COSC 480B, Midterm
October 2023

Name: _____

Write your name. Do not start the exam until instructed to do so.

You have 75 minutes to complete this exam.

There are 4 questions and a total of 58 points available for this exam. Don't spend too much time on any one question.

If you want partial credit, show as much of your work and thought process as possible.

If you run out of space for answering a question, you can continue your answer on one of the blank pages at the end of the exam. If you do so, be sure to indicate this in two places: (1) below the question, indicate which blank page contains your answer, and (2) on the blank page, indicate which question you are answering.

The last page of the exam contains documentation for string and list methods.

Question	Points	Score
1	13	
2	14	
3	8	
4	23	
Total:	58	

1. Consider the following functions: `payBill`, `payZelle`, `atmWithdrawal` and `updateBalance` of the `BankAccount` class:

```
// constructor and other functions

public void updateBalance(float amount){
    assert amount >= 0:"Negative amounts are not allowed";
    float oldBalance = accountBalance;
    // code to change accountBalance
    assert accountBalance == oldBalance - amount:"Bug in updateBalance";
    assert accountBalance >= 0:"Account balance must always be positive";
}

public void payBill(float amount){
    updateBalance(amount);
}

public void payZelle(float amount){
    assert amount > 0: "The amount must be a value greater than 0";
    assert amount <= 1000: "The amount must be a value upto 1000";
    updateBalance(amount);
    assert accountBalance >= 20:
            "Must have at least $20 left in your account";
}

public void atmWithdrawal(int amount){
    assert amount % 20 == 0: "Dollar amounts must be a multiple of 20";
    assert amount <= 300: "The amount must be a value upto 300";
    updateBalance(amount);
    assert accountBalance > -1:"Account balance must always be positive";
}
```

(a) (4 points) Are the pre- and post-conditions of `payBill` more or less inclusive than `updateBalance`? Given the dependencies between them, is there a possibility of failure due to the pre- or post conditions or both?

Solution:

```
pre: more inclusive, which means failure when negative
post: more inclusive, which means no failure
```

(b) (4 points) Are the pre- and post-conditions of `payZelle` more or less inclusive than `updateBalance`? Given the dependencies between them, is there a possibility of failure due to the pre- or post conditions or both?

Solution:

```
pre: less inclusive, no failure
post: less inclusive, which means failure for values 0 to 20
```

(c) (4 points) Are the pre- and post-conditions of `atmWithdrawal` more or less inclusive than `updateBalance`? Given the dependencies between them, is there a possibility of failure due to the pre- or post conditions or both?

Solution:

```
pre: neither less or more, failure for negative values
post:  more inclusive, which means no failure
```

(d) (1 point) What would be a good invariant for the `BankAccount` class?

Solution:

```
assert accountBalance >= 0
```

2. You are tasked with testing the method `getHonors(float gpa)` that returns `Summa cum laudae` for a gpa greater or equal to 3.9, `Magna cum laudae` for a gpa greater or equal to 3.7, `Cum laudae` for a gpa greater or equal to 3.5, and `No honors` otherwise. If the gpa is an invalid value (less than 0 or greater than 4), the function throws an `IllegalArgumentException`.

   (a) (5 points) What are all the partitions for this problem? Specify one testcase per each partition.

Solution:

```
Invalid: -1.0 or 5.0
Valid
- summa 3.95
- magna 3.8
- cum 3.6
- no honors 3.0
```

   (b) (5 points) What are important boundaries for this problem? Specify enough testcases per each boundary (`on-point`, `off-point`, `in-point`, `out-point`).

Solution:

```
Invalid and No honors: 0.0 - on point, -0.1 is off point
Invalid and summa: 4.0 on point and 4.1 is off point
Summa and magna: 3.9 on point and 3.89 is off point
Magna and cum: 3.7 on point and 3.69 is off point
Cum and no-honors: 3.5 is on-point and 3.49 is off point
```

(c) (2 points) The code for this function can be found on the code snippets handout. Take a look at the code, indicate how many testcases are enough for structure-based testing (in the least) and explain why.

Solution:

6 because there are 6 different conditions chained together

(d) (2 points) Explain how you would implement property-based testing for this problem (that is, how the input and expected output would be generated).

Solution:

For each of the 6 conditions generate random data in the specific intervals:
<0
>4
[3.9,4]
[3.7,3.9)
[3.5,3.7)
[0,3.5)
The first 2 expect IllegalArgumentException, while the others expect their respective string
(Summa, Magna, Cum, No honors)

3. You are tasked with testing the method `getComplementMod5(int a)` that returns the positive complement modular 5 of `a`. Note that a complement can take similar values as the modulus or remainder, that is starting from 0 to n-1, where n is the divisor.

   (a) (3 points) What are all the partitions for this problem? Specify one testcase per each partition.

Solution:

```
Invalid: -1 for a
Valid: a is
- 0
- 1
- 2
- 3
- 4
```

   (b) (1 point) What are important boundaries for this problem? Specify enough testcases per each boundary (`on-point`, `off-point`, `in-point`, `out-point`).

Solution:

```
Invalid and Valid: 0 and -1 for a
```

(c) (2 points) The code for this function can be found on the code snippets handout. Take a look at the code, indicate how many testcases are enough for structure-based testing (in the least), list them and explain why.

Solution:

```
2: valid and invalid
```

(d) (2 points) Explain how you would implement property-based testing for this problem (that is, how the input and expected output would be generated).

Solution:

```
For each of the 6 conditions generate random data:
<0
with a filtering predicate
multiple of 5
multiple of 5 + 1
multiple of 5 + 2
multiple of 5 + 3
multiple of 5 + 4
```

4. You are tasked with testing the method `getCheapestPrice` that receives two arrays representing the seats, as well as an integer which indicates the number of requested seats. It calculates the cheapest total price for these seats. If the total price is greater than 100, we apply a discount.

   The documentation and code for this method can be found on the code snippets handout.

   (a) (5 points) What are all the individual partitions for the input and output for this problem?

   Solution:

```
- prices and taken
null
empty
single or multiple (repeated prices)
- numberOfSeats
negative
positive
- prices vs taken lengths
not equal (less and more)
equal
- numberOfSeats vs number of truths in taken
less
equal
greater
```

(b) (9 points) What are the most important combined partitions for this problem? Specify testcases for each partition.

Solution:

```
null x 2 -> IAC
diff sizes x 2 -> IAC
negative seats -> IAC
empty x 2 -> 0
number of seats less than truths
number of seats equals to truths
number of seats greater than truths
equal prices
total over 100
```

(c) (4 points) Take a look at the code, indicate how many testcases are enough for structure-based testing (in the least), list them and explain why.

Solution:

```
2 for conditions
2 for other IAC conditions
3 to cover for
and maybe 1 extra one for the last condition
7 or 8 in the minimum
```

(d) (5 points) What test suite (s) achieve 100% MC/DC for the expression `A | (B & C)`? Make sure to include all the steps, including drawing the truth table and clearly indicating which test pairs can cover which individual condition.

Solution:

```
A - {2,6} {3,7} {4,8}
B - {5,7}
C - {5,6}

All:
2,5,6,7
or
3,5,6,7
```