Inheritance, Overriding and Dynamic Binding

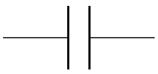
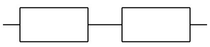
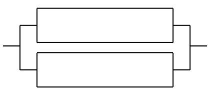
Prog1, Scala, L3

2015

1 Electrical Dipoles

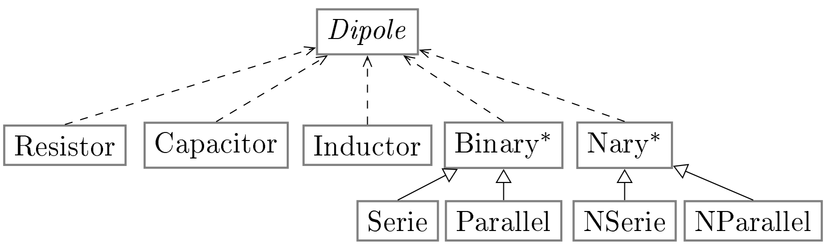
Every electrical circuit is composed of differing components such as resistors, capacitors, diodes and electromagnetic coils. They can be assembled in either series or parallel circuits. Depending on their component, each circuit present a specific resistance to the current when a voltage is applied. The **impedance** extends this notion of resistance to alternating currents.

Given ω the angular frequency of the current, the impedance z of the circuit can be computed as follows (with the constant $i = 1\angle\frac{\pi}{2} = e^{j\frac{\pi}{2}}$).

Symbol	*Description*	*Impedance*
r in Ω 	A **resistor** of value r expressed in ohms (noted Ω)	$z = r$
l in H 	An **inductor** of value l expressed in henries (noted H)	$z = i(\omega * l)$
c in F 	A **capacitor** of value c expressed in farad (noted F)	$z = i(\frac{-1}{\omega * c})$
	A **serie circuit** with 2 dipoles of impedance z_1 and z_2	$z = z_1 + z_2$
	A **parallel circuit** with 2 dipoles of impedance z_1 and z_2	$z = \frac{1}{\frac{1}{z_1} + \frac{1}{z_2}}$

1.1 Modeling `Dipole`

We will use the following class hierarchy to model the electrical dipoles.

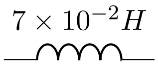


The *Dipole* trait provides an abstract method `impedance(omega:Double):Complex`, implemented in each concrete class of the hierarchy. The parameter `omega` represents the angular frequency of the current. The impedance is a complex number (only resistors have a real impedance).

1.2 Implementing `Dipole`

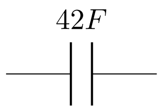
Download and unpack the provided code to your local disk. Rename the `template` directory to `src`[1], open the sources with your favorite editor (or `geany` if you have no favorite editor yet). The provided code contains a `build.sbt` file for easy compilation, as well as a set of unit tests checking the features that you should implement. You should run the tests often during your work[2] to track your progress. Of course, the tests for a given feature will fail until you implement that feature.

▷ **Question 1:** We will first implement the simple dipoles. Fill the classes `Resistor`, `Capacitor` and `Inductor` in file `src/main/scala/ dipole/SimpleDipoles.scala` to pass the first set of tests. You have to implement both `impedance()` and `toString()`.



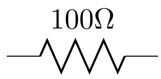
($z \approx 22j\ \Omega$)

Tested Inductor.



($z \approx -7.6 \times 10^{-5}j\ \Omega$)

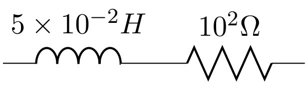
Tested Capacitor.



($z = 100\ \Omega$)

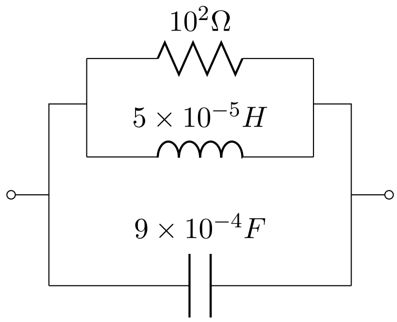
Tested Resistor.

▷ **Question 2:** We will now implement the binary circuits, either built in parallel or in serie. Check your implementation with the provided tests, that use the following circuits.



($z \approx 100.0 + 15.70j\ \Omega$)

Tested Serie Circuit.



($z \approx 0.2079 + -4.55j\ \Omega$)

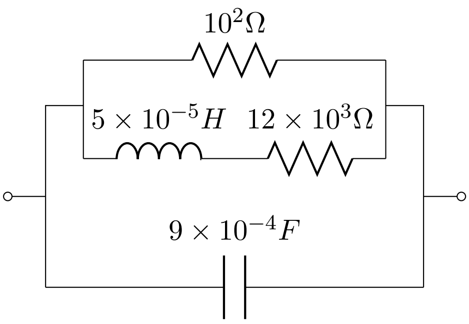
Tested Parallel Circuit.

▷ **Question 3:** We will now implement N-ary circuits. The impedance of a serie circuit is simply given by $\sum_{i=1}^{n} \omega_i$ while the impedance of a parallel circuit is given by $\frac{1}{\sum_{i=1}^{n} \frac{1}{\omega_i}}$.

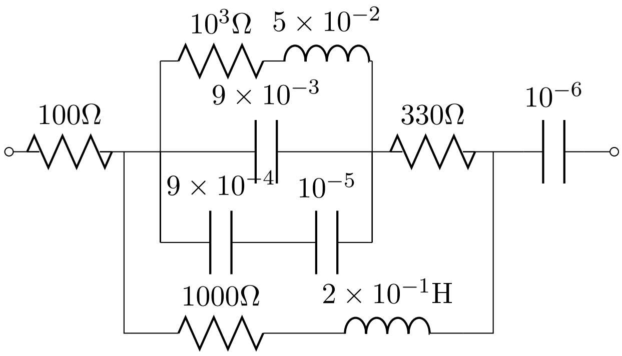
Note that you also have to implement a `Nary.::(head:Dipole)` method, enabling to build a new circuit by appending a dipole to the currently defined one.

▷ **Question 4:** You will now fill the file `src/main/scala/dipole/Instances.scala` to define the following instances of dipoles:

[1] mv template src
[2] sbt test



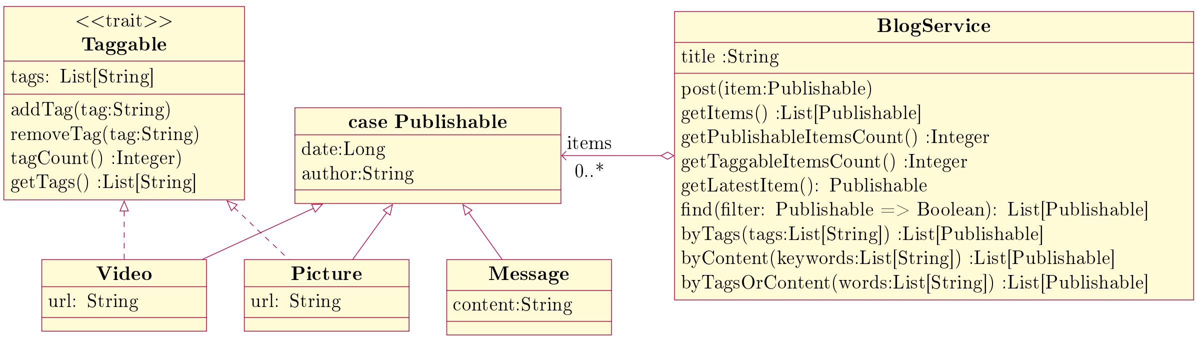
The `dip1` dipole.



The `dip2` dipole.

2 Blogging

We will now implement a blogging micro-system: a web site constituted of posts aggregated over time. These posts can be text messages, images or videos. *Tags* can be attached to images or videos, to select the ones that match a given set of keywords. The textual messages cannot be tagged, but the textual search should operate on their content directly.



▷ **Question 1:** Implement this hierarchy of classes, and test your work with an appropriate specification.