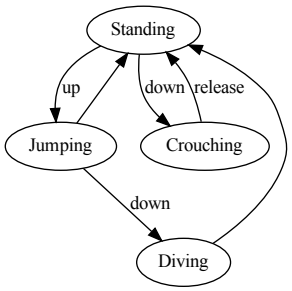
Génération du State pattern

FIL — M2GL

Le design pattern *State*[1] est une variation du pattern *Strategy*[2]. Comme avec *Strategy*, un objet *contexte* délègue une partie de son comportement et l'état associé à une famille d'objets *état* ; cependant, avec *State*, les états concrets sont responsables des transitions entre eux.

On souhaite faciliter le développement de classes Java à partir de la représentation d'une machine à états donnée dans un sous-ensemble de la syntaxe Graphviz[3]. Par exemple, le fichier `hero.dot` suivant décrit une interface `HeroState` et les quatre classes `Standing`, `Jumping`, `Diving`, et `Crouching` implémentant cette interface. Les transitions du graphe correspondent à trois messages dans `HeroState` représentant les actions du joueur : `up()`, `down()`, et `release()`. On utilisera un quatrième message `next()` pour les transitions qui se font automatiquement sans action explicite du joueur, par exemple quand le héros touche le sol suite à un saut.




```
// dot -Tpdf -o hero.pdf hero.dot
digraph HeroState {
  Standing          -> Jumping   [ label = "up"      ];
  Jumping           -> Diving    [ label = "down"    ];
  {Jumping, Diving} -> Standing;
  Standing          -> Crouching [ label = "down"    ];
  Crouching         -> Standing  [ label = "release" ];
}
```

Exercice

1. Implémenter le DSL et le générateur de code suggérés.
2. Construire un démonstrateur à partir d'un squelette généré.
3. Quelles sont les limitations d'une approche par génération de code ?
4. Proposer une approche alternative dans le langage de votre choix.

[1] `https://refactoring.guru/design-patterns/state`
[2] `https://refactoring.guru/design-patterns/strategy`
[3] `https://graphviz.org/doc/info/lang.html`