Ejercicios Colecciones Java (Soluciones)

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.
[**ver vídeo con su resolución paso a paso en Eclipse**]

```
public class IntersectionNoRep {
  private static Collection<Integer> commonNoDup(List<Integer> list1, List<Integer> list2) {
        List<Integer> copy = new ArrayList<Integer>(list1);      // copy to avoid modifying list1
        copy.retainAll(list2);       // retain the elements in list1 that are in list2
        return new LinkedHashSet<>(copy);
  }

  public static void main(String[] args) { // Example of use
        List<Integer> list1 = Arrays.asList(10, 1, 3, 4, 4, 10);
        List<Integer> list2 = Arrays.asList(4, 2, 10, 34, 3, -2);

        Collection<Integer> noDup = commonNoDup(list1, list2);
        System.out.println(noDup);
  }
}
```

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
[**ver vídeo con su resolución paso a paso en Eclipse**]

```
public class Main {
  public static void main(String[] args) {
	CotizacionesBolsa cb = new CotizacionesBolsa("IBM", "Amazon", "Google");
	cb.registraObservador("IBM", new ObservadorConsola());
	cb.registraObservador("Amazon", new ObservadorFichero());
	cb.setCotizacion("IBM", 0);
	cb.setCotizacion("Amazon", 10);

	System.out.println(cb);
  }
}

public interface IObservador { void changed(String empresa, int from, int to); }

public class ObservadorConsola implements IObservador{
  @Override
  public void cambio(String empresa, int valor1, int valor2) {
	System.out.println("La empresa "+empresa+" cambia su valor de "+valor1+" a "+valor2);
  }
}

public class ObservadorFichero implements IObservador{
  @Override
  public void cambio(String empresa, int valor1, int valor2) {
	try {
		PrintWriter pw = new PrintWriter(new FileWriter("cotiz.txt", true));
		pw.println("La empresa "+empresa+" cambia cotizacion de: "+valor1+" a "+valor2);
		pw.close();
	} catch (IOException e) {
		e.printStackTrace();
	}
  }
}

public class CotizacionesBolsa {
  private Map<String, Integer> cotizacion = new TreeMap<>();
  private Map<String, List<IObservador>> observadores = new HashMap<>();

  public CotizacionesBolsa(String ...empresas) {
	for (String e : empresas)
		this.cotizacion.put(e, 0);
  }

  public void setCotizacion(String e, int valor) {
	for (IObservador o : this.observadores.getOrDefault(e, new ArrayList<>()))
		o.cambio(e, this.cotizacion.getOrDefault(e, 0), valor);
	this.cotizacion.put(e, valor);
  }

  @Override public String toString() {
	return this.cotizacion.toString();
  }

  public void anyadeObservador(String e, IObservador obs) {
	if (!this.observadores.containsKey(e))
		this.observadores.put(e, new ArrayList<>());
	this.observadores.get(e).add(obs);
  }
}
```

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.

```
public class ZapatoTester {
	private static void testHashSet(List<Zapato> lista) {
		Set<Zapato> pers = new HashSet<Zapato>(lista);
		System.out.println(pers);
	}

	private static void testLinkedHashSet(List<Zapato> lista) {
		Set<Zapato> pers = new LinkedHashSet<Zapato>(lista);
		System.out.println(pers);
	}

	private static void testTreeSet(List<Zapato> lista) {
		SortedSet<Zapato> pers = new TreeSet<Zapato>(lista);
		System.out.println(pers);
	}

	public static void main(String[] args) {
		List<Zapato> zaps = Arrays.asList(  new Zapato(34, "Kawasaki", Color.BLANCO),
		                                    new Zapato(43, "Adidas", Color.ROJO),
		                                    new Zapato(45, "Piccolino", Color.NEGRO),
		                                    new Zapato(43, "Adidas", Color.ROJO),
		                                    new Zapato(37, "Adidas", Color.ROJO));
		// Evitar repeticion
		testHashSet(zaps);
		testLinkedHashSet(zaps);
		// ordena por modelo, numero y color
		testTreeSet(zaps);
	}
}
enum Color { BLANCO, NEGRO, ROJO }

public class Zapato implements Comparable<Zapato> {
	private int numero;
	private String modelo;
	private Color color;

	public Zapato (int n, String m, Color c) {
		this.numero = n; this.modelo = m; this.color = c;
	}

	public String toString() {
		return "("+this.modelo+", "+this.numero+", "+this.color+")";
	}

	@Override
	public boolean equals(Object o) {
		if (o == this) return true;
		if (!(o instanceof Zapato)) return false;
		Zapato z = (Zapato) o;
		return  this.compareTo(z)==0;
	}

	@Override
	public int hashCode() {
		return this.numero+this.modelo.hashCode()+this.color.hashCode();
	}

	@Override
	public int compareTo(Zapato o) {
		int diff = this.modelo.compareTo(o.modelo);
		if (diff!=0) return diff;
		diff = this.color.compareTo(o.color);
		if (diff!=0) return diff;
		return this.numero-o.numero;
	}
}
```

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]}
```

Solución:

```
package aeropuertos;

import java.util.*;

enum Aeropuerto { BCN, CDG, JFK, MAD; }

class TrayectoDirecto  implements Comparable<TrayectoDirecto>{
	private Aeropuerto origen, destino;
	public Aeropuerto getOrigen() { return origen; }
	public Aeropuerto getDestino() { return destino; }
	public String toString() { return "(" + origen + "<>" + destino + ")"; }

	public TrayectoDirecto(Aeropuerto origen, Aeropuerto destino) {
		this.origen = origen; this.destino = destino;
	}

	public int compareTo(TrayectoDirecto t) {
		if ( this.origen.equals(t.origen) && this.destino.equals(t.destino) ||
				this.destino.equals(t.origen) && this.origen.equals(t.destino)) return 0;
		return this.toString().compareTo(t.toString());
	}
}

class ConexionesAereas {
	private Map<TrayectoDirecto, Set<String>> conexiones = new TreeMap<TrayectoDirecto,Set<String>>();
	public void add(TrayectoDirecto trayecto, String aerolinea) {
		if (! conexiones.containsKey(trayecto)) conexiones.put(trayecto, new TreeSet<String>());
		conexiones.get(trayecto).add(aerolinea);
	}
	public String toString() { return conexiones.toString(); }
	public boolean remove(TrayectoDirecto trayecto, String aerolinea) {
		if (!this.conexiones.containsKey(trayecto)) return false;
		return this.conexiones.get(trayecto).remove(aerolinea);
	}
}

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
}
```