Instructions: Answer all questions. Please add your name to every page, just in case…

Total Questions: 8 Total points: 100.

1. Determine if the following statements are TRUE or FALSE. Circle the answer (20 points, 2 points for each correct answer)
	* TRUE FALSE : If an exception occurs in a try { } block at runtime, program execution is altered to enter the corresponding catch { } block
	* TRUE FALSE : One way to implement Encapsulation in a Java class is to make class variables public.
	* TRUE FALSE : Polymorphism allows a programmer to write the same action in different ways
	* TRUE FALSE : Inheritance allows a for all methods to be written only once
	* TRUE FALSE : A Constructor is not a method, but does have a return type
	* TRUE FALSE : If a class has static method, it can be invoked without creating an instance of that class
	* TRUE FALSE : If a class has a toString() method, it can only be invoked if you call this method directly in your code
	* TRUE FALSE : Code within a do-while loop must be executed at least once.
	* TRUE FALSE : These statements will compile without error in a Java program:

Double x = 3.14;

String str = “Value: ” + (int)x;

* + TRUE FALSE : The break; statement will, when encountered within a loop, cause program execution to jump to the beginning of the loop for its next iteration.
1. Consider the following code (10 points):

public class RemoveContinueStmt {

public static void main(String[] args){

 for (int j = 0; j < 5; j++) {

 if (j==3) {

 continue;

 }

 System.out.println(j+" ");

 }

}

Part (A) What is the output when this program runs (3 points)?

PART (B) Rewrite the code within main() so that you use a while{ } loop **and** that you do **not** use a continue statement. (7 points)

public class RemoveContinueStmt {

public static void main(String[] args){

// add your code here…

1. Consider the following class (15 points):

public class ArrayAdder {

 int [] cells; // the only data field in ArrayAdder

public ArrayAdder(int size, int howMany) {

/\*\*

 \* @param size - The maximum size of the array

 \* @param howMany – number elements to initialize with a random number

 \* @note - we require howMany <= size. how to do it ??1?

 \*/

 this.cells = new int[size]; // reserve memory for array

// first initialize every element to zero

 for (int i=0; i<size; i++) {

 this.cells[i] = 0;

 }

// next, place random numbers into "howMany" elements

// ALSO: add 1 to the random number to ensure the int > 0

 for (int i=0; i<howMany; i++) {

 this.cells[i] = (int) (Math.random()\*10 + 1);

 }

}

public int addEvens() {

/\*\*

 \* Method to add together the elements of cells containing even numbers

 \* How to do it ??!?

 \* @return - The sum of the elements in cell that are even.

 \*/

//= = > implement this method efficiently.

//= = > since the length of the array can be greater than the

//= = > nonzero elements in the array, stop adding when an element is 0

}

public static void main(String[] args){

 ArrayAdder aa(120, 59);

 int sumTotalOfEvens;

 sumTotalofEvens = aa.addEvens();

 System.out.println(sumTotalOfEvens+" ");

 }

}

Part (A) Describe, or, write the code to show how you would “ensure that howMany <= size” in the constructor. (3 points)

Part (B) Implement the addEvens() method. That is, return the sum of the values in

cells[ ] whose contents are even. Recall the calculator assignment if you need help here. Follow the edict in the comments. That is, use only one loop and stop adding when an element is zero. (6 points)

Part (C) Create another constructor that accepts an integer array as the only parameter:

(int[] array) and populates the cells array with the even indices. Consider 0 to be an even number… continued on next page.

For example, if the array {1,2,3,4,5} was passed in as a parameter to the constructor, then cells would contain just three elements, {1,3,5} (6 points)

1. Consider the following code from a recent lecture. Rewrite the switch statement using if/else statements instead (10 points):

char grade;

… // assign grade, somehow

switch (grade){

 case 'A':

 System.out.println("Great job!");

 break;

 case 'B':

 System.out.println("Good!");

 break;

 case 'C':

 System.out.println("OK");

 case 'D':

 System.out.println("Could be better");

 break;

 case 'F':

 System.out.println("You need a break");

 break;

 default:

 System.out.println(grade + " is not a proper grade");

}

1. Recall the recent lab where you created a Matrix class. For this problem, the data fields in the Matrix class are (15 Points) :

 int[][] m; // matrix 2d-array

 int nrows; // the number of rows in m

 int ncols; // the number of columns in m

Consider the method called add(). The signature of this method is:

Public void add(Matrix);

The add() method accepts a single parameter of type Matrix.

Part (A) Which OOP pillar is being violated by this implementation of add()? (3 Points)

Part (B) How could you update the signature for the add method so that there are no OOP pillars being violated? (6 Points)

Part (C) In order for Part (B) to work, what method would you need to create? Write this method. (6 Points)

Questions 6, 7 and 8 can be thought of as one problem.

1. Create a “Currency” class. It will have 3 integer variables (data fields) that, once initialized, will remain constant (10 points):
	* ONE – to represent a one dollar bill
	* FIVE – to represent a five dollar bill
	* TEN – to represent a ten dollar bill

The only method is a constructor that initializes the variables to their proper value

1. Create a “Cash” class that inherits from the “Currency” class. It will have three integer variables that represents the number of bills of each denomination (10 points):
	* ones – the number of one dollar bills
	* fives – the number of five dollar bills
	* tens – the number of ten dollar bills

There is one constructor that takes 3 parameters – the number of ones, fives and tens, and two methods.

There are also two additional methods:

* + total() takes no parameters and returns the value of the cash object. For example, if ones = 2, fives = 1, and tens = 3, total will return 37;
	+ toString() which takes no parameters and returns a String with the number of ones, fives, tens, and the total amount of cash. Instead of using numbers for value of the currency as numbers directly, use variables from the Currency class.
1. Create a “Wallet” class (10 points):

It contains two variables

* + empty of type boolean
	+ amount of type Cash

…and these methods:

* + A constructor that accepts an integer representing the total amount of cash and creates a new Cash object with it (Recall the calculator homework to help you figure out the number of ones, fives and tens you have. Eg $209 represents 20 tens, 1 five and 4 ones). If the amount of cash is zero, then the wallet is empty and empty is true, otherwise it is false
	+ isempty() takes no parameters and returns the value of empty.
	+ toString()takes no parameters and returns a String that says the wallet is empty if it is empty, or the number of ones, fives, tens, and the total amount of cash in the wallet.