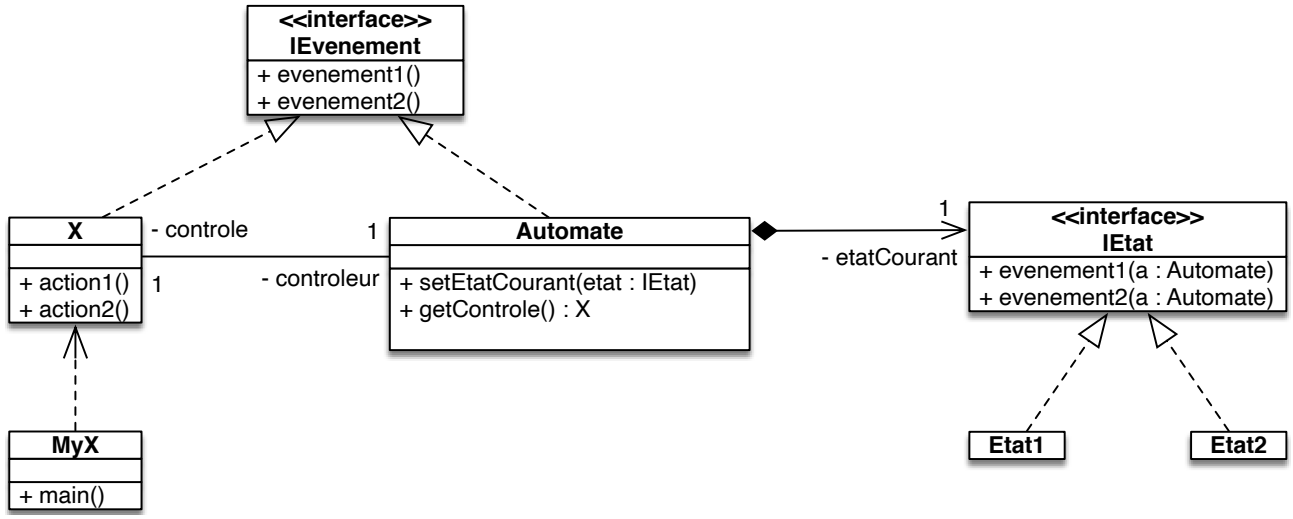
FEUILLE TP N°7 *Pattern state (suite)*

Objectifs

- Savoir interpréter une spécification UML (diagrammes de classes, d'états-transitions et de collaboration)
- Exploiter l'évolutivité offerte par le pattern state
- Combiner plusieurs design pattern

Exercice 1 *À vos marques...*

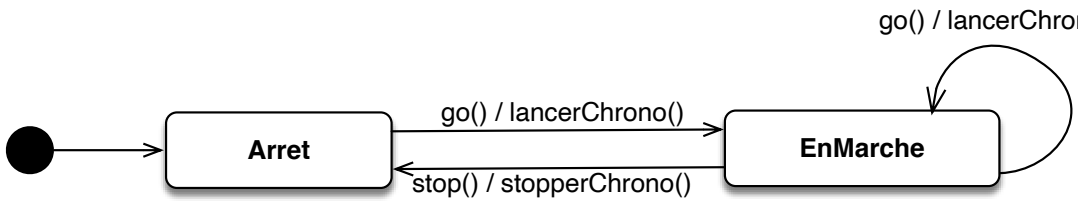
Dans le TP précédent nous avions réalisé la conception suivante :



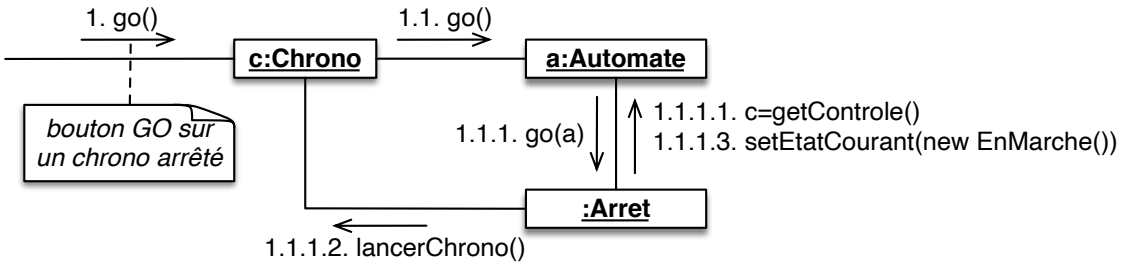
1.1 reprenez le projet BoUML du TP précédent (ou à défaut regénérez le diagramme de classes à partir du code Java)

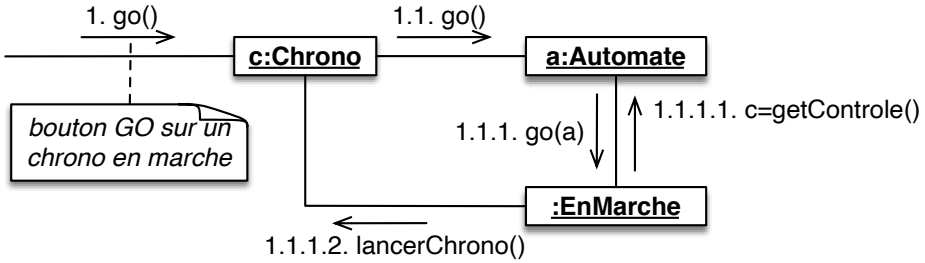
1.2 modifiez (renommez/complétez) votre conception de sorte à controler à présent un `:Chrono` à deux états (*Arret* et *EnMarche*) possédant deux boutons `GO` et `STOP` et deux opérations métier `lancerChrono()` et `stopperChrono()`.

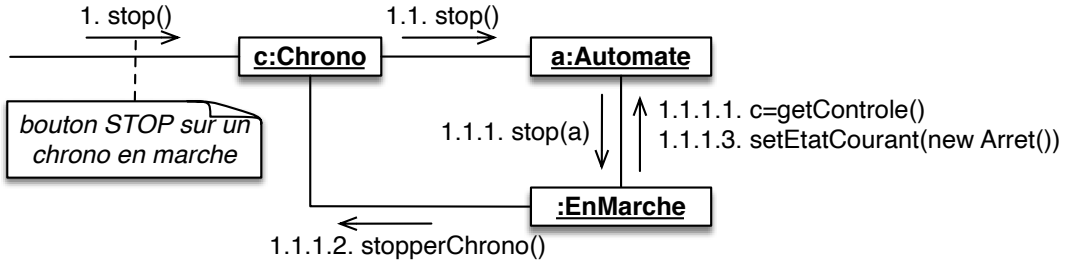
1.3 modifiez si besoin les règles de fonctionnement (initialisation, changements d'états, appels d'opérations métier) de sorte que votre `:Chrono` respecte l'ensemble des spécifications suivantes [1] :



1. Dans les autres situations il ne doit rien se passer.



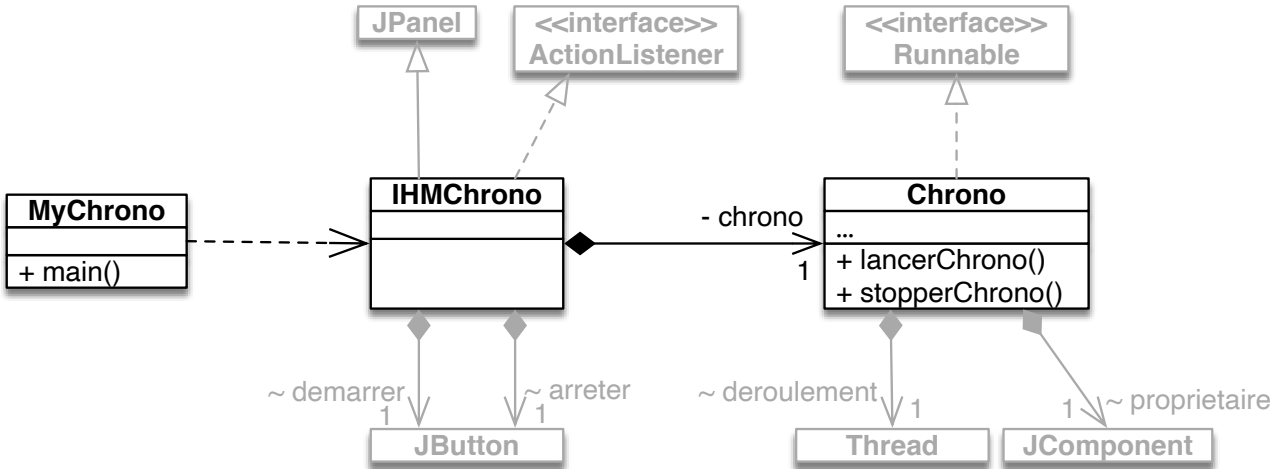




1.4 vérifiez le fonctionnement de votre `:Chrono` sur une séquence d'événements.

Exercice 2 *Prêts...*

2.1 Récupérez (dans Célène) l'archive *Chrono1.zip* contenant trois classes définissant un chronomètre graphique :



2.2 "Branchez" à présent cette classe `Chrono` sur votre automate :

- `Chrono` doit réaliser l'interface `IEvenement` (implémentez les opérations)
- `Chrono` et `Automate` doivent être mutuellement liés.

2.3 Testez le fonctionnement de votre chronomètre graphique

Exercice 3 *Partez ! !*

L'intérêt du pattern *State* est de faciliter l'évolution des règles de fonctionnement d'un objet par l'explicitation de l'automate sous-jacent.

Nous souhaitons donc améliorer l'ergonomie de notre `Chrono` en considérant cette fois deux boutons et cinq opérations métier :

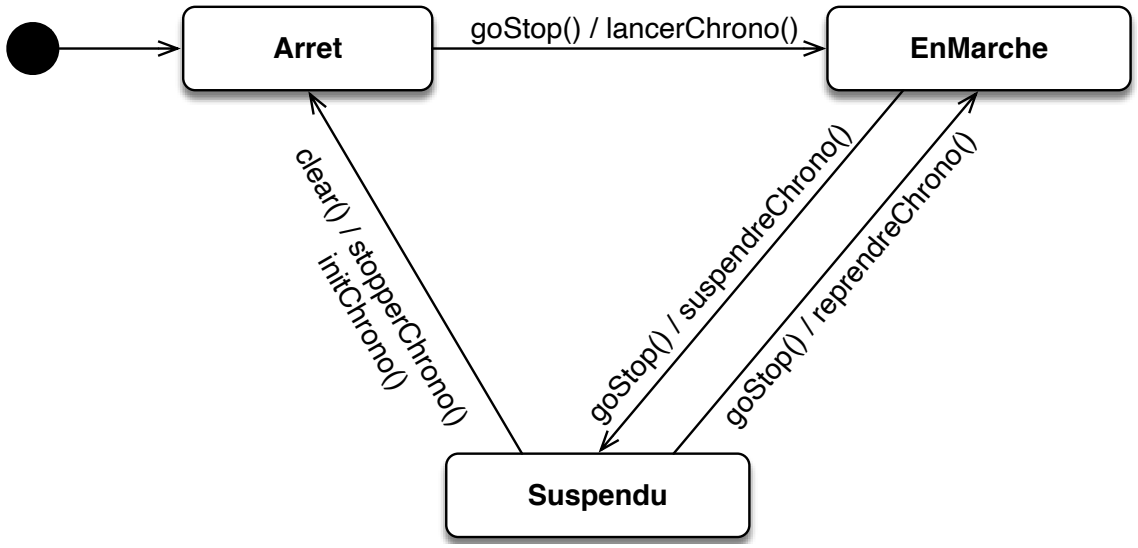
Boutons :

- `GO/STOP` : qui permet de démarrer/arrêter le chronomètre (avec un seul et même bouton)
- `CLEAR` : qui permet de remettre à zéro un chronomètre arrêté

Opérations métier :

- *lancerChrono()* : lance un processus (*thread*) chargé de redessiner le chronomètre en fonction du temps écoulé depuis le lancement,
- *stopperChrono()* : arrête le processus précédent,
- *suspendreChrono()* : met en pause le processus (sans l'arrêter),
- *reprendreChrono()* : permet de reprendre le processus,
- *initChrono()* : réinitialise le temps écoulé et redessine le chronomètre

3.1 Récupérez (dans Célène) l'archive *Chrono2.zip* contenant une version plus complète du chronomètre graphique et branchez-le sur votre automate dont vous améliorerez le fonctionnement en respectant la spécification suivante :



Exercice 4 *Sprint final...*

Utilisez le pattern singleton afin d'économiser des instances d'états...