Name: Key UTA ID:

Instructions:

1. The test is worth 100 points. The point value of each question is given with the question. There are also extra credit questions at the end.
2. Read all of the instructions for each question and answer what is asked. Do not write down random stuff if you don’t know the answer.
3. All questions that have the same question number are related to each other. However, they are not all necessarily dependent on each other so you can skip around if needed.
4. The test is open book and open notes for all printed and hand-written material. You may NOT bring an electronic book or any electronic device to use during the test (no computer, no smart phone, etc.) You may use as much printed or written material as desired including copies of code examples.
5. You will write your answers on the test pages. If additional space is needed, you may use the backs or bottoms of the pages or extra paper. Please make a note on the test page whenever your answer continues onto another page and indicate where the answer is. Please staple all extra pages to your test when you turn it in.
6. Please write legibly and large enough to read easily. Your writing should be readable if the test is sitting on a desk in front of me. I am not looking for perfect handwriting but it does need to be legible. I will deduct points if your answers are much more difficult to read than those of the general student.
7. For multiple choice questions, circle the letter of your answer choice. Circle only one choice for your answer unless the instructions direct otherwise.
8. For short answer and code questions, answer the question as best you can but do not write random stuff that does not relate to the question being asked. If your answer is not related to the question and appears to be copied from some source just to write something down, points may be deducted as a penalty.
9. If you have a question during the test, please raise your hand. The TA and I will be available to come hear your question. Sometimes we may not be able to answer your question because it gives you too much information but you should always ask.
10. You have 1 hour and 20 minutes to complete the test.

Y. Answer the questions below based on this problem statement and the code below:

The customer MavBikes would like to have an app developed that keeps track of loaner bikes that are stored on a college campus. There are racks of bikes on each campus. Each bike has unique ID number between 1000 and 999,999 and a campus registration number of 4 digits. If a bike is currently rented, then bike also has an associated first name, last name, cell number, and e-mail address. There is also a current cost for any rental.

final int MAXCAMPUSES = 250; // Max number of unique participating campuses

final int UTACAMPUSID = 76019; // Identifier for unique UTA campus

final int MAXRACKS = 15; // Max racks on a campus

final int MAXBIKESPERRACK = 12; // Max bikes in a rack

final int MAXBIKEID = 999; // Max valid ID number for a bike

final int MINBIKEID = 10; // Min valid ID number for a bike

final int MAXCAMPUSREG = 9999; // Max value for a bike's campus registration

final int MINCAMPUSREG = 1000; // Min value for a bike's campus registration

final int MAXCUSTOMERS = 500;

final double COSTPERQUARTERHR = .25;

// bikelist - campus ID, # racks, # bikes per rack, bike ID, campus regisration #

int[][] bikeList = new int[MAXCAMPUSES][5];

// bikeCustomers - first name, last name, cell phone, e-mail, campus ID, customer ID

String [][] bikeCustomers = new String[MAXCUSTOMERS][6];

// bikeCosts - customer ID, current number of quarter hours elapsed, current cost (dd.cc),

double [][] bikeCosts = new double[MAXBIKEID][5]; // month and day as MM.DD, year as YYYY

// assume that all data in the arrays is stored in the same fashion as the data shown below and that

// there are currently 150 bikes in the whole bikeList and 250 customers in the bikeCustomer list

bikeList[145][0] = 76019; // campus ID number - this is the UTA number (zip code)

bikeList[145][1] = 6; // number of bike racks at UTA

bikeList[145][2] = 8; // number of bikes in each bike rack

bikeList[145][3] = 145; // the bike ID is the MavBikes number for this bike

bikeList[145][4] = 338; // the campus registration number is the number for this bike in UTA's system

bikeCustomers[127][0] = "09260504"; // customer ID in MavBikes system

bikeCustomers[127][1] = "Izzard"; // customer last name

bikeCustomers[127][2] = "Eddie"; // customer first name

bikeCustomers[127][3] = "8174709988"; // customer phone number as a string

bikeCustomers[127][4] = "eddieizzard@beesknees.com"; // customer e-mail

bikeCustomers[127][5] = "76019"; // campus the customer is on

bikeCosts[145][0] = 9260504; // customer ID

bikeCosts[145][1] = 20; // current number of quarter hours the customer has rented the bike, 0 = not rented

bikeCosts[145][2] = -1;// payment owed after a rental, -1 indicates that the bike is currently rented

bikeCosts[145][3] = 4.06; // rental date month and day as MM.dd

bikeCosts[145][4] = 2018; // rental date year as YYYY

bikeCosts[145][5] = 145; // the bike ID is the MavBikes number for this bike

1.a) Assuming that every bike ride requires a $2.50 non-refundable deposit and uses the quarter mile (hour) cost defined as a constant in the code, write a line(s) of Java code that will calculate exactly how many quarter hours any bicycle was last rented for. Use the bikeCost array values and the constant to calculate this and store the result in a new variable called lastRideLength. {8 points}

So if a ride costs Kosts, then Kosts = 2.50 + (QuarterHrCost \* QuarterHrs)

Then QuarterHrs = (Kosts – 2.50) / QuarterHrCost

lastRideLength = (bikeCosts[ bike ][2] - 2.50) / COSTPERQUARTERHR

1.b) In the bikeCost array defined above, the row index value refers to: {4 points}

A. the total number of bikes in the MavBikes system

B. the campus registration number of the current bike

C. the current bike

D. the bike ID in the MavBikes system

1.c) In the arrays defined above, the campus ID : {4 pts}

A. is stored as a number in each row of the bikeList and bikeCustomer arrays

B. refers to the campus registration number of the current bike

C. is a number in the bikeList array and a string in the bikeCustomer array

D. is the area code where the campus is

1.d) In bikeCustomers above, if no customer is stored at row 200, then *bikeCustomer[200][1]* will:

A. cause a Java nullPointerException {4 pts}

B. ask the user to enter a last name

C. return an empty string

D. store the name HAL

1.e) True / False questions (Answer with T or F) {3 pts each}

F ZERO would be a good name for a constant to use as an index value

F ( bikeCustomers[i][0] == bikeList[j][0] ) is a valid comparison that will compile

F For US customers, bikeCustomers[i][3].length() should be 9

T The bikeCosts array connects the bikeList array and the bikeCustomers array

F Given the data above, the maximum number of bikes on campuses would be 250\*(999-10)

F The arrays bikeList, bikeCustomers, and bikeCosts are all given initial values by Java

1.f) The table below shows of all the current amounts owed for each bike rental (or the information about the last rental of the bike) for the first bikes in the bikeCosts array that have a non-zero customer ID. The 8-column table gives the customer's full name, the bike number, and the rental date, then the table should have either a number of quarter hours rented or and the amount the customer owes. If the bike is being rented, the quarter hours amount will be greater than 0 and payment will be -1. After the rental, the quarter hours will be 0 and payment will be greater than or equal to 0.

Bike 15 Min. Amount

Last, First No. Date Intervals Due

Ngai, Dr. Efraim 55 11/17/2009 0.00 90.95

Marten, Miiyahbin 133 12/ 7/2017 70.00 -1.00

Jessell, Dr. Sally-Jo 56 8/ 2/2016 8.00 -1.00

Bandaranaike, Akiya 1 3/12/2010 0.00 13.98

, Cuauhtli 40 9/27/2015 0.00 87.75

Sanchez, Alex 2 11/ 6/2009 0.00 18.45

Ngala, Khairi 109 10/21/2017 18.00 -1.00

Hunyango, Alyas 3 3/ 9/2018 0.00 91.37

Hoshi, Dr. Kimiyo 62 8/18/2011 0.00 93.56

ibn Sallah al-Rashid, Haroum 81 5/ 3/2012 0.00 2.06

Langkowski, Dr. Walter 63 8/ 1/2009 0.00 27.52

Blanc-Dumont, Andre 4 9/ 6/2013 0.00 66.43

Tao-Yu, Ho 85 5/17/2011 0.00 28.52

Ucaroglu, Murat 139 8/14/2012 0.00 78.83

Bat, Ogon 144 6/ 1/2017 9.00 -1.00

Select the Java code fragment that would output the rows for the above: {6 points}

A. System.out.printf("%20s,%20s %6d%4d/%2d/%4d%10.2f%8.2f\n",

bikeCustomers[nameIndex][2], bikeCustomers[nameIndex][1],

i, ((int)(bikeCosts[i][3])), ((int)(bikeCosts[i][3] \* 100) % 100),

(int)(bikeCosts[i][4]), bikeCosts[i][1], bikeCosts[i][2]);

B. System.out.printf("%20s,%20s %6d%4d/%2d/%4d%10.2f%8.2f\n",

bikeCustomers[i][2], bikeCustomers[i][1],

i, ((int)(bikeCosts[costIndex][4])),

((int)(bikeCosts[costIndex][3] \* 100) % 100),

(int)(bikeCosts[costIndex][3]), bikeCosts[costIndex][1],

bikeCosts[costIndex][2]);

C. System.out.printf("%20s,%20s %6d%4d/%2d/%4d%10.2f%8.2f\n",

bikeCustomers[i][2], bikeCustomers[i][1],

bikeData[i][0], ((int)(bikeCosts[costIndex][4])),

((int)(bikeCosts[costIndex][3] \* 100) % 100),

(int)(bikeCosts[costIndex][3]), bikeCosts[costIndex][1],

bikeCosts[costIndex][2]);

D. System.out.printf("%20s,%20s %6d%4d/%2d/%4d%10.2f%8.2f\n",

bikeCustomers[nameIndex][2], bikeCustomers[nameIndex][1],

i, ((int)(bikeCosts[i][3])), ((int)(bikeCosts[i][3] \* 100),

(int)(bikeCosts[i][4]), bikeCosts[i][1], bikeCosts[i][2]);

1.g) Write a method called *makeMMDD* that takes in integer month and day values as parameters and returns the double value that contains *mm.dd* . Your method needs to do error checking on the month and on the day (just check for 31 days or less – don’t check by month. After the values are error-checked, create the mm.dd value and return it. {12 pts}

// Something like this

public static double makeMMDD(int month, int day)

{ double mmdd = 0;

if ((month > 0) && (month < 13))

{ if ((day > 0) && (day < 32))

{ mmdd = month + (day / 100.0);

}

}

return mmdd;

}

1.h) Write a method called *storeDate* that takes in the *bikeCosts* array, an index value, and the mmdd value from *makeMMDD* and a year value. Your method needs to do error checking on the year to make sure that the year is greater than or equal to 2010 when the MavBikes company started. After the values are error-checked, store the mm.dd value and the year value into the array at the location given by the index. {12 pts}

// something like this

public static void storeDate(double[][] costs, int index,

int year, double mmdd)

{

if (year >= 2010)

{

costs[index][3] = mmdd;

costs[index][4] = year;

}

}

1.i) If you were to write a bubble sort algorithm to sort the bikeCustomers array, which of the following tests would sort the array in alphabetical order by last name? {4 pts}

A. ((bikeCustomers[j][index].compareToIgnoreCase(bikeCustomers[j+1][index])) < 0)

B. ((bikeCustomers[j][index].compareToIgnoreCase(bikeCustomers[j+1][index])) > 0)

C. (bikeList[j][index] > bikeList [j+1][index])

D. (bikeList[j][index] <= bikeList [j+1][index])

1.j) If the output shown above shows the order of elements in the arrays, what type of algorithm would be most effective to find an element in these arrays? {4 pts}

NOTE: This question is too vague. If look at sorted part, bin search; if unsorted, lin search

A. binary search C. linear search

B. quick sort D. bubble sort

1.k) In order to use binary search, what must be true? {4 pts}

A. No other search algorithm can be used at all

B. The data in the array must be integer data

C. Linear search must have already failed on the array

D. The array data must be in sorted order

1.l) In a method, called flash, that takes in a 2 dimensional array *values*, what would the parameters definition and the method call look like? {4 pts}

A. public static void flash(int [][] values, int cat) ; flash((int) inFile, 6);

B. public static void flash(int [][] values, int cat) ; flash(6, inFile);

C. public static void flash(int cat, int [][] values) ; flash(inFile, 6);

D. public static void flash(int [][] values, int cat) ; flash(inFile, 6);

2) A selection sort searches an array looking for the largest element in the array. Then the largest element is swapped with the first element of the array. The process is then repeated for the subarray beginning with the second element of the array. Each pass of the array results in one element being placed in its proper location. When the subarray being processed contains one element, the array is sorted. Write a function using loops to perform this algorithm. Your array can have any type of values you want. Just write a comment to state what type of values you are choosing to sort. {10 points}

Extra credit: Write the method (function) as a recursive function instead of one with loops. {+5}

public static void selsort(int arr[], int max)

{

int tempmax = arr[0];

int tempindex = 0;

int i = 0;

int arrLength = arr.length;

boolean tempTF = true;

int lgindex = 0;

// Recurse down the array until 1 element left

while (arrLength > 1)

{

tempmax = arr[lgindex];

tempindex = lgindex;

// Find the max element

for ( i = (lgindex+1); i < max; i++)

{

if (tempmax > arr[i])

{

tempindex = i;

tempmax = arr[i];

}

}

// Swap max with first element

if (tempindex != lgindex)

{

arr[tempindex] = arr[lgindex];

arr[lgindex] = tempmax;

}

lgindex++;

arrLength--;

// if ()

//return selsort(&(arr[1]),max-1);

}

}

Deductions:

Not recursive –3

Does not follow instructions for finding largest value each pass -2

Does not follow instructions for swapping only two values each pass -2

Doesn’t use correct names – 1

Doesn’t correctly shorten the subarray – 2

Does not include comment about type of values to sort -2

Error in logic –various

Not a function -3

Extra credit

XC1) List one benefit and one drawback with using two-dimensional arrays for storing a variety of data of the same type. {4 points}

Benefit: All data in same storage structure

Drawback: Having to use indices to access the various elements rather than descriptive names

Many reasonable choices here.

XC2) Write a bubble sort algorithm for an array of 1000 integers. {4 points}

public static void bubCust(int[] cust, int index)

{

int inPlace = cust.length;

int temp = 0;

for (int i = 0; i < cust.length; i++)

{

inPlace--;

for (int j = 0; j < inPlace; j++)

{

if (cust[j]> cust[j+1])

{

temp = cust[j];

cust[j] = cust[j+1];

cust[j+1] = temp;

}

}

}

}

XC3) Write a few sentences on why this class has been your favorite.

{Any polite answer will receive 2 points.}

2) XC recrusive

public static int[] recselsort(int[] arr, int max)

{

int[] tempArray = new int[max-1]; // for recursion

int tempmax = arr[0];

int tempindex = 0;

int i = 0;

int arrLength = arr.length;

boolean tempTF = true;

int lgindex = 0;

// Recurse down the array until 1 element left

tempmax = arr[lgindex];

tempindex = lgindex;

// Find the max element

for ( i = (lgindex+1); i < max; i++)

{

if (tempmax > arr[i])

{

tempindex = i;

tempmax = arr[i];

}

}

// Swap max with first element

if (tempindex != lgindex)

{

arr[tempindex] = arr[lgindex];

arr[lgindex] = tempmax;

}

lgindex++;

if (max > 1) // stops recursion

{

tempArray = recselsort( Arrays.copyOfRange(arr,1,arrLength) ,max-1); // recur call

}

for (i = 1; i < arrLength; i++) // put sorted subarray back into original array

{

arr[i] = tempArray[i-1];

}

return arr;

}