

SAE R201  
Ma Calculatrice

**La SAE est à faire en binomes. Vous devez rendre vos fichiers (diagramme UML, sources Java, exécutable et un fichier help si besoin) à vos chargés de TDs selon leurs directives.**

**A ce document est joint un rappel rapide des notions UML vues en cours**

**<https://lipn.fr/~azzag/R201/SAE/>**

**PARTIE I : Date limite du rendu 9 MAI 2022**

**PARTIE II : Date limite du rendu 23 MAI 2022**

**Important : les responsables de SAE pour chaque groupe :**

- Groupe Draco : Dominique Bouthinon ([bouthinon@univ-paris13.fr](mailto:bouthinon@univ-paris13.fr))
- Groupe Cygnus : Francesco Pisanu ([pisanu@lipn.univ-paris13.fr](mailto:pisanu@lipn.univ-paris13.fr))
- Groupe Andromeda : Bouchaib Lemaire ([bouchaib.lemaire@univ-paris13.fr](mailto:bouchaib.lemaire@univ-paris13.fr))
- Groupe Pegasus : Hanane Azzag ([azzag@univ-paris13.fr](mailto:azzag@univ-paris13.fr))
- Groupe Pheonix : Pierre Gerard ([pierre.gerard@univ-paris13.fr](mailto:pierre.gerard@univ-paris13.fr))

## Partie I : opérations simples

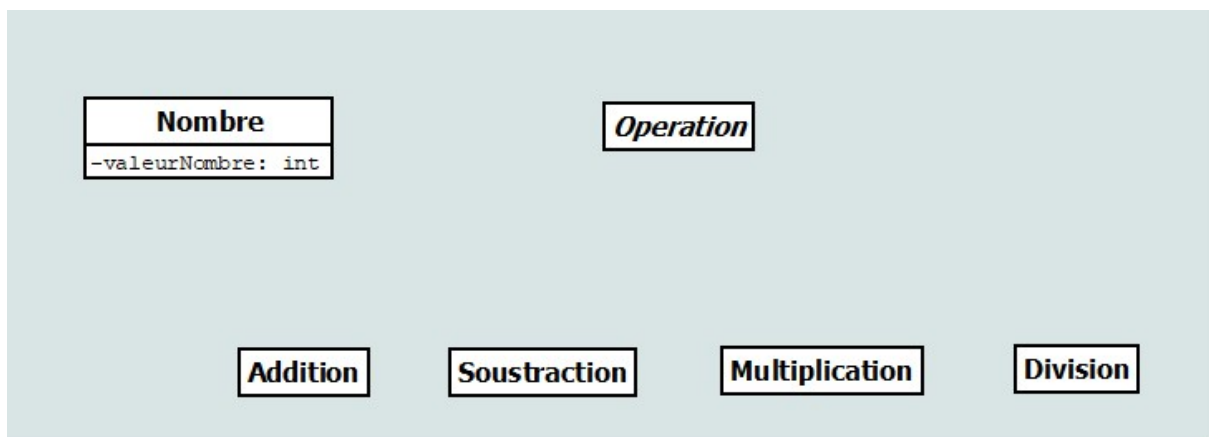
L'objectif du projet est de modéliser en UML et coder en java une calculatrice effectuant des opérations sur des nombres entiers.

On dispose des opérations suivantes : l'addition (+), la soustraction (-), la multiplication (\*) et la division entière (/) de deux nombres entiers, par exemple 5+3.

**Noter** qu'une opération binaire simple est composée de 3 éléments : 1 opérateur et deux opérandes de type entier

### I. UML

On considère les classes suivantes :



La classe **Operation** écrite en italiques est abstraite. La classe **Nombre** sera utilisée pour représenter une valeur entière (elle contient un attribut **valeurNombre** de type *int*) et possédera des méthodes (voir ci-dessous).

Concevoir le digramme UML en répondant aux questions suivantes :

1. Ajouter les liens (associations et héritages) entre ces classes de façon à modéliser toutes les opérations simples possibles.
2. Compléter le diagramme précédent en fonction des indications suivantes :
  - a. Munir les classes d'une méthode *valeur()* retournant un *int* représentant la valeur du **Nombre** ou le résultat de l'opération. Noter que cette méthode est abstraite dans la classe **Operation**.
  - b. Munir les classes (sauf la classe **Operation**) d'une méthode *toString()* retournant un **String** représentant le **Nombre** ou l'opération.
  - c. Munir la classe **Operation** de deux méthodes *getOperande1()* : Nombre et *getOperande2()* : Nombre.

## II. JAVA

Implémenter (en java) et tester les classes précédentes (commencer par la classe **Nombre**).

Le classe **Division** doit obligatoirement prévenir une division par zéro au moyen d'une exception. Il y a deux options possibles :

- utiliser une **Exception** dans le constructeur, empêchant de créer une **Division** avec un dénominateur nul.
- Déléguer une **ArithmeticException** automatiquement créée par une division par zéro dans la méthode *valeur*.

Dans tous les cas les exceptions doivent être surveillées (*try*) et traitées (*catch*) dans la méthode *main* de la classe **CalculatriceSimple** décrite ci-dessous.

Créer une classe **CalculatriceSimple** dont la méthode *main* teste toutes les opérations. Le test d'une opération pourra s'effectuer de la façon suivante :

```
Nombre six = new Nombre(6) ;
```

```
Nombre dix = new Nombre(10) ;
```

```
Operation s = new Soustraction(dix,six) ;
```

```
System.out.println(s + " = " + s.valeur()) ; // doit afficher : (10 – 6) = 4
```