Junio 2018

3. (6 puntos) Se desea desarrollar una aplicación que permita gestionar un parque natural. En el parque natural hay ecosistemas, cada uno identificado a través de un nombre único. Cada ecosistema contiene un cierto número de ejemplares de cada una de las distintas especies que en él habitan. En un ecosistema no pueden habitar dos especies con igual identificador, aunque una misma especie (igual identificador) sí puede habitar en dos o más ecosistemas distintos. Se pueden añadir nuevos ecosistemas en el parque; en un ecosistema se puede añadir una nueva especie con un cierto número de ejemplares o incrementar el número de ejemplares de una especie ya existente en el ecosistema; se puede consultar las últimas especies nuevas que han pasado a formar parte de un ecosistema; se puede consultar el número de ejemplares que hay de una determinada especie en un ecosistema; se puede obtener un listado ordenado alfabéticamente de todas las especies del parque; por último, se puede consultar el número de ejemplares que hay de una determinada especie en el parque.

Se pide implementar un TAD `ParqueNatural` que proporcione las siguientes operaciones:

- `crea()`: Crea un parque natural vacío.
- `an_ecosistema(ecosistema)`: Añade un nuevo ecosistema con identificador `ecosistema` al parque. Si el ecosistema ya existe, levanta una excepción `EEcosistemaDuplicado`.
- `an_ejemplares(ecosistema,especie,n)`: Añade `n` ejemplares de la especie con identificador `especie` al ecosistema con identificador `ecosistema`. Si `ecosistema` no existe, levanta una excepción `EEcosistemaNoExiste`. Si la especie ya habita en el ecosistema se incrementará su número de ejemplares en `n`; si la especie no habita en el ecosistema, se registrará en el ecosistema esa nueva especie con ese número de ejemplares.
- `lista_especies_ecosistema(ecosistema,n)->lista`: Devuelve una lista con los identificadores de las `n` últimas nuevas especies añadidas al ecosistema de identificador `ecosistema`, ordenadas por orden inverso de inserción (es decir, primero la última nueva añadida, segundo la penúltima nueva añadida, y así sucesivamente). Si `ecosistema` no existe, levanta una excepción `EEcosistemaNoExiste`.
- `numero_ejemplares_en_ecosistema(ecosistema,especie)->numero`: Devuelve la cantidad de ejemplares de la especie `especie` que habitan el ecosistema `ecosistema`. Si `ecosistema` no existe, levanta una excepción `EEcosistemaNoExiste`. Si la especie no habita en el ecosistema, la operación devolverá `0` ejemplares.
- `lista_especies_parque()->lista`: Devuelve una lista de todas las especies del parque, ordenada alfabéticamente.
- `numero_ejemplares_en_parque(especie)->numero`: Devuelve el número de ejemplares que hay de la especie `especie` en el parque. Si la especie no habita en el parque, la operación devolverá `0` ejemplares.

Debes elegir la representación del TAD teniendo en cuenta que debes conseguir que la implementación de las operaciones sea lo más eficiente posible. Debes, así mismo, indicar justificadamente la complejidad de cada una de estas operaciones.

Septiembre 2018

3. (6 puntos) Nos han encargado implementar un sistema para la gestión de las listas de espera en un sistema de venta online de entradas de conciertos.

Cuando un cliente se registra en el sistema, se le asigna un código de identificación único (cadena de caracteres). El sistema permite además dar de alta conciertos identificados también mediante un código único (cadena de caracteres), así como gestionar las listas de espera de clientes para comprar las entradas de los conciertos dados de alta.

Para llevar a cabo la implementación de este sistema hemos decidido desarrollar un TAD `SistemaVentas` con las siguientes operaciones:

- `crea()`: Operación constructora que crea un sistema de gestión de venta de entradas vacío.
- `an_concierto(codigo_concierto)`: Añade un nuevo concierto al sistema, con código de identificación `código_concierto`. Si ya está dado de alta un concierto con dicho código, la operación lanzará una excepción `EConciertoExistente`.

- `an_cliente(codigo_cliente, codigo_concierto)`: Añade un nuevo cliente al sistema con código de identificación `codigo_cliente`,y lo pone a la espera de compra de entradas para el concierto con código de identificación `codigo_concierto`. En caso de que ya exista el cliente, o no exista el concierto, la operación lanzará una excepción `EAltaNoAdmitida`.
- `borra_concierto(codigo_concierto)`: Elimina el concierto con código de identificación `codigo_concierto` del sistema. Para ello dicho concierto debe existir; si no es así, la operación lanzará una excepción `EConciertoInexistente`. Así mismo, su lista de espera debe estar vacía; en otro caso, la operación lanzará una excepción `EConciertoConEsperas`.
- `borra_cliente(codigo_cliente)`: Elimina todo rastro del cliente con código de identificación `codigo_cliente` del sistema. Si el cliente no existe, la operación lanzará una excepción `EClienteInexistente`.
- `hay_clientes_en_espera(codigo_concierto)->boolean`: Devuelve *cierto* si hay clientes a la espera de comprar entradas para el concierto `codigo_concierto`, y *falso* en otro caso. El código del concierto debe existir; si no, la operación lanzará un excepción `EConciertoInexistente`.
- `proximo_cliente(codigo_concierto)->codigo_cliente`: Devuelve el código `codigo_client` del primer cliente en la lista de espera para el concierto `codigo_concierto`. Si no existe un concierto con el código dado, la operación lanzará una excepción `EConciertoInexistente`. Si el concierto existe pero no tiene lista de espera, la operación lanzará una excepción `EConciertoSinEsperas`.
- `venta(codigo_concierto)`: Realiza la venta de entrada para el concierto `codigo_concierto`. Para ello, elimina el primer cliente de la lista de espera, y dicho cliente queda en disposición de realizar nuevas compras. Lanzará la excepción `EConciertoInexistente` en caso de que el concierto con codigo `codigo_concierto` no exista. Lanzará la excepción `EConciertoSinEsperas` en caso de que el concierto exista pero no tenga lista de espera.
- `abandona(codigo_cliente)`: Registra el abandono, por parte del cliente `codigo_cliente`, de la lista donde se encuentra esperando. Como consecuencia, el cliente se elimina de dicha lista de espera y el cliente pasa a estar en diposición de realizar nuevas compras. Se lanzará la excepcion `EClienteInexistente` en caso de que el cliente no exista. Si el cliente existe pero no se encuentra esperando en ninguna lista de espera, la operación no tendrá efecto.
- `pon_en_espera(codigo_cliente,codigo_concierto)`: Pone al cliente `codigo_cliente` en espera para comprar una entrada en el concierto `codigo_concierto`. El sistema admite únicamente que un cliente esté esperando para comprar entradas en, a lo sumo, un concierto. La operación elevará una excepción `EEsperaNoAdmitida` si no existe un cliente con código `codigo_cliente`, si no existe un concierto con código `codigo_concierto` o si el cliente está ya en una lista de espera.
- `clientes()->lista`: Devuelve una lista, ordenada alfabéticamente, con los códigos de los clientes registrados en el sistema.
- `num_clientes()->entero`: Devuelve el número de clientes registrados en el sistema.
- `num_conciertos()->entero`: Devuelve el número de conciertos registrados en el sistema.

Dado que éste es un sistema crítico, la implementación de las operaciones debe ser lo más eficiente posible. Por tanto, debes elegir una representación adecuada para el TAD, implementar las operaciones y justificar la complejidad de cada una de ellas.

Mayo 2019

3. (5 puntos) La tienda `Outlet aGoGo` nos ha encargado programar un TAD para gestionar su sistema de venta online de ofertas. Este sistema permite ofertar cantidades limitadas de sus productos a precios especiales. Cada vez que un cliente desea beneficiarse de una oferta, el sistema lo pone en la lista de espera para dicha oferta, gestionando las ventas por estricto orden de llegada, hasta que se agoten las unidades disponibles (un cliente podrá esperar simultáneamente en más de una lista de espera). El TAD deberá incluir las siguientes operaciones:

- `crea`: Crea un sistema de venta vacío.
- `an_oferta(producto, num_unidades)`: Crea una nueva oferta para el producto `producto`, con `num_unidades` unidades disponibles. Esta es una operación parcial: no debe existir ya un producto con el mismo nombre en el sistema, y, además, el número de unidades debe ser positivo.
- `pon_en_espera(cliente, producto)`: Añade al cliente `cliente` a la lista de espera de la oferta para el producto `producto`. En caso de que el cliente ya esté esperando para comprar dicho producto, la operación no tendrá ningún efecto. Esta es una operación parcial: el producto al que se hace referencia debe estar ofertándose actualmente en el sistema.
- `cancela_espera(cliente, producto)`: Elimina al cliente `cliente` de la lista de espera correspondiente al producto en oferta `producto`. Si el cliente no está esperando en la lista de espera del producto, la operación no tendrá ningún efecto. Esta es una operación parcial: el producto debe estar ofertándose en el sistema.
- `num_en_espera(producto)->num_clientes`: Devuelve el número de clientes que están esperando para comprar el producto `producto`. Esta es una operación parcial: el producto debe estar ofertándose en el sistema.
- `venta(producto, num_unidades)`: Registra una venta de `num_unidades` unidades del producto `producto` al primer cliente de la lista de espera. Dicho cliente se elimina de la lista de espera. En caso de que, tras dicha venta, no queden más unidades, la venta para el producto se cerrará, eliminado la oferta del sistema (y, por tanto, todos los clientes que están esperando se quedarán sin producto). Esta es una operación parcial: el producto debe estar ofertándose actualmente en el sistema, la lista de espera para dicho producto no debe estar vacía, y el número de unidades solicitado no debe sobrepasar el número de unidades disponibles.
- `primero_en_espera(producto)->cliente`: Devuelve el nombre del primer cliente que está esperando para comprar el producto `producto`. Esta es una operación parcial: el producto se debe estar ofertando actualmente, y su lista de espera no debe estar vacía.
- `lista_ventas()->Lista`. Devuelve una lista del número de unidades vendido para cada producto desde la puesta en marcha del sistema. Cada elemento de esta lista consiste en: (i) el nombre del producto; y (ii) todas las unidades vendidas de este producto (si un producto se ha ofertado varias veces, este valor será el total vendido para cada oferta). La lista estará ordenada alfabéticamente por los nombres de productos.

Aparte de realizar la implementación del TAD, debes indicar justificadamente cuál es la complejidad de cada operación. Al tratarse de un sistema crítico, la implementación debe ser lo más eficiente posible.

Julio 2019

2. (6.5 puntos) Nos han encargado implementar un módulo para la gestión de surtidores en una gasolinera. La gasolinera tiene surtidores, cada uno de los cuáles puede servir distintos tipos de combustible. Cuando un vehículo llega a la gasolinera, se pone en la cola de espera de alguno de los surtidores, y reposta combustible (los detalles de pagos no se cubrirán en este módulo). Así mismo, en cualquier momento un vehículo puede abandonar la cola de espera, y seguir su camino. La implementación de este módulo se llevará a cabo como un TAD `GestorSurtidores` con las siguientes operaciones:

- `crea`: Creación de un nuevo gestor de surtidores.
- `an_surtidor(id_surtidor)`: Se añade un nuevo surtidor, con identificador `id_surtidor`, al sistema. Si el surtidor ya existe en el sistema, se señala un error (mediante la excepción `ESurtidorDuplicado`).
- `carga(id_surtidor, tipo_combustible, num_litros)`: Carga `num_litros` de combustible `tipo_combustible` en el surtidor `id_surtidor`. Si el surtidor no existe en el sistema, se señala un error (excepción `ESurtidorNoExiste`).
- `pon_en_espera(id_vehículo, id_surtidor)`: Pone en espera el vehículo `id_vehículo` en el surtidor `id_surtidor`. Si el vehículo ya está esperando en otro surtidor, o bien `id_surtidor` no existe, se señala un error (excepción `ELlegadaVehiculo`).

- `vende(id_surtidor,tipo_combustible,num_litros)->resul`: Actualiza el sistema tras realizar una venta de `num_litros` de combustible `tipo_combustible` al primer vehículo que está esperando en el surtidor `id_surtidor`. Tras la venta, se actualiza la reserva de combustible del surtidor, y el vehículo abandona la gasolinera. Una vez realizada la venta, la operación devuelve un objeto que contiene los siguientes campos: (i) el identificador del vehículo al que se ha realizado la venta; y (ii) la cantidad de combustible de tipo `tipo_combustible` que queda aún en el surtidor. Para que esta operación puede realizarse: (i) el surtidor debe existir; (ii) debe haber vehículos esperando para abastecerse en el mismo; (iii) el surtidor debe disponer de la cantidad suficiente de combustible solicitado; y (iv) `num_litros` debe ser mayor que `0`. Si se viola alguna de estas condiciones, la operación señala un error (excepción `EErrorVenta`)
- `abandona(id_vehiculo)`: El vehículo `id_vehiculo` abandona la cola en la que está esperando, y sale de la gasolinera. Si el vehículo no está esperando en ninguna cola, la operación no tiene efecto.

Dado que este módulo es un sistema crítico, debe elegirse una representación que permita implementar lo más eficientemente posible las operaciones pedidas, así como llevar a cabo dicha implementación. Para cada operación debe indicarse, además, justificadamente su complejidad.