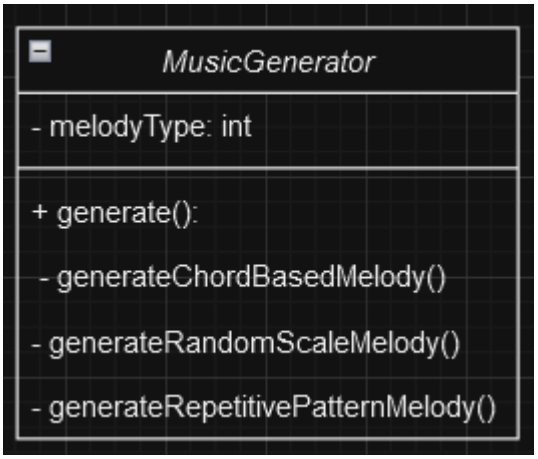
Résumé Pattern Strategy

Le pattern stratégie appartient aux patterns comportementaux (comme les patterns visiteur ou le pattern observateur).
Il s'agit d'un pattern visant à répartir les responsabilités et simplifier les ajouts de fonctionnalités d'un programme orienté objet par la création d'interfaces et de classes afin d'implémenter simplement des comportements.

Quand ?

Le pattern stratégie s'utilise quand une classe implémente un comportement multiple, par exemple :

On veut un programme générant de petites mélodies, on veut pouvoir les générer de trois façons : une mélodie sur une suite d'accords, une mélodie aléatoire sur une gamme et une mélodie répétitive basée sur un motif.
D'après l'énoncé on obtient le diagramme suivant :



La méthode generate aura donc grossièrement un code avec soit un CASE:

```
case 1 : generateChordBasedMelody();
         break;
case 2 : generateRandomScaleMelody();
         break:
case 3 : generateRepetitivePatternMelody();
         break;
```

Ou avec des IF ELSE :

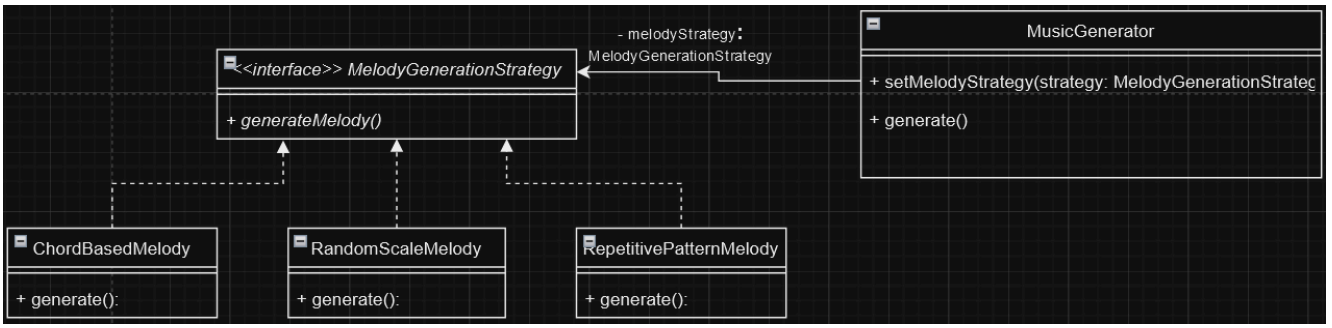
```
if (melodyType == 1) { generateChordBasedMelody(); }
else if (melodyType == 2) { generateRandomScaleMelody(); }
else if (melodyType == 3) { generateRepetitivePatternMelody(); }
```

Dans les deux cas, ce n'est pas très propre et cela ne respecte pas le OCP puisque l'ajout de fonctionnalités (comme l'ajout de nouvelles façon de créer de la musique) dans ce cas nécessite une modification de la classe elle-même.

Le design pattern stratégie :

Le design pattern stratégie consiste donc en l'extraction des méthodes sous la forme d'une interface et de la création de classes implémentant cette interface :

On crée donc l'interface MelodyGenerationStrategy { void generateMelody(); }
afin de distinguer le comportement des méthodes créatrices de musique, que l'on placera dans des classes implémentant cette interface.



Pour faire appel à la création de mélodie sur accord, une fera :
MelodyGenerationStrategy chordBased = new ChordBasedMelody();
On définit cette stratégie comme celle de notre Music Generator (avec le setteur) puis on appelera simplement la méthode generate().
Ainsi l'ajout d'un comportement se fera juste par la création d'une nouvelle classe héritant de MelodyGenerationStrategy et par l'Override de la méthode generate() selon le comportement attendu.

Cette solution respecte donc les principes SOLID:
SRP : les responsabilités sont biens séparées par classes.
OCP : la fonctionnalité est facile à développer et améliorer sans casser le fonctionnement
LSP : (pas d'héritage)
ISP : ?
DIP : les détails (comportements divers) dépendent d'abstractions (l'interface).