GGF International Summer School on Grid Computing 2005 (ISSGC05)

Day 2 Exercise – Using established Web Services

1 Introduction to the Exercise

In this exercise you will develop a number of Java programs to generate and display samples of a surface. You will then use similar programs that have been set up and made available as web services by writing a *client* program that invokes those services. This should give you experience of the detailed mechanisms that enable web service identification and invocation.

Much of the work will involve you editing code that has been provided for you, possibly only in skeleton form. There is an appendix listing all files that have been provided for this purpose, and also showing the directory structures that you will need to set up for the exercises. The code itself is available on the web page below, along with the resources needed for the "Introduction to Java" exercises and a "Quick Guide to GnuPlot".

http://www.gs.unina.it/repository/tuesday-12

2 Learning Goals

When you have completed this exercise, you should have:

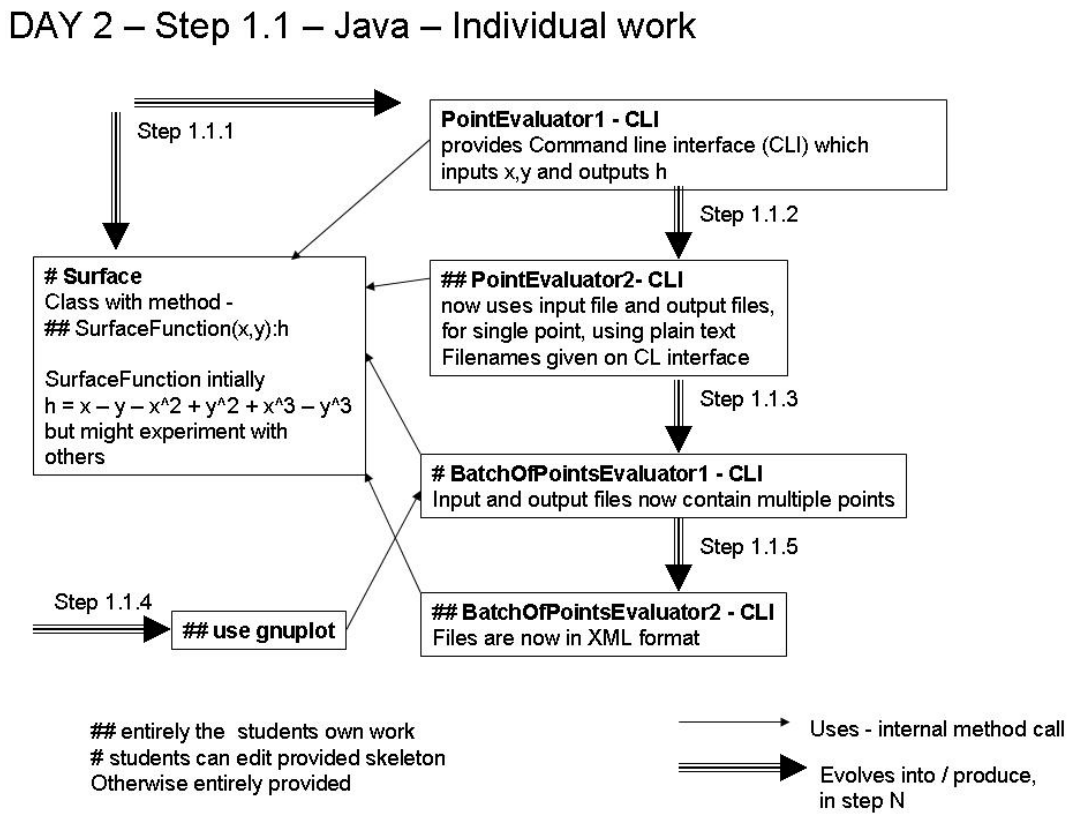
1. Increased ability with Java programming (for those who are not already Java programmers)
2. An initial understanding of the kind of programs and algorithms that will be used throughout the extended exercise
3. Several programs to be used later
4. Familiarity with a simple plotting / visualization tool to be used in later exercises
5. A preliminary experience of using a web service from a client program.

3 Stages in the Exercise

The exercise is presented as a series of stages so that you can incrementally learn about the simulated application scenario and the use of web services. You should work individually until the end of step 3.1, with then step 3.2 being done as a team.

.

Step 1.1: Develop a program to compute a set of points



The overall flow of step 1.1 is shown in the diagram.

You are going to develop a series of programs that compute `h` where

```
h = f( x, y )
```

The result, `h`, is the height of the surface at the given x-y point.

We will use terminology: "`1D`" to mean a single x or y value; "`2D`" to mean an x-y pair; "`3D`" to mean an x-y pair and its associated h value; "box" to mean a rectangular region of a surface, `((xlow, ylow), (xhigh, yhigh))`

Initially we recommend you use a simple function, `surfaceFunction,` where `surfaceFunction` is defined by the code:

```
public double surfaceFunction( double x, double y ) {
  return x – y – x^2 + y^2 + x^3 – y^3;
}
```

This function is defined within a class `Surface.` The function is to be defined within a specified bounding box and will only be evaluated for 2D points within the boundary. This can be implemented using the class `Box` provided:

```
Surface s = new Surface();
double xlow = -10 ; double ylow = -10;
double xhigh = 10; double yhigh = 10;
Box b = new Box(xlow,ylow,xhigh, yhigh);
if(!b.inBox(x,y)) throw new BoxException("Out of bounds "+x+" "+y);
double h = s.surfaceFunction(x,y);
```

Step 1.1.1: Using a point from the command line

Download `Surface.java`, `PointEvaluator1.java`, `Box.java` and `BoxException.java` from the student resource page.

Edit the provided class `Surface.java` to include the specified function. The provided program `PointEvaluator1.java` implements a command line interface: to input a 2D

point; use `surfaceFunction` to generate the height of the surface within a specified boundary as given in the code above; output that height . Compile all code and run the program several times to verify that it works. If there is any doubt, use a very simple version of `surfaceFunction` first, e.g. `h = 2.0`; then `h = y;` and then `h = 0.5*x`.

Use enough samples to convince yourself that your surface is working correctly. Explore what happens when you request a point outside the bounding box.

Step 1.1.2: Using a point from a file

This part of the exercises is to enable you to experience handling files in Java.

Modify the program `PointEvaluator1.java` to produce a program `PointEvaluator2.java` which reads a 2D point from an input file, `file2D`, and writes the corresponding 3D point to the file `file3D`. The files are text files with each line comprising either two real numbers, `x` and `y`, for `file2D`, and three real numbers, `x, y` and `h`, for `file3D`. The actual file names are given as parameters on the command line. A file name can only contain aphanumerics, ".", "-" and "_". Test this program as before.

Note that there are now additional errors due to the properties of files and file names. You will need to handle files from Java in later exercises. It is therefore sensible to develop or acquire code that handles these file related errors, see the "Introduction To Java" document for examples.

Step 1.1.3: Handling batches of points

Now modify your tested `PointEvaluator2.java` to produce a program `BatchOfPointsEvaluator1.java.` You may also start from the provided skeleton version of `BatchOfPointsEvaluator1.java.` This should take an input file `file2D` that has up to `maximumBatchSize` 2D points in it. It should evaluate `surfaceFunction` for each of the points in `file2D` and write a corresponding point in `file3D`. Use `maximumBatchSize` = 20 in your final version of this program. You can decide whether to return any results if the requested sample is too large, but you should not generate more than `maximumBatchSize` samples.

Step 1.1.4: Visualising the Surface

The data in `file3D` should now be visualised using the visualisation tool `gnuPlot and GhostView.` If you are unfamiliar with gnuPlot please consult the "GnuPlot Guide" document. `GhostView` can be invoked on the command line using,

```
gv  file.eps
```

where `file.eps` is a postscript file.

Step 1.1.5: Visualising the Surface

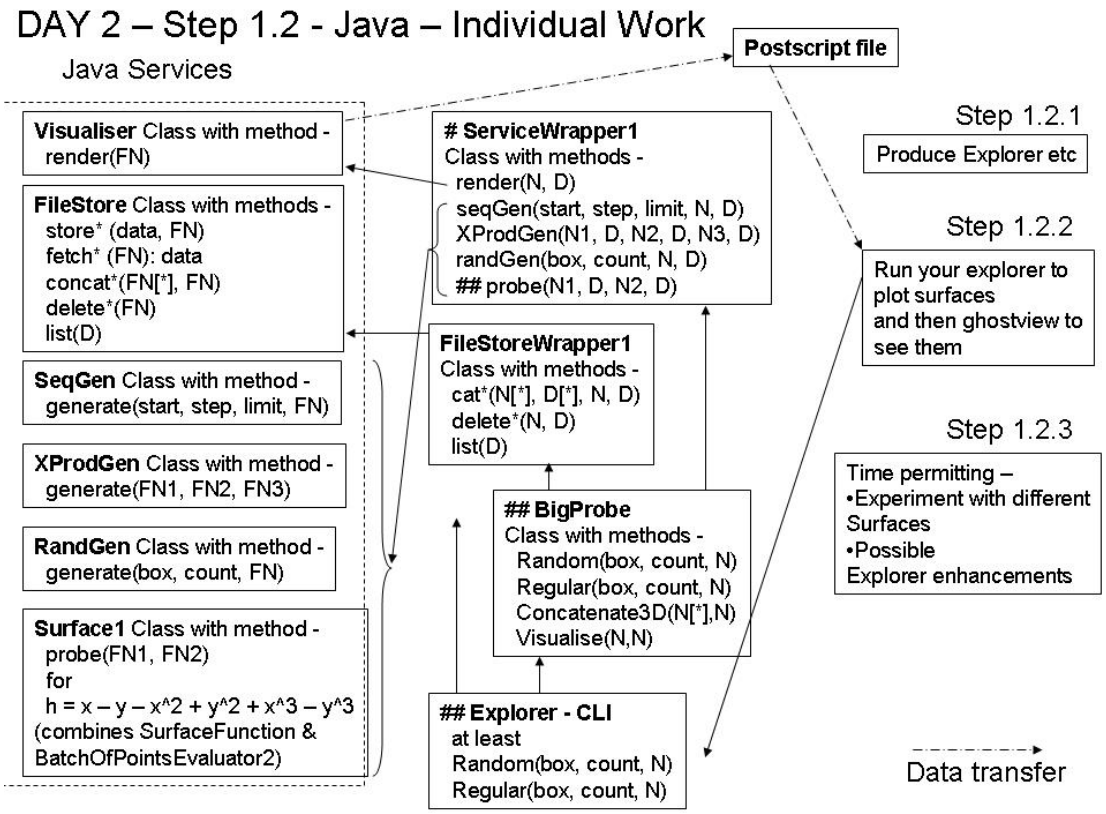
Once you are confident the program is thoroughly tested modify `BatchOfPointsEvaluator1.java` so that the 3D points are generated in XML format (because that is the format to be used in the latter Web Services exercises). Call this program `BatchOfPointsEvaluator2.java.` The xml should have the format,

```
<?xml version="1.0"?>
<ThreeDFile>
<Triple><p><x>1.0</x><y>2.0</y></p><h>-11.0</h></Triple>
</ThreeDFile>
```

and can be produced "by hand", for example, with the code

```
PrintWriter pw = new PrintWriter(new FileWriter("file3D.xml"));
pw.println("<?xml version=\"1.0\"?> \n <ThreeDFile>\n<Triple>");
pw.println("<p><x>"+x+"</x><y>"+y+"</y></p><h>"+h+"</h>");
pw.println("</Triple></ThreeDFile>");
pw.close();
```

Step 1.2: Sampling a surface



In the following you will sample a surface in a more systematic way using java code which the web services in step 1.7 are based on. This will allow you to become familiar with the classes and methods provided.

The diagram illustrates the software provided and the suggested structure that you should employ for using it.

Step 1.2: The Explorer Program

The Explorer program is to provide with a command line interface (CLI) as a tool for exploring and visualising a surface. In the following days you will develop this tool to incorporate use of additional facilities that you will learn about, and possibly incorporate more sophisticated surface exploration tactics.

The explorer program should make use of a number of provided classes (here we use the parameter names as shown in the above diagrams, which may be different from those in the Java code) –

- **`FileStore`** – handles storing and retrieving data in named files. Each file is either a 1D file, a 2D file or a 3D file, containing respectively a number of 1D, 2D or 3D points. The operations of this class and other classes use parameters – `FN`: this is an object containing a file name N and a directory name D**.** The directory name is the subdirectory in FileStore where the data is stored. The subdirectories are automatically created by FileStore and do not have to exist beforehand. **Each student should use a single directory, with name being their login name (so that it is unique).** There are three types of FN – FileName1D, FileName2D and FileName3D, for FileName3D for 1D, 2D and 3D data respectively. There is no logical difference between these filename classes, however, it is useful to make the information the filename represents explicit. The operations of `FileStore` are `Store`, `Fetch`, `Concatenate` and `Delete`, each with a different version for the different types of file. `Concatenate` takes an array of `FN[*]` for its input files and a single `FN` for its output file which will be the

concatenation of its input files. You should adopt a wite-once policy for files, i.e. once you hae dreated a file with data you should not change its data.

- **`Surface1`** – the probe method samples the surface defined by the surfaceFunction in step 1. It takes the name of an input file (`FN1`) and an output file (`FN2`). The input file should contain a number (`M`) of 2D points and the result is the same number of 3D points in the output file. There a maximum for M, `maximumBatchSize = 20`.
- **`SeqGen`** – this has a single generate operation with parameters- start, step, limit, FN. It produces a 1D file as filename N as contained in `FN`. This contains the numbers `start, start + step, start + 2*step, …..` up to the largest which is less than or equal to `limit`.
- **`RandGen`** – this has a generate operation with parameters – `box, count, FN`. It produces a 2D file as filename `N` in directory `D` as contained in FN. This contains `count` 2D points randomly selected from within the area defined by `box`.
- **`XProdGen`** - this has a generate operation with parameters – `FN1, FN2, FN3. F1` and `F2` are names of two 1D files. If `F1` has points `x1 … xM1` and `F2` has points `y1 … yM2` then the result is the set of `M1*M2` 2D points `(x1,y1) … (xM1,yM2)`, which are stored in the file named `FN3`.
- **`Visualiser`** – this invokes `GnuPlot` via its .method `render`. That has a parameter `FN` identifying a 3D file of points to be plotted. Its output is a postscript file.

In this stage we suggest that the Explorer provide two commands

- **`Random`**, with inputs:
  - `Box – xlow, ylow, xhigh, yhigh`. These four input numbers define an area on the surface to be sampled
  - `Count` – the number of sample points
  - `N` – the name of the file to be used for the file of results – a 3D file

  This will do random sampling of the surface, using `RandGen` class to produce the input sample for `Surface1`. The output file of `Surface1` is then passed to `Visualiser`. The issue is that `Count` will typically be bigger than `maximumBatchSize` for the surface, so there will have to be several invocations of probe, with the results combined using file concatenation.
- **`Regular`**. This has the same inputs as `Random` and similar functionality. The difference is that instead of using a random set of sample points, it uses an equally-spaced mesh of sample points, which it should generate using `SeqGen` and `XProdGen`. As you will see, this is harder to do than randomly generating the sample points, so we suggest that you get `Random` working before you attempt `Regular`. **Hint**: For `Regular`, it may be easier if you interpret `Count` as the minimum/target number of samples required, and allow an algorithm that may produce a somewhat different number of samples.

We strongly recommend structuring the explore program along the following lines, using additional classes which have been partially provided –

- **`BigProbe`**. This has the same Random/Regular interface as the explorer command line, but with count being such that it can achieve its result with one call of probe – i.e. within the maxSampleSize of the Surface. The sampling strategy adopted by Explorer will be implemented by a number of calls to BigProbe. The name of the directory you are using in the FileStore is specified in this class (and hence does not have to be passed as a parameter in the methods of BigProbe). So that all calls to ServiceWrapper and FileStoreWrapper methods use the same directory in FileStore there are also methods in BigProbe for concatenating and visualising the 3D data.
- **`ServiceWrapper / FileStoreWrapper`**

In subsequent exercises the roles of the `Visualiser`, `FileStore` etc classes will be implemented by web services, and then grid services. These wrappers are to isolate to one place the changes need for that. Due to the fact that the storing and fetching of data is done in the services RandGen, SeqGen, XProdGen, Surface1, Visualiser etc there is no need for the FileStoreWrapper to provide store and fetch methods. The wrapper classes take as parameters strings specifying the filename (N) and the directory (D). Within the methods the appropriate FN type instance is created (FileName1D, FileName2D, FileName3D) using N and D which is then passed as parameters to the Java Services.

The Java Services classes, for RandGen, SeqGen, XProdGen, Visualiser, Surface1 and FileStore, are provided as a jar file, `school.jar,` containing the compiled classes; the source code is not visible. Javadoc for the classes is provided on the web page mentioned in the introduction. In order to use the services there must be a directory named **filestore** and another named **visualiser** in your home directory, for use by the FileStore and Visualiser Java services respectively. The directory for the Visualiser is only for the creation of temporary files, the postscript output is not stored there.

The client programs which use these classes can be compiled using the `classpath` option, for example:

```
javac –classpath school.jar:. Random.java BigProbe.java
FileStoreWrapper.java ServiceWrapper.java
```

where `school.jar` must be in the same directory as `Random.java` etc. The classpath option tells the java compiler where to find user class files. The "`.`" must also be included so that the class defined within `BigProbe` is found by the compiler. Note that the classpath option must also be used when running the programs, i.e. use

```
java –classpath school.jar:. Random
```

when running `Random`.

Step 1.2.2 Plot a sampled surface

Record the time that it takes you to perform this step and the times that the jobs you run take to execute. You will need these times during Thursday's exercise.

By invoking your `Explorer` CLI obtain 1,000 samples of the surface. Visualise the resulting file for 3D plotting / viewing. Does it have the form that you expected? E.g. There is a valley of height zero on the `x=y` diagonal.

Notice that the construction of the samples, the multiple executions, the concatenation, reformatting and visualisation can be thought of as a directed acyclic graph (DAG) – the exercise on Thursday will present an alternative way of evaluating this DAG.

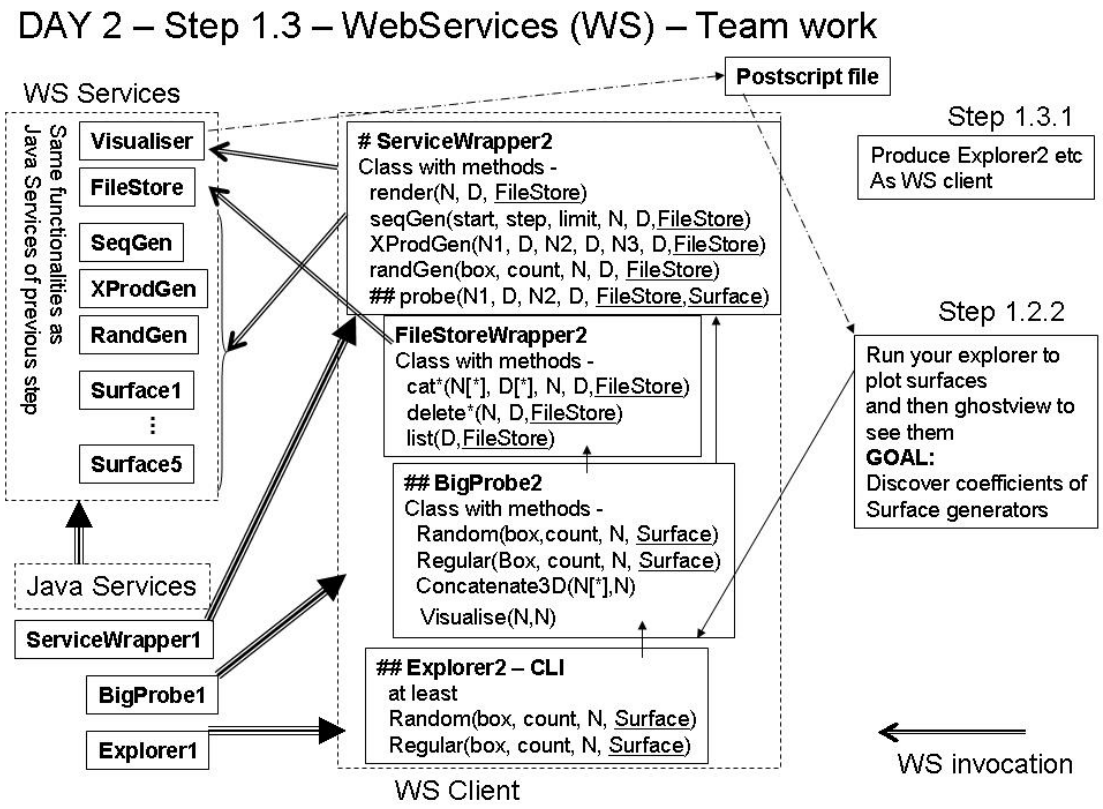
Step 1.2.3

If you have time you can over-ride the surface function defined in `Surface1` using your `Surface` class in step 1.1.1. Modify `Surface` in the following way

```
import school.*;
public class Surface extends Surface1{
    public double surfaceFunction(double x, double y){}
}
```

The `surface1` object instantiated in `BigProbe` method `sampleSurface`, above, can now be replaced by a `Surface` object. Re-run the program `SampleSurface` and check the surface function has changed. You might also consider whether there are improvements you might make to your Explorer program.

Step 1.3 Write a web service client to sample and visualise surfaces



In this exercise you will complete a client Java program that uses several web services in order to sample and visualise a surface. The diagram shows how this step relates to step 1.2.

For comparison purposes note how long the parts of this step take (time to understand and time to implement) and how long programs take to execute. You will, for example, compare this with time to do similar tasks using GT4 on day 5.

Step 1.3.1 Converting Explorer to Web Services

Whereas in step 1.2 we had "Java Services" for use by `Explorer`, these have now become remotely accessed Web Services. The step is to convert your Explorer program to use these, this involves,

- Using a web browser go to the URLs for the services, listed below and view the WSDL for them. You should check that you understand at least the WSDL for one of the Surface services (they are all the same), and one of the others.
- Changing the code in the ServiceWrapper and FileStoreWrapper. A modified ServiceWrapper is provided with only the probe method left for you to complete. The parameter list for this method has changed to include the URL of the Surface service (**Surface** in the figure) as there are now multiple surfaces to sample. The URL of the FileStore service (**FileStore**) is also needed. Within the ServiceWrapper methods the FileStore URL is wrapped up with the FN type (FileName1D etc) into a new URLFN type (URLFileName1D, URLFileName2D, URLFileName3D). Instances of URLFN are passed to the WS services to enable these services to fetch/store the data by contacting the FileStore web service. The FileStoreWrapper is provided, fully modified. Look in the methods of this class to see how to change the endpoint address of the web service that is invoked. You will need to know this in order to make probe switch to different Surface services using **Surface.**
- Changing the code in BigProbe. Only minor modifications to your previous version of this class should be necessary. The parameter list for the Random and Regular methods

should be expanded to include **Surface.** In addition the URL of the FileStore service to be used needs to be set in this class (in the same way as sub-directory D).

- Changing the code in Explorer. The only modification here is to take the Surface URL as input and to include this in the BigProbe method calls.

The WS services are deployed on three servers, server4.gs.unina.it, server5.gs.unina.it and server6.unina.it. Each server has one instance of each service. To spread the load you will assigned to use a particular server. The URLs of the services for server4 are given below. Those for server5 and server6 are the same (e.g. replacing server4 by server5).

WS name	URL
SeqGen	http://server4.gs.unina.it:8080/SeqGen/seqgen
XProduct	http://server4.gs.unina.it:8080/XProduct/xprodgen
RandGen	http://server4.gs.unina.it:8080/RandGen/randgen
FileStore1	http://server4.gs.unina.it:8080/FileStore1/filestore
Visualiser	http://server4.gs.unina.it:8080/Visualiser/visualiser

WS name	URL
Surface1	http://server4.gs.unina.it:8080/Surface1/surface
Surface2	http://server4.gs.unina.it:8080/Surface2/surface
Surface3	http://server4.gs.unina.it:8080/Surface3/surface
Surface4	http://server4.gs.unina.it:8080/Surface4/surface
Surface5	http://server4.gs.unina.it:8080/Surface5/surface
Surface6	http://server4.gs.unina.it:8080/Surface6/surface

The WSDLs at these URLs will be used to generate the "glue" components to enable your ServiceWrapper and FileStoreWrapper to contact the web services. The glue components are generated in the following way (following the same procedure as for the "Quote of the Day" tutorial):

1. Set up the correct CLASSPATH by sourcing classpath.sh (see appendix for where to find this).
2. For each Web Service (SeqGen, XProduct, RandGen, FileStore1, Visualiser, Surface1)
   - Edit the XML config file (e.g. named config_randgen.xml) to include the correct URL for the service. The config are contained in the configs_vico.tar.gz (listed in the appendix)
   - Compile the glue components using the command

     ```
     wscompile –gen:client –keep –d . config_randgen.xml
     ```

     This will generate a directory with the name as specified by the "package" parameter in the config file. The java glue code is inside this directory.

Note that the glue components for only one Surface service need to be generated. All Surface services use the same glue and can be accessed by simply changing the service endpoint address. Once you have the generated all the glue components for SeqGen, XProduct, RandGen, FileStore1, Visualiser and Surface1 you can compile your modified Explorer/BigProbe/FileStoreWrapper/ServiceWrapper code. This is done using the command,

```
javac –classpath $CLASSPATH Random.java BigProbe.java
FileStoreWrapper.java ServiceWrapper.java
```

The same classpath option must be used when running Random.

Note that the directories containing the glue code also contain the definitions of the classes that are passed as parameters to the web services, e.g. package randgen contains the classes URLFileName2D and GeneratorException. However, package xprodgen also contains these classes. The Java compiler will complain when there exists more than one class with the same

name unless the classes are qualified with the package name. If you look in ServiceWrapper you will see this has been done for the methods provided. When you complete the probe method the class names must be qualified by the "surface" package name.

Step 1.3.2 Exploring the Surfaces

Each of the six surfaces is defined only in the bounding box [-10,-10; 10, 10]. Each service will handle batches of up to 20 points. Each surface is continuous and has the form:

$$z = \Sigma a_{ij} * x^i * y^j$$

Where i and j are integers in the range [0; 5] and no more than 4 of the coefficients a_{ij} are non-zero. Surface1 uses the same surfaceFunction as has been used in step 1.2.

Working as a team identify which of the coefficients is non-zero in the other five surfaces. If you have time and inclination estimate the values of those coefficients. At this point you might decide as team to all use the same `Explorer` program.

4 Appendix

The exercises are split into three main areas

- Steps 1.1.1 to 1.1.3 a very basic set of surface sampling programs.
- Step 1.2 sampling a surface in a more sophisticated way with helper programs provided.
- Step 1.3 sampling a surface using web services.

Organise your code accordingly and have three separate directories in your home directory. The code required for each stage is available at

http://www.gs.unina.it/repository/tuesday-12

Specifically,

- Steps 1.1.1 to 1.1.3 you are provided with
  - Surface.java
  - PointEvaluator1.java
  - BatchOfPointsEvaluator1.java (skeleton)
  - Box.java
  - BoxException.java

where "skeleton" refers to the fact that some or all of the code is missing but comments are provided as suggestions or hints.

- Step 1.2 you are provided with
  - ServiceWrapper.java (skeleton - one method to add)
  - FileStoreWrapper.java
  - BigProbe.java (skeleton)
  - school.jar

The javadoc for the classes and methods packaged in school.jar are also available at the web page above.

- Step 1.3 you are provided with
  - classpath.sh
  - ServiceWrapper.java (skeleton – one method to fill in)
  - FileStoreWrapper.java
  - BigProbe.java
  - configs_vico.tar.gz