TP4 : Héritage, redéfinition, liaison dynamique.

POO : Programmation Orientée Objet

Première année

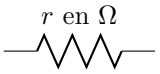
L'objectif de ce TP est de concevoir un programme permettant de modéliser les dipôles électriques, puis de calculer leur *impédance*.

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*.

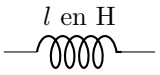
Soit ω un nombre réel appelé *pulsation* et i le nombre complexe de module 1 et d'argument $\pi/2$. *L'impédance* z d'un dipôle est un nombre complexe.

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



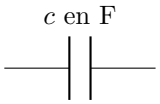
$z = r$

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



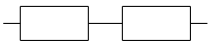
$z = i(\omega * l)$

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



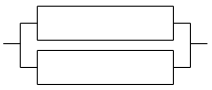
$z = i(\frac{-1}{\omega * c})$

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



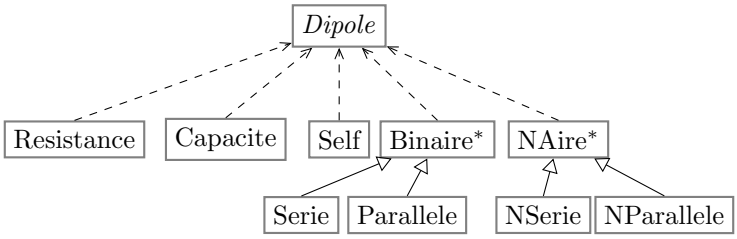
$z = z_1 + z_2$

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 `Complexe 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

Élements fournis. Vous trouverez dans `/home/depot/1A/POO/TP4` la classe `Complexe` 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 aussi dans le même répertoire les squelettes des différentes classes et interfaces (`Dipole`, `Resistance`, `Binaire`, `NAire`, etc) réalisant la hiérarchie de classes présentée précédemment. Vous trouverez enfin la classe `TestsAutomatiques` qui contient des tests

jUnit pour le code que vous devez écrire. Pour l'exécuter depuis éclipse, ouvrez ce fichier, puis exécutez "Run as.. junit test". Pour l'exécuter depuis la ligne de commande, exécutez la ligne suivante :

```
java -cp junit-4.5.jar:. org.junit.runner.JUnitCore dipole.TestsAutomatiques
```

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 implanter.

Notation. Dans le langage Java, la valeur 10e-2 peut s'écrire `1e-2`

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

▷ **Question 1:** Remplissez la classe `Self` permettant de calculer son impédance.

▷ **Question 2:** Vérifiez le bon fonctionnement de votre code grâce au test TestSelf, qui calcule l'impédance du dipôle figure 1 pour $\omega = 314.16$. Le résultat escompté est environ $0.0 + 21.99i$.

▷ **Question 3:** De la même façon, écrivez la classe `Capacite` implantant le dipôle élémentaire capacité ainsi que sa méthode pour calculer son impédance. Cette classe est testée par `testCapacite()`.

▷ **Question 4:** Enfin, écrivez la classe `Resistance` implantant le dipôle élémentaire résistance ainsi que sa méthode pour calculer son impédance. Cette classe est testée par `testResistance()`.

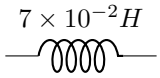


FIGURE 1 – La self testée.

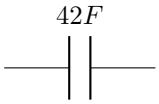


FIGURE 2 – La capacité testée.

FIGURE 3 – La résistance testée.

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

▷ **Question 1:** Écrivez la classe `Serie` permettant de définir un dipôle composé de deux dipôles en série et son impédance. Votre classe `Serie` devra hériter de la classe `Binaire`.

▷ **Question 2:** Vérifiez le bon fonctionnement de votre code grâce au test `testSerie`, qui calcule l'impédance du dipôle de la figure 4 pour $\omega = 314.16$ *(résultat escompté ≈ 100.0 + 15.70i).*

▷ **Question 3:** De la même façon, écrivez la classe `Parallele` implantant un dipôle combinant deux dipôles montés en parallèle et permettant de calculer son impédance. Votre classe `Parallele` devra hériter de la classe `Binaire`.

▷ **Question 4:** Vérifiez le bon fonctionnement de votre code grâce au test `testParallele`, qui calcule l'impédance du dipôle de la figure 5 pour $\omega = 314.16$ *(résultat escompté ≈ 0.2079 + -4.55i).*

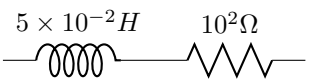


FIGURE 4 – Montage en série testé.

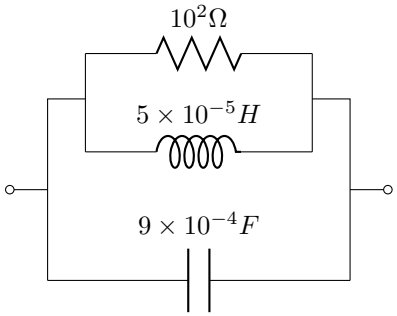


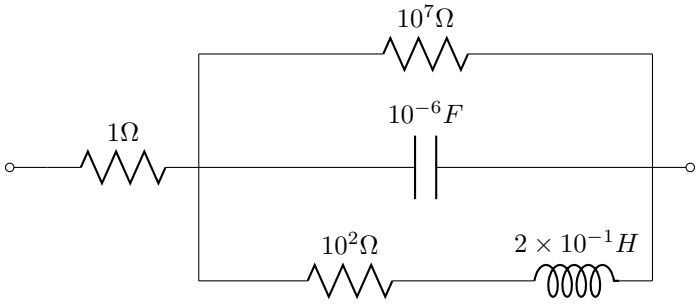
FIGURE 5 – Montage en parallèle testé.

★ **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 `NAire` qui vous est fournie.

▷ **Question 1:** Écrivez la classe `NSerie` 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 ω_i montés en série peut se calculer en utilisant la formule suivante : $\sum_{i=1}^{n} \omega_i$.

▷ **Question 2:** Écrivez la classe `NParallele` 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 ω_i montés en parallèle peut se calculer en utilisant la formule suivante : $\frac{1}{\sum_{i=1}^{n} \frac{1}{\omega_i}}$.

▷ **Question 3:** Testez votre code grâce qu test `testNAire` permettant de définir le dipôle suivant en utilisant ces nouveaux types (sans utiliser les types `Serie` ou `Parallele` précédemment définis) et de calculer son impédance pour $\omega = 314.16$. *Résultat escompté* $\approx$ *104.9604 + 60.764i.*



★ **Exercice 4:** On souhaite maintenant pouvoir ajouter dynamiquement des dipôles dans un circuit. Autrement dit, on souhaite remplacer l'implémentation de la classe abstraite `NAire` utilisant un tableau de taille fixe par une implémentation utilisant un tableau de taille variable (par exemple, une instance de la classe `java.util.ArrayList`.

▷ **Question 1:** Complétez la nouvelle implémentation de la classe `NAire` dans le fichier `NAireVariable.java` Vous ajouterez les méthodes d'ajout et de suppression d'un dipôle à son interface.

▷ **Question 2:** Implémentez les classes `VParallele` et `VSerie` qui sont des ensembles de dipoles comprenant un nombre variable d'éléments, et montés respectivement en parallèle ou en série. Vous pouvez être amenés à utiliser un itérateur.

▷ **Question 3:** Testez votre code avec le test `testVariable`, qui utilise les mêmes dipoles que l'exercice précédent, mais avec les nouvelles implémentations.

★ **Exercice 5:** L'objectif est maintenant de vous faire *instancier* des dipoles complexes.

▷ **Question 1:** Implémentez les méthodes `String toString()` dans toutes vos classes, et testez votre travail avec `testToString()`.

▷ **Question 2:** Complétez le code de la classe `Instances` pour réaliser les dipoles des figures 6 et 7. Ils sont testés par `testInstances()`.

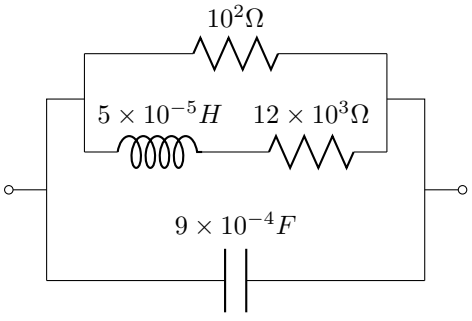


FIGURE 6 – Le dipole `dip1` à réaliser.

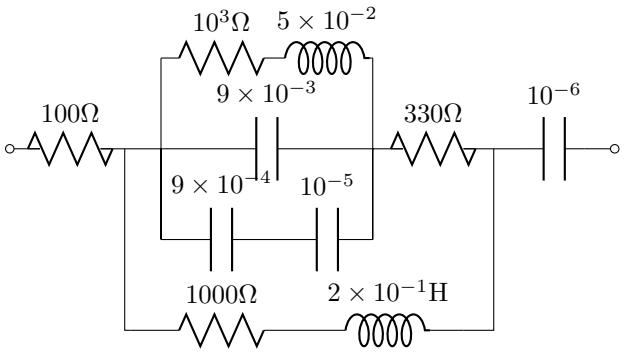


FIGURE 7 – Le dipole `dip2` à réaliser.