

Question 1. [17 MARKS]**Part (a)** [7 MARKS]

Complete the body of method `lengths` below. `lengths` returns an array containing the the length of each `String` in `list`. For example, if the array `["A", "big", "fat!", "bean", "", "1"]` is the argument to `lengths`, then this array is returned: `[1, 3, 4, 4, 0, 1]`.

You may assume that `list != null`, and you may write helper methods if you wish.

```
public static int[] lengths(String[] list) {
```

Part (b) [10 MARKS]

"Mary" and "Mercy" each have `M` at index 0, and `r` at index 2. The `y` appears in both `Strings`, but at a different index in each `String`. We say that two `Strings` "share" a character when that character appears at exactly the same index in each `String`. So "Mary" and "Mercy" share 2 characters: `M` and `r`.

On the next page, write the body of method `sameChars`. You may assume that `list != null`, and you may write helper methods if you wish.

```
/**
 * Return a two-dimensional array that contains, for each pair of
 * Strings in 'list', the number of characters that pair shares.
 *
 * Example: if 'list' is ["Mary", "Mercy", "Jim", "O'Ronald Jim"] then this
 * array is returned:
 *
 *           [ [4, 2, 0, 0],
 *             [2, 5, 0, 0],
 *             [0, 0, 3, 0],
 *             [0, 0, 0, 12] ]
 *
 * Each String shares all its characters with itself; hence the diagonal.
 */
public static int[][] sameChars(String[] list) {
```

Question 2. [10 MARKS]

Given the following declarations:

```
String s1 = "";
String s2 = "";
String s3 = " ";
String s4 = "          ";

boolean b = ('a' < 'b') || !('a' < 'b');
double f = 21.6;

int i = 2;
int j = 4;
int k = -6;
```

In the spaces provided, write the result of evaluating the following expressions.

Result	Expression
_____	s1.equals(s2)
_____	!(s1.equals(s3))
_____	(i != j)
_____	(s3.length() < s4.length()) && (!b ((i+j) < k))
_____	(i-3 == -1) (s4.substring(i-3, i-2).equals("a"))

Question 3. [8 MARKS]

The code fragment at the bottom of this page should print this n-row table of numbers:

```

1  2  3  4  5
2  4  6  8 10
3  6  9 12 15
...
n  n*2 n*3 n*4 n*5

```

After r iterations of the outer loop, the following table should be printed:

```

1  2  3  4  5
2  4  6  8 10
...
r  r*2 r*3 r*4 r*5

```

Complete the fragment so that it works correctly. You may only write in the underlined areas.

```

// Print n rows of the table.

for (int r = 0; r != _____ ; r++) {

    // Print a single row of the table.

    for (int c = _____ ; c != _____ ; c++) {

        // \t is the tab character; print it and then the next entry in the row.

        System.out.print("\t" + _____ * (c+1) );

    }

    System.out.println(); // Start a new row.
}

```

Question 4. [20 MARKS]

A dog has a name, a breed, and an age (in years, an `int`). Parts (a) through (d) all deal with the contents of class `Dog`, below. Write your answers in the spaces provided.

Part (a) [2 MARKS] Write instance variables for the `Dog` class.

Part (b) [2 MARKS] Write an appropriate constructor with three parameters.

Part (c) [4 MARKS] Write `equals`. Two `Dogs` are equal if the breeds are the same and the ages differ by one year or less.

Part (d) [2 MARKS] Write `toString`. For example, for an object representing a three year old Samoyed named "Toby", the method should return "Toby - Breed: Samoyed, Age: 3".

```
public class Dog {
    // Parts (a) and (b) here. -----
```

```
// Part (c) here. -----
```

```
// Part (d) here. -----
```

Part (e) [10 MARKS]

This is possibly the most challenging question, so you may want to leave it to the end.

A veterinarian's office specializes in dogs. They need a program to keep track of the dogs in the waiting room. They treat dogs in the order the dogs come in, except that equal dogs are treated all at once. When the waiting room is empty, they stop for a coffee break.

Write a class `WaitingList` that manages a `Vector` of `Dogs`. `Dogs` are added to a `WaitingList` one at a time, but are removed in groups: the return type of your `get` method should be `Vector`, and it should remove and return the first `Dog` in the `Vector` and all `Dogs` equal to that one. (Note that often your `Vector` will only contain one `Dog`.) Include a way to determine whether the `WaitingList` is empty.

It is up to you to choose your method signatures and instance variables, except that you must not make anything `static`.

Note that you are writing a single class, not a main program.

We recommend starting at the end of the `Vector` and working toward the beginning when you're looking for and removing equal `Dogs`.

Question 5. [6 MARKS]

What is the output of UsePuzzle?

```
public class Puzzle {
    private static int changeValue = 0;
    private int k = 0;

    public Puzzle(int j) {
        k = j;
    }

    public void change() {
        k = k * 2;
        changeValue = changeValue + 1;
    }

    public static void changeK(int k) {
        k = k + 1;
        changeValue = k + changeValue;
    }

    public static void changeP(Puzzle p) {
        p.k = p.k * 3;
        changeValue = changeValue + 1;
    }

    public String toString() {
        return "" + k + " " + changeValue;
    }
}
```

```
public class UsePuzzle {
    public static void main(String[] args) {
        Puzzle p1 = new Puzzle(5);
        Puzzle p2 = new Puzzle(1);

        p1.change();
        p2.change();

        Puzzle p3 = p2;
        p3.change();

        Puzzle.changeP(p1);

        int k = 9;
        Puzzle.changeK(k);

        System.out.println(p1);
        System.out.println(p2);
        System.out.println(p3);
        System.out.println(k);
    }
}
```

Output:

Question 6. [10 MARKS]**Part (a)** [5 MARKS]

Complete method `isAllCapitals`. You may assume that `s != null`.

```
/** Return true if s contains only capital letters, and false otherwise. */  
public static boolean isAllCapitals(String s) {
```

```
}
```

Part (b) [5 MARKS]

In Assignment 6 you were asked to write a method `extractCapitals`, which looks like this:

```
/** Return a new String consisting of the capital letters in s. */  
public static String extractCapitals(String s)
```

You're designing the test cases for the automarking program, and have decided to have seven test cases that test `extractCapitals` as fully as possible.

We have given you two test cases; provide *exactly* 5 more. Provide a concise description as part of each test case. No marks will be given for test cases without a description.

Argument	Expected return	Reason
"a"	""	A string of length 1 with no capitals
"A"	"A"	A string of length 1 that contains a capital

Question 7. [12 MARKS]

The following three classes compile without errors.

```
class Vehicle {
    private int numWheels;

    public Vehicle(int nw) {
        numWheels = nw;
    }

    public int getNumWheels() {
        return numWheels;
    }
}

class Truck extends Vehicle {
    private double loadCapacity;

    public Truck (int nw, double lc) {
        super(nw);
        loadCapacity = lc;
    }

    public double getLoadCapacity() {
        return loadCapacity;
    }
}

class Car extends Vehicle {
    private int numSeats;

    public Car(int ns) {
        super(4);
        numSeats = ns;
    }

    public int getNumSeats() {
        return numSeats;
    }
}
```

The statements on the next page use classes `Vehicle`, `Car`, and `Truck`, and might appear in a `main` method. Some statements contain errors. The order of the statements matters, but errors in earlier statements don't invalidate later ones.

Every statement has 1 of four possibilities:

1. This statement can be compiled without producing any error messages, and runs without error.
2. This statement can be compiled without producing any error messages, but crashes when running.
3. This statement contains compile errors, and can be repaired by either adding or removing a cast.
4. This statement contains compile errors, and cannot be repaired by adding or removing a cast.

To the right of to each line, place a checkmark in the appropriate column. We have done the first three lines for you. Every correct checkmark is worth 1 mark. Every incorrect checkmark is worth -1/2 mark.

```

Vehicle v1 = new Vehicle(3);

System.out.println(v1.getNumWheels());

System.out.println(((Car) v1).getNumSeats());

Car c1 = new Car(7);

System.out.println(c1.getNumWheels());

System.out.println(c1.getNumSeats());

System.out.println(((Truck) c1).getLoadCapacity());

Truck t1 = new Truck(14, 1000.0);

System.out.println(t1.getNumWheels());

System.out.println(((Vehicle) t1).getLoadCapacity());

Vehicle v2 = new Car(9);

System.out.println(((Car) v2).getNumWheels());

System.out.println(v2.getNumSeats());

System.out.println(((Truck) v2).getLoadCapacity());

Car c2 = new Truck(18, 1500.0);
    
```

1	2	3	4
✓			
✓			
	✓		
X	X	X	X
X	X	X	X
X	X	X	X
X	X	X	X

Question 8. [7 MARKS]

A *strictly increasing sequence* is a sequence of numbers where each successive number is larger than the previous one. 1, 3, 4, 12 is a strictly increasing sequence. 1, 4, 3, 12 is not. 1, 4, 4, 12 is not either, because each number must be greater than the previous.

Complete the loop below so that it prints **true** if the input forms a strictly increasing sequence, and **false** if it does not. The input is terminated by "quit". You may assume correct input, and you may assume that at least 1 number will be entered.

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
boolean isIncreasing = true;
int prev = Integer.parseInt(br.readLine());
String s = br.readLine();
while (!s.equals("quit")) {

}

System.out.println(isIncreasing);
```

This page is provided for rough work and any answers that didn't fit.

This page is provided for rough work and any answers that didn't fit.

Total Marks = 90

Student #: _____

Page 14 of 14

END OF EXAMINATION