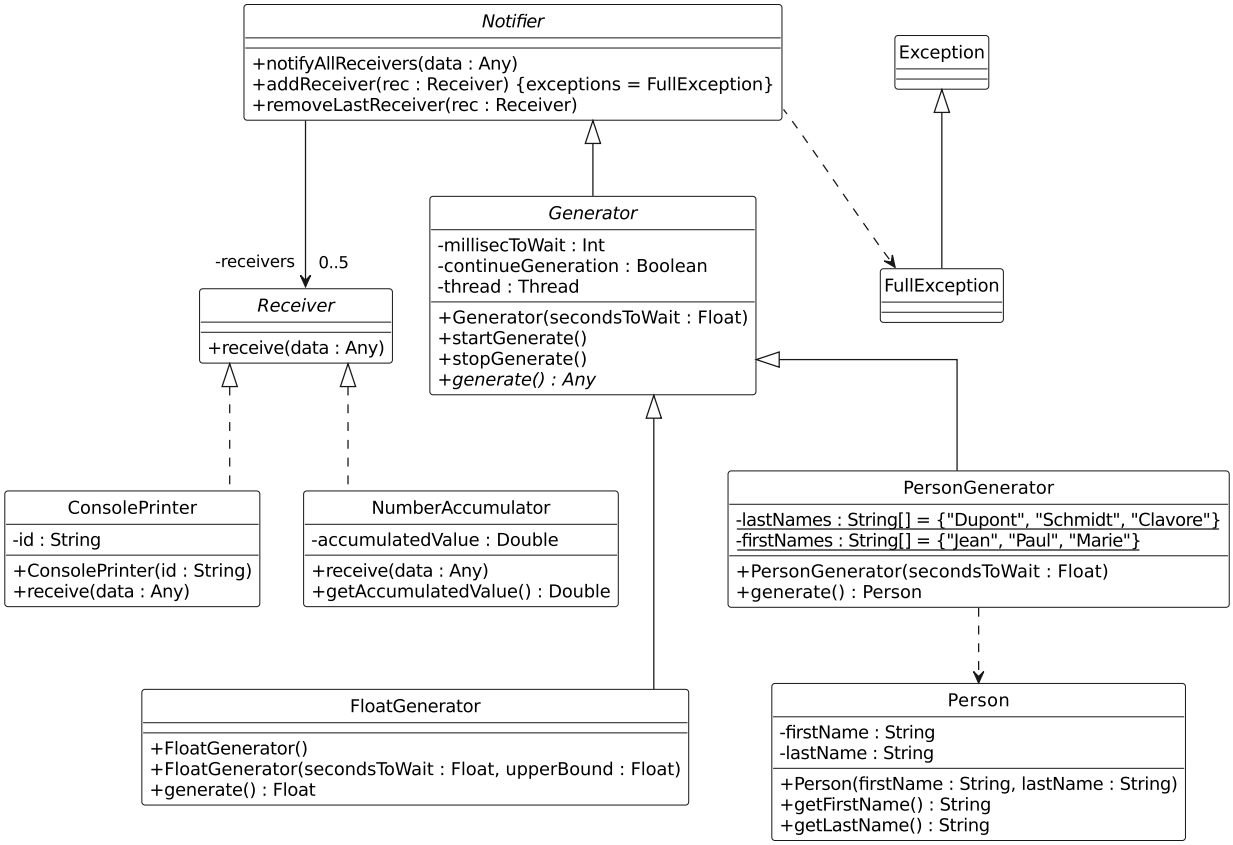
TP 1bis : Notifiers

1 Énoncé

Le but de cet exercice est d'implémenter un petit système de notifications simple, dans lequel un `Notifier` peut prévenir d'autres objets, les `Receiver`, qu'un événement intéressant s'est produit. Les seuls *notifiers* que nous créerons sont des générateurs d'objets qui notifient leurs *receivers* dès qu'un nouvel objet a été créé. Ces différents éléments sont architecturés comme décrit dans le diagramme suivant :



2 Exercice

Il vous est demandé de coder cette conception en Kotlin.

Attention, le but n'est pas d'aller le plus rapidement possible, mais de prendre le temps d'explorer les possibilités du langage pour essayer d'être concis et d'adopter le Kotlin spirit.

Les noms des méthodes sont explicites, vous devez être capable de déterminer quoi code. Voici juste quelques informations supplémentaires qui pourront vous aider :

- Les *receivers* seront stockés dans un simple tableau;

- La méthode `addReceiver(…)` lèvera une exception lorsque le tableau des *receivers* sera plein;
- La classe `ConsolePrinter`, dès qu'elle sera notifiée de la réception d'une donnée, affichera cette donnée sur la sortie standard, préfixée par «id:» où id est évidemment l'`id` du `ConsolePrinter`;
- La classe `NumberAccumulator`, dès qu'elle sera notifiée de la réception d'une donnée, ajoutera cette donnée à son `accumulatedValue` si cette dernière est un nombre (peu importe son type réel);
- La classe `Generator` doit permet de générer un objet (grâce à `generate(…)`) toutes les `secondsToWait` secondes. Chaque générateur possèdera son thread pour effectuer cette génération.
- La classe `FloatGenerator` générera un flottant compris entre 0 et `upperBound`. Sans informations explicites, cette génération se fera toutes les secondes et la borne supérieure sera 1000.
- La classe `PersonGenerator` génèrera des personnes dont la combinaison «Prénom Nom» est choisie aléatoirement parmi les tableaux `firstNames` et `lastNames`.

Le programme suivant devra compiler :

```
fun main() {
    val personGen = PersonGenerator(0.8f)
    val numGen = FloatGenerator(0.2f, 91827364.5f)
    val acc = NumberAccumulator()

    personGen.addReceiver(::println)
    personGen.addReceiver(acc)
    numGen.addReceiver(::println)
    numGen.addReceiver(acc)

    personGen.startGenerate()
    numGen.startGenerate()
    Thread.sleep(5000)
    personGen.stopGenerate()
    numGen.stopGenerate()

    println(acc.accumulatedValue)
}
```