```java
package measure;

import io.MeasurementWriter;
import main.StartUp;
import org.hyperic.sigar.CpuPerc;
import org.hyperic.sigar.Mem;
import org.hyperic.sigar.Sigar;
import org.hyperic.sigar.SigarException;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.concurrent.TimeUnit;

public class Measurer {

    private static final Sigar sigar = new Sigar();
    private static CpuPerc cpuMeasurer;
    private static Mem memMeasurer;

    private static final String BASH_COMMAND = "top -b -n1 | grep %d | awk '{ if ($9 != \"0,0\") print $9 \" \" $10 }'";

    private static double getCpuLoadPercentage() {
        if (cpuMeasurer == null) {
            try {
                cpuMeasurer = sigar.getCpuPerc();
            } catch (SigarException e) {
                e.printStackTrace();
            }
        }
        return cpuMeasurer.getCombined() * 100;
    }

    private static double getMemoryLoadPercentae() {
        if (memMeasurer == null) {
            try {
                memMeasurer = sigar.getMem();
            } catch (SigarException e) {
                e.printStackTrace();
            }
        }
        return memMeasurer.getUsedPercent();
    }

    public static void measureSigar(int ID, Parameter parameter) throws InterruptedException {
        for (int index = 0; index < 50; index++) {
            Measurement measurement = new Measurement(ID, index, parameter, getCpuLoadPercentage(), getMemoryLoadPercentae());
            MeasurementWriter.write(measurement);
            System.out.println(measurement);
            TimeUnit.MILLISECONDS.sleep(10);
        }
    }

    public static void measureTop(int ID, Parameter parameter) throws InterruptedException, IOException {
        int index = 0;
        String[] command = new String[] {"/bin/sh", "-c", String.format(BASH_COMMAND, StartUp.getPID())};
        Process process; BufferedReader reader; String output;
        Runtime runtime = Runtime.getRuntime();
        while (index < 50) {
            process = runtime.exec(command);
            reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
            output = reader.readLine();
            reader.close();
            process.waitFor();
            double cpuLoad = 0;
            try {
                cpuLoad = Float.valueOf(output.replaceAll(",", ".").split(" ")[0]);
                if (cpuLoad > 0) {
                    double memLoad = Float.valueOf(output.replaceAll(",", ".").split(" ")[1]);
                    Measurement measurement = new Measurement(ID, index, parameter, cpuLoad, memLoad);
                    MeasurementWriter.write(measurement);
                    index++;
                }
            } catch (NullPointerException ex) {
                ex.printStackTrace();
            } catch (NumberFormatException ex) {
                ex.printStackTrace();
            }
        }
    }

}
```

```java
package measure;

public class Parameter extends Tuple {

    private int voices;
    private int voicesToEQandComp;
    private int effects;
    private int voicesToEffects;

    public Parameter(int voices, int voicesToEQandComp, int effects, int voicesToEffects) {
        super(voices, voicesToEQandComp, effects, voicesToEffects);
        this.voices = voices;
        this.voicesToEQandComp = voicesToEQandComp;
        this.effects = effects;
        this.voicesToEffects = voicesToEffects;
    }

    public int getVoices() {
        return voices;
    }

    public int getVoicesToEQandComp() {
        return voicesToEQandComp;
    }
```

```
    public int getEffects() {
        return effects;
    }

    public int getVoicesToEffects() {
        return voicesToEffects;
    }

}
```

```
package measure;

public class Tuple {

    private Number[] data;

    public Tuple(Number... data) {
        this.data = data;
    }

    @Override
    public String toString() {
        StringBuilder tupleWriter = new StringBuilder(data[0].toString());
        for (int i = 1; i < data.length; i++) {
            tupleWriter.append(",").append(data[i]);
        }
        return tupleWriter.toString();
    }

}
```

```
package measure;

public class Measurement extends Tuple {

    public Measurement (int ID, int index, Parameter parameter, double cpuLoad, double memLoad) {
        super(ID, index, parameter.getVoices(), parameter.getVoicesToEQandComp(), parameter.getEffects(), parameter.getVoicesToEffects(), cpuLoad, memLoad);
    }

}
```