```
import java.text.*;
import java.util.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import edu.cwru.dipzoom.lib.*;

public class infocomDemo {
  public static void main(String[] args) throws Exception  {
    String region1 = "NY";
    String region2 = "CA";
    if (args.length >= 2){
      region1 = args[0];
      region2 = args[1];
    }
   //login
   DipzoomClientLibrary dipzoomClientLibrary = new
DipzoomClientLibrary();
   dipzoomClientLibrary.login("login.xml");

   // declare ArrayLists for ticket queues
   ArrayList<Measurement> pendingMeasurements = null;
   ArrayList<Measurement> finishedMeasurements = new
ArrayList<Measurement>();
   Measurement resultMeasurement = null;

   Hashtable<String, String> measuringPointParameters = new
Hashtable<String,String>();
   //set region 1, default is NY
   measuringPointParameters.put("region",  region1 );

   //get MPs from region 1 (NY) support nslookup
   measuringPointParameters.put("measurementType",
DipzoomConstants.NSLOOKUP);
   MeasuringPoint[] measuringPoints_Region1_nslookup = null;
   while (measuringPoints_Region1_nslookup == null) {
     try{
       measuringPoints_Region1_nslookup =
dipzoomClientLibrary.getMeasuringPointList(measuringPointParameters);
       pause(2000);
     }catch (Exception e1) {
       e1.printStackTrace();
     }
   }

//get MPs from region 1 (NY) support wget
   measuringPointParameters.put("measurementType",
DipzoomConstants.WGET);
   MeasuringPoint[] measuringPoints_Region1_Wget = null;
   while (measuringPoints_Region1_Wget == null){
     try{
       measuringPoints_Region1_Wget =
dipzoomClientLibrary.getMeasuringPointList(measuringPointParameters);
       pause(2000);
     }catch (Exception e1) {
       e1.printStackTrace();
     }
   }

//get intersection of above MPs that support both wget and nslookup and is from
region 1 (NY)
   ArrayList<MeasuringPoint> measuringPoints_Region1_list = new
ArrayList();
   if (measuringPoints_Region1_Wget != null &&
measuringPoints_Region1_nslookup != null) {
       for (MeasuringPoint mp : measuringPoints_Region1_Wget){
         for (MeasuringPoint mp2 : measuringPoints_Region1_nslookup){
           if (mp2.getId().equals(mp.getId())){
             measuringPoints_Region1_list.add(mp);
             break;
           }
         }
       }
   }

   //set region 2, default is CA
   measuringPointParameters.put("region",  region2 );

   //get MPs from region 2 (CA) support nslookup
   measuringPointParameters.put("measurementType",
DipzoomConstants.NSLOOKUP);
   MeasuringPoint[] measuringPoints_Region2_nslookup = null;
   while (measuringPoints_Region2_nslookup == null) {
     try{
       measuringPoints_Region2_nslookup =
dipzoomClientLibrary.getMeasuringPointList(measuringPointParameters);
       pause(2000);
     }catch (Exception e1) {
       e1.printStackTrace();
     }
   }

   //get MPs from region 2 (CA) support wget
   measuringPointParameters.put("measurementType",
DipzoomConstants.WGET);
   MeasuringPoint[] measuringPoints_Region2_Wget = null;
   while (measuringPoints_Region2_Wget == null){
     try{
       measuringPoints_Region2_Wget =
dipzoomClientLibrary.getMeasuringPointList(measuringPointParameters);
       pause(2000);
     }catch (Exception e1) {
       e1.printStackTrace();
     }
   }

   //get intersection of above MPs that support both wget and nslookup and is from
region 2 (CA)
   ArrayList<MeasuringPoint> measuringPoints_Region2_list = new ArrayList();
   if (measuringPoints_Region2_Wget != null &&
measuringPoints_Region2_nslookup != null) {
       for (MeasuringPoint mp : measuringPoints_Region2_Wget){
         for (MeasuringPoint mp2 : measuringPoints_Region2_nslookup){
           if (mp2.getId().equals(mp.getId())){
             measuringPoints_Region2_list.add(mp);
             break;
           }
         }
       }
   }
   //get one MP from each region that support both wget and nslookup
   MeasuringPoint[] measuringPoints = new
MeasuringPoint[]{measuringPoints_Region1_list.get(0),measuringPoints_Region2
_list.get(0)};

   System.out.println("The returned mp list is:");
   for (MeasuringPoint mp : measuringPoints){
     System.out.println(mp);
   }

   //ask each MP to resolve the Akamai supported domain name
images.pcworld.com
   Measurement measurementRequest = new Measurement();
   Hashtable<String,String> parameters = new Hashtable<String,String>();
   parameters.put("target", "images.pcworld.com");
   parameters.put("type", DipzoomConstants.NSLOOKUP );
   parameters.put("number", "1");
   measurementRequest.setParameters(parameters);


   // send one measuringRequest to all measuringPoints
   for (int i = 0; i<measuringPoints.length; i++){
     System.out.println("Doing "+measurementRequest.getType()+" to
"+measurementRequest.getTarget()+" from "+measuringPoints[i].getHost() + " at
"+measuringPoints[i].getRegion());
   }

   while (pendingMeasurements == null){
     try{
       pendingMeasurements =
dipzoomClientLibrary.sendRequest(measurementRequest, measuringPoints);
       pause(5000);
     }catch (Exception e1) {
       e1.printStackTrace();
     }
   }

   // in case there are outstanding tickets that are ready for decoding
   pendingMeasurements = dipzoomClientLibrary.getTicketStatusAsArrayList();
   System.out.println("Total request size: " + pendingMeasurements.size());


   // decode results as they come in
```

```
    //  note: this loop will not exit until ALL measurements are in
    //        it could just as easily be re-coded to only wait for a threshold of results
    //        or spawn a new thread to deal with each result as it comes in
    while (pendingMeasurements.size() > 0)
    {
                pendingMeasurements =
dipzoomClientLibrary.getTicketStatusAsArrayList();
        pause(15000);
                for (int i = 0; i < pendingMeasurements.size(); i++)
        {
                            // current status of the ticket -- informational purposes
                System.out.println(pendingMeasurements.get(i).getHost() + " - " +
DipzoomClientLibrary.statusCodeToString(Integer.toString(pendingMeasureme
nts.get(i).getTransactionStatus())));
                if (pendingMeasurements.get(i).getTransactionStatus() ==
DipzoomConstants.RESULT_RECEIVED)
                {
                        resultMeasurement = null;
                        while (resultMeasurement == null){
                            try{
                                resultMeasurement =
dipzoomClientLibrary.requestResult(pendingMeasurements.get(i));
                                pause(2000);
                            }catch (Exception e){
                                e.printStackTrace();
                            }
                        }

                finishedMeasurements.add(pendingMeasurements.get(i));
                                                        pendingMeasurements.remove(i);
                }
        }
        System.out.println("Outstanding measurements: " +
pendingMeasurements.size()); // informational purposes
    }

    // analyze the results, get all the IPs of the domain name images.world.com
    String cdnSelectedServerIP = "";
    ArrayList<String> totaliplist = new ArrayList<String>();
    for (Measurement measurement:finishedMeasurements){
        if (!measurement.getType().equalsIgnoreCase("nslookup")
|| !measurement.getTarget().equalsIgnoreCase("images.pcworld.com")) continue;
        NsLookupResult nslookupResult = new NsLookupResult(measurement);
        ArrayList<String> ipList = nslookupResult.gettarget_ipAddrs();
        if (measuringPoints[0].getId().equals(nslookupResult.getId()))
cdnSelectedServerIP = ipList.get(0);
        for (String ip : ipList){
            if (!totaliplist.contains(ip))
                totaliplist.add(ip);
        }
    }
    System.out.println("totalIPs:"+totaliplist.size());
    System.out.println("Returned IP addresses for domain name:
"+parameters.get("target")+ " is "+totaliplist);

    // use one MP to download the target jpeg file using all the returned IP
addresses
    Measurement[] measurementRequests = new Measurement[totaliplist.size()];
    for (int i = 0; i< totaliplist.size(); i++){
        measurementRequests[i] = new Measurement();
        Hashtable<String,String> wget_parameters = new
Hashtable<String,String>();
        wget_parameters.put("target",
totaliplist.get(i)+"/images/header/logo_hd.jpg");
        wget_parameters.put("parameter", "--header Host:images.pcworld.com");
        wget_parameters.put("type", DipzoomConstants.WGET);
        wget_parameters.put("number", "1");
        measurementRequests[i].setParameters(wget_parameters);
    }

    // send all measuringRequests to one measuringPoint
    for (int i = 0; i < measurementRequests.length; i ++){
        System.out.println("Doing "+measurementRequests[i].getType()+" to
"+measurementRequests[i].getTarget()+" from "+measuringPoints[0].getHost()
+ " at "+measuringPoints[0].getRegion());
    }

    pendingMeasurements = null;
    while (pendingMeasurements == null){
        try{
                pendingMeasurements =
dipzoomClientLibrary.sendRequest(measurementRequests, new
MeasuringPoint[]{measuringPoints[0]});
                pause(15000);
            }catch (Exception e1) {
                e1.printStackTrace();
            }
    }
    // in case there are outstanding tickets that are ready for decoding
    pendingMeasurements = dipzoomClientLibrary.getTicketStatusAsArrayList();
    System.out.println("Total request size: " + pendingMeasurements.size());


    // decode results as they come in
    //  note: this loop will not exit until ALL measurements are in
    //        it could just as easily be re-coded to only wait for a threshold of results
    //        or spawn a new thread to deal with each result as it comes in
    finishedMeasurements.clear();
    while (pendingMeasurements.size() > 0)
    {
      pause(15000);
      pendingMeasurements = dipzoomClientLibrary.getTicketStatusAsArrayList();
      for (int i = 0; i < pendingMeasurements.size(); i++)
      {
            // current status of the ticket -- informational purposes
            System.out.println(pendingMeasurements.get(i).getHost() + " - " +
DipzoomClientLibrary.statusCodeToString(Integer.toString(pendingMeasurements.
get(i).getTransactionStatus())));
            if (pendingMeasurements.get(i).getTransactionStatus() ==
DipzoomConstants.RESULT_RECEIVED)
            {
                    resultMeasurement = null;
                    while (resultMeasurement == null){
                        try{
                            resultMeasurement =
dipzoomClientLibrary.requestResult(pendingMeasurements.get(i));
                            pause(2000);
                        }catch (Exception e){
                            e.printStackTrace();
                        }
                    }
                    finishedMeasurements.add(pendingMeasurements.get(i));
                    pendingMeasurements.remove(i);
            }
      }
      System.out.println("Outstanding measurements: " +
pendingMeasurements.size()); // informational purposes
    }
    WgetResult[] wgetResults = new WgetResult[finishedMeasurements.size()];
    for (int i = 0; i < finishedMeasurements.size(); i++){
        wgetResults[i] = new WgetResult(finishedMeasurements.get(i));
        System.out.println("The CompletionTime and download speed for
"+wgetResults[i].getType() + " "+ wgetResults[i].getTarget() + " is
"+wgetResults[i].getCompletionTime()/1000 + " seconds and
"+wgetResults[i].getDownloadRate()/1000 + "KB/s");
    }
    // findFastestServer and findCDNSelectedServer are user-defined functions,
    // They are not part of DipZoom API.
    WgetResult fastestServer = findFastestServer(wgetResults);
    System.out.printf("The fastest server to reach images.pcworld.com is %s with a
time of %d seconds and downloadrate of %f KB/s.\n", fastestServer.getTargetIP(),
fastestServer.getCompletionTime()/1000, fastestServer.getDownloadRate()/1000);
    WgetResult cdnSelectedServer=
findCDNSelectedServer(wgetResults,cdnSelectedServerIP);
    System.out.printf("The CDN server to reach images.pcworld.com is %s with a
time of %d seconds and downloadrate of %f KB/s.\n",
cdnSelectedServer.getTargetIP(),cdnSelectedServer.getCompletionTime()/1000,
cdnSelectedServer.getDownloadRate()/1000);

    try {
            dipzoomClientLibrary.logout();
    } catch (Exception de) {
            de.printStackTrace();
            System.err.println("Unable to logout from core");
            System.exit(-1);
    }
    System.exit(0);
  }
  public static boolean pause(int milli)
  {
        try {
                Thread.sleep(milli);
                return true;
            } catch (InterruptedException e) {
                e.printStackTrace();
                return false;
            }

    }

    /**
     * findCDNSelectedServer
     *
     * @param wgetResults WgetResult[]
     * @param cdnSelectedServerIP String
     * @return WgetResult
     */
     public static WgetResult findCDNSelectedServer(WgetResult[] wgetResults,
             String cdnSelectedServerIP) {
         if (wgetResults.length == 0 ) return null;
         for (int i = 0; i<wgetResults.length; i++){
             if (wgetResults[i].getTargetIP().equals(cdnSelectedServerIP))
                 return wgetResults[i];
         }
         return null;
     }

    /**
     * findFastestServer
     *
     * @param wgetResults WgetResult[]
     * @return WgetResult
     */
     public static WgetResult findFastestServer(WgetResult[] wgetResults) {
         if (wgetResults.length == 0 ) return null;
         WgetResult fastestServer = wgetResults[0];
         for (int i = 1; i<wgetResults.length; i++){
             if (fastestServer.getDownloadRate() <
wgetResults[i].getDownloadRate())
                 fastestServer = wgetResults[i];
         }
         return fastestServer;
     }

}

class NsLookupResult extends Measurement {
    private String dnsServerAddr = "";
    private ArrayList<String> target_ipAddrs = new ArrayList<String>();
    private ArrayList<String> target_Hostnames = new ArrayList<String>();
    private boolean resultParsed = false;
    public String getDnsServerAddr() {
        if (!resultParsed) parseResult();
        return dnsServerAddr;
    }

    public ArrayList<String> gettarget_ipAddrs() {
        if (!resultParsed) parseResult();
        return target_ipAddrs;
    }

    public ArrayList<String> gettarget_Hostnames() {
        if (!resultParsed) parseResult();
        return target_Hostnames;
    }

    /**
     * isIPAddress
     *
     * @param input String
     * @return boolean
     */
    private boolean isIPAddress(String input) {
        Pattern targetpattern = Pattern.compile("(^(?:\\d{1,3}\\.){3}\\d{1,3}$)");
        Matcher matcher = targetpattern.matcher(input);
        return matcher.find();
    }

    /**
     * parseResult
     */
    private void parseResult() {
        String result = this.getResult();
        if (result == null) return;
        resultParsed = true;
        String[] lines = result.split("\n");
        boolean named = false;
        for (int i = 0; i< lines.length; i++){
            String line = lines[i];

            if (line.startsWith("Server:")){
                String IPAddr = line.substring("Server:".length()).trim();
                if (isIPAddress(IPAddr)) {
                    dnsServerAddr = IPAddr;
                }
                continue;
            }

            if (line.startsWith("Name:")){
                named = true;
                String hostName = line.substring("Name:".length()).trim();
                if (!target_Hostnames.contains(hostName))
target_Hostnames.add(hostName);
                continue;
            }
            if (line.startsWith("Aliases:")){
                String AliasList = line.substring("Aliases:".length()).trim();
                String Aliases[] = AliasList.split(",");
                for (int j = 0; j < Aliases.length; j++){
                    String Alias = Aliases[j];
                    if (!target_Hostnames.contains(Alias)) target_Hostnames.add(Alias);
                }
                continue;
            }
            if (line.startsWith("Address:")){
                String IPAddr = line.substring("Address:".length()).trim();
                if (isIPAddress(IPAddr)) {
                    if (named) target_ipAddrs.add(IPAddr);
                    else {
                        dnsServerAddr = IPAddr;
                    }
                }
                continue;
            }

            if (line.startsWith("Addresses:")){
                String IPAddrList = line.substring("Addresses:".length()).trim();
                String IPAddrs[] = IPAddrList.split(",");
                for (int j = 0; j < IPAddrs.length; j++){
                    String IPAddr = IPAddrs[j].trim();
                    if (isIPAddress(IPAddr)) target_ipAddrs.add(IPAddr);
                }
                continue;
            }

            if (line.contains("name = ")){
                String dnsName = line.substring(line.indexOf("name = ")+"name =
".length()).trim();
                if (!target_Hostnames.contains(dnsName))
target_Hostnames.add(dnsName);
                if (isIPAddress(this.getTarget())) target_ipAddrs.add(this.getTarget());
            }
        }
    }

    /**
     * NsLookupResult
     *
     * @param measurement Measurement
     */
    public NsLookupResult(Measurement measurement) {
    this.setNonce(measurement.getNonce());
    this.setId(measurement.getId());
    this.setIp(measurement.getIp());
    this.setHost(measurement.getHost());
    this.setType(measurement.getType());
    this.setTarget(measurement.getTarget());
    this.setNumber(measurement.getNumber());
    this.setParameter(measurement.getParameter());
    this.setResult(measurement.getResult());
    this.setResultFilename(measurement.getResultFilename());
    this.setRequestedTimestamp(measurement.getRequestedTimestamp());
    this.setTransactionStatus(measurement.getTransactionStatus());
    if (this.getResult() != null) this.parseResult();

    }
}
class WgetResult extends Measurement {
    /**
     * WgetResult
     */
    public WgetResult() {
    }

    /**
     * WgetResult
     *
     * @param measurement Measurement
     */
    public WgetResult(Measurement measurement) {
        this.setNonce(measurement.getNonce());
        this.setId(measurement.getId());
        this.setIp(measurement.getIp());
        this.setHost(measurement.getHost());
        this.setType(measurement.getType());
        this.setTarget(measurement.getTarget());
        this.setNumber(measurement.getNumber());
        this.setParameter(measurement.getParameter());
        this.setResult(measurement.getResult());
        this.setResultFilename(measurement.getResultFilename());
        this.setRequestedTimestamp(measurement.getRequestedTimestamp());
        this.setTransactionStatus(measurement.getTransactionStatus());
        if (this.getResult() != null) this.parseResult();
    }

    private int length;
    private double downloadRate;
    private long completionTime;
    private long startTime;

    private boolean resultParsed = false;
    /**
     * parseResult
     */
    private void parseResult() {
        String result = this.getResult();
        if (result == null) return;
        resultParsed = true;
        String[] lines = result.split("\n");
        DateFormat date_formatter = new SimpleDateFormat("hh:mm:ss");
        for (int i = 0; i< lines.length; i++){
            String line = lines[i].trim();
            if (line.startsWith("--") ){
                if (startTime != 0) continue;
                String[] segments = line.split(" ");
                String starttimeStr = segments[0].substring("--
".length(),segments[0].lastIndexOf("--"));
                try{
                    startTime = ((Date) date_formatter.parse(starttimeStr)).getTime();
                }catch (ParseException e) {
                }
                continue;
            }
            if (line.startsWith("Connecting to")){
                String[] segments = line.split(" ");
                String hostipStr = segments[2].substring(0,segments[2].indexOf(":"));
                if (hostipStr.contains("[")){
                    int start = hostipStr.indexOf("[");
                    int end = hostipStr.indexOf("]");
                    targetHost = hostipStr.substring(0,start);
                    targetIP = hostipStr.substring(start+1,end);
                }
                else {
                    if (hostipStr.contains("|")){
                        int start = hostipStr.indexOf("|");
                        int end = hostipStr.lastIndexOf("|");
                        targetHost = hostipStr.substring(0,start);
                        targetIP = hostipStr.substring(start+1,end);
                    }
                    else{
                        if (isIPAddress(hostipStr)){
                            targetHost = hostipStr;
                            targetIP = hostipStr;
                        }
                    }
                }
                continue;
            }
            if (line.startsWith("Resolving"))
                continue;
            if (line.startsWith("Location") || line.contains("request sent"))
                continue;
            if (line.startsWith("Length:")){
                NumberFormat number_formatter = NumberFormat.getInstance();
                String[] segments = line.split(" ");
                try{
                    length = number_formatter.parse(segments[1]).intValue();
                }catch (ParseException e) {
                }
                continue;
            }
            if (line.contains("saved")){
                String[] segments = line.split(" ");
                try{
                    finishTime = ((Date) date_formatter.parse(segments[0])).getTime();
                }catch (ParseException e) {
                }
                completionTime = finishTime - startTime;
                downloadRate = Double.parseDouble(segments[1].substring(1));
                if (segments[2].contains("KB/s")) downloadRate *= 1000;
                else if (segments[2].contains("MB/s")) downloadRate *= 1000000;
            }
        }
    }

    /**
     * isIPAddress
     *
     * @param input String
     * @return boolean
     */
    private boolean isIPAddress(String input) {
        Pattern targetpattern = Pattern.compile("(^(?:\\d{1,3}\\.){3}\\d{1,3}$)");
        Matcher matcher = targetpattern.matcher(input);
        return matcher.find();
    }

    private long finishTime;
    private String targetHost;
    private String targetIP;

    public int getLength() {
        return length;
    }

    public double getDownloadRate() {
        return downloadRate;
    }

    public long getCompletionTime() {
        return completionTime;
    }

    public long getStartTime() {
        return startTime;
    }

    public long getFinishTime() {
        return finishTime;
    }

    public String getTargetHost() {
        return targetHost;
    }

    public String getTargetIP() {
        return targetIP;
    }
}
```