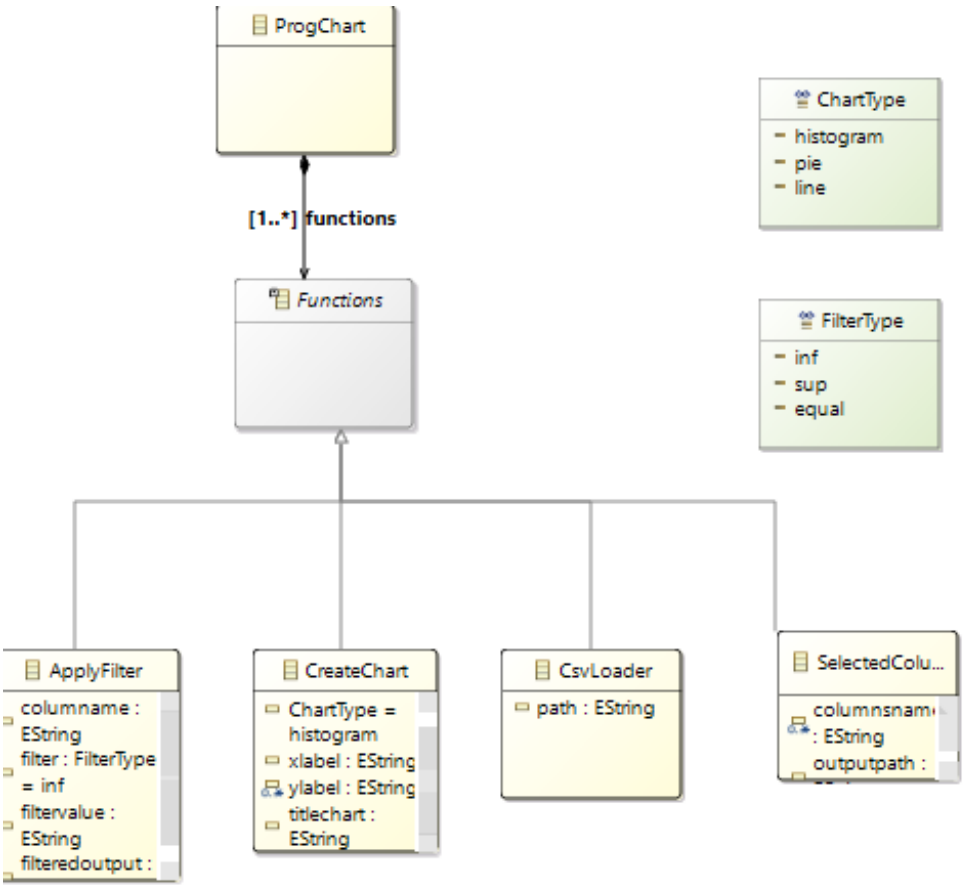
Projet Language et Compilation ChartIt

EL BAHRI SAWSEN

Explication du projet :

1/ méta modèle



- progChart contient 1 ou plusieurs Functions

- Applyfilter , creatChart , CsvLoader ,Selectedcolumns sont des fonctions :

  -Createchart : contient 3 attribut ; chart de type chartType qui est une enum (histogram , line , pie) Xlabel et ylabel (peut être 1 ou plusieurs) pour les axes de mon chart et bien sur titlechart pour le titre de graphique generée .

  -Applyfilter : qui contient columname , filtervalue , filter de type filterType (sup , inf equal) et filteredoutput le lien de mon csvfiltred generée

  -CsvLoader : qui contient l'attribut path

  SelectedColumns : columname et outputpath aussi pour générer le fichier du selectedcsv

2/ fichier XTEXT

```
import "http://www.example.org/programChartSawsen"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

ProgChart returns ProgChart:
    'Start' '{' functions+=Functions ( ";" functions+=Functions)* '}' ;

Functions returns Functions:
    ApplyFilter | CreateChart | CsvLoader |SelectedColumns;

ApplyFilter returns ApplyFilter:
    {ApplyFilter}
    'ApplyFilter'
    '('
        ( columname=EString)?
        ( filter=FilterType)?
        ( filtervalue=EString)? ","
        ( filteredoutput=EString )?
    ')';

CreateChart returns CreateChart:
    {CreateChart}
    'CreateChart' "("('chart' chart=ChartType) ","  ( xlabel=EString) ","   ( ylabel=EString) ")"?;
SelectedColumns returns SelectedColumns:
'SelectedColumns' '(' columnsname+=EString (',' columnsname+=EString)*  (',' 'outputpath' '=' outputpath=EString)? ')';

CsvLoader returns CsvLoader:
    {CsvLoader}
    'CsvLoader''('('path' '=' path=EString) ')'?;

EString returns ecore::EString:
    STRING | ID| 'C:/Users/DELL/Downloads/selected_cars.csv';

enum FilterType returns FilterType:
               inf = 'inf' | sup = 'sup' | equal = 'equal';

enum ChartType returns ChartType:
               histogram = 'histogram' | pie = 'pie' | line = 'line';
```

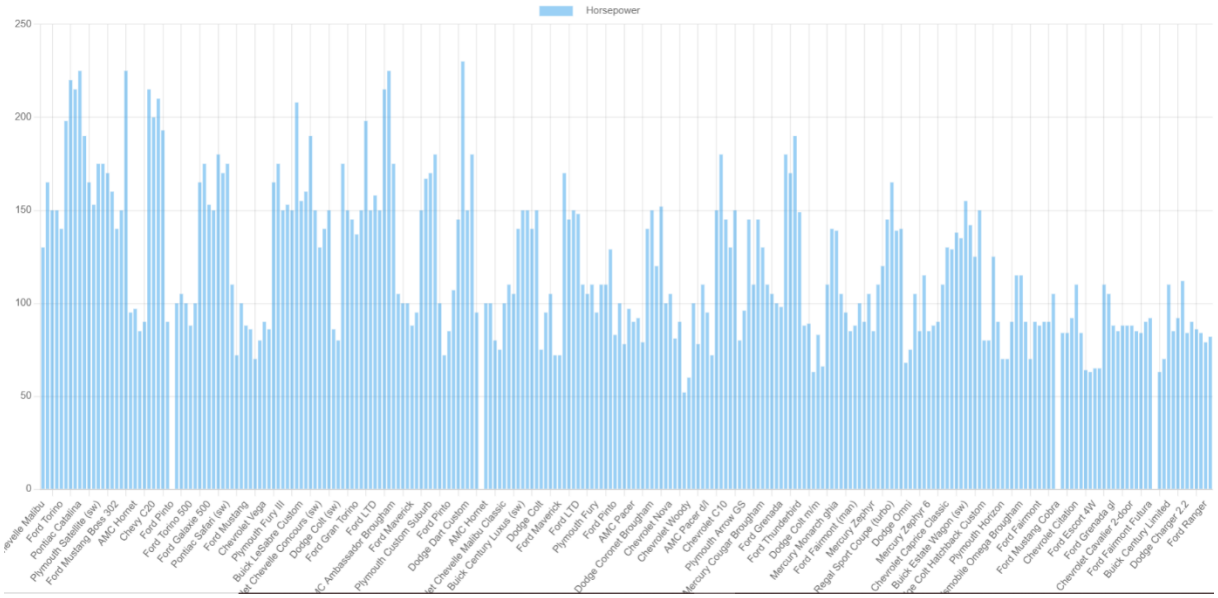
- **Il faut suivre les règles afin d'utiliser le programme :**

  Voici un example :

```
Start
{
CsvLoader(path ="C:/Users/DELL/Downloads/factboo.csv") ;
SelectedColumns( "Country","Population","Telephonesmainlinesinuse",outputpath="C:/Users/DELL/Downloads/selectedfactboo.csv");
ApplyFilter("Telephonesmainlinesinuse" sup "0", "C:/Users/DELL/Downloads/filteredfactboo.csv");
CreateChart( chart line ,title : "graphique sawsen" ,  xlabel : "Country" ,ylabel :  "Population"  "Telephonesmainlinesinuse" )
}
```

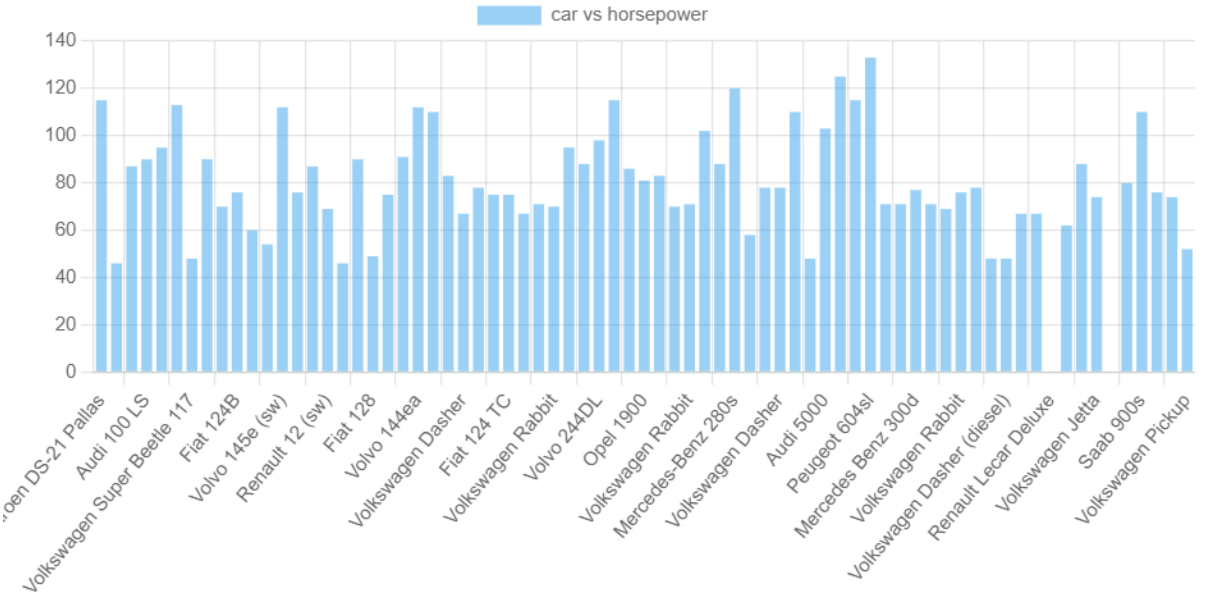
3/Les scenarios demandés :

- **Le 1er scenario :** car in us vs horsepower

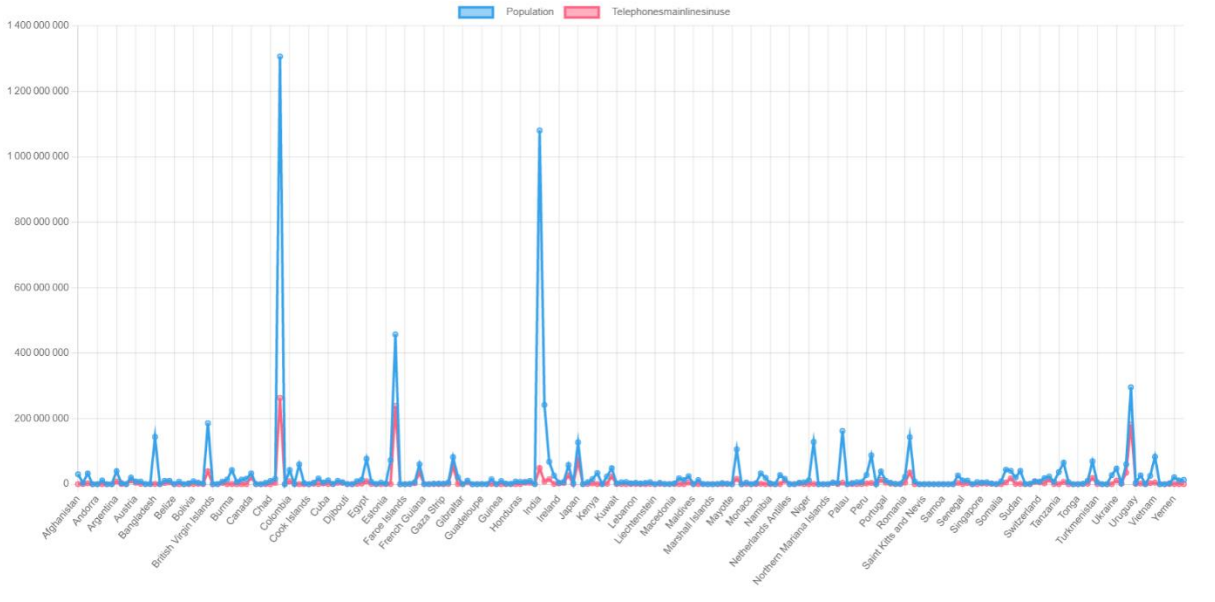



```
Start
{
CsvLoader(path ="C:/Users/DELL/Downloads/cars.csv") ;
SelectedColumns( "Car","Horsepower","Origin",outputpath="C:/Users/DELL/Downloads/selectedcars.csv");
ApplyFilter("Origin" equal "US", "C:/Users/DELL/Downloads/filteredcars.csv");
CreateChart( chart histogram ,title : "graphique sawsen" ,  xlabel : "Car" ,ylabel :  "Horsepower" )
}
```

- j'ai essayé aussi car in europe vs horsepower



- **2éme scenario :**




```
Start
{
CsvLoader(path ="C:/Users/DELL/Downloads/factboo.csv") ;
SelectedColumns( "Country","Population","Telephonesmainlinesinuse",outputpath="C:/Users/DELL/Downloads/selectedfactboo.csv");
ApplyFilter("Telephonesmainlinesinuse" sup "0", "C:/Users/DELL/Downloads/filteredfactboo.csv");
CreateChart( chart line ,title : "graphique sawsen" ,  xlabel : "Country" ,ylabel :  "Population"  "Telephonesmainlinesinuse" )
}
```

3/Le code xtend :

1/ fonction csvloader :

```
class MyDslGenerator extends AbstractGenerator {
  override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
    val prog = resource.contents.get(0) as ProgChart
    var selectedCsvContent = new ArrayList<String>() as List<String>
    var filteredCsvContent = new ArrayList<String>() as List<String>
    var csvFilePath = ""
    for (function : prog.functions) {
      if (function instanceof CsvLoader) {
        val csvLoader = function as CsvLoader
        csvFilePath = csvLoader.path
      }
    }
val csvContent = Files.readAllLines(Paths.get(csvFilePath))
val headers = csvContent.get(0).split(";")
val cleanedHeaders = headers.map[header | header.replaceAll("[^a-zA-Z0-9àâçéèêëîïôûùüÿñæœÀÂÇÉÈÊËÎÏÔÛÙÜŸÑÆŒ]", "")]
csvContent.set(0, cleanedHeaders.join(";"))
fsa.generateFile("output.csv", csvContent.join("\n"))
```

Explication : au début pour le premier scenario j'ai écrit une seule ligne pour lire le csv , mais après j'ai remarqué que le csv factbook contient des symboles : j'ai du ajouter une ligne de code qui permet de remplacer tout les caractères spéciaux par le vide (il faut bien voir les nouveaux noms des colonnes pour utiliser les autres fonctions) , il faut aussi mettre tout le chemin pour csv et ça genere un nouveau fichier « output.csv »

2/ fonction SelectedColumns :

```
for (function : prog.functions) {
    if (function instanceof SelectedColumns) {
      val selectedCols = function as SelectedColumns
      val selectedColumns = selectedCols.columnsname
      val outputpath = selectedCols.outputpath
      selectedCsvContent = processSelectedColumns(selectedColumns,csvContent)
    fsa.generateFile(selectedCols.outputpath,  selectedCsvContent.join("\n"))

    }
  }
```

```
def processSelectedColumns(List<String> selectedColumns, List<String> csvContent) {
    csvContent.stream().map [line |
        val splitLine = line.split(";")
        val selectedLine = selectedColumns.map[colName |
            val colIndex = csvContent.get(0).split(";").indexOf(colName)
            if (colIndex >= 0 && colIndex < splitLine.length) {
                splitLine.get(colIndex)
            } else {
                ""
            }
        ]
        if (selectedLine.contains[it == ""]) {
            null
        } else {
            selectedLine.join(";")
        }
    ].filter[it != null && it != ""].collect(Collectors.toList())
}
```

Explication : même pour cette fonction au début j'ai pas eu des problèmes pour sélectionner les colonnes, mais avec le 2éme scenario qui contient des valeurs manquantes : il m'affiche des erreurs : donc j'ai du changer ma fonction processSelectedColumns : lorsqu'on trouve une ligne dont il ya pas de valeurs on la garde aussi. La fonction SelectedColumns a aussi un outputpath . donc vous devez avoir un nouveau csv avec les colonnes sélectionnées

3/fonction ApplyFilter

```
for (function : prog.functions) {
    if (function instanceof SelectedColumns) {
      val selectedCols = function as SelectedColumns
      val selectedColumns = selectedCols.columnsname
      val outputpath = selectedCols.outputpath
      selectedCsvContent = processSelectedColumns(selectedColumns,csvContent)
    fsa.generateFile(selectedCols.outputpath,  selectedCsvContent.join("\n"))

    }
  }
```

```
def applyFilterfun(String columname, FilterType filter, String filtervalue, List<String> selectedCsvContent) {
  val filteredCsvContent = selectedCsvContent.stream().limit(2).collect(Collectors.toList()) // copier les deux premières lignes
  filteredCsvContent.addAll(selectedCsvContent.stream().skip(2).filter [line |
    val splitLine = line.split(";")
    val colIndex = selectedCsvContent.get(0).split(";").indexOf(columname)
    if (colIndex >= splitLine.length) {
      false // valeur manquante, on saute la ligne
    } else {
      val colValue = splitLine.get(colIndex)
      switch(filter) {
        case FilterType.EQUAL: colValue.equals(filtervalue)
        case FilterType.INF: Integer::parseInt(colValue) < Integer::parseInt(filtervalue)
        case FilterType.SUP: Integer::parseInt(colValue) > Integer::parseInt(filtervalue)
      }
    }
  ].collect(Collectors.toList()))
  return filteredCsvContent
}


}
```

Explication :applyfilterfun d'appliquer le filter sur le csv tout en gardant les premiers lignes (noms des colonnes et les types) , après pour la comparaison lorsque la ligne est vide on la saute sans faire la comparaison

Remarque : pour avoir le graphique à la fin il faut utiliser les 3 fontions : même ci il y a rien a filtrer vous pouvez comparez (sup 0)

```
ApplyFilter("Telephonesmainlinesinuse" sup "0", "C:/Users/DELL/Downloads/filteredfactboo.csv");
```

4/fonction CreateChart :

```
   val yLabels = new ArrayList<String>()
for (function : prog.functions) {
  if (function instanceof CreateChart) {
    val createChart = function as CreateChart
    val chart= createChart.chart
    val xlabel = createChart.xlabel
     val titlechart = createChart.titlechart
  val yLabelList = createChart.ylabel
    yLabels.addAll(yLabelList)
    var chartContent = ""

    switch (chart) {
      case ChartType.HISTOGRAM:
        chartContent = generateHistogramChart(xlabel, yLabels , filteredCsvContent, titlechart)
      case ChartType.PIE:
        chartContent = generatePieChart(xlabel,yLabels , filteredCsvContent , titlechart)
      case ChartType.LINE:
        chartContent = generateLineChart(xlabel,yLabels , filteredCsvContent , titlechart)

    }
      if (createChart.chart != null) {
      fsa.generateFile(createChart.chart + ".html", chartContent)
    }
  }
```

```
}}
def generateHistogramChart(String xlabel,  List<String> yLabels, List<String> filteredCsvContent , String titlechart) {
  // Convert filteredCsvContent to a JavaScript object array
  val headers = filteredCsvContent.get(0).split(";")
  val constdata = new ArrayList<String>()

  for (i : 2 .. filteredCsvContent.size - 1) {
    val line = filteredCsvContent.get(i)
    val row = line.split(";")
    val rowData = new StringBuilder()

    rowData.append("{")
    for (j : 0 .. headers.size - 1) {
      rowData.append("\"").append(headers.get(j)).append("\":\"").append(row.get(j)).append("\",")
    }
    rowData.deleteCharAt(rowData.length() - 1)
    rowData.append("}")

    // Replace the "=" sign with a ":" sign
    val jsonString = rowData.toString().replace("=", ":")
    constdata.add(jsonString)
  }
  // Create datasets
 var datasets = ""
 for (i : 0 ..< yLabels.size) {
    datasets += '''
        {
          label:'   '''+{yLabels.get(i)}+ ''' ',
          data: data.map(row => row.'''+{yLabels.get(i)} + ''')
        },
    '''
 }

// Create the HTML content
val chartContent = '''
   <html>
   <head>
     <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
     <title>'''+ titlechart + '''</title>
   </head>
   <body>
     <canvas id="myChart"></canvas>
     <script>
        const CHART_COLORS = {
          redt: 'rgba(255, 99, 132, 0.6)',
          red: 'rgb(255, 99, 132)',
          orange: 'rgb(255, 159, 64)',
          yellow: 'rgb(255, 205, 86)',
          green: 'rgb(75, 192, 192)',
          bluet: 'rgba(54, 162, 235,0.6)',
          blue: 'rgb(54, 162, 235)',
          purple: 'rgb(153, 102, 255)',
          grey: 'rgb(201, 203, 207)'
        };

        const NAMED_COLORS = [
          CHART_COLORS.red,
          CHART_COLORS.orange,
          CHART_COLORS.yellow,
          CHART_COLORS.green,
          CHART_COLORS.blue,
          CHART_COLORS.purple,
          CHART_COLORS.grey,
        ];
```

```
const data = ''' + constdata.toString.replaceAll("\\[", "\\[").replaceAll("\\]", "\\]") + ''';
console.log(data);
 new Chart(
    document.getElementById('myChart'),
    {
      type: 'bar',
     data: {
                labels: data.map(row => row.'''+ xlabel +'''),
                datasets: [ ''' + datasets + ''' ],
              }
            }
          );
          </script>
        </head>
        <body>
        </body>
      </html>
    '''  return chartContent
```

Explication : j'ai transformé filtrecsvcontent en tableau json dans constdata , et la boucle for pour parcourir tout les éléments dans Ylabels (dans le cas de scenario 2 on a 2 ylabels) et les stocker dans la variable datasets , j'ai ainsi appeler le titlechart dans la balise <title>.

Les 2 autres fonctions generatelinechart et generatepiechart ont le même code juste j'ai changè le type dans le html