

VERSION B

Name: _____

1 Files and Exceptions

1.1

What is the output generated by the following code?

```
public class Stacker {
    static void stack(int n) {
        if(n == 0) throw new RuntimeException();
        System.out.println("before="+n);
        stack(n-1);
        System.out.println("after="+n);
    }
    public static void main(String[] args) {
        Integer a = null;
        try {
            stack(2);
        } catch(RuntimeException re) {
        }
    }
}
// output?
before=2
before=1
```

1.2

Fill in the missing code. The number of lines corresponds to the answer key. Your code may vary.

```
public class Show {
    public static void main(String[] args) {
        new Exception().printStackTrace();
        .....
        System.err.println("Exception printed!");
    }
}
// output:
java.lang.Exception
    at Show.main(Show.java:3)
Exception printed!
```

1.3

Fill in the missing code.

```

import java.io.FileWriter;
-----
import java.io.IOException;
import java.util.Scanner;
public class FilesWrite {
    public static void main(String[] args) throws IOException {
        String fileName = "/tmp/out.txt";
                new FileWriter(fileName);
        FileWriter fw = -----
        fw.write("Hello");
        fw.close();
    }
}

```

1.4

Fill in the missing code.

```

String in = "225-123-4567\n982-333-4444\n115-765-4321\n";
Scanner s = new Scanner(in);
while(s.hasNextLine()) {
                "(\d+)-(\d+)-(\d+)";
String pattern = -----
if(s.findInLine(pattern) != null) {
    MatchResult mr = s.match();
                Integer.parseInt(mr.group(1))
    int areaCode = -----;
                Integer.parseInt(mr.group(2))
    int exchange = -----;
                Integer.parseInt(mr.group(3))
    int number = -----;
    System.out.printf("(%03d)%03d-%04d\n", areaCode, exchange, number);
} else { System.out.println("Incorrect");
} s.nextLine();
}

```

// output:

```

(225)123-4567
(982)333-4444
(115)765-4321

```

2 Other

2.1

Fill in the missing code.

```

interface GetInt {
    public int value();
}
public class Anon2 {
    public static void main(String[] args) {
                new GetInt() {
GetInt gi = -----
    public int value() {
        -----
        return 33;
        -----
    }
    -----
};
}

```

```
        System.out.println("value="+gi.value());
    }
}
// output:
value=33
```

3 Linked Lists

4 Stacks and Queues

4.1

What is the difference between a Java **Set** and a **List**?

Elements in a **Set** have to be unique, while they don't need to be for a **List**.

4.2

Which interface directly extends the **Collection** interface and is implemented in the **ArrayList** class?

List

4.3

Name four interfaces within the Java collections framework.

List, Queue, Set, Collection, (SortedSet, NavigableSet, Deque, Iterable)

4.4

Which class is **Stack** inheriting from, and which interface is that class implementing?

Vector, which is implementing List.

4.5

Assume elements 2, 4, 6, 8 being put element-wise first onto a stack, taken out again, then put into a queue, taken out again, put onto a stack, and taken out again. In which order do you now have these elements, and which order were they after each step?

after stack: 8, 6, 4, 2 (reverse)

after queue: 8, 6, 4, 2 (reverse)

after stack: 2, 4, 6, 8 (original)

4.6

What is wrong with the following code to evaluate the top of the operators and numbers stack of an expression evaluator? How could you fix it?

```
static Stack<Double> numbers = new LinkedList<>();
static Stack<String> operators = new LinkedList<>();
static void evaluateTop() {
    Double n2 = numbers.pop();
    Double n1 = numbers.pop();
    String op = operators.pop();
    if (op.equals("-"))
        numbers.push(n1-n2);
    else if (op.equals("+"))
        numbers.push(n1+n2);
    else if (op.equals("*"))
        numbers.push(n1*n2);
    else if (op.equals("/"))
        numbers.push(n1/n2);
}
```

A **Stack** is not a subclass or implementation of **LinkedList**, but it's own class. Use **Stack** instead of **LinkedList**.

4.7

Complete the code within the `containsNot()` method, returning `false` if the doubly linked list contains an element with the value of `i`, and `true` otherwise.

```
class Node {
    Integer data;
    Node previous;
    Node next;
}

public class MyList {
    Node start;
    Node end;

    public boolean containsNot(Integer i) {
        if (start == null) return true;
        Node current = start;
        while (current != null) {
            if (i.equals(current.data)) return false;
            current = current.next;
        }
        return true;
    }
}
```