Estructura de Datos
Grado II

PECL1 2019-20

SIMULACIÓN DEL CONTROL DE ADUANAS EN UN AEROPUERTO

Carlos Javier Hellín Asensio

ÍNDICE

TAD`s creados y definición de las operaciones del TAD

spec PILA[PASAJERO]
uso NATURALES
parámetro formal
    géneros pasajero
fparametro
géneros pila
operaciones
pvacia: → *pila* {crear una pila vacía}
insertar: pasajero pila → *pila* {poner un pasajero en la pila}
parcial *extraer: pila* → pasajero {quitar un pasajero de la pila}
tipos
enlace-pila = **puntero a** nodo-pila
nodo-pila = **reg**
    valor: pasajero
    siguiente: enlace-pila
    longitud: natural
freg
pila = enlace-pila
ftipos

spec COLA[PASAJERO]
uso NATURALES
parámetro formal
    géneros pasajero
fparametro
géneros cola
operaciones
cvacía: → *cola {crea una cola vacía}*
insertar: pasajero → *cola {poner un pasajero en la cola}*
insertarConPrioridad: pasajero → *cola {poner un pasajero ordenado por prioridad en la cola}*
parcial eliminar: cola -> pasajero {quitar un pasajero de la cola}
tipos
enlace-cola = **puntero a** nodo-cola
nodo-cola = **reg**
    valor: pasajero
    siguiente: enlace-cola
freg
cola = **reg**
    primero: enlace-cola
    ultimo: enlace-cola
    longitud: natural
freg
ftipos

spec LISTA[BOX]
usa BOOLEANOS, NATURALES
parámetro formal
    géneros pasajero
fparametro
géneros lista
operaciones
clista: → *lista {crea una lista vacía}*
insertar: box → lista {poner un box en la lista}
insertarOrdenado(Box *v); box → lista {poner un box ordenado por el id en la lista}
eliminar: enlace-lista → enlace-lista {eliminar un box de la lista}

colaMasCorta(): nada → box {devolver el box de la lista con la cola más corta}
estanOcupados(): nada → bool {devolver true si los boxes de la lista están ocupados}
estanLibres(): nada → bool {devolver true si los boxes de la lista están libre}
eliminarVacios(): nada → nada {quitar los boxes vacíos de la lista}
eliminarPasajeros(): nada → bool {quitar los pasajeros ya atendidos de los boxes de la lista}
restarDuracionPasajeros(): nada → natural {restar la duración de los pasajeros siendo atendidos en los boxes de la lista}
totalLongitudColas(): nada → natural {devolver la longitud total de las colas de los boxes en la lista}

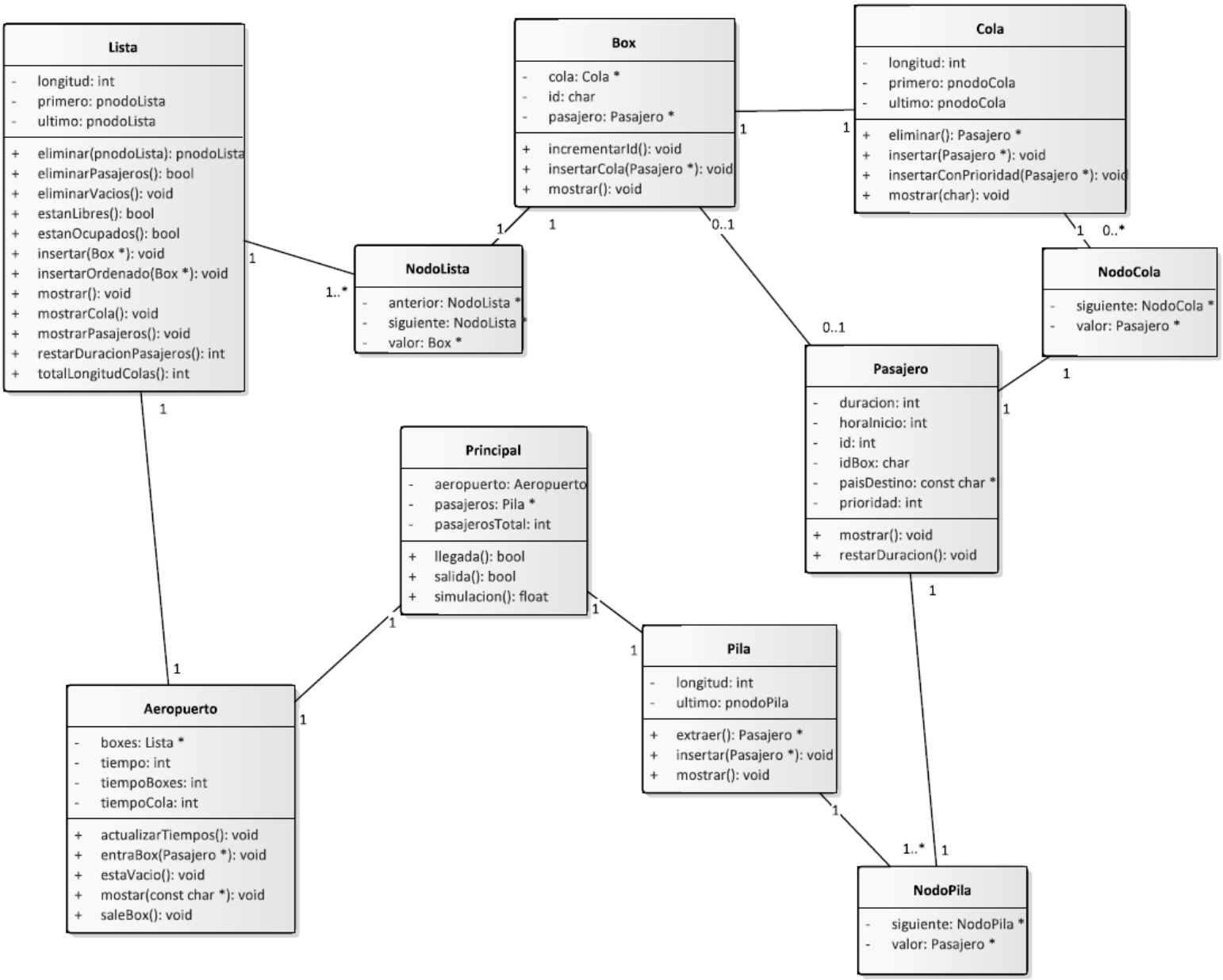
tipos

enlace-lista = **puntero a** nodo-lista
nodo-lista = **reg**
    valor: pasajero
    siguiente: enlace-lista
    anterior: enlace-lista
freg
lista = **reg**
    primero: enlace-lista
    ultimo: enlace-lista
    longitud: natural
freg
ftipos

Solución adoptada: descripción de las dificultades encontradas

- Box con identificador en código ASCII: para poder ordenar la lista de boxes se utiliza como código identificador para el box un char como tipo de dato y así almacenar una *letra (A, B, C…) que también se muestre en consola, por ejemplo, como BOX A.*
- Mostrar en la consola las colas debajo de su respectivo box: se ha adoptado a mostrarlo como en el ejemplo de la simulación de las colas en la Parte I, y se muestran todas las colas de los boxes seguidas y no debajo de cada box.

Diagrama UML



Explicación de los métodos más destacados

Clase Principal
simulacion(): entra en un bucle while con un sleep de 1 segundo para poder mostrar en tiempo real en la consola los resultado de la simulación. Para ello llama a los métodos llegada() y salida() que simulan los eventos del mismo nombre. También se encarga de llamar a actualizarTiempos() en aeropuerto para posteriormente cuando el aeropuerto se encuentre vacío, que se comprueba llamando al método estaVacio(), pueda devolver el tiempo medio de atención en minutos.
llegada(): se encarga de extraer un pasajero de la pila y comprobar si su tiempo de hora inicio es igual al tiempo del simulador, de ser así llama a entraBox(pasajero) del aeropuerto. Devuelve true si se ha producido el evento para luego mostrarlo en la consola.
salida(): simplemente llama al método saleBox() del aeropuerto para efectuar las salidas de los pasajeros en los boxes. También devuelve true si se ha producido el evento para mostrarlo en consola.

Clase Aeropuerto
actualizarTiempos(): actualiza las variables tiempoBoxes cuando va restado la duración de los pasajeros en los boxes que están siendo atendidos, tiempoCola con la longitud total de las colas de todos los boxes e incrementa tiempo para mostrar la hora de la simulación en la consola.
entraBox(Pasajero *pasajero): obtiene el box con la cola más corta, comprueba si ese box está vacío para asociarle con un Pasajero o sino lo inserta en la cola del propio box. Al finalizar comprueba si todos los boxes tienen más de 2 pasajeros en la cola para añadir un box nuevo.
saleBox(): elimina los pasajeros cuya duración sea 0 en los boxes atendidos. Por último, comprueba si hay dos o más boxes vacíos para eliminarlos de la lista. Devuelve true si se ha producido una salida de algún pasajero en el box que estaba siendo atendido.
estaVacio(): Si el tiempo es mayor que 0 y los boxes están vacíos devuelve true, que junto a la longitud de la pila de pasajeros cuando sea 0 da como significado que la simulación ya ha terminado.

Clase Pasajero
restarDuracion(): decrementa la duración del pasajero siempre que se encuentre siendo atendido por un box.

Explicación del comportamiento del programa

Se inicia en el punto de entrada main del programa que crea una pila donde se insertan los pasajeros para realizar las pruebas de la simulación. El último de la pila es el que más tarda en salir y el primero es el que sale antes.

Se inicia la clase Principal pasando al constructor el puntero de la pila y se llamada a la función simulación() que devolverá el tiempo medio de atención en minutos para luego ser mostrado en la consola.

El constructor de la clase Principal obtiene la pila de pasajeros y guarda en una variable la longitud de la pila para luego calcular la media. También tiene como atributo la clase Aeropuerto *cuyo constructor crea una nueva lista para los boxes, inserta el primer box con identificador 'A' e* inicializa los tiempos a 0.

En simulacion() entra en el bucle while que duerme el proceso mediante sleep cada 1 segundo para mostrar en tiempo real el resultado de la simulación. Empieza llamando a los métodos llegada() y salida() para simular los eventos del aeropuerto, comprueba que si el aeropuerto está vacío y no hay más pasajeros en la pila entonces devuelve la media. En caso contrario continua en el bucle actualizando los tiempos necesarios para luego poder calcular la media.

Cuando se produce un evento de llegada entonces la Lista se encarga de obtener la cola más corta de todos los boxes que hay en su interior, para posteriormente asociar ese pasajero al box vacío y si no está vacío lo inserta en la cola que tiene el box. Durante este evento, y una vez terminado lo anterior, también se añade un nuevo box a la lista cuando en todos los boxes hay más de 2 pasajeros en cada cola.

En el evento de salida se van eliminado los pasajeros que han terminado de ser atendidos en los boxes, que esto es cuando la duración del pasajero es igual a 0, y borra los boxes vacíos cuando hay dos o más libres, pero siempre manteniendo al menos un box en la lista.

La clase Lista del tipo doblemente enlazada es la encarga de la gestión de todos los boxes que tiene en cada momento. Tiene la función de obtener la cola más corta de todos los boxes aprovechando que la clase Cola tiene el atributo longitud que va incrementando cuando se inserta un nuevo pasajero y decrementando cuando se elimina. También inserta los boxes ordenados alfabéticamente usando el identificador del box como una letra en código ASCII (A es 65, B es 66, etc..) de esta forma el método sabe en qué orden debe insertar el box. Además, comprueba si los boxes están ocupados y libres, así como eliminar los boxes vacíos y los pasajeros que han terminado de ser atendidos.

Tanto la clase Pila como la Cola tienen su funcionamiento estándar de estas estructuras de datos, exceptuando la Cola que tiene como método insertar con prioridad para que la cola tenga los pasajeros ordenados de mayor a menor prioridad.

Box y Pasajero tienen los atributos necesarios para usar su método mostrar en pantalla el resultado además de los exigidos en el enunciado de la práctica. Box se encarga de asociar un Pasajero a sí mismo, de insertar el Pasajero a su cola asociada y de incrementar el identificador cuando se trata de un Box nuevo que todavía no ha sido insertado en la Lista y ya existe un Box con el mismo identificador.

Para finalizar, el comportamiento de Pasajero es la de reducir la duración del tiempo que está en un Box mientras es atendido además de su método de mostrar en pantalla.