Ejercicios Colecciones Java

Semana del 23 y 30 de Marzo

1) Crea un programa que, dadas dos listas de enteros, devuelve una colección con los elementos comunes, sin repetición.

2) Crea una clase que mantenga cotizaciones en bolsa de empresas. Debes considerar objetos observadores que se puedan registrar en la clase para obtener los cambios en la cotización de las empresas. Realiza un diseño general, que permita tener distintos tipos de observadores. Por ejemplo, un observador que imprima por la consola los cambios en la cotización, otro que lo imprima por fichero, etc.

Modifica el programa para que los observadores se registren a cotizaciones de empresas concretas, y no de todas ellas.
Nota: puedes usar el patrón de diseño Observer: https://en.wikipedia.org/wiki/Observer_pattern

3) **Comparación de objetos:** Crea una **c**lase zapato, descrita por un número, modelo y color. Añade métodos para comparar (igualdad y orden natural), que considere primero el modelo alfabéticamente, luego el color alfabeticamente, y finalmente por número ascendente.

4) **Ejercicio del examen final de 2014:** Dadas las siguientes clases Java, **se pide** completar los espacios señalados (cuando sea necesario hacerlo) para que el método `main` produzca la salida indicada abajo. El propósito de la clase `ConexionesAereas` es almacenar, en orden alfabético, los nombres de las aerolíneas que ofrecen vuelos directos entre cada dos aeropuertos dados. Para cada trayecto directo entre dos aeropuertos, debe evitarse almacenar por duplicado la misma información para el trayecto en sentido inverso, es decir, intercambiando aeropuerto origen y aeropuerto destino. Además, según muestra la salida esperada del programa, la información de todas las conexiones aéreas almacenadas se mostrará ordenada por los trayectos directos almacenados (primero por aeropuerto de origen y después por aeropuerto de destino), y las aerolíneas que sirven cada trayecto, también han de presentarse por orden alfabético. La clase `ConexionesAereas` debe también tener un método para borrar una aerolínea de un trayecto.

```
public enum Aeropuerto { BCN, CDG, JFK, MAD; }
public class TrayectoDirecto { // completar la clase (1) si es necesario
       private Aeropuerto origen, destino;
       public Aeropuerto getOrigen() { return origen; }
       public Aeropuerto getDestino() { return destino; }
       public String toString() { return "(" + origen + "<>" + destino + ")"; }

       // (1)

} // end clase TrayectoDirecto

public class ConexionesAereas { // completar la clase (2) si es necesario

       // (2)

} // end clase ConexionesAereas

public class Ejercicio1 {
  public static void main(String[] args) {
   ConexionesAereas c = new ConexionesAereas();
   c.add( new TrayectoDirecto( Aeropuerto.MAD, Aeropuerto.JFK ), "NeverCrash" );
   c.add( new TrayectoDirecto( Aeropuerto.JFK, Aeropuerto.MAD ), "NeverCrash" );
   c.add( new TrayectoDirecto( Aeropuerto.MAD, Aeropuerto.JFK ), "EspaFlai" );
   c.add( new TrayectoDirecto( Aeropuerto.MAD, Aeropuerto.BCN ), "NeverCrash" );
   c.add( new TrayectoDirecto( Aeropuerto.BCN, Aeropuerto.MAD ), "EspaFlai" );
   c.add( new TrayectoDirecto( Aeropuerto.CDG, Aeropuerto.JFK ), "SubidonFree" );
   c.add( new TrayectoDirecto( Aeropuerto.JFK, Aeropuerto.CDG ), "FlaiJai" );
   c.add( new TrayectoDirecto( Aeropuerto.MAD, Aeropuerto.BCN ), "EspaFlai" );
   c.add( new TrayectoDirecto( Aeropuerto.BCN, Aeropuerto.MAD ), "EspaFlai" );

   if (c.remove(new TrayectoDirecto( Aeropuerto.JFK, Aeropuerto.CDG ), "FlaiJai"))
     System.out.println("FlaiJai borrado");
   System.out.println(c);
  } // end main
} // end clase Ejercicio1
```

Salida esperada:

```
FlaiJai borrado
{(CDG<>JFK)=[SubidonFree], (MAD<>BCN)=[EspaFlai, NeverCrash], (MAD<>JFK)=[EspaFlai, NeverCrash]}
```