Ejercicios colecciones

Para cada uno de los siguientes ejercicios, se debe realizar el diagrama UML y escribir el código Java. Se deben desarrollar los métodos indicados por las funcionalidades y también cualquier otro que fuera necesario para cumplir con las mismas, en las clases en que corresponda.

1. Un parque de **Kartings** nos pide desarrollar un programa para administrar la cola de personas que quieren jugar, la lista de autos disponibles y la recaudación resultante. Las personas que quieren jugar tienen que anotarse primero en una lista de espera. De cada persona sabemos el nombre, el apellido y su edad. Si una persona ya está anotada no puede volver a anotarse hasta tanto haya jugado. La edad mínima es 12 años y la máxima es 65. Los menores de 18 años pagan una tarifa reducida. Por otra parte el parque tiene una cantidad fija de kartings la cual es conocida. Cada karting es identificado con un número de 1 a N, puede estar operativo o no y también lleva un acumulador de su monto recaudado. Al principio todos los kartings arrancan en estado operativo. Las tarifas son de $17000 por cada menor y $25000 por cada mayor que usa el vehículo. Además cada karting registra la última persona que usó el vehículo.

   Se debe poder anotar una persona registrando la misma en la lista de espera. Esta funcionalidad recibirá el nombre, el apellido y la edad de la persona que se desea anotar. Esta acción debe tener uno de los siguientes resultados:

   - NOMBRE_INVALIDO: si el nombre o el apellido está en blanco.
   - EDAD_INVALIDA: si la persona tiene menos de 12 o más de 65 años
   - YA_ESTA_ANOTADA: si la persona ya figura en la lista de espera.
   - ANOTADA_OK: si la persona pudo ser registrada.

   También se debe poder cambiar de estado de un karting, recibiendo su número. Si el número existe se debe invertir el estado operativo del mismo. Si pasa a estado no operativo debe eliminarse la referencia a su ocupante (debe ser null). Este método debe devolver verdadero si se pudo invertir el estado operativo del karting o falso si no se pudo.

   Otra acción a implementar es ubicar personas. Funciona así: en caso de que haya por lo menos dos personas en la lista de espera se asigna a cada karting en estado operativo una persona de la lista, en el orden en que se fueron anotando. Al subirse al auto las personas son eliminadas de la lista de espera y cada auto registra la recaudación tomando en cuenta si la persona es menor o mayor. Si la lista de personas no alcanza a cubrir todos los autos, los coches restantes deben ser seteados a null. Este método devuelve la cantidad de personas de la lista de espera que pudieron ser ubicadas en los distintos vehículos.

   Por último se necesita mostrar la recaudación, indicando todos los datos de cada vehículo y finalmente el total recaudado.

   Se deben probar las funcionalidades solicitadas desde la clase PruebaKarting.

2. La empresa de **alquiler turístico AirUtn** desea tener un sistema para poder administrar los alojamientos que tiene en alquiler en varios destinos turísticos de la Argentina. Cada destino tiene un nombre y una lista de los alojamientos destinados a alquiler turístico. Para cada alojamiento se tiene una dirección que es su identificación (String), su precio por noche y su capacidad (cantidad de pasajeros). Se necesitan las siguientes funcionalidades:

   - Mostrar todos los alojamientos de un destino, se recibirá el nombre de un destino y se mostrarán por pantalla sus alojamientos indicando su dirección, su capacidad y su precio.
   - Obtener el destino con mayor cantidad de alojamientos. Se devolverá una lista que contenga el o los destinos con más alojamientos.
   - Poder cambiar un alojamiento a otro destino. Esto es necesario cuando un alojamiento fue cargado en un destino incorrecto. Se recibe la dirección del alojamiento y se debe devolver un valor booleano que indique si pudo hacerse el cambio.
   - Borrar un alojamiento conociendo su dirección. Debe devolver el alojamiento que se eliminó o nulo si no pudo ser borrado.

   Se probará el funcionamiento de todos los puntos en la clase pruebaAirUtn.

3. Nos piden desarrollar la **aplicación de citas UTtinder.** El objetivo de la misma es encontrar coincidencias en una lista de personas. Estas se registran indicando datos personales y otros relativos a sus preferencias de búsqueda. Luego cada persona puede ver una lista con todos los usuarios que coinciden con su criterio de búsqueda y donde además, ella misma cumple con los criterios de búsqueda del otro (coincidencia recíproca).

   De cada persona se conoce nombre, dirección de mail, género, edad, género buscado, rango de edad buscada (mínima y máxima). El nombre y la dirección de mail no pueden tener repetidos. Los géneros pueden ser FEMENINO, MASCULINO, INDISTINTO.

   Se necesitan las siguientes funcionalidades:

   - Registrar una persona, evitando los duplicados.
   - Verificar coincidencia parcial. Se recibirán dos personas y se devolverá verdadero si la segunda persona coincide con lo requerido por la primera persona.

- Verificar coincidencia total. Se recibirán dos personas y se devolverá verdadero si ambas cumplen con los criterios de búsqueda de la otra persona.

- Mostrar listado de coincidencias. Se recibirá el mail de una persona (que exista) y se mostrarán todas las personas que se ajusten completamente a sus preferencias de búsqueda.

Se probará el funcionamiento en la clase pruebaTinder..

4. Un famoso cocinero tiene una **casa inteligente.** En ella todos los artefactos pueden comunicarse entre sí. El cocinero tiene problemas de organización y se olvida de comprar algunos productos necesarios. Por eso decidió comprarse un recetario electrónico, que puede comunicarse con su heladera y con su alacena, los cuales son depósitos también inteligentes. La diferencia entre la heladera y la alacena es que la primera está refrigerada. De cada producto que hay en la alacena y en la heladera se sabe su nombre, cantidad, y fecha de vencimiento. Cada receta del recetario tiene un código, su nombre y una lista con los ingredientes necesarios para la preparación de una porción, indicando la cantidad necesaria y si dicho ingrediente es refrigerado o no.

   Debemos desarrollar una solución para implementar el nuevo recetario electrónico. Las funcionalidades necesarias son:

   - Preparar listado de lo que hay que comprar. Se recibirá el nombre de la receta a cocinar, la cantidad de porciones deseadas y un depósito. El método devolverá una lista con los nombres de los ingredientes que faltan y la cantidad que hace falta comprar para poder prepararla. Esta lista puede quedar vacía (si no hace falta comprar nada) y deberá ser completada buscando los ingredientes en cada depósito dependiendo de si el ingrediente es un producto refrigerado o no refrigerado.

   - Obtener una lista de las recetas posibles de preparar. Se recibirá un depósito y un entero indicando la cantidad de porciones deseadas y se debe devolver una lista con el o los nombres de la Recetas posibles que se pueden cocinar con los ingredientes que se tienen.

   Probar el recetario en la clase PruebaRecetario.

5. Una empresa **de trenes de carga** necesita desarrollar un sistema para administrar el traslado de granos de trigo. Solo se transporta un único producto (trigo) y todo lo transportado es homogéneo. Los trenes se identifican por un número correlativo, a partir de 1 en adelante. Esta numeración es automática (no la ingresa el usuario sino que la indica el sistema cuando se crea un nuevo tren). Cada formación (cada tren) tiene hasta 30 vagones. Existen tres tipos de vagones, denominados SMALL, MEDIUM y LARGE que son capaces de cargar 30, 40 y 50 toneladas de trigo respectivamente. Se necesitan las siguientes funcionalidades:

- Crear una formación. Se crea un tren sin vagones. A cada tren se le asigna un número correlativo. No se recibirá ningún parámetro y se debe retornar el número de tren asignado a esa formación.

- Agregar vagones. Se recibirá un número de tren, una cantidad de vagones y un tipo de vagón para (de ser posible) crear vagones del tipo requerido agregándolos al final de la formación. Se devolverán tres posibles valores:

  - NO_EXISTE_TREN: si el número no corresponde a un tren existente.
  - CANT_VAGONES_INVALIDA: si la cantidad de vagones es menor o igual que cero, o bien si la cantidad total de vagones (los existentes en el tren más los que se quieren agregar) excede el largo máximo posible.
  - AGREGADO_OK: si la operación se pudo realizar correctamente.

- Cargar un tren. Se recibirá un número de tren y una cantidad de toneladas de trigo a cargar. Para realizar la carga de los granos el tren debe existir y tener capacidad suficiente para acomodar la totalidad de la carga. Este método debe retornar verdadero si se pudo realizar la operación y falso en caso contrario.

- Listar la capacidad disponible. Debe mostrar todos los trenes con el porcentaje de espacio libre de cada formación (la carga acumulada de todos los vagones contra la capacidad total de cada tren).

- Sacar los vagones vacíos. Se recibirá un número de tren y se extraerán del mismo los vagones que están totalmente vacíos. Este método debe retornar cuántos vagones fueron eliminados. Si el tren no existe debe devolver -1.

6. Un prestigioso banco nos solicita desarrollar su propia **Billetera Virtual** . Por promoción de lanzamiento, la billetera no cobra intereses sobre las cuotas aplicadas. La aplicación brinda servicios a personas, por ello la misma tiene una lista de clientes.

   De cada cliente se tiene la siguiente información:
   - Dni (String, se supone único).
   - Lista de tarjetas (son las que el mismo usuario registrará en la aplicación).
   - Cantidad de compras realizadas (debe establecerse inicialmente en 0).

   De cada tarjeta se conoce:
   - Número de tarjeta (String, por ejemplo 4444333322221111).
   - Nombre de tarjeta (UTNPLUS, UTNCARD, FACULCARD).
   - Monto disponible (el mismo debe ser mayor o igual a 0).

   Cada cliente puede registrar las tarjetas que desee para usar en la aplicación. Las tarjetas no deben estar repetidas por usuario.

En líneas generales, la aplicación debe permitir:

- Agregar un cliente a la aplicación. recibe los datos necesarios para agregar un cliente en la aplicación (no se puede repetir el dni). El método retorna si pudo completar la operación.

- Registrar una tarjeta para un cliente. recibe un dni y los datos de la tarjeta. El usuario debe existir en la app y la tarjeta no debe existir previamente en la lista de tarjetas del cliente, Devuelve un valor para saber si pudo agregar la tarjeta.

- Mostrar tarjetas que pueden comprar: recibe el dni del cliente y un importe. Muestra por pantalla la información de las tarjetas que pueden realizar compras por el importe recibido.

- Mostrar tarjetas con saldo; muestra por pantalla los datos de los usuarios registrados y de sus tarjetas con saldo disponible.

- Obtener compras: devuelve una lista que contiene el DNI de cada usuario junto con la cantidad de compras que hizo hasta el momento.

- Realizar una compra: recibe un dni, un importe y una cantidad de cuotas y realiza la compra actualizando el saldo de la tarjeta y contando dicha compra. Debe utilizar la tarjeta que más saldos tenga y que pueda abonar el monto recibido.
  Retorna uno de los siguientes resultados:

  - TRANSACCION_OK: si pudo realizar la compra.
  - SIN_TARJETA_PARA_COMPRA: no existe alguna tarjeta con disponible suficiente para realizar la compra.
  - USUARIO_INEXISTENTE: el usuario indicado no está en la app.
  - ERROR: monto o cantidad de cuotas inválidas.