Prueba 2 de Evaluación Continua
Análisis y Diseño de Software (2015/2016)

Contesta cada apartado en hojas separadas. El ejercicio 3 puedes contestarlo en esta hoja.

Apellidos:
Nombre:

Ejercicio 1: Programación Orientada a Objetos básica (3.5 puntos)

Se quiere realizar una aplicación para el montaje de video digital. La aplicación debe permitir crear planos fijos y secuencias de planos. Cada secuencia de planos está formada por planos fijos y otras secuencias de planos, que se especifican en el momento de la creación de esa nueva secuencia de planos. Tanto los planos como las secuencias de planos tienen un nombre, y el sistema debe asignarles un identificador único. Cada plano fijo tiene una duración que se establece en su creación. La duración de una secuencia de planos se calcula sumando la duración de los planos que contiene.

Se pide codificar las clases necesarias para completar la aplicación descrita arriba, de forma que el siguiente programa produzca la salida de más abajo.

```
public class Videos {
  public static void main(String[] args) {

      PlanoFijo intro = new PlanoFijo("introduccion", 20);
      Plano    presen = new PlanoFijo("presentacion", 150);
      SecuenciaPlanos epsPresen = new SecuenciaPlanos("Present. EPS", intro, presen);
      Plano    video  = new SecuenciaPlanos("Video EPS", epsPresen,
                                             new PlanoFijo("conclusiones", 30));

      System.out.println(video+" con duracion "+video.duracion()+" segundos");
  }
}
```

Salida esperada:

```
[[introduccion (id:0), presentacion (id:1)], conclusiones (id:3)] con duracion 200 segundos
```

Solución ejercicio 1:

```
public abstract class Plano {
	private String name;
	private int id;
	protected static int lastId = 0;

	public Plano(String n) {
		this.name = n;
		this.id = Plano.lastId++;
	}

	@Override
	public String toString() {
		return this.name+" (id:"+this.id+")";
	}

	abstract public int duracion();
}

public class PlanoFijo extends Plano {
	private int duracion;
	public PlanoFijo (String n, int precio) {
		super(n);
		this.duracion = precio;
	}
	@Override
	public int duracion() { return this.duracion; }
}

import java.util.*;
public class SecuenciaPlanos extends Plano {
	List<Plano> componentes = new ArrayList<Plano>();

	public SecuenciaPlanos (String nombre, Plano... partes) {
		super(nombre);
		this.componentes.addAll(Arrays.asList(partes));
	}
	@Override
	public int duracion() {
		int total = 0;
		for (Plano p : this.componentes) total += p.duracion();
		return total;
	}

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

Ejercicio 2: Interfaces y excepciones (3.5 puntos)

Se quiere realizar una aplicación para liquidar inversiones en monedas internacionales convirtiéndolas a efectivo en una moneda posiblemente distinta a la moneda de la inversión. Las inversiones se crean con *una* de las siguientes monedas: *DOLAR*, *EURO*, *LIBRA*, *RUBLO* y *YEN*, que **están ordenadas de mayor a menor *"preferencia bancaria***". Las inversiones deben cumplir el interfaz común `Inversion`, con un método `liquidar` (para convertir la inversión en efectivo). El método recibe como parámetro una moneda (posiblemente distinta a la de la inversión), y deberá lanzar una excepción *si la moneda parámetro es de mayor preferencia bancaria* que la moneda con que se creó la inversión.

Se pide codificar las clases necesarias para completar la aplicación descrita arriba, así como el siguiente programa, de tal forma que se produzca la salida de más abajo.

```
public class Cuentas {
	public static void main(String[] args) {

		Inversion[] cuentas = {	new CuentaAcciones(Moneda.EURO),
					new CuentaAcciones(Moneda.DOLAR),
					new CuentaAcciones(Moneda.RUBLO)   };
			try   {			// Completar (1)
				for (Inversion c : cuentas)
				   c.liquidar(Moneda.LIBRA);
			} catch (ErrorMoneda em)  {	// Completar (2)
				System.out.println(em);
			}
	}
}
```

Salida esperada:

```
Conversión correcta de cuenta en EURO a efectivo en LIBRA
Conversión correcta de cuenta en DOLAR a efectivo en LIBRA
Error convirtiendo cuenta en RUBLO a moneda LIBRA
```

Solución ejercicio 2:

```
public class ErrorMoneda extends Exception {
	private Moneda cuenta, destino;
	public ErrorMoneda(Moneda cuenta, Moneda destino) {
		this.cuenta = cuenta; this.destino = destino;
	}
	@Override public String toString() {
		return "Error convirtiendo cuenta en "+this.cuenta+" a moneda "+this.destino;
	}
}

public enum Moneda {
	DOLAR, EURO, LIBRA, RUBLO, YEN;
}

public interface Inversion {
	void liquidar(Moneda moneda) throws ErrorMoneda;
}

public class CuentaAcciones implements Inversion {
	private Moneda m;
	public CuentaAcciones(Moneda m) {
		this.m = m;
	}

	@Override
	public void liquidar(Moneda liquidacion) throws ErrorMoneda {
		if (liquidacion.ordinal() < this.m.ordinal())
			throw new ErrorMoneda(this.m, liquidacion);
		System.out.println("Conversion correcta de cuenta en "+this.m
							+" a efectivo en "+liquidacion);
	}
}
```

Ejercicio 3: Colecciones (3 puntos)

Desarrollando una aplicación para almacenar relaciones de amistad entre usuarios de una red social, y *asignar un valor de intensidad a cada relación*, se han creado las clases `Usuario` (con nombre y provincia) y `RedSocial` dadas abajo.

Se pide completar dichas clases, según sea necesario para que el programa dado abajo produzca la salida indicada.
Nota: Asumimos que no existirán dos usuarios con el mismo nombre en la misma provincia.

Solución al ejercicio 3:

```
public class Usuario {    no hacía falta completar nada    {  // completar si fuese necesario
	private String nombre, prov;
	public Usuario(String n, String p) { nombre = n; prov = p; }
	public String getNombre() { return nombre; }
	public String getProv() { return prov; }
	public String toString() { return nombre+"("+prov+")"; }
	// completar si fuese necesario

	@Override
	public boolean equals(Object obj) {
		if (! (obj instanceof Usuario)) return false;
		Usuario u = (Usuario) obj;
		return this.nombre.equals(u.nombre) && this.prov.equals(u.prov);
	}
	@Override
	public int hashCode() {
		return this.getNombre().hashCode() + 101 * this.getProv().hashCode();
	}

}
```

```
import java.util.*;
public class RedSocial {

	private  Map<Usuario, Map<Usuario,Integer>>  amistades =  new HashMap<>() ;
	// completar con método necesario


	public void asignaAmistad(Usuario u, Usuario a, Integer intensidad) {
		if (! amistades.containsKey(u))
			amistades.put(u, new HashMap<Usuario,Integer>());
		amistades.get(u).put(a,intensidad);
	}

	public String toString() { return amistades.toString(); }
}
```

```
public class MainRedSocial {
  public static void main(String[] args) {
	RedSocial rs = new RedSocial();
	Usuario u1 = new Usuario("Luis", "Leon");
	Usuario u2 = new Usuario("Ana", " Leon");
	Usuario u3 = new Usuario("Sol", "Alava");

	rs.asignaAmistad( u1, u3, 1 ); // asigna a u1 amistad con u3, intensidad 1
	rs.asignaAmistad( u1, u2, 2);
	rs.asignaAmistad( u2, u1, 4);
	rs.asignaAmistad( new Usuario("Ana", " Leon"), u3, 8 );
	rs.asignaAmistad( u3, u1, 0);

	System.out.println( rs );
  }
}
```

Salida esperada:

```
{Sol(Alava)={Luis(Leon)=0}, Ana(Leon)={Sol(Alava)=8, Luis(Leon)=4}, Luis(Leon)={Sol(Alava)=1, Ana(Leon)=2}}
```