# Lab 3, CSC/CPE 203
# Interface, Classes
# Due: 10/23

Remember the greenhouse gas emissions data we played with in Lab 2? In this lab, we will create some simple but powerful visualizations of the data to help us understand it a bit better.

## Objectives

- To develop and demonstrate basic object-oriented development skills. Much of the structure of the solution is given—use good judgement when "filling in the blanks".
- To become familiar with Java interfaces and the concept of polymorphism.
- To practice using the `instanceof` operator.
- More practice using existing classes and interfaces from the Java standard library, such as: `List`, `ArrayList` and `LinkedList`.

## Given files

Obtain the starter code for this lab from Canvas.

A number of files are provided to you to begin with. Take a moment to peruse these files: * `Main.java`: The entry point for this program. This class simply creates a `Plotter` (see next). No modifications are needed to this file. * `Plotter.java`: Plots greenhouse gas emissions data. Some modifications will be made to this file. * `EmissionsDatabase.java`: Loads emissions data from provided data files and makes it available for analysis. Small changes will be needed to this file. Take some time to read through the methods (or method comments) in this file to understand the functionality it provides.

**If you have any questions about these files, don't hesitate to ask the instructor or your neighbor.**

## Setup

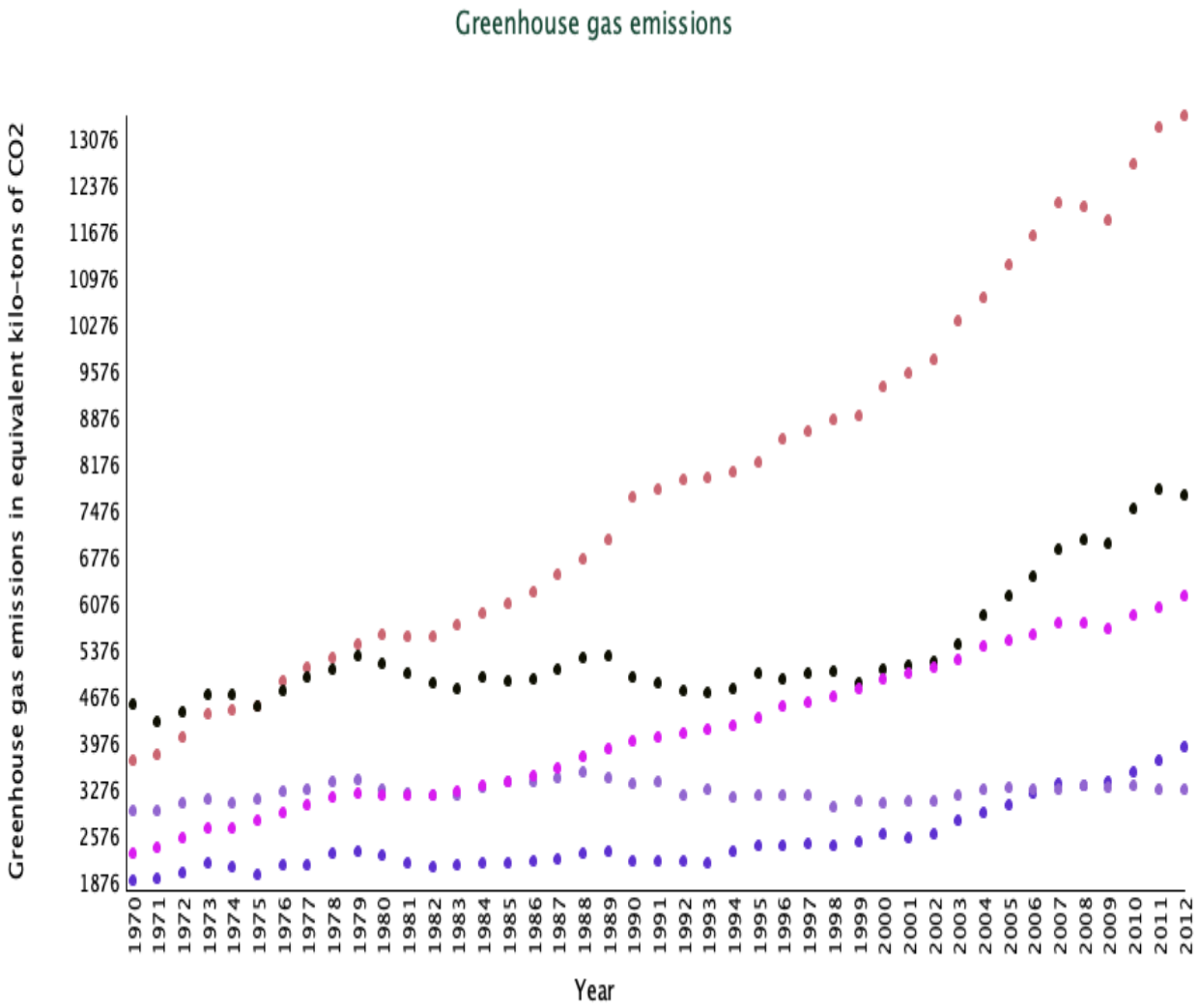
You will notice that the starter code contains a bunch of syntax errors. This is because it references classes that do not exist, namely `Country` and `Sector`.

So, before doing anything else:

- Copy the *final* (`part3`) versions of `Country.java`, `Sector.java`, and `Emission.java` from Lab 2 into the `src` directory of Lab 3.
- Be sure to remove the `package` declaration at the top of each file.[1]

- If you don't have a working copy of those files, talk to your neighbor or to the instructor to get up to speed. If you have a complete copy of the files and your neighbor does not, help them get up to speed.

Once you have done this, the syntax errors should disappear. Run the program from the `main` method. You should see figure like the following (point colors may be different):




Screenshot of sector emissions data

The figure displays global emissions data for different industry sectors from 1970 to 2012. We can immediately notice some trends from the data. For example, the Power industry's greenhouse gas emissions have risen quite dramatically over the last two decades.

# Tasks

## Part 1: Creating an interface

Look at `Plotter.java`. This file makes use of a library called *processing* to help create the graph. Take some time to read the method comments in this file, then focus your attention on the `plotSectorData` method.

Note the following:

- The method takes in a `List<Sector>`. This list can be obtained from the `EmissionsDatabase`.
- The graph plotting code makes use of the minimum and maximum Sector emissions, also obtained from the `EmissionsDatabase`.
- *Importantly, the method only lets us plot Sector data.*

Our task in this lab is to modify this method to allow it to work with both `Sector` data and `Country` data. We will accomplish this using the Object-oriented construct of an *interface*.

### 1. Create a new interface called `GreenhouseGasEmitter`.

The interface should have the following abstract methods:

- `String getName()`
- `double getTotalEmissionsInYear(int year)`

### 2. Make `Country.java` and `Sector.java implement` the interface you just created.

This means that `Country.java` and `Sector.java` ought to have the following methods:

- `public String getName()`: Return the name of the `Country` or `Sector`
- `public double getTotalEmissionsInYear(int year)`: For the `Sector`, simply return the value in its map of emissions corresponding to the given year. For the `Country`, recall that the emissions are stored as `Emission` objects, keeping track of CO2, CH4 and N20 emissions. Return the *total* emissions for the given year, i.e., CO2 + CH4 + N20.

  Important: Run the code again to check that nothing is broken! Move on *only if* you see the visualization pop up like before.

### 3. Modify the `plotSectorData` method to work with `GreenhouseGasEmitters`.

Currently, the `draw` method calls the `plotSectorData` method, but has no way to plot `Country` data.

This part is a bit more work than the rest. But we'll approach it step-by-step! In high-level terms, we want to *refactor* the method so that it is not working specifically on `Sectors`, but rather can work on any `GreenhouseGasEmitters`. Read the code in the method and see if you can identify spots where we need to become a bit more general and less "sector-y".

I see the following:

- The method name, `plotSectorData` definitely needs to change.
- The method takes in a `List<Sector>` as its parameter.
- The for-each loop on line 72 is looping through this list of `Sectors`.
- Line 65 draws the y-axis for the graph. To do this, it refers to minimum and maximum `Sector` emissions obtained from the `EmissionsDatabase` methods `getSectorMinEmissions` and `getSectorMaxEmissions`, respectively.
- Line 85 figures out the y-coordinate where the data point will be drawn—this line also refers to `Sector`-specific min and max values.

Below, we will understand and tackle these changes one-by-one. Read the following carefully, and make the changes marked by ***TODO***

***TODO*** The first step is to rename the method. Let's call it `plotEmissionsData`. (Be sure to also make the change in the place where the method is *called*; see the `draw` method in `Plotter`).

We could change the `List<Sector>` to work with `GreenhouseGasEmitter` right now, but the other changes are more local to `Plotter.java`. We can make our lives easier by tackling them first.

So we move on to line 65. Currently, the line reads:

```
writeEmissionsAxis(this.db.getSectorMinEmission(),
this.db.getSectorMaxEmission());
```

The method `writeEmissionsAxis` needs the min and max value for the y-axis, and we are currently passing it the min and max values for `Sectors` specifically. Let's give the `draw` method more control over this, since that is where the plotting method is actually called.

***TODO*** Modify the `plotEmissionsData` method signature so that it takes two additional parameters, like so:

```
plotEmissionsData(List<Sector> sectors, double min, double max)
```

***TODO*** Change the call to `writeEmissionsAxis` on line 65 to use these provided `min` and `max` values.

***TODO*** Do the same for the call to `mapEmissions` on line 85.

***TODO*** Finally, since you have changed the `plotEmissionsData` method definition, you also need to change its usage. So in the `draw` method (on line 56), where the method is called without specific min and max values, change it to use the Sector min and max values:

```
this.plotEmissionsData(this.db.getSectors(), this.db.getSectorMinEmissions(),
this.db.getSectorMaxEmissions())
```

> Think: Why is it okay to be "sector-specific" here? Because here, we are a *consumer* of the `plotEmissionsData` method—we are making the decision to plot sector data, but we are not *forced* to do so. Well, we are, but we'll soon change that! :-)

Run the program and confirm that you're still seeing the sector data graph.

Our goal now is to complete this change to be able to plot `Country` data if we so choose.

Remember the `GreenhouseGasEmitter` interface? Let's use it.

***TODO*** Change the `plotEmissionsData` method signature to accept a `List<GreenhouseGasEmitter>` instead of a `List<Sector>`. For the sake of consistency, also rename the parameter variable from `sectors` to `emitters`.

Uh-oh, we now have syntax errors!

***TODO*** Remember the for-each loop on line 72? Modify it to work on the `emitters` object.

```
for (GreenhouseGasEmitter emitter : emitters) // for each
GreenhouseGasEmitter in emitters...
```

Notice that there is now a syntax error on line 56. If you hover your mouse over it, it will tell you that the `plotEmissionsData` method expects `List<GreenhouseGasEmitter>`, but you are giving it a `List<Sector>`.

But how can this be? Since `Sector implements GreenhouseGasEmitter`, isn't a list of `Sectors` the same as a list of `GreenhouseGasEmitters`?

Well, yes, but the Java compiler doesn't know that until *run time*, i.e., when the program is running. At *compile time*, it only knows that the method expected a `List<GreenhouseGasEmitter>` and was given a `List<Sector>`. **Recall the difference between static types and dynamic types.**

Therefore, we will change the program so that when we call `plotEmissionsData` we give it an object of static type `List<GreenhouseGasEmitter>`. To make this change, we will venture into `EmissionsDatabase.java`, which is where the `List<Sector>` comes from. Open that file.

Note that the instance variable `sectors` in this class is of type `List<Sector>`. Relatedly, the getter method `getSectors` returns a `List<Sector>`.

***TODO*** Change both the variable type and the getter method's return type to `List<GreenhouseGasEmitter>`.

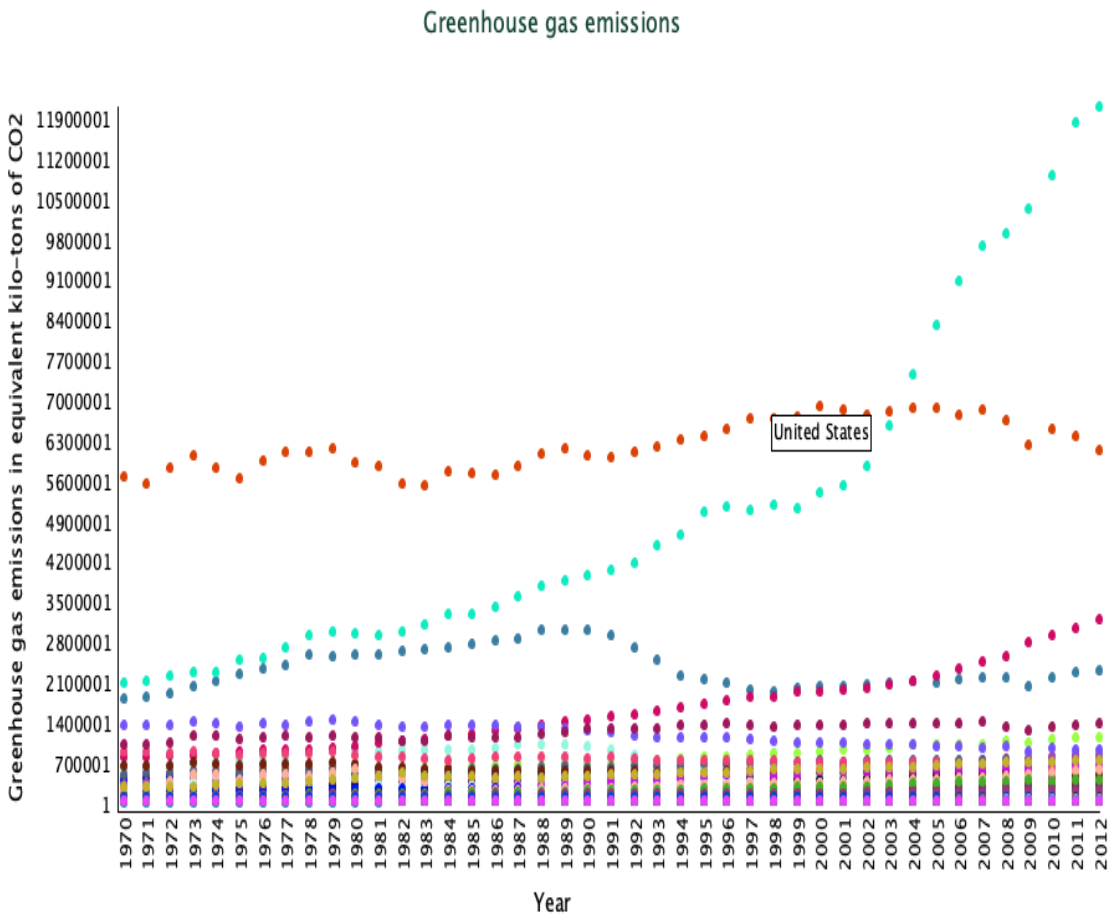
***TODO*** Since the goal is to easily plot `Sector` or `Country` data, do the same for the `countries` instance variable and `getCountries` getter method. (After all, `Country implements GreenhouseGasEmitter` as well.)

Upon returning to `Plotter.java`, see that the syntax errors have disappeared. Run the program again, ensuring that the graph showing Sector data appears.

> **So, what have we achieved?** We have now made many changes to our program, but it is functionally identical to what we had before. However, we've improved the quality of our code—it is now easier to modify to accommodate changing requirements. This is a core tenet of *agile software development*. In this case, the changed requirement is that we should be able to plot either `Sector` data or `Country` data, as we choose.

***TODO*** Modify the arguments passed to `plotEmissionsData` so that it plots `Country` data instead of `Sector` data.

When you're finished, you should see the following plot (colors might be different):




Screenshot of Country emissions data

---

1. This is because the files were originally in the `part3` package in Lab 2, but are in the *default* package in Lab 3 (i.e., directly within the `src`) directory.↩