Sobreescritura de métodos y tratamiento de excepciones

Veremos una serie de combinaciones posibles/imposibles.

Si un método de una superclase no declara una excepción, no es posible sobrescribir dicho método y que declare(throws) una excepción verificada.

Observamos el error en tiempo de compilación

```
import java.io.*;
class Padre {
    void saludo() {
        System.out.println("Padre");
    }
}

class Hijo extends Padre {
    void saludo() throws IOException {
        System.out.println("Hijo");
    }
}
class App {
    public static void main(String args[]) {
        Padre p = new Hijo();
        p.saludo();
    }
}
```

En cambio la situación anterior es posible si la excepción es no verificada.
Observa cómo al cambiar a ArithmeticException si es posible lo anterior

```
class Padre {
    void saludo() {
        System.out.println("Padre");
    }
}

class Hijo extends Padre {
    void saludo() throws ArithmeticException {
        System.out.println("Hijo");
    }
}
class App {
    public static void main(String args[]) {
        Padre p = new Hijo();
        p.saludo();
    }
}
```

Si un método de una superclase declara una excepción, es posible sobrescribir dicho método si se declara(throws) la misma excepción o más concreta (subclase) nunca más genérica.

Ejemplo: con mismo tipo OK

```
class Padre{
  void saludo()throws ArithmeticException{System.out.println("Padre");}
}

class Hijo extends Padre{
  void saludo()throws ArithmeticException{System.out.println("soy hijo");}
}
class App{
    public static void main(String args[]){
   Padre p=new Hijo();
   try{
   p.saludo();
   }catch(Exception e){}
  }
}
```

Ejemplo: si en subclase es más específica también OK

```
class Padre{
  void saludo()throws Exception{System.out.println("Padre");}
}

class Hijo extends Padre{
  void saludo()throws ArithmeticException{System.out.println("soy hijo");}
}
class App{
    public static void main(String args[]){
   Padre p=new Hijo();
   try{
   p.saludo();
   }catch(Exception e){}
  }
}
```

Ejemplo: si en subclase es más genérica ¡ERROR!

```
class Padre{
  void saludo()throws ArithmeticException{System.out.println("Padre");}
}

class Hijo extends Padre{
  void saludo()throws Exception{System.out.println("soy hijo");}
}
class App{
    public static void main(String args[]){
   Padre p=new Hijo();
   try{
   p.saludo();
   }catch(Exception e){}
  }
}
```

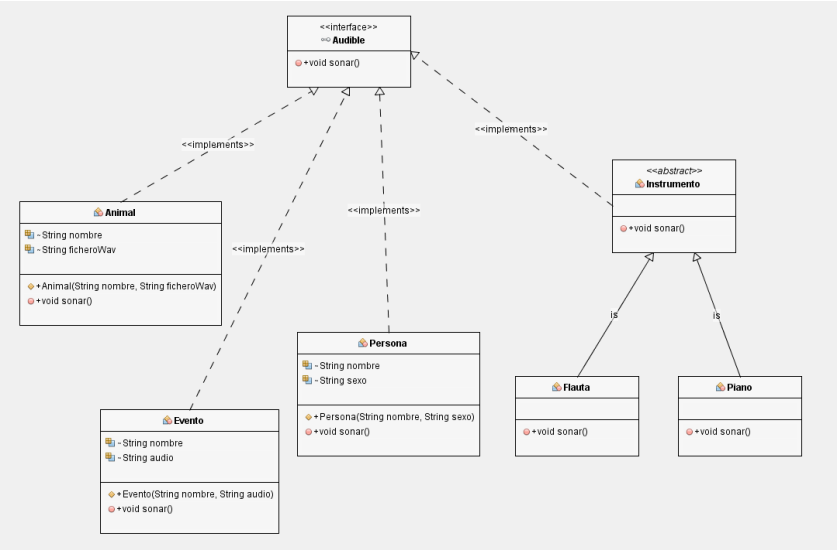
Si un método de una superclase declara una excepción, es posible sobrescribir dicho método si se declara(throws) NINGUNA excepción (está es una combinación que me resulta rara pero funciona)

```
import java.io.IOException;

class Padre {

    void saludo() throws IOException {
        System.out.println("Padre");
    }
}

class Hijo extends Padre {

    void saludo() {
        System.out.println("soy hijo");
    }
}

class App {

    public static void main(String args[]) {
        Padre p = new Hijo();
        try {
            p.saludo();
        } catch (Exception e) {
        }
    }
}
```

Observa que si en el main no hay try/catch salta error compilación por ser verificada, pero en cambio en método saludo() de Hijo no salta error alguno.

EJERCICIO:
Repasamos conceptos vistos de jerarquías y excepciones. Al sobreescribir métodos que propagan excepciones tendrás que tener en cuenta los casos de arriba.

Tenemos una serie de clases muy diferentes que implementan el mismo interface Audible según el siguiente diagrama UML.



Para generar sonido real utilizamos las clases SonidoMuestreado y SonidoSintetizado. Son dos clases muy diferentes utilizadas a propósito para incidir en que las diversas implementaciones pueden ser muy diferentes, recuerda uno de los objetivos del concepto de interface
"un interface muchas implementaciones".

Para los instrumentos musicales utilizamos sonido sintetizado(MIDI) y para el resto sonido muestreado(ficheros wav). No tienes que entender su código, simplemente observar que para generar sonido de un fichero .wav creamos un objeto .
new SonidoMuestreado().play(new File(s));
Donde s es un string con el nombre del fichero que contiene el sonido que queremos reproducir. Los ficheros almacénalos en la carpeta del proyecto (no dentro de src)

y para generar un sonido MIDI para flauta y piano
new SonidoSintetizado(73);//para flauta
new SonidoSintetizado(2);//para piano

Las clase SonidoMuestreado genera excepciones y tendrás que decidir qué hacer con ellas. aunque no entiendas el código,

clase SonidoMuestreado

```
import java.io.File;
import java.io.IOException;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.sound.sampled.FloatControl;
import javax.sound.sampled.LineEvent;
import javax.sound.sampled.LineListener;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.UnsupportedAudioFileException;

public class SonidoMuestreado implements LineListener {


    private Clip clip;
    private AudioInputStream audioIn;
    private boolean finCancion;

    public SonidoMuestreado() throws LineUnavailableException {
        //aquí podríamos cargar una lista de reproducción por defecto
        //u otras cuestiones que surgirían en un programa más elaborado
        clip = AudioSystem.getClip();
    }

    public void play(File f) throws UnsupportedAudioFileException, IOException, LineUnavailableException {
        //antes de hacer sonar una canción cerramos el stream y el clip que podrían estar actios
        if (audioIn != null) {
            audioIn.close();
        }
        if (clip != null) {
            clip.close();
        }
        //nueva canción, la que indique el File
        audioIn = AudioSystem.getAudioInputStream(f);
        clip.open(audioIn);
        clip.start();
        finCancion = false;
        clip.addLineListener(this);
        while (!finCancion) {
            // cada medio segundo miramos si acabo la canción
            try {
                Thread.sleep(500);
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
        clip.close();
    }



    public void volumen(float ganancia) {
        if (clip.isRunning()) {
            //si no hay un clip en start() (no suena nada). No puedo modificar volumen o se produce excepcion
            FloatControl volume = (FloatControl) clip.getControl(FloatControl.Type.MASTER_GAIN);
            float volActual = volume.getValue();
            System.out.println("Volumen actual " + volActual);
            if (volActual + ganancia > -10 && volActual + ganancia < 6.01) {
                volume.setValue(volActual + ganancia);
            } else if (volActual >= -10 && ganancia < 0) {
                volume.setValue(-80);
            } else if (volActual == -80 && ganancia > 0) {
                volume.setValue(-9);
            }
        }
    }

    @Override
    public void update(LineEvent event) {
        if (event.getType() == LineEvent.Type.STOP) {
            finCancion = true;

        }
    }
}
```

clase SonidoSintetizado

```
import javax.sound.midi.MidiChannel;
import javax.sound.midi.MidiSystem;
import javax.sound.midi.Synthesizer;
```

```
class SonidoSintetizado{

   public SonidoSintetizado(int instrumento){
      try {
         Synthesizer synth = MidiSystem.getSynthesizer();
         synth.open();
         MidiChannel miCanal = synth.getChannels()[0];
         int volume = 80; // entre  0 y  127
         int duration = 200; // en milisegundos
         miCanal.programChange(instrumento);
         int[] notas={60,62,64,65,67,69,71,72};
         for(int nota: notas){
            miCanal.noteOn( nota, volume );
            Thread.sleep(duration);
            miCanal.noteOff(nota);
         }
         Thread.sleep( 500 );
         synth.close();
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }

}
```

SE PIDE: Escribir la jerarquía anterior y un main que la utilice
Haz tu un main a tu estilo, pero como ejemplo, el profesor ejecuta una demostración y escucharás los sonidos. En el main debes de incluir un Audible[]
Tienes ficheros wav en una carpeta al principio del curso moodle, pero puedes usar tus propios ficheros.