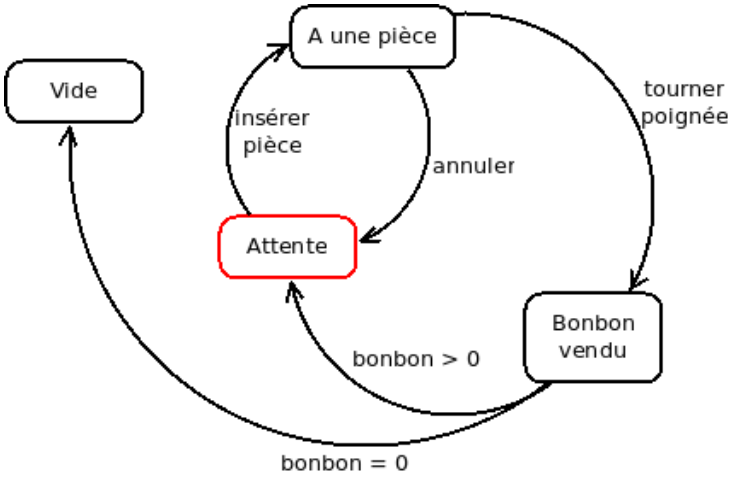
# State

## Exercice 1 (Le distributeur de bonbon)

Le but est ici de programmer un distributeur de bonbon à l'ancienne (avec une poignée à tourner pour servir le bonbon), avec des fonctionnalités basiques, puis d'y ajouter une nouvelle fonctionnalité.

Voici le diagramme d'état modélisant le comportement du distributeur.




Toutes les 5 secondes, le distributeur appelle une méthode `message` qui affiche un message à l'écran. Ce message est en fonction de l'état du distributeur.

1. Écrire `Etat` qui contiendra (au moins) des méthodes associées à chaque transition et à l'action `message`.
2. Écrire la classe de chaque état possible.
3. Écrire la classe `Distributeur` ayant comme attributs une variable de type Etat pour chaque état possible et le nombre de bonbon en stock, un constructeur initialisant chaque état et le nombre de bonbon, et les méthodes correspondant à l'action `message` et aux transitions que peut effectuer un utilisateur du distributeur. (attention, petit piège)
4. On souhaite modifier le comportement du distributeur ainsi : lorsque quelqu'un tourne la poignée, il aura une chance sur 10 d'avoir deux bonbons au lieu d'un, si les stocks le permettent. Modifier le diagramme d'état ainsi que votre programme en vous appuyant sur l'aide suivante :

```
1 // Bibliotheque a importer pour les diverses fonctionnalites
2 // concernant les generateurs pseudo-aleatoires.
3 import java.util.*;
4
5 // Initialiser un generateur pseudo-aleatoire
6 Random generateur = new Random( System.currentTimeMillis() );
7
8 // Tirer (pseudo) aleatoirement un nombre entre 0 et n-1 inclus.
9 int nombre = generateur.nextInt(n);
```

## Exercice 2 (La Google Car)

La Google Car est une voiture sans chauffeur, entièrement automatisée, développée conjointement par Google et l'Université de Stanford. Depuis octobre 2012, ce type de véhicule peut librement circuler dans plusieurs états des États-Unis.

On va simplifier le fonctionnement de la Google Car de la manière suivante. On considère que l'on part d'un état où le véhicule est à l'arrêt, le moteur ne tournant pas. En pressant un bouton, l'utilisateur peut démarrer le moteur passant dans un état d'attente de saisie de l'adresse de destination, et ré-éteindre le moteur par une pression de ce même bouton. Si le véhicule attend la saisie d'une adresse, une fois celle-ci tapée, il commence à rouler vers sa destination en entrant dans un mode Conduite. Arrivé à sa destination, le véhicule attend à nouveau la saisie d'une nouvelle adresse.

À tout moment, l'utilisateur peut demander à la Google Car de passer en conduite sportive, mais ceci ne sera pris en compte que si la voiture roule. Réciproquement, si la voiture est en conduite sportive, l'utilisateur peut lui demander de revenir en conduite normale, et cette tâche sera par contre prise en compte à tout moment. Faire le moins de méthodes possibles pour coder ce comportement.

1. Faire sur papier le diagramme d'état pour la Google Car.
2. Coder.

## Exercice 3 (Billets d'avion)

Vous programmez une application de réservation de billets d'avion. Nous allons considérer qu'un billet réservé peut être modifié ou annulé. Lorsque le billet est confirmé et payé, il n'est plus possible de l'annuler, mais la date reste modifiable. Après le départ du vol concerné, le billet ne peut être modifié ni annulé.

Cette description est volontairement ambigüe et/ou redondante.

1. Modélisez ce programme en UML à l'aide du pattern State.
2. Écrire le programme. Les actions consisteront à afficher à l'écran indiquant si l'opération est possible ou non.