

JSYN

LA LIBRERIA JSYN

- JSyn è una libreria per la sintesi audio in Java, utilizzabile per generare suoni, effetti, ambienti audio e musica
- Il paradigma è quello delle unità singole che possono essere connesse tra loro per creare suoni e ambienti complessi
- Si può usare per ambienti desktop ma anche per lo sviluppo Android
- È disponibile con licenza Apache V2, il codice è presente su GitHub
- <http://www.softsynth.com/jsyn/>

IMPORTARE JSYN (GRADLE)

- Per utilizzare l'API di JSyn in un progetto NetBeans (usando Gradle):
 - Si scarica il file jar dal sito ufficiale in una cartella del progetto (ad esempio creando `libs`)
 - Si aggiunge il file jar tra i riferimenti del progetto:
 - Tasto destro nell'area Projects su Libraries
 - Add JAR/Folder
 - Si seleziona il file jar

IMPORTARE JSYN (MAVEN)

- Per utilizzare l'API di JSyn in un progetto NetBeans (usando Maven):
 - Si aggiungono al file pom.xml (Project Files) le seguenti righe:

```
...
<repositories>
  <repository>
    <id>clojars.org</id>
    <url>https://repo.clojars.org</url>
  </repository>
</repositories>
<dependencies>
  <!-- https://mvnrepository.com/artifact/com.jsyn/jsyn -->
  <dependency>
    <groupId>com.jsyn</groupId>
    <artifactId>jsyn</artifactId>
    <version>20170815</version>
  </dependency>
</dependencies>
</project>
```

IMPORTARE JSYN

- Dopo aver importato la libreria, si importa nel sorgente con `import com.jsyn...`
 - `import com.jsyn.*`
 - Classe base e interfaccia sintetizzatore
 - `import com.jsyn.swing.*`
 - Classi dedicate a componenti swing (come la manopola)
 - `import com.jsyn.unitgen.*`
 - Generatori (come l'oscillatore sinusoidale)

PRIMI PASSI

- La prima cosa che occorre fare in JSyn è creare un sintetizzatore:
 - `Synthesizer synth = JSyn.createSynthesizer();`
- Per avviare poi il sintetizzatore creato si usa uno dei suoi metodi seguenti, che creano un thread in background per la generazione della sintesi audio:
 - `start()`: usa un frame rate di 44100 Hz e un output stereo
 - `start(int frameRate)`: usa il frame rate specificato e un output stereo
 - `start(int frameRate, int inputDeviceID, int numInputChannels, int outputDeviceID, int numOutputChannels)`
 - `frameRate, numInputChannels, numOutputChannels`: autoesplicativi!
 - `inputDeviceID, outputDeviceID`: si può usare per entrambi la costante `AudioDeviceManager.USE_DEFAULT_DEVICE` per il dispositivo di input/output di default oppure selezionare un particolare device (usando l'interfaccia `AudioDeviceManager`)
- Quando il programma termina, fermare il sintetizzatore con il suo metodo `stop()`

UNIT GENERATOR

- Dopo aver creato l'oggetto `Synthesizer` per produrre suoni occorre creare degli `Unit Generator`, che possono essere di diversi tipi:
 - Aritmetici e logici
 - Di controllo
 - Filtri
 - Rumore
 - Oscillatori e generatori
 - Trattamento campioni
 - Vari
- Tutti derivano dalla classe `UnitGenerator` (`com.jsyn.unitgen.UnitGenerator`)
- Uno dei più importanti è `LineOut`, che si occupa di mandare l'audio in output

UNIT GENERATOR

- I generatori devono essere creati e aggiunti al sintetizzatore con il metodo `add()`. Ad es.:

```
SineOscillator osc = new SineOscillator();  
synth.add(osc);  
LineOut lineOut = new LineOut();  
synth.add(lineOut);
```

- Esiste anche il metodo `remove()`, che rimuove uno unit generator precedentemente aggiunto. Ad es.:

```
synth.remove(osc);
```


COLLEGARE UNIT GENERATOR

- Gli unit generator possono essere connessi tra loro, in modo che l'output di uno di essi possa fungere da input di un altro
- Gli unit generator possono avere una proprietà `input` e una proprietà `output`
- Ogni output può essere connesso a diversi input e ogni input può avere in ingresso diversi output (che vengono in questo caso sommati tra loro)

COLLEGARE UNIT GENERATOR

- Per connettere tra loro gli unit generator si utilizza il metodo connect. Ad es.:

```
noise.output.connect(filter.input);
```

- Alcuni unit generator hanno porte costituite da più *parti* (*parts*), ad es. LineOut, che ha 2 input, uno per il canale sinistro e uno per il destro:

```
osc.output.connect(0, lineOut.input, 0);
```

(il primo e il terzo parametro corrispondono rispettivamente alla parte di output (in questo caso l'oscillatore ha una sola parte) e alla parte di input del secondo parametro (in questo caso il canale sinistro))

- Per scollegare uno unit generator da un altro si usa il metodo disconnect, specificando l'input da cui scollegarlo, oppure il metodo disconnectAll. Ad es.:

```
osc.output.disconnect(lineOut.input);
```

```
osc.output.disconnect(0, lineOut.input, 0);
```

```
osc.output.disconnectAll();
```

IMPOSTAZIONE DI PARAMETRI

- La maggior parte degli unit generator posseggono delle *porte* che controllano la loro operatività. Le proprietà `input` e `output` viste prima sono gli esempi più comuni (`input` è di classe `com.jsyn.ports.UnitInputPort`, mentre `output` è di classe `com.jsyn.ports.UnitOutputPort`). Altri esempi: `SineOscillator` possiede le due porte di input `frequency` e `amplitude`.
- Per impostare una porta si può usare il metodo `set()`. Ad es.:

```
osc.frequency.set(440.0); // Frequenza in Hz
osc.amplitude.set(0.5);   // Ampiezza tra 0..1
```
- Usando il metodo `connect` si possono impostare i valori delle porte all'output di altri unit generator, invece che a costanti. Ad es.:

```
osc2.frequency.set(2);
osc2.output.connect(osc.amplitude);
```

AVVIARE GLI UNIT GENERATOR

- Il sintetizzatore produce i suoni che arrivano dagli unit generator che sono stati avviati con il metodo `start()`
- Non è però necessario avviare esplicitamente tutti gli unit generator: quando uno di essi viene fatto partire, preleva i dati dai suoi input, facendo implicitamente partire gli unit generator connessi
- Questo significa che normalmente basta avviare l'ultimo unit generator della catena (e solitamente è `LineOut`):

```
lineOut.start()
```

TEMPORIZZAZIONE

- Il sintetizzatore di JSyn incorpora un timer interno che parte alla chiamata del suo metodo `start()`
- Per conoscere il tempo corrente (in secondi) si utilizza il metodo di `Synthesizer` `double getCurrentTime()`
- Per mettere in sleep il processo del sintetizzatore ci sono 2 metodi:
 - `synth.sleepFor(double duration)`
attende la generazione per `duration` secondi (si può accumulare ritardo)
 - `synth.sleepUntil(double time)`
attende la generazione fino al tempo d'esecuzione `time`(Nota: per entrambi i metodi va gestita l'eccezione `InterruptedException`)

SCHEDULAZIONE

- Per schedulare delle operazioni si possono usare i metodi che ricevono in input parametri di tipo `com.softsynth.shared.time.TimeStamp`, che rappresenta un istante di tempo del sintetizzatore
- Un oggetto `TimeStamp` si può ottenere:
 - dal costruttore `TimeStamp(double time)`, che crea un istante passandogli un parametro in secondi
 - da `Synthesizer`, usando il metodo `TimeStamp createTimeStamp()`, che crea un istante con l'attuale tempo di esecuzione
- `TimeStamp` possiede il metodo `TimeStamp makeRelative(double delta)`, che restituisce un nuovo istante situato `delta` secondi nel futuro

SCHEDULAZIONE

- Esempi di metodi che ricevono in ingresso un TimeStamp:
 - `start(TimeStamp timeStamp)`
Avvia un unit generator al tempo specificato
 - `stop(TimeStamp timeStamp)`
Ferma un unit generator al tempo specificato
 - `noteOn(double freq, double ampl, TimeStamp timeStamp)`
Negli oscillatori (`com.jsyn.unitgen.UnitOscillator`) suona una “nota” al tempo specificato
 - `noteOff(TimeStamp timeStamp)`
Negli oscillatori (`com.jsyn.unitgen.UnitOscillator`) ferma la generazione
 - `set(double value, TimeStamp timeStamp)`
Nelle porte di input (`com.jsyn.ports.UnitInputPort`) imposta il valore al tempo specificato
- Esempio di schedulazione di un parametro:

```
TimeStamp timeStamp = synth.createTimeStamp();  
TimeStamp futureTime = timeStamp.makeRelative(5.0);  
osc.frequency.set(220.0, futureTime);
```

ESEMPIO: GENERAZIONE SINUSOIDE

(PlayTone.java)

```
public class PlayTone {
    Synthesizer synth;
    UnitOscillator osc;
    LineOut lineOut;

    private void test() {
        synth = JSyn.createSynthesizer();
        synth.start();
        synth.add(osc = new SineOscillator());
        synth.add(lineOut = new LineOut());
        osc.output.connect(0, lineOut.input, 0);
        osc.output.connect(0, lineOut.input, 1);
        osc.frequency.set(345.0);
        osc.amplitude.set(0.6);
        lineOut.start();
        System.out.println("You should now be hearing a sine wave");
        try {
            double time = synth.getCurrentTime();
            System.out.println("time = " + time);
            synth.sleepUntil(time + 4.0);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Stop playing");
        synth.stop();
    }

    public static void main(String[] args) {
        new PlayTone().test();
    }
}
```


ESEMPIO: GENERAZIONE NOTE

(PlayNotes.java)

```
public class PlayNotes {
    Synthesizer synth;
    SineOscillator osc;
    LineOut lineOut;

    private void test() {
        synth = JSyn.createSynthesizer();
        synth.add(osc = new SineOscillator());
        synth.add(lineOut = new LineOut());
        osc.getOutput().connect(0, lineOut.input, 0);
        osc.getOutput().connect(0, lineOut.input, 1);
        synth.start();

        double timeNow = synth.getCurrentTime();
        TimeStamp timeStamp = new TimeStamp(timeNow + 0.5);
        lineOut.start(timeStamp);
        double freq = 220.0;
        double timeBetweenNotes = 1.0;
        double noteDuration = 0.3;
        osc.noteOn(freq, 0.5, timeStamp);
        osc.noteOff(timeStamp.makeRelative(noteDuration));
        timeStamp = timeStamp.makeRelative(timeBetweenNotes);
        freq *= 3.0 / 2.0; // up a perfect fifth
        osc.noteOn(freq, 0.5, timeStamp);
        osc.noteOff(timeStamp.makeRelative(noteDuration));
        timeStamp = timeStamp.makeRelative(timeBetweenNotes);
        freq *= 4.0 / 3.0; // up a perfect fourth
        osc.noteOn(freq, 0.5, timeStamp);
        osc.noteOff(timeStamp.makeRelative(noteDuration));

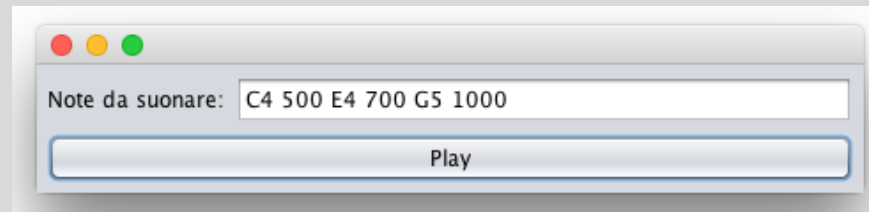
        try {
            synth.sleepUntil(timeStamp.getTime() + 2.0);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        synth.stop();
    }

    public static void main(String[] args) {
        new PlayNotes().test();
    }
}
```

ESERCIZIO

(PlayNotesSwing.java)

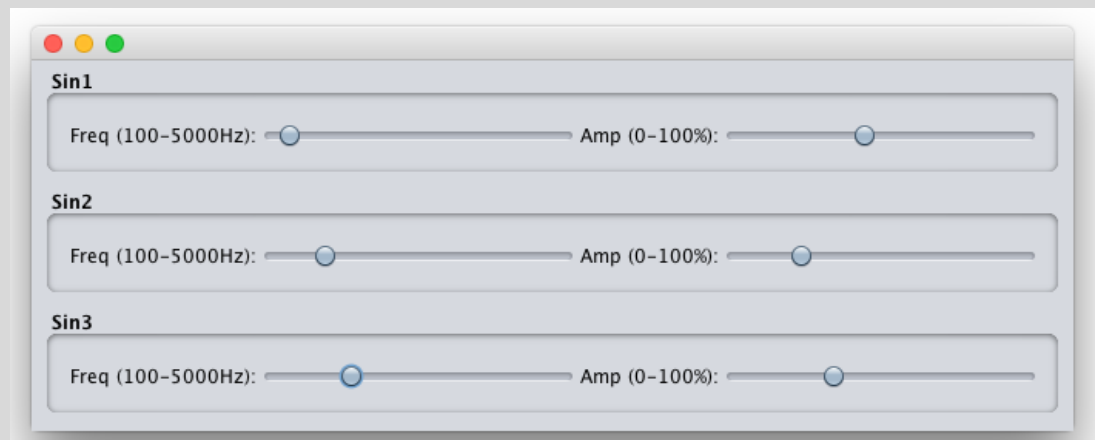
- Creare un'applicazione Swing che suoni una serie di note scritte in una casella di testo
- Si usi un oscillatore sinusoidale per la produzione delle note
- Il formato delle note nella casella di testo è il seguente: "nO time nO time nO time...", dove n è una nota (C,D,E,F,G,A,B), O è l'ottava, time espresso in millisecondi è il tempo di play della nota scritta prima



ESERCIZIO

(Additive.java)

- Creare un'applicazione Swing per la sintesi additiva: 3 segnali sinusoidali con frequenze e ampiezze diverse vengono sommati per formare un suono risultante
- Per la gestione di frequenze (in Hertz) e ampiezze (in percentuale) usare il componente Swing JSlider
 - Dalla palette di NetBeans è possibile aggiungerlo al JFrame
 - I parametri principali (accessibili dalle proprietà del controllo) sono minimo, massimo e valore attuale (minimum, maximum e value, settabili con i metodi `setMinimum()`, `setMaximum()` e `setValue()`)
 - Il valore corrente è restituito dal metodo `int getValue()`
 - L'evento di modifica della posizione è `ChangeEvent` (evento `stageChanged` in NetBeans)
- Alla modifica di un'ampiezza, per evitare clipping, occorre diminuire automaticamente gli altri 2 slider per arrivare alla somma del 100% massimo



USO DI CAMPIONI AUDIO

- Per utilizzare campioni audio per la riproduzione è disponibile la classe `com.jsyn.data.AudioSample`
- I campioni possono essere letti da file, caricati da uno stream o creati alitmicamente
- Esistono due sottoclassi di `AudioSample`:
 - `com.jsyn.data.FloatSample`: i dati sono numeri floating point a 32 bit (più usato)
 - `com.jsyn.data.ShortSample`: i dati sono interi short a 16 bit

CARICAMENTO DI CAMPIONI

- Per caricare campioni audio nelle strutture dati è presente la classe `com.jsyn.util.SampleLoader`, che ha i seguenti metodi principali (tutti statici):

- `FloatSample loadFloatSample(java.io.File fileIn)`
- `FloatSample loadFloatSample(java.io.InputStream inputStream)`
- `FloatSample loadFloatSample(java.net.URL url)`

- Esempi:

```
File sampleFile = new File("guitar.wav");  
FloatSample sample = SampleLoader.loadFloatSample(sampleFile);
```

CREAZIONE MANUALE DI CAMPIONI

- I campioni si possono anche creare alitmicamente, usando uno dei costruttori di FloatSample:
 - FloatSample(int numFrames): segnali mono della lunghezza passata in ingresso
 - FloatSample(int numFrames, int channelsPerFrame): segnali multi-canale della lunghezza passata in ingresso
 - FloatSample(float[] data): segnali mono con i dati passati in ingresso
 - FloatSample(float[] data, int channelsPerFrame): segnali multi-canale con i dati passati in ingresso
- Esempio: creazione di un'onda quadra di 50Hz

```
float[] data = new float[44100];
float value = 1.0f;
for (int i = 0; i < data.length; i++) {
    data[i] = value;
    if (i % 441 == 0)
        value = -value;
}
FloatSample sample = new FloatSample(data);
```

SUONARE CAMPIONI AUDIO

- Per suonare un campione audio caricato o generato si possono usare due classi:
 - `com.jsyn.unitgen.VariableRateMonoReader`: per segnali mono
 - `com.jsyn.unitgen.VariableRateStereoReader`: per segnali stereo
- Esempio:

```
VariableRateMonoReader samplePlayer = new VariableRateMonoReader();
```
- Entrambe le classi leggono i campioni ad una frequenza variabile e interpolano campioni vicini
- Per impostare la frequenza di lettura dei frame si usa la porta `rate`. Ad es.:

```
samplePlayer.rate.set(sample.getFrameRate() * 2);
```

Legge al doppio del frame rate originale, quindi riproduce un'ottava sopra
- Queste classi hanno una porta speciale chiamata `dataQueue`, che riceve campioni in ingresso per produrre il suono

ACCODARE CAMPIONI AUDIO

- Per accodare campioni audio sulla porta dataQueue sono disponibili – tra gli altri – i suoi metodi:
 - `queue(SequentialData queueableData)`: accoda i campioni passati
 - `queue(SequentialData queueableData, int startFrame, int numFrames)`: accoda `numFrames` campioni passati, partendo dalla posizione `startFrame`
 - `queueLoop(SequentialData queueableData)`: accoda i campioni passati mandando in loop la lettura
 - `queueLoop(SequentialData queueableData, int startFrame, int numFrames)`: accoda `numFrames` campioni passati, partendo dalla posizione `startFrame`, mandando in loop la lettura
 - `queueLoop(SequentialData queueableData, int startFrame, int numFrames, int numLoops)`: accoda `numFrames` campioni passati, partendo dalla posizione `startFrame`, mandando in loop la lettura per il numero di volte specificato

ACCODARE CAMPIONI AUDIO

- Quando si accodano campioni audio si possono prendere gli stessi da più "sorgenti": essi verranno suonati nell'ordine di inserimento in coda
- Quando viene specificato un loop senza impostare il numero di ripetizioni, esso continua fino a quando viene aggiunto qualche altro campione in coda. In altre parole, quando il sistema incontra un loop lo esegue, e ogni volta alla fine dello stesso controlla se ci sono altri dati successivi in coda: se non ci sono ritorna all'inizio del loop, altrimenti continua con i dati seguenti
- Un esempio di utilizzo quando si opera su un campione che ha una fase d'attacco lunga `attackSize` frame, una fase di sustain lunga `sustainSize` frame e una fase di release lunga `releaseSize` frame potrebbe essere il seguente:

```
samplePlayer.dataQueue.queue(mySample, 0, attackSize);  
samplePlayer.dataQueue.queueLoop(mySample, attackSize, sustainSize, 5);  
samplePlayer.dataQueue.queue(mySamp, attackSize + sustainSize, releaseSize);
```

ESEMPIO: GENERAZIONE ONDA QUADRA

(PlaySquare.java)

```
public class PlaySquare {
    Synthesizer synth;
    UnitOscillator osc;
    LineOut lineOut;
    VariableRateDataReader samplePlayer;

    private void test() throws InterruptedException {
        synth = JSyn.createSynthesizer();
        synth.start();
        synth.add(lineOut = new LineOut());
        lineOut.start();
        float[] data = new float[44100];
        float value = 0.7f;
        for (int i = 0; i < data.length; i++) {
            data[i] = value;
            // cambia segno ogni 441 valori (50Hz visto che il sample rate è 44100 di default)
            if (i % 441 == 0)
                value = -value;
        }
        FloatSample sample = new FloatSample(data);
        synth.add(samplePlayer = new VariableRateMonoReader());
        // imposta la velocità di lettura dei frame
        // al doppio del frame rate: 8va
        samplePlayer.rate.set(sample.getFrameRate() * 2);
        samplePlayer.output.connect(0, lineOut.input, 0);
        samplePlayer.output.connect(0, lineOut.input, 1);
        samplePlayer.dataQueue.queueLoop(sample, 0, sample.getNumFrames(), 1);
        do {
            synth.sleepFor(1.0);
        } while (samplePlayer.dataQueue.hasMore());
        synth.stop();
    }

    public static void main(String[] args) throws InterruptedException {
        new PlaySquare().test();
    }
}
```

ESEMPIO: PLAY DI UN FILE AUDIO

(PlaySample.java)

```
public class PlaySample {
    Synthesizer synth;
    VariableRateDataReader samplePlayer;
    LineOut lineOut;
    private void test() throws IOException, InterruptedException {
        synth = JSyn.createSynthesizer();
        synth.start();
        synth.add(lineOut = new LineOut());
        File sampleFile = new File("beep.wav");
        FloatSample sample = SampleLoader.loadFloatSample(sampleFile);
        if (sample.getChannelsPerFrame() == 1) {
            synth.add(samplePlayer = new VariableRateMonoReader());
            samplePlayer.output.connect(0, lineOut.input, 0);
        } else if (sample.getChannelsPerFrame() == 2) {
            synth.add(samplePlayer = new VariableRateStereoReader());
            samplePlayer.output.connect(0, lineOut.input, 0);
            samplePlayer.output.connect(1, lineOut.input, 1);
        } else {
            throw new RuntimeException("Formato non supportato");
        }
        samplePlayer.rate.set(sample.getFrameRate());
        lineOut.start();
        samplePlayer.dataQueue.queue(sample);
        do {
            synth.sleepFor(1.0);
        } while (samplePlayer.dataQueue.hasMore());
        synth.sleepFor(0.5);
    }
    public static void main(String[] args) throws IOException, InterruptedException {
        new PlaySample().test();
    }
}
```

ESERCIZIO

(PlaySampleSwing.java)

- Creare un'applicazione Swing che suoni una serie di intervalli scritti in una casella di testo
- Si carichi un file audio contenente il campione da riprodurre
- Il formato delle note nella casella di testo è il seguente: "i1 i2 i3", dove "in" è un intervallo in semitoni rispetto alla nota base del campione (può essere negativo)
- Si consideri il file audio come "nota di partenza", su cui poi effettuare le esecuzioni dipendenti dagli intervalli della casella di testo
- Le esecuzioni del campione si susseguono una dopo l'altra; da notare che anche il tempo di esecuzione dipende dalla velocità di lettura del campione: a note più alte corrisponde un minore tempo di esecuzione

INVILUPPI

- Spesso è necessario controllare i parametri alla base degli unit generator con degli *inviluppi* che descrivano l'evoluzione del parametro
- Il caso più comune è l'inviluppo d'ampiezza di un suono, con le varie fasi di attacco, decadimento, sostegno e rilascio (ADSR)
- In JSyn si può usare la classe `com.jsyn.data.SegmentedEnvelope`, che descrive un inviluppo come serie di segmenti di linee
- La creazione di un inviluppo è simile alla creazione di un campione audio, la differenza è che in questo caso il singolo frame contiene coppie di valori che descrivono durata e valore
- Per creare un nuovo inviluppo, si può definire un array di `double`, formato da coppie di valori indicanti:
 - Quanto tempo occorre per arrivare al valore specificato
 - Il valore a cui arrivare, a partire dal valore del frame precedente

INVILUPPI

- Una volta creato l'array di valori, si crea un nuovo `SegmentedEnvelope` con il relativo costruttore
- Esempio:

```
double[] data = {  
    0.02, 1.0, // duration,value pair for frame[0]  
    0.30, 0.1, // duration,value pair for frame[1]  
    0.50, 0.7, // duration,value pair for frame[2]  
    0.50, 0.9, // duration,value pair for frame[3]  
    0.80, 0.0 // duration,value pair for frame[4]  
};  
SegmentedEnvelope envelope = new SegmentedEnvelope(data);
```

In questo esempio l'involuppo parte da 0 (default) e arriva al valore 1.0 in 0.02 secondi, poi passa al valore 0.1 in 0.30 secondi, poi al valore 0.7 in 0.50 secondi, ecc.

USO DI INVILUPPI

- Per usare un involuppo per il controllo di un parametro di un unit generator occorre creare un lettore di involuppo come il già visto `VariableRateMonoReader`
- Al lettore creato vengono accodati i dati dell'involuppo con i metodi `queue` e `queueLoop`

- Ad esempio, partendo dall'involuppo creato in precedenza:

```
VariableRateMonoReader envPlayer = new VariableRateMonoReader();  
envPlayer.dataQueue.queue(envelope);  
envPlayer.output.connect(osc.amplitude);
```

crea un player, accoda i dati dell'involuppo e li usa per controllare l'ampiezza di un oscillatore. In alternativa alla seconda riga:

```
envPlayer.dataQueue.queue(envelope, 0, 3);  
envPlayer.dataQueue.queueLoop(envelope, 1, 2, 5);  
envPlayer.dataQueue.queue(envelope, 3, 2);
```

presenta una fase d'attacco di 0.02+0.30+0.50 secondi, poi crea un loop di 5 ripetizioni che fa variare i valori tra 0.1 e 0.7 in $5 * (0.30+0.50)$ secondi, per poi presentare una fase di release di 0.50+0.80 secondi

ESERCIZIO

(PlayToneADSRSwing.java)

- Creare un'applicazione Swing che presenti un solo pulsante Play, alla cui pressione/rilascio venga suonato un segnale sinusoidale con un certo inviluppo d'ampiezza
- L'inviluppo deve presentare:
 - un attacco della durata di 0.02 secondi, fino al valore d'ampiezza di 0.9
 - un decadimento della durata di 0.1 secondi, fino al valore d'ampiezza 0.7
 - un sostegno che dura fino al rilascio del pulsante
 - un rilascio della durata di 0.5 secondi
- Suggestimenti:
 - alla pressione del pulsante è opportuno svuotare la coda dell'inviluppo con il metodo `dataQueue.clear()`
 - per intercettare il pulsante premuto/rilasciato si possono usare i gestori degli eventi `mousePressed` e `mouseReleased`

INPUT DA MICROFONO

- Per usare l'input del segnale microfonico occorre specificare il numero di canali di ingresso quando si avvia il sintetizzatore:

```
synth.start(44100,  
AudioDeviceManager.USE_DEFAULT_DEVICE, 2,  
AudioDeviceManager.USE_DEFAULT_DEVICE, 2);
```

- Se si tratta di un segnale stereo, per ottenere l'input audio si usa `LineIn`:

```
synth.add(lineIn = new LineIn());
```

- `LineIn` ha una porta `output` che veicola il segnale in ingresso

PITCHDETECTOR E MULTIPLY

- L'unità `com.jsyn.unitgen.PitchDetector` si può usare per stimare la frequenza di un segnale
- `PitchDetector` possiede le porte:
 - `input`: unica porta di input, riceve il segnale da analizzare
 - `frequency`: porta di output con la frequenza stimata
 - `period`: porta di output con il periodo stimato
 - `confidence`: porta di output che indica l'accuratezza presunta dell'operazione di stima della frequenza (tra 0 e 1)
- Ci sono molti altri unit generator, anche semplicemente dedicati a svolgere operazioni aritmetiche o logiche. Ad esempio esiste `Multiply`, che ha due porte di input (`inputA` e `inputB`) e una porta di output in cui viene mandato il prodotto $\text{inputA} * \text{inputB}$

ESERCIZIO

(PitchDetectorTest.java)

Creare un'applicazione che:

- Riceva in input il segnale microfonico
- Effettui il riconoscimento del pitch del segnale microfonico
- Crei due segnali audio sinusoidali in uscita (uno per il canale sinistro e uno per il canale destro), la cui frequenza sia la metà e il doppio del pitch riconosciuto

Suggerimento: si usi la porta confidence di PitchDetector per controllare l'ampiezza delle sinusoidi

LA CLASSE CIRCUIT

- Quando si crea un insieme di unità collegate tra loro che formano una soluzione completa può essere utile poter creare una nuova *macrounità* che *inscatoli* l'intero sistema
- La classe `Circuit` serve a questo: con il suo utilizzo si possono utilizzare i circuiti creati esattamente come le altre unità predefinite (con porte di ingresso/uscita, ecc.)
- Si definisce una sottoclasse di `Circuit`:

```
public class WindSound extends Circuit
```
- All'interno della classe si definiscono le unità incluse, e nel costruttore vengono effettuate le operazioni preliminari:
 - Creazione delle istanze delle unità definite
 - Aggiunta dell'unità al circuito, come succede per i sintetizzatori (metodo `add`)
 - Aggiunta di porte di ingresso/uscita del circuito

ESEMPIO: SUONO DEL VENTO CON FILTRAGGIO DI RUMORE BIANCO

(WindCircuit.java e WindSoundUse.java)

SYNTONA

- Per creare patch in jSyn per via grafica è stato sviluppato Syntona (scaricabile dal sito di jSyn)
- Syntona è un editor in stile PD, Max/MSP... che permette di creare patch jSyn all'interno di un ambiente grafico integrato
- Permette di ascoltare direttamente il risultato ottenibile dalla patch
- Permette di generare il codice sorgente Java che implementa le patch create

ESEMPIO D'USO DI SYNTONA

- Aprire in Syntona il progetto TestSawVoiceVibrato.xml
- Abilitare la tastiera come input
 - CAPS LOCK oppure
 - Play Keys (nella toolbar)
- Usando i tasti della tastiera (I-7, Q-U, A-J, Z-M), il suono viene riprodotto direttamente all'interno di Syntona
- Aprire la patch interna di generazione (doppio clic su voice)
- Esportare in Java (File > Export Java Source As...)
- Viene generato un file Java che può essere poi incluso in soluzioni che utilizzano il suono della patch

- Esempio: **Syntona.java**

ESEMPIO D'USO DI SYNTONA (2)

(**syntona1.xml**)

- Creazione di una nuova patch:
 - Add Voice
 - jSyn/lineOut
 - Doppio clic su *voice* (si apre la finestra di progettazione della voce)
 - jSyn/sources/sineOsc
 - jSyn/misc/aInput (x2) + jSyn/misc/aOutput
 - Rinominare i 2 input a *frequency* e *amplitude* (doppio clic)
 - Rinominare *aOutput* a *output*
 - Collegare input e output
 - Collegare nella patch principale l'output di *voice* a *lineOut*
 - La patch inizia a emettere il suono (abilitando la tastiera si può suonare con la stessa)

ESERCIZIO

(syntona2.xml)

- Modificare la patch precedente per creare un suono con tremolo
- Il segnale modulante ha una frequenza che può essere fatta variare tra 1 e 30 Hz