Introduction

This problem set will give you practical experience with domain specific languages, and will help shed light on the quality impacts of language choice. The assignment will also help you reflect on the failure discovery and fault diagnosis process. Finally, if you wish you can use the assignment to experience pair programming by teaming up with exactly one other individual in the class to pursue the entire problem set.

Well organized and neat formatting of submissions will be favorably assessed. This includes any programmatic output, recorded data and brief essays.

This course is about quality and, by extension, code robustness. For problems 1-4, your code's ability to handle unexpected inputs and unusual situations will therefore be weighed heavily in the grading.

Assignment Context

You'll be provided with two Shakespearean plays - Macbeth and The Tempest - which have been marked up according to the provided DTD[1]. These files are for testing. We will be using other files as well for marking. Those two plays (called macbeth.xml and tempest.xml) as well as the DTD (in play.dtd) can all be found on webct.

For both problems 1 and 2 you will create programs to process an arbitrary file. For files in the correct format (as specified by the DTD, which you can assume is fixed for the purposes of the exercise), the programs you create must print out all the speeches in that file that contain a TARGETWORD (single word) provided at run time. The speeches are to be output in an ordered fashion in HTML format, sorted by the first word of the speech text. After each quote, cite the speaker's name, the play, the act and the scene. Do not include any subtitles in the Act or Scene names. For files not in the correct format, the program must at the least not crash, and should offer at messages for any problem that would interfere with carrying out the necessary extraction. The programs should not abort in response to errors that do not interfere with the task at hand, but they may choose to issue warnings in response to these conditions.

Note: For neither problem are you permitted to use external libraries or utilities (e.g. lex, bison/yacc, XML parsing, etc.)

The file hamlet.html provided with the file shows a sample output for the word 'dog' run against the Hamlet play file.

PROBLEM 1: GENERAL-PURPOSE PROGRAMMING LANGUAGE (30%)

Write a C program to perform the task describe above. Please use only one C file for this assignment (The Standard C libraries may be #include'd, but please do not create your own .h file). Please name the C file *nsid*.c (where *nsid* is your U of S nsid).

The program you submit will be compiled on Linux using the GNU C compiler with the command

```
gcc -o NSID NSID.c.
```

The compiled submitted program will be run with the command

```
NSID TARGETWORD < PLAYNAME.xml >PLAYNAME-NSID.html
```

[1] We wish to acknowledge and express our appreciation to Jon Bosak of the University of Wisconsin for providing the Shakespeare markup online. Please note that Jon retains the copyright on these files and that the terms of that copyright require that the files not be modified in any way.

Due November 27, 2006

This means that your program can expect the file on STDIN, will output on STDOUT, and will not have to deal explicitly with file opening and closing.

You will presumably want to test and debug this program before handing it in. When past the compilation stage (i.e. when testing the executable program), please measure and record the following quantities:

1) The testing time needed to detect each successive failure.
2) Any comments on the failure you may wish to share.
3) For each fault, indicate
    a. The failure whose investigation first exposed you to this fault.
    b. Whether this fault was apparently caused by a previously attempted fault fix.
    c. The time needed to fix the fault
    d. The broad nature of the fault (coding fault, design fault). For a coding fault, the pre-conceived design plan was correct but there was a mistake (often a logic mistake) by which it was not correctly realized (this is often associated with a conviction that "I meant to do A but actually did B"). For a design fault, the design intention was off – it hadn't been closely enough thought through.
    e. Any comments on the fault you may wish to share.

Please note that the above does *not* assume either that you will be testing and debugging the program during a single contiguous session (if you use several sessions, please record the time spent on the task even over multiple sessions – not the calendar time elapsed) or that you are finding fixing the responsible faults immediately after each failure (again, just be sure to indicate the time required on each particular defect removal, even if it occurs at a later time).

NB: Problem set marks will *not* be directly influenced by the above information. The marks for Problem 4 will, however, reflect (among other things) the care with which these values have been recorded and (more importantly) the quality of analysis brought to bear on them.

Your deliverable for this section should be the C file and a text (or Excel) file named CFaultsFailures.{csv,xls} that records the following information for each fault and failure found. This information on a specific fault or failure should be placed on a single line in "comma separated variable" format – that is, each successive value should be separated by a comma. Time entries should be in the form *hh:mm:ss*. Note that in recording this information, you may wish to jot down additional information not submitted on each fault, so that you can easily identify any previous fault fix responsible for introducing one or more new faults.

Failures:
Failure #
Time to detect
Any comments (in quotes, and containing no internal quotes)

Faults:
Fault #
Associated Failure #
Preexisting fault fix # whose attempted fix caused this Fault (NULL if none known)
Time to fix (NULL if not fixed)
Type of fault (Coding/Design)
Any comments (in quotes, and containing no internal quotes)

Due November 27, 2006

PROBLEM 2: GENERAL-PURPOSE PROGRAMMING LANGUAGE (30%)

Repeat the exercise above, using an XSLT transformation script instead of C. The XSMLT screent should be named *nsid*.xslt
Submissions will be assessed in Linux using the GNU C compiler and the Gnome XSLT processor called xsltproc.

The XSLT program will be executed using the command

```
xsltproc -o PLAYNAME-NSID.html –stringparam target TARGETWORD NSID.xslt PLAYNAME.xml
```

NB: This means that your transform script will be expected to have an internal variable called target that is used to conduct the search.

As in Problem 1, please record the information on failures and faults encountered, in the same format as specified there. As in Problem 1, you should only begin doing this once you arrive at the stage at which the xslt script is correctly passed by the XSLT processor.

Although it will not bear on your mark, you may wish to familiarize yourself with XSLT prior to starting on this problem, as it may be easier to learn on a smaller example in which the processed text can be easily modified by yourself to check your understanding.

Your deliverable for this portion will be the xlst file and a text (or Excel) file in the same comma separated variable format as above, named XSLTFaultsFailures.{csv,xls}

PROBLEM 3: LANGUAGE EXERCISE ASSESSMENT (20%)

Write a two (or fewer) page assessment that summarizes your analysis of the advantages and disadvantages of each language for this particular project. Comment specifically on what impact you feel the use of each language would have on quality. How might these results scale for other, larger software projects? How might results change if you were to start on each of these problems again, having already gone through this experience? Would the relative benefits of a general purpose language or domain-specific language change if you were required to be flexible in terms of changes to the DTD?

In order to understand the context, please indicate whether you pursued the assignment individually or using pair-programming; if the latter, please note the name of the partner. Finally, please indicate your (and any pair-programming partner's) level of experience with both C and XSLT prior to beginning this assignment.

The deliverable for this problem is the two page pdf document (in a file called *nsidLanguage*.pdf).

Due November 27, 2006

PROBLEM 4: FAULT AND FAILURE ASSESSMENT (20%)

Write a two (or fewer) page summary of reflections from the fault and failure information gained from each exercise. You may find it useful to graph the results in a tool such as Excel (which can read either .csv or .xls files). Do you think any of the observed patterns would be likely to carry over to a larger project? How might these patterns change if you were to redo the assignment given your learning? How might the time required or pattern of faults change if the problem statement had required you only to handle legally formatted text?

The deliverable for this problem is the two page pdf document (in a file called *nsidFaultFailures*.pdf) summarizing your reflections.

Due November 27, 2006

PROBLEM 5: PIPES AND FILTERS AND LANGUAGE TRANSFORMATION (30%)

Please note that this problem may be handed in as an alternative to Problem 1 (if no answer for Problem 1 is turned in) or may be used as a credit towards Problem Set 3.
Within this problem you will be leveraging the understanding of xslt you gain in problem 2 to extract program structure information from source code. You will also experiment with the pipes and filters pattern.

Download from WebCT the java2xml.jar.gz file containing the java2xml code and (if desired) the java-ml.dtd file that specifies the associated DVD. Information on usage can be found here:
https://java2xml.dev.java.net/

Download and install the Graphviz Open Source graph visualization program from
http://www.graphviz.org/Download..php

java2xml takes one or more java source files as input and outputs an xml file that encodes the java program within xml – including detailed markup of constructs within the code, such as the details of expressions and statements.

Start by using java2xml to convert each of the two given java source files (SugarModel.java and Main. to xml in turn.

Your tasks in this assignment will be to extract information from (something very close to) the xml files created by java2xml using xslt scripts of your own creation. This extracted information should be rendered by the xslt files into (something very close to) the DOT Language of graphviz; the Graphviz program DOT should then be used to process this into a jpg (using the Graphviz –Tjpg output option). You may wish to consult with the following page for reference material on Dot:
http://www.graphviz.org/Documentation.php

The “very close to” references above indicates that your xml and/or xslt file output may have to be “fixed up” by piping and filtering using sed, awk or common unix commands such as “tail” in order to “glue” together the different processing steps. If you don’t need this additional support, all the better, but in the event you do, please know that you have the option of doing postprocessing using such tools. You may only use these tools for obvious textual “fixup” and not to perform the key mappings. Please document why you are using these tools when you do so.

Please provide a linux shell script named “Java2JPG” that performs the complete process of mapping a Java file and xslt file to a jpg file. Specifically, this shell script should take the name of the xslt file and the names of one or more .java files on the command line, extracting the desired information via an xslt script and outputting a single jpg file on standard input.

To get you started, please try your hand at the following two exercises:

1) The call graph of a program shows the methods in a program, and places a single edge from method A to method B iff there exists a call from A to B within the program. Create an XSLT script called CallGraph.xslt to extract the directed call graph from the two sample files and render it as an image. For this purpose, you may confine your attention to calls that are statically obvious (i.e. no need for reasoning about polymorphism and the fact that A may call B by calling a superclass C of B). You may also ignore the containing class of a method, the distinction between two methods with the same name but distinguished in different classes. Please create jpgs representing the call graphs of the two example java files (Naming each <*original file name*>.CallGraph.jpg.
2) Please create another graph which shows each class within the java file as a box labeled with the name of the class. For each instance variable and method in that class, the box should contain a rectangle or ellipse (the standard shape) respectively. The graph contains a single undirected

edge between the shape for the method and the shape for the instance variable iff the method directly assigns to the instance variable within that method. Create an XSLT script called InstanceAssignments.xslt to extract this graph from the two sample files and render it as an image. Using this file please create jpgs representing these graphs for the two example java files (Naming each <*original file name*>.CallGraph.jpg.

3) Using all or a subset of these tools, please create one or more of your own custom software visualizations or software information extractions (e.g. software metrics) that you feel could be useful. Extra credit will be awarded for visualizations that are particularly useful or compelling. For this part of the assignment, you should only use the tools discussed above, but you can work on other java files besides the test files provided, and you need not use the Java2JPG script if you feel it is too restrictive; in this case, please create and document at a basic level any additional end-to-end scripts that you create. Please document your reasoning as to why you feel each visualizations may be useful.
4) Please comment on any insights you have gained on the use of domain specific languages from this exercise (general problems or advantages of the technique). Please specifically discuss what you believe would be required to make this technology robust. Could such pipe-and-filter techniques be used to create industrial strength applications; if so, at what cost? Also please note any problems/nuisances that you encountered prevented you from realizing a more compelling contribution in problem 3. Please also discuss any aspects of these technologies that you think may be particularly powerful or attractive.

Due November 27, 2006

CMPT 816 Problem Set 2

Svetlana Slavova
sds797@mail.usask.ca

Problem 1: General-Purpose Programming Language

This problem is not solved. I have started solving the problem but I realized that due to less experience in pure C, I would need much more time to develop a fully-functional program. However, the experience that I gained is used in problem 3 and problem 4.

The functionality that I have developed is as follows:

- The program is able to run as:
  sds797 word < macbeth.xml > output.txt
  It checks how many/which arguments are passed to the program, taking into account the values of *argc* and *argv*.
- The program is able to read the given play from the standard in.
- The program is able to verify (at some degree!) whether the given play corresponds to the xml standard: <?xml version="1.0"?>, etc. It returns an error, if the play is not in this format.
- I have a function: ***getTagName***, which returns the n-th tag of a given row of the play. For example, it can give me: <?xml version="1.0"?>.
- At this point the program only prints out the play in an output file.

CMPT 816 Problem Set 2

Svetlana Slavova
sds797@mail.usask.ca

Problem 2: General-Purpose Programming Language

This document describes my solution for problem 2. The XSLT file is tested under Windows OS, using XSLT processor **xsltproc**.

1. **Steps of the solution**
    1) Get familiar with XSLT – the basic constructs;
    2) Try simple examples, in order to learn the features and the behavior of the language;
    3) Read the whole play in xml format and to output it into an html file;
    4) Display the target word;
    5) Sort the play by first line of the speech – as a result the speeches are sorted by first word, as required;
    6) Display the speeches that contain the target word;
    7) Test of the XSLT script, in order to observe how the program behaves under different conditions.

2. **How to run the XSLT script**
    - The XSLT script is saved in file **sds797.xslt**
    - The xslt script can be run with the following command, as described in the given problem set:

```
xsltproc –o PLAYNAME-sds797.html –stringparam target TARGETWORD sds797.xslt PLAYNAME.xml
```

For example, the following command runs the script with **macbeth.xml** and target word **dog**:

```
xsltproc –o macbeth-sds797.html –stringparam target dog sds797.xslt macbeth.xml
```

- The generated output file is PLAYNAME-sds797.html.
  For example: **macbeth-sds797.html**.

3. **Content of folder Problem2**
    - PS2-Problem2-SvetlanaSlavova.pdf – The current document;
    - macbeth.xml & tempest.xml – The given plays;
    - play.dtd – The DTD of the plays;
    - macbeth-sds797.html – Output of the script, using macbeth.xml and the keyword dog;

CMPT 816 Problem Set 2

Svetlana Slavova
sds797@mail.usask.ca

- tempest-sds797.html – Output of the script, using tempest.xml and the keyword dog;
- sds797.xslt – The XSLT script;
- XSLTFaultsFailures.csv – Contains faults and failures report.

CMPT 816 Problem Set 2

Svetlana Slavova
sds797@mail.usask.ca

Problem 3: Language Exercise Assessment

Note: The whole problem set is pursued individually by Svetlana Slavova.
Prior to beginning this problem set, I had limited experience in pure C programming and no experience in XSLT programming.

When a developer starts working on a software project, he/she will face the language assessment phase. The chosen language or a set of language for a particular project could be crucial for its success. Therefore, before jumping into a particular language, it must be evaluated in terms of whether or not it meets the needs of the project and what are the advantages and the disadvantages of the language.

1. **Language C**
   C is a general-purpose procedural programming language, which provides flexibility – it allows dealing with a variety of problems. The biggest advantage of the language is its speed – C programs run much faster than programs written in other high-level programming languages. C is more appropriate for low-level access and manipulations. However, this freedom impacts the quality of the created applications. The developer must be very careful when dealing with low-level constructs, such as pointers. Bug-fixing is a time-consuming process, compared to other languages.

   In particular, I do not find the language the best choice for XML parsing. Although such a program would have better response time, the developer must create high-level constructs, in order to deal with string manipulations and pointers in an elegant manner. On the other hand, failure detection and correction require very careful planning and testing, even for a small project, in order to capture all possible cases that might occur. In addition, the development of a flexible program that is able to recover from a failure is a challenging process.

2. **Language XSLT**
   XSLT is an extended domain-specific language. Its main usage is related to transformations of XML-based documents.

   XSLT language is appropriate for this project. It allows developing a script, which parses a given xml file, much faster than using C. On the other hand, the XSLT processor takes care of the correctness of the input xml file. I.e., the developer does not have to worry about whether the given xml file is in the proper form or not. This is another advantage of the language, which impacts directly the quality of the program. In comparison, the C developer must take into account the cases in which the input is not in the correct form.

However, XSLT version 1 has a limited usage. Version 2 is more flexible, but it is not compatible with the XSLT processor **xsltproc**. This is the main reason, which does not allow me to create a more interesting solution for problem 2 and problem 5 of the problem set.

3. Conclusions

For this small academic project, the usage of XSLT language is a better choice. However, if the DTD must be flexible, the XSLT language might not be applicable, whereas the general-purpose language, C, would be able to deal with this issue in a much better way.

On the other hand, in case of an industrial product, which must deal with XML parsing, C is more appropriate, because it gives a faster system. I.e., once the application is developed and tested, it would have good performance.

For larger software projects, the response time of the system is crucial. For example, the domain of Web Services relies on message exchange in XML format. It means that the language which implements the communication protocol is of significant importance for the performance of the Web Services applications. If the communication protocol is implemented in C, then the XML parsing would not require significant amount of time. However, the complexity and the interoperability of such a protocol might not allow using C for the implementation, due to quality issues.

The limitations of the XSLT language do not allow creating flexible programs. For example, a simple operation such as incrementing the value of a variable cannot be performed. This results in the development of scripts which have a restricted usage and impact the quality of the application respectively. Therefore, it would be much harder to apply the language in a larger software project.

Finally, even if I had to start again on solving the problems of the problem set, I would take the same steps during the problem solving process and I would get similar results in terms of quality scripts, but I would need less time for scripts development.

CMPT 816 Problem Set 2

Svetlana Slavova
sds797@mail.usask.ca

Problem 3: Fault and Failure Assessment

Since I chose to deal with XSLT, rather than C, I faced a limited number of faults and failures in my script of problem 2. XSLT is a domain-specific language which takes into account whether or not the input is a correct xml file and does not allow file processing if the file is not in the right format. This helps the focus of the developer to be on the xml extraction, rather than on the input verification.

In case of C, the developer must spend a significant amount of time just to check whether the given input file is in the appropriate format. However, if the given xml files are always in the right format, then it would influence the time needed for the development of the C program.

I would say that the number of faults would be much higher in a C program, but such a program would provide better response time and a higher flexibility.

The failures that I have detected are independent, i.e. one fault does not cause another fault. More of the failures occurred because I have not had experience with the language before starting solving the problem set. If I had experience, I would know what I could expect from the language and more importantly, its advantages and disadvantages.

The following diagram shows how much time I spent on trying to fix the faults. Faults 2 and 4 are not fixed due to language limitations.

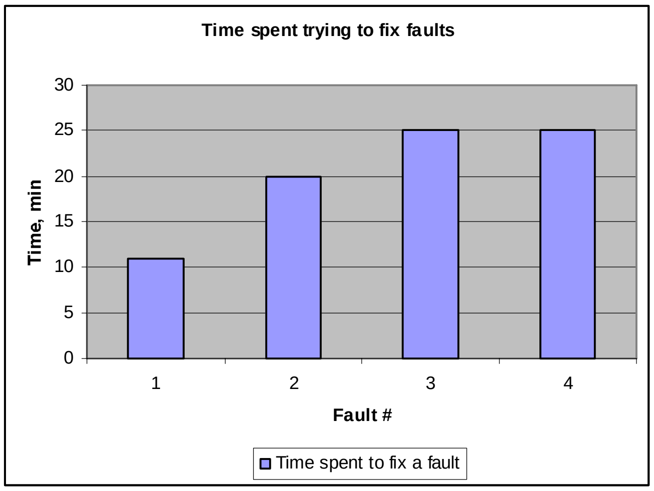


Figure 1. Time spent trying to fix faults

Svetlana Slavova
sds797@mail.usask.ca

In case of a larger project, the developers will face the limitations of XSLT even stronger. The language is powerful, saves time, but this influences on its flexibility, i.e. the programmer does not have a full control on the language. In contrast, C is flexible, but requires more time to be spent as well as there is a necessity of dealing with input verification. If the input is legally formatted all the time, then the time required as well as the faults number will be reduced.

All this suggests that there is a tension between the quality metrics of the developed system, depending on the chosen language.

Finally, if I had to redo the assignment, knowing the advantages and the disadvantages of the XSLT, I would spend less time for developing the script and I would not try to fix faults in the way I tried.

CMPT 816 Problem Set 2

Svetlana Slavova
sds797@mail.usask.ca

Problem 5: Pipes and Filters and Language Transformation

This problem is solved using the following software tools and environment:

- **Windows XP Professional**;
- **java2xml** tool that converts a given java file(s) to the corresponding xml file;
- **Graphviz** Open Source graph visualization program;
- **sed** stream editor that performs basic text transformations;
- **xsltproc** tool as an XSLT processor.

1. Steps of the solution

1) To convert the java files to corresponding xml files using java2xml tool;
2) To create a sed script which fixes the generated xml files so that they can be processed by xslt scripts in step 3;
3) To create several xslt scripts, which are able to extract information from the fixed xml files (in step 2) and to generate text files in Graphviz format;
4) To create a sed script which removes the duplicated lines of the generated text files (in step 3);
5) To visualize the extracted information (in step 3) using Graphviz tool by passing as input the fixed text files (in step 4);
6) To create a batch file that performs the complete process of representing particular information from a java file in an image.

2. Completion of the steps of the solution

1) Convert a java file to an xml file using java2xml tool. The tool generates a file called output.xml.

   Command that runs the script:
   java -jar java2xml-1.0.jar *filename.java*

   where ***filename.java*** corresponds to the java file that is converted.

2) sed script which fixes the generated xml file; The script is called **sed_script_fix_xml.txt**
   Input: xml file
   Algorithm: Replace xml special characters as follows:
   - "" with "
   - "<" with " &lt;"
   - "&&" with "&amp;&amp;"

   Output: fixed xml file

Svetlana Slavova
sds797@mail.usask.ca

Command that runs the script:
sed -f sed_script_fix_xml.txt output.xml > *filename.xml*

where ***filename.xml*** corresponds to the generated xml file.

3) Four xslt scripts are developed. They extract specific information of a given xml file and generate a text file, which is later used for visualization. The xslt scripts are as follows:
   - **CallGraph.xslt** – extract information about the method calls in a given java program in xml format;
   - **InstanceAssignments.xslt** – extract information about the names of the classes, the instance variables and the methods of a java program in xml format, as well as extract information regarding which methods of the class assign values to the instance variables of the class;
   - **CustomGraph1.xslt** – create the general format of a java program; xml file does not have to be specified. For more details, please section: My (custom) visualizations;
   - **CustomGraph2.xslt** – create the format of a java program, using the structure of the java program, represented by the xml file. For more details, please section: My (custom) visualizations.

   Command that runs the script:
   xsltproc -o *filename.txt script.xslt filename.xml*

   where ***filename.txt*** corresponds to the generated text file;
   script.xslt corresponds to an xslt script above;
   filename.xml corresponds to the fixed xml file (in step 2).

4) sed script which removes consecutive duplicating lines of a generated text file by an xslt script file. The script is called **sed_script_remove_duplicated_lines.txt**.
   Note: This script is taken from the documentation of the editor – *sed, a stream editor, version 4.1.4, 26 December 2004, page 24.*
   Input: txt file
   Algorithm: Remove consecutive duplicating lines
   Output: fixed txt file

   Command that runs the script:

Svetlana Slavova
sds797@mail.usask.ca

---

sed -f sed_script_remove_duplicated_lines.txt *filename.txt* > *filename_fixed.txt*

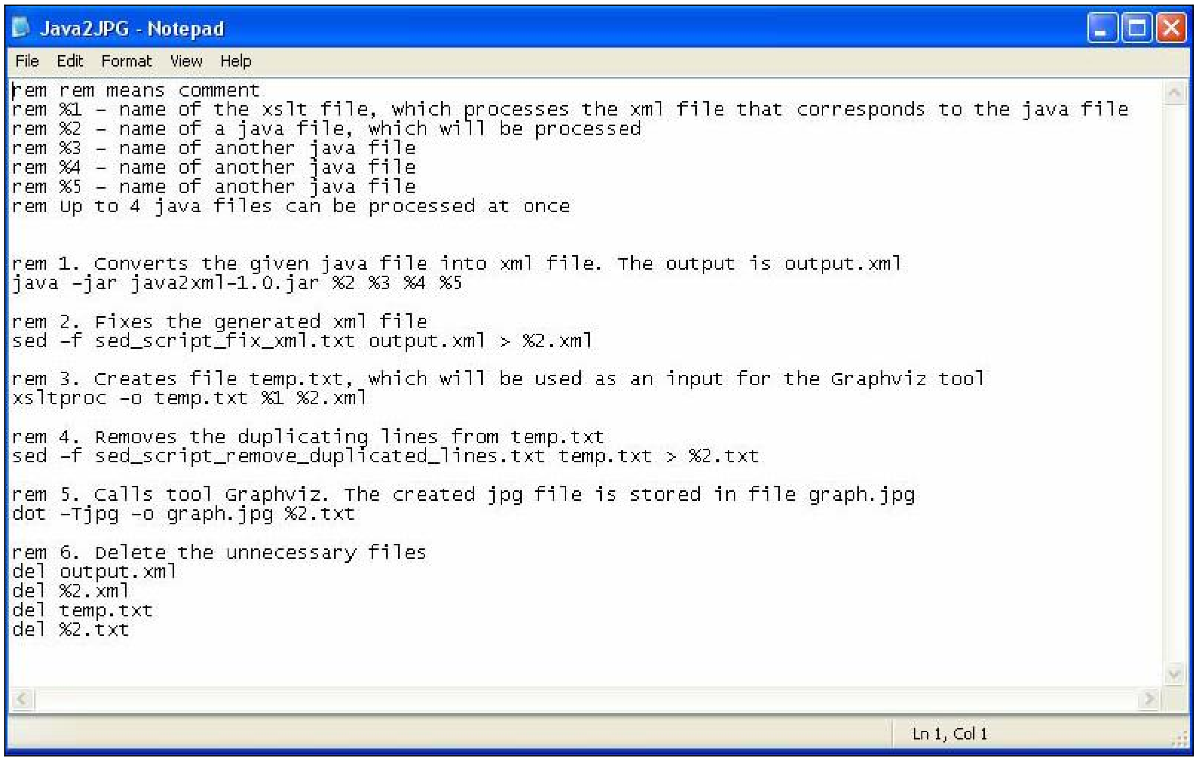
where ***filename.txt*** is the text file generated in step 3;
filename_fixed.txt corresponds to the generated fixed text file.

5) Visualize the extracted information in txt format via Graphviz, dot application. The tool generates a jpg file.

   Command that runs the tool:
   dot -Tjpg -o *graph.jpg filename_fixed.txt*

   where ***graph.jpg*** corresponds to the created image;
   filename_fixed.txt corresponds to the output file in step 4.

6) The batch file, which performs the complete process of representing particular information from a java file in an image, is the following:



```
rem rem means comment
rem %1 - name of the xslt file, which processes the xml file that corresponds to the java file
rem %2 - name of a java file, which will be processed
rem %3 - name of another java file
rem %4 - name of another java file
rem %5 - name of another java file
rem Up to 4 java files can be processed at once


rem 1. Converts the given java file into xml file. The output is output.xml
java -jar java2xml-1.0.jar %2 %3 %4 %5

rem 2. Fixes the generated xml file
sed -f sed_script_fix_xml.txt output.xml > %2.xml

rem 3. Creates file temp.txt, which will be used as an input for the Graphviz tool
xsltproc -o temp.txt %1 %2.xml

rem 4. Removes the duplicating lines from temp.txt
sed -f sed_script_remove_duplicated_lines.txt temp.txt > %2.txt

rem 5. Calls tool Graphviz. The created jpg file is stored in file graph.jpg
dot -Tjpg -o graph.jpg %2.txt

rem 6. Delete the unnecessary files
del output.xml
del %2.xml
del temp.txt
del %2.txt
```

Svetlana Slavova
sds797@mail.usask.ca

Commands that run the batch file:
Java2JPG.bat ***script.xslt filename.java***
Or
Java2JPG.bat ***script.xslt filename.java filename2.java***
Or
Java2JPG.bat ***script.xslt filename.java filename2.java filename3.java***
Or
Java2JPG.bat ***script.xslt filename.java filename2.java filename3.java filename4.java***

where ***script.xslt*** corresponds to the name of the XSLT script to be used; ***filename.java***, ***filename2.java***, ***filename3.java***, ***filename4.java*** correspond to the java files that are processed.

Note: In order to run the batch file, make sure that the PATH variable of the environment is properly setup. The batch file runs on **Windows OS**.

3. My (custom) visualizations

- **Custom visualization 1** – Represents the general format of a Java program. This visualization can be used by tutorial leaders at the Department of Computer Science who teach basics of Java.

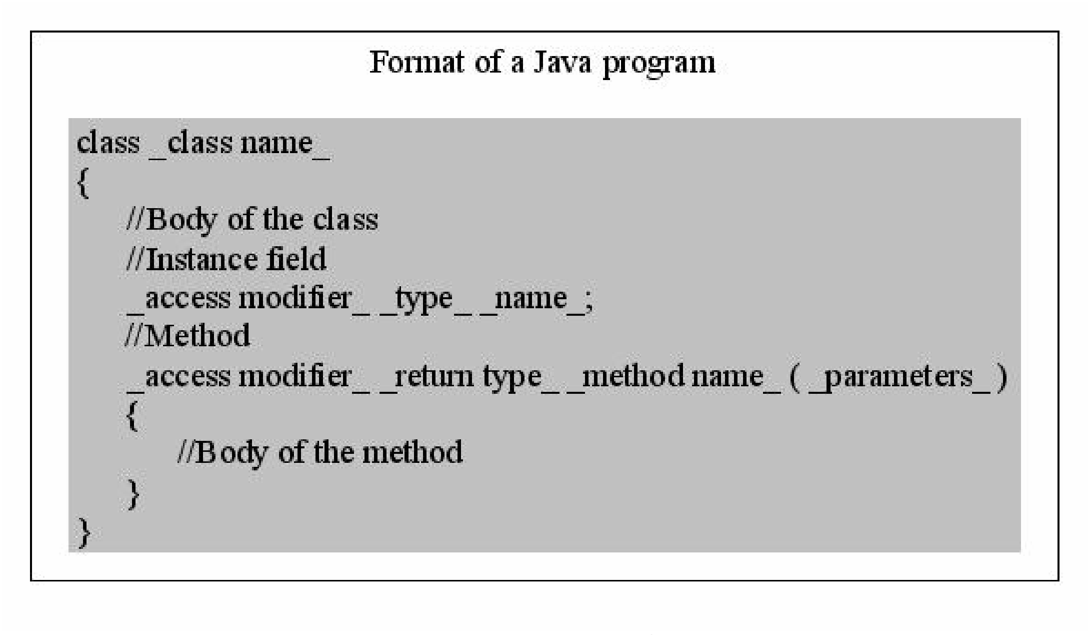
  To create such visualization, run batch file **Java2JPG_Custom1** with parameter **CustomGraph1.xslt** and ***filename.java*** (where filename.java is a java program), as follows:

  Java2JPG_Custom1.bat CustomGraph1.xslt ***filename.java***

  The generated file is **CustomGraph1.jpg**.

  The bat file performs the following steps:
  1) Convert the given java file into an xml file;
  2) Fix the generated xml file using **sed_script_fix_xml.txt**, described above;
  3) Run the xslt script in order to create a txt file;
  4) The created txt file is used as input for the Graphviz application tool to generate the desired visualization;
  5) Delete the unnecessary files which are created during the pipes-and-filters process.

Svetlana Slavova
sds797@mail.usask.ca


Format of a Java program

```
class _class name_
{
    //Body of the class
    //Instance field
    _access modifier_ _type_ _name_;
    //Method
    _access modifier_ _return type_ _method name_ ( _parameters_ )
    {
        //Body of the method
    }
}
```

Figure 1. CustomGraph1.jpg - generated via batch file Java2JPG_Custom1

Note: My initial idea was to have angle brackets ("<" and ">") instead of "_" in the visualization. However, the Graphviz tool does not allow having them in the code of the input txt file when the txt file is generated by the xslt script.

- **Custom visualization 2 -** Represents the format of a Java program, using a particular example of a Java program. This visualization can be used by tutorial leaders at the Department of Computer Science who teach basics of Java.

  To create such visualization, run batch file **Java2JPG_Custom2** with parameters **CustomGraph2.xslt** and ***filename.java*** (where filename.java is a java program), as follows:

  Java2JPG_Custom2.bat CustomGraph2.xslt ***filename.java***

  The generated file is **CustomGraph2.jpg**.

  The bat file performs the following steps:
  1) Convert the given java file into an xml file;

Svetlana Slavova
sds797@mail.usask.ca

2) Fix the generated xml file using sed_script_fix_xml.txt, described above;
3) Run the xslt script in order to create a txt file;
4) The created txt file is used as input for the Graphviz application tool to generate the desired visualization;
5) Delete the unnecessary files.

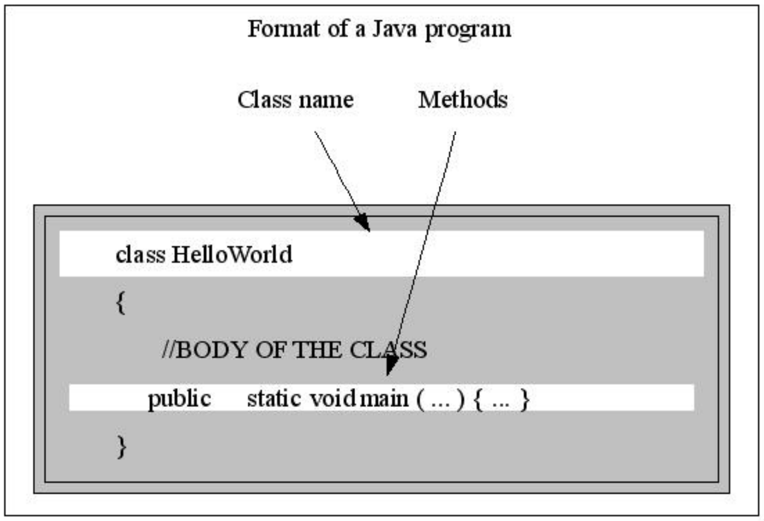


Figure 2. Example 1 of custom visualization 2

Svetlana Slavova
sds797@mail.usask.ca

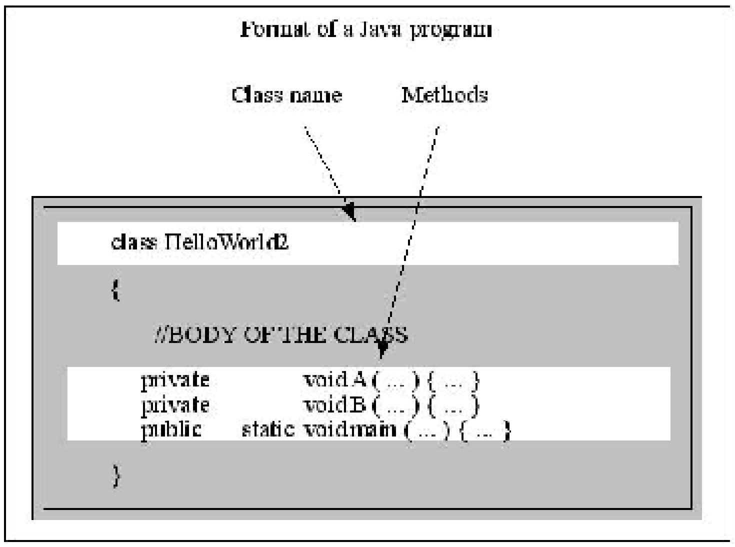

Format of a Java program
Class name
Methods
class HelloWorld2
{
//BODY OF THE CLASS
private void A (...) { ... }
private void B (...) { ... }
public static void main (...) { ... }
}

Figure 3. Example 2 of custom visualization 2

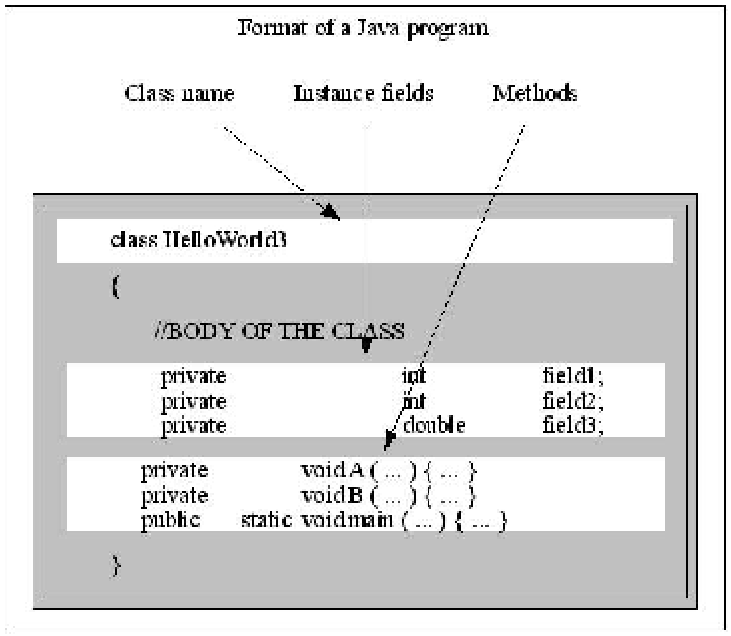
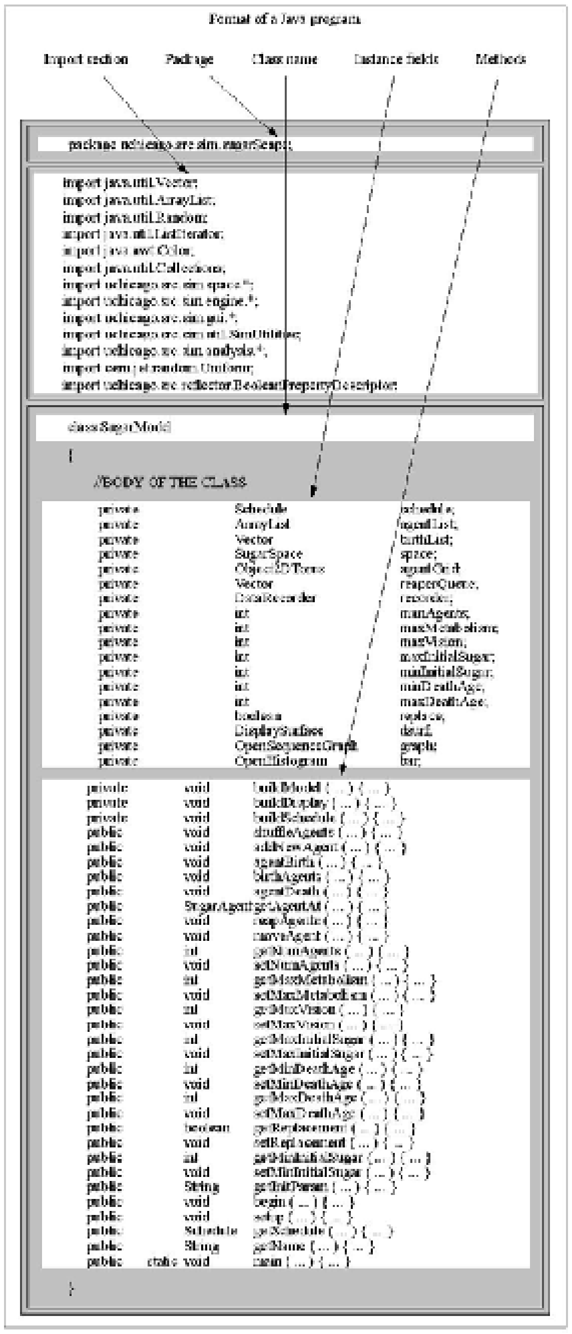


Figure 4. Example 3 of custom visualization 2

Svetlana Slavova
sds797@mail.usask.ca



```
package uchicago.src.sim.sugarScape;

import java.util.Vector;
import java.util.ArrayList;
import java.util.Random;
import java.util.ListIterator;
import java.awt.Color;
import java.util.Collections;
import uchicago.src.sim.space.*;
import uchicago.src.sim.engine.*;
import uchicago.src.sim.gui.*;
import uchicago.src.sim.util.SimUtilities;
import uchicago.src.sim.analysis.*;
import cern.jet.random.Uniform;
import uchicago.src.reflector.BooleanPropertyDescriptor;

class SugarModel

{

    //BODY OF THE CLASS

    private        Schedule              schedule;
    private        ArrayList             agentList;
    private        Vector                birthList;
    private        SugarSpace            space;
    private        Object2DTorus         agentGrid;
    private        Vector                reaperQueue;
    private        DataRecorder          recorder;
    private        int                   numAgents;
    private        int                   maxMetabolism;
    private        int                   maxVision;
    private        int                   maxInitialSugar;
    private        int                   minInitialSugar;
    private        int                   minDeathAge;
    private        int                   maxDeathAge;
    private        boolean               replace;
    private        DisplaySurface        dsurf;
    private        OpenSequenceGraph     graph;
    private        OpenHistogram         bar;

    private          void        buildModel ( ... ) { ... }
    private          void        buildDisplay ( ... ) { ... }
    private          void        buildSchedule ( ... ) { ... }
    public           void        shuffleAgents ( ... ) { ... }
    public           void        addNewAgent ( ... ) { ... }
    public           void        agentBirth ( ... ) { ... }
    public           void        birthAgents ( ... ) { ... }
    public           void        agentDeath ( ... ) { ... }
    public           SugarAgent  getAgentAt ( ... ) { ... }
    public           void        reapAgents ( ... ) { ... }
    public           void        moveAgent ( ... ) { ... }
    public           int         getNumAgents ( ... ) { ... }
    public           void        setNumAgents ( ... ) { ... }
    public           int         getMaxMetabolism ( ... ) { ... }
    public           void        setMaxMetabolism ( ... ) { ... }
    public           int         getMaxVision ( ... ) { ... }
    public           void        setMaxVision ( ... ) { ... }
    public           int         getMaxInitialSugar ( ... ) { ... }
    public           void        setMaxInitialSugar ( ... ) { ... }
    public           int         getMinDeathAge ( ... ) { ... }
    public           void        setMinDeathAge ( ... ) { ... }
    public           int         getMaxDeathAge ( ... ) { ... }
    public           void        setMaxDeathAge ( ... ) { ... }
    public           boolean     getReplacement ( ... ) { ... }
    public           void        setReplacement ( ... ) { ... }
    public           int         getMinInitialSugar ( ... ) { ... }
    public           void        setMinInitialSugar ( ... ) { ... }
    public           String      getInitParam ( ... ) { ... }
    public           void        begin ( ... ) { ... }
    public           void        setup ( ... ) { ... }
    public           Schedule    getSchedule ( ... ) { ... }
    public           String      getName ( ... ) { ... }
    public    static void        main ( ... ) { ... }

}
```

Figure 5. Example 4 of custom visualization 2

Svetlana Slavova
sds797@mail.usask.ca

---

4. Content of folder Problem5

- File **PS2-Problem5-SvetlanaSlavova.pdf** – The current document;
- Folder **pipes_and_filters**:
  - java2xml-1.0.jar – converts a java file into an xml file;
  - CustomGraph1.jpg – Result of my visualization 1;
  - CustomGraph2_HelloWorld.jpg – Result of my visualization 2 using HelloWorld.java;
  - CustomGraph2_HelloWorld2.jpg – Result of my visualization 2 using HelloWorld2.java;
  - CustomGraph2_HelloWorld3.jpg – Result of my visualization 2 using HelloWorld3.java;
  - CustomGraph2_Maind.jpg – Result of my visualization 2 using Main.java;
  - CustomGraph2_SugarModel.jpg – Result of my visualization 2 using SugarModel.java;
  - Java2JPG.bat – Windows shell script;
  - Java2JPG_Custom1.bat – Windows shell script for my custom visualization 1;
  - Java2JPG_Custom2.bat – Windows shell script for my custom visualization 2;
  - sed_script_fix_xml.txt – sed script which fixes xml files;
  - sed_script_remove_duplicated_lines.txt – sed script that removes consecutive lines that are equal;
  - HelloWorld2.java – My java program;
  - HelloWorld3.java – My java program;
  - HelloWorld4.java – My java program;
  - HelloWorld5.java – My java program;
  - HelloWorld.java – My java program;
  - Main.java – Given java program;
  - SugarModel.java – Given java program;
  - CallGraph.xslt – XSLT script;
  - CustomGraph1.xslt – XSLT script for custom visualization 1;
  - CustomGraph2.xslt – XSLT script for custom visualization 2;
  - InstanceAssignments.xslt – XSLT script;
  - SugarModel.InstanceAssignments.jpg, Main.InstanceAssignments.jpg;
  - SugarModel.CallGraph.jpg, Main.CallGraph.jpg;
  - HelloWorld.CallGraph.jpg, HelloWorld5.CallGraph.jpg;
  - HelloWorld.InstanceAssignments.jpg;
  - HelloWorld5.InstanceAssignments.jpg.

CMPT 816 Problem Set 2

Svetlana Slavova
sds797@mail.usask.ca

---

5. Insights from problem 5 of PS2

This problem represents a pipe-and-filter architecture for generating images from java files. The methodology allows creating automatically a large set of images by providing the input java files and the specific XSLT script.

The methodology has various advantages:

- Once the pipe-and-filter application is created, it can be used by different people to create in seconds domain-specific visualizations;
- It can be implemented on different environments, such as Windows and Linux. Migration from Windows to Linux requires small changes to be done in the script of the batch file, in order to represent a Linux shell script;
- The set of tools are available freely, which does not require any investments;
- The tools fit nicely together;
- This methodology is useful for software developers in representing domain-specific information.

Disadvantages of the methodology:

- XSLT has limited flexibility, which does not allow performing some simple and complex operations with the data. For example, the developer cannot change the value of a variable (such as i = i + 1). This requires extra steps (using the stream editor sed) to be done during the process of mapping between a java file to an image;
- XSLT combines both the way of visualization of the image and the extraction (what to visualize). I.e., both the HTML code and the business logic are part of the xslt script. This makes debugging much harder and does not allow to separate "What to extract" and "How to visualize";
- It is hard to control the order of the elements in the image. The default organization leads to not compact visualizations;
- The key constraint of the methodology is that it is useful mainly for software developers because it requires the users of the pipe-and-filter application to be familiar with the set of tools and to have some programming skills. This means that the methodology can be hardly applied in a domain different than Computer Science. For example, an accountant may need such a tool to extract data from one or more documents and to visualize it. However, if the person needs to modify slightly the behavior of the tool, he/she would not be able to do so, since

Svetlana Slavova
sds797@mail.usask.ca

such a modification requires particular knowledge which is outside of the background and the training of the user.

In order to apply widely such methodology, the user should be able to modify the behavior of the application in a transparent manner, i.e. without changing manually the source code of the pipe-and-filter system. This requires a more complex system to be built, in order to take into account various cases of usage that may occur. It means that investments, in terms of IT developers, time, and money, should be done for the development a robust application, which is able to meet the needs and the preferences of many different groups of users. On the other hand, when the system becomes more complex, it is expected that its performance might decrease due to some pipes-and-filters disadvantages (such as creating files at every step, file locking, scalability issues, etc.).

Pipe-and-filter techniques could be useful to create industrial strength applications when they are combined with the advantages of other techniques, such as layering, OOP model, etc. If the pipe-and-filter model is used stand-alone, it could solve small amount of tasks of a limited group of people.