TP4 : Héritage, redéfinition, liaison dynamique.

POO : Programmation Orientée Objet
Première année

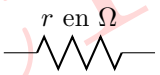
L'objectif de ce TP est de concevoir un programme permettant de modéliser les dipôles électriques, puis de calculer leur *impédance*.

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)

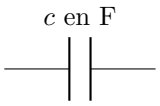


$z = r$

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

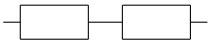
$z = i(\omega * l)$

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



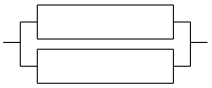
$z = i(\frac{-1}{\omega * c})$

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



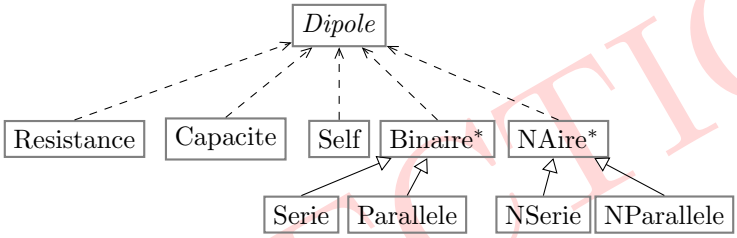
$z = z_1 + z_2$

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-4.10.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 $0.0 + 21.99i$.

Réponse

Le code présenté dans la correction est barbouillé de marqueurs pour qu'une moulinette puisse construire la solution toute seule à partir du code correct. Le plus simple pour vous est d'ignorer tous les commentaires.

Si vous voulez comprendre, c'est très simple. Toutes les lignes contenant la chaine KILLIT sont effacées du template. Les lignes entre BEGINKILL et ENDKILL sont également supprimées. S'il y a un REMPLACE entre les deux, le bloc est remplacé par ce qu'il y a entre REPLACE et ENDKILL.

Ainsi, dans ce cas, l'attribut et le contenu du constructeur sont cachés, tandis que le contenu de la methode est remplacé par un simple `return null`.

Self.java

```
1  package dipole;
2
3  public class Self implements Dipole {
4      private double valeur; //KILLIT
5
6      public Self(double val) {
7          valeur = val; //KILLIT
8      }
9
10     public Complexe impedance(double omega) {
11         //BEGINKILL
12         return new Complexe(0,omega*valeur);
13         // REPLACE
14         // return null;
15         // ENDKILL
16     }
17     public String toString() {
18         // BEGINKILL
19         return "Self("+valeur+")";
20         // REPLACE
21         //return null;
22         // ENDKILL
23     }
24 }
```

Fin réponse

▷ **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()`.

Réponse

```
1  package dipole;
2
3  public class Capacite implements Dipole {
4      private double val;//KILLIT
5
6      public Capacite(double d) {
7          val=d;//KILLIT
8      }
9
10     public Complexe impedance(double omega) {
11         // BEGINKILL
```

```
        return new Complexe(0., -1/(omega*val));
        // REPLACE
        //return null;
        // ENDKILL
    }
    public String toString() {
        // BEGINKILL
        return "Capacite("+val+")";
        // REPLACE
        //return null;
        // ENDKILL
    }
}
```

Fin réponse

▷ **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()`.

Réponse

```
package dipole ;

public class Resistance implements Dipole {
    double resistance ; //KILLIT

    public Resistance(double r){
    resistance = r;//KILLIT
    }

    public Complexe impedance(double omega) {
        /* BEGINKILL */
        return new Complexe(resistance, 0);
        /* REPLACE
        return null;
         ENDKILL */
    }
    public String toString() {
        // BEGINKILL
        return "Resistance("+resistance+")";
        // REPLACE
        //return null;
        // ENDKILL
    }
} // Resistance
```

Fin réponse

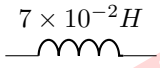


FIGURE 1 – La self testée.

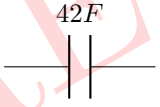


FIGURE 2 – La capacité testée.

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.** 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`.

Réponse

```
package dipole;

public class Serie extends Binaire implements Dipole {
```

```
    public Serie(Dipole d1,Dipole d2) {
        super(d1,d2);
    }
    @Override //KILLIT
    public Complexe impedance(double omega) {
        /* BEGINKILL */
        return dip1.impedance(omega).add(dip2.impedance(omega));
        /* REPLACE
        return null;
         ENDKILL */
    }
    public String toString() {
        // BEGINKILL
        return "Serie("+dip1.toString()+", "+dip2.toString()+")";
        // REPLACE
        // return "Serie(??)";
        // ENDKILL
    }

}
```

Fin réponse

▷ **Question 6.** 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 7.** 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`.

Réponse

```
package dipole;

public class Parallele extends Binaire implements Dipole {
    public Parallele(Dipole d1,Dipole d2) {
        super(d1,d2);
    }
    @Override
    public Complexe impedance(double omega) {
        /* BEGINKILL */
        Complexe c1 = dip1.impedance(omega);
        Complexe c2 = dip2.impedance(omega);
        c1.inverse();
        c2.inverse();
        c1.add(c2);
        c1.inverse();
        return c1;
        /* REPLACE
        return null;
         * ENDKILL */
    }
    public String toString() {
        // BEGINKILL
        return "Parallele("+dip1.toString()+", "+dip2.toString()+")";
        // REPLACE
        // return "Parallele(??)";
        // ENDKILL
    }
}
```

Fin réponse

▷ **Question 8.** 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).*

★ **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.

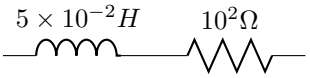


FIGURE 4 – Montage en série testé.

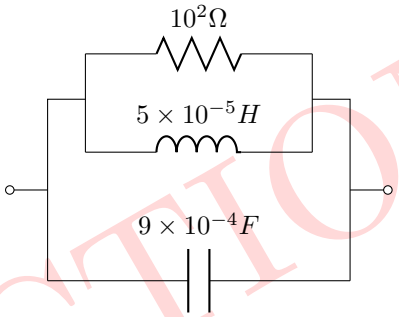


FIGURE 5 – Montage en parallèle testé.

▷ **Question 9.** 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 : $\sum_{i=1}^{n} \omega_i$.

Réponse

```java
package dipole;

public class NSerie extends NAire {
    public NSerie(Dipole[] t) {
        super(t);
    }

    @Override
    public Complexe impedance(double omega) {
        /* BEGINKILL */
        Complexe res = new Complexe(0,0);
        for (Dipole d: tab) {
            res.add(d.impedance(omega));
        }
        return res;
        /* REPLACE
        return null;
         ENDKILL */
    }
    public String toString() {
        // BEGINKILL
        StringBuffer sb = new StringBuffer();
        sb.append("Serie(");
        boolean first = true;
        for (Dipole d: tab) {
                if (first) {
                        first = false;
                } else {
                        sb.append(", ");
                }
                sb.append(d.toString());
        }
        sb.append(")");
        return sb.toString();
        // REPLACE
        // return "Serie(??)";
        // ENDKILL
    }

}
```

Fin réponse

▷ **Question 10.** 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

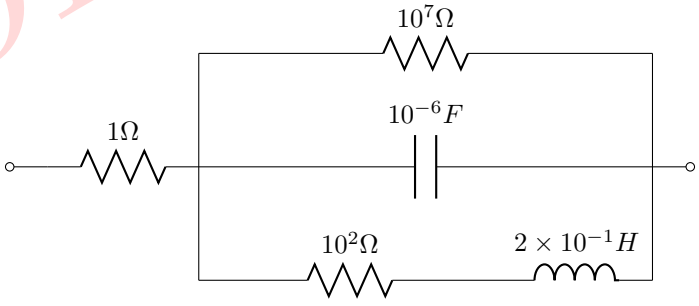
en parallèle peut se calculer en utilisant la formule suivante : $\frac{1}{\sum_{i=1}^{n} \frac{1}{\omega_i}}$.

Réponse

```
package dipole;

public class NParallele extends NAire {

    public NParallele(Dipole[] t) {
        super(t);
    }

    @Override
    public Complexe impedance(double omega) {
        /* BEGINKILL */
        Complexe res = new Complexe(0,0);
        for (Dipole d: tab) {
            Complexe c = d.impedance(omega);
            c.inverse();

            res.add(c);
        }
        res.inverse();
        return res;
        /* REPLACE
        return null;
         ENDKILL */
    }
    public String toString() {
        // BEGINKILL
        StringBuffer sb = new StringBuffer();
        sb.append("Parallele(");
        boolean first = true;
        for (Dipole d: tab) {
                if (first) {
                        first = false;
                } else {
                        sb.append(", ");
                }
                sb.append(d.toString());
        }
        sb.append(")");
        return sb.toString();
        // REPLACE
        // return "Parallele(??)";
        // ENDKILL
    }

}
```

Fin réponse

▷ **Question 11.** 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 12.** 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.

Réponse

```java
package dipole;

import java.util.ArrayList;

public abstract class NAireVariable implements Dipole {
    ArrayList<Dipole> dips = new ArrayList<Dipole>(); //KILLIT

    public void addDipole(Dipole d) {
        dips.add(d);//KILLIT
    }
}
```

Fin réponse

▷ **Question 13.** 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.

Réponse

```java
package dipole;

public class VParallele extends NAireVariable {

    @Override
    public Complexe impedance(double omega) {
        /* BEGINKILL */
        Complexe res = new Complexe(0,0);
        for (Dipole d: dips) {
            Complexe c = d.impedance(omega);
            c.inverse();

            res.add(c);
        }
        res.inverse();
        return res;
        /* REPLACE
        return null;
         ENDKILL */
    }

    public String toString() {
        // BEGINKILL
        StringBuffer sb = new StringBuffer();
        sb.append("Parallele(");
        boolean first = true;
        for (Dipole d: dips) {
                if (first) {
                        first = false;
                } else {
                        sb.append(", ");
                }
                sb.append(d.toString());
        }
        sb.append(")");
        return sb.toString();
        // REPLACE
        // return "Parallele(??)";
        // ENDKILL
    }

}
```

```java
package dipole;

public class VSerie extends NAireVariable {

    @Override
    public Complexe impedance(double omega) {
        /* BEGINKILL */
        Complexe res = new Complexe(0,0);
        for (Dipole d: dips) {
            res.add(d.impedance(omega));
        }
        return res;
        /* REPLACE
        return null;
         ENDKILL */
    }
    public String toString() {
        // BEGINKILL
        StringBuffer sb = new StringBuffer();
        sb.append("Serie(");
        boolean first = true;
        for (Dipole d: dips) {
                if (first) {
                        first = false;
                } else {
                        sb.append(", ");
                }
                sb.append(d.toString());
        }
        sb.append(")");
        return sb.toString();
        // REPLACE
        // return "Serie(??)";
        // ENDKILL
    }

}
```

Fin réponse

▷ **Question 14.** 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 15.** Implémentez les méthodes `String toString()` dans toutes vos classes, et testez votre travail avec `testToString()`.

▷ **Question 16.** Complétez le code de la classe `Instances` pour réaliser les dipoles des figures 6 et 7. Ils sont testés par `testInstances()`.

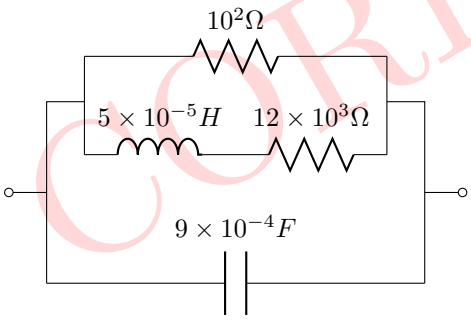


FIGURE 6 – Le dipole `dip1` à réaliser.

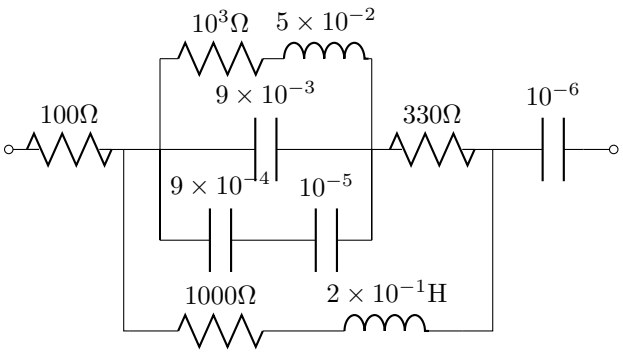


FIGURE 7 – Le dipole `dip2` à réaliser.