Question 1. [20 MARKS]

Your task is to write a Python program called `yank.py` to extract information from files containing information about professors and students. Your program's first command-line argument determines what output it is to produce; it must also take one or more filenames as command-line arguments. The program must read all of the data from all of the files, and then print it out as described below. If any of the files are incorrectly formatted, your program must print out the name of the file in which the first error was detected, and the line number where the error occurred, and produce no other output.

Each file contains zero or more sections. Each section describes a single professor or student. (You are **not** responsible for detecting redundancy: it is not an error for a person to have several entries in the same or different files.) Each section begins with the word `PROF` or `STUDENT`, followed by the person's name. This keyword must be the first thing on the line: leading whitespace is not allowed.

Each starting line can be followed by zero or more lines of extra information. Each of these lines contains a `+` sign in the first column, followed by a keyword, a colon, and a value. There may or may not be whitespace between these elements, i.e. between the `+` and the keyword, between the keyword and the colon, between the colon and the start of the value, and after the value. Note that the value may contain whitespace, i.e. may be multiple words. Keywords may be made up of alphanumeric characters and the underscore `_`. A keyword may appear several times for a single person.

No blank lines are allowed in the file.

Finally, lines beginning with a `%` character are comments, and must be ignored. Note that `%` may appear within the values associated with keywords; in this case, it does **not** begin a comment.

The following is a typical input file:

```
PROF Dr. Gregory V. Wilson
+ office : Bahen 4240
+ email : gvwilson@cs.utoronto.ca
+ Favorite_Sport : Ultimate frisbee
+ Favorite_Language : Python
% this is a comment
STUDENT Dweezel B. Fickenwhacker
+email:the_dweeze_himself@hotmail.com
+millimarks: -5 for sneezing
+millimarks: +1 for his haircut
STUDENT Adm. Grace Hopper
+ status: not really registered...
+ Achievements: pioneer of early computing
STUDENT Dr. Gregory V. Wilson
+ email : greg@nowhere.com
+ remark : yes, this is redundant, just to make a point
+ also_note : you can have % inside a value
```

Your program's output must be one line for each person for which data is associated with the keyword given to your program as its first argument. The first entry in each line is the person's name, in double quotes. Each other entry must be separated from the one before it by a single vertical bar character `|`, and must be a value associated with that keyword within that entry. You must **not** combine data from separate entries in the file: if a person appears twice, and has the required keyword both times, your program must produce two separate lines of output.

For example, suppose the sample file above was called `test.dat`, and the program was run like this:

```
python yank.py millimarks test.dat
```

then the output must be:

```
Dweezel B. Frickenwhacker|-5 for sneezing|+1 for his haircut
```

If the program is run like this:

```
python yank.py email test.dat
```

The output must be:

```
Dr. Gregory V. Wilson|gvwilson@cs.utoronto.ca
Dweezel B. Frickenwhacker|the_dweeze_himself@hotmail.com
Dr. Gregory V. Wilson|greg@nowhere.com
```

A clean solution is shown below. Important points to note are:

- uses a class to parse files (and to carry around extra information, like the current line number)

- uses a filter to get non-comment lines from input (rather than handling in-line)

- uses the classify/process pattern in the main parsing loop

- accumulates partial results so that if an error message is printed out, *only* the error message appears

- uses regular expressions (rather than substring) to extract data (although it's OK to use substrings to classify input lines, since the rules are pretty simple)

- checks after the main parsing loop to "close off" the last record

```
#!/usr/bin/env python

import sys, re

"""Solution for information-yanking program (Question 1 on final exam)."""

# Parse a single open file, accumulating results.
class Parser:

    # Set up.
    def __init__(self, filename, key):
        self.filename = filename
        self.key = key
        self.input = None
        self.lineNum = 0
```

```
    # Parse input data.
    def parse(self, result):

        # Open input file
        try:
            self.input = open(self.filename, 'r')
        except IOError:
            print >> sys.stderr, "Unable to open %s" % self.filename
            sys.exit(1)

        # Read data a line at a time
        currName = None
        temp = []
        line = self.readline()
        while line:
            if self.isStart(line):
                if currName and temp:
                    result.append(self.makeEntry(currName, temp))
                currName = self.getName(line)
                temp = []
            elif self.isData(line):
                data = self.extractData(line)
                if data:
                    temp.append(data)
            else:
                print >> sys.stderr, "Bad line at %s:%d" % (self.filename, self.lineNum)
                sys.exit(1)
            line = self.readline()
        if currName and temp:
            result.append(self.makeEntry(currName, temp))

        # Close input file cleanly
        self.input.close()

    # Get the next interesting line of input.
    def readline(self):
        line = self.input.readline()
        self.lineNum += 1
        while line:
            if line[0] != '%':
                return line.rstrip()
            line = self.input.readline()
            self.lineNum += 1
        return None

    # Is this a starting line?
    def isStart(self, line):
```

```
        return (line[:4] == 'PROF') or (line[:7] == 'STUDENT')

    # Is this a data line?
    def isData(self, line):
        return line[0] == '+'

    # Get the name field from a data line
    def getName(self, line):
        line = line.strip()
        assert line
        p = re.compile(r'^(PROF|STUDENT)\s+(.+)')
        m = p.search(line)
        if not m:
            print >> sys.stderr, "Bad data line at %s:%d" % (self.filename, self.lineNum)
            sys.exit(1)
        name = m.group(2)
        return name

    # Extract data from a line if the keyword matches, or return None
    def extractData(self, line):
        line = line.strip()
        p = re.compile(r'^\+\s*(\w+)\s*:\s*(.+)')
        m = p.search(line)
        if not m:
            print >> sys.stderr, "Bad data line at %s:%d" % (self.filename, self.lineNum)
            sys.exit(1)
        if m.group(1) == self.key:
            return m.group(2)
        return None

    # Make an output entry
    def makeEntry(self, currName, entries):
        assert entries
        return currName + '|' + '|'.join(entries)


# Main body
if __name__ == '__main__':

    # Must have at least two command-line arguments (key and filename)
    if len(sys.argv) < 3:
        print >> sys.stderr, "Usage: yank key filename..."
        sys.exit(1)

    # Handle arguments
    key = sys.argv[1]
    result = []
    for a in sys.argv[2:]:
```

```
        p = Parser(a, key)
        p.parse(result)

    # If we made it this far, we should display results
    for r in result:
        print r
```

Question 2. [10 MARKS]

Assume that a directory contains the following files:

Makefile

```
# How to compile and run Java.
JAVA_FLAGS = -classpath .:/java/lib/required.jar
JAVA_CMPL  = javac ${JAVA_FLAGS} -source 1.4
JAVA_RUN   = java ${JAVA_FLAGS} -enableassertions

first :
        @echo "Running First Target"

clean :
        @rm -f *~ *.class

quest2d: Five.class
        ${JAVA_RUN} Five

quest2a:
        cat two.txt three.txt four.txt | grep three

quest2e: Seven.class
        @${JAVA_RUN} Seven > seven.out

quest2f: Eight.class
        @echo "ten drummers drumming"

Seven.class:  Six.class Twelve.class
Eight.class:  Seven.class Nine.class

quest2g:
        diff two.txt three.txt

quest2h: three.txt two.txt
        cat two.txt $<

%.class : %.java
        ${JAVA_CMPL} $<
```

two.txt

```
a partridge in a pear tree
two turtle doves
```

three.txt

```
a partridge in a pear tree
two turtle doves
three french hens
```

four.txt

```
a partridge in a pear tree
two turtle doves
three french hens
four calling birds
```

Five.java

```
public class Five {
  public static void main(String[] args) {
    System.out.println("5 golden rings");
  }
}
```

Six.java

```
public class Six {
  public static void main(String[] args) {
    System.out.println("6 geese a laying");
  }
}
```

Seven.java

```
public class Seven {
  public static void main(String[] args) {
    System.out.println("7 swans a swimming");
  }
}
```

Eight.java

```
public class Eight {
  public static void main(String[] args) {
    System.out.println("8 maids a milking");
  }
}
```

Nine.java

```
public class Nine {
  public static void main(String[] args) {
    System.out.println("9 pipers piping");
  }
}
```

Twelve.java

```
public class Twelve {
  public static void main(String[] args) {
    System.out.println("12 ladies dancing?");
  }
}
```

Part (a) [8 MARKS]

For one mark each, explain what happens (i.e. what files are created, modified, or deleted), and what (if anything) is printed to the screen, when the commands given below are run in the order shown.

a. `make quest2a`

```
concatenates the 3 files then pulls out and prints lines with "three" in them
OUTPUT:  three french hens
         three french hens

Marking: 1/2 if they get idea but only put two lines with "three"
         full marks for just the correct output
         1/2 if they say the number of occurrences of string "three"
```

b. `make`

```
OUTPUT:  Running First Target
Marking: 1/2 if they also say it echo's that
```

c. `make clean`

```
OUTPUT: none
Action: For full marks they only need to say it deletes .class files. We
aren't worried about the *~ for the marks
1/2 marks for answer "deletes temporary files"
```

d. `make quest2d`

```
OUTPUT: 5 golden rings
 (actually prints out instructions about compiling and running but we don't
care if they say this or not. We marked the understanding of @ in part a)
ACTION: compiles Five.java to Five.class then runs Five.class
MARKING: give 1/2 mark for the correct output but no mention of compiling
```

e. `make quest2e`

```
Compiles Six.java, Twelve.java and then Seven.java
then runs Seven and sends output to seven.out which gets created
OUTPUT: nothing except the compiling messages
Marking: 1/2 for noting that Six and Twelve are also compiled
         1/2 for noting that seven.out is created
```

f. make quest2f

```
compiles Nine.java
compiles Eight.java
Output: ten drummers drumming
Marking:  the mark here is for saying that Nine and Eight get compiled and
          for NOT saying that seven gets compiled. They don't have to
         explicitly say that seven doesn't but they only get the mark if
         they do list Nine and Eight and don't list Seven.
No half marks for the output here. We are testing the understanding of the
fact that uptodate targets aren't redone.
```

g. make quest2g

```
ACTION: finds difference between two.txt and three.txt and prints to screen.
  also gets Error Code and reports fatal error but we don't care if they
  note this
OUTPUT: we don't care about the actual diff format as long as they point out
that the different line is three french hens.
Marking: 1/2 for explaining what is going on
         1/2 for output with "three french hens" and no partridges or doves
```

h. make quest2h

```
I expected it to concatenate two.txt and three.txt because I expected
$< to reference the first prerequesite (in this case three.txt) but
it doesn't. It just does cat two.txt. So we will take either answer.
ACTION: concatenates two.txt and three.txt OR  just prints two.txt
Output: depends on which interpretation they gave
```

Part (b) [2 MARKS]

Explain what you would change in this Makefile and/or elsewhere so that people could run it on machines that had "required.jar" installed in different places, or on Windows rather than on Unix.

```
ANSWER: Create an environment variable to hold the class path and then use the
environment variable inside the Makefile.
MARKING: 2 marks for this answer more or less
         1 mark for explaining that you change the JAVA_FLAGS macro or
           that we use ; as path separator on Windows and quotations
         1 mark for saying we need to use $SCJAVAPATH but not a clear
             explanation of what that is
         1.5 for showing the Makefile lines with $SCJAVAPATH (or some
             other environment variable) but not saying that it is an
             environment variable or that it gets defined outside of Makefile
         .5 for just saying switch : to ;
```

Question 3. [10 MARKS]

Part (a) [2 MARKS]

Provide a regular expresion that will match all the strings in A and none of the strings in B. (Note that the double quotes are **not** considered to be parts of the strings.)

A - match: "some" "summary" "awesome" "does much" "assimilate" "resemble"
B - no match "asthma" "miasma" "spasm" "Sam" "smoke" "seem"

```
.*s.{1}m.*  or .*s.m.*
```

Part (b) [2 MARKS]

Supply a regular expression which will match any standard, 10-digit North American phone number at the beginning of a line, with or without a long-distance (1) or operator (0) prefix. The area code (first three) and the exchange (second three) may not start with 0. The digit groups must be separated; the separators may be any of the currently popular ones: space, dash (minus sign) or dot ".". The last two of these three separators must be the same, i.e. any separator may be used after the dialling prefix area code, but for the next two separators, if a dash is used after the area code, then a dash must appear after the exchange; and similarly for a space or a dot.

416 287 7219	Match	
416-597-7438	Match	
1.800.555.1211	Match	
0-309-666-8762	Match	
0.905-828-8762	Match	
011 416 345 8765	No Match	prefix is three digits, not one
058-033-3546	No Match	area code starts with 0
1-087.478-0239	No Match	mixed separators
345-568-709	No Match	last field doesn't have 4 digits

```
^((0|1)(\s|\.|-)){0,1}[1-9]\d{2}((\s|\.|-)|\3)[1-9]\d{2}(\4|\3)\d{4}
```

Part (c) [2 MARKS]

You are helping to write a spell-checker for Maori, a language spoken by the first inhabitants of New Zealand. Double vowels may occur, but not double consonants. Write a regular expression that will match against illegal words, i.e. words that contain double consonants. If a word contains double vowels, but not double consonants, your regular expression must not match it.

Match: Takki illo hoorroi wakkarongo hommai kappu

No match: Maaori puurongo tootika

```
Solution: "\b.*([^aeiouAEIOU0-9])\1.*\b"
Note that a pattern that matches any two consonants in should also be
accepted.
```

Part (d) [2 MARKS]

Recall the definition of an ATOM line in a PDB file from E02:

ATOM lines specify the atoms in the molecule. Each line has six fields, which are separated by tabs and/or spaces:

- The word `ATOM`, which must appear at the very beginning of the line.
- A non-negative integer identifier.
- The chemical symbol for the atom, which is one or two letters. The first letter must be upper case; if the second letter is present, it must be lower case.
- The XYZ coordinates of the atom, which are three floating-point numbers with an optional minus sign, one or more digits before the decimal point, and one or two digits after the decimal point.

Nothing else may appear on an ATOM line.

Write a regular expression that will match all (and only) the lines in a PDB file that represent ATOM lines.

```
"^ATOM\s+\d+[A-Z][a-z]?\s+(-?\d+\.\d\d)\s+(-?\d+\.\d\d)\s+(-?\d+\.\d\d)\s*$"
```

Part (e) [2 MARKS]

The Python program shown below reads data from standard input, and prints out values extracted from lines that match a regular expression. Write the regular expression so that when the program is given the input shown, it produces the output shown.

Program

```
import sys, re

regexp = re.compile("YOUR EXPRESSION GOES HERE")

for line in sys.stdin.readlines():
    matchObj = regexp.match(line)
    if matchObj:
        print m.group(1), m.group(3)
```

Output

```
Faster Or
Why It
I It
He You
```

Input

```
"Faster," he said, "Or we'll be late."
"Why rush?" she asked.  "It's only the final exam."
"What do you mean, it's only the final exam?"
"I mean, it's only the final exam.  It's no big deal."
He looked at her.  "You know, Professor Craig,
sometimes I think you're a little too laid back."
```

```
".?([A-Z][a-z]*)([^A-Z]*)([A-Z][a-z]*)"
```

Question 4. [10 MARKS]

A prefix sum is a series of partial sums; the first value in the output is the first value from the input, the second value in the output is the sum of the first two values of the input, the third value in the output is the sum of the first three values from the input, and so on. The examples below show the prefix sums for three short sequences.

Input	Output
1	1
1, 3, 5	1, 4, 9
1, -1, 1, -1	1, 0, 1, 0

A colleague of yours has written a Java method called `IntPrefixSum` that calculates the prefix sum of an array of integers, and returns a new array containing the result. Fill in the table below to show the **five most useful** tests you could use to check that her method is working correctly. For each test, explain:

- **why** you think it is important
- **how** you might expect it to fail
- **what** you would learn if it failed

```
2 marks per case. No marks is all 3 parts aren't given
For why/how/what we are not looking for a lot of insight into what but
for 2 marks they must give a clear description of the category of the
test case and they must give a decent explanation (maybe output) that
would be expected in failure. Give 1 mark if they have a correct case but
the explanation is there but weak. No marks if it is wrong. Do not give
marks for essentially the same case twice. Note below that my cases 2,3&4
could be considered essentially the same case (4 includes 2 and 3) To have
these count as "different" cases, the explanations have to be clear about
how they are different.

Note also that the case of an input array that contains other than integers
is not a valid test case. It wouldn't compile in the first place.

Possible Cases
----------------
input array of length 0
array of length >1 where sums are positive increasing (boring standard case)
array of length >1 where some inputs are negative but outputs +ve
where output array where sum is negative at some points
array where intermediate sum is max int
array of length >1 where sums are positive
very long array
input array which is null
```

Question 5. [10 MARKS]

Consider a Java method `IntegerPrefixSum` that performs the same calculation as the `IntPrefixSum` method in the previous question, except that `IntegerPrefixSum`'s input and output are both `Lists` of `Integer` objects (rather than arrays of int's). `IntegerPrefixSum` throws an `InvalidDataException` if its input argument is null, or if any of the elements in that `List` are not `Integers`. Using `junit.framework.TestCase` and other classes from JUnit, write five test methods for this `IntegerPrefixSum`. Your code will be marked for style, as well as for your actual tests.

```
This is not marked 2 marks per test case but instead marked as follows:
  1 mark for the fact that they have written separate junit test methods
      each testing one thing (not multiple things)
  1 mark for the fact that the names represent well what is being tested in
      each method and start with "test"
  1 mark for using the junit AssertEquals or AssertTrue methods
         OR for correct test suite set-up
  2 marks for correctly writing code that tests one of the cases that
     is expected to throw an Invalid Data Exception
    (this code has to be correct i.e. cause the excpetion to be thrown and
     then catch it and fail if the exception is not thrown)
  2 marks for correctly written code that tests something where you do not
     expect an exception.
    -1 here if assume output of program is just a single int and not list
    -1 here if python-like syntax used to input lists
          List a = [1,2,"hello"];
    no deduction if forget to cast result of get(x) method to Integer and
        then convert to int
    no deduction for forgetting to wrap int with Integer when adding to List
    no deduction for trying to instantiate List instead of some
           List-implemening class  (List a = new List();)
    -1 for mention of completely extraneous hunks of code (factory, HashMap
        etc.) that likely comes from copying examples without understanding
  3 marks for the choice of tests:
      should include any 5 of the following
        1. input List is null
        2. input List contains a null entry
        3. input List contains a non-null non-Integer entry
        4. input List is non-null but length 0
        5. input List has valid Integer in List of length 1
        6. input List has valid Integers in List of length >1
         (plus any of the tests from previous question)

    Note: give 3/3 if they have written 5 methods from above list even if they
aren't perhaps the "best" 5. If they omit 1,2 and 3 they will have lost 2 marks
earlier.  Give 2/3 if they have only 3 or 4  methods or if they duplicate
a case within their 5 instead of 5 different methods. Give 1 if they have
1 or 2 different cases and 0 if they have no reasonable test cases.
```

Additional Note: for case 4 above the program should return an empty List
not throw an exception. I counted returning an exception as a wrong case
so depending on how many other "wrong" cases this may have affected this last
out-of-3 grade.

Question 6. [15 MARKS]

Write a Java method which reads an XML configuration file for filters and returns a `Map` whose keys are the names of the filters used in the configuration file, and whose values are a `Set` of the names of the parameters used with that filter. If the format of the configuration file is invalid, your program must throw an `InvalidConfigDataException`.

The format of the configuration file is the same as used in lectures. The root element must be `MultiFilter`; it must contain only ignorable whitespace (tabs, spaces, newlines, etc.) and `Filter` elements. Each `Filter` element must have a `className` attribute, whose value must be a non-empty string. Each `Filter` element may contain ignorable whitespace, and zero or more `Param` elements. Each `Param` element must have name and value attributes, the values of which must be non-empty strings. Note that you are **not** responsible for checking that class and parameter names are legal Java names.

Here is an example of a typical configuration file:

```
<?xml version="1.0" ?>
<MultiFilter>
    <Filter className="FilterRepeat">
        <Param name="times" value="3"/>
    </Filter>
    <Filter className="FilterSample">
        <Param name="skip" value="2"/>
    </Filter>
    <Filter className="FilterEcho"/>
    <Filter className="FilterRepeat">
        <Param name="times" value="5"/>
        <Param name="limit" value="1000"/>
    </Filter>
</MultiFilter>
```

Your method must produce a `Map` of `String`s to `Set`s of `String`s. For the input above, your method's output must be equivalent to the following:

```
{
  "FilterRepeat" : {"times", "limit"},
  "FilterSample" : {"skip"},
  "FilterEcho"   : {}
}
```

```
A clean solution is shown below (including all of the import statements
required to compile it).  Important points to note are:

- Only the 'analyze()' method is to be marked.

- It parses the XML document specified by a filename.  Some students
  may pass an open file to the method; this is acceptable, and should
  not be penalized.  Passing a DOM document (that has already been
  parsed) will incur a -1 mark penalty, since "file" cannot reasonably
  be interpreted to mean "DOM document".

- It examines the children of the root, and their children, with
```

```
  nested loops rather than recursion.  Since the structure of the
  document is well-defined, and not nested, this is OK; students can
  recurse (e.g. using a Visitor) if they want to.

- It checks the result map to see if there is already a Set associated
  with a particular Filter, and creates one if there is not.

- It uses the values associated with the attributes 'className' and
  'name', rather than the name of the DOM element.  If students use
  the names of the DOM elements, the only entry in their result will
  have the key 'Filter', and the set associated with it will only
  contain the word 'Param'.

- This code does *not* check for errors;

import java.util.Map;
import java.util.HashMap;
import java.util.Set;
import java.util.HashSet;
import java.util.Iterator;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.input.SAXBuilder;

class Config {

    public static void main(String[] args) {
        if (args.length != 1) {
            System.err.println("Usage: Config filename");
            System.exit(1);
        }

        Map result = analyze(args[0]);
        System.out.println(result);
    }

    public static Map analyze(String filename) {

        // Parse the XML
        Document doc = null;
        try {
             SAXBuilder builder = new SAXBuilder();
             doc = builder.build(filename);
        }
        catch (Exception e) {
             System.err.println(e);
             System.exit(1);
        }
```

```
        // Create the result
        Map result = new HashMap();

        // Handle each Filter in turn
        Element multiFilter = doc.getRootElement();
        Iterator outer = multiFilter.getChildren().iterator();
        while (outer.hasNext()) {
            Element filter = (Element)outer.next();

            // Make sure the map contains a set for this Filter
            String filterName = filter.getAttributeValue("className");
            if (! result.containsKey(filterName)) {
                result.put(filterName, new HashSet());
            }
            Set nameSet = (Set)result.get(filterName);

            // Add the Param names to the set
            Iterator inner = filter.getChildren().iterator();
            while (inner.hasNext()) {
                Element param = (Element)inner.next();
                String name = param.getAttributeValue("name");
                nameSet.add(name);
            }
        }

        return result;
    }
}
```

Question 7. [10 MARKS]

Write a Java method called `GetAllStrings()` that takes a single parameter of type `Object`, and returns a `Map` whose keys are the names of the member variables of that object which are of type `String`, and whose values are the values of those member variables for that object. The code below shows an example of how this method will be called and its output.

```
public class Pair {
    public Pair(String left, String right) {
        fLeft = left;
        fRight = right;
    }
    public String fLeft, fRight;
}

public static void main(String[] args) {
    Pair p = new Pair("black", "white");
    Map m = GetAllStrings(p);
    Iterator i = m.keySet().iterator();
    while (i.hasNext()) {
        String key = (String)i.next();
        System.out.println("\"" + key + "\" == \"" + m.get(key) + "\"");
    }
}
```

Output

```
"fLeft" == "black"
"fRight" == "white"
```

```
 1    public static Map GetAllStrings(Object o) throws IllegalAccessException {
 2      Map result = new HashMap();
 3      String fieldName;
 4      String fieldValue;
 5
 6      // get the class type of the object, and its fields
 7
 8      Class c = o.getClass();
 9      Field fields[] = c.getFields();
10
11      //Walk through the fields:
12
13      for (int i=0; i < fields.length; i ++){
14
15        //find fields of type String:
16
17        if (fields[i].getType().getName().equals("java.lang.String")){
18
19          //get the field name and the value; store them in result
20
```

```
21          fieldName = fields[i].getName();
22          fieldValue = (String)fields[i].get(o);
23          result.put(fieldName, fieldValue);
24        }
25      }
26      return result;
27    }
28  }
```

Question 8. [15 MARKS]

Part (a) [3 MARKS]

What is multiple inheritance? Describe one of the difficulties in allowing it and how a language might deal with that difficulty.

Multiple inheritance means deriving *implementation* (not just interface) from two or more parent classes. Difficulty is collision between names, e.g. two or more parents both define a method foo() or a member variable bar. Solutions are (a) forbid multiple inheritance (Java), (b) specify conflict resolution rules (Python and C++), or (c) allow users to specify which one they wants.

Part (b) [3 MARKS]

Lillian has been working on Exercise 12 on her computer at home. She goes into the university and does a "`cvs update`" in her e12 directory. She sees the following:

```
U first.java
C second.java
```

Explain what the "U" and "C" mean. What actions if any should Lillian now take?

U means ``updated'': the file has been changed somewhere else, and those changes are being brought down. C means ``conflict'': the file has been changed locally, *and* somewhere else, and the changes collide. Lillian should examine second.java to resolve the conflict (1 mark) and then check in her changes.

Part (c) [3 MARKS]

What is a breakpoint? When and why would you use one?

A breakpoint is a location where the program is to stop so that its state can be examined. You would use one when debugging, in order to check that your assumptions about what a program is doing are correct.

Part (d) [3 MARKS]

Douglas has been given the job of improving a Java program which marks the style of source code written by CSC207 students. He finds that the original author has identified blank lines in the input as follows:

```
if (line == "") {
    numBlankLines += 1;
}
```

Briefly explain two things that are wrong with this, and describe what Douglas could do to improve this code.

The two errors are using ‘‘==’’ instead of of the ‘‘equals()’’ method, and comparing to the empty whitespace-only lines. The solution is either to trim the line (using the string trim() method in Java) before comparing it to the empty string, and/or to use a regular expression to check the line's contents.

Part (e) [3 MARKS]

Pick **one** of the following design patterns and briefly explain it (only one will be marked). Indicate which you are explaining by checking the appropriate box. ☐ Visitor ☐ Filter ☐ Wrapper

These patterns are straight out of the notes. A Visitor is a class that visits each element of a complex structure in turn, giving the user a chance to do something at each node (either via a callback, or through subclassing the Visitor). A Filter transforms an input stream into an output stream, and a Wrapper encloses an instance of one class, making it look like another (or altering the behavior of one or more of the wrapped object's methods, e.g. to log calls).

Total Marks = 100