

L'objectif de ce TP est de concevoir un programme permettant de modéliser les dipôles électriques, puis de calculer leur *impédance*.

Modalités de rendu

L'ensemble du code source que vous aurez réalisé est à rendre sur la plateforme GitLab de l'école dans un dépôt `git` qui vous a été créé spécifiquement pour ce travail

https://gitlab.telecomnancy.univ-lorraine.fr/oop2k21/lab3-<votre_id_ul>

Il vous est demandé de ne rendre que les fichiers `.java`, ceux-ci étant rangés dans un répertoire `src/` (et éventuellement dans des sous répertoires correspondant aux packages que vous aurez définis).

À chaque question devra correspondre un ou plusieurs commits.

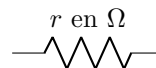
Un premier rendu devra avoir lieu à la fin de la séance de TD/TP et pourra (devrait) être complété ultérieurement si nécessaire. Le travail final est à rendre au plus tard le **mardi 9 février 2021 à 14:00** sur la plateforme GitLab de l'école.

1 Le problème

Les dipôles sont composés de composants "élémentaires" : *résistance*, *self* et *capacité*. Ces composants élémentaires peuvent être combinés par un montage *en série* ou *en parallèle*.

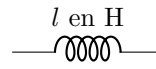
Soit ω un nombre réel appelé *pulsation* et i le nombre complexe de module 1 et d'argument $\pi/2$. L'*impédance* z d'un dipôle est un nombre complexe.

Une *résistance* de valeur r exprimée en ohms (symbole Ω)



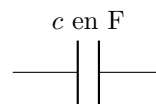
$$z = r$$

Une *self* de valeur l exprimée en henrys (symbole H)



$$z = i(\omega * l)$$

Une *capacité* de valeur c exprimée en farad (symbole F)



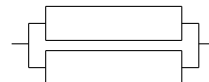
$$z = i\left(\frac{-1}{\omega * c}\right)$$

Un dipôle composé de deux dipôles d'impédance z_1 et z_2 montés *en série*



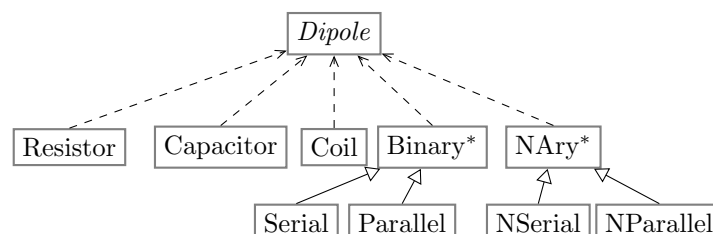
$$z = z_1 + z_2$$

Un dipôle composé de deux dipôles d'impédance z_1 et z_2 montés *en parallèle*



$$z = \frac{1}{\frac{1}{z_1} + \frac{1}{z_2}}$$

Nous allons modéliser les dipôles électriques en utilisant la hiérarchie de classes suivante :



Chaque classe définira la méthode `Complex impedance(double omega)` permettant d'évaluer l'impédance du dipôle manipulé (le paramètre `omega` représentant la pulsation sous la forme d'un nombre réel).

L'impédance d'un dipôle est en général un nombre complexe. On peut remarquer que l'impédance d'une résistance est un nombre réel et que l'impédance d'une self ou d'une capacité sont des nombres complexes imaginaires purs.

2 Sujet du TP

Éléments fournis. Vous trouverez dans le répertoire `src/` la classe `dipole.Complex` permettant de modéliser un nombre complexe. Cette classe permet de faire l'addition de deux nombres complexes et de calculer l'inverse d'un nombre complexe.

Vous trouverez également dans le même répertoire les squelettes des différentes classes et interfaces (`Dipole`, `Coil`, `Binary`, `Nary`, etc) réalisant une partie de la hiérarchie de classes présentée précédemment que vous devrez compléter au fur et à mesure de ce devoir.

Réflexion préliminaire à mener. Il est important qu'avant de commencer à programmer quoique ce soit, vous parcouriez le code source des classes fournies pour avoir un aperçu des méthodes que vous pouvez utiliser et celles que vous devrez implémenter.

Compilation. Pour compiler vos classes vous utiliserez donc les commandes suivantes. On supposera que que votre répertoire courant est le répertoire de votre dépôt git local. Les fichiers compilés seront placés dans un dossier `build/`.

```
1 # pour compiler vos classes
2 javac -d build/ -cp lib/junit-platform-console-standalone-1.7.0.jar:build/ src/dipole/*.java
```

Test unitaires. Un ensemble de tests unitaires vous sont fournis afin de vérifier la correction de votre code. Ces tests sont présents dans le dossier `test/`. Ils sont écrits en utilisant la librairie `JUnit5`. Cette librairie vous est fournie sous la forme d'une archive `.jar` présente dans le répertoire `lib/`.

Pour exécuter une classe de tests, vous devrez d'abord la compiler, puis l'exécuter.

```
1 # pour compiler une classe de test (dipole.TestCoil)
2 javac -d build/ -cp lib/junit-platform-console-standalone-1.7.0.jar:build/:test/ test/dipole/CoilTest.java
3
4 # pour exécuter cette classe de tests
5 java -jar lib/junit-platform-console-standalone-1.7.0.jar -cp build --select-class=dipole.CoilTest
6
7 # pour exécuter toutes les classes de tests du package dipole
8 java -jar lib/junit-platform-console-standalone-1.7.0.jar -cp build --select-package=dipole
```

Vous pouvez obtenir plus d'informations sur comment compiler et exécuter ces tests en consultant le fichier `README.md` fourni.

★ **Exercice 1.** Nous souhaitons tout d'abord manipuler des dipôles élémentaires.

- ▷ **Question 1.** Remplissez la classe `dipole.Coil` permettant de calculer son impédance.
- ▷ **Question 2.** Vérifiez le bon fonctionnement de votre code grâce au test fourni qui calcule l'impédance du dipôle figure 1 pour $\omega = 314.16$, $\omega = 3.1416$ et $\omega = 0$. Dans le langage Java, la valeur 5×10^{-2} peut s'écrire `5e-2`.
- ▷ **Question 3.** De la même façon, complétez la classe `dipole.Capacitor` implémentant le dipôle élémentaire capacité ainsi que sa méthode pour calculer son impédance. Vérifiez son bon fonctionnement.
- ▷ **Question 4.** Enfin, écrivez entièrement la classe `dipole.Resistor` implantant le dipôle élémentaire résistance ainsi que sa méthode pour calculer son impédance.

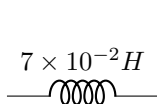


Figure 1: La self testée.

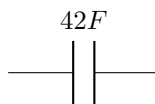


Figure 2: La capacité testée.



Figure 3: La résistance testée.

★ **Exercice 2.** Nous souhaitons maintenant construire des dipôles combinant plusieurs dipôles (élémentaires ou résultant déjà d'une composition).

- ▷ **Question 5.** Complétez la classe `dipole.Serial` permettant de définir un dipôle composé de deux dipôles en série et de calculer son impédance. Votre classe devra hériter de la classe abstraite `dipole.Binary` fournie.
- ▷ **Question 6.** Vérifiez le bon fonctionnement de votre code grâce au test fourni qui calcule l'impédance du dipôle de la figure 4 pour $\omega = 314.16$ et d'autres valeurs d'omega.
- ▷ **Question 7.** De la même façon, écrivez la classe `dipole.Parallel` implantant un dipôle combinant deux dipôles montés en parallèle et permettant de calculer son impédance. Votre classe devra hériter de la classe `dipole.Binary`.
- ▷ **Question 8.** Vérifiez le bon fonctionnement de votre code grâce au test fourni qui calcule l'impédance du dipôle de la figure 5 pour $\omega = 314.16$.

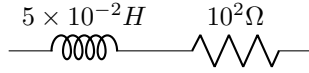


Figure 4: Montage en série testé.

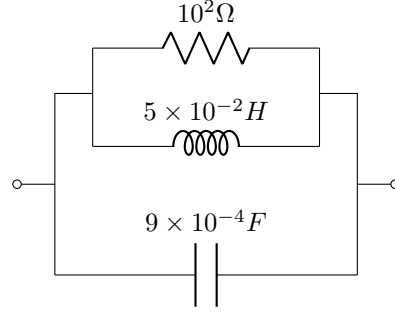


Figure 5: Montage en parallèle testé.

★ **Exercice 3.** On voudrait maintenant pouvoir modéliser la mise en série ou en parallèle d'un nombre quelconque de dipôles. Pour cela, vous allez définir des sous-classes de la classe abstraite `dipole.NAry` qui vous est fournie.

▷ **Question 9.** Écrivez la classe `dipole.NSerial` pour définir un dipôle composé d'un nombre quelconque de dipôles en série et le calcul de son impédance. L'impédance de n dipôles d'impédance z_i montés en série peut se calculer en utilisant la formule suivante : $\sum_{i=1}^n z_i$.

▷ **Question 10.** Écrivez la classe `dipole.NParallel` pour définir un dipôle composé d'un nombre quelconque de dipôles en parallèle et le calcul de son impédance. L'impédance de n dipôles d'impédance z_i montés en parallèle peut se calculer en utilisant la formule suivante : $\frac{1}{\sum_{i=1}^n \frac{1}{z_i}}$.

▷ **Question 11.** Testez votre code grâce au test fourni permettant de définir le dipôle de la figure 6 en utilisant ces nouveaux types (sans utiliser les types `dipole.Serial` ou `dipole.Parallel` précédemment définis) et de calculer son impédance pour $\omega = 314.16$.

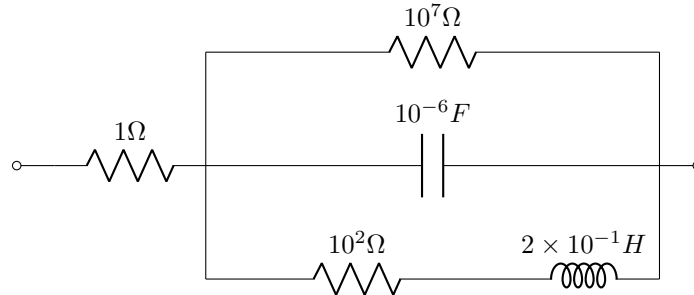
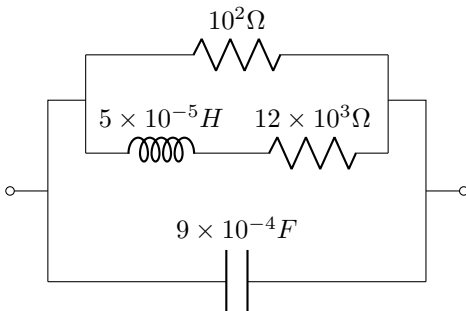


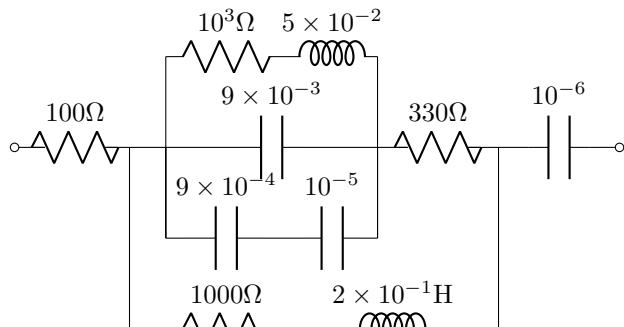
Figure 6: Montage testé.

★ **Exercice 4.** L'objectif est maintenant de vous faire *instancier* des dipôles plus complexes.

▷ **Question 12.** Complétez le code de la classe `dipole.Instances` pour réaliser les dipôles des figures 7a et 7b. Testez votre code avec le test fourni.



(a) Le dipole `dip1` à réaliser.



(b) Le dipole `dip2` à réaliser.