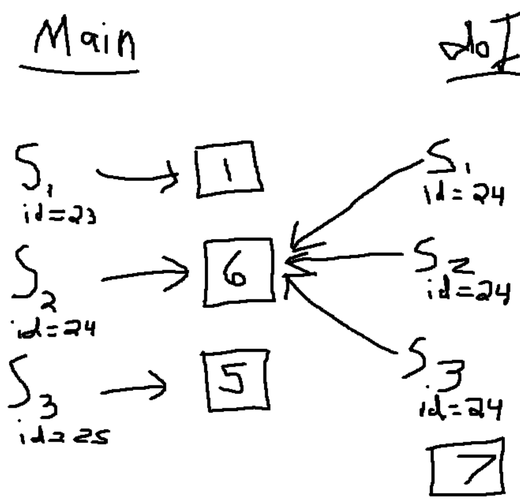
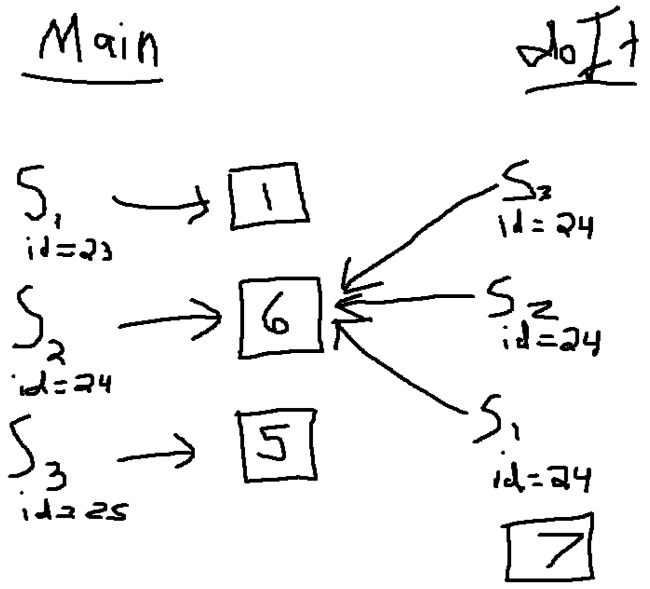
CompSci Extra Credit Chase Mulder

1.

Part 1:



Part 2:



2.

A) What is the output of the given code (no change)?
Explain why this path through the code is taken (e.g., which exceptions are thrown, why they are thrown, when they get handled, why the code proceeds to the next place, etc.)

1: illegalArgumentException because you can't create an ArrayList of Strings given -3 as input.
3: Because it's in the finally. The finally always runs.
4: Is printed because we don't throw any new exceptions in the illegalArgumentException block.
5: Is printed because the method() didn't throw any exceptions, so it does the whole try block.
9: Is always run because it's not in a try catch block in the main() method.

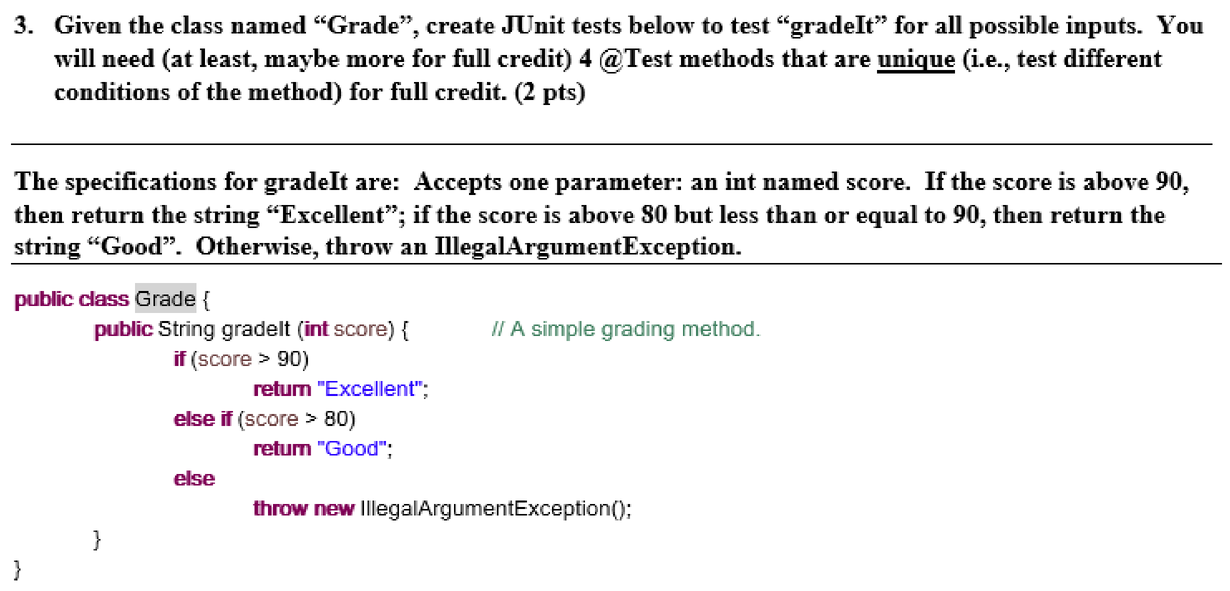
B) What is the output of the given code if replaced the 3rd line down with:
ArrayList<String> l = new ArrayList<String>(3);
Explain why this path through the code is taken (e.g., which exceptions are thrown, why they are thrown, when they get handled, why the code proceeds to the next place, etc.)

A: l.add("A") gets added to array 'l' at index 0, then 'A' is printed by (l.get(0));
2: l.get(1) throws indexOutOfBounds because index 1 doesn't exist yet, so caught exception 2
3: Because it's in the finally. The finally always runs.
No 4: Because a new exception was thrown from caught exception 2 and will be caught in the main() method.
No 5: Because in the try block in the main method an exception is thrown from caught exception 2 in method(), so method() ends after caught exception 2 throws an exception and the finally block prints 3. The code continues in main() having q.method() throw and exception and because it's in a try block the caught exception will be the first catch for an exception.
6: idexOutOfBoundsException is caught because caught exception 2 in method() throws e, so the first catch will catch it.
9: Is always run because it's not in a try catch block in the main() method.

C) What is the output of the given code with ArrayList<String> l = new ArrayList<String>(3); and remove the throw e; statement (the 13th line down)?
Explain why this path through the code is taken (e.g., which exceptions are thrown, why they are thrown, when they get handled, why the code proceeds to the next place, etc.)

A: l.add("A") gets added to l index 0, 'A' is printed by S.O.P(l.get(0));
2: l.get(1) throws indexOutOfBounds because index 1 doesn't exist yet, so caught exception 2.
3: Because it's in the finally. The finally always runs.
4: No exception is thrown in e, so no exception is thrown in method(), so '4' is printed at the end of method().
5: No exception is thrown in method(), so it the try block in main() is able to get through q.method(); with no exceptions, so the try block is able to S.O.P '5'.
9: Is always run because it's not in a try catch block in the main() method.

3.



3. Given the class named "Grade", create JUnit tests below to test "gradeIt" for all possible inputs. You will need (at least, maybe more for full credit) 4 @Test methods that are unique (i.e., test different conditions of the method) for full credit. (2 pts)

The specifications for gradeIt are: Accepts one parameter: an int named score. If the score is above 90, then return the string "Excellent"; if the score is above 80 but less than or equal to 90, then return the string "Good". Otherwise, throw an IllegalArgumentException.

```
public class Grade {
          public String gradeIt (int score) {          // A simple grading method.
                    if (score > 90)
                              return "Excellent";
                    else if (score > 80)
                              return "Good";
                    else
                              throw new IllegalArgumentException();
          }
}
```

```
@Test public void testNormalExcellent() {
Grade s = new Grade();
assertTrue(s.gradeIt(91).equals("Excellent"));
}

@Test public void testNotExcellent() {
Grade s = new Grade();
assertFalse(s.gradeIt(89).equals("Excellent"));
}

@Test public void testNormalGood() {
Grade s = new Grade();
assertTrue(s.gradeIt(88).equals("Good"));
}

@Test public void testNotGood() {
Grade s = new Grade();
assertFalse(s.gradeIt(91).equals("Good"));
}


@Test (expected = illegalArgumentException.class){
Grade s = new Grade();
s.grateIt(0);
}
```

4.

a) Is there an error in this statement (Y/N)? If yes, why is there an error? If not, why is this code "allowed"?
```
Exam e = new Exam();
```

No, java creates a default constructor for Exam even though Exam doesn't have a default constructor for Exam().

c) Is there an error in the following code sequence (Y/N)? If yes, why is there an error? If not, why is this code "allowed"?
```
Exam e;
e = new Midterm();
```

No error, Midterm extends Exam, so it is allowed to make an object of type Exam and apply it to Midterm.

d) Are there any methods that *still* need to be implemented in the class Midterm? If so, what are they, and why must they be implemented?

Yes, 'public void baz()' must be implemented because Midterm implements IMid, so the methods in IMid must be implemented.

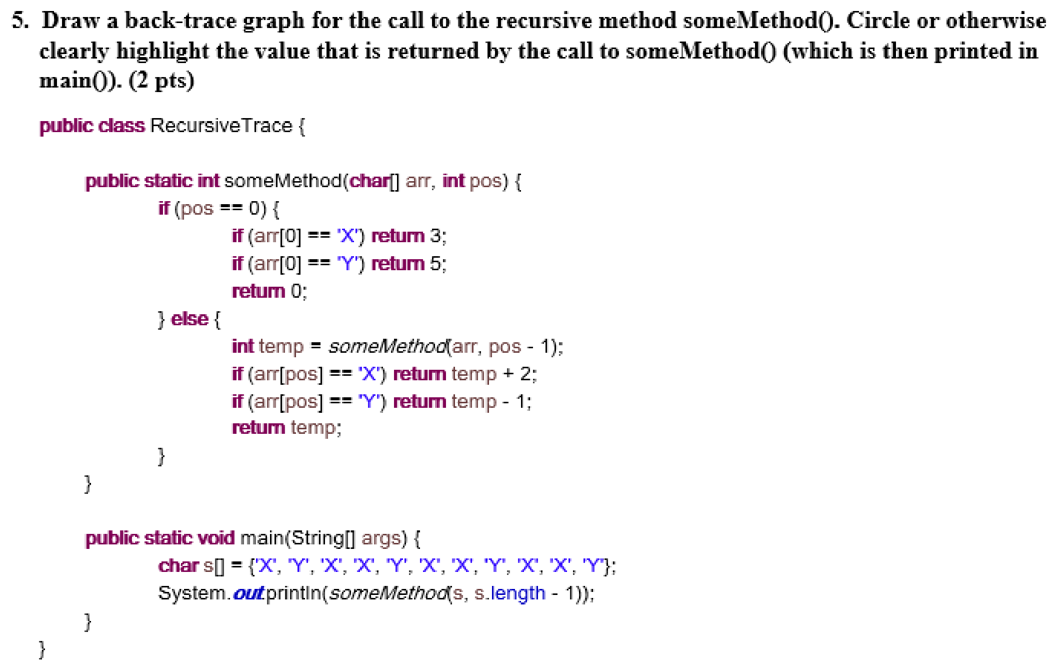
e) Which methods, if any, are polymorphic (i.e., overridden) in the class Midterm?

The method 'foo(int i)' in Midterm is overloaded by foo() in Exam because foo() in midterm has an input variable and the foo() in Exam has no input variable, thus overloaded. Foo() is not polymorphic.

f) Are there any shadowed variables? If yes, name them and explain why/how they are shadowed:

'private int total' in Exam is an unused variable.
'int total' in DriveInheritStuff is an unused variable.
'private int total' & 'int total' are not shadowed because their scopes don't overlap.

5.



5. **Draw a back-trace graph for the call to the recursive method someMethod(). Circle or otherwise clearly highlight the value that is returned by the call to someMethod() (which is then printed in main()). (2 pts)**

```
public class RecursiveTrace {

    public static int someMethod(char[] arr, int pos) {
            if (pos == 0) {
                    if (arr[0] == 'X') return 3;
                    if (arr[0] == 'Y') return 5;
                    return 0;
            } else {
                    int temp = someMethod(arr, pos - 1);
                    if (arr[pos] == 'X') return temp + 2;
                    if (arr[pos] == 'Y') return temp - 1;
                    return temp;
            }
    }

    public static void main(String[] args) {
            char s[] = {'X', 'Y', 'X', 'X', 'Y', 'X', 'X', 'Y', 'X', 'X', 'Y'};
            System.out.println(someMethod(s, s.length - 1));
    }
}
```

Index of Array: 0x 1y 2x 3x 4y 5x 6x 7y 8x 9x 10y

1. SOP(someMethod(s, 10))
2. pos10 = y ret -1
3. someMethod (s, 9)
4. pos9 = x ret + 2
5. someMethod (s, 8)
6. pos9 = x ret + 2
7. someMethod (s, 7)
8. pos8 = y ret -1
9. someMethod (s, 6)
10. pos7 = x ret + 2
11. someMethod (s, 5)
12. pos6 = x ret + 2
13. someMethod (s, 4)
14. pos5 = y ret -1
15. someMethod (s, 3)
16. pos4 = x ret + 2
17. someMethod (s, 2)
18. pos3 = x ret + 2
19. someMethod (s, 1)
20. pos2 = y ret -1
21. someMethod (s, 0)
22. pos0 = x ret + 3
23. the main() then prints ‘11’

someMethod() prints 11 because -1 + 2 + 2 -1 + 2 + 2 – 1 + 2 + 2 – 1 + 3 = 11