```
1  public class Q1NumberThing
2  {
3      /*************************************************************
4      * Sums all numbers in the list greater than y.               *
5      * Divides the total by the magnitude of the list.            *
6      * If anything is invalid, -1.0 is returned instead.          *
7      *************************************************************/
8      public static double numberThing(List<Integer> intList, int y)
9      {
10         if (intList == null)
11             return -1.0;
12         else
13         {
14             double sum = 0.0;
15             for(Integer nextInt: intList)
16             {
17                 if(nextInt > y)
18                     sum = sum + (double)nextInt;
19             }
20
21
22
23
24             double result = sum / (double)intList.size();
25             return result;
26         }
27     }
28
29     public static void main(String[] args)
30     {
31         double result = numberThing(/*parameters go here*/);
32         System.out.println(result);
33     }
34 }
```

(a) Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code. **[4 marks]**

(b) If possible, give a test case that does not execute the fault. If not, briefly explain why not. **[2 marks]**

(c) If possible, give a test case that executes the fault, but does not result in an error state. If not, briefly explain why not. **[3 marks]**

(d) If possible give a test case that results in an error, but not a failure. If not, briefly explain why not. **[2 marks]**

(e) In the given code, describe the first error state. Be sure to describe the complete state. **[3 marks]**

(a) Use the following method for the questions below:

```
1  public static void gradeCalculator(double[] aMax, double[][] marks)
2  {
3      for(int j = 0; j < marks.length; j++)
4      {
5          double maxMark = 0.0, minMark = aMax[j], average = 0;
6          int pass = 0;
7          for(int i = 0; i < aMax.length; i++)
8          {
9              minMark = Math.min(minMark, marks[i][j]);
10             maxMark = Math.max(maxMark, marks[i][j]);
11             average += marks[i][j];
12             if (marks[i][j] >= (aMax[j]/2.0))
13                 pass++;
14         }
15         average = average / (double)aMax.length;
16         System.out.println("\nFor assessment " + (j+1)
17                             + "\n\t Max mark is: " + maxMark
18                             + "\n\t Min mark is: " + minMark
19                             + "\n\t Average mark is: " + average
20                             + "\n\t Pass rate is: " + pass);
21     }
22 }
```

(i) Draw the Control Flow Graph (CFG). **[4 marks]**

(ii) Enumerate the test requirements for Node Coverage, Edge Coverage, Edge-Pair Coverage and Prime Path Coverage for the graph. If test requirements for some criteria are the same as for others, you do not need to copy them, just write "same as X".

NOTE: If your graph has more than 12 prime paths then list only 12.**[10 marks]**

Node Coverage:

Edge Coverage:

Edge-Pair Coverage:

Prime Path coverage:

(b) Consider the graph:

```
N  = {1, 2, 3, 4, 5, 6, 7, 8, 9}
N₀ = {1}
Nf = {9}
E  = {(1, 2), (2, 3), (2, 9), (3, 4), (4, 5), (4, 8),
      (5, 6), (6, 7), (6, 5), (7, 4), (8, 2)}

def(pass) = {1, 8}
def(final) = {1, 3, 7}
def(mark) = {5}

use(pass) = {8, 9}
use(final) = {7, 8}
use(mark) = {6, 7}
```

(i) Draw the appropriate graph. **[2 marks]**

(ii) Identify the du-paths for `pass`, `final`, and `mark`. If a def and use are at the same node, assume the use comes first.
Hint: labelling your paths will be useful for the next question. **[3 marks]**

mark:

pass:

final:

iii) Propose test paths that will fully test your du-paths.