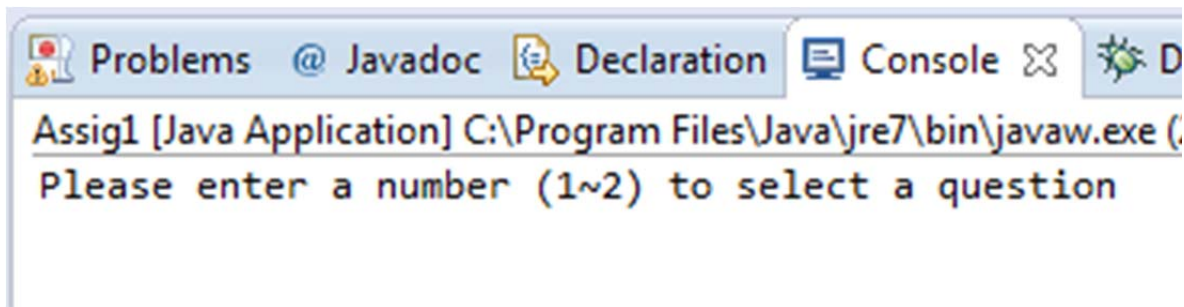


Laboratory Five

Activity 0: Preparation

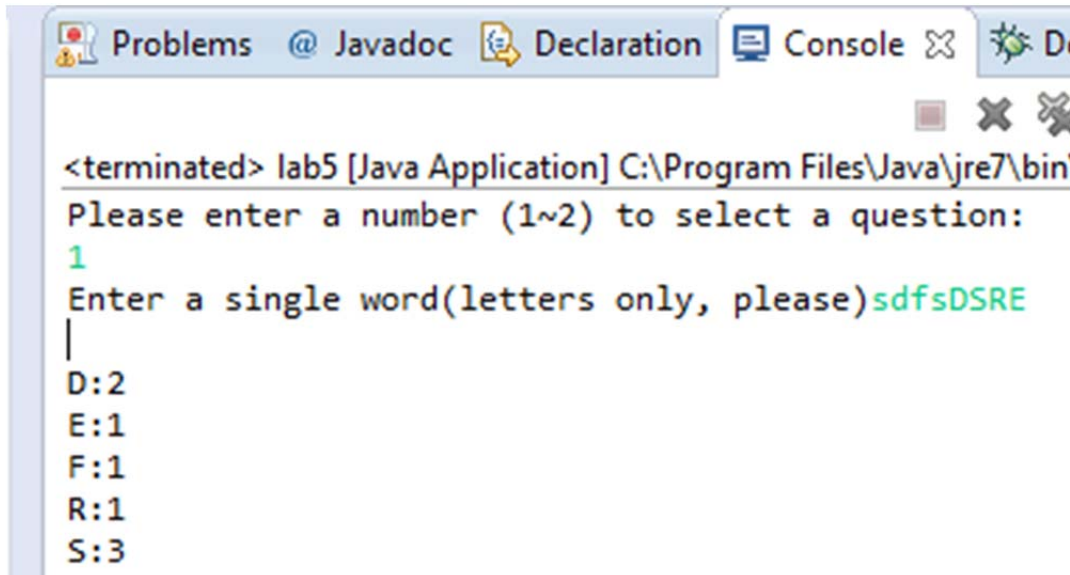
1. Create a project called “lab4” under Eclipse;
2. Copy “lab4” into project;
3. Copy “questions” into project;

Make sure your program can run correctly. An expected output of the UI is shown below.



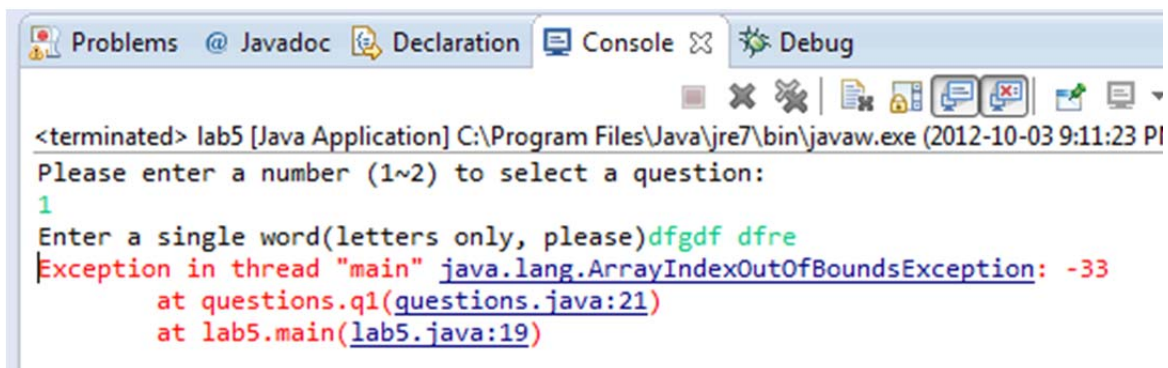
Activity 1: Count Letters (2 marks)

q1 contains a program which reads a word from the user and prints the number of each letter in the word. Compile your project and run it to check the output. In reading the code, note that the word is converted to all upper case first, then each letter is translated to a number in the range 0~25 (by subtracting 'A') for use as an index. Now test this method by inputting a word consisted of only letter (A~z), an expected output is shown below:



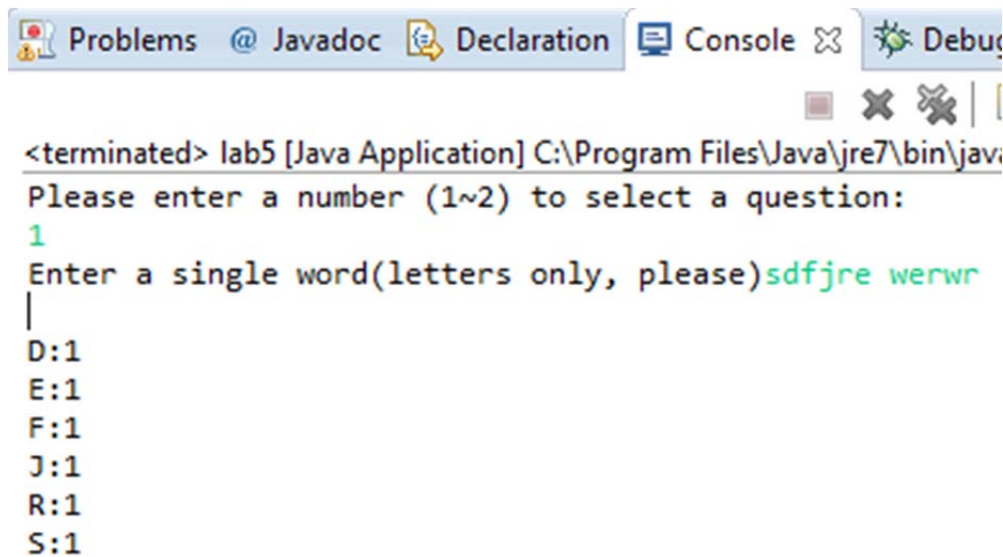
```
<terminated> lab5 [Java Application] C:\Program Files\Java\jre7\bin\
Please enter a number (1~2) to select a question:
1
Enter a single word(letters only, please)sdfsDSRE
|
D:2
E:1
F:1
R:1
S:3
```

Now let's try to input a string containing some other characters, such as space or digital. Unfortunately, this time we encounter an exception called “*ArrayIndexOutOfBoundsException*” shown below:



```
<terminated> lab5 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (2012-10-03 9:11:23 PI
Please enter a number (1~2) to select a question:
1
Enter a single word(letters only, please)dfgdf dfre
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: -33
    at questions.q1(questions.java:21)
    at lab5.main(lab5.java:19)
```

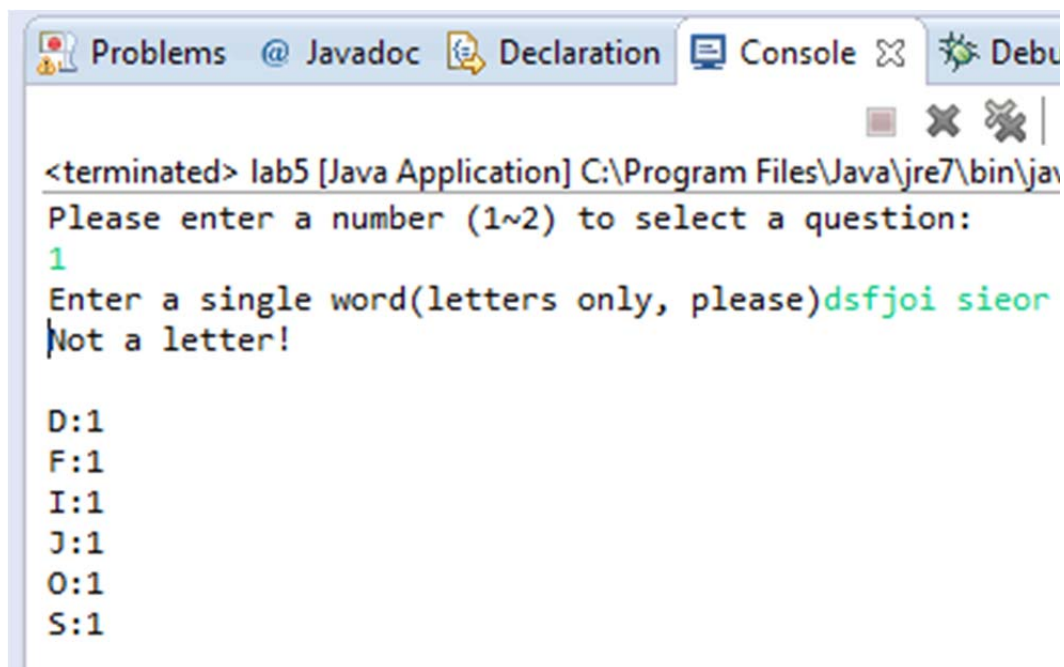
1) Now, let's check the source code, there are two “for” loops, select one of them and surround it with *try...catch* block. This time, don't do anything in catch block. Run it again and check the output. Your code should be able to deal with illegal input.



The screenshot shows an IDE console window with tabs for Problems, Javadoc, Declaration, Console, and Debug. The console output is as follows:

```
<terminated> lab5 [Java Application] C:\Program Files\Java\jre7\bin\jav
Please enter a number (1~2) to select a question:
1
Enter a single word(letters only, please)sdfjre werwr
|
D:1
E:1
F:1
J:1
R:1
S:1
```

2) Now modify the body of *catch* block so that it prints a useful message (e.g., “Not a letter”) followed by the exception. Compile and run the program. An expected output is shown below:



The screenshot shows the same IDE console window as before, but with an additional line of output:

```
<terminated> lab5 [Java Application] C:\Program Files\Java\jre7\bin\jav
Please enter a number (1~2) to select a question:
1
Enter a single word(letters only, please)dsfjoi sieor
Not a letter!
D:1
F:1
I:1
J:1
O:1
S:1
```

Activity 2: Reading from and Writing to Text Files (3 marks)

In *q2* contains a skeleton of the program that writes a program that will read a file of student academic credit data and create a list of students on academic warning. The list of students on warning will be written to a file. Each line of the input file will contain the student name (a single String with no spaces), the number of semester hours earned (an integer), the total quality points earned (a double). You can find a file called “student.txt” in lab5 package.

The program should compute the GPA (grade point or quality point average) for each student (the total quality points divided by the number of semester hours) then write the student information to the output file if that student should be put on academic warning.

A student will be on warning if he/she has a GPA less than 1.5 for students with fewer than 30 semester hours credit, 1.75 for students with fewer than 60 semester hours credit, and 2.0 for all other students. Do the following:

1. Set up a Scanner object scan from the input file and a *PrintWriter* **outFile** to the output file inside the try clause (see the comments in the program). Note that you’ll have to create the *PrintWriter* from a *FileWriter*, but you can still do it in a single statement.
2. Inside the while loop add code to read and parse the input—get the name, the number of credit hours, and the number of quality points. Compute the GPA, determine if the student is on academic warning, and if so write the name, credit hours, and GPA (separated by spaces) to the output file.
3. After the loop close the *PrintWriter*.
4. Think about the exceptions that could be thrown by this program
 - a. A *FileNotFoundException* if the input file does not exist
 - b. A *NumberFormatException* if it can’t parse an int or double when it tries to – this indicates an error in the input file format.
 - c. An *IOException* if something else goes wrong with the input or output stream
 - d. Add a catch for each of these situations, and in each case give as specific a message as you can. The program will terminate if any of these exceptions is thrown, but at least you can supply the user with useful information.
5. Test the program. Test data is in the file *students.txt*.

An expected output file called “warning.txt” is shown below:

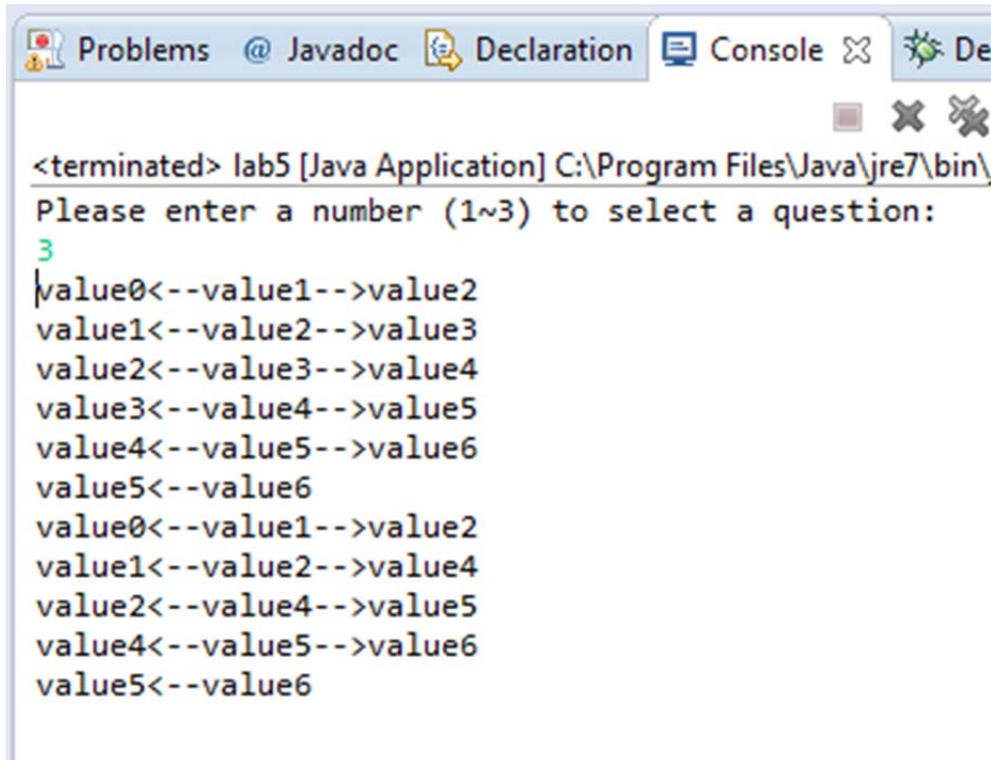
1	
2	Student on Academic Warning
3	
4	Jones
5	Walker
6	Street
7	Davis
8	Summers

Activity 3: Double linked list (5 marks)

Linked list are useful data structures, and the ability to link various elements together provides navigability. To gain a better understanding a custom doubled linked list is to be created:

1. A simple *Node* class to represent the nodes
 - a. Nodes are container for *Object* and contain references to the surrounding nodes;
 - b. A Node can only have a single link for its next and previous. It should not be possible to have two Nodes linking to the same Node.
 - c. A Node must have following attributes:
 - i. *id*: Integer
 - ii. *nextNode*: Node
 - iii. *previousNode*: Node
 - iv. *value*: Object
 - d. Must have following methods:
 - i. *setNext(Node node): void*
 - ii. *setPrevious(Node node): void*
 - iii. *setValue(Object value):void*
2. A simple *DoubleLinkedList* class that acts as the container for the Nodes
 - a. Holds a collection of Nodes
 - b. Must have following attributes:
 - i. *nextId*: Integer
 - ii. *nodes*: ArrayList <Nodes>
 - c. Must have following methods:
 - i. *insert(Node node, Integer index): void*
 - ii. *remove(Node node): void*
 - iii. *removeAt(Integer index): void*
 - iv. *isEmpty(): Boolean*
 - v. *clear(): void*
3. A simple *DoubleLinkedListIterator* class that can iterate over the *DoubleLinkedList* collection
 - a. Must have following attributes:
 - i. *collection*: DoubleLinkedList
 - b. Must have following methods:
 - i. *next(): Node*
 - ii. *previous(): Node*
 - iii. *value(): Object*
 - iv. *getTuple(): String*
 - v. *remove(): void*
4. In *q3*, a drive program is ready. 6 nodes are created at beginning and added into a double linked list. Compile your program and check the output. Now, delete node 3 and print out

such link. An expected result is shown below (the first part print out all elements store in our double linked list, the second one print out the result after we delete node 3).



```
<terminated> lab5 [Java Application] C:\Program Files\Java\jre7\bin\  
Please enter a number (1~3) to select a question:  
3  
value0<--value1-->value2  
value1<--value2-->value3  
value2<--value3-->value4  
value3<--value4-->value5  
value4<--value5-->value6  
value5<--value6  
value0<--value1-->value2  
value1<--value2-->value4  
value2<--value4-->value5  
value4<--value5-->value6  
value5<--value6
```