C212 Midterm Exam Makeup (80 points)
Mar 06, 2024

C212 Midterm Exam Rubric

1. (20 points) Design the `double computeBonusPay(double baseSalary, int yearsOfService, boolean achievedTarget, double salesAmount, double targetSales)` method that calculates the amount of bonus pay (**only the bonus pay**) given to an employee under the following conditions:

   - A base bonus rate of 10% of the base salary is given to employees who have achieved their sales target.
   - Employees with more than 5 years of service receive an additional 5% bonus of their base salary.
   - If the sales amount exceeds the target sales by more than 50%, the employee receives an additional bonus of 25% of the base salary.
   - If the employee has not achieved their sales target, they receive a flat bonus of 2% of their base salary, regardless of sales amount or years of service.
   - The total bonus amount is reduced by a flat tax rate of 25%.

   In designing this method, follow the design recipe from class; write the signature, purpose statement, testing, and *then* do the implementation. You should probably use simple numbers for the inputs so you can calculate the values in your head. You must write tests that *fully cover* all possible kinds of inputs.

Solution.

```
@Test
void testComputeBonusPay() {
  assertAll(
    () -> assertEquals(150, Problem1.computeBonusPay(10000, 6, 4, 5)),
    () -> assertEquals(750, Problem1.computeBonusPay(10000, 1, 5, 5)),
    () -> assertEquals(1125, Problem1.computeBonusPay(10000, 6, 5, 5)),
    () -> assertEquals(2625, Problem1.computeBonusPay(10000, 1, 15, 5)),
    () -> assertEquals(3000, Problem1.computeBonusPay(10000, 6, 15, 5)));
}

static double computeBonusPay(double baseSalary, int yearsOfService,
                              double salesAmount, double rgetSalesAmount) {
  double bonusPay = 0;
  if (salesAmount >= targetSalesAmount) {
    bonusPay += baseSalary * 0.10;
    if (yearsOfService > 5) {
      bonusPay += baseSalary * 0.05;
    }
    if (salesAmount * 0.5 > targetSalesAmount) {
      bonusPay += baseSalary * 0.25;
    }
  } else {
    bonusPay = baseSalary * 0.02;
  }
  return bonusPay * 0.75;
}
```

2. (25 points) This question has three parts.

   You are developing a system to help monitor and reduce electricity usage in households. The system calculates the total energy consumption based on the number of hours each appliance is used daily. Different appliances have different power ratings, which affects their energy consumption.

   Solution.

   (a)
```
static double calculateEnergy(double[] P, double[] H) {
  return calculateEnergyHelper(P, H, 0) * 0.001;
}

private static double calculateEnergyHelper(double[] P, double[] H, int i) {
  if (i == P.length) {
    return 0;
  } else if (H[i] == 0) {
    return calculateEnergyHelper(P, H, i + 1);
  } else {
    return P[i] * H[i] + calculateEnergyHelper(P, H, i + 1);
  }
}
```

(b)
```
static double calculateEnergyTR(double[] P, double[] H) {
  return calculateEnergyTRHelper(P, H, 0, 0) * 0.001;
}

private static double calculateEnergyTRHelper(double[] P, double[] H, int i, double e) {
  if (i == P.length) {
    return e;
  } else if (H[i] == 0) {
    return calculateEnergyTRHelper(P, H, i + 1, e);
  } else {
    return calculateEnergyTRHelper(P, H, i + 1, e + P[i] * H[i]);
  }
}
```

```
static double calculateEnergyLoop(double[] P, double[] H) {
  double e = 0;
  for (int i = 0; i < P.length; i++) {
    if (H[i] != 0) {
      e += P[i] * H[i];
    }
  }
  return e * 0.001;
}
```

3. (15 points) Design the `nthCommonWordLength` method that, when given a *non-empty* list of strings W and an `int` n, returns the length of the n^{th} most common word length in the list. Assume $1 \leq n \leq$ the number of unique word lengths in W. We provide four examples, but you **cannot** use these in your tests. **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. You can use abbreviated array notation, e.g., $[1, 2, 3, ..., n]$, in your tests instead of Java code.

   The skeleton code is on the next page.

   This problem is harder than it looks at first glance. As a hint, compute a map of word lengths to a list of words that have that length. Then, find the most-common word length and remove that key/value pair. Keep doing this until n is zero.

```
nthCommonWordLength(["apple", "bat", "cat", "dog", "elephant", "fish", "goat"], 2) => 4
nthCommonWordLength(["hello", "world", "java", "python", "code"], 1) => 4
nthCommonWordLength(["one", "two", "three", "four", "five", "six"], 3) => 5
nthCommonWordLength(["cb", "bc", "a", "abc", "abc", "abc", "abc", "abc"], 2) => 2
```

   Solution.

```
static int nthCommonWordLength(String[] words, int n) {
  Map<Integer, List<String>> M = new TreeMap<>();
  for (String word : words) {
    int len = word.length();
    if (!M.containsKey(len)) {
      M.put(len, new ArrayList<>());
    }
    M.get(len).add(word);
  }

  int ke = -1;
  while (n > 0) {
    // Remove the key/value pair that has the largest value.
    int lv = -1;
    for (int k : M.keySet()) {
      if (M.get(k).size() > lv) {
        lv = M.get(k).size();
        ke = k;
      }
    }
    M.remove(ke);
    n--;
  }
  return ke;
}
```

4. (20 points) Oh no! Joshua's cat, Nebraska, scratched away part of this exam and we need you to add the missing code. Fill in the blanks to complete this method implementation. Additionally, write at least three examples, one for each case (you need to figure out what the cases are). You can use abbreviated set notation, e.g., $\{1, 2, 3, ..., n\}$, in your tests instead of Java code.

   Solution.

```
import java.util.List;
import java.util.ArrayList;
import java.util.PriorityQueue;

class MergeKSortedLists {

  /**
   * Merges k sorted lists into one sorted list and returns it.
   * The input is a list of lists, where each inner list is sorted in ascending order.
   * @param lists - a list containing k sorted lists of integers.
   * @return a single list containing all elements from the k lists in sorted order.
   */
  static <T> List<T> mergeKSortedLists(List<List<T>> lists) {
    PriorityQueue<T> pq = new PriorityQueue<>();
    for (List<T> list : lists) {
      for (T value : list) {
        pq.add(value);
      }
    }

    List<Integer> result = new ArrayList<>();
    while (!pq.isEmpty()) {
      result.add(pq.poll());
    }
    return result;
  }
}
```