciembre de 2016

Fecha de entrega

20 de diciembre de 2016

Formato de entrega

La entrega debe hacerse a través del espacio de Entrega y registro de EC con un único fichero preferentemente en formato PDF o, si no es posible, Word o OpenOffice. Este fichero contendrá la solución (incluyendo las clases Java). Por favor, no copiéis el enunciado, haced constar vuestro nombre en cada página (por ejemplo, con un pie de página), y numerad las páginas.

Enunciado

En esta PEC continuamos diseñando una estructura de datos para gestionar las infracciones de tráfico y el carnet por puntos. Ahora ampliaremos las funcionalidades y trabajaremos con un volumen de información grande, que requerirán el uso de estructuras de datos de los módulos 4 al 7. Concretamente, para la realización del ejercicio consideramos:

De la PEC1:

- El número de tipos de infracción TI sigue siendo pequeño y acotado, sobre un centernar.
- El número de vehículos de un conductor VC sigue siendo pequeño y acotado a 10.

Cambios respecto a la PEC1:

- El número de conductores C pasa ahora a ser muy grande, pero sigue siendo desconocido.
- El número de vehículos V sigue siendo conocido, pero ahora lo consideraremos muy grande, del orden de cientos de miles.
- El número de infracciones de un conductor IC pasará a ser muy grande y en constante aumento.

Nuevo en la PEC2:

- El número de agentes de policía AP no será muy grande, del orden de unos cientos y será desconocido.
- El número de infracciones que reporta un agente de policía IAP será desconocido y puede llegar a ser muy grande.

Ejercicio 1 [2’5 puntos]

Especificad un TAD ***GestionInfracciones*** que permita:

De la PEC1 (nótese que, aunque la firma de las operaciones no cambia, puede que la implementación sí que lo haga debido a la nueva elección de las estructuras de datos):

- Añadir un nuevo conductor al sistema. De cada conductor sabremos su DNI que lo identificará, su nombre y los apellidos. Si ya existe un usuario con este DNI, actualizamos sus datos.
- Añadir un nuevo vehículo al sistema. De cada vehículo sabremos la matrícula que lo identificará, la marca, el modelo y el DNI del conductor habitual. Si el vehículo ya existe devuelve un error. Si el DNI del conductor habitual no existe, devuelve un error. Tened en cuenta que un vehículo sólo puede tener un conductor habitual pero que un conductor puede ser conductor habitual de hasta 10 vehículos. Si el conductor habitual a vincular con el vehículo ya es conductor habitual de 10 vehículos devuelve un error.
- Añadir un nuevo tipo de infracción al sistema. De cada tipo de infracción sabremos un identificador, el artículo del código de circulación infringido, una descripción y el número de puntos que descuenta. Si el tipo de infracción ya existe actualizamos los datos (no hace falta recalcular los puntos a descontar por los usuarios que hayan cometido esta infracción)
- Consultar el historial de infracciones de un conductor. Considerad que el conductor debe existir. El historial tiene que mostrar todas las infracciones cometidas por el conductor con todos los vehículos que es conductor el habitual por orden cronológico.
- Consultar los puntos del conductor habitual de un vehículo. Considerad que el vehículo debe y que sólo se proporciona la matrícula del vehículo. Si el conductor habitual del vehículo tiene 0 puntos disponibles, se debe indicar un mensaje: “El infractor tiene retirado el carnet.
- Consultar los 5 tipos de infracción más habituales ordenados de mayor a menor. En caso de empate no importa el orden.

Cambios respecto a la PEC1 (operaciones en las que cambia la firma respecto a la que le proporcionamos en la PEC1):

- Registrar una infracción cometida por un vehículo. De cada infracción cometida sabremos el vehículo, la fecha y hora, el agente de policía que la reporta y un texto descriptivo. La infracción se asignará al conductor habitual del vehículo y le descontará automáticamente los puntos asociados al tipo de infracción. Si el agente, vehículo o el tipo de infracción no existen, devuelve un error. Los pasos a realizar para esta operación son los siguientes:
    - Si el infractor tiene 0 puntos disponibles antes de la infracción, se debe imprimir un mensaje de aviso (“El infractor ya tenía retirado el carné") y registrar la infracción.

- Se debe hacer el descuento de puntos (en caso de que el descuento dé un número negativo el infractor se quedará con 0 puntos)
- Si como resultado de la sanción del infractor tiene 0 puntos disponibles se debe imprimir un mensaje de aviso (“El infractor tiene retirado el carné")
- Si como resultado de la sanción del infractor tiene 0 puntos disponibles este conductor pasa a ser candidato para ser enviado al curso de recuperación de puntos.
- Si como resultado de la sanción del infractor aún tiene puntos disponibles también se debe imprimir un mensaje de aviso (“Al infractor le quedan X puntos")

Nuevo en la PEC2:

- Añadir un nuevo agente de policía al sistema. De cada agente en sabremos su número identificativo, su nombre y apellidos. Si el agente ya existe devuelve un error.
- Seleccionar un conductor para ser enviado al curso de recuperación de puntos. El conductor seleccionado será el conductor que haga más tiempo que esté con 0 puntos. Esta operación en ningún caso puede ser lineal respecto al número de conductores.
- Consultar las infracciones que ha notificado un agente de policía. Considerad que el agente de policía debe existir. El historial debe mostrar todas las infracciones notificadas por el policía en orden cronológico.
- Consultar los 10 agentes de policía que han notificado más infracciones ordenados de mayor a menor. En caso de empate no importa el orden.

Apartado a) [1 punto]

Especificad la firma del TAD ***GestionInfracciones*** de los métodos modificados respecto a la PEC1 y de los nuevos métodos de la PEC2. Es decir, indicad el nombre que darías a las operaciones encargadas de cada funcionalidad requerida. Indicad, también, los parámetros de entrada y de salida cuando sean necesarios.

Solución:

Cambios respecto a la PEC1:

- registerTrafficViolation(carPlate, policemanId, idTrafficViolation, date, time, description)

Nuevo en la PEC2:

- addPoliceman(policemanId, name, surname)
- sendDriverToPointRecoveryCourse(): Driver
- getPolicemanTrafficViolations(policemanId): Iterador
- topPolicemen(): Iterador

Apartado b) [1,5 puntos]

Haced la especificación contractual de las operaciones del TAD ***GestionInfracciones***, modificadas o nuevas en la PEC2. En la redacción de la especificación puedes usar, si se requiere, cualquiera de las operaciones del TAD. Tomad como modelo la especificación del apartado 1.2.3 del Módulo 1 de los materiales docentes. Se valorará especialmente la concisión (ausencia de elementos redundantes o innecesarios), precisión (definición correcta del resultado de las operaciones), completitud (consideración de todos los casos posibles en que se puede ejecutar cada operación) y carencia de ambigüedades (conocimiento exacto de cómo se comporta cada operación en todos los casos posibles) de la solución. Es importante responder este apartado usando una descripción condicional y no procedimental. La experiencia nos demuestra que no siempre resulta fácil distinguir entre las dos descripciones, es por eso que hacemos especial énfasis insistiendo que pongáis mucha atención en vuestras definiciones.

A título de ejemplo indicaremos que la descripción condicional (la correcta a utilizar en el contrato) de llenar un vaso vacío con agua sería:

> @pre el vaso se encuentra vacío.
> @post el vaso está lleno de agua.

En cambio, una descripción procedimental (incorrecta para utilizar en el contrato) tendría una forma parecida a:

> @pre el vaso tendría que encontrarse vacío, si no se encontrase vacío se tendría que vaciar.
> @post se acerca el vaso al grifo y se echa agua hasta que esté lleno.

También debéis tener en cuenta el invariante si éste es necesario para describir el TAD.

Solución:

@pre cierto

@post las infracciones del conductor serán las mismas más una nueva con los datos indicados. El número de puntos del conductor serán los mismos menos los puntos que queda la infracción cometida excepto en el caso en que el resultado sea menor que 0, en cuyo caso el número de puntos del conductor será 0, la pila de conductores candidatos a hacer el curso de recuperación de puntos tendrá el conductor que ha cometido la infracción y se notificará. Las infracciones notificadas por el agente serán las mismas más la nueva infracción. Si no existen el vehículo, el tipo de infracción o el agente de policía devuelve error.

- registerTrafficViolation (carPlate, policemanId, idTrafficViolation, date, time, description)

@pre cierto

@post La cola de conductores candidatos a ir al curso de recuperación de puntos tiene los mismos elementos menos el conductor que hace más tiempo que está ahí. Devuelve el conductor que hace más tiempo que está en la cola.

- sendDriverToPointRecoveryCourse (): Driver

@pre cierto.

@post si el código de agente es nuevo, los agentes serán los mismos más un nuevo agente con los datos indicados. Si existe un agente con código de agente *agentId* devuelve error.

- addPoliceman (policemanId, name, surname)

@pre existe un agente con identificador *policemanId*

@post devuelve un iterador para recorrer las infracciones notificadas por el agente de policía en orden cronológico.

- getPolicemanTrafficViolations (policemanId): Iterador

@pre cierto

@post devuelve un Iterador para recorrer los 10 agentes que han notificado más infracciones ordenados según el número total de infracciones notificadas.

- topPolicemen(): iterador

Ejercicio 2 [3,5 puntos]

En el ejercicio 1 habéis definido la especificación del TAD ***GestionInfracciones***. Ahora os pedimos que hagáis el diseño de las estructuras de datos que formarán este TAD. Diseñad, pues, el sistema para que sea el máximo de eficiente posible, tanto a nivel de eficiencia espacial como temporal, teniendo en cuenta los volúmenes de información y las restricciones especificadas en el enunciado.

Tened en cuenta sólo las operaciones que se piden en el enunciado al hacer este diseño.

Apartado a) [0,5 puntos]

Dudamos entre utilizar una tabla de dispersión, un AVL o continuar con un vector ordenado (como la PEC1) para almacenar los conductores. Justificad cuál creéis que es la mejor opción.

Solución:
La mejor opción es un AVL ya que el número de conductores pasa a ser muy grande e ilimitado. Con la lista encadenada las consultas serían asintóticamente peores y con volúmenes de datos grandes se afectaría mucho el rendimiento. Una tabla de dispersión no es adecuada si la información que queremos almacenar no está limitada. Elegimos un **AVL** que nos dará costes logarítmicos para las consultas.

Apartado b) [0,5 puntos]

Dudamos entre utilizar una tabla de dispersión, un AVL o continuar con un vector ordenado para almacenar los vehículos. Justificar cuál cree que es la mejor opción.

Solución:
Como el volumen de datos pasa a ser muy grande descartamos seguir utilizando un vector, tenemos que elegir entre el AVL y la tabla de dispersión. Como nos dicen que el número de vehículos es conocido y no necesitamos ningún recorrido ordenado elegiremos una **tabla de dispersión** para conseguir un acceso de consulta constante.

Apartado c) [0,5 puntos]
Dudamos entre utilizar una cola prioritaria, una lista encadenada o un AVL para almacenar los agentes de policía. Justificad cuál creéis que es la mejor opción.

Solución:
La mejor opción es una lista encadenada: Las colas prioritarias permiten ordenar los elementos para una prioridad. Pero los agentes no tienen ninguna prioridad asignada, por tanto, en este caso, no tendría sentido. El número de agentes nos dicen que no es muy grande pero que es muy variable, tanto las listas encadenadas como los AVL irían bien. Como el volumen no es muy grande podemos asumir costes lineales, nos decantamos por una simple **lista encadenada**.

Apartado d) [0,5 puntos]
Dudamos entre utilizar un AVL, una vector o continuar con una lista encadenada para almacenar las infracciones de un conductor. Justificar cuál cree que es la mejor opción.

Solución:
De las infracciones de un conductor nos dice el enunciado que sigue siendo variable pero que pasa a ser muy grande, esto hace que descartamos el vector. Con la lista encadenada las consultas serían asintóticamente peores y con volúmenes de datos grandes se afectaría mucho el rendimiento. Un AVL nos dará costes logarítmicos para las consultas. Elegimos pues un **AVL**.

Apartado e) [0,5 puntos]
Razonar si necesitamos alguna estructura de datos para guardar los conductores que se quedan sin puntos. En caso afirmativo cual y por qué.

Solución:
El enunciado nos dice que tenemos que ofrecer una operación que seleccione un conductor para ser enviado al curso de recuperación de puntos. El conductor seleccionado debe ser el conductor que haga más tiempo que esté con 0 puntos. También nos dice que esta operación no puede ser lineal respecto al número de conductores. Esto hace que **SI** que tengamos que guardar los conductores que se quedan sin puntos en una estructura de datos y la más adecuada es una **Cola** ya que la operación de selección de conductores debe volver al conductor que haga más tiempo que se ha quedado sin puntos.

Apartado f) [0,5 puntos]
Justificar el resto de estructuras de datos para hacer el diseño del TAD, incluyendo las que no cambian respecto a la PEC1. La justificación de cada una de las estructuras de datos debe ser del estilo:
"Para guardar XXX elegimos una lista encadenada ordenada ya que el número de elementos no es muy grande, nos hace falta acceso directo y necesitamos recorridos ordenados."

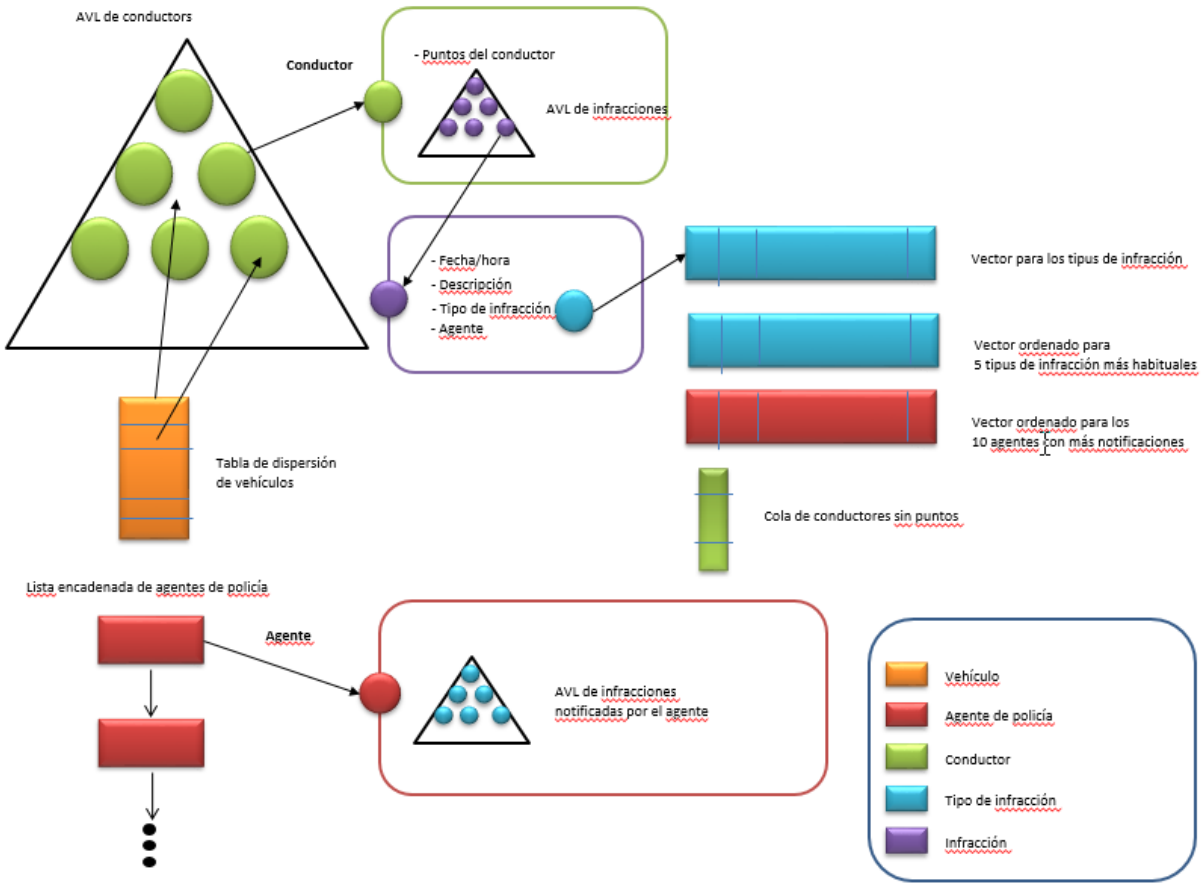
Solución:

- Para guardar los 5 tipos de infracción más habitual seguiremos utilizaremos un **vector ordenado** de 5 posiciones ya que el enunciado no varía las condiciones para esta estructura de datos.
- Para guardar los tipos de infracción seguiremos utilizaremos un **vector** ya que el enunciado no varía las condiciones para esta estructura de datos.
- Para guardar los 10 agentes de policía que han notificado más infracciones ordenados de mayor a menor utilizaremos un **vector ordenado** de 10 posiciones. Al igual que en el caso de las 5 infracciones más habituales, debido al número pequeño de elemento también sería válido un vector sin ordenación, pero en este caso habría una estructura auxiliar ya que la operación de consultar los 10 tipos de infracción más habituales los devolverá ordenados de mayor a menor.
- Como el número de infracciones que reporta un agente de policía es desconocido y puede llegar a ser muy grande utilizaremos un **AVL**.

Apartado g) [0,5 puntos]

Haced un dibujo de la estructura de datos global por TAD ***GestionInfracciones*** donde se vean claramente las estructuras de datos que elijáis para representar cada una de las partes y las relaciones entre ellas. Se debe hacer el dibujo de la estructura completa, con todas las estructuras que os permitan implementar las operaciones definidas en la especificación.

Solución:



Ejercicio 3 [3 puntos]

En el ejercicio 1 habéis definido la especificación del TAD ***GestionInfracciones*** con sus operaciones y en el ejercicio 2 habéis elegido las estructuras de datos para cada parte del TAD. En este ejercicio os pedimos que os fijéis en los algoritmos que os servirán para implementar algunas de las operaciones especificadas y en el estudio de eficiencia de las mismas. Tened en cuenta que la implementación de las operaciones va estrechamente ligada a la elección de las estructuras de datos que hayáis hecho.

Apartado a) [1,5 puntos]

Describid y haced el estudio de eficiencia de la operación que permite seleccionar un conductor para ser enviado al curso de recuperación de puntos. Hacedlo primero suponiendo que no elegimos ninguna estructura de datos para guardar los conductores que se quedan sin puntos y luego suponiendo que guardamos los conductores en una estructura adicional.
Para hacerlo tenéis que describir brevemente su comportamiento indicando los pasos que la componen (con frases como, por ejemplo: “insertar en el árbol AVL / borrar de la tabla de dispersión / consulta del pilón / ordenar el vector...”), especificando la eficiencia asintótica de cada paso y dando la eficiencia total de la operación. No hay que hacer el pseudocódigo, solo describir los pasos.

Solución:
Si no utilizamos ninguna estructura adicional para guardar los conductores que se quedan sin puntos necesitaremos recorrer todos los conductores para seleccionar los que estén sin puntos e ir comparando hasta obtener el que haga más tiempo que se quedó sin puntos. Como el AVL de las infracciones está ordenado por fecha la complejidad asintótica para encontrar la infracción más reciente de un conductor será logarítmica respecto en número de infracciones. Con todo ello la complejidad temporal será:

- Recorrer todos los conductores => O(C)
- Para cada conductor:
  - Buscar la última infracción cometida por el conductor en el AVL de infracciones => O(log IC)
  - Si la fecha de la infracción es menor que la fecha de infracción del conductor candidato cambiarlos => O(1)

Por tanto, el coste total de la operación sería de:
O(C * log IC)

Esta complejidad se podría reducir a **O(C)** si añadimos un atributo a los conductores con la fecha de la última infracción y la vamos actualizando cada vez que un conductor comete una infracción.

Si utilizamos una cola como estructura de datos adiciona la complejidad se reduce a **O(1)** ya que la operación sólo deberá consultar el elemento de la cabeza de la cola. La operación que registra una infracción será necesario que añada el infractor a la cola si el infractor se queda sin puntos.

Apartado b) [1,5 puntos]

Volved a hacer el estudio de eficiencia que hicisteis en la PEC1 para la operación que permite registrar una infracción cometida por un vehículo.

Solución:

- Buscar el vehículo en la tabla de dispersión de vehículos => O (1)
- Acceder a su conductor habitual => O (1)
- Añadir la infracción al AVL de infracciones del conductor => O (log IC)
- Actualizar el número de puntos restantes del conductor => O (1)
- Añadir el conductor en la pila de conductores candidatos a recibir el curso de recuperación de puntos si el número de puntos se pasa a ser 0 o menor que 0 => O (1)
- Buscar el agente de policía que ha notificado la infracción a la lista encadenada de agentes => O (AP)
- Añadir la infracción al AVL de infracciones notificadas por el agente => O (log IAP)
- Actualizamos los 10 agentes con más notificaciones => O (1)
    - Si el vector de agentes con más notificaciones está lleno:
        - Tomar el elemento con menos notificaciones del vector de agentes con más notificaciones (la llamamos B) => O (1)
        - Comparar el número de notificaciones de B con el del agente que notifica la infracción que estamos tratando y sustituirlo si es necesario => O (1)
    - Si el vector de agentes con más notificaciones no está lleno añadimos el agente => O(1)
- Buscar el tipo de infracción en el vector de tipo de infracciones => O(TI)
- Incrementar el contador de número de infracciones de este tipo que se han cometido => O (1)
- Actualizamos los 5 tipos de infracción más habituales => O (1)
    - Si el vector de tipo de infracción más habituales está lleno:
        - Tomar el elemento con menos infracciones al vector de tipo de infracción más habituales (la llamamos B) => O (1)
    - Comparar el número de infracciones de B con el de la infracción cometida y sustituirlo si es necesario => O (1)
    - Si el vector de tipo de infracción más habituales no está lleno añadimos el tipo de infracción => O (1)

Total: **O(log IC + log IAP + TI)** y, como TI está acotado a unos cientos, podemos decir que la eficiencia asintótica es **O(log IC + log IAP)**

Ejercicio 4 [1 punto]

Indicad cuál de los TADs de la biblioteca de TADs de la asignatura te parecen más adecuados para utilizarlos en la implementación de cada una de las estructuras de datos definidas para el TAD ***GestionInfracciones***.

Solución:

- Para los guardar los conductores, infracciones de un conductor e infracciones de un agente de policía elegimos un **DiccionarioAVLImpl**.
- Para guardar los agentes de policía elegimos una **ListaEncadenada**.
- Para guardar los vehículos elegimos **TablaDispersion**.
- Para guardar conductores que se quedan sin puntos elegimos una **ColaVectorImpl**.
- Para guardar los tipos de infracción más habituales y los agentes de policía con más notificaciones, implementaremos una nueva clase que implemente un vector ordenado con operaciones para añadir ordenadamente y hacer consultas mediante búsqueda dicotómica. Para integrar esta clase en la biblioteca de clases, implementaremos las interfaces de **ContenedorAcotado** y de **Diccionario**.
- Para guardar los tipos de infracción, implementaremos una nueva clase que implemente un vector con operaciones para añadir y hacer consultas mediante búsqueda lineal. Para integrar esta clase en la biblioteca de clases, implementaremos las interfaces de

ContenedorAcotado y de **Diccionario**. Si pudiéramos utilizar las colecciones del jdk nos serviría un simple Vector.