

Universidad de Buenos Aires

Facultad de Ingeniería

75.10 Técnicas de Diseño

Trabajo Práctico 1

GRUPO N° 20

Integrantes:

CHELOTTI, ADRIANA GRETTEL	83513	adrichelo84@yahoo.com.ar
PEREZ STALTARI, DARIO MARTIN	83514	dmpstaltari@yahoo.com.ar

Índice

Enunciado	3
Explicación	4
Diagrama de Clases	7
Diagrama de Secuencia	8
Documentación	9
Interface Accion	9
Clase Cancelador	9
Clase CanceladorPorDefecto	10
Clase CanceladorTotal.....	11
Clase Evaluador	12
Clase EvaluadorIgualdadConjunto	12
Clase EvaluadorPorDefecto.....	12
Clase EvaluadorSecuenciaContinua	12
Clase EvaluadorSecuenciaDiscontinua	13
Clase Implicacion	13
Clase ManejadorDeSucesos	14
Clase Suceso	16
Código Fuente	17
Pruebas	36

Enunciado

Diseño e implementación de una API para manejo de sucesos

Objetivo

Se pide diseñar y construir una API que permita coordinar sucesos ocurridos en un dominio cualquiera. La API debe permitir configurar para un suceso o conjunto de sucesos ocurridos una respuesta para aquellos objetos del dominio que reaccionan ante dichos sucesos. Así mismo objetos del dominio le informarán a la API cuando realizan sucesos.

Por ejemplo, dado un dominio donde hay un tanque y una bomba de agua. Podría haber un suceso llamado “límite inferior de tanque lleno” y otro “límite superior de tanque lleno”, que el tanque emite cuando su nivel de agua supera o es inferior a ciertos valores. Por otro lado la bomba espera dichos sucesos para encenderse o apagarse y mantener el tanque con agua.

Podría haber otro caso donde para que ocurra una acción sobre un objeto de dominio es necesario que sucedan un conjunto de sucesos, y quizás en cierto orden, por ejemplo volviendo al caso del tanque, podría haber otro objeto de dominio que mide la presión de la red de agua potable, que informa dos sucesos de acuerdo a su presión en la entrada (presión suficiente o presión insuficiente), y por lo que para que la bomba encienda, deben darse las dos condiciones que el tanque haya pasado su límite inferior y que la presión de agua sea la suficiente como para que la bomba trabaje sin romperse.

También debe poder configurarse para soportar sucesos que cancelan a otros sucesos. Por ejemplo, si ocurre un suceso de “límite inferior de tanque lleno” y luego ocurre otro de “límite superior de tanque lleno”, este segundo podría cancelar al primero, es decir el primero ya caducó o ya no es válido en este contexto.

El objetivo es diseñar y construir una API que permita manejar este tipo de sucesos. Que sea independiente del dominio que la utilice, y que soporte configurarse para distintos casos como un solo suceso, un conjunto de sucesos, un conjunto de sucesos en orden, con sucesos cancelables, etc.

Se pide entregar el código fuente, explicación y diagramas UML que muestren y expliquen el diseño tanto estático como dinámico de la solución, y un conjunto sólido de test unitarios que prueben los distintos casos soportados por la API.

Se evaluará en este TP, la correcta resolución del problema dado, los diagramas entregados, y la legibilidad y diseño del código y test unitarios.

Explicación

El Manejador de Suceso es una API que permite configurar sucesos ocurridos en un dominio cualquiera. Esta API permite que objetos puedan suscribir implicaciones a las mismas. Con implicaciones nos referimos a un conjunto de sucesos que tienen que ocurrir para que se produzca una acción de interés. Los objetos que quieran utilizar la API, deben crear dichos sucesos diferenciados por un id que los caractericen y pueden tener un id de un suceso que lo cancele. Además deben crear una clase que implemente la interfaz Accion. Dentro de esta acción se debe implementar la acción a efectuar por el objeto cliente.

El cliente tendrá la posibilidad de configurar la API, según el comportamiento que desee.

Tenemos 4 configuraciones posibles:

- Secuencia Continua
- Secuencia Discontinua
- Por defecto
- Conjunto Iguales

Secuencia Continua

En la configuración Secuencia Continua, se comparan los sucesos de las implicaciones suscriptas a la API con los sucesos ocurridos. Y la secuencia que sucesos deben coincidir en forma exacta.

Secuencia Discontinua

En la configuración Secuencia Discontinua, se comparan los sucesos de las implicaciones suscriptas a la API con los sucesos ocurridos. Y la secuencia que sucesos deben coincidir pero no en forma exacta. Podemos verlo con un ejemplo de esto último

Tenemos los siguientes sucesos ocurridos

“llueve” “haceCalor” “bajaPresion” “esTarde”

Y una implicación espera “llueve” “esTarde”. En este caso se cumple la secuencia, sin importa los sucesos intermedios.

Secuencia Por Defecto

En esta configuración, la comparación se da si el conjunto de sucesos esperados por la implicación está incluido en el conjunto de sucesos ocurridos.

Secuencia Igualdad conjunto

En esta configuración, la comparación se da si el conjunto de sucesos esperados por la implicación es igual al conjunto de sucesos ocurridos sin importar el orden.

Otra posibilidad que ofrece la API, es permitir habilitar o deshabilitar las cancelaciones

Los tipos de cancelaciones que se pueden realizar entre sucesos son.

- Por Defecto
- CanceladorTotal

Cancelador Por defecto

En este caso un suceso ocurrido se cancela con algún otro de la lista de sucesos.

Cancelador Total

En este caso un suceso ocurrido cancela con todos los sucesos que sean cancelables por este.

El objeto cliente puede agregar suceso o conjuntos de sucesos a la API; como así también notificar a la API que un suceso o conjunto de sucesos deben ser notificado. En este caso la API mirara las implicaciones inscriptas a la misma y mediante un evaluador corroborar si los sucesos ocurrieron o no. En caso de cumplir con los sucesos de la implicación según la configuración establecida procederá a ejecutar la acción de interés para el objeto cliente.

Diagrama de Clases

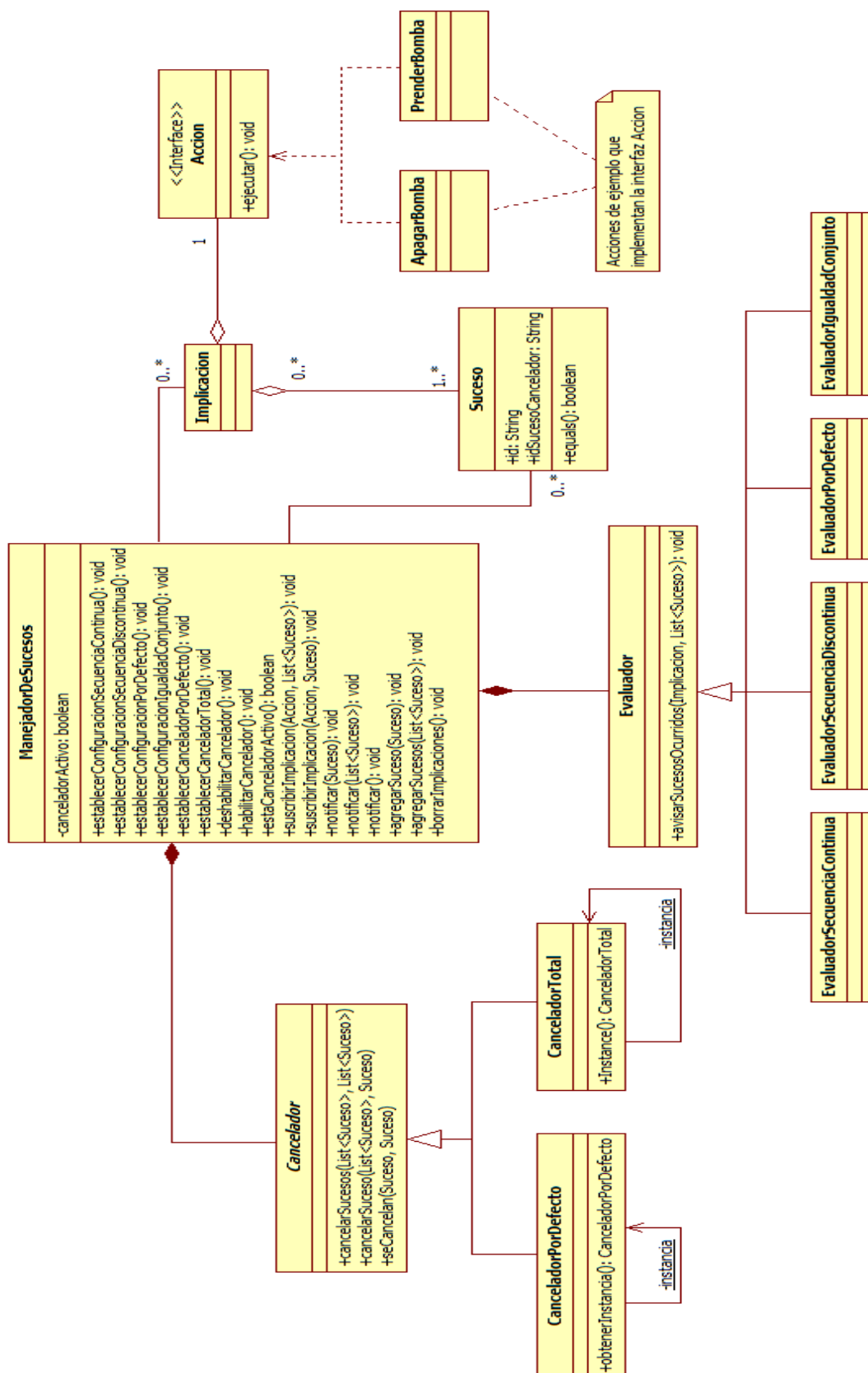
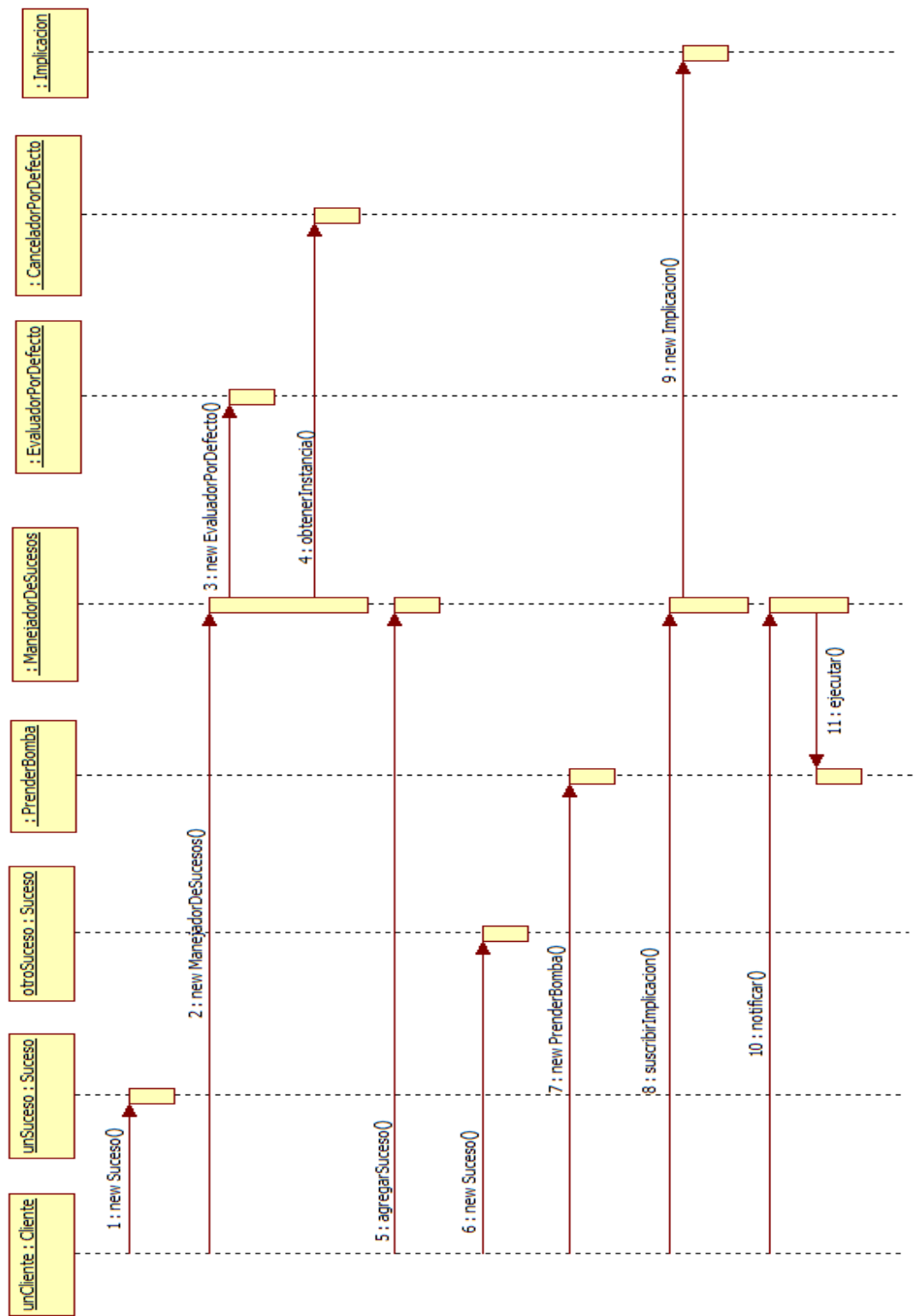


Diagrama de Secuencia



Documentación

Interface Accion

public interface **Accion**

Interfaz que deben implementar las acciones a efectuar.

Métodos

Void	ejecutar()
	Metodo en donde se deben implementar las acciones a realizar.

Clase Cancelador

public abstract class **Cancelador**

Clase abstracta que estable la estructura de un cancelador.

Metodos

abstract void	cancelarSuceso (List<Suceso> sucesosExistentes, Suceso sucesoNuevo)
	Cancela los sucesos cancelables entre el conjunto y el suceso pasados como parámetro. Parametros: sucesosExistentes - conjunto de sucesos existentes. sucesosNuevos - conjunto de sucesos nuevos.
abstract void	cancelarSucesos (<Suceso> sucesosExistentes, List<Suceso> sucesosNuevos)
	Cancela los sucesos cancelables entre los conjuntos pasados como parametro. Parametros: sucesosExistentes - conjunto de sucesos existentes. sucesosNuevos - conjunto de sucesos nuevos.

Clase CanceladorPorDefecto

public class **CanceladorPorDefecto** extends Cancelador

Clase que modela un cancelador, las cancelaciones de sucesos cancelables entre si se producen uno a uno.

Métodos	
void	<p>cancelarSuceso(List<Suceso> sucesosExistentes, Suceso sucesoNuevo)</p> <p>Cancela los sucesos cancelables entre el conjunto y el suceso pasados como parametro,</p> <p>Parámetros: sucesosExistentes - conjunto de sucesos existentes. sucesoNuevo - suceso nuevo.</p>
void	<p>cancelarSucesos(List<Suceso> sucesosExistentes, List<Suceso> sucesosNuevos)</p> <p>Cancela los sucesos cancelables entre los conjuntos pasados como parametro.</p> <p>Parámetros: sucesosExistentes - conjunto de sucesos existentes. sucesosNuevos - conjunto de sucesos nuevos.</p>
static CanceladorPorDefecto	<p>obtenerInstancia()</p> <p>Obtiene la instancia de la clase</p> <p>Retorna: instancia del cancelador por defecto.</p>

Clase CanceladorTotal

public class **CanceladorTotal** extends Cancelador

Clase que modela un cancelador, se produce la cancelacion de todos los sucesos que sean cancelables.

Métodos	
void	cancelarSuceso (List<Suceso> sucesosExistentes, Suceso sucesoNuevo) Cancela los sucesos cancelables entre el conjunto y el suceso pasados como parametro, Parámetros: sucesosExistentes - conjunto de sucesos existentes. sucesoNuevo - suceso nuevo.
void	cancelarSucesos (List<Suceso> sucesosExistentes, List<Suceso> sucesosNuevos) Cancela los sucesos cancelables entre los conjuntos pasados como parametro. Parámetros: sucesosExistentes - conjunto de sucesos existentes. sucesosNuevos - conjunto de sucesos nuevos.
static CanceladorPorTotal	obtenerInstancia () Obtiene la instancia de la clase Retorna: instancia del cancelador total.

Clase Evaluador

```
public abstract class Evaluador
```

Clase abstracta que estable la estructura de un evaluador. Evalua un conjunto de sucesos con el antecedente de una implicacion en caso de cumplirse se ejecuta el consecuente.

Metodos	
abstract void	avisarSucesosOcurridos (Implicacion implicacion, List<Suceso> sucesos) Evalua si los sucesos cumplen con el antecedente de la implicacion. En caso afirmativo se ejecuta la accion de la implicacion. Parámetros: implicacion - implicacion a evaluar. sucesos - sucesos a evaluar.

Clase EvaluadorIgualdadConjunto

```
public class EvaluadorIgualdadConjunto extends Evaluador
```

Clase que modela un evaluador. Evalua un conjunto de sucesos con el antecedente de una implicacion en el caso de que los conjuntos sean iguales se ejecuta la accion del consecuente.

Clase EvaluadorPorDefecto

```
public class EvaluadorPorDefecto extends Evaluador
```

Clase que modela un evaluador. Evalua un conjunto de sucesos con el antecedente de una implicacion, en el caso de que los sucesos a evaluar esten incluidos en el antecedente se ejecuta la accion del consecuente.

Clase EvaluadorSecuenciaContinua

```
public class EvaluadorSecuenciaContinua extends Evaluador
```

Clase que modela un evaluador. Evalua un conjunto de sucesos con el antecedente de una implicacion, en el caso de que los sucesos a evaluar esten incluidos en el antecedente formando una secuencia continua se ejecuta la accion del consecuente.

Clase EvaluadorSecuenciaDiscontinua

public class **EvaluadorSecuenciaDiscontinua** extends Evaluador

Clase que modela un evaluador. Evalua un conjunto de sucesos con el antecedente de una implicacion, en el caso de que los sucesos a evaluar esten incluidos en el antecedente formando una secuencia se ejecuta la accion del consecuente.

Clase Implicacion

public class **Implicacion**

Clase que modela un implicacion. Una implicacion esta formada por un antecedente compuesto por un conjunto de sucesos y un consecuente compuesto por una accion. En caso de que se cumpla con el antecedente se ejecutara el consecuente.

Métodos	
Accion	getAccion() Obtiene la accion del consecuente. Retorna: accion que forma el consecuente.
List<Suceso>	getSucesos() Obtiene el conjunto de sucesos del antecedente. Retorna: conjuntos de sucesos que forman el antecedente.
void	setAccion(Accion accion) Carga la accion del consecuente. Parámetros: accion - accion a cargarse.
void	setSucesos(List<Suceso> sucesos) Carga la lista de sucesos del antecedente. Parámetros: sucesos - conjuntos de sucesos a cargarse

Clase ManejadorDeSucesos

public class **ManejadorDeSucesos**

Clase encargada del manejo de sucesos.

Metodo	
void	agregarSuceso (Suceso sucesoAgregar) Agrega un suceso al conjunto de sucesos pendientes de notificacion. Parámetros: sucesoAgregar - suceso a agregar.
void	agregarSucesos (List<Suceso> sucesosAgregar) Agrega un conjunto de sucesos al conjunto de sucesos pendientes notificacion. Parámetros: sucesosAgregar - conjunto de sucesos a agregar.
void	borrarImplicaciones () Elimina las implicaciones almacenadas.
void	deshabilitarCancelador () Deshabilita la cancelacion de sucesos.
void	establecerCanceladorPorDefecto () Establece el cancelador por defecto.
void	establecerCanceladorTotal () Establece el cancelador total.
void	establecerConfiguracionIgualdadConjunto () Cambia la configuracion a igualdad de conjunto de sucesos.
void	establecerConfiguracionPorDefecto () Cambia la configuracion a secuencia por defecto de sucesos.
void	establecerConfiguracionSecuenciaContinua () Cambia la configuracion a secuencia continua de sucesos.
void	establecerConfiguracionSecuenciaDiscontinua ()

	Cambia la configuracion a secuencia discontinua de sucesos.
boolean	estaCanceladorActivo() Obtiene el estado del cancelador.
void	habilitarCancelador() Habilita la cancelacion de sucesos.
void	notificar() Notifica los sucesos pendientes.
void	notificar (java.util.List<Suceso> sucesos) Notifica los sucesos pendientes junto con los pasados como parametro. Parámetros: sucesos - sucesos a notificar.
void	notificar (Suceso sucesoActual) Notifica los sucesos pendientes junto con el pasado como parametro. Parámetros: sucesoAgregar - suceso a agregar.
void	suscribirImplicacion (Accion accionCliente, List<Suceso> sucesos) Suscribe una implicacion a evaluar.
void	suscribirImplicacion (Accion accionCliente, Suceso suceso) Suscribe una implicacion a evaluar.

Clase Suceso

public class **Suceso**

Clase que modela un suceso.

Constructores	
	Suceso (String idSuceso) Constructor. Parámetros: idSuceso - identificador del suceso.
	Suceso (String idSuceso, String idSucesoCancelador) Constructor. Parámetros: idSuceso - identificador del suceso. idSucesoCancelador - identificador del suceso con el que se cancela.
Métodos	
String	getIdSuceso() Obtiene el identificador del suceso. Retorna: identificador del suceso.
String	getIdSucesoCancelador() Obtiene el identificador del suceso que lo cancela. Retorna: identificador del suceso con el que se cancela.

Código Fuente

Accion.java

```
package source;
/**
 * Interfaz que deben implementar las acciones a efectuar.
 *
 * @author Grupo20
 *
 */
public interface Accion {

    /**
     * Metodo en donde se deben implementar las acciones a realizar.
     */
    public void ejecutar();
}
```

Cancelador.java

```
package source;
import java.util.List;

/**
 * Clase abstracta que estable la estructura de un cancelador.
 *
 * @author Grupo20
 *
 */
public abstract class Cancelador {

    /**
     * Cancela los sucesos cancelables entre los conjuntos pasados como parametro.
     * @param sucesosExistentes conjunto de sucesos existentes.
     * @param sucesosNuevos conjunto de sucesos nuevos.
     */
    public abstract void cancelarSucesos(List<Suceso> sucesosExistentes, List<Suceso>
                                         sucesosNuevos);

    /**
     * Cancela los sucesos cancelables entre el conjunto y el suceso pasados como parametro,
     * @param sucesosExistentes conjunto de sucesos existentes.
     * @param sucesoNuevo suceso nuevo.
     */
    public abstract void cancelarSuceso(List<Suceso> sucesosExistentes, Suceso sucesoNuevo);
```

```

/**
 * Determina si dos sucesos son cancelables entre si.
 * @param suceso1 suceso a evaluar.
 * @param suceso2 suceso a evaluar.
 * @return true si los sucesos se cancelan.
 *         false en caso contrario.
 */
protected boolean seCancelan(Suceso suceso1, Suceso suceso2)
{
    String idSuceso1 = suceso1.getIdSuceso();
    String idSuceso2 = suceso2.getIdSuceso();
    String idCanceladorSuceso1 = suceso1.getIdSucesoCancelador();
    String idCanceladorSuceso2 = suceso2.getIdSucesoCancelador();

    if(idCanceladorSuceso1==null && idCanceladorSuceso2==null)
        return false;

    if(idCanceladorSuceso1!=null && idCanceladorSuceso1.equals(idSuceso2))
        return true;

    if(idCanceladorSuceso2!=null && idCanceladorSuceso2.equals(idSuceso1))
        return true;

    return false;
}
}

```

CanceladorPorDefecto.java

```
package source;

import java.util.Iterator;
import java.util.List;

/**
 * Clase que modela un cancelador, las cancelaciones de sucesos cancelables
 * entre si se producen uno a uno.
 *
 * @author Grupo20
 */
public class CanceladorPorDefecto extends Cancelador {

    /**
     * Instancia unica de la clase.
     */
    private static CanceladorPorDefecto INSTANCIA = new CanceladorPorDefecto();

    /**
     * Constructor
     */
    private CanceladorPorDefecto(){}

    /**
     * Obtiene la instancia de la clase.
     * @return instancia del cancelador por defecto.
     */
    public static CanceladorPorDefecto obtenerInstancia(){
        return CanceladorPorDefecto.INSTANCIA;
    }

    @Override
    public void cancelarSucesos(List<Suceso> sucesosExistentes, List<Suceso> sucesosNuevos) {
        for(Suceso sucesoActual: sucesosNuevos){
            cancelarSuceso(sucesosExistentes, sucesoActual);
        }
    }

    @Override
    public void cancelarSuceso(List<Suceso> sucesosExistentes, Suceso sucesoNuevo){
        Iterator<Suceso> it = sucesosExistentes.iterator();
        Suceso sucesoCancelable = null;
        boolean cancelacion = false;
        while(it.hasNext() && !cancelacion){
```

```

        sucesoCancelable = it.next();
        cancelacion = seCancelan(sucesoCancelable, sucesoNuevo);
    }
    if(cancelacion) sucesosExistentes.remove(sucesoCancelable);
}
}

```

CanceladorTotal.java

```

package source;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

/**
 * Clase que modela un cancelador, se produce la cancelacion de todos los sucesos
 * que sean cancelables.
 *
 * @author Grupo20
 */
public class CanceladorTotal extends Cancelador {

    /**
     * Instancia unica de la clase.
     */
    private static CanceladorTotal INSTANCIA = new CanceladorTotal();

    /**
     * Constructor
     */
    private CanceladorTotal(){}

    /**
     * Obtiene la instancia de la clase.
     * @return instancia del cancelador total.
     */
    public static CanceladorTotal obtenerInstancia(){
        return CanceladorTotal.INSTANCIA;
    }

    @Override
    public void cancelarSucesos(List<Suceso> sucesosExistentes, List<Suceso> sucesosNuevos){
        for(Suceso sucesoActual: sucesosNuevos){
            cancelarSuceso(sucesosExistentes, sucesoActual);
        }
    }
}

```

```

    }

    @Override
    public void cancelarSuceso(List<Suceso> sucesosExistentes,Suceso sucesoNuevo) {
        Iterator<Suceso> it = sucesosExistentes.iterator();
        Suceso sucesoCancelable = null;
        boolean cancelacion = false;
        List<Suceso> listaSucesosCancelables = new ArrayList<Suceso>();
        while(it.hasNext()){
            sucesoCancelable = it.next();
            if(seCancelan(sucesoCancelable, sucesoNuevo)){
                listaSucesosCancelables.add(sucesoCancelable);
                cancelacion = true;
            }
        }
        if(cancelacion){
            List<Suceso> listaAux = new ArrayList<Suceso>();
            listaAux.addAll(sucesosExistentes);
            listaAux.retainAll(listaSucesosCancelables);
            sucesosExistentes.removeAll(listaAux);
        }
    }
}

```

Evaluador.java

```
package source;

import java.util.List;

/**
 * Clase abstracta que estable la estructura de un evaluador.
 * Evalua un conjunto de sucesos con el antecedente de una implicacion en
 * caso de cumplirse se ejecuta el consecuente.
 *
 * @author Grupo20
 */
public abstract class Evaluador {

    /**
     * Evalua si los sucesos cumplen con el antecedente de la implicacion.
     * En caso afirmativo se ejecuta la accion de la implicacion.
     * @param implicacion implicacion a evaluar.
     * @param sucesos sucesos a evaluar.
     */
    public abstract void avisarSucesosOcurridos(Implicacion implicacion, List<Suceso>sucesos);

}
```

EvaluadorIgualdadConjunto.java

```
package source;

import java.util.List;

/**
 * Clase que modela un evaluador.
 * Evalua un conjunto de sucesos con el antecedente de una implicacion en el
 * caso de que los conjuntos sean iguales se ejecuta la accion del consecuente.
 *
 * @author Grupo20
 */
public class EvaluadorIgualdadConjunto extends Evaluador {

    /**
     * Instancia unica de la clase.
     */
    private static EvaluadorIgualdadConjunto INSTANCIA = new EvaluadorIgualdadConjunto();

    /**
```

```

    * Constructor
    */
    private EvaluadorIgualdadConjunto(){}

    /**
     * Obtiene la instancia de la clase.
     * @return instancia del evaluador igualdad conjunto.
     */
    public static EvaluadorIgualdadConjunto obtenerInstancia(){
        return EvaluadorIgualdadConjunto.INSTANCEIA;
    }

    @Override
    public void avisarSucesosOcurridos(Implicacion implicacion, List<Suceso> sucesos) {
        if (sucesos.containsAll(implicacion.getSucesos())
            &&(implicacion.getSucesos().size() == sucesos.size()))
            implicacion.getAccion().ejecutar();
    }
}

```

EvaluadorPorDefecto.java

```

package source;

import java.util.List;

/**
 * Clase que modela un evaluador.
 * Evalua un conjunto de sucesos con el antecedente de una implicacion, en el
 * caso de que los sucesos a evaluar esten incluidos en el antecedente se
 * ejecuta la accion del consecuente.
 *
 * @author Grupo20
 */
public class EvaluadorPorDefecto extends Evaluador {

    /**
     * Instancia unica de la clase.
     */
    private static EvaluadorPorDefecto INSTANCIA = new EvaluadorPorDefecto();

    /**
     * Constructor
     */
    private EvaluadorPorDefecto(){}

    /**

```

```

        * Obtiene la instancia de la clase.
        * @return instancia del evaluador por defecto.
        */
    public static EvaluadorPorDefecto obtenerInstancia(){
        return EvaluadorPorDefecto.INSTANCIA;
    }

    @Override
    public void avisarSucesosOcurridos(Implicacion implicacion, List<Suceso> sucesos) {
        if (sucesos.containsAll(implicacion.getSucesos()))
            implicacion.getAccion().ejecutar();
    }
}

```

EvaluadorSecuenciaContinua.java

```

package source;

import java.util.ArrayList;
import java.util.List;

/**
 * Clase que modela un evaluador.
 * Evalua un conjunto de sucesos con el antecedente de una implicacion, en el
 * caso de que los sucesos a evaluar esten incluidos en el antecedente formando una
 * secuencia continua se ejecuta la accion del consecuente.
 *
 * @author Grupo20
 */
public class EvaluadorSecuenciaContinua extends Evaluador {

    /**
     * Instancia unica de la clase.
     */
    private static EvaluadorSecuenciaContinua INSTANCIA=new EvaluadorSecuenciaContinua();

    /**
     * Constructor
     */
    private EvaluadorSecuenciaContinua(){}

    /**
     * Obtiene la instancia de la clase.
     * @return instancia del evaluador de secuencia continua.
     */
    public static EvaluadorSecuenciaContinua obtenerInstancia(){

```



```

        return EvaluadorSecuenciaContinua.INSTANCE;
    }

    @Override
    public void avisarSucesosOcurridos(Implicacion implicacion, List<Suceso> sucesos) {

        int tamanoAntecedente = implicacion.getSucesos().size();
        int tamanoSucesos = sucesos.size();

        if(tamanoAntecedente == tamanoSucesos){
            if (implicacion.getSucesos().equals(sucesos))
                implicacion.getAccion().ejecutar();
        }
        else if(tamanoSucesos >= tamanoAntecedente)
        {
            int posicionInicio = 0;
            int posicionFin = 0;
            boolean valido = true;
            List<Suceso> listAux = new ArrayList<Suceso>();

            while(valido)
            {
                posicionFin = posicionInicio + tamanoAntecedente;
                if(posicionFin<=tamanoSucesos){
                    listAux = sucesos.subList(posicionInicio, posicionFin);
                    if (implicacion.getSucesos().equals(listAux)){
                        implicacion.getAccion().ejecutar();
                        valido= false;
                    }
                    posicionInicio++;
                }
                else
                    valido = false;
            }
        }
    }
}

```

EvaluadorSecuenciaDiscontinua.java

```
package source;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

/**
 * Clase que modela un evaluador.
 * Evalua un conjunto de sucesos con el antecedente de una implicacion, en el
 * caso de que los sucesos a evaluar esten incluidos en el antecedente formando una
 * secuencia se ejecuta la accion del consecuente.
 *
 * @author Grupo20
 */
public class EvaluadorSecuenciaDiscontinua extends Evaluador {

    /**
     * Instancia unica de la clase.
     */
    private static EvaluadorSecuenciaDiscontinua INSTANCIA=new EvaluadorSecuenciaDiscontinua();

    /**
     * Constructor
     */
    private EvaluadorSecuenciaDiscontinua(){}

    /**
     * Obtiene la instancia de la clase.
     * @return instancia del evaluador de secuencia discontinua.
     */
    public static EvaluadorSecuenciaDiscontinua obtenerInstancia(){
        return EvaluadorSecuenciaDiscontinua.INSTANCIA;
    }

    @Override
    public void avisarSucesosOcurridos(Implicacion implicacion, List<Suceso> sucesos) {

        int tamanoAntecedente = implicacion.getSucesos().size();
        int tamanoSucesos = sucesos.size();

        if(tamanoAntecedente == tamanoSucesos){
            if (implicacion.getSucesos().equals(sucesos))
                implicacion.getAccion().ejecutar();
        }
    }
}
```

```

    }
    else if(tamanoSucesos >= tamanoAntecedente){
        int posicionInicio = 0;
        boolean encontrado = true;
        List<Suceso> listAux = new ArrayList<Suceso>();

        Iterator<Suceso> it = implicacion.getSucesos().iterator();
        Suceso sucesoAux = null;

        while(it.hasNext() && encontrado){
            sucesoAux = it.next();
            listAux = sucesos.subList(posicionInicio, tamanoSucesos);
            posicionInicio = listAux.indexOf(sucesoAux);
            if(posicionInicio<0)
                encontrado = false;
            else{
                posicionInicio++;
                if(posicionInicio>=tamanoSucesos)
                    encontrado = false;
            }
        }
        if(encontrado)
            implicacion.getAccion().ejecutar();
    }
}
}
}

```

Implicacion.java

```

package source;
import java.util.ArrayList;
import java.util.List;
/**
 * Clase que modela un implicacion.
 * Una implicacion esta formada por un antecedente compuesto por un conjunto de
 * sucesos y un consecuente compuesto por una accion.
 * En caso de que se cumpla con el antecedente se ejecutara el consecuente.
 *
 * @author Grupo20
 */
public class Implicacion {

    /**
     * Accion que representa al consecuente.
     */

```

```

private Accion accion;

/**
 * Conjunto de sucesos que representan al antecedente.
 */
private List<Suceso> sucesos = new ArrayList<Suceso>();

/**
 * Obtiene el conjunto de sucesos del antecedente.
 * @return conjuntos de sucesos que forman el antecedente.
 */
public List<Suceso> getSucesos() {
    return sucesos;
}

/**
 * Carga la lista de sucesos del antecedente.
 * @param sucesos conjuntos de sucesos a cargarse.
 */
public void setSucesos(List<Suceso> sucesos) {
    this.sucesos = sucesos;
}

/**
 * Obtiene la accion del consecuente.
 * @return accion que forma el consecuente.
 */
public Accion getAccion() {
    return accion;
}

/**
 * Carga la accion del consecuente.
 * @param accion accion a cargarse.
 */
public void setAccion(Accion accion) {
    this.accion = accion;
}
}

```

ManejadorDeSucesos.java

```
package source;

import java.util.ArrayList;
import java.util.List;

/**
 *
 * Clase encargada del manejo de sucesos.
 *
 * @author Grupo20
 *
 */
public class ManejadorDeSucesos {

    /**
     * Evaluador de sucesos ocurridos.
     */
    private Evaluador evaluador;

    /**
     * Cancelador de sucesos.
     */
    private Cancelador cancelador;

    /**
     * Estado del cancelador.
     */
    private boolean canceladorActivo;

    /**
     * Estado de etapa de notificacion.
     */
    private boolean notificado = false;

    /**
     * Implicaciones agregadas.
     */
    private List<Implicacion> implicaciones;

    /**
     * Sucesos no notificados.
     */
    private List<Suceso> sucesosPendientesNotificacion;
```

```

/**
 * Constructor de la clase.
 */
public ManejadorDeSucesos(){
    implicaciones = new ArrayList<Implicacion>();
    sucesosPendientesNotificacion = new ArrayList<Suceso>();
    evaluador = EvaluadorPorDefecto.obtenerInstancia();
    cancelador = CanceladorPorDefecto.obtenerInstancia();
}

/**
 * Cambia la configuracion a secuencia continua de sucesos.
 */
public void establecerConfiguracionSecuenciaContinua(){
    this.evaluador = EvaluadorSecuenciaContinua.obtenerInstancia();
}

/**
 * Cambia la configuracion a secuencia discontinua de sucesos.
 */
public void establecerConfiguracionSecuenciaDiscontinua(){
    this.evaluador = EvaluadorSecuenciaDiscontinua.obtenerInstancia();
}

/**
 * Cambia la configuracion a secuencia por defecto de sucesos.
 */
public void establecerConfiguracionPorDefecto(){
    this.evaluador = EvaluadorPorDefecto.obtenerInstancia();
}

/**
 * Cambia la configuracion a igualdad de conjunto de sucesos.
 */
public void establecerConfiguracionIgualdadConjunto(){
    this.evaluador = EvaluadorIgualdadConjunto.obtenerInstancia();
}

/**
 * Establece el cancelador por defecto.
 * El modo de cancelacion por defecto proporciona la cancelacion uno a uno de
 * sucesos cancelables.
 */
public void establecerCanceladorPorDefecto(){
    this.cancelador = CanceladorPorDefecto.obtenerInstancia();
}

/**

```

```

* Establece el cancelador total.
* El modo de cancelacion total proporciona la cancelacion de todos los sucesos
* que sean cancelables.
*/
public void establecerCanceladorTotal(){
    this.cancelador = CanceladorTotal.obtenerInstancia();
}

/**
* Deshabilita la cancelacion de sucesos.
*/
public void deshabilitarCancelador(){
    this.canceladorActivo = false;
}

/**
* Habilita la cancelacion de sucesos.
*/
public void habilitarCancelador(){
    this.canceladorActivo = true;
}

/**
* Obtiene el estado del cancelador.
* @return true si el cancelador esta activo.
*         false en caso contrario.
*/
public boolean estaCanceladorActivo(){
    return this.canceladorActivo;
}

/**
* Suscribe una implicacion a evaluar.
* @param accionCliente accion a ejecutar si se cumplen los sucesos.
* @param sucesos conjuntos de sucesos a evaluar.
*/
public void suscribirImplicacion(Accion accionCliente, List<Suceso> sucesos){
    //Si la lista de sucesos es nula no suscribimos la implicacion
    if (sucesos!=null){
        //sacamos los suceso que son nulos de la lista
        this.eliminarSucesosNulos(sucesos);
        Implicacion nuevaImplicacion = new Implicacion();
        nuevaImplicacion.setAccion(accionCliente);
        nuevaImplicacion.setSucesos(sucesos);
        this.implicaciones.add(nuevaImplicacion);
    }
}

```

```

/**
 * Suscribe una implicacion a evaluar.
 * @param accionCliente accion a ejecutar si se cumple el suceso.
 * @param suceso suceso a evaluar.
 */
public void suscribirImplicacion(Accion accionCliente, Suceso suceso){
    //Si el suceso es nulo no suscribimos la implicacion
    if(suceso!=null){
        List<Suceso> lista = new ArrayList<Suceso>();
        lista.add(suceso);
        this.suscribirImplicacion(accionCliente, lista);
    }
}

/**
 * Notifica los sucesos pendientes junto con el pasado como parametro.
 * @param sucesoActual suceso a notificar.
 */
public void notificar(Suceso sucesoActual){
    if (notificado!=true){
        notificado=true;
        this.agregarSuceso(sucesoActual);
        for (Implicacion relacion : this.implicaciones) {
            this.evaluador.avisarSucesosOcurridos(relacion,
                this.sucesosPendientesNotificacion);
        }
        this.sucesosPendientesNotificacion.clear();
        notificado=false;
    }
}

/**
 * Notifica los sucesos pendientes junto con los pasados como parametro.
 * @param sucesos sucesos a notificar.
 */
public void notificar(List<Suceso> sucesos){
    this.agregarSucesos(sucesos);
    for (Implicacion relacion : this.implicaciones) {
        this.evaluador.avisarSucesosOcurridos(relacion,
            this.sucesosPendientesNotificacion);
    }
    this.sucesosPendientesNotificacion.clear();
}

/**
 * Notifica los sucesos pendientes.
 */
public void notificar(){

```



```

        if (notificado!=true){
            notificado=true;
            for (Implicacion relacion : this.implicaciones) {
                this.evaluador.avisarSucesosOcurridos(relacion,
                    this.sucesosPendientesNotificacion);
            }
            notificado=false;
            this.sucesosPendientesNotificacion.clear();
        }
    }

    /**
     * Agrega un suceso al conjunto de sucesos pendientes de notificacion.
     * @param sucesoAgregar suceso a agregar.
     */
    public void agregarSuceso(Suceso sucesoAgregar){
        if (sucesoAgregar!=null){
            if(canceladorActivo)
                this.cancelador.cancelarSuceso(this.sucesosPendientesNotificacion
                    ,sucesoAgregar);
            this.sucesosPendientesNotificacion.add(sucesoAgregar);
        }
    }

    /**
     * Agrega un conjunto de sucesos al conjunto de sucesos pendientes notificacion.
     * @param sucesosAgregar conjunto de sucesos a agregar.
     */
    public void agregarSucesos(List<Suceso> sucesosAgregar){
        //controlo que la lista no sea null
        if (sucesosAgregar!=null){
            //Puede contener elementos que sean nulos, entonces los sacamos
            this.eliminarSucesosNulos(sucesosAgregar);
            if(canceladorActivo)
                this.cancelador.cancelarSucesos(this.sucesosPendientesNotificacion,
                    sucesosAgregar);
            this.sucesosPendientesNotificacion.addAll(sucesosAgregar);
        }
    }

    /**
     * Elimina las implicaciones almacenadas.
     */
    public void borrarImplicaciones(){
        this.sucesosPendientesNotificacion.clear();
    }

    /**

```

```

        * Elimina los elementos nulos de un conjunto de sucesos.
        */
        private void eliminarSucesosNulos(List<Suceso> listaSucesos){
            List<Suceso> listaCopia = new ArrayList<Suceso>();
            for (Suceso suceso : listaSucesos) {
                if(suceso!=null){
                    listaCopia.add(suceso);
                }
            }
            listaSucesos=listaCopia;
        }
    }
}

```

Suceso.java

```

package source;

/**
 * Clase que modela un suceso.
 *
 * @author Grupo20
 */
public class Suceso {

    /**
     * Identificador del suceso.
     */
    private String idSuceso;

    /**
     * Identificador del suceso con el que se cancela.
     */
    private String idSucesoCancelador;

    /**
     * Constructor.
     * @param idSuceso identificador del suceso.
     */
    public Suceso(String idSuceso)
    {
        this.idSuceso = idSuceso;
    }

    /**
     * Constructor.
     * @param idSuceso identificador del suceso.
     */
}

```

```

    * @param idSucesoCancelador identificador del suceso con el que se cancela.
    */
    public Suceso(String idSuceso, String idSucesoCancelador)
    {
        this(idSuceso);
        this.idSucesoCancelador = idSucesoCancelador;
    }

    /**
     * Obtiene el identificador del suceso que lo cancela.
     * @return identificador del suceso con el que se cancela.
     */
    public String getIdSucesoCancelador() {
        return idSucesoCancelador;
    }

    /**
     * Obtiene el identificador del suceso.
     * @return identificador del suceso.
     */
    public String getIdSuceso() {
        return idSuceso;
    }

    @Override
    public boolean equals(Object obj) {
        if (((Suceso)obj).getIdSuceso().equals(this.idSuceso))
            return true;
        else
            return false;
    }

    @Override
    public String toString() {
        return idSuceso+"-"+idSucesoCancelador;
    }
}

```

Pruebas

TestCancelador.java

```
package prueba;

import junit.framework.TestCase;
import source.Suceso;

/**
 * Test de funcionamiento del cancelador.
 * @author Dario
 */
public class TestCancelador extends TestCase {

    @Override
    protected void setUp() throws Exception {
        super.setUp();
    }

    private boolean seCancelan(Suceso suceso1, Suceso suceso2)
    {
        String idSuceso1 = suceso1.getIdSuceso();
        String idSuceso2 = suceso2.getIdSuceso();
        String idCanceladorSuceso1 = suceso1.getIdSucesoCancelador();
        String idCanceladorSuceso2 = suceso2.getIdSucesoCancelador();

        if(idCanceladorSuceso1==null && idCanceladorSuceso2==null)
            return false;

        if(idCanceladorSuceso1!=null && idCanceladorSuceso1.equals(idSuceso2))
            return true;

        if(idCanceladorSuceso2!=null && idCanceladorSuceso2.equals(idSuceso1))
            return true;

        return false;
    }

    public void testFuncionSeCancelan(){

        Suceso suceso1 = new Suceso("A","B");
        Suceso suceso2 = new Suceso("C","B");
```

```
assertEquals(false, seCancelan(suceso1, suceso2));

Suceso suceso3 = new Suceso("B","D");
assertEquals(true, seCancelan(suceso1, suceso3));
assertEquals(true, seCancelan(suceso2, suceso3));

Suceso suceso4 = new Suceso("D","B");
assertEquals(true, seCancelan(suceso3, suceso4));


Suceso suceso5 = new Suceso("D");
assertEquals(true, seCancelan(suceso3, suceso5));
assertEquals(false,seCancelan(suceso4, suceso5));

Suceso suceso6 = new Suceso("A");
assertEquals(false, seCancelan(suceso5, suceso6));

Suceso suceso7 = new Suceso("B");
assertEquals(true, seCancelan(suceso1, suceso7));

    }

}
```



TestCanceladorPorDefecto.java

```
package prueba;

import java.util.ArrayList;
import java.util.List;

import junit.framework.TestCase;
import source.CanceladorPorDefecto;
import source.Suceso;

/**
 * Test de funcionamiento del cancelador por defecto.
 *
 * @author Grupo20
 */
public class TestCanceladorPorDefecto extends TestCase {

    private CanceladorPorDefecto cancelador;

    private List<Suceso> sucesosExistentes;

    private List<Suceso> sucesosNuevos;

    private List<Suceso> sucesosResultantes;

    private Suceso suceso;

    protected void setUp() throws Exception {
        super.setUp();
        this.cancelador = CanceladorPorDefecto.obtenerInstancia();
        this.sucesosExistentes = new ArrayList<Suceso>();
        this.sucesosNuevos = new ArrayList<Suceso>();
        this.sucesosResultantes = new ArrayList<Suceso>();
    }

    public void testCancelarSuceso(){

        this.suceso = new Suceso("pocaAgua");
        sucesosExistentes.add(new Suceso("pocaAgua","muchagua"));
        sucesosExistentes.add(new Suceso("muchapresion","pocapresion"));
        sucesosExistentes.add(new Suceso("muchagua","pocaagua"));
        sucesosExistentes.add(new Suceso("pocaagua","muchagua"));
        sucesosExistentes.add(new Suceso("muchagua"));

        cancelador.cancelarSuceso(sucesosExistentes, suceso);

        sucesosResultantes.add(new Suceso("pocaagua","muchagua"));
    }
}
```

```

sucesosResultantes.add(new Suceso("muchPresion","pocaPresion"));
sucesosResultantes.add(new Suceso("pocaAgua","muchAgua"));
sucesosResultantes.add(new Suceso("muchAgua"));

assertEquals(sucesosResultantes, sucesosExistentes);

sucesosExistentes.clear();
sucesosExistentes.add(new Suceso("pocaAgua","muchAgua"));
sucesosExistentes.add(new Suceso("muchPresion","pocaPresion"));
sucesosExistentes.add(new Suceso("muchAgua","pocaAgua"));
sucesosExistentes.add(new Suceso("muchAgua","pocaAgua"));
sucesosExistentes.add(new Suceso("pocaAgua","muchAgua"));
sucesosExistentes.add(new Suceso("muchAgua"));

cancelador.cancelarSuceso(sucesosExistentes, suceso);

sucesosResultantes.clear();
sucesosResultantes.add(new Suceso("pocaAgua","muchAgua"));
sucesosResultantes.add(new Suceso("muchPresion","pocaPresion"));
sucesosResultantes.add(new Suceso("muchAgua","pocaAgua"));
sucesosResultantes.add(new Suceso("pocaAgua","muchAgua"));
sucesosResultantes.add(new Suceso("muchAgua"));

assertEquals(sucesosResultantes, sucesosExistentes);

sucesosExistentes.clear();
sucesosExistentes.add(new Suceso("pocaAgua","muchAgua"));
sucesosExistentes.add(new Suceso("muchPresion","pocaPresion"));
sucesosExistentes.add(new Suceso("muchAgua","pocaAgua"));
sucesosExistentes.add(new Suceso("muchAgua","pocaAgua"));
sucesosExistentes.add(new Suceso("pocaAgua","muchAgua"));
sucesosExistentes.add(new Suceso("muchAgua"));

cancelador.cancelarSuceso(sucesosExistentes, new Suceso("pocaTemperatura"));

assertEquals(sucesosExistentes, sucesosExistentes);

sucesosExistentes.clear();
sucesosExistentes.add(new Suceso("pocaAgua","muchAgua"));
sucesosExistentes.add(new Suceso("muchPresion","pocaPresion"));
sucesosExistentes.add(new Suceso("muchAgua","pocaAgua"));
sucesosExistentes.add(new Suceso("pocaAgua","muchAgua"));
sucesosExistentes.add(new Suceso("muchAgua"));

cancelador.cancelarSuceso(sucesosExistentes, new Suceso
    ("pocaPresion","pocaTemperatura"));

sucesosResultantes.clear();

```

```

    sucesosResultantes.add(new Suceso("pocaAgua","muchagua"));
    sucesosResultantes.add(new Suceso("muchagua","pocaAgua"));
    sucesosResultantes.add(new Suceso("pocaAgua","muchagua"));
    sucesosResultantes.add(new Suceso("muchagua"));

    assertEquals(sucesosResultantes, sucesosExistentes);
}

public void testCancelarSucesos(){

    sucesosExistentes.add(new Suceso("pocaAgua","muchagua"));
    sucesosExistentes.add(new Suceso("muchapresion","pocapresion"));
    sucesosExistentes.add(new Suceso("muchagua","pocaAgua"));
    sucesosExistentes.add(new Suceso("pocaAgua","muchagua"));
    sucesosExistentes.add(new Suceso("muchagua"));

    sucesosNuevos.add(new Suceso("muchagua"));
    sucesosNuevos.add(new Suceso("muchapresion","pocapresion"));
    sucesosNuevos.add(new Suceso("muchagua","pocaAgua"));

    cancelador.cancelarSucesos(sucesosExistentes, sucesosNuevos);

    sucesosResultantes.add(new Suceso("muchapresion","pocapresion"));
    sucesosResultantes.add(new Suceso("muchagua","pocaAgua"));
    sucesosResultantes.add(new Suceso("muchagua"));

    assertEquals(sucesosResultantes, sucesosExistentes);

    sucesosExistentes.clear();
    sucesosNuevos.clear();
    sucesosExistentes.add(new Suceso("pocaAgua","muchagua"));
    sucesosExistentes.add(new Suceso("muchapresion","pocapresion"));
    sucesosExistentes.add(new Suceso("muchagua","pocaAgua"));
    sucesosExistentes.add(new Suceso("muchagua","pocaAgua"));
    sucesosExistentes.add(new Suceso("pocaAgua","muchagua"));
    sucesosExistentes.add(new Suceso("muchagua"));

    sucesosNuevos.add(new Suceso("muchagua"));
    sucesosNuevos.add(new Suceso("muchapresion","pocapresion"));
    sucesosNuevos.add(new Suceso("pocaAgua","muchagua"));

    cancelador.cancelarSucesos(sucesosExistentes, sucesosNuevos);

    sucesosResultantes.clear();
    sucesosResultantes.add(new Suceso("muchapresion","pocapresion"));
    sucesosResultantes.add(new Suceso("muchagua","pocaAgua"));
    sucesosResultantes.add(new Suceso("pocaAgua","muchagua"));
    sucesosResultantes.add(new Suceso("muchagua"));

```



```

assertEquals(sucesosResultantes, sucesosExistentes);

sucesosExistentes.clear();
sucesosNuevos.clear();
sucesosExistentes.add(new Suceso("pocaAgua","muchagua"));
sucesosExistentes.add(new Suceso("muchapresion","pocapresion"));
sucesosExistentes.add(new Suceso("muchagua","pocaagua"));
sucesosExistentes.add(new Suceso("muchagua","pocaagua"));
sucesosExistentes.add(new Suceso("pocaagua","muchagua"));
sucesosExistentes.add(new Suceso("muchagua"));

sucesosNuevos.add(new Suceso("bajaTemperatura"));
sucesosNuevos.add(new Suceso("altaTemperatura","bajaTemperatura"));
sucesosNuevos.add(new Suceso("pocaluz","muchaluz"));

cancelador.cancelarSucesos(sucesosExistentes, sucesosNuevos);

assertEquals(sucesosExistentes, sucesosExistentes);

sucesosExistentes.clear();
sucesosNuevos.clear();
sucesosExistentes.add(new Suceso("pocaAgua","muchagua"));
sucesosExistentes.add(new Suceso("muchapresion","pocapresion"));
sucesosExistentes.add(new Suceso("muchagua","pocaagua"));
sucesosExistentes.add(new Suceso("muchagua","pocaagua"));
sucesosExistentes.add(new Suceso("pocaagua","muchagua"));
sucesosExistentes.add(new Suceso("muchagua"));

sucesosNuevos.add(new Suceso("bajaTemperatura"));
sucesosNuevos.add(new Suceso("pocapresion","bajaTemperatura"));
sucesosNuevos.add(new Suceso("pocaagua","muchaluz"));

cancelador.cancelarSucesos(sucesosExistentes, sucesosNuevos);

sucesosResultantes.clear();
sucesosResultantes.add(new Suceso("pocaAgua","muchagua"));
sucesosResultantes.add(new Suceso("muchagua","pocaagua"));
sucesosResultantes.add(new Suceso("pocaAgua","muchagua"));
sucesosResultantes.add(new Suceso("muchagua"));

assertEquals(sucesosResultantes, sucesosExistentes);
}

}

```

TestCanceladorTotal.java

```
package prueba;

import java.util.ArrayList;
import java.util.List;

import junit.framework.TestCase;
import source.CanceladorTotal;
import source.Suceso;

/**
 * Test de funcionamiento del cancelador total.
 *
 * @author Grupo20
 */
public class TestCanceladorTotal extends TestCase {

    private CanceladorTotal cancelador;

    private List<Suceso> sucesosExistentes;

    private List<Suceso> sucesosNuevos;

    private List<Suceso> sucesosResultantes;

    private Suceso suceso;

    protected void setUp() throws Exception {
        super.setUp();
        this.cancelador = CanceladorTotal.obtenerInstancia();
        this.sucesosExistentes = new ArrayList<Suceso>();
        this.sucesosNuevos = new ArrayList<Suceso>();
        this.sucesosResultantes = new ArrayList<Suceso>();
    }

    public void testCancelarSuceso(){

        sucesosExistentes.clear();
        sucesosNuevos.clear();

        this.suceso = new Suceso("pocaAgua");
        sucesosExistentes.add(new Suceso("pocaAgua","muchachaAgua"));
        sucesosExistentes.add(new Suceso("muchachaPresion","pocaPresion"));
    }
}
```

```

sucesosExistentes.add(new Suceso("muchagua", "pocaagua"));
sucesosExistentes.add(new Suceso("pocaagua", "muchagua"));

cancelador.cancelarSuceso(sucesosExistentes, suceso);

sucesosResultantes.add(new Suceso("pocaagua", "muchagua"));
sucesosResultantes.add(new Suceso("muchapresion", "pocapresion"));
sucesosResultantes.add(new Suceso("pocaagua", "muchagua"));

assertEquals(sucesosResultantes, sucesosExistentes);

sucesosExistentes.clear();
sucesosExistentes.add(new Suceso("pocaagua", "muchagua"));
sucesosExistentes.add(new Suceso("muchapresion", "pocapresion"));
sucesosExistentes.add(new Suceso("muchagua", "pocaagua"));
sucesosExistentes.add(new Suceso("muchagua", "pocaagua"));
sucesosExistentes.add(new Suceso("pocaagua", "muchagua"));

cancelador.cancelarSuceso(sucesosExistentes, suceso);

sucesosResultantes.clear();
sucesosResultantes.add(new Suceso("pocaagua", "muchagua"));
sucesosResultantes.add(new Suceso("muchapresion", "pocapresion"));
sucesosResultantes.add(new Suceso("pocaagua", "muchagua"));

assertEquals(sucesosResultantes, sucesosExistentes);

sucesosExistentes.clear();
sucesosExistentes.add(new Suceso("pocaagua", "muchagua"));
sucesosExistentes.add(new Suceso("muchapresion", "pocapresion"));
sucesosExistentes.add(new Suceso("muchagua", "pocaagua"));
sucesosExistentes.add(new Suceso("muchagua", "pocaagua"));
sucesosExistentes.add(new Suceso("pocaagua", "muchagua"));
sucesosExistentes.add(new Suceso("muchagua"));

cancelador.cancelarSuceso(sucesosExistentes, new Suceso("pocatemperatura"));

assertEquals(sucesosExistentes, sucesosExistentes);

sucesosExistentes.clear();
sucesosExistentes.add(new Suceso("pocaagua", "muchagua"));
sucesosExistentes.add(new Suceso("muchapresion", "pocapresion"));
sucesosExistentes.add(new Suceso("muchagua", "pocaagua"));
sucesosExistentes.add(new Suceso("pocaagua", "muchagua"));
sucesosExistentes.add(new Suceso("muchapresion", "pocapresion"));
sucesosExistentes.add(new Suceso("muchagua"));
sucesosExistentes.add(new Suceso("pocatemperatura", "pocapresion"));

```

```

cancelador.cancelarSuceso(sucesosExistentes, new Suceso
                           ("pocaPresion","pocaTemperatura"));

sucesosResultantes.clear();
sucesosResultantes.add(new Suceso("pocaAgua","muchagua"));
sucesosResultantes.add(new Suceso("muchagua","pocaAgua"));
sucesosResultantes.add(new Suceso("pocaAgua","muchagua"));
sucesosResultantes.add(new Suceso("muchagua"));

assertEquals(sucesosResultantes, sucesosExistentes);
}

public void testCancelarSucesos(){

    sucesosExistentes.add(new Suceso("pocaAgua","muchagua"));
    sucesosExistentes.add(new Suceso("muchaguaPresion","pocaPresion"));
    sucesosExistentes.add(new Suceso("muchagua","pocaAgua"));
    sucesosExistentes.add(new Suceso("pocaAgua","muchagua"));
    sucesosExistentes.add(new Suceso("muchagua"));

    sucesosNuevos.add(new Suceso("muchagua"));
    sucesosNuevos.add(new Suceso("muchaguaPresion","pocaPresion"));
    sucesosNuevos.add(new Suceso("muchagua","pocaAgua"));

    cancelador.cancelarSucesos(sucesosExistentes, sucesosNuevos);

    sucesosResultantes.add(new Suceso("muchaguaPresion","pocaPresion"));
    sucesosResultantes.add(new Suceso("muchagua","pocaAgua"));
    sucesosResultantes.add(new Suceso("muchagua"));

    assertEquals(sucesosResultantes, sucesosExistentes);

    sucesosExistentes.clear();
    sucesosNuevos.clear();
    sucesosExistentes.add(new Suceso("pocaAgua","muchagua"));
    sucesosExistentes.add(new Suceso("muchaguaPresion","pocaPresion"));
    sucesosExistentes.add(new Suceso("muchagua","pocaAgua"));
    sucesosExistentes.add(new Suceso("muchagua","pocaAgua"));
    sucesosExistentes.add(new Suceso("pocaAgua","muchagua"));
    sucesosExistentes.add(new Suceso("pocaPresion","muchaguaPresion"));
    sucesosExistentes.add(new Suceso("muchagua"));

    sucesosNuevos.add(new Suceso("muchagua"));
    sucesosNuevos.add(new Suceso("muchaguaPresion","pocaPresion"));
    sucesosNuevos.add(new Suceso("pocaAgua","muchagua"));

    cancelador.cancelarSucesos(sucesosExistentes, sucesosNuevos);

```

```

    sucesosResultantes.clear();
    sucesosResultantes.add(new Suceso("muchPresion","pocaPresion"));

    assertEquals(sucesosResultantes, sucesosExistentes);

    sucesosExistentes.clear();
    sucesosNuevos.clear();
    sucesosExistentes.add(new Suceso("pocaAgua","muchAgua"));
    sucesosExistentes.add(new Suceso("muchPresion","pocaPresion"));
    sucesosExistentes.add(new Suceso("muchAgua","pocaAgua"));
    sucesosExistentes.add(new Suceso("muchAgua","pocaAgua"));
    sucesosExistentes.add(new Suceso("pocaAgua","muchAgua"));
    sucesosExistentes.add(new Suceso("muchAgua"));

    sucesosNuevos.add(new Suceso("bajaTemperatura"));
    sucesosNuevos.add(new Suceso("altaTemperatura","bajaTemperatura"));
    sucesosNuevos.add(new Suceso("pocaLuz","muchLuz"));

    cancelador.cancelarSucesos(sucesosExistentes, sucesosNuevos);

    assertEquals(sucesosExistentes, sucesosExistentes);

    sucesosExistentes.clear();
    sucesosNuevos.clear();
    sucesosExistentes.add(new Suceso("pocaAgua","muchAgua"));
    sucesosExistentes.add(new Suceso("muchPresion","pocaPresion"));
    sucesosExistentes.add(new Suceso("muchAgua","pocaAgua"));
    sucesosExistentes.add(new Suceso("muchAgua","pocaAgua"));
    sucesosExistentes.add(new Suceso("pocaAgua","muchAgua"));

    sucesosNuevos.add(new Suceso("bajaTemperatura"));
    sucesosNuevos.add(new Suceso("pocaPresion","bajaTemperatura"));
    sucesosNuevos.add(new Suceso("pocaAgua","muchLuz"));

    cancelador.cancelarSucesos(sucesosExistentes, sucesosNuevos);

    sucesosResultantes.clear();
    sucesosResultantes.add(new Suceso("pocaAgua","muchAgua"));
    sucesosResultantes.add(new Suceso("pocaAgua","muchAgua"));

    assertEquals(sucesosResultantes, sucesosExistentes);
}

}

```

TestEvaluadores.java

```
package prueba;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

import cliente.AccionApagarBomba;
import cliente.AccionPrenderBomba;
import cliente.Bomba;
import cliente.Tanque;

import source.ManejadorDeSucesos;
import source.Suceso;
import junit.framework.TestCase;

public class TestEvaluadores extends TestCase {
    private ManejadorDeSucesos manejadorSucesos;

    private Tanque tanque;

    private Bomba bomba;

    private Suceso sucesoPocaAgua;

    private Suceso sucesoPresionAlta;

    private List<Suceso> sucesos;

    private AccionPrenderBomba accionPrenderBomba;

    private AccionApagarBomba accionApagarBomba;
    protected void setUp() throws Exception {
        super.setUp();
        this.manejadorSucesos = new ManejadorDeSucesos();
        this.bomba = new Bomba();
        this.tanque = new Tanque();
        this.sucesoPocaAgua = new Suceso("pocaAgua");
        this.sucesoPresionAlta = new Suceso("presionAlta");
        this.sucesos = new LinkedList<Suceso>();
        this.accionApagarBomba = new AccionApagarBomba();

        this.accionApagarBomba.setBomba(bomba);
        this.accionApagarBomba.setTanque(tanque);

        this.accionPrenderBomba = new AccionPrenderBomba();
```

```

        this.accionPrenderBomba.setBomba(bomba);
        this.manejadorSucesos.borrarImplicaciones();
    }

    public void testEvaluadorSecuenciaContinua(){
        //Si pocaAgua ^ presionAlta----->PrenderBomBa
        sucesos.add(sucesoPocaAgua);
        sucesos.add(sucesoPresionAlta);
        sucesos.add(new Suceso("tanqueLleno"));
        this.manejadorSucesos.suscribirImplicacion(accionPrenderBomba, sucesos);
        assertEquals(false, this.bomba.isEncendida());
        // Se establece la configuracion de Susecos en secuencia Continua
        this.manejadorSucesos.establecerConfiguracionSecuenciaContinua();

        this.manejadorSucesos.agregarSuceso(new Suceso("pocaAgua"));
        this.manejadorSucesos.agregarSuceso(new Suceso("presionAlta"));
        this.manejadorSucesos.agregarSuceso(new Suceso("tanqueLleno"));
        this.manejadorSucesos.notificar();
        assertEquals(true, this.bomba.isEncendida());

        this.bomba.setEncendida(false);
        this.manejadorSucesos.agregarSuceso(new Suceso("presionAlta"));
        this.manejadorSucesos.agregarSuceso(new Suceso("pocaAgua"));
        this.manejadorSucesos.agregarSuceso(new Suceso("tanqueLleno"));
        this.manejadorSucesos.notificar();
        assertEquals(false, this.bomba.isEncendida());

        this.bomba.setEncendida(false);
        this.manejadorSucesos.agregarSuceso(new Suceso("presionAlta"));
        this.manejadorSucesos.agregarSuceso(new Suceso("pocaAgua"));
        this.manejadorSucesos.agregarSuceso(new Suceso("tanqueRoto"));
        this.manejadorSucesos.notificar();
        assertEquals(false, this.bomba.isEncendida());

        this.bomba.setEncendida(false);
        this.manejadorSucesos.agregarSuceso(new Suceso("pocaAgua"));
        this.manejadorSucesos.agregarSuceso(new Suceso("presionAlta"));
        this.manejadorSucesos.agregarSuceso(new Suceso("tanqueLleno"));
        this.manejadorSucesos.agregarSuceso(new Suceso("tanqueRoto"));
        this.manejadorSucesos.notificar();
        assertEquals(true, this.bomba.isEncendida());
    }
}

```

```

public void testEvaluadorSecuenciaDiscontinua(){
    //Si pocaAgua ^ presionAlta----->PrenderBomBa
    sucesos.add(sucesoPocaAgua);
    sucesos.add(sucesoPresionAlta);
    sucesos.add(new Suceso("tanqueLleno"));
    this.manejadorSucesos.suscribirImplicacion(accionPrenderBomba, sucesos);
    assertEquals(false, this.bomba.isEncendida());

    this.manejadorSucesos.establecerConfiguracionSecuenciaDiscontinua();

    this.manejadorSucesos.agregarSuceso(new Suceso("pocaAgua"));
    this.manejadorSucesos.agregarSuceso(new Suceso("presionBaja"));
    this.manejadorSucesos.agregarSuceso(new Suceso("presionAlta"));
    this.manejadorSucesos.agregarSuceso(new Suceso("algoDeAgua"));
    this.manejadorSucesos.agregarSuceso(new Suceso("tanqueLleno"));
    this.manejadorSucesos.notificar();
    assertEquals(true, this.bomba.isEncendida());

    this.bomba.setEncendida(false);
    this.manejadorSucesos.agregarSuceso(new Suceso("pocaAgua"));
    this.manejadorSucesos.agregarSuceso(new Suceso("pocaAgua"));
    this.manejadorSucesos.agregarSuceso(new Suceso("presionBaja"));
    this.manejadorSucesos.agregarSuceso(new Suceso("presionAlta"));
    this.manejadorSucesos.agregarSuceso(new Suceso("algoDeAgua"));
    this.manejadorSucesos.agregarSuceso(new Suceso("tanqueLleno"));
    this.manejadorSucesos.agregarSuceso(new Suceso("tanqueLleno"));
    this.manejadorSucesos.notificar();
    assertEquals(true, this.bomba.isEncendida());

    this.bomba.setEncendida(false);
    this.manejadorSucesos.agregarSuceso(new Suceso("AlgoDeAgua"));
    this.manejadorSucesos.agregarSuceso(new Suceso("presionBaja"));
    this.manejadorSucesos.agregarSuceso(new Suceso("presionAlta"));
    this.manejadorSucesos.agregarSuceso(new Suceso("pocaAgua"));
    this.manejadorSucesos.notificar();
    assertEquals(false, this.bomba.isEncendida());

    this.bomba.setEncendida(false);
    this.manejadorSucesos.agregarSuceso(new Suceso("pocaAgua"));
    this.manejadorSucesos.notificar();
    assertEquals(false, this.bomba.isEncendida());

    this.bomba.setEncendida(false);
    this.manejadorSucesos.agregarSuceso(new Suceso("pocaLuz"));
    this.manejadorSucesos.agregarSuceso(new Suceso("pocoAire"));

```



```

        this.manejadorSucesos.agregarSuceso(new Suceso("pocaTierra"));
        this.manejadorSucesos.notificar();
        assertEquals(false, this.bomba.isEncendida());
    }

    public void testEvaluadorPorDefecto(){
        // muchoViento ^ pocaAgua ----->accionApagarBomba
        this.manejadorSucesos.agregarSuceso(new Suceso("pocaAgua"));
        this.manejadorSucesos.agregarSuceso(new Suceso("muchoRuido"));
        this.manejadorSucesos.agregarSuceso(new Suceso("muchoViento"));
        this.manejadorSucesos.agregarSuceso(new Suceso("presionBaja"));

        sucesos.add(new Suceso("muchoViento"));
        sucesos.add(new Suceso("pocaAgua"));

        this.bomba.setEncendida(true);

        this.manejadorSucesos.suscribirImplicacion(accionApagarBomba,sucesos);
        // solo se fija si la lista de sucesos suscriptos esta dentro de los sucesos agregados
        this.manejadorSucesos.establecerConfiguracionPorDefecto();
        this.manejadorSucesos.notificar();

        assertEquals(false, this.bomba.isEncendida());

        this.manejadorSucesos.agregarSuceso(new Suceso("presionBaja"));

        sucesos.add(new Suceso("muchoViento"));
        sucesos.add(new Suceso("pocaAgua"));

        this.manejadorSucesos.suscribirImplicacion(accionPrenderBomba,sucesos);
        this.manejadorSucesos.agregarSuceso(new Suceso("muchoViento"));
        this.manejadorSucesos.agregarSuceso(new Suceso("presionBaja"));
        this.manejadorSucesos.notificar();
        assertEquals(false, this.bomba.isEncendida());

        sucesos.add(new Suceso("muchoViento"));
        sucesos.add(new Suceso("pocaAgua"));

        this.manejadorSucesos.suscribirImplicacion(accionPrenderBomba,sucesos);
        this.manejadorSucesos.agregarSuceso(new Suceso("muchoViento"));
        this.manejadorSucesos.notificar();
        assertEquals(false, this.bomba.isEncendida());

        List<Suceso> listaSucesos = new ArrayList<Suceso>();

        listaSucesos.add(new Suceso("pocaAgua"));
    }

```

```

        listaSucesos.add(new Suceso("muchoRuido"));
        listaSucesos.add(new Suceso("muchoViento"));
        listaSucesos.add(new Suceso("presionBaja"));

        sucesos.add(new Suceso("muchoViento"));
        sucesos.add(new Suceso("pocaAgua"));

        this.manejadorSucesos.suscribirImplicacion(accionPrenderBomba,sucesos);

        this.manejadorSucesos.notificar(listaSucesos);
        assertEquals(true, this.bomba.isEncendida());

    }

    public void testEvaluadorIgualdadConjunto(){
        // muchoViento ^ pocaAgua ----->accionApagarBomba
        this.manejadorSucesos.agregarSuceso(new Suceso("pocaAgua"));
        this.manejadorSucesos.agregarSuceso(new Suceso("muchoRuido"));
        this.manejadorSucesos.agregarSuceso(new Suceso("muchoViento"));
        this.manejadorSucesos.agregarSuceso(new Suceso("presionBaja"));

        sucesos.add(new Suceso("muchoViento"));
        sucesos.add(new Suceso("pocaAgua"));

        this.bomba.setEncendida(false);

        this.manejadorSucesos.suscribirImplicacion(accionApagarBomba,sucesos);
        // solo se fija si la lista de sucesos suscriptos esta dentro de los sucesos agregados
        this.manejadorSucesos.establecerConfiguracionIgualdadConjunto();
        this.manejadorSucesos.notificar();

        assertEquals(false, this.bomba.isEncendida());

        sucesos.add(new Suceso("muchoViento"));
        sucesos.add(new Suceso("pocaAgua"));
        sucesos.add(new Suceso("muchoCalor"));

        this.manejadorSucesos.suscribirImplicacion(accionPrenderBomba,sucesos);
        this.manejadorSucesos.agregarSuceso(new Suceso("muchoViento"));
        this.manejadorSucesos.agregarSuceso(new Suceso("presionBaja"));
        this.manejadorSucesos.notificar();
        assertEquals(false, this.bomba.isEncendida());

        sucesos.add(new Suceso("muchoViento"));
        sucesos.add(new Suceso("pocaAgua"));

```

```
sucesos.add(new Suceso("pocoSodio"));
sucsesos.add(new Suceso("bajaTemperatura"));

this.manejadorSucsesos.suscribirImplicacion(accionPrenderBomba,sucesos);
this.manejadorSucsesos.agregarSuceso(new Suceso("muchoViento"));
this.manejadorSucsesos.agregarSuceso(new Suceso("pocaAgua"));
this.manejadorSucsesos.agregarSuceso(new Suceso("pocoSodio"));
this.manejadorSucsesos.agregarSuceso(new Suceso("bajaTemperatura"));
this.manejadorSucsesos.notificar();
assertEquals(false, this.bomba.isEncendida());

}

}
```

TestManejadorDeSucesos.java

```
package prueba;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import cliente.AccionAgregarGaseosa;
import cliente.AccionApagarBomba;
import cliente.AccionPrenderBomba;
import cliente.AccionQuitarGaseosa;
import cliente.AccionRepararMaquinaGasesosa;
import cliente.AccionTirarGaseosa;
import cliente.Bomba;
import cliente.Gaseosa;
import cliente.MaquinaGaseosa;
import cliente.Tanque;
import source.*;
import junit.framework.TestCase;

public class TestManejadorDeSucesos extends TestCase {

    private ManejadorDeSucesos manejadorSucesos;

    private Tanque tanque;

    private Bomba bomba;

    private Suceso sucesoPocaAgua;

    private Suceso sucesoPresionAlta;

    private List<Suceso> sucesos;

    private AccionPrenderBomba accionPrenderBomba;

    private AccionApagarBomba accionApagarBomba;

    @Override
    protected void setUp() throws Exception {
        super.setUp();
        this.manejadorSucesos = new ManejadorDeSucesos();
        this.manejadorSucesos.establishConfiguracionPorDefecto();
        this.bomba = new Bomba();
        this.tanque = new Tanque();
        this.sucesoPocaAgua = new Suceso("pocaAgua");
        this.sucesoPresionAlta = new Suceso("presionAlta");
        this.sucesos = new LinkedList<Suceso>();
    }
}
```

```

        this.accionApagarBomba = new AccionApagarBomba();

        this.accionApagarBomba.setBomba(bomba);
        this.accionApagarBomba.setTanque(tanque);

        this.accionPrenderBomba = new AccionPrenderBomba();
        this.accionPrenderBomba.setBomba(bomba);
        this.manejadorSucesos.borrarImplicaciones();
    }

    public void testSuscribirImplicacionYNotificarSuceso(){
        //si pocaAgua ---> accionPrenderBomba
        this.manejadorSucesos.suscribirImplicacion(accionPrenderBomba,
                                                    sucesoPocaAgua);

        assertEquals(false, this.bomba.isEncendida());
        // notificacion sucesoPocaAgua
        this.manejadorSucesos.notificar(sucesoPocaAgua);
        assertEquals(true, this.bomba.isEncendida());
    }

    public void testSuscribirImplicacionYNotificarSucesoInvalido(){
        //si pocaAgua ---> accionPrenderBomba
        this.manejadorSucesos.suscribirImplicacion(accionPrenderBomba,
                                                    sucesoPocaAgua);

        assertEquals(false, this.bomba.isEncendida());
        //notificacion sucesoPresionAlta
        this.manejadorSucesos.notificar(sucesoPresionAlta);
        assertEquals(false, this.bomba.isEncendida());
    }

    public void testSuscribirImplicacionYNotificarMultiplesSucesos(){
        //Si pocaAgua ^ presionAlta----->PrenderBomba
        sucesos.add(sucesoPocaAgua);
        sucesos.add(sucesoPresionAlta);
        this.manejadorSucesos.suscribirImplicacion(accionPrenderBomba, sucesos);
        assertEquals(false, this.bomba.isEncendida());
        //notificacion presionAlta
        this.manejadorSucesos.notificar(sucesoPresionAlta);
        assertEquals(false, this.bomba.isEncendida());
        //notificacion pocaAgua
        this.manejadorSucesos.notificar(sucesoPocaAgua);
        assertEquals(false, this.bomba.isEncendida());

        Suceso nuevoSucesoPocaAgua = new Suceso("pocaAgua");
        Suceso nuevoSucesoPresionAlta = new Suceso("presionAlta");
        List<Suceso> sucesosNuevos = new ArrayList<Suceso>();
        sucesosNuevos.add(nuevoSucesoPocaAgua);
    }

```

```

        sucesosNuevos.add(nuevoSucesoPresionAlta);
        //notificacion pocaAgua ^ presionAlta
        this.manejadorSucesos.notificar(sucesosNuevos);
        assertEquals(true, this.bomba.isEncendida());
    }

    /**
     * Se testea la suscripcion de implicaciones y la notificacion de varios sucesos
     * ocurridos
     */
    public void testSuscribirImplicacionYNotificarMultiplesSucesosInvalidos(){
        //Si pocaAgua ^ presionAlta----->PrenderBomBa
        sucesos.add(sucesoPocaAgua);
        sucesos.add(sucesoPresionAlta);
        this.manejadorSucesos.suscribirImplicacion(accionPrenderBomba, sucesos);
        assertEquals(false, this.bomba.isEncendida());

        Suceso nuevoSucesoPocaAgua = new Suceso("pocaAgua");
        Suceso nuevoSucesoTanqueLleno = new Suceso("tanqueLleno");
        List<Suceso> sucesosNuevos = new ArrayList<Suceso>();
        sucesosNuevos.add(nuevoSucesoPocaAgua);
        sucesosNuevos.add(nuevoSucesoTanqueLleno);
        //notificacion pocaAgua ^ presionAlta
        this.manejadorSucesos.notificar(sucesosNuevos);
        assertEquals(false, this.bomba.isEncendida());
    }

    /**
     * Se realizan pruebas para ver si la funcionalidad de suscribir
     * sucesos y acciones se realizan de la forma correcta
     */
    public void testAgregarSucesosYNotificar(){
        //Si pocaAgua ^ presionAlta----->PrenderBomBa
        sucesos.add(sucesoPocaAgua);
        sucesos.add(sucesoPresionAlta);
        this.manejadorSucesos.suscribirImplicacion(accionPrenderBomba, sucesos);
        assertEquals(false, this.bomba.isEncendida());

        this.manejadorSucesos.agregarSuceso(new Suceso("pocaAgua"));
        this.manejadorSucesos.agregarSuceso(new Suceso("presionAlta"));
        //notifica pocaAgua ^ presionAlta
        this.manejadorSucesos.notificar();
        assertEquals(true, this.bomba.isEncendida());

        this.bomba.setEncendida(false);
        //notifica pocaAgua ^ presionAlta ^ tanqueLleno
        this.manejadorSucesos.agregarSuceso(new Suceso("pocaAgua"));
        this.manejadorSucesos.agregarSuceso(new Suceso("presionAlta"));
    }

```

```

        this.manejadorSucesos.agregarSuceso(new Suceso("tanqueLleno"));
        this.manejadorSucesos.notificar();
        assertEquals(true, this.bomba.isEncendida());

        this.bomba.setEncendida(false);
        //notifica pocaAgua ^ tanqueLleno
        this.manejadorSucesos.agregarSuceso(new Suceso("pocaAgua"));
        this.manejadorSucesos.agregarSuceso(new Suceso("tanqueLleno"));
        this.manejadorSucesos.notificar();
        assertEquals(false, this.bomba.isEncendida());

        this.bomba.setEncendida(false);
        this.manejadorSucesos.notificar();
        assertEquals(false, this.bomba.isEncendida());
    }
    /**
     * En este test se evalua que el manejador de sucesos controle si
     * el suceso o conjuntos de sucesos que le llegan sean nulos o si la
     * lista contiene algun elemento nulo
     */
    public void testNotificacionSucesosNulos(){
        sucesos.add(sucesoPocaAgua);
        sucesos.add(sucesoPresionAlta);
        this.manejadorSucesos.suscribirImplicacion(accionPrenderBomba, sucesos);
        assertEquals(false, this.bomba.isEncendida());
        //no agrega el suceso nulo, entonces no notifica nada
        this.manejadorSucesos.agregarSuceso(null);
        this.manejadorSucesos.notificar();
        assertEquals(false, this.bomba.isEncendida());

        //ignora el suceso nulo y notifica los sucesos
        this.manejadorSucesos.agregarSuceso(sucesoPocaAgua);
        this.manejadorSucesos.agregarSuceso(sucesoPresionAlta);
        this.manejadorSucesos.agregarSuceso(null);
        this.manejadorSucesos.notificar();
        assertEquals(true, this.bomba.isEncendida());

        bomba.setEncendida(false);
        this.manejadorSucesos.agregarSucesos(null);
        this.manejadorSucesos.notificar();
        assertEquals(false, this.bomba.isEncendida());

        this.manejadorSucesos.agregarSuceso(sucesoPocaAgua);
        this.manejadorSucesos.agregarSuceso(sucesoPresionAlta);
        List<Suceso> listaNula=null;

        this.manejadorSucesos.notificar(listaNula);
        assertEquals(true, this.bomba.isEncendida());
    }

```

```

        bomba.setEncendida(false);
        this.manejadorSucesos.agregarSuceso(sucesoPocaAgua);
        this.manejadorSucesos.agregarSuceso(sucesoPresionAlta);
        Suceso sucesoNulo = null;
        this.manejadorSucesos.notificar(sucesoNulo);
        assertEquals(true, this.bomba.isEncendida());
    }
    /**
     * En esta test se prueba el funcionamiento del manejador de Sucesos
     * con 10 implicaciones y se evaluan los resultados de las accion.
     */
    public void testDeEstres(){
        List<Suceso> lista0 = new ArrayList<Suceso>();
        List<Suceso> lista1 = new ArrayList<Suceso>();
        List<Suceso> lista2 = new ArrayList<Suceso>();
        List<Suceso> lista3 = new ArrayList<Suceso>();
        List<Suceso> lista4 = new ArrayList<Suceso>();
        List<Suceso> lista5 = new ArrayList<Suceso>();
        List<Suceso> lista6 = new ArrayList<Suceso>();
        List<Suceso> lista7 = new ArrayList<Suceso>();
        List<Suceso> lista8 = new ArrayList<Suceso>();
        List<Suceso> lista9 = new ArrayList<Suceso>();

        lista0.add(new Suceso("ponerFicha"));
        lista0.add(new Suceso("seleccionaCoca"));
        lista0.add(new Suceso("apretaBoton"));

        lista1.add(new Suceso("noHayCoca"));
        lista1.add(new Suceso("noHayCocaLight"));
        lista1.add(new Suceso("noHaySprite"));

        lista2.add(new Suceso("ingresaMoneda"));
        lista2.add(new Suceso("importeInsuficiente"));

        lista3.add(new Suceso("apretaBoton"));
        lista3.add(new Suceso("seleccionaNada"));

        lista4.add(new Suceso("ingresoSobranteMoneda"));
        lista4.add(new Suceso("noHayCambio"));

        lista5.add(new Suceso("ponerFicha"));
        lista5.add(new Suceso("fichaTrabada"));

        lista6.add(new Suceso("gaseosaCaliente"));

        lista7.add(new Suceso("pocaGaseosa"));
    }

```



```

lista7.add(new Suceso("maquinaDesconectada"));

lista8.add(new Suceso("gaseosaFria"));
lista8.add(new Suceso("stockLleno"));

lista9.add(new Suceso("maquinaRota"));

Gaseosa gaseosa = new Gaseosa();
Gaseosa gaseosa1 = new Gaseosa();
gaseosa.setFrio(true);
gaseosa.setNombre("Coca");
gaseosa.setStock(10);

gaseosa.setFrio(true);
gaseosa.setNombre("Sprite");
gaseosa.setStock(10);

MaquinaGaseosa maquina = new MaquinaGaseosa();
maquina.agregarGaseosa(gaseosa1);
maquina.agregarGaseosa(gaseosa);

AccionAgregarGaseosa accionAgregarGaseosa = new AccionAgregarGaseosa();
accionAgregarGaseosa.setGaseosa(new Gaseosa("Fanta",10,false));

AccionRepararMaquinaGasesosa accionReparar = new
    AccionRepararMaquinaGasesosa();
accionReparar.setMaquina(maquina);

AccionQuitarGaseosa accionQuitarGaseosa = new AccionQuitarGaseosa();
accionQuitarGaseosa.setGaseosa(new Gaseosa("coca", 20, false));
//para mayor facilidad pongo que si cumple la implicancia
assertEquals(false,maquina.isRota());
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista0);
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista1);
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista2);
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista3);
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista4);
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista5);
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista6);
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista7);
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista8);
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista9);

//Suceso que ninguna de las implicaciones espera
this.manejadorSucesos.agregarSuceso(new Suceso("stockLleno"));
this.manejadorSucesos.notificar();
assertEquals(false,maquina.isRota());

```

```
//la implicacion de la lista 8 espera este conjunto de sucesos
this.manejadorSucesos.agregarSuceso(new Suceso("stockLleno"));
this.manejadorSucesos.agregarSuceso(new Suceso("gaseosaFria"));
this.manejadorSucesos.notificar();
assertEquals(true,maquina.isRota());
```

```
//como se que se ejecuta el comando de la implicacion 8, le cambio el comando
assertEquals(20,accionQuitarGaseosa.getGaseosa().getStock());
this.manejadorSucesos.borrarImplicaciones();
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista0);
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista1);
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista2);
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista3);
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista4);
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista5);
this.manejadorSucesos.suscribirImplicacion(accionReparar, lista6);
this.manejadorSucesos.suscribirImplicacion(accionAgregarGaseosa, lista7);
this.manejadorSucesos.suscribirImplicacion(accionQuitarGaseosa, lista8);
this.manejadorSucesos.suscribirImplicacion(accionQuitarGaseosa, lista9);
this.manejadorSucesos.agregarSuceso(new Suceso("stockLleno"));
this.manejadorSucesos.agregarSuceso(new Suceso("gaseosaFria"));
this.manejadorSucesos.notificar();
assertEquals(19,accionQuitarGaseosa.getGaseosa().getStock());
```

```
//Para el caso que se ejecute las acciones de las implicaciones de la
//lista 8 y 9
this.manejadorSucesos.agregarSuceso(new Suceso("stockLleno"));
this.manejadorSucesos.agregarSuceso(new Suceso("gaseosaFria"));
this.manejadorSucesos.agregarSuceso(new Suceso("maquinaRota"));
this.manejadorSucesos.notificar();
```

```
assertEquals(17,accionQuitarGaseosa.getGaseosa().getStock());
```

```
this.manejadorSucesos.agregarSuceso(new Suceso("stockLleno"));
this.manejadorSucesos.agregarSuceso(new Suceso("gaseosaFria"));
this.manejadorSucesos.agregarSuceso(new Suceso("maquinaRota"));
this.manejadorSucesos.agregarSuceso(new Suceso("pocaGaseosa"));
this.manejadorSucesos.agregarSuceso(new Suceso("maquinaDesconectada"));
this.manejadorSucesos.notificar();
assertEquals(15,accionQuitarGaseosa.getGaseosa().getStock());
assertEquals(11,accionAgregarGaseosa.getGaseosa().getStock());
```

```
}
```

```

public void testAgregarSucesoEnNotificar(){
    MaquinaGaseosa maquina = new MaquinaGaseosa();
    //Esta accion agrega un suceso dentro del ejecutar
    AccionTirarGaseosa accionTirarGaseosa = new AccionTirarGaseosa
                                                (this.manejadorSucesos);
    AccionAgregarGaseosa accionAgregarGaseosa = new AccionAgregarGaseosa();
    accionAgregarGaseosa.setGaseosa(new Gaseosa("Fanta",10,false));

    sucesos.add(new Suceso("ponerFicha"));
    sucesos.add(new Suceso("seleccionaCoca"));
    sucesos.add(new Suceso("apretaBoton"));
    assertEquals(false, maquina.isRota());

    this.manejadorSucesos.suscribirImplicacion(accionTirarGaseosa, sucesos);
    this.manejadorSucesos.suscribirImplicacion(accionAgregarGaseosa, new Suceso
                                                ("pocaGaseosa"));

    this.manejadorSucesos.notificar();
    assertEquals(10, accionAgregarGaseosa.getGaseosa().getStock());
}

public void testCancelacionPorDefecto(){

    this.manejadorSucesos.habilitarCancelador();

    //Si pocaAgua----->PrenderBomBa
    sucesos.add(sucesoPocaAgua);
    this.manejadorSucesos.suscribirImplicacion(accionPrenderBomba, sucesos);

    this.manejadorSucesos.agregarSuceso(sucesoPocaAgua);
    this.manejadorSucesos.agregarSuceso(sucesoPresionAlta);
    this.manejadorSucesos.notificar();
    assertEquals(true, this.bomba.isEncendida());

    this.bomba.setEncendida(false);
    this.manejadorSucesos.agregarSuceso(sucesoPocaAgua);
    this.manejadorSucesos.agregarSuceso(sucesoPresionAlta);
    this.manejadorSucesos.agregarSuceso(new Suceso("muchAgua", "pocaAgua"));
    this.manejadorSucesos.notificar();
    assertEquals(false, this.bomba.isEncendida());

    this.bomba.setEncendida(false);
    this.manejadorSucesos.agregarSuceso(sucesoPocaAgua);
    this.manejadorSucesos.agregarSuceso(sucesoPresionAlta);
    this.manejadorSucesos.agregarSuceso(new Suceso("presionBaja", "presionAlta"));
    this.manejadorSucesos.notificar();
}

```

```

assertEquals(true, this.bomba.isEncendida());

List<Suceso> sucesosCanceladores = new ArrayList<Suceso>();
sucesosCanceladores.add(new Suceso("muchAguA", "pocaAguA"));
sucesosCanceladores.add(new Suceso("presionBaja", "presionAlta"));
sucesosCanceladores.add(new Suceso("bajaTemperatura", "altaTemperatura"));

this.bomba.setEncendida(false);
this.manejadorSucesos.agregarSuceso(sucesoPocaAguA);
this.manejadorSucesos.agregarSuceso(sucesoPresionAlta);
this.manejadorSucesos.agregarSucesos(sucesosCanceladores);
this.manejadorSucesos.notificar();
assertEquals(false, this.bomba.isEncendida());

this.bomba.setEncendida(false);
this.manejadorSucesos.agregarSuceso(sucesoPocaAguA);
this.manejadorSucesos.agregarSuceso(sucesoPresionAlta);
this.manejadorSucesos.notificar(sucesosCanceladores);
assertEquals(false, this.bomba.isEncendida());

sucesosCanceladores.remove(new Suceso("muchAguA", "pocaAguA"));
this.bomba.setEncendida(false);
this.manejadorSucesos.agregarSuceso(sucesoPocaAguA);
this.manejadorSucesos.agregarSuceso(sucesoPresionAlta);
this.manejadorSucesos.notificar(sucesosCanceladores);
assertEquals(true, this.bomba.isEncendida());

sucesosCanceladores.add(new Suceso("muchAguA", "pocaAguA"));
this.bomba.setEncendida(false);
this.manejadorSucesos.agregarSuceso(sucesoPocaAguA);
this.manejadorSucesos.agregarSuceso(sucesoPresionAlta);
this.manejadorSucesos.notificar(sucesosCanceladores);
assertEquals(false, this.bomba.isEncendida());
}

public void testHabilitacionDeshabilitacionCancelador(){

    sucesos.add(sucesoPocaAguA);
    this.manejadorSucesos.suscribirImplicacion(accionPrenderBomba, sucesos);

    this.manejadorSucesos.habilitarCancelador();
    this.bomba.setEncendida(false);
    this.manejadorSucesos.agregarSuceso(sucesoPocaAguA);
    this.manejadorSucesos.agregarSuceso(sucesoPresionAlta);
    this.manejadorSucesos.agregarSuceso(new Suceso("muchAguA", "pocaAguA"));
    this.manejadorSucesos.notificar();
    assertEquals(false, this.bomba.isEncendida());
}

```

```

        this.manejadorSucesos.deshabilitarCancelador();
        this.bomba.setEncendida(false);
        this.manejadorSucesos.agregarSuceso(sucesoPocaAgua);
        this.manejadorSucesos.agregarSuceso(sucesoPresionAlta);
        this.manejadorSucesos.agregarSuceso(new Suceso("muchAgua", "pocaAgua"));
        this.manejadorSucesos.notificar();
        assertEquals(true, this.bomba.isEncendida());
    }

    public void testCancelacionTotal(){

        this.manejadorSucesos.habilitarCancelador();

        //Si pocaAgua----->PrenderBomBa
        sucesos.add(sucesoPocaAgua);
        this.manejadorSucesos.suscribirImplicacion(accionPrenderBomba, sucesos);

        this.manejadorSucesos.agregarSuceso(sucesoPocaAgua);
        this.manejadorSucesos.agregarSuceso(sucesoPocaAgua);
        this.manejadorSucesos.agregarSuceso(sucesoPresionAlta);
        this.manejadorSucesos.agregarSuceso(new Suceso("muchAgua", "pocaAgua"));
        this.manejadorSucesos.notificar();
        assertEquals(true, this.bomba.isEncendida());

        this.manejadorSucesos.establecerCanceladorTotal();
        this.bomba.setEncendida(false);
        this.manejadorSucesos.agregarSuceso(sucesoPocaAgua);
        this.manejadorSucesos.agregarSuceso(sucesoPocaAgua);
        this.manejadorSucesos.agregarSuceso(sucesoPresionAlta);
        this.manejadorSucesos.agregarSuceso(new Suceso("muchAgua", "pocaAgua"));
        this.manejadorSucesos.notificar();
        assertEquals(false, this.bomba.isEncendida());
    }

}

```