```
package com.thinking.machines.nafserver.tool;
import com.thinking.machines.nafserver.model.*;
import com.thinking.machines.nafserver.annotation.*;
import java.lang.reflect.*;
import java.util.*;
import java.io.*;
import java.net.*;
import java.util.zip.*;
public class ApplicationUtility
{
private static Application application=null;
private ApplicationUtility(){}
public static Application getApplication()
{
if(application!=null) return application;
String mainPackage=null;
try
{
throw new RuntimeException();
}catch(RuntimeException re)
{
StackTraceElement e[]=re.getStackTrace();
String className=e[e.length-1].getClassName();
try
{
mainPackage=Class.forName(className).getPackage().getName();
}catch(ClassNotFoundException cnfe)
{
System.out.println("************ SERIOUS PRROBLEM ***********");
System.exit(0);
}
}
HashMap<String,Service> services=new HashMap<>();
LinkedList<ModuleMistake> moduleMistakes=new LinkedList<>();
String packageToAnalyze=mainPackage;
try
{
URLClassLoader ucl=(URLClassLoader)ClassLoader.getSystemClassLoader();
URL urls[]=ucl.getURLs();
String classPathEntry;
ZipInputStream zis;
ZipEntry ze;
String zipEntryName;
String packageName;
String className;
int dotPosition;
String folderName;
File directory;
File files[];
```

```
String fileName;
for(URL u:urls)
{
classPathEntry=u.getFile();
if(classPathEntry.endsWith(".jar"))
{
// code to analyze jar file contents
zis=new ZipInputStream(u.openStream());
ze=zis.getNextEntry();
while(ze!=null)
{
zipEntryName=ze.getName();
if(zipEntryName.endsWith(".class"))
{
zipEntryName=zipEntryName.replaceAll("\\\\","\\.");
zipEntryName=zipEntryName.replaceAll("/","\\.");
dotPosition=zipEntryName.lastIndexOf(".",zipEntryName.length()-7);
if(dotPosition==-1)
{
packageName="";
className=zipEntryName;
}
else
{
packageName=zipEntryName.substring(0,dotPosition);
className=zipEntryName.substring(dotPosition+1);
}
//1 System.out.println(zipEntryName);
//1 System.out.println(packageName);
//1 System.out.println(className);
if(packageName.startsWith(packageToAnalyze))
{
//2 System.out.println(zipEntryName);
try
{
Class ccc=Class.forName(zipEntryName.substring(0,zipEntryName.length()-6));
moduleScanner(ccc,services,moduleMistakes);
}catch(Throwable ee)
{
System.out.println(ee); // remove after testing
}
}
}
ze=zis.getNextEntry();
}
}
else
{
// code to analyze folder
```

```
folderName=classPathEntry+packageToAnalyze;
if(File.separator.equals("\\\\"))
{
folderName=folderName.replaceAll("\\.","\\\\");
}
else
{
folderName=folderName.replaceAll("\\.","/");
}
directory=new File(folderName);
if(directory.exists()==false) continue;
Stack<File> stack=new Stack<>();
stack.push(directory);
File fifi;
while(stack.size()>0)
{
fifi=stack.pop();
files=fifi.listFiles();
for(File file:files)
{
if(file.isDirectory())
{
stack.push(file);
continue;
}
if(file.getName().endsWith(".class"))
{
className=file.getName();
packageName=file.getAbsolutePath().substring(classPathEntry.length()-1);
packageName=packageName.substring(0,packageName.length()-className.length()-1);
packageName=packageName.replaceAll("\\\\","\\.");
packageName=packageName.replaceAll("/","\\.");
// 3 System.out.println("Package : "+packageName);
// 3 System.out.println("Class name : "+className);
try
{
Class ccc=Class.forName(packageName+"."+className.substring(0,className.length()-6));
moduleScanner(ccc,services,moduleMistakes);
}catch(Throwable ee)
{
System.out.println(ee); // remove after testing
}
}
}
} // stack.size()>0
}
}
}catch(Exception e)
{
```

```
StackTraceElement tt[]=e.getStackTrace();
for(StackTraceElement t:tt)
{
System.out.println(t);
}
}
// class path analysis for classes belonging to main package as well as its sub packages ends here
for(Class entry:classes)
{
System.out.println(entry.getName());
}
application=new Application();
return application;
}
private static void moduleScanner(Class ccc,HashMap<String,Service>
services,LinkedList<ModuleMistake> moduleMistakes)
{
Class pathClass=Path.class;
Path modulePath=null;
String modulePathString=null;
if(ccc.isAnnotationPresent(pathClass))
{
modulePath=(Path)ccc.getAnnotation(pathClass);
modulePathString=modulePath.value();
if(isValidPath(pathString)==false)
{
// ?????????
}
}
Method methods[]=ccc.getDeclaredMethods();
Path methodPath=null;
String methodPathString=null;
for(Method m:methods)
{
if(m.isAnnotationPresent(pathClass))
{
methodPath=(Path)m.getAnnotation(pathClass);
methodPathString=methodPath.value();
if(!isValidPath(methodPathString))
{
?????
}
if(!Modifier.isPublic(m.getModifiers()))
{
?????????
}
if(modulePath==null)
{
??????
```

```
}
servicePath=modulePath+methodPath

}
}
}
private static boolean isValidPath(String path)
{
return true;
}
}
```