CS61BL Summer 2013 Midterm 1

Sample Solutions (Version A) + Common Mistakes

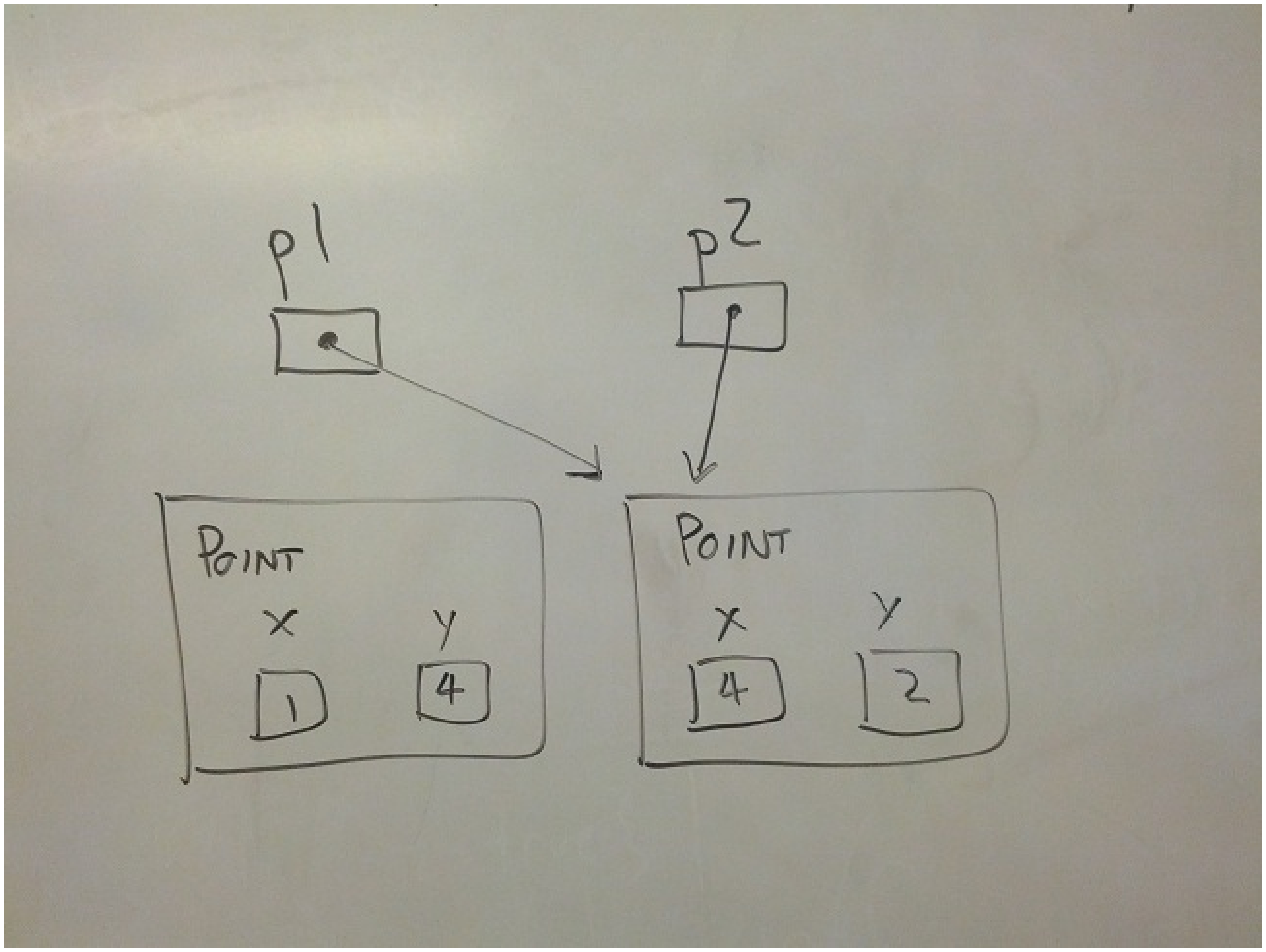
There were four versions of the exam; the version appears at the bottom of the front page. Versions differed in trivial ways. The answers below are for version A.

Question 0:

Each of the following cost you .5 on this problem: you earned some credit on a problem and did not put your five-digit on the page, you did not adequately identify your lab section, or you failed to put the names of your neighbors on the exam. The reason for this apparent harshness is that exams can get misplaced or come unstapled, and we want to make sure that every page is identifiable. We also need to know where you will expect to get your exam returned. Finally, we occasionally need to verify where students were sitting in the classroom while the exam was being administered.

Question 1:

Sample Solution:



Common Mistakes:

A common mistake was having an int reference (arrow) pointing to an int value when students should just have the int value inside the int reference box.

Question 2A:

Sample Solution:

assertEquals calls StringBuilder's .equals method, which only compares the contents of the reference variables s1 and s2, not the Strings they point to (since StringBuilder inherits the Object class's .equals method).

Common Mistakes:

Most students got this question correct. Possible mistakes include not mentioning one of the following:

- assertEquals calls the StringBuilder class's equals() method
- The StringBuilder class inherits the Object class's equals() method / the StringBuilder .equals() method compares object references

Question 2B:

Sample Solution:

No. .equals() has to be a method of the StringBuilder class. Any additions to the TestCase class wouldn't help.

Common Mistakes:

The most common mistake was stating that .equals() can be overridden in the StringBuilder class. (This would involve changing the Java library!) This was likely a misreading of the question statement. Students also tried to rewrite the assert statement; this was also caused by misreading or misinterpreting the question.

Question 3A:

Sample Solution:

```
public static void edit(StringBuilder line) {
    for(int k = 0; k < line.length(); ) {
        if (line.charAt(k) == '#') {
            if (k==0) {
                line.deleteCharAt(k);
            } else {
                line.deleteCharAt(k);
                line.deleteCharAt(k-1);
                // these two statements could also have been rewritten as
                // line.delete (k-1, k+1)
                k--;
            }
        } else if (line.charAt(k) == '%') {
            line.deleteCharAt(k);
            k++;
        } else {
            k++;
        }
    }
}
```

Common Mistakes:

> Missing a k++ or a k-- statement
> Having k += 2 instead of k++ when processing '%', which skips over the next character
> Forgetting the k==0 case when processing '#'

Question 3B:

Solution:

```
public void testNoSpecialChars() {
      StringBuilder sb = new StringBuilder("ABC");
      ExamString.edit(sb);
      String s = new String(sb);
      assertEquals(s, "ABC");
}
```

Common Mistakes:

Many students did not realize that you cannot compare a StringBuilder and a String, so for the solution to work you must create a String version of the edited StringBuilder object to pass to assertEquals.

Many students also struggled with how to call the static method (ie, they called sb.edit, or called edit() without ExamString in front of it). Additionally, many students thought that edit() would return a value, instead of simply editing the object it was passed. Covering how to call a static method, along with how to discern whether a method will modify an argument object vs creating

a new one, is recommended.
A lot of the errors simply came from people editing/converting StringBuilders and Strings in the wrong way, so a quick overview of how to change from a String object to a StringBuilder and vice versa is probably good.

Question 4a:
Solution:
```
while (arrayIndex < myStrings.size ( )
        && (myStrings.get (arrayIndex) == null
            || stringIndex >= myStrings.get (arrayIndex).length ( ))) {
      arrayIndex++;
      stringIndex = 0;
}
```

Common Mistakes:
- Many students didn't use a loop or recursion at all.
- A lot of people tried calling methods on String objects before checking if they were null.
- Didn't know what the empty string was.
- A lot of off-by-one errors when checking for legal indices.
- Some students didn't know how to check if a reference is null.

Question 4b:
Sample Solution:
1 point each was awarded for the following:

- null or empty at the start,
- null immediately followed by empty or vice versa,
- all nulls and empties.

All other test cases were viewed either as duplicating some other listed test case, or as providing much less information compared to the tests in the list above. The purpose of this problem was not just to provide three test cases but to provide the three best test cases. In general you needed each of these three cases in separate tests to get full credit; however, grading was more lenient if you mentioned in writing that you were intending to test each of them.

Question 4c:

Sample solution (fast)

```
public boolean isOK() {
     try {
          myStrings.get(arrayIndex).charAt(stringIndex);
          return true;
     } catch (Exception e) {
          return !hasMoreChars();
     }
}
```

Other solution:

```
public boolean isOK() {
    if (arrayIndex >= 0 && arrayIndex < myString.size()) {
        if (myString.get(arrayIndex) != null) {
            if (stringIndex < myString.get(arrayIndex).length()
                && stringIndex >= 0) {
                return true;
            }
        }
    }
    return (!hasMoreChars());
}
```

Common errors:

- Not checking lower bound before indexing array/string
- Assuming that if hasMoreChars is false, isOK must be false
- Not checking if string was null before calling methods on it
- being generally lost (for loops, initIterator, next()...)

Question 5:

Solution:

```
try {
    myYear = intScanner.nextInt();
}
catch (NullPointerException e) {
    throw new IllegalArgumentException("string is null");
}
catch (NoSuchElementExecption e) {
    throw new IllegalArgumentException("the string only has whitespace");
}
catch (InputMismatchException e) {
    throw new IllegalArgumentException("the string contains a non-integer");
}
if (intScanner.hasNext()) {
    throw new IllegalArgumentException("the string contains more than one integer");
}
else if (myYear < 1900 || myYear > 1999) {
    throw new IllegalArgumentException("year out of range");
    // some versions had different ranges
}
```

Common mistakes:

1) Confusion about how scanner takes in input. If s = "2056 2010 abcd" and you put it into scanner, nextInt() will return 2056. Calling nextInt() again will return 2010, and calling nextInt() again will throw an InputMismatchException error. 2056, 2010, and "abcd" are the 3 tokens in the string. If I call intScanner.hasNext() when at the "abcd" token, it will return false because there are no more tokens left in the string.

2) Not realizing that scanner throws exceptions that can be caught. You can try checking the string manually (doing something like if s == null throw exception) but most people who did this missed some kind of exception. Instead it's a lot easier to let scanner tell you if some kind of exception occurred, so you can catch that exception and throw IllegalArgumentException.

-- if s == null, or if s == "   ", or s == "abc" or whatever, when you call intScanner.nextInt() it'll look at s and see if it has any error, and will throw the appropriate exception if it does. You don't need to have multiple test cases of s because using scanner does this for you.

3) Not throwing IllegalArgumentExceptions like the question asked. Also doing System.out.println("error message") instead of throwing an actual error. In addition, we docked you off for doing something like

```
System.out.println("error message");
throw new IllegalArgumentException();
```

instead the error message should go inside the arguments for IllegalArgException -

```
throw new IllegalArgumentException("error message");
```

4) You needed to catch all the possible errors to get full credit. This includes the 3 exceptions thrown by scanner, checking the bounds of the year (read the first sentence of the problem!), and making sure there is only one token (one year) in the string. Do that by calling intScanner.hasNext() to make sure there are no more tokens after the first one.