Instituto de Tecnología ORT
Carrera de Analista de Sistemas
Materia: Programación 2

Propuestas Propuestos para Unidad 1

1. Hotel

Se trata de administrar un hotel. Del hotel sabemos el nombre, la dirección y las habitaciones que tiene. De cada habitación sabemos el número, el nombre del cliente (si está ocupada), el precio, la fecha de llegada y partida. Cada habitación puede tener adicionales. Los adicionales tienen un precio, fecha de prestación, y un tipo (Desayuno, Room-Service, Gym, o Traslado).

Realizar:

a. Diagrama de Clases
b. La explotación del método **obtenerAdicionales** de la clase **Habitación**, que devuelve una lista con los adicionales prestados a la misma.

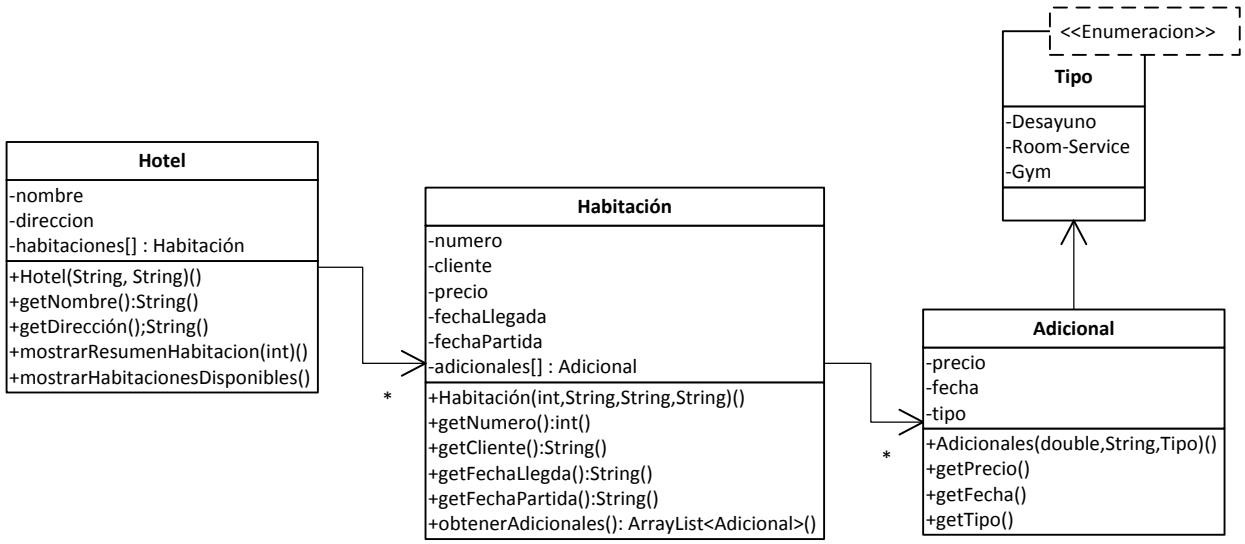
Nota: Consideramos que cuando un cliente toma una habitación la lista de adicionales se pone en cero o se vacía.

c. La explotación del método **mostrarResumenHabitación** de la clase **Hotel**, que recibe como parámetro el número de habitación y muestra por pantalla el nombre del hotel, la dirección, el nombre del cliente, el número de habitación, y un detalle con el costo de alojamiento (precio de la habitación entre días de llegada y partida) y los adicionales consumidos de la misma.

Nota: Consideramos que cuando se pide el resumen el cliente está de salida.

d. La explotación del método **mostrarHabitacionesDisponibles** de la clase **Hotel** mostrando el número de habitación, el precio y la fecha de partida (de la última vez que se ocupó si la tiene)

Bosquejo de Resolución – Diagrama de Clases



Estrategia general para resto de los puntos:

b. obtenerAdicionales()**:** a recorrer completamente con for/for each la lista de adicionales y devolver la lista.
c. mostrarResumenHabitación(): a recorrer con ciclo condicional en la lista de habitaciones para encontrar la habitación, usando el método anterior, recorrer completamente en la lista devuelta, sumarizar y armar el listado.
d. mostrarHabitacionesDisponibles(): a recorrer la lista de habitaciones con ciclo condicional y mostrar los datos de la habitación.

2. Banco

Un banco digital nos pide una solución para la gestión de clientes y cuentas. Del banco sabemos el nombre, la dirección, y las cuentas que tiene. Cada cuenta tiene un número, un saldo, fecha de apertura, cliente y un tipo (caja de ahorro, cuenta sueldo, o cuenta corriente), un monto descubierto (lo evalúa el banco cuando abre la cuenta).

A las cuentas, se les puede consultar el saldo, hacer depósitos y extracciones. El saldo de las extracciones no puede superar el mono del descubierto. El monto de descubierto de la caja de ahorro es cero. Cada cuenta conserva una lista de sus transacciones (para los depósitos, extracciones, y consultas de saldo). La transacción guarda la fecha, monto, y el tipo de operación (consulta de saldo, depósito, extracción).

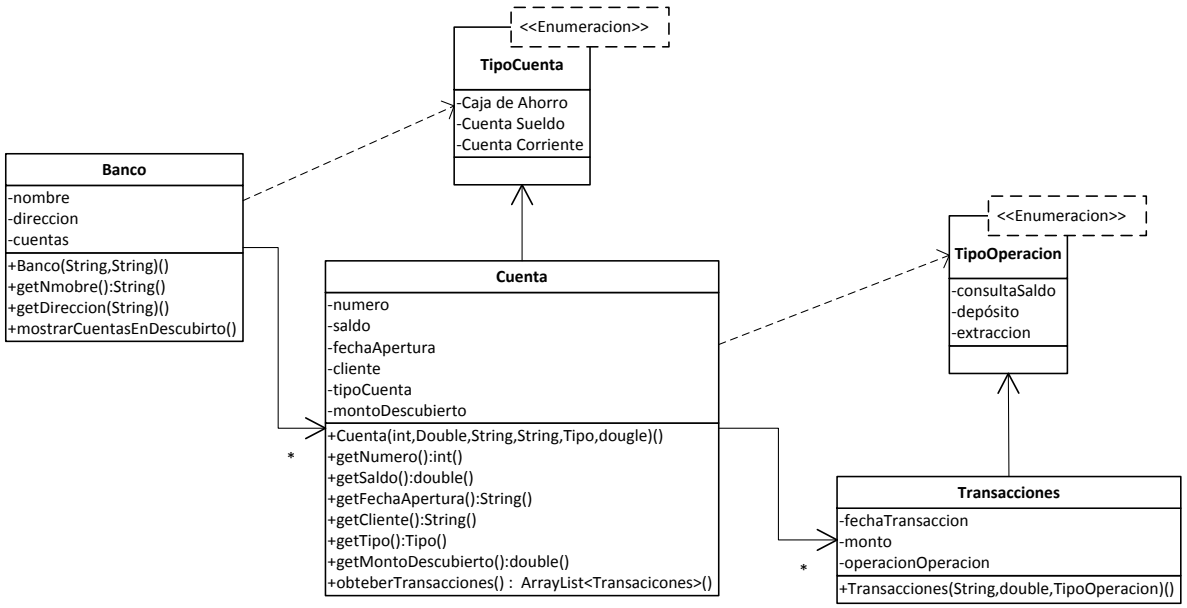
Se pide:

a. Diagrama de Clases
b. Desarrollar constructor parametrizado de la clase Cuenta y los métodos consulta de saldo, depósito y extracción.

Nota: Considere que los últimos 3 métodos mencionados anteriores crean transacciones.

c. La explotación del método **obtenerTransacciones** de la clase Cuenta, que devuelve la lista con las transacciones de la cuenta.
d. La explotación del método **mostrarCuentasEnDescubierto** de la clase Banco, que muestra por pantalla las cuentas del tipo Cuenta Sueldo y Cuenta Corriente con saldo negativo y sus 10 últimas transacciones.

Bosquejo de Resolución – Diagrama de Clases



Estrategia General para resto de los puntos:

b. Constructor y métodos**:** nada especial, solo que en el cuerpo se cree el objeto "transacción con el tipo de operación adecuado.
c. obtenerTransacción()**:** a recorrer completamente con for/for each en la lista de transacciones y devolver una lista.
d. mostrarCuentasEnDescubierto(): a recorrer la lista de habitaciones con ciclo condicional y mostrando las ultimas 10 transacciones.

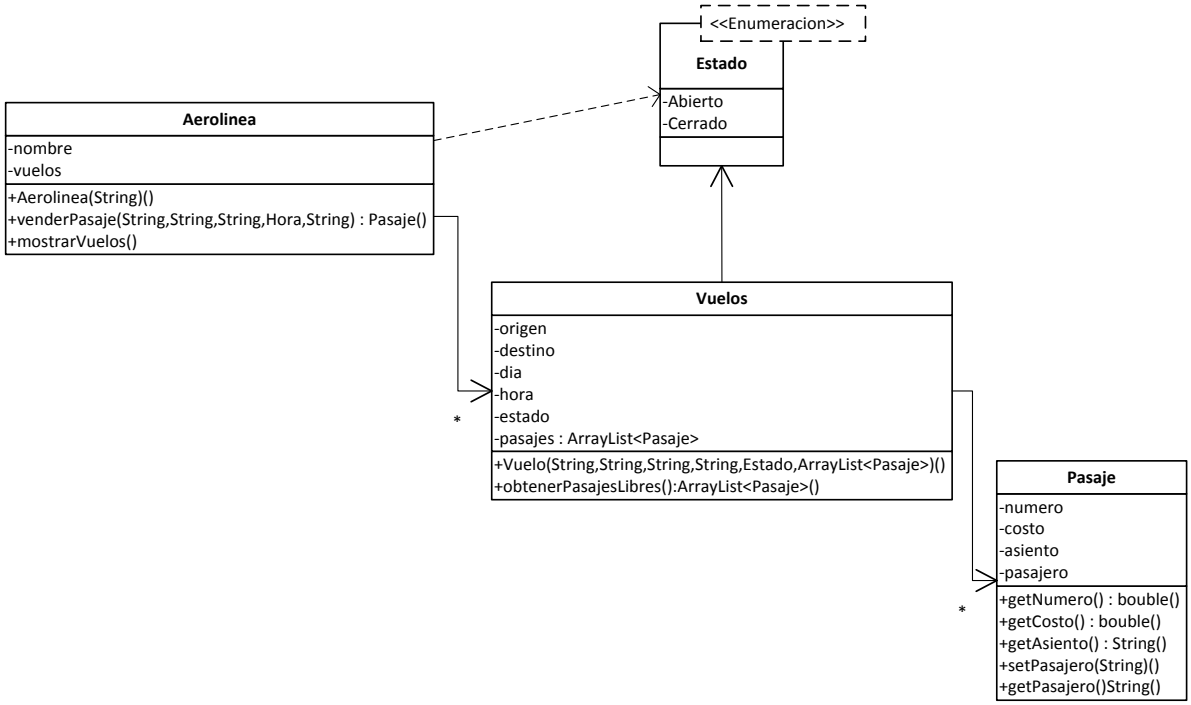
3. Aerolínea

Una nueva aerolínea low-cost, nos pide le ayudemos con la gestión de los pasajes de sus vuelos. De la empresa sabemos el nombre y los vuelos que maneja. De cada vuelo sabemos origen y destino, día y hora, el estado (abierto, cerrado) y la lista pasajes disponibles. Del pasaje sabemos su número, el costo, el asiento, el pasajero (si está en nulo o libre aún no está vendido).

Se pide:

a. Diagrama de Clases
b. La explotación del método **obtenerPasajesLibres** de la clase **Aerolinea**, que devuelve la lista con los pasajes libres de los vuelos abiertos. Si no hay pasajes devuelve un "null".
c. La explotación del método **venderPasaje** de la clase **Aerolínea**, que recibe origen, destino, fecha, hora, ek nombre del pasajero busca y devuelve un pasaje en los vuelos abiertos. Si hay uno disponible le asigna el nombre del pasajero, lo devuelve y lo lista por pantalla, sino devuelve un "null" y cambia el estado del vuelo a cerrado.
d. La explotación del método **mostrarVuelos** de la clase **Aerolínea**, que recibe un estado, busca los vuelos y los muestra.

Bosquejo de Resolución – Diagrama de Clases



Estrategia General para resto de los puntos:

b. obtenerPasajesLibres()**:** a recorrer la lista de pasajes con ciclo condicional armando una lista con la Info. de cada pasaje. Refuerza el tema de no generar la lista si no hay pasajes.
c. vernderPasaje()**:** requiere modularizar para la búsqueda de vuelo abierto (recorriendo con ciclo condicional), y llamar a obtenerPasajesLibres(). Refuerza el tema de cambiar el estado si ya no hay pasajes.
d. mostrarVuelos(): recibe como parámetro un "Estado", implica recorrer la lista de Vuelos con ciclo condicional.