If the instructor is unavailable in the examination room and if doubt exists as to the interpretation of any problem, the candidate is urged to submit with the answer paper a clear statement of any assumptions made.

Proctors are unable to respond to queries about the interpretation of exam questions. Do your best to answer exam questions as written.

Please write your answers in the boxes provided. Extra space is available on page 16 of the exam. The back of any page can be used for rough work. This exam refers exclusively to the use of the Java language. Comments are not required in the code you write.

This is a closed book exam. No computers or calculators are allowed.
Problem 1) [marks]

Three classes form an object hierarchy. Here are two of the three classes, ClassR and ClassA:

```java
public abstract class ClassR {
    final int ONE = 1;
    abstract int method1 (int a, int b);
    public int method2 (int a) {
        return a + a;
    }
}

public class ClassA extends ClassR {
    public ClassA (int a, int b) {
        super(a, b);
    }
    public int method3 (int a) {
        return ONE + super.method3(a);
    }
    public int method4 () {
        return getX() + getY();
    }
}
```

On the next page is a separate testing class with a copy of its console output, which illustrates the operation of the methods in the hierarchy:
public class MainClass {

    public static void main(String[] args) {

        ClassA ca = new ClassA(100, 1000);

        System.out.println(ca.method1(10, 20));
        System.out.println(ca.method2(30));
        System.out.println(ca.method3(40));
        System.out.println(ca.method4());

    } // end main

} // end MainClass

/* OUTPUT:
30
60
1141
1100
*/

On the next page, write the complete code for the missing class. Write only the code required to complete the hierarchy in the most efficient way. Note that the only arithmetic operation used in any of the methods is addition.
Problem 2) [ marks]

For this problem you will need to write a complete class called SurfaceShape, with all the methods that should be present when you follow good encapsulation practices. You will need to use some other classes that are provided below. You will also be given a syntax summary for the ArrayList<T> class.

SurfaceShape is designed to hold a collection of points in space that all lie on a plane. Use an ArrayList<T> for the collection. The constructor of the class will need to calculate the equation of the plane for the points, and it will also store this equation as an object.

Here is an exception class to be used by SurfaceShape:

public class SurfaceShapeException extends Exception {
    public SurfaceShapeException (String s) {
        super(s);
    }
    public SurfaceShapeException () {
        super("Illegal surface shape!");
    }
}

Points will be stored as objects, as defined by the Point class:

public class Point {
    private double x, y, z;

    public Point (double x, double y, double z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    public double getX () { return x; }
    public double getY () { return y; }
    public double getZ () { return z; }

    public boolean equals (Object p) {
        boolean check = false;
        Point pp;
        double tolerance = 1e-6;

        if (p instanceof Point) {

pp = (Point)p;
check = Math.abs(pp.x - x) < tolerance;
check = check && Math.abs(pp.y - y) < tolerance;
check = check && Math.abs(pp.z - z) < tolerance;

} // end if

return check;

} // end equals

public Object clone () {
    return new Point (x, y, z);

} // end close

} // end Point class

The equation of the surface will be stored in a SurfaceEquation object:

public class SurfaceEquation {
    double a, b, c;

    public SurfaceEquation (double a, double b, double c) {
        this.a = a;
        this.b = b;
        this.c = c;
    } // end SurfaceEquation constructor

    public double calcZ (double x, double y) {
        return a * x + b * y + c;
    } // end calcZ

    public String toString () {
        return String.format("z = %6.3f * x + %6.3f * y + %6.3f", a, b, c);
    } // end toString

    public Object clone () {
        return new SurfaceEquation(a, b, c);
    } // end clone

} // end SurfaceEquation
Assume the existence of a class called `MatrixSolver` that contains a static method called `gaussJordan` that provides the solution to a system of simultaneous equations. Remember assignment 2? The system of equations is described by:

\[
A \mathbf{x} = \mathbf{b}
\]

where \( A, \mathbf{x} \) and \( \mathbf{b} \) are matrices. \( A \) is \( n \times n \), and \( \mathbf{x} \) and \( \mathbf{b} \) are of size \( n \). The `gaussJordan` method accepts the matrices \( A \) and \( \mathbf{b} \) as arrays of `double`'s, and returns the matrix \( \mathbf{r} \) as an array of `double`'s, where \( \mathbf{r} \) is defined as:

\[
\mathbf{I} \mathbf{x} = \mathbf{r}
\]

The matrix \( \mathbf{I} \) is the identity matrix. Consider describing a plane, in three dimensions (\( x, y \) and \( z \)) using the general equation:

\[
z = ax + by + c
\]

If you had three points: \((x_1, y_1, z_1), (x_2, y_2, z_2)\) and \((x_3, y_3, z_3)\), then you could create the following system of equations:

\[
\begin{align*}
ax_1 + by_1 + c &= z_1 \\
ax_2 + by_2 + c &= z_2 \\
ax_3 + by_3 + c &= z_3
\end{align*}
\]

Or, in matrix form:

\[
\begin{bmatrix}
  x_1 & y_1 & 1.0 \\
  x_2 & y_2 & 1.0 \\
  x_3 & y_3 & 1.0
\end{bmatrix}
\begin{bmatrix}
  a \\
  b \\
  c
\end{bmatrix}
= 
\begin{bmatrix}
  z_1 \\
  z_2 \\
  z_3
\end{bmatrix}
\]

A solution to this matrix equation would yield the equation coefficients, \( a, b \) and \( c \).

The `SurfaceShape` class' constructor accepts an `ArrayList<T>` which has been typed to contain `Point` objects. The constructor will throw an exception if the `ArrayList` is either `null` or contains fewer than three `Point`'s. Next, the constructor will have to determine the equation of the plane using the first three `Point`'s in the `ArrayList`. After creating a `SurfaceEquation` object, you will throw an exception if any of the remaining `Point`'s in the `ArrayList` do not lie on the plane. For a `Point` to be said to lie on the plane, calculated and given \( z \) values must agree to less than \( 1 \times 10^{-6} \).
You will find the following `ArrayList<T>` methods useful:

- `size()`  // returns the number of occupied positions in the ArrayList collection
- `add(Object)`  // adds the provided object to the end of the ArrayList collection
- `get(int)`  // returns the object at the given position in the ArrayList collection as an Object
- `contains(Object)`  // Returns true if the collection contains the Object provided as a parameter

Do not write any mutator methods for your `SurfaceShape` class. You will need to write the two accessors, one for the `ArrayList` and the other for the `SurfaceEquation` object. Your `equals` method should override the `equals` method inherited from `Object`. Equality is defined as both `ArrayList` attributes having the same size and containing all the same `Point`'s in any order. Compare `SurfaceShape` objects based only on the size of their `ArrayList` collections. Return a positive integer if the supplied object has more `Point`'s. Your `toString()` method should return a `String` containing just the number of `Point`'s and the equation describing the plane. For example, it could return:

```
10 points, on plane: z = 4.000 * x + 5.000 * y + 7.000
```

Finally, include a `clone()` method that returns an `Object`. No other methods are required.

That was a very long description! Your solution should be shorter. Write your class in the space provided on the next three pages.
Problem 3) [ marks]

For this problem you will complete the code in the following GUI window definition. When your window first opens, it looks like:

![GUI Layout Problem](image)

When you click on the “Make a Wish” button, you will see:

![GUI Layout Problem](image)

The window is displayed by the code on the next page:
public class GUIExamProblem {
    public static void main(String[] args) {
        int numMessages = 40;
        GUIExamProblemWindow gep = new GUIExamProblemWindow(numMessages);
        gep.setVisible(true);
    } // end main
} // end GUIExamProblem

You will find the following methods useful:

    setHorizontalAlignment(JLabel.CENTER)
    add(component)
    add(component, BorderLayout.position)
    addActionListener(class_implementing_ActionListener)
    setText(String)

Inside your ActionListener class, you will need to define the following method:

    public void actionPerformed (ActionEvent e)

Here is the start of the GUIExamProblemWindow class definition:

import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JLabel;
import java.awt.FlowLayout;
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class GUIExamProblemWindow extends JFrame {
    private final int WIDTH = 440;
    private final int HEIGHT = 300;
// Add more class attributes here, if necessary
int numMessages;

public GUIExamProblemWindow (int numMessages) {
    super();
    this.numMessages = numMessages;
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setTitle("GUI Layout Problem");
    setSize(WIDTH, HEIGHT);

    // Complete the class below and on the next page: