C212 Midterm Exam (80 points)
Oct 11, 2023

Please read these directions before starting your exam.

This is a closed-note exam aside from your one page of notes, double-sided. You may not use any electronic devices to complete this exam, nor can you communicate with anyone besides the proctors and professor. *If you are caught cheating, you will receive an F in the course.*

For any question, unless specified otherwise, you may use any class without a corresponding `import`. E.g., if you want to use `HashMap`, you do not need to also import `java.util.HashMap`.

Unless otherwise stated, you do not need to spell out the "full design recipe", i.e., write the signature, documentation comments, and tests. Of course, doing so may aid you in your solution.

If you find a mistake, please raise your hand and let one of the proctors know; we will determine whether or not this is the case.

When you are finished, turn in your exam and notes sheet if you have one, then quietly exit.

You have 75 minutes to complete the exam, but it is designed to take only 60 minutes.

Good luck!

Question	Points	Score
1	20	
2	20	
3	20	
4	20	
Total:	80	

Name: _____

IU Email: _____

Section (Sam/Joshua): _____

1. (20 points) Design the `computeDiscount(double itemCost, int age, boolean isStudent)` method that computes a discount for some item based on their age and student status according to the following criteria:

   - If $age < 18$, apply a 20% discount.
   - If $18 \leq age \leq 25$ and they are a student, apply a 25% discount. If they are not a student, do not apply a discount.
   - If $age \geq 65$ and they are a student, apply a 30% discount. If they are not a student, apply a 15% discount.
   - All other cases should not have a discount applied.

   Your method should return the total cost of the item after applying the discount. In designing this method, follow the template from class; write the signature, purpose statement, testing, and *then* do the implementation. You should probably use simple numbers for the `itemCost` so you can calculate the discounts in your head. The skeleton code is on the next page.

```
import static Assertions.assertAll;
import static Assertions.assertEquals;

class ComputeDiscountTester {

  @Test
  void computeDiscountTest() {









  }
}

class ComputeDiscount {

  /**
   *
   *
   *
   *
   */
  _________ _________ computeDiscount(double itemCost, int age, boolean isStudent) {









  }
}
```

2. (20 points) This question has three parts.

   (a) *(6 points)* Design the *standard recursive* `countdown` method, which receives an `int` $n \geq 0$ and returns a `String` containing a sequence of the even numbers from n down to 0 inclusive, separated by commas.

```
countdown(10) => "10,8,6,4,2,0"
countdown(23) => "22,20,18,16,14,12,10,8,6,4,2,0"
countdown(0)  => "0"
```

   (b) *(7 points)* Design the `countdownTR` and `countdownTRHelper` methods. The former acts as the driver to the latter; the latter solves the same problem as `countdown` does, but it instead uses tail recursion. Remember to include the relevant access modifiers!

(c) *(7 points)* Design the `countdownLoop` method, which solves the problem using either a `while` or `for` loop.

3. (20 points) Design the `moreThanThree` method that, when given an `int[]` A, returns a new `HashSet<Integer>` of values containing those values from A that occur strictly more than three times. **You cannot use the Stream API.** In designing this method, follow the template from class; write the signature, purpose statement, testing, and *then* do the implementation.

```
import static Assertions.assertAll;
import static Assertions.assertEquals;

class MoreThanThreeTester {

  @Test
  void moreThanThreeTest() {









  }
}
```

```
import java.util.*; // Import all necessary collections.

class MoreThanThree {

  /**
   *
   *
   */
  ____________ ____________________ moreThanThree(__________ __________) {









  }
}
```

4. (20 points) Oh no! Sam's cat, Marmalade, has scratched part of this exam away and we need you to fix the missing code. Fill in the blanks to complete this generic method implementation. Additionally, write at least two examples where each example uses a different key type. You can write an instance of a `LinkedHashMap` as {<k_1, v_1>, <k_2, v_2>, ..., <k_n, v_n>} in your examples to compensate for time.

```
import static Assertions.assertAll;
import static Assertions.assertEquals;

class FindMaxStringTester {

  @Test
  void findMaxStringTest() {









  }
}

import java.util.LinkedHashMap;

class FindMaxString {

  /**
   * Finds the maximum string in a map of some keys
   * of an arbitrary type to strings.
   */
  static _________ ____ findMaxString(LinkedHashMap<_____, String> map) {
    String max = "";
    for (____ t : map.keySet()) {
      if (_______________________________________) {
        max = map.get(_____);
      }
    }
    return max;
  }
}
```

Scratch work

Scratch work