

Kwon-Young Choi  
Julien De Loor  
Maxime Cadoret  
Nadher Ali



## Projet d'informatique S6P

# LE SYNTHETISEUR MILLENNIUM V1.1

## Sommaire

I-Présentation générale.....	3
1) L'interface graphique.....	3
2) Générer le son.....	3
3) Captures d'écran.....	4
II-Diagramme de fonctionnalités (Usecases).....	5
III-Scénarios.....	6
IV-Diagrammes de séquence.....	8
V- Détails des classes (class & state machine).....	11
Signal .....	11
AbstractSignalModifier.....	11
AbstractSignalGenerator.....	12
Oscillator (Oscillateur).....	13
InstrumentVoice .....	14
AbstractInstrument.....	15
Instrument.....	15
Note .....	16
InstrumentParameter .....	17
AudioStream .....	18
AudioDriver .....	19
MouseCatcher.....	20
Knob .....	21
ScrollBar .....	22
Interface .....	23
VI – Diagrammes d'activité .....	24
1) Jouer une note de musique.....	24
2) Modification d'un paramètre de l'instrument.....	25
3) Fonctionnement du driver.....	26
4) Fonctionnement général du programme.....	27
VII – Ressources et informations complémentaires.....	28

# I-Présentation générale

Le but du projet est de réaliser un outil de génération de musique virtuel en synthétisant le son en « temps réel » (le son n'est pas pré-enregistré depuis un instrument réel mais est créé directement par le programme). Pour cela nous nous baserons sur des éléments couramment utilisés dans la synthèse de son : oscillateurs, modulation de fréquence, filtres...

L'intérêt est de proposer un moyen de créer de la musique juste avec son ordinateur, avec ou sans instrument physique. L'autre intérêt est de pouvoir modifier le son comme on le souhaite.

Fondamentalement, notre programme se divise en deux parties. La partie génération de son et la partie graphique qui fera l'interface avec l'utilisateur.

## 1) L'interface graphique

L'interface sera composée dans un premier temps uniquement d'un panneau de visualisation des paramètres de l'instrument, et la gestion des touches se fera uniquement via le clavier de l'ordinateur.

Par la souris, nous pourrions régler différents paramètres comme le volume du son joué par des boutons circulaires tournant (potentiomètre ou knob en anglais) et d'autres boutons.

## 2) Générer le son

Voici la logique de génération de son. Tout d'abord, notre programme proposera différents timbres sonores ici appelés des voix d'instruments. Les voix d'instruments (instrument voice) sont des agglomérats d'oscillateur de base sur lesquels sont effectués des opérations de base (addition, mixage ...). C'est aussi dans les voix d'instruments que sont utilisés tous les paramètres réglable sur l'interface graphique, par exemple : le rapport cyclique pour un signal carré.

Un oscillateur est en fait une formule mathématique, qui décrit des signaux de base (signal sinusoïdal, signal carré, signal triangulaire, bruit blanc ...).

L'utilisateur choisit des paramètres de l'instrument via l'interface graphique et appuie sur une touche. L'information est transmise à un gestionnaire de voix d'instrument qui, dans notre cas s'appellera instrument. L'instrument va affecter une de ses voix pour générer la note demandé par l'utilisateur, si toutes ses voix sont déjà en train de jouer une note il le signalera via l'interface graphique. Quand une voie est affectée à une note alors elle utilise ses oscillateurs pour générer les signaux sonores.

La conversion analogique numérique (c'est-à-dire passer de valeurs continues, ici des formules mathématiques, à des valeurs discrètes) est indispensable car le haut parleur crée le son par différentes positions discrètes de sa membrane. C'est ici que l'outil signal intervient. Un signal est un tableau d'échantillons généré par l'instrument selon les notes appuyées et des paramètres sélectionnés.

Enfin, le signal est envoyé au haut parleur par l'intermédiaire des outils

fournis par des bibliothèques comme BASS ou BASSASIO.

### 3) Captures d'écran

Instrument : NELEAD 6

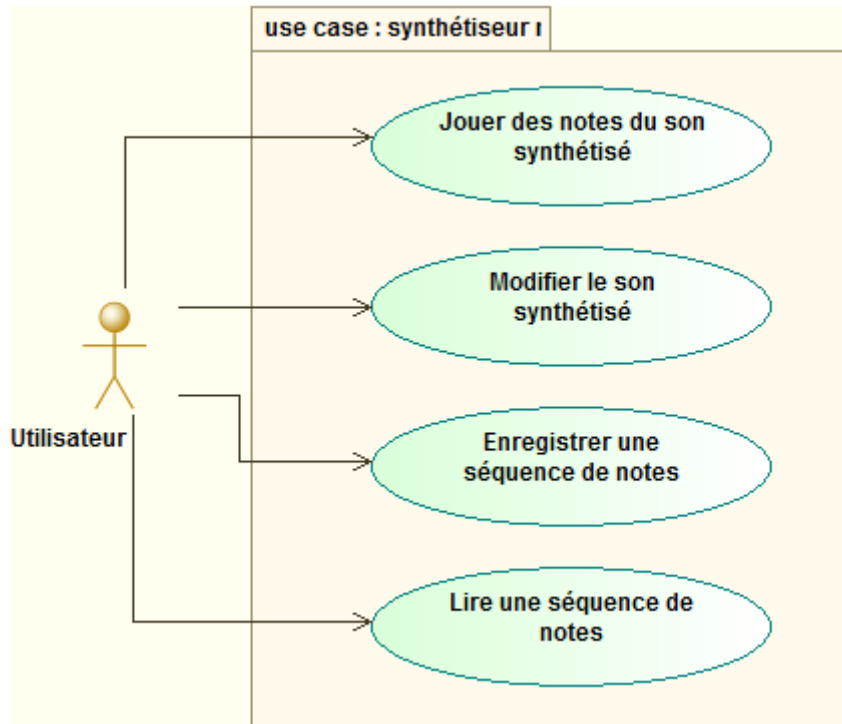


Instrument : pure square synthétiseur



## II-Diagramme de fonctionnalités (Usecases)

Les 4 fonctionnalités principales de notre synthétiseur:



Les 2 premières fonctionnalités seront disponible dès la version 1

- Jouer des notes du son synthétisé
- Modifier le son synthétisé

Les deux fonctionnalités suivantes seront implémentés dans une prochaine version :

- Enregistrer une séquence de notes
- Lire une séquence de notes

### III-Scénarios

Voici les scénarios d'utilisation liés aux cas d'utilisations, comme les 2 premiers cas ne sont pas indissociables nous les avons regroupés dans le premier tableau.

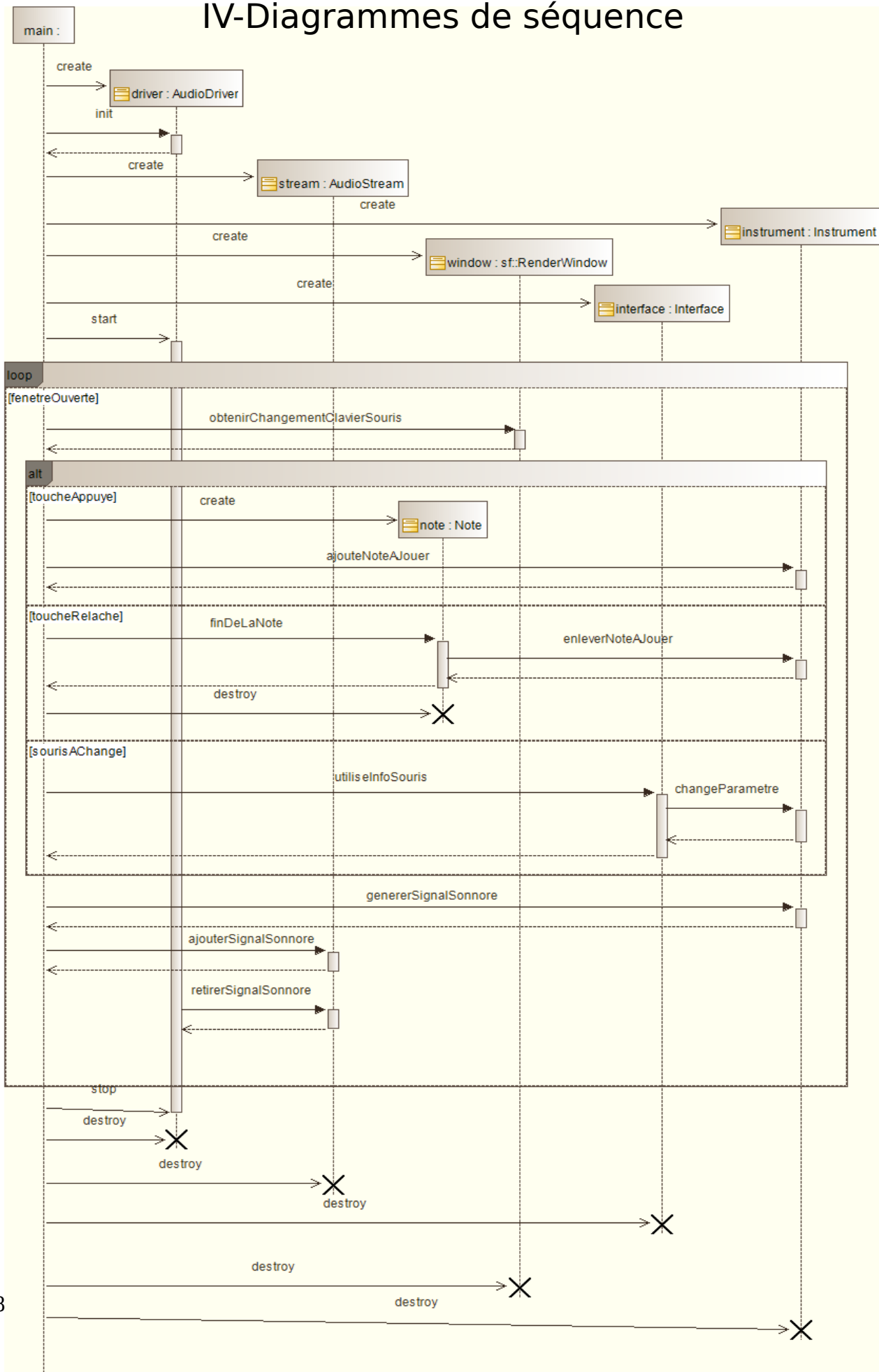
Nom	Jouer des notes du son synthétisé
Scénario d'utilisation	Scénario optimal : 1) Régler différents paramètres de l'instrument 2) Appuyer sur une touche 3) Jouer un son 4) Relâcher la touche 5) Recommencer selon la volonté de l'artiste
	Scénario alternatif : 1) Régler différents paramètres de l'instrument 2) Appuyer sur une touche 3) Jouer un son 4) Relâcher la touche 5) Changer un réglage 6) Recommencer selon la volonté de l'artiste
	Scénario d'erreur: 1) Appuyer sur une touche 2) Problème de librairie audio (par exemple pas de haut parleur) 3) Affichage d'un message d'erreur correspondant

Les scénarios suivant ne seront pas disponibles dans la première version du programme...

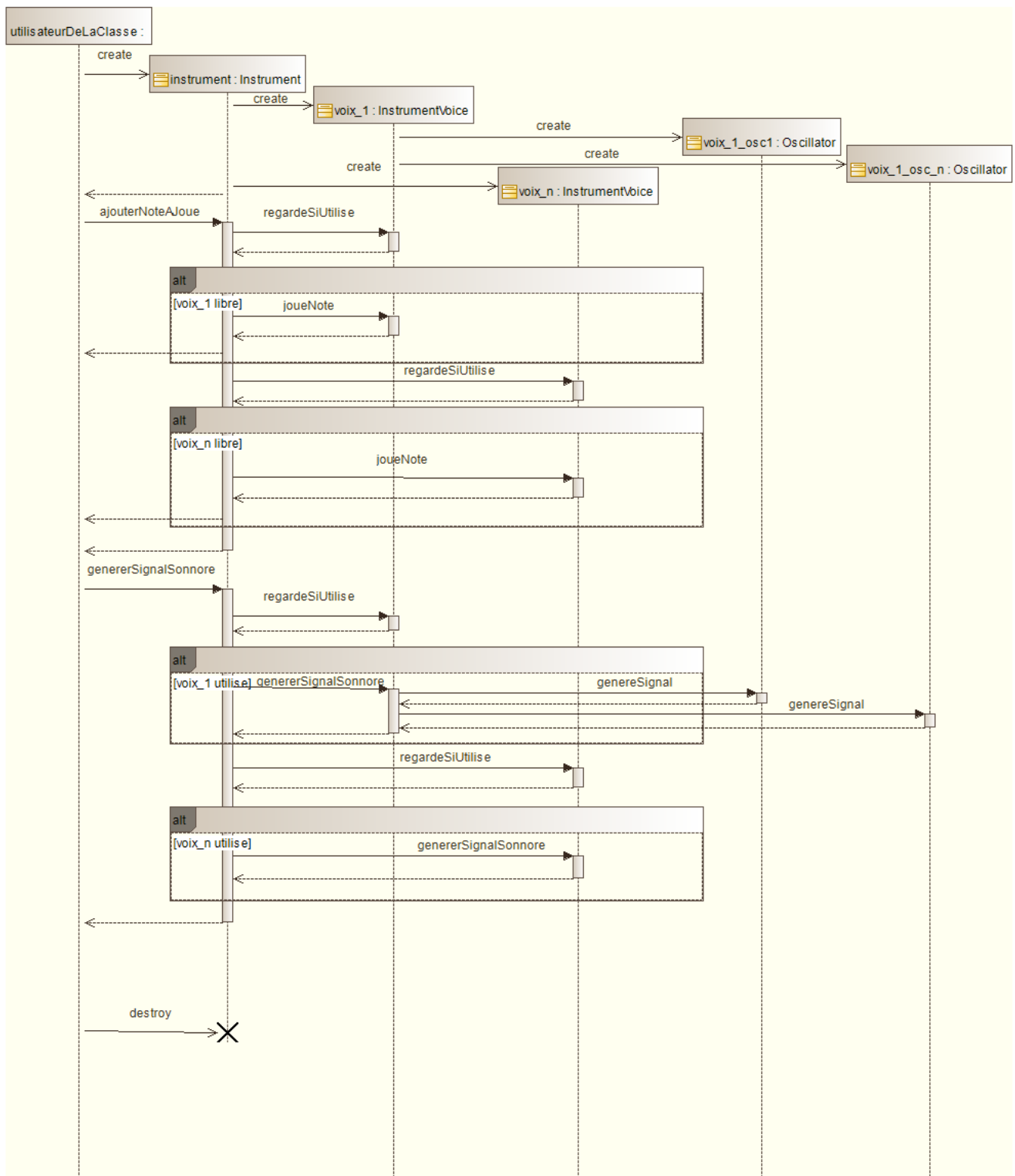
Nom	Lire une séquence de notes
Scénario d'utilisation	Scénario optimal : 1) Sélectionner "ouvrir" 2) Choisir le fichier 3) Appuyer sur la Touche "Jouer" pour jouer l'enregistrement
	Scénario optimal : 1) Sélectionner "ouvrir" 2) Choisir le fichier 3) Appuyer sur la Touche "Jouer" pour jouer l'enregistrement 4) Après écoute il s'agissait du mauvais fichier

	5) Sélectionner "ouvrir" 6) Choisir le fichier 7) Appuyer sur la Touche "Jouer" pour jouer l'enregistrement
	Scénario d'erreur: 1) Sélectionner "ouvrir" 2) Choisir le fichier 3) Erreur le fichier n'est pas un fichier de séquence de notes 4) Affichage du message d'erreur
Nom	Enregistrer une séquence de notes
Scénario d'utilisation	Scénario optimal : 1) Appuyer sur la Touche "Enregistrer" pour commencer l'enregistrement 2) Jouer des notes du son synthétisé 3) Appuyer sur la Touche "Enregistrer" pour arrêter l'enregistrement 4) Saisir le nom du fichier 5) Enregistrer le fichier
	Scénario alternatif : 1) Appuyer sur la Touche "Enregistrer" pour commencer l'enregistrement 2) Jouer des notes du son synthétisé 3) Appuyer sur la Touche "Enregistrer" pour arrêter l'enregistrement 4) Appuyer sur la Touche "Jouer" pour rejouer l'enregistrement 5) Saisir le nom du fichier 6) Enregistrer le fichier
	Scénario d'erreur: 1) Appuyer sur la Touche "Enregistrer" pour commencer l'enregistrement 2) Jouer des notes du son synthétisé 3) Appuyer sur la Touche "Enregistrer" pour arrêter l'enregistrement 4) Saisir le nom du fichier 5) Nom du fichier incorrecte ou impossible de l'ouvrir 6) Affichage d'un message d'erreur 7) Saisir le nom du fichier 8) Enregistrer le fichier

## IV-Diagrammes de séquence







Comme le premier et 2ième scénario d'utilisation utilisent des fonctionnalités croisées et donc les diagrammes seront similaires : nous avons choisi de regrouper les 2 premiers scénarios

Le premier diagramme montre le déroulement et la transmission des informations entre l'interface/clavier et le générateur de son (classe

instrument), tandis que le second détaille le fonctionnement interne de la classe instrument lorsque les différentes méthodes (appelées dans le premier diagramme) sont utilisés. Ici le diagramme est en français pour faciliter la compréhension mais nous préférons l'anglais par la suite.

## V- Détails des classes (class & state machine)

### Signal

---

Signal	
+ samples : float [size]	
+ add(in s: Signal)	
+ mix(in s: Signal)	
+ addOffset(in offset: float)	
+ scale(in scale: float)	
+ constant(in constant: float)	
+ reset()	
+ Signal()	CC
+ Signal(in signal: Signal)	CC
+ ~Signal()	DC

Unité de "son", tout son continu est décomposé en signaux (un tableau d'échantillons), c'est ce qu'un instrument doit générer. On peut le modifier avec des opérations classiques du traitement du signal (ajout d'un offset, addition, multiplication, etc).

### ***AbstractSignalModifier***

---

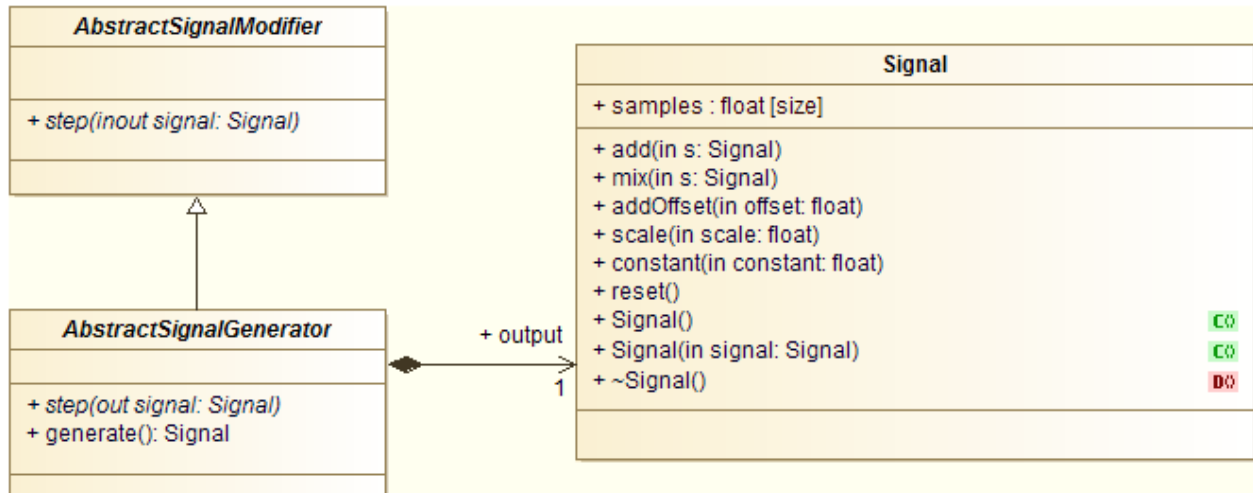
<i>AbstractSignalModifier</i>
+ <i>step(inout signal: Signal)</i>

Toute classe qui hérite de AbstractSignalModifier doit fournir la méthode step qui modifie le signal que l'on lui fournit.

---

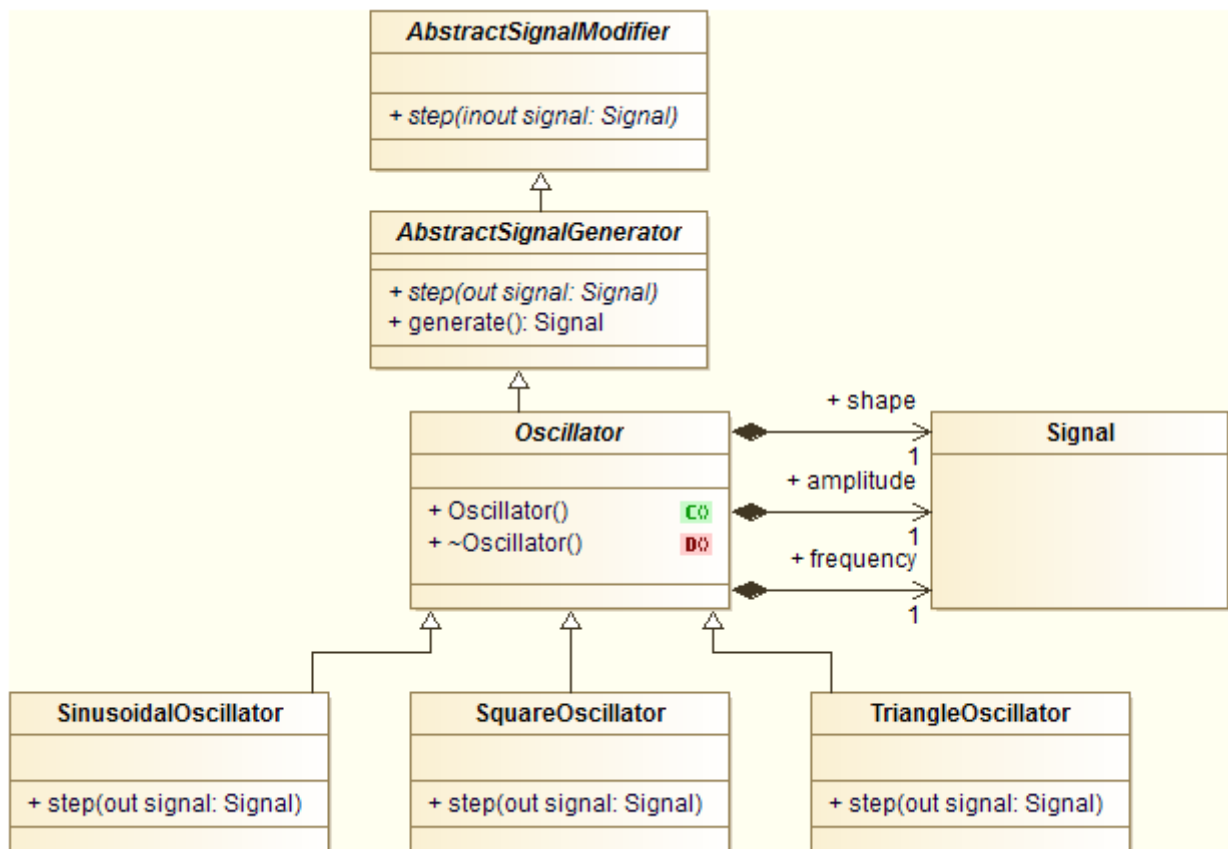
## AbstractSignalGenerator

---



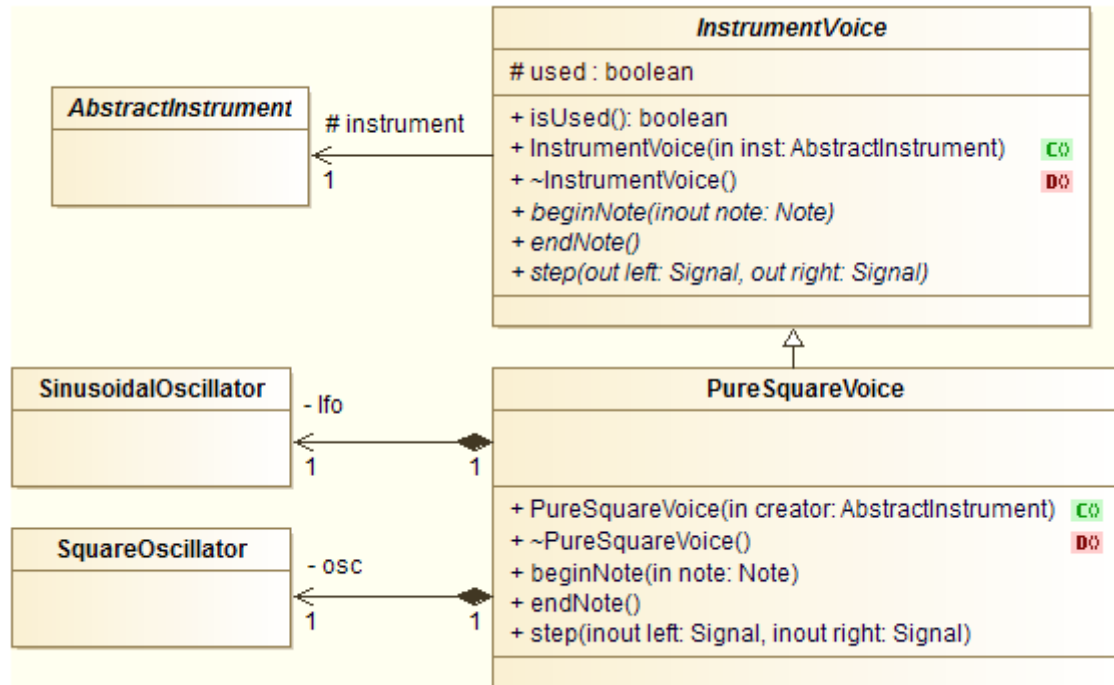
AbstractSignalGenerator se sert de la méthode `step` pour générer son signal de sortie. La méthode `step` est donc uniquement valable en sortie. Le générateur peut également générer le signal sur son attribut `output` et retourner cet attribut via la méthode `generate` (tout dépend de ce que l'utilisateur veut faire...).

## Oscillator (Oscillateur)

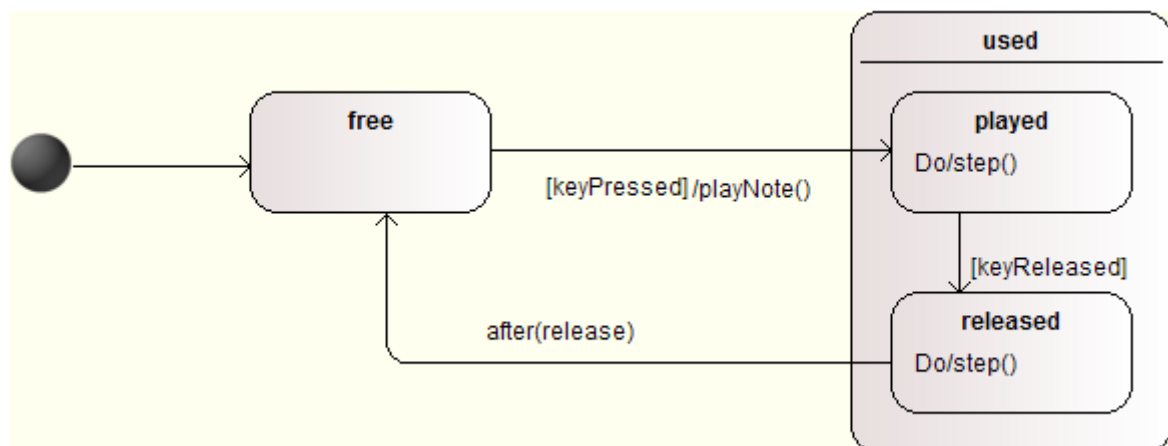


L'Oscillator est un générateur de signal périodique en fonction de 3 paramètres : l'amplitude, la fréquence et la forme. Il génère un signal grâce à la méthode "generate", cependant il est aussi possible de générer le signal sur un objet Signal extérieur grâce à la méthode "step". L'oscillateur a une horloge interne qu'il met lui même à jour. Oscillator est une classe abstraite, les classes concrètes qui en hérites doivent juste fournir la méthode step() qui génère le signal...

## InstrumentVoice



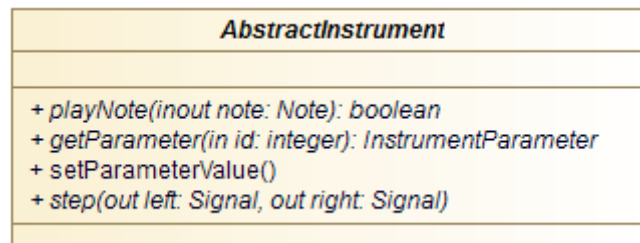
L'InstrumentVoice est une classe abstraite, c'est un générateur de signal (comme un oscillateur). Le signal qui en sort est le son produit par une seule note. Sur ce schéma la classe PureSquareVoice génère le son de la note grâce à un oscillateur carré et un LFO (oscillateur de modulation de paramètre). L'instrument PureSquare se charge de transmettre les paramètres que l'utilisateur modifie via l'interface graphique à chacun des PureSquareVoice. Ci-dessous son diagramme uml stm behaviour (machine à états) :



---

## AbstractInstrument

---

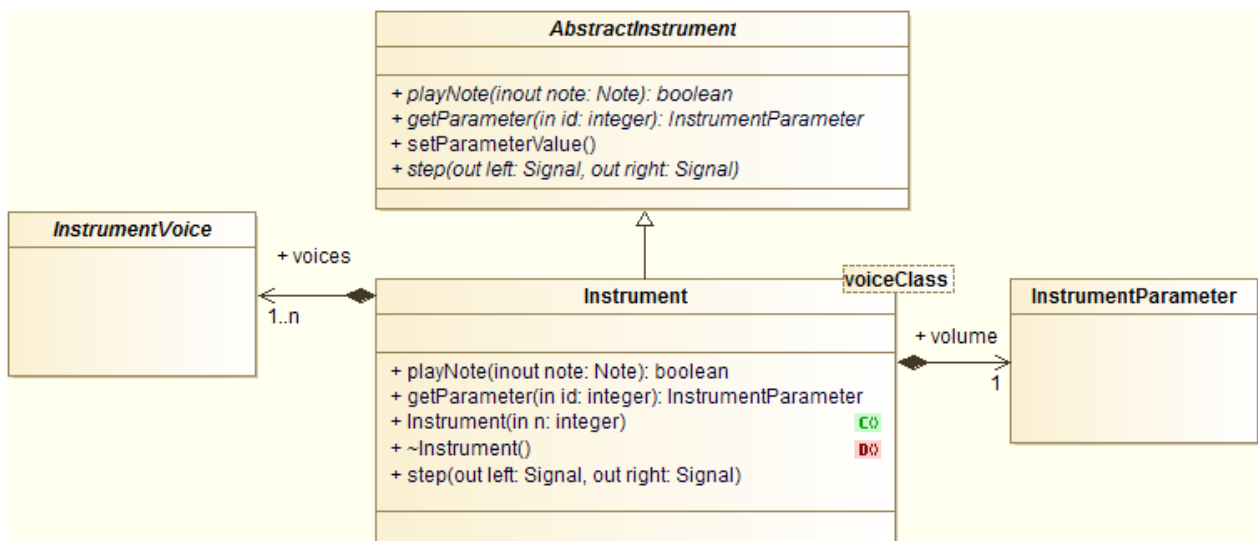


L'AbstractInstrument est la base d'un Instrument. Elle permet à un gestionnaire d'instrument de gérer les différents "Instrument" sans connaître les "InstrumentVoice" associés.

---

## Instrument

---



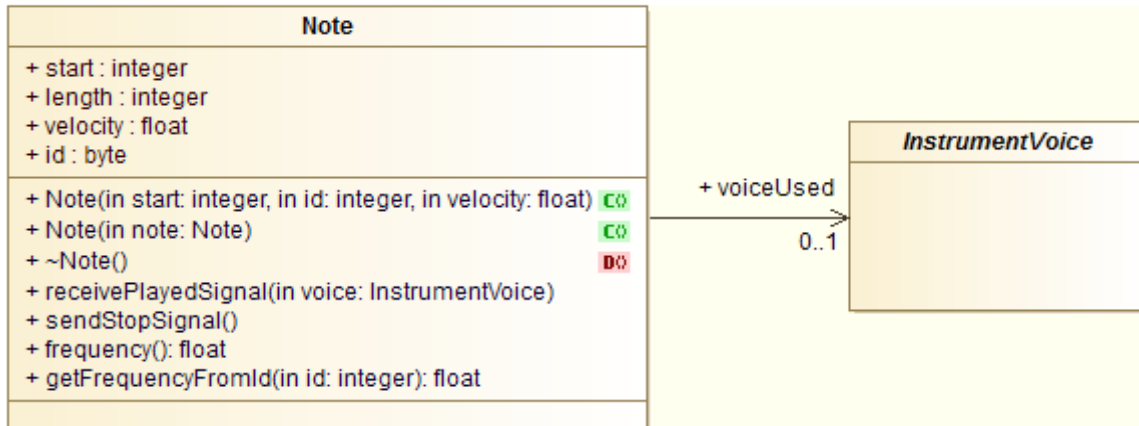
L'Instrument est un modèle, il regroupe plusieurs InstrumentVoice, et affecte chacun d'entre eux à une note qu'il reçoit. Il fait également le mixage final entre tous les signaux générés par ces InstrumentVoice. La classe Instrument a déjà un minimum de code fournis (gestion de la polyphonie des notes). Il devient donc facile de créer un instrument en héritant simplement de cette classe

N.B : polyphonie = plusieurs notes.

---

## Note

---



Elle représente une note de musique : fréquence, vélocité, durée mais aussi quand elle a été jouée ! Quand une note est créée elle est envoyée à l'Instrument (via la méthode `playNote(Note & n)`) qui l'affectera, si il le peut, à un **InstrumentVoice**. Pour cela on utilisera la méthode `beginNote(Note& n)` qui elle-même préviendra la note via `receivePlayedSignal(InstrumentVoice* v)`.



Lorsqu'elle est terminée la note doit prévenir l'**InstrumentVoice** (méthode `sendStopSignal()`) afin que celui-ci démarre son Release Time (méthode `endNote()`). Ainsi, une note peut être terminée mais l'**InstrumentVoice** continue d'être "utilisé" tant qu'il n'a pas fini son release time.



---

## InstrumentParameter

---

InstrumentParameter	
<ul style="list-style-type: none"><li>- modified : boolean</li><li>- value : short</li><li>- min : short</li><li>- max : short</li></ul>	
<ul style="list-style-type: none"><li>+ InstrumentParameter(in value: short, in min: short, in max: short)</li><li>+ InstrumentParameter(in instrumentParameter: InstrumentParameter)</li><li>+ operator=(in instrumentParameter: InstrumentParameter): InstrumentParameter</li><li>+ active(): boolean</li><li>+ toggle()</li><li>+ on()</li><li>+ off()</li><li>+ setValueFromUnsigned(in value: integer, in max: integer)</li><li>+ getValueToUnsigned(in max: integer)</li><li>+ getRanged(): short</li><li>+ valueUsed()</li><li>+ notify()</li></ul>	<ul style="list-style-type: none"><li></li><li></li></ul>

Elle représente un paramètre de l'instrument codé sur un entier signé de 16 bits. A l'initialisation l'instrument donne la plage de valeur (que le paramètre ne doit pas dépasser). Ensuite différentes méthodes peuvent modifier cette valeur en utilisant une autre plage de valeurs (par exemple `setValueFromUnsigned`). `InstrumentParameter` sert à faire le lien( via une remise à l'échelle) entre le potentiomètre et l'`Instrument`.

---

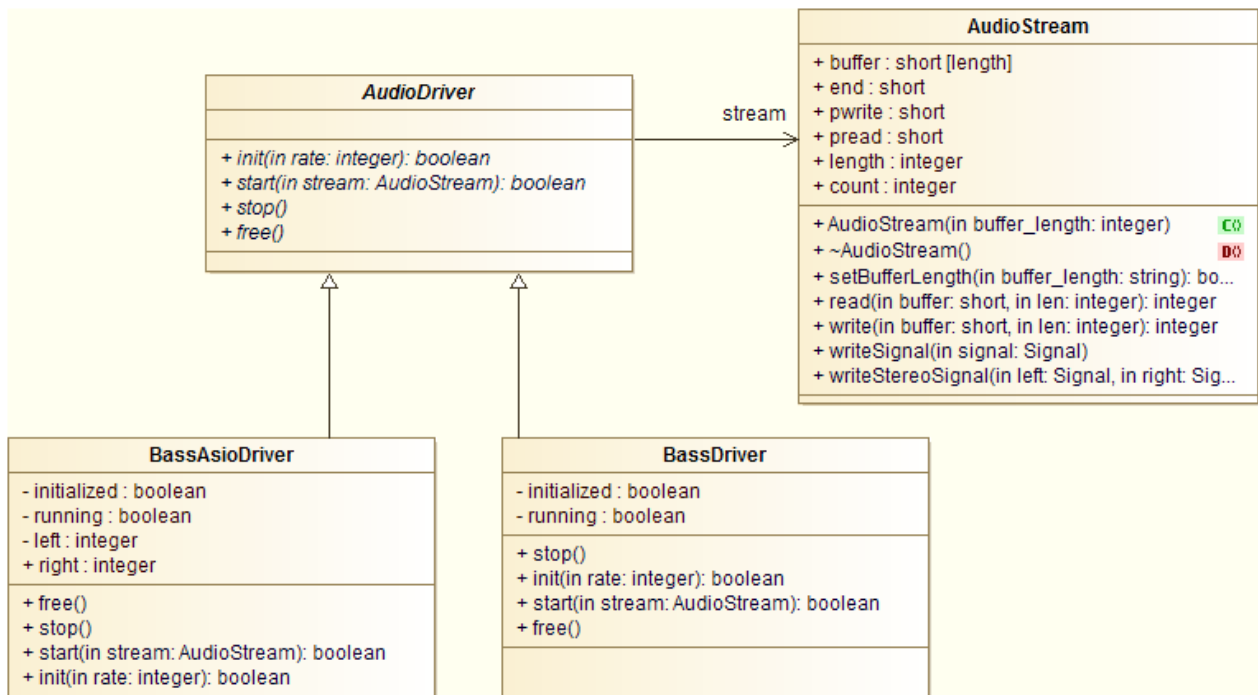
## AudioStream

---

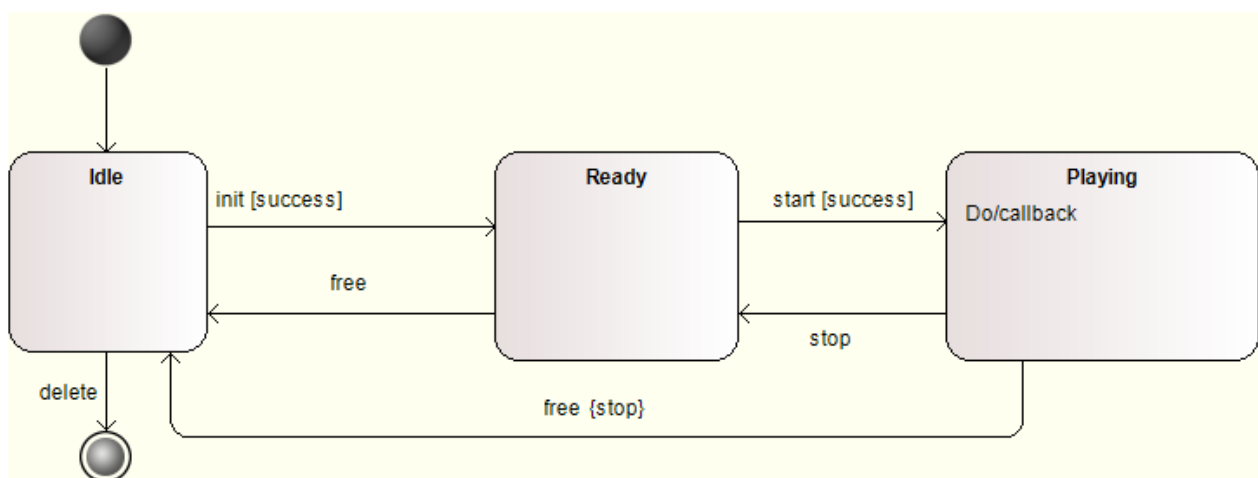
AudioStream	
+ buffer : short [length]	
+ end : short	
+ pwrite : short	
+ pread : short	
+ length : integer	
+ count : integer	
+ AudioStream(in buffer_length: integer)	C
+ ~AudioStream()	D
+ setBufferLength(in buffer_length: string): boolean	
+ read(in buffer: short, in len: integer): integer	
+ write(in buffer: short, in len: integer): integer	
+ writeSignal(in signal: Signal)	
+ writeStereoSignal(in left: Signal, in right: Signal)	

Elle représente un flux, le son continu qui va sortir des haut parleurs. Il prend en entrée un objet Signal. Il s'occupe de convertir les données du signal en échantillons utilisables par le driver audio. D'un point de vu purement technique : Il s'agit d'un buffer circulaire de type FIFO. L'atout qu'il a en plus c'est qu'il hérite d'une classe mutex il est donc verrouillable et déverrouillable (méthodes lock() et unlock()) pour éviter que 2 threads l'utilise en même temps.

## AudioDriver



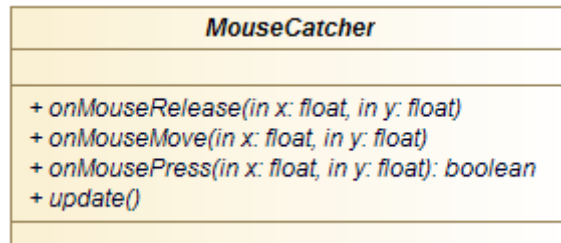
Elle représente le lien entre le Haut-Parleur et le signal converti par `AudioStream` lors de son démarrage (méthode `start`). On lui passe un `AudioStream` qu'il devra lire (via la méthode `read`) lorsque la carte son a besoin de nouveaux échantillons. Ci-dessous son diagramme uml stm behaviour (machine à états) :



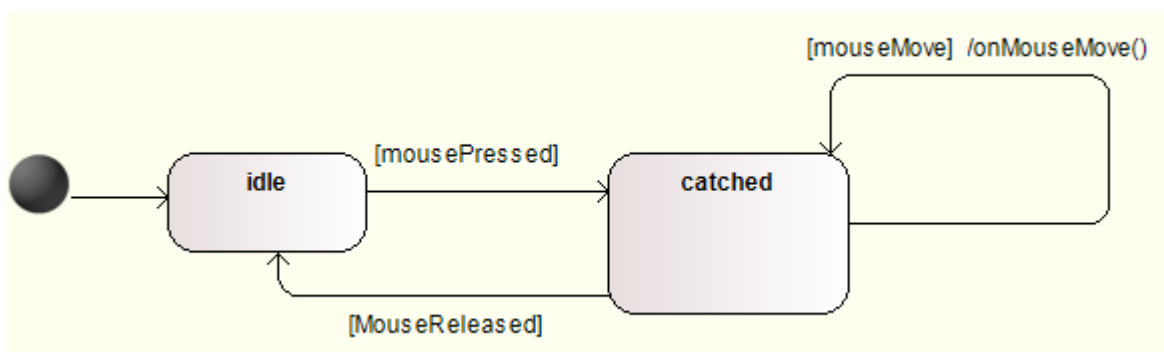
---

## MouseCatcher

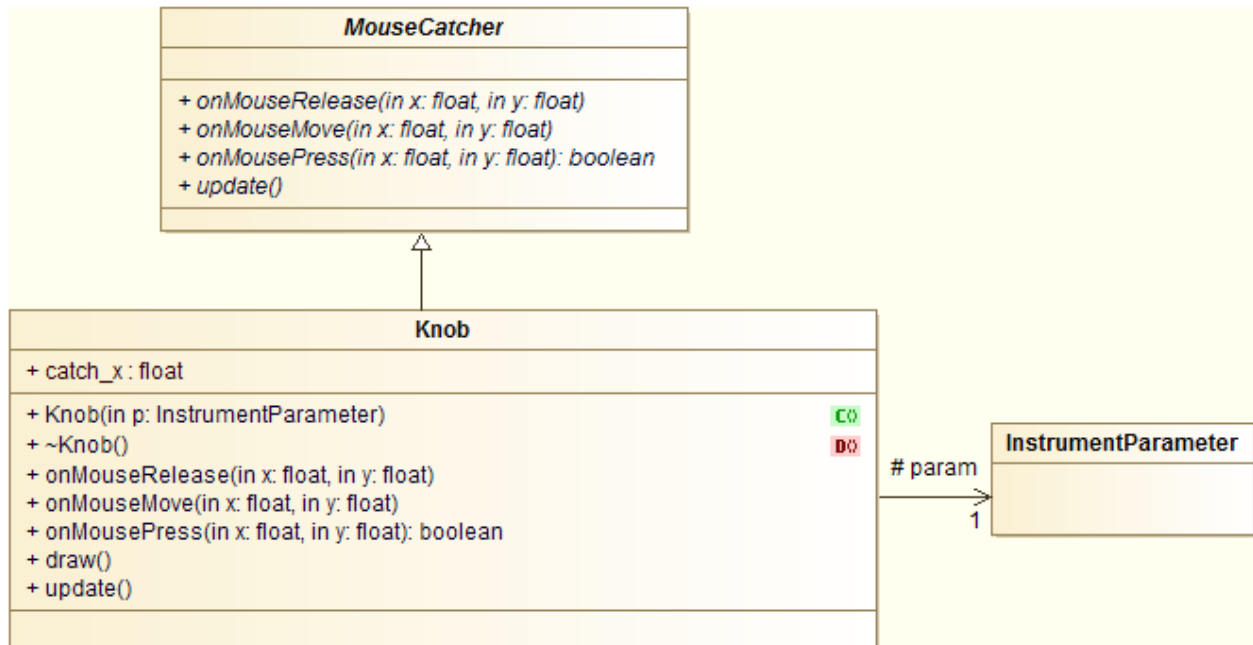
---



Elle représente un élément de l'interface graphique qui utilise la souris. Dans un premier temps l'utilisateur clique sur l'élément (exemple potard) (méthode onMousePress) ce qui a pour effet "d'attraper la souris". Une fois qu'un MouseCatcher a attrapé la souris, il faut attendre que l'utilisateur relâche le bouton de la souris pour en attraper un nouveau. C'est la fonction main qui s'en charge pour le moment. Ci-dessous son diagramme uml stm behaviour (machine à états) :



## Knob

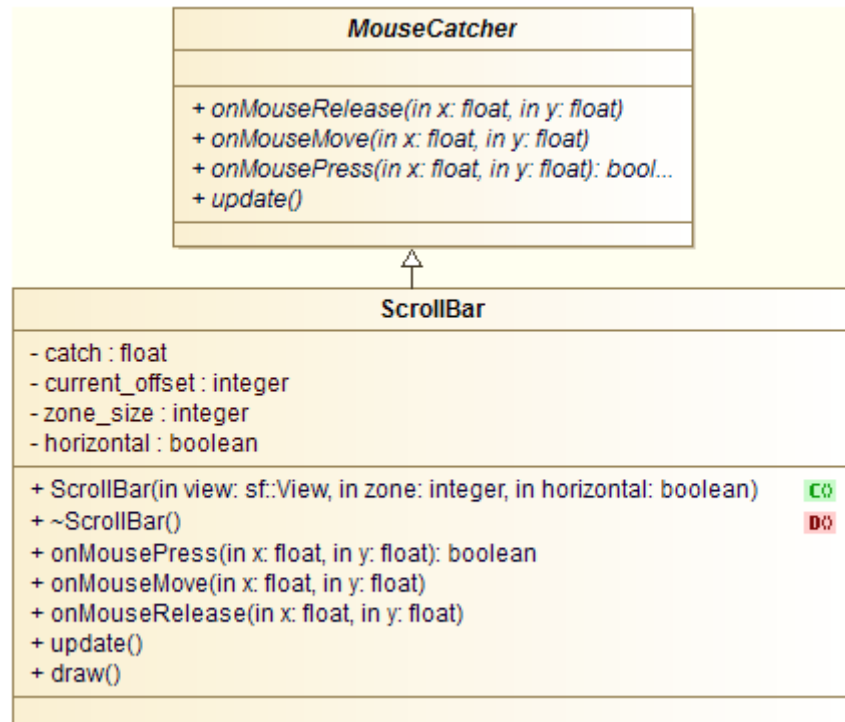


Knob veut dire potentiomètre en anglais. Cette classe est un **MouseCatcher**. Elle doit être liée à un **InstrumentParameter** pour agir directement dessus.

---

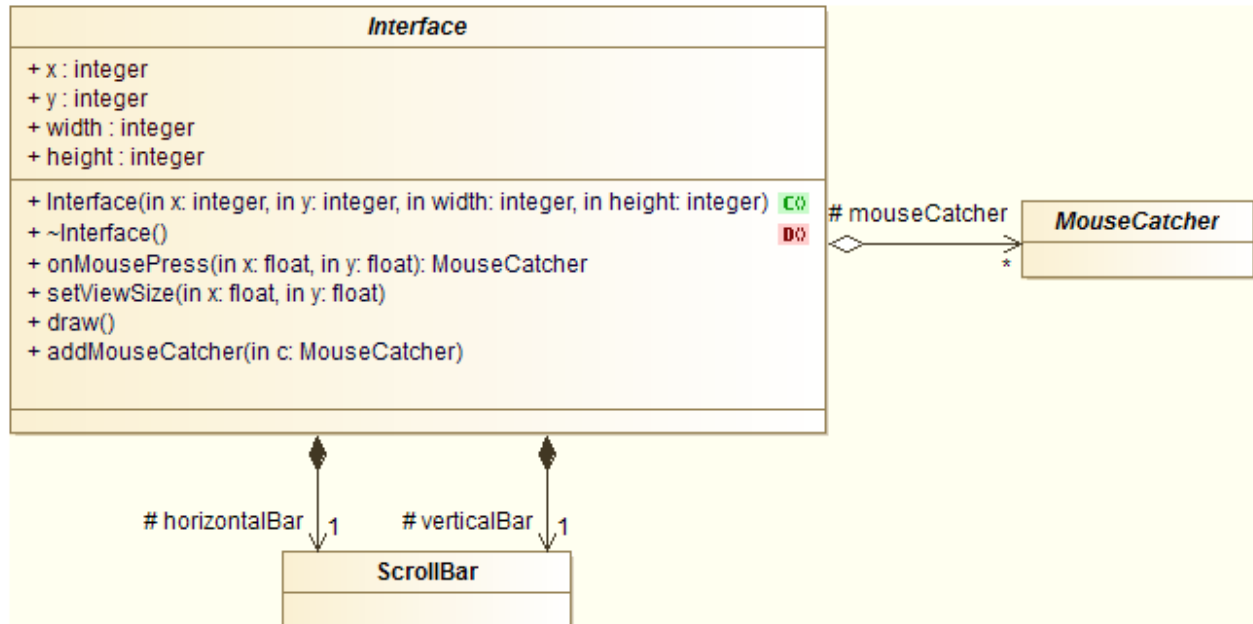
## ScrollBar

---



Scrollbar (barre de défilement) permet de gérer la vue d'une Interface si elle est trop petite pour afficher l'intégralité de l'Interface.

## Interface

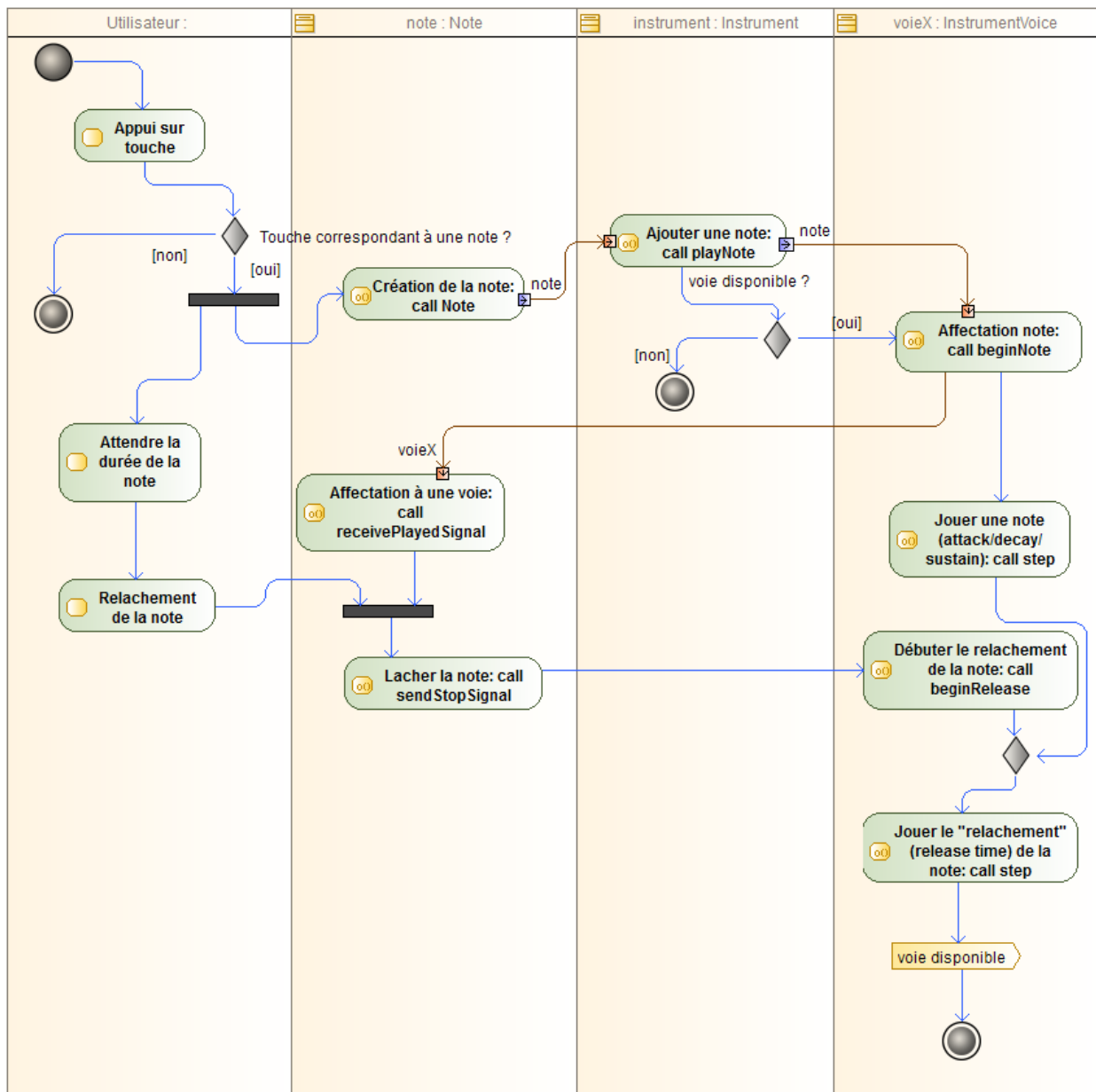


Interface gère des choses affichables dont des MouseCatcher. Elle s'occupe de les faire afficher dans l'espace que l'on lui donne !

## VI - Diagrammes d'activité

Ces diagrammes expliquent le déroulement des fonctionnalités du programme.

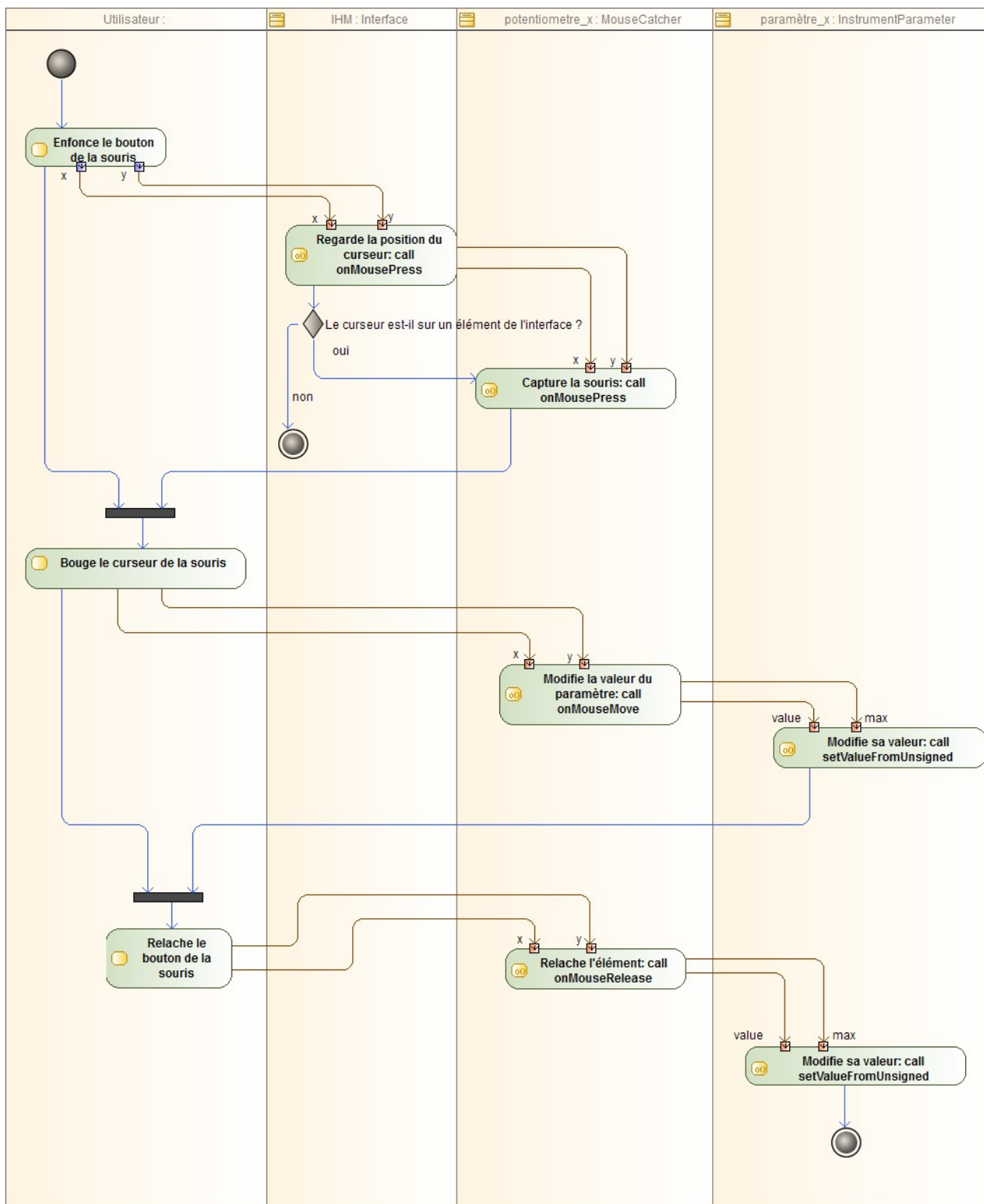
### 1) Jouer une note de musique



Ce diagramme illustre la création d'un objet note suite à l'appui de l'utilisateur sur une touche du clavier. Il détaille les différentes interactions entre les objets utilisant cette note.

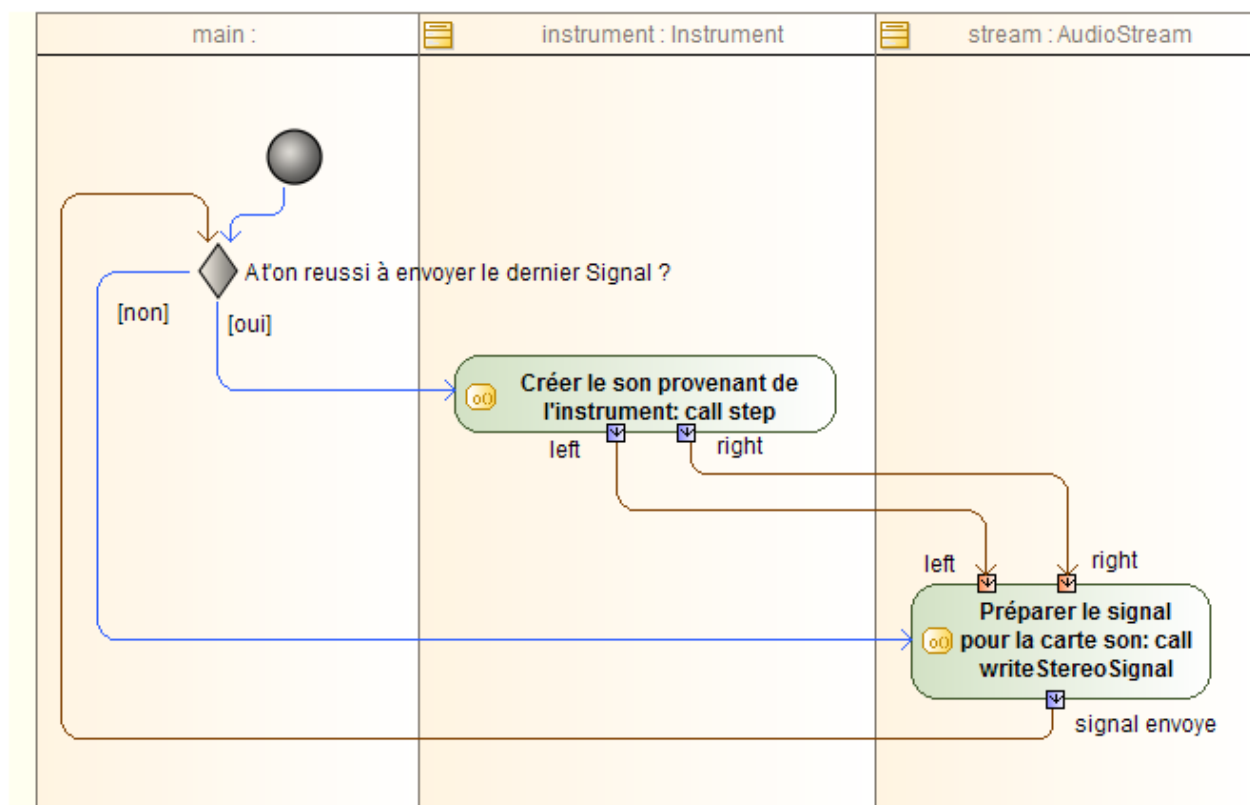


## 2) Modification d'un paramètre de l'instrument



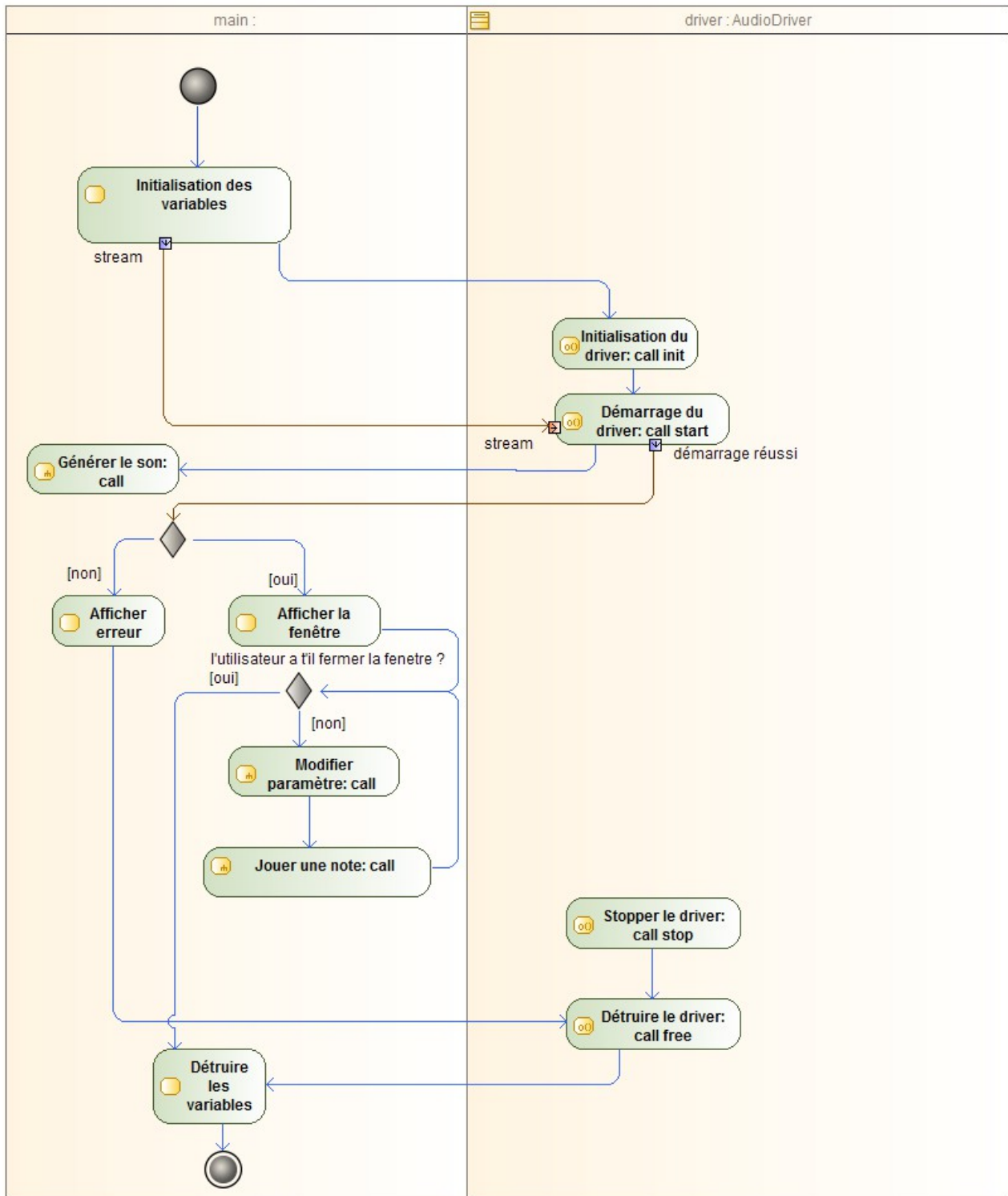
Ce diagramme illustre la modification d'un paramètre via l'IHM (Interface Homme Machine).

### 3) Fonctionnement du driver



Voici comment le son est envoyé aux haut-parleurs : On regarde en permanence si le buffer du driver est plein. Si il ne l'est pas, on génère le son via la méthode `step` puis on l'envoie aux haut-parleurs via la méthode `writeStereoSignal`. Le retour de cette dernière méthode permet de savoir si le signal a bien été envoyé et permet de savoir si l'on doit générer du son pour le prochain tour de boucle.

#### 4) Fonctionnement général du programme



C'est ici que l'on appelle les diagrammes d'activités que l'on a vu précédemment. On commence par initialiser les variables et lancer le driver via l'appel du diagramme d'activité du driver. On commence la boucle infini qui compose le programme. Dans cette boucle on affiche l'interface Homme-Machine et on appelle les diagrammes d'activités pour modifier les paramètres des instruments et pour jouer des notes.

## VII – Ressources et informations complémentaires

Le projet sera codé en C++ et est prévu pour fonctionner sous windows (windows 7 minimum) et sous linux (avec des restrictions et peut-être des problèmes de son selon les distributions). Le compilateur utilisé sera g++ (et l'équivalent sous windows : mingw) et le makefile est portable : make win32 pour windows et make linux64 pour linux. La carte graphique doit supporter OpenGL et avoir une carte son fonctionnelle avec le système installé.

Pour travailler sur ce projet, nous avons mis en place un git sur github. Voici le lien du projet : <https://github.com/lamogui/POO/> (notez que le projet inclus les sources de la SFML 2.1 modifiées pour fonctionner à l'ENIB car freetype est obsolète sur les merveilleuses distributions linux présente dans les salles de TP)

Un driver ASIO est fortement conseillé pour diminuer la latence entre l'appui de la note et la sortie du son, ce type de driver n'existe malheureusement que sur windows. Il existe un driver ASIO gratuit compatible avec n'importe quelle carte son il s'agit de ASIO4ALL disponible ici:

<http://asio4all.com/>

Listes des librairies qui seront utilisés

- SFML 2.1 Graphics
- SFML 2.1 Window
- SFML 2.1 System
- BASS
- BASSASIO

dépendances de ces librairies :

- WINAPI (windows uniquement)
- freetype
- X11 (linux uniquement)
- pthread (linux uniquement)
- Xrandr (linux uniquement)
- GL
- GLU
- GLEW
- libJPEG