Thiago Silva dos Santos

Advanced Object

Network Flow Analyser

Revision 1.0

1. Introduction

1.1 Purpose

This design will detail the implementation of the requirements as defined in the Software Requirements Specification – Network Flow Analyser – Phase 1.

1.2 System Overview

This project implements an object oriented application to analyse data files on network traffic.

1.3 Design Map

The application read a file and analyse its content as follows: Using MVC where the Model define the objects, the View represents the visualization of the data that the model contains. Strategy Pattern to create objects representing the strategies, which changes per context object, such as: Clients connections attempts, Server connections received, and etc. Also, using Factory Pattern to create the charts representing the strategies defined.

1.4 Definitions and Acronyms

MVC – Model View Controller

2. Documentation

2.1 Code Snippets

The UML diagram presented on Figure 1 (Attached) defines the Design Map presented on section 1.3.

2.2 Code Snippets

The 3 snippets of code presented on Figure 2, Figure 3 and Figure 4 (Attached) defines the 3 most impressive snippets in the code.

2.3 Work Evaluation

The presented application is based on different Java Design Patterns to perform different strategies, and complies with Object Oriented design. However, the application uses disk I/O operations to read the file and it rely on other resources, such as memory, cache, cpu

in order to have a good performance. A way to overcome that would be to create a multi threaded application instead of creating one single thread to read the file, and perform its operations.
On the application side, I believe the Observer Pattern Design could be implemented to define the Java JFX creation on the UI side, and could have a better utilization of Java best practices such as: encapsulation, inheritance and dynamic binding, for example.
The charts are been generated by JFrame library and would be a better approach to define the charts by using Javafx as well.

3. Human Interaction Design

This section provides the user interface design developed on the project. Below is presented Figure 5, which represents the main screen of the application. Figure 6, which represents the Connections Menu, and submenus Client and Server. Figure 7, which represents the Protocol Menu, and submenu. Figure 8, which represents the Destination Port Menu, and submenu, and Figure 9 which represents the Top 10 Clients Pie Chart output from clicking on Connections -> Clients.

3.1 Application’s Main Screen

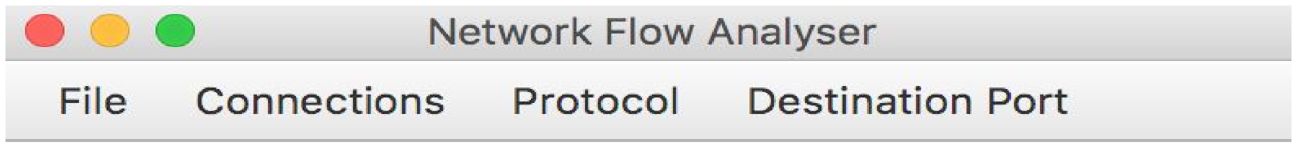


Figure 5: Application’s Main Screen

3.2 Connections Menu and Submenus

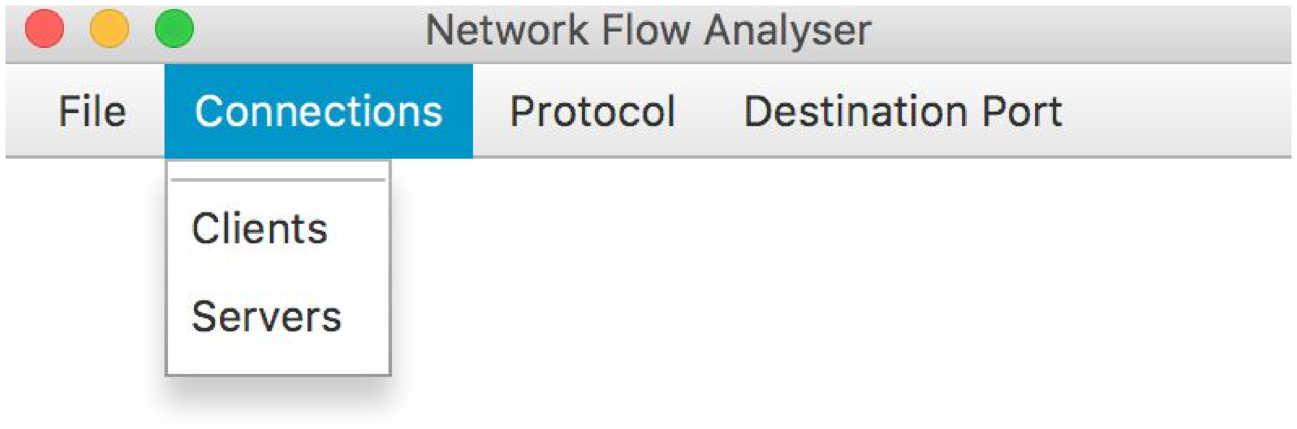


Figure 6: Connections Menu and Submenus Clients and Servers.

3.3 Protocol Menu and Submenu

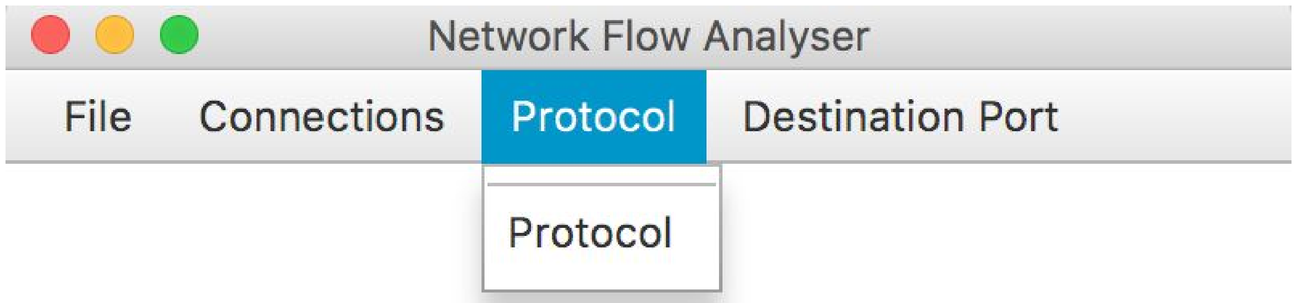


Figure 7: Protocol Menu and Submenu.

3.4 Destination Port Menu and Submenu

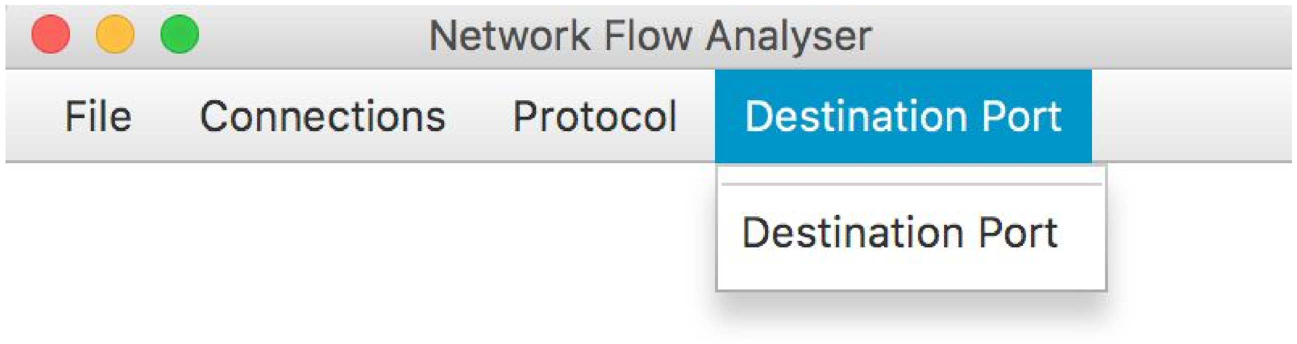


Figure 8: Destination Port Menu and Submenu.

3.5 Top 10 Clients Connections

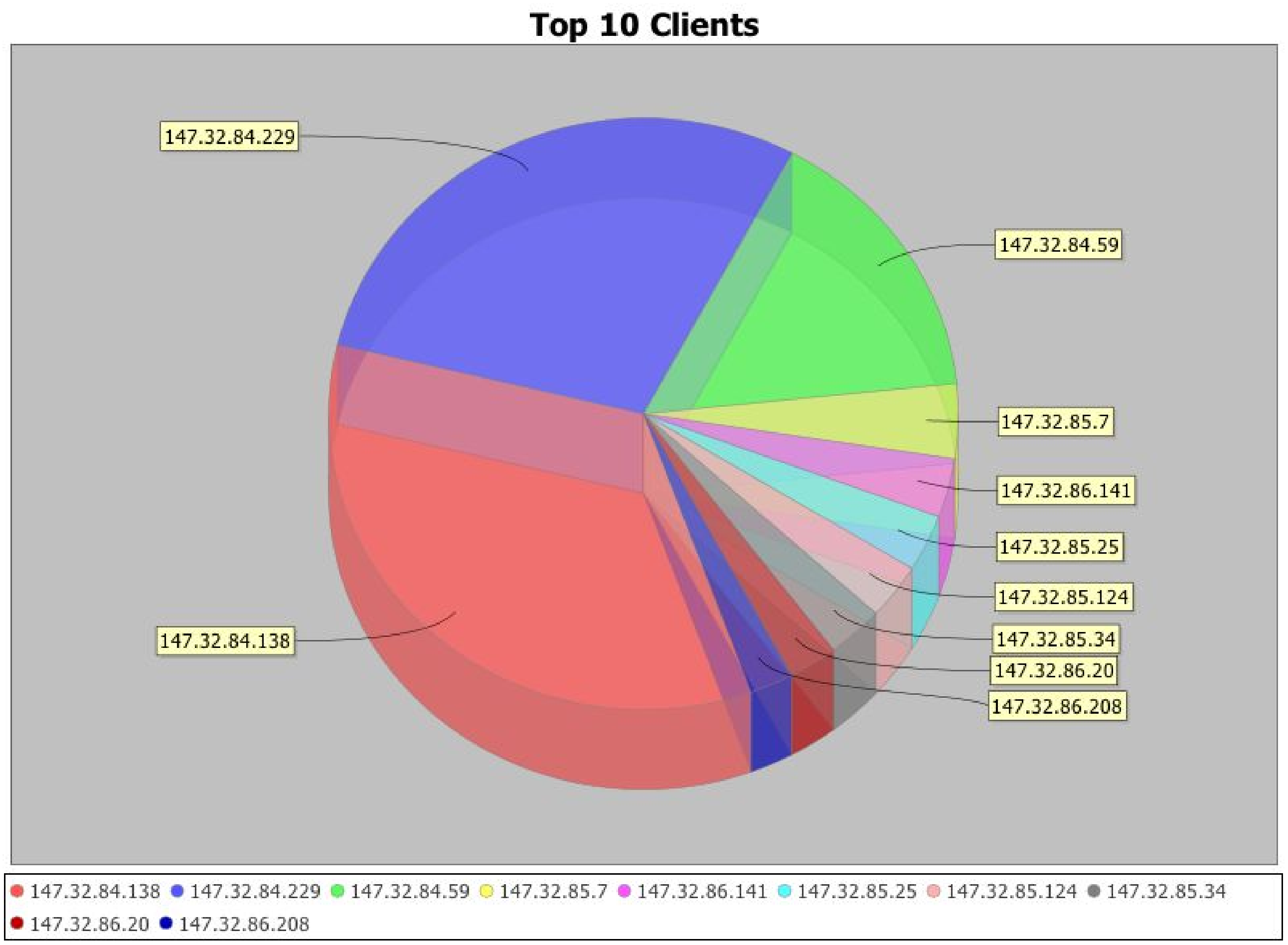
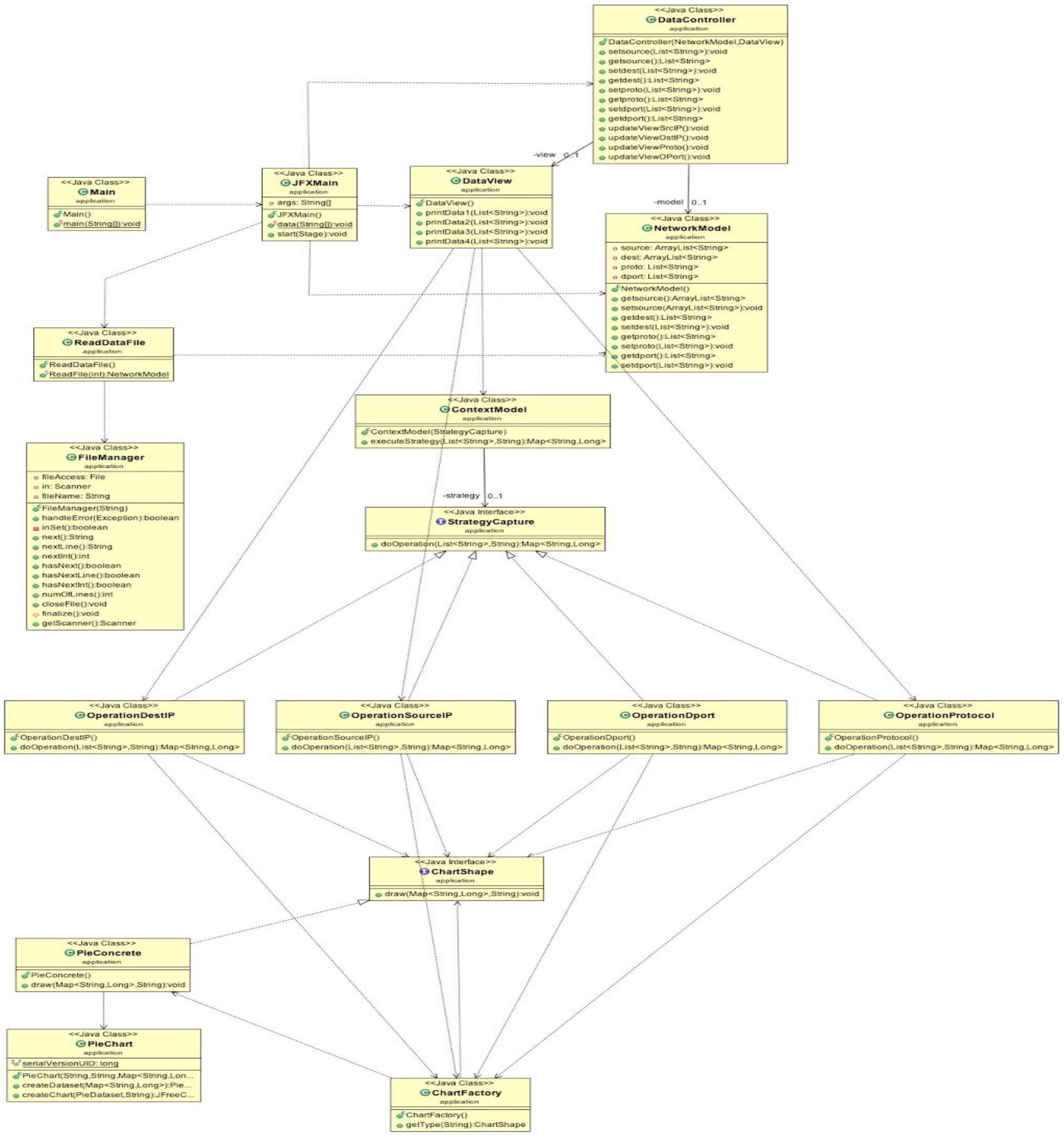


Figure 9: Top 10 Clients - Percentage

4. Appendix

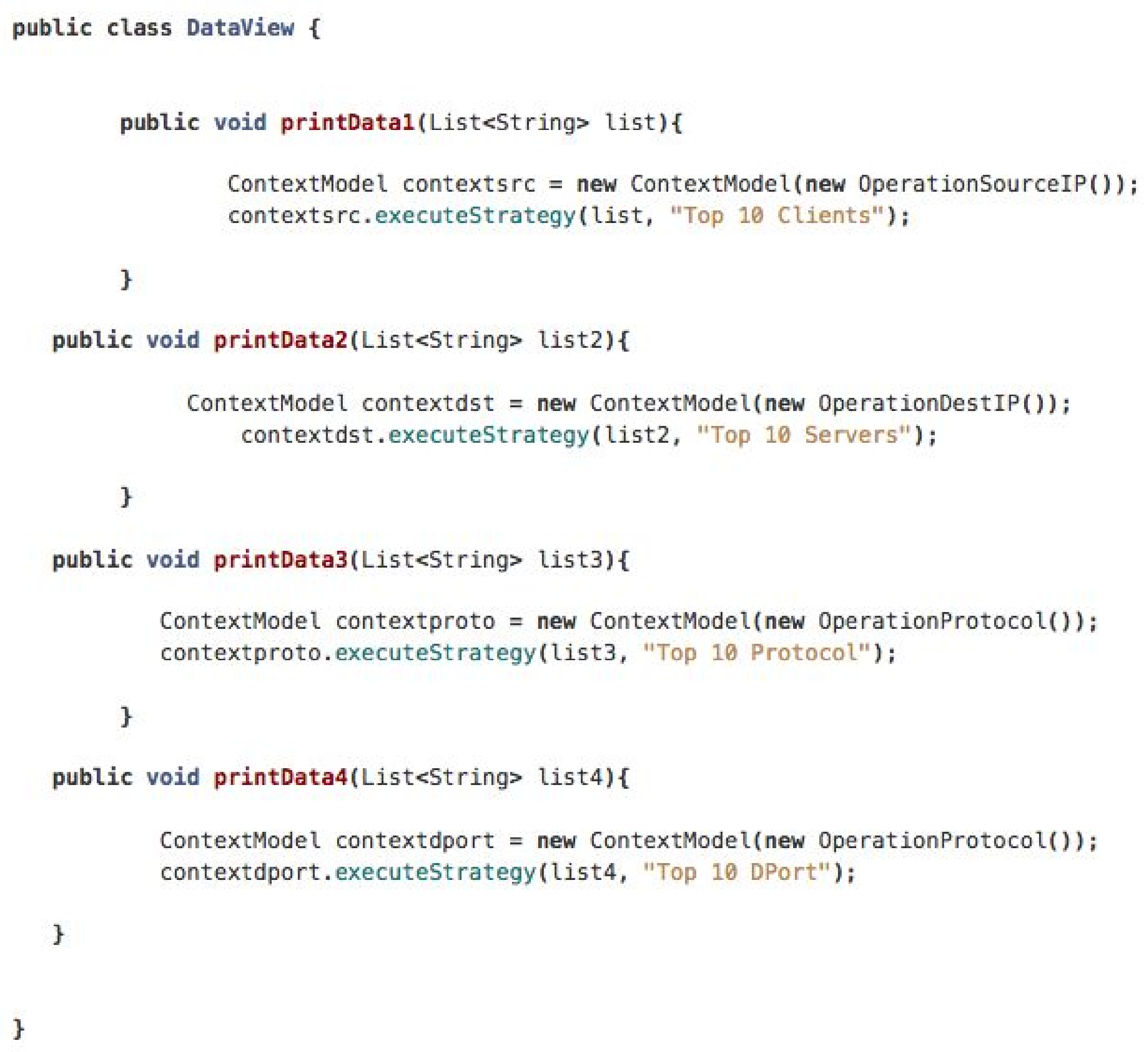
4.1 Figure 1 - UML Diagram



<<Java Class>> DataController application
DataController(NetworkModel,DataView)
setsource(List<String>):void
getsource():List<String>
setdest(List<String>):void
getdest():List<String>
setproto(List<String>):void
getproto():List<String>
setdport(List<String>):void
getdport():List<String>
updateViewSrcIP():void
updateViewDstIP():void
updateViewProto():void
updateViewDPort():void
-view 0..1
-model 0..1
<<Java Class>> Main application
Main()
main(String[]):void
<<Java Class>> JFXMain application
args: String[]
JFXMain()
data(String[]):void
start(Stage):void
<<Java Class>> DataView application
DataView()
printData1(List<String>):void
printData2(List<String>):void
printData3(List<String>):void
printData4(List<String>):void
<<Java Class>> NetworkModel application
source: ArrayList<String>
dest: ArrayList<String>
proto: List<String>
dport: List<String>
NetworkModel()
getsource():ArrayList<String>
setsource(ArrayList<String>):void
getdest():List<String>
setdest(List<String>):void
getproto():List<String>
setproto(List<String>):void
getdport():List<String>
setdport(List<String>):void
<<Java Class>> ReadDataFile application
ReadDataFile()
ReadFile(int):NetworkModel
<<Java Class>> ContextModel application
ContextModel(StrategyCapture)
executeStrategy(List<String>,String):Map<String,Long>
-strategy 0..1
<<Java Interface>> StrategyCapture application
doOperation(List<String>,String):Map<String,Long>
<<Java Class>> FileManager application
fileAccess: File
in: Scanner
fileName: String
FileManager(String)
handleError(Exception):boolean
inSet():boolean
next():String
nextLine():String
nextInt():int
hasNext():boolean
hasNextLine():boolean
hasNextInt():boolean
numOfLines():int
closeFile():void
finalize():void
getScanner():Scanner
<<Java Class>> OperationDestIP application
OperationDestIP()
doOperation(List<String>,String):Map<String,Long>
<<Java Class>> OperationSourceIP application
OperationSourceIP()
doOperation(List<String>,String):Map<String,Long>
<<Java Class>> OperationDport application
OperationDport()
doOperation(List<String>,String):Map<String,Long>
<<Java Class>> OperationProtocol application
OperationProtocol()
doOperation(List<String>,String):Map<String,Long>
<<Java Interface>> ChartShape application
draw(Map<String,Long>,String):void
<<Java Class>> PieConcrete application
PieConcrete()
draw(Map<String,Long>,String):void
<<Java Class>> PieChart application
serialVersionUID: long
PieChart(String,String,Map<String,Lon...
createDataset(Map<String,Long>):Pie...
createChart(PieDataset,String):JFreeC...
<<Java Class>> ChartFactory application
ChartFactory()
getType(String):ChartShape

Figure 1: Presents the UML Diagram

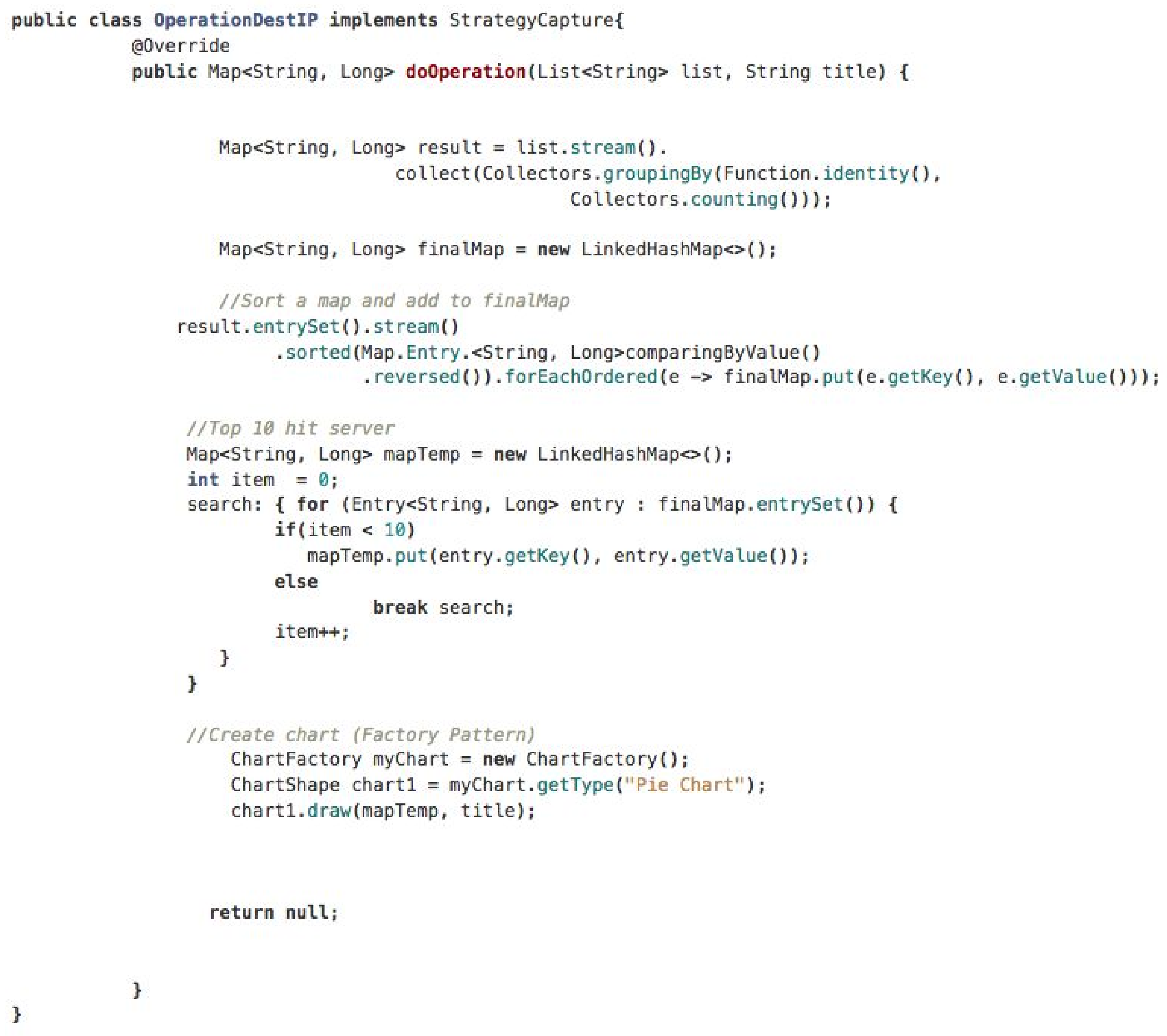
4.2 Code Snippet 1 - DataView.java



```java
public class DataView {


        public void printData1(List<String> list){

                ContextModel contextsrc = new ContextModel(new OperationSourceIP());
                contextsrc.executeStrategy(list, "Top 10 Clients");

        }

   public void printData2(List<String> list2){

               ContextModel contextdst = new ContextModel(new OperationDestIP());
                   contextdst.executeStrategy(list2, "Top 10 Servers");

        }

   public void printData3(List<String> list3){

             ContextModel contextproto = new ContextModel(new OperationProtocol());
             contextproto.executeStrategy(list3, "Top 10 Protocol");

        }

   public void printData4(List<String> list4){

             ContextModel contextdport = new ContextModel(new OperationProtocol());
             contextdport.executeStrategy(list4, "Top 10 DPort");

   }


}
```

Figure 2: Defines the data view strategy

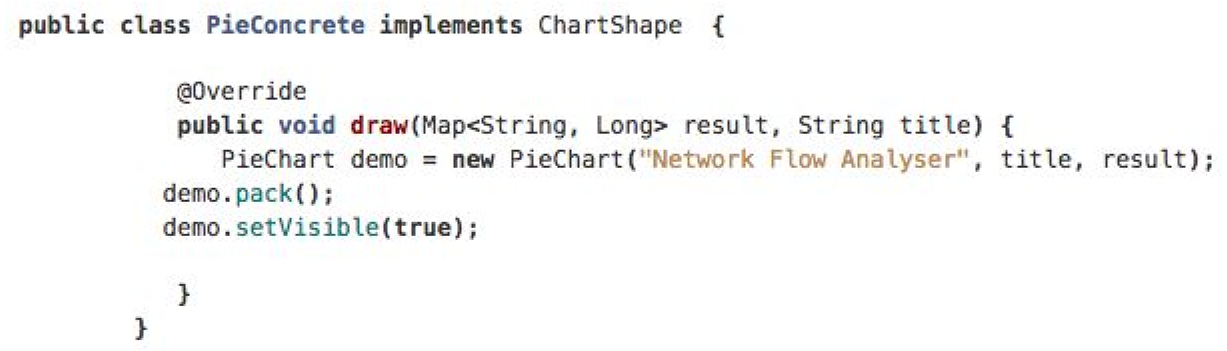
4.3 Code Snippet 2 - OperationDestIP.java



```
public class OperationDestIP implements StrategyCapture{
          @Override
          public Map<String, Long> doOperation(List<String> list, String title) {


                 Map<String, Long> result = list.stream().
                                collect(Collectors.groupingBy(Function.identity(),
                                               Collectors.counting()));

                 Map<String, Long> finalMap = new LinkedHashMap<>();

                 //Sort a map and add to finalMap
              result.entrySet().stream()
                      .sorted(Map.Entry.<String, Long>comparingByValue()
                              .reversed()).forEachOrdered(e -> finalMap.put(e.getKey(), e.getValue()));

               //Top 10 hit server
               Map<String, Long> mapTemp = new LinkedHashMap<>();
               int item  = 0;
               search: { for (Entry<String, Long> entry : finalMap.entrySet()) {
                       if(item < 10)
                          mapTemp.put(entry.getKey(), entry.getValue());
                       else
                                break search;
                       item++;
                 }
               }

               //Create chart (Factory Pattern)
                   ChartFactory myChart = new ChartFactory();
                   ChartShape chart1 = myChart.getType("Pie Chart");
                   chart1.draw(mapTemp, title);



                return null;


          }
}
```

Figure 3: Defines the strategy operation to define the object context on the Strategy Pattern Design.

4.4 Code Snippet 3 - PieConcrete.java



```
public class PieConcrete implements ChartShape  {

          @Override
          public void draw(Map<String, Long> result, String title) {
             PieChart demo = new PieChart("Network Flow Analyser", title, result);
         demo.pack();
         demo.setVisible(true);

          }
       }
```

Figure 4: Defines the implementation of Factory Pattern to generate the charts.