**Review Problems**

Below you are given problem statements followed by multiple solutions to each problem. Assume that the solutions are correct; ignore any small deviations in behavior. Also assume that any missing helper functions are defined in the obvious way. Finally, ignore stylistic differences in naming. Instead, focus on the *structure* of the solutions.

For each problem, rank the solutions in order (from most to least) of your preference. Explain why you picked that ordering. You are allowed to have ties.

The solutions are labeled **A**, **B**, etc. Indicate your ordering with the labels and >, using commas for ties. For instance, the ordering B > A, C > D means you liked B the most, followed by A and C (tied), followed by D.

Remember to explain your choice!

**1 Rainfall**

Design a program called rainfall that consumes a list of real numbers representing daily rainfall readings. The list may contain the number -999 indicating the end of the data of interest. Produce the average of the non-negative values in the list up to the first -999 (if it shows up). There may be negative numbers other than -999 in the list (representing faulty readings). Assume that there is at least one non-negative number before -999.

Example:

rainfall takes in an ArrayList containing(1, -2, 5, -999, 8) and returns 3

|  |
| --- |
| Solution **A**:  import java.util.ArrayList ;  public class Rainfall1 {  public double rainfall (ArrayList<Double> data) {  double total = 0;  int days = 0 ;    for (double d : data) {  if (-999 == d) {  return (total / days) ;  }  else if (d >= 0) {  total = total + d ;  days = days + 1 ;  }  }  return (total / days) ;  }  }    Solution **B**:  public class Rainfall2 {  public double sumRelevant (ArrayList<Double> data) {  double total = 0 ;    for (double d : data) {  if (-999 == d) {  return(total);  }  else if (d >= 0) {  total = total + d;  }  }  return total;  }    public int countRelevant (ArrayList<Double> data) {  int count = 0 ;    for (double d : data) {  if (-999 == d) {  return(count);  }  else if (d >= 0) {  count = count + 1;  }  }  return count;  }    public double rainfall (ArrayList<Double> data) {  return(sumRelevant(data) / countRelevant(data));  }  }      Solution **C**:  public class Rainfall3 {  public ArrayList<Double> cleanData (ArrayList<Double> data){  ArrayList<Double> cleanData = new ArrayList<Double>();    for (double d : data) {  if (-999 == d) {  return(cleanData);  }  else if (d >= 0) {  cleanData.add(d);  }  }  return cleanData;  }    public double sum(ArrayList<Double> data) {  double total = 0;  for (double d : data) {  total = total + d;  }  return total;  }    public double rainfall (ArrayList<Double> data) {  ArrayList<Double> relevant = cleanData(data) ;  return(sum(relevant) / relevant.size());  }  } |

**2 Length of Triples**

Design a program called maxTripleLength that consumes an ArrayList of strings and produces the length of the longest concatenation of three consecutive elements. Assume the input contains at least three strings. Assume we are also given

public class Triple {

private String \_a, \_b, \_c;

Triple (String a, String b, String c) {

\_a = a ;

\_b = b ;

\_c = c ;

}

public int totalLen () {

return (a.length() + b.length() + c.length());

}

}

|  |
| --- |
|  |

Example:

|  |
| --- |
| maxTripleLength takes in an ArrayList containing ("a", "bb", "c", "dd") and returns 5 |
| Solution **A**:  public class MaxTripleOne {    public ArrayList<Triple> BreakIntoTriples (ArrayList<String> args) {  ArrayList<Triple> TripList = new ArrayList<Triple>();    for (int i= 0; i < args.length-2; i++) {  TripList.add(new Triple(args[i], args[i+1], args[i+2]));  }    return(TripList);  }    public int maxTripleLength (ArrayList<String> args) {  int maxLength = 0 ;    for (Triple t : BreakIntoTriples(args)) {  if (maxLength < t.totalLen()) {  maxLength = t.totalLen();  }  }  return(maxLength);  }  }    Solution **B**:  public class MaxTripleTwo {  public int maxTripleLength (ArrayList<String> args) {  int maxLength = 0;  String concatenated = "";    for (int i = 0; i < args.length - 2; i++) {  concatenated = args[i] + args[i+1] + args[i+2];  if (maxLength < concatenated.length()) {  maxLength = concatenated.length();  }  }  return(maxLength);  }  } |
|  |

**3 Shopping Discount**

An online clothing store applies discounts during checkout. A shopping cart is an ArrayList of the items being purchased. Each item has a name (a string like “shoes”) and a price (a real number like 12.50). Design a program called checkout that consumes a shopping cart and produces the total cost of the cart after applying the following two discounts:

* if the cart contains at least 100 worth of shoes, take 20% off the cost of all shoes (match only items whose exact name is "shoes")
* if the cart contains at least two hats, take 10 off the total of the cart (match only items whose exact name is "hat")

Assume the cart is represented as follows:

|  |
| --- |
| public class CartItem {  private String \_name;  private double \_cost;    CartItem(String n, double c) {  \_name = n;  \_cost = c;  }    public boolean isShoe() {  return \_name.equals("shoes");  }    public boolean isHat() {  return \_name.equals("hat");  }  } |
|  |

Example:

|  |
| --- |
| checkout takes in an ArrayList containing (CartItem("shoes", 25), CartItem("bag", 50), CartItem("shoes", 85), CartItem("hat", 15)) and returns 153  Solution **A**:  public class ShoppingOne {  public double checkout (ArrayList<CartItem> cart) {  double shoeCost = 0 ;  double shoeDiscount = 0 ;  for (CartItem ci : cart) {  if (ci.isShoe()) {  shoeCost = shoeCost + ci.cost ;  }  }  if (shoeCost >= 100) {  shoeDiscount = shoeCost \* .20;  }    int hatCount = 0 ;  double hatDiscount = 0;  for (CartItem ci : cart) {  if (ci.isHat()) {  hatCount = hatCount + 1;  }  }  if (hatCount >= 2) {  hatDiscount = 10;  }    double cartTotal = 0 ;  for (CartItem ci : cart) {  cartTotal = cartTotal + ci.cost;  }    return (cartTotal - shoeDiscount - hatDiscount);  }  }    Solution **B**:  public class ShoppingTwo {    public double checkout (ArrayList<CartItem> cart) {  double shoeCost = 0;  double hatCount = 0;  double cartTotal = 0;  double shoeDiscount = 0;  double hatDiscount = 0;    for (CartItem ci : cart) {  cartTotal = cartTotal + ci.cost;  if (ci.isShoe()) {  shoeCost = shoeCost + ci.cost;  }  if (ci.isHat()) {  hatCount = hatCount + 1;  }  }    if (shoeCost >= 100) {  shoeDiscount = shoeCost \* .20;  }  if (hatCount >= 2) {  hatDiscount = 10;  }  return (cartTotal - shoeDiscount - hatDiscount);  }  } |
|  |