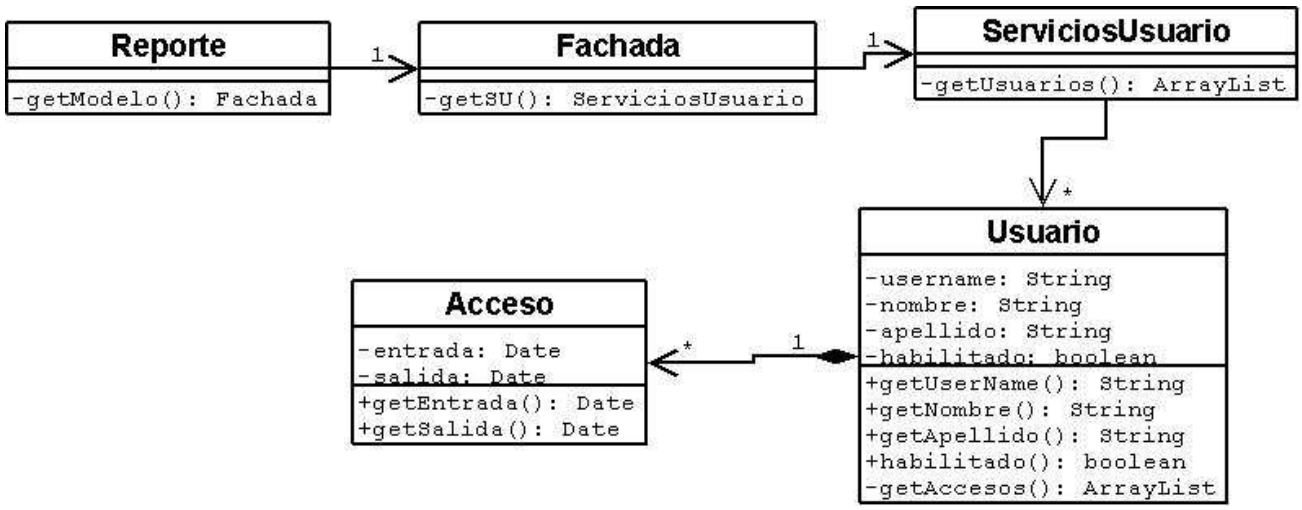
UNIVERSIDAD ORT
Uruguay

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel 902 15 05 Fax 908 13 70
www.ort.edu.uy

EVALUACION	EXAMEN	**GRUPO**	TODOS	**FECHA**	26/02/2013
MATERIA	DISEÑO Y DESARROLLO DE APLICACIONES				
CARRERA	ANALISTA EN TECNOLOGIAS DE INFORMACIÓN / ANALISTA PROGRAMADOR				
CONDICIONES	**- Puntos: 100** **- Duración 3 HORAS** **- Sin material**				

Ejercicio 1 (20 ptos)

Dado el siguiente modelo:



Implementar en Java el siguiente reporte:

Mostrar por consola aquellos usuarios que han accedido al sistema alguna vez antes de una fecha dada. Esta fecha será pasada por parámetro al reporte.
En el reporte se debe mostrar la siguiente información de los usuarios:
Nombre ,Apellido

Reglas:

- Se asume que todos los métodos y relaciones que figuran en el modelo ya están implementados.
- **No asuma** la existencia de **ningún** método que no figure en el modelo (incluyendo getter's y setter's)
- No es posible crear nuevas clases ni nuevas relaciones de ningún tipo, **salvo dependencias**
- No es posible agregar atributos a las clases.
- Se debe agregar el método que ejecuta el reporte (en la clase Reporte) y todos aquellos métodos que considere necesarios para colaborar con este. Incluya un comentario antes del cabezal de cada método que indique en que clase se implementa cada método.
- **No se considerará valida ninguna solución que viole el GRASP Experto en cualquiera de los métodos que participan en la solución. Tampoco se considerarán válidas aquellas soluciones que violen la división lógica establecida. Esto es, lógica en la interfaz de usuario (clase Reporte) o presentación de la información en el modelo (resto de las clases).**

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel 902 15 05 Fax 908 13 70
www.ort.edu.uy

- No es relevante realizar optimizaciones de performance.

NOTA: En el diagrama no figuran las operaciones necesarias para el mantenimiento de la información dado que son irrelevantes para el problema planteado.
Recuerde que para comparar fechas puede usar el método compareTo(…) de clase Date.

Ej: int r = f1.compareTo(f2); //si r es menor que cero f1 es menor que f2, si es mayor que cero f1 es mayor y si es cero es porque son iguales.

Solución:

a)

```
//Clase reporte
public void reporte(Date fHasta){
        ArrayList usuarios = getModelo().usuariosQueAccedieronAntesA(fHasta);
        for(int x=0;x<usuarios.size();x++){
                Usuario u = (Usuario) usuarios.get(x);
                System.out.println(u.getNombre() + "," + u.getApellido())
        }
}

//Clase Fachada
public ArrayList usuariosQueAccedieronAntesA(Date fHasta){

        return getSU().usuariosQueAccedieronAntesA(fHasta);
}

//Clase ControlUsuarios
public ArrayList usuariosQueAccedieronAntesA(Date fHasta){

        ArrayList usuarios = getUsuarios();
        ArrayList retorno = new ArrayList();
        for (int x=0;x<usuarios.size();x++){
                Usuario u = (Usuario) usuarios.get(x);
                if (u.accedioAntesQue(fHasta) {
                        retorno.add(u);
                }
        }
        return rertorno;
}

//Clase Usuario
public boolean accedioantesQue(Date fHasta) {
        ArrayList accesos = getAccesos();
        boolean r = false;
        for(int x=0;x<accesos.size() && !r;x++){

                if ( (Acceso)accesos.get(x)).anteriorA(fHasta)   r=true;
```

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel 902 15 05 Fax 908 13 70
www.ort.edu.uy

```
        }
        return r;
}

//clase Acceso
public boolean anteriorA(Date f){
        return getEntrada().compareTo(f) < 0;
```

Ejercicio 2 (20 ptos)

Dado el siguiente código:

```
public class B{
        public void z(){
                //5 líneas de código (distintas a las de z en C)
        }
        public void m(){
                //20 líneas de código, las mismas que el método m() de la clase C
        }
}
public class C{
        public void z(){
                //5 líneas de código (distintas a las de z en B)
        }
        public void m(){
                //20 líneas de código, las mismas que el método m() de la clase B
        }
}
```

Se desea modificar el diseño del mismo para lograr dos objetivos:

1) Eliminar la repetición del código del método m() , que es igual en B y en C.
2) Poder trabajar indistintamente con objetos B y C (polimorfismo)

Ejemplo :

```
        Public class Test{
                public static void main(String[] args){
                        prueba(new B());
                        prueba(new C());
                }
                public static void prueba (¿? param){
                        param.m(); //se ejecuta el código de m
                        param.z(); // se ejecuta el código de z de B en el primer caso y de C en el segundo
                }
        }
```

Se pide:
Implemente en java una nueva versión del código que **cumpla con los dos objetivos** planteados.
Para que la solución sea valida: **NO puede utilizar HERENCIA ni establecer ninguna relación entre B y C o viceversa.**

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel 902 15 05 Fax 908 13 70
www.ort.edu.uy

Solución:

```
public interface A{
        public void m();
        public void z();
}
public class M{
        public void m(){
                //20 líneas de código
        }
}
public class B implements A{
        private void M unM = new M();
        public void z(){
                //5 líneas de código
        }
        public void m(){
                unM.m();
        }
}
public class C implements A{
        private void M unM = new M();
        public void z(){
                // otras 5 líneas de código
        }
        public void m(){
                unM.m();
        }
}
```

Ejercicio 3 – (45 ptos)

Se desea implementar parte de un prototipo para un sistema que utiliza sensores.
El sistema trabaja con sensores de Movimiento, Presión y Temperatura que se encargan de detectar actividad de movimiento, cambio de presión y cambio de temperatura respectivamente.
Cada sensor debe poder ser prendido y apagado. Cuando se prende un sensor, debe automáticamente comenzar a detectar actividad (de movimiento, cambio de presión o cambio de temperatura según corresponda).
Todos los sensores prenden y apagan de la misma forma.
Además de cada sensor se conoce su id (int), modelo (String) y fecha de instalación (Date).

Se pide:

a) Realice un diagrama de clases para modelar esta situación, teniendo en cuenta que un sensor no puede cambiar la forma en que detecta actividad. **(5)**

b) Realice un diagrama de clases para modelar esta situación, teniendo en cuenta que un sensor debe poder cambiar la forma en que detecta actividad (de movimiento a temperatura, etc.) sin que esto implique perder su identidad como objeto; de forma que fácilmente pueda conservarse su id, modelo y fecha de instalación. **(10)**

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel 902 15 05 Fax 908 13 70
www.ort.edu.uy

c) Se desea que un conjunto indeterminado de alarmas emitan un sonido cada vez que algún sensor detecta actividad. A cada alarma se le debe poder especificar el sensor que hará que ella suene en caso de que este detecte actividad. Además, la alarma debe permitir que este sensor le sea cambiado en cualquier momento (por otro o por ninguno).
i) Realice el diagrama de clases que modela esta situación. **(10)**
ii) Implemente en java la clase Alarma. (para emular el método que emite el sonido use salida por consola) **(15)**

c) Reconoce algún patrón de diseño en la parte a, b o c ? En caso afirmativo indique su(s) nombre(s) **(5)**

Reglas:
- No es necesario modelar interfaces de usuario o clases de servicios. Solo las clases del dominio del problema descrito con los métodos correspondientes para satisfacer los requerimientos planteados.
TODOS LOS DIAGRAMAS DEBEN TENER NIVEL DE DETALLE ALTO

Solución:
a)
Herencia
b)
Strategy
c) Alarma observa Sensor. Cada Alarma tiene un sensor y un método cambiarSensor. En la implementación es necesario mostrar cuando la alarma se registra al sensor y cuando se des-registra(al cambiar uno por otro)
d) Template Method (prender()) en parte a, Strategy en parte b y Observer en parte c

Ejercicio 4 – (15 ptos)

Implementar en Java una clase de nombre X tal que:

- Cada instancia de X tiene un atributo "numero" de tipo int. Este valor se asigna en el momento de la creación del objeto y no puede ser cambiado posteriormente, si tiene que poder ser consultado.
- Se debe asegurar que solo la clase X pueda crear objetos X.
- La cantidad máxima de objetos X que la clase X creará es por defecto 3.
- Debe existir un método para acceder a las instancias de X. Este método retornará un objeto X. Para esto calculará un número aleatorio entre 0 y la cantidad máxima de objetos X. **Si no existe un objeto X con ese valor, creará un objeto, le asignará ese valor y retornará una referencia al objeto creado. Si ya existe un objeto X con ese valor retornará una referencia al objeto existente.**
- La cantidad máxima de objetos X podrá ser variada en cualquier momento mediante un método que deberá implementar a tales efectos. Cuando se varia la cantidad máxima, todos los objetos X creados hasta el momento se "pierden" o "eliminan" . (Al menos en lo que respecta a la clase X).

NOTAS:
- Solo debe implementar una clase: X
- Se recomienda modelar la solución como una variante del patrón Singleton
- Random r = new Random();
r.nextInt(n); -> retorna un numero aleatorio mayor que 0 y menor que n.

Solucion:

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel 902 15 05 Fax 908 13 70
www.ort.edu.uy

```
import java.util.Random;
public class X {
	private static X[] instancias = null;
	private static int cantidad = 3;
	private static Random generador = new Random();
	private int numero;
	public int getNumero(){
		return numero;
	}
	private X(int num){
		numero = num;
	}
	private static void setCantidad(int c){
		if (cantidad<1)return;
		cantidad = c;
		instancias = null;
	}

	public static X getInstancia(){
		if (instancias==null) instancias = new X[cantidad];
		int num = generador.nextInt(cantidad);
		X x = instancias[num];
		if (x==null){
			x = new X(num);
			instancias[num]=x;
		}
		return x;
	}
```