University of Chicago
Laboratory Schools
Advanced Placement Computer Science
Quarter 1 Practice Exam
1.) (10 percent) Write a segment of code that will produce the following pattern of X’s in a console window. A solution that does not use nested for-loops will not receive full credit.

X
XX
XXX
XXXX
XXXXX
XXXXXX
XXXXXXX

for(int i=0; i<7; i++){
    for(int j=0; j<7; j++){
        if(i<=j) System.out.print("X");
    }
    System.out.println();
}

2.) (20 percent) Write a method called arrayDiff which accepts two arrays of ints as parameters, compares the two arrays and returns an array of booleans that indicates which indices in the two arrays hold different values. The example below shows two arrays arr1 and arr2 and the array of booleans that should be returned (where “T” = true, and “F”=false).

arr1 ->  [3][4][5][2][8][6][4][5][9][1][7]
arr2 ->  [3][4][3][2][7][6][4][5][5][1][7]

returns->  [F][F][T][F][T][F][F][F][T][F][F]

In the example above the array of booleans was created by repeatedly comparing the elements in arr1 and arr2 at some index i. If the values in arr1 and arr2 at index i are different, then index i in a boolean array of the same length as arr1 should be set to true, otherwise false. If the two arrays passed to the method are not equal in length, the method should return null.

Write the method below. Be sure to write a proper method header.
3. (30 points)
Write a simple class that represents a bank account called `BankAccount`. A bank account has an account number and some amount of money in it. At this bank you can carry a negative balance under certain conditions – read the method descriptions to find out the conditions. A bank account has the following methods:

- `getBalance` – returns the current account balance.
- `deposit` – deposits some amount of money into the bank account. If the account currently has a negative balance, the amount deposited must be enough to make the balance positive or zero, otherwise the deposit is not accepted.
- `withdraw` – takes some amount of money out of the account but only if the account currently has a positive balance and the result of withdrawing won’t make the account have less than $-1000.

Write the entire class choosing appropriate data types, return types, private variables and method parameters. The `BankAccount` constructor should initialize the bank account with a balance of 0. There should be NO CALLS to `System.out.println` in this code.

```java
public class BankAccount{
    private int accountNum;
    private double balance;
    public BankAccount(int acctNum){
        balance = 0;
        accountNum = acctNum;
    }
    public double getBalance(){
        return balance;
    }
    public boolean deposit(double amt){
        if(amt+balance >= 0){
            balance += amt;
            return true;
        }
        return false;
    }
    public boolean withdraw(double amt){
        if(balance>0 && balance - amt >= -1000){
            balance -= amt;
            return true;
        }
        return false;
    }
}
```
4. A robot starts on 3,2 facing East. There will be a row of beepers in front of the robot of indeterminate length. The problem will take you through several steps of building recursive methods. First consider the class in which you will be writing these methods:

```java
public class CountBot extends Robot{
    public CountBot(int s, int a, Direction d, int b){
        super(s,a,d,b);
    }
    private int countPile(){
        /* Assume this method works.
        It will count a pile of beepers on the corner this robot is currently standing it.
        It will return the pile to its original state and return the number of beepers on the pile.
        */
    }
    public int getLength(){
        //problem 1
    }
    public void outAndBack(){
        //problem 2
    }
    public int getLengthAndBack(){
        //problem 3
    }
    public int addAndGetLength(int sum){
        //problem 4
    }
}
```

1. Write the recursive method called “getLength” which returns an integer that is the number of consecutive beeper piles that the robot is facing (In the example above the method would return 4).

This method should only use one robot.

Here is the algorithm:
   The robot should move forward
   If it’s not next to a beeper it should return 0.
   If it’s next to a beeper it should return 1 + another call to getLength()
2. Write the method “outAndBack” which is similar to getLength. It should recursively, walk to the end of the pile and then walk back to where it started. For this one, at you stopping case, have the robot turn around. The recursive calls should have the robot move, recurse, and move again – if the robot turns around at the end then the second move will be moving it the opposite direction.

```java
public int outAndBack(){
  move();
  if(!nextToABeeper()){
    turnaround();
    move();
  } else{
    outAndBack();
    move();
  }
}
```

3. Write `getLengthAndBack` which combines `getLength` and `outAndBack`. It should return the number of beeper piles and the robot should end up where it started. This is really just a combination of the code you wrote for 1 and 2.

```java
public int getLengthAndBack(){
  move();
  if(!nextToABeeper()){
    turnaround();
    move();
    return 0;
  } else{
    int result = 1+getLength();
    move();
    return result;
  }
}
```

4. Write “addAndGetLength” which does the same thing as `getLengthAndBack` AND it also deposits a beeper pile at the end of the row containing a number of beepers equal to the sum of all the piles. This method requires your recursion do work on the way “down” as well as on the way back “up” the recursive stack.

The sum should be accumulated as the robot walks “down” the row by passing a value down through the method argument “sum.” When it gets to the end, it should
deposit a beeper pile (you can use World.placeBeepers), turnaround and walk back returning the length of the row as it goes.
5.) (20 points)
Write a console application that repeatedly asks the user for, and reads in, a pair of strings. The program should count the number of times the strings are the same and the number of times they’re different.

The program should stop asking for strings when: you’ve collected 100 pairs; either of the strings is “quit”; or you’ve collected six consecutive pairs that are the same. Before the program exits, you should print out the number of times the pair was the same and the number of times the pair was different.
5. Please know the definitions of these terms and be able to use them with examples:

- Inheritance
- Polymorphism
- Interface
- Abstract Class
- Abstract Method
- Strategy

Think of classes of real things that you could describe with a Java class structure. Example: A coffee shop sells multiple items. Every item has a price and a stock number. The coffee shop sells multiple kinds of coffee and multiple kinds of snacks. Every type of coffee has a size. There is regular coffee and Espresso drinks which include latte and cappuccino.

Draw a class diagram with boxes and arrows, etc. that describes the Objects at this coffee shop. Be sure to label your diagram with the list of words shown above where appropriate. Also invent your own hierarchy for some snacks that a coffee shop might offer. (NOTE: there are many “correct” solutions to this, but more credit will be given for designs that use more than just plain classes in ways that are robust and appropriate.)