TD de programmation orientée objet en Java
Les objets

On étudie dans cette série d'exercices la classe `BigDecimal` dont le rôle est de représenter des nombres réels en précision arbitraire. Son fonctionnement est similaire à celui des `BigInteger` à une important différence près : les calculs pour les `BigInteger` sont toujours exacts alors que ceux des `BigDecimal` sont approchés. De ce fait, on utilise souvent des objets de la classe `MathContext` qui permettent d'indiquer la qualité souhaitée pour l'approximation. Plus précisément, nous utiliserons dans les exercices suivants le constructeur `MathContext(k)` qui crée un objet `MathContext` associé à une représentation des nombres réels avec `k` chiffres.

D'autre part, nous utiliserons les méthodes et constructeurs suivants pour les `BigDecimal` :

Constructeurs :
- `BigDecimal(s)` : création d'un réel à partir de la chaîne `s`
- `BigDecimal(d)` : création d'un réel à partir d'un double `d`
- `BigDecimal(bi)` : création d'un réel à partir d'un `BigInteger bi`

Méthodes :
- méthodes de la forme `op(val)` : ces méthodes réalisent un calcul exact impliquant l'objet appelant et le paramètre `val` (de type `BigDecimal`). Attention, si le calcul exact n'est pas possible, une erreur se produit. Les opérations disponibles sont `add`, `subtract`, `multiply` et `divide`. Seule cette dernière opération peut poser problème.
- méthodes de la forme `op(val, mc)` : il s'agit des mêmes méthodes que précédemment, mais avec un paramètre `mc` de type `MathContext` qui précise comment le résultat est arrondi. Ces méthodes fonctionnent toujours, mais peuvent donner un résultat approximatif.
- la méthode `compareTo(val)` permet de comparer deux `BigDecimal`. Elle renvoie -1 si l'objet appelant est strictement plus petit que `val` , 1 si `val` est strictement plus petit que l'objet appelant et 0 s'ils sont égaux.

Constantes : comme pour les `BigInteger`, on dispose des constantes `ZERO`, `ONE` et `TEN`.

Exercice 1 : Inversion d'un nombre

Écrire un programme qui demande à l'utilisateur un `BigDecimal` (sous la forme d'une chaîne de caractères) et une précision, puis affiche l'inverse du nombre avec la précision souhaitée. Si cette précision est nulle, on affichera l'inverse exact du nombre (ce qui provoquera une erreur quand cette valeur n'est pas calculable).

Exercice 2 : Calcul de π

On souhaite calculer une valeur précise de π en utilisant les `BigDecimal`. Plusieurs séries entières convergent vers π (à un facteur multiplicatif près), ce qui permet de calculer une valeur approchée. Par exemple on sait que

$$\arctan z = \sum_{n=0}^{\infty} \frac{(-1)^n z^{2n+1}}{2n+1},$$

ce qui conduit à approcher π par

$$\pi = 4 \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}.$$

Cette formule, dite de Gregory–Leibniz, converge cependant lentement. On lui préfère d'autres formules, comme par exemple celle de John Machin basée sur l'égalité suivante

$$\frac{\pi}{4} = 4 \arctan \frac{1}{5} - \arctan \frac{1}{239}.$$

Écrire un programme qui demande à l'utilisateur une précision de calcul (pour le `MathContext`) et un nombre d'itérations, puis affiche les approximations de π obtenues avec les deux formules proposées après le nombre d'itérations choisi.

Écrire un programme qui demande à l'utilisateur une précision de calcul puis utilise la formule de Machin pour calculer π avec la précision demandée en déterminant automatiquement le nombre d'itérations à réaliser.