CS2312 Problem Solving and Programming
Mock paper for Midterm (A short version, 50 minutes only)

- Apply best practices and skills of **object-oriented programming**.
- Pay attention to **code design, naming and code formatting**.
- Give precise code.
- **Program execution speed** is **NOT** a major concern**.**
- **Do not remove any given code.**

Additional marking concerns:
Whenever appropriate, you should use the @Override annotation, for-each loop, proper access modifiers.
Marks may be deducted for redundant or unnecessary code.

Duration: 50 minutes. Total mark: 40 marks

Question 1 (15 marks)

We are to analyze the profits earned in 12 months stored a file. For simplicity, only the total is reported, as shown below:

Test cases:		*Expected Outputs*: .
Case 1: The file *profits.txt* has 12 numbers:	113890000 101200000 98040000 97670000 94740000 102990000 114920000 113180000 82950000 119060000 80620000 93560000	Total: 1212820000 From main(): Reporting is OK.
Case 2: The file *profits.txt* does not exist.		Cannot open input file!
Case 3: The file *profits.txt* exists but has less than 12 numbers.		Insufficient data! Total: [cannot be calculated]
Case 4: The file *profits.txt* has non-integer like "oneMillion"		Wrong data! Total: [cannot be calculated]

For simplicity, assume that case 3 and case 4 will not happen at the same time.

Mary has written the program on next page. However, she did not make use of Exception Handling correctly. Mary's program needs to be corrected and improved. Rewrite it with proper code.
Note: the program outputs should be the same as the *Expected Outputs*.

Mary's program:

```
public static void main(String[] args)
{
   try {
      String report = analyze();
      System.out.println(report);
   } catch (FileNotFoundException e) {
      System.out.println("Cannot open input file!");
   }

   System.out.println("From main(): Reporting is OK.");
}


--------------------------------------------------------------


private static String analyze() throws FileNotFoundException {

   int total=0;

   Scanner f = new Scanner(new File("profits.txt"));

   for (int i=0;i<12;i++)
   {
      try {
           int value = f.nextInt();
           total+=value;              //for simplicity, assume no overflow problem
      } catch (InputMismatchException e) {
           System.out.println("Wrong data!");
           total=-1;
      } catch (NoSuchElementException e) { //NoSuchElementException happens when it
                                           //attempts to read after the end of the file
           System.out.println("Insufficient data!");
           total=-1;
      }
   }

   f.close();

   if (total==-1)
      return "Total: [cannot be calculated]";
   else
      return "Total: "+total;
}
```

Question 2 (25 marks)

Students often need to look for teammates for doing projects. To do so very often there are multiple rules used by individual students, and one may change his rules at any time. For example, the program below contains 6 students and 2 rules. At first Ada and Billy do not have any rule and so all others are in their target list. Then Ada adopts the rule of *same gender*, then her target list is reduced to Candy, Daisy and Emily only. She then adds one more rule: *Similar Background* (For simplicity, we simply check whether the students are in the same programmes), then her target list contains Candy and Daisy only.

```
public class Main {
    public static void main(String[] args) {
        Student a = new Student("Ada", 'F', "CS"); //name, gender, programme
        Student b = new Student("Billy", 'M', "Math");
        Student c = new Student("Candy", 'F', "CS");
        Student d = new Student("Daisy", 'F', "CS");
        Student e = new Student("Emily", 'F', "Math");

        Rule rG = new RuleSameGender();
        Rule rBk = new RuleSimilarBackground();

        a.listTargets(); //[Output] Ada: Billy Candy Daisy Emily
        b.listTargets(); //[Output] Billy: Ada Candy Daisy Emily

        a.addRule(rG);
        a.listTargets(); //[Output] Ada: Candy Daisy Emily
        a.addRule(rBk);
        a.listTargets(); //[Output] Ada: Candy Daisy

        Student f = new Student("Fanny", 'F', "CS");
        a.listTargets(); //[Output] Ada: Candy Daisy Fanny
        f.listTargets(); //[Output] Fanny: Ada Billy Candy Daisy Emily
    }
}
```

(a) (22 marks)

Implement all classes/interfaces for this program:

- `Student`: 15 marks
- `Rule`: 3 marks
- `RuleSameGender`: 4 marks
- `RuleSimilarBackground`: 0 marks (You don't need to implement it because it is similar to `RuleSameGender`.

The **outputs** are given as underlined comments in the `main()` method.
Note: Do not use the **instanceof** operator. Otherwise you will get 0 mark for this question.

(b) (3 marks)

How does your solution achieve the *Open-close principle* of OO design? Briefly explain in at most 40 words.

-- END OF QUESTIONS --

Question 1.

```
public static void main(String[] args)
{


    try {
        String report = analyze();
        System.out.println(report);

















}
-------------------------------------------------------------
private static String analyze() throws FileNotFoundException
{

    int total=0;

    Scanner f = null;



















}
```

Question 2.

You may add more pages if needed.