To ensure you can run these programs without any prior knowledge, I have organized this guide into three parts: **The Setup**, **The Short Programs**, and **The Execution Steps**.

PART 1: The Setup (First-Time Only)

Before running any code, you need the "Tools."

1. **Install Java (JDK):** Download and install the Java Development Kit from Oracle.
2. **Install NetBeans:** Download and install the Apache NetBeans IDE.
3. **Create a Project:**
   - Open NetBeans.
   - Go to **File** > **New Project**.
   - Select **Java with Ant** > **Java Application**. Click **Next**.
   - Name it TermWork and click **Finish**.

PART 2: The Programs (Short & Examiner-Ready)

Each program below includes **Try-Catch** (to prevent crashes) and **Scanner** (to take your input).

1. Stream Operations

Theory: Moves data in three ways: to a permanent File, into temporary RAM (Byte), or via a speed-boosting "Buffer."

Java

```
import java.io.*;
import java.util.Scanner;

class StreamDemo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("1.File 2.Byte 3.Buff: ");
            int ch = sc.nextInt(); sc.nextLine();
            System.out.print("Enter text: ");
            String s = sc.nextLine();

            if(ch == 1) { // Physical File
```

```
            new FileOutputStream("f.txt").write(s.getBytes());
            FileInputStream fi = new FileInputStream("f.txt");
            int i; while((i=fi.read()) != -1) System.out.print((char)i);
        } else if(ch == 2) { // RAM Memory
            ByteArrayInputStream bi = new ByteArrayInputStream(s.getBytes());
            int i; while((i=bi.read()) != -1) System.out.print((char)i);
        } else { // High Speed Buffer
            BufferedOutputStream bo = new BufferedOutputStream(new
FileOutputStream("b.txt"));
            bo.write(s.getBytes()); bo.flush(); bo.close();
            BufferedInputStream bi = new BufferedInputStream(new FileInputStream("b.txt"));
            int i; while((i=bi.read()) != -1) System.out.print((char)i);
        }
    } catch(Exception e) { System.out.println(e); }
  }
}
```

2. Binary Read/Write

Theory: Computers store data as "Bytes" (numbers). This program converts your text into raw bytes and saves it.

Java

```
import java.io.*;
import java.util.Scanner;

class BinaryDemo {
  public static void main(String[] args) {
    try {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter text: ");
        byte[] d = sc.nextLine().getBytes();
        new FileOutputStream("bin.dat").write(d); // Writes bytes

        FileInputStream fi = new FileInputStream("bin.dat");
        int b; while((b=fi.read()) != -1) System.out.print((char)b + " ");
    } catch(Exception e) { System.out.println(e); }
  }
}
```

3. File IO Operations

Theory: The standard method to create a .txt file on your computer and read its content back.

Java

```
import java.io.*;
import java.util.Scanner;

class FileIODemo {
    public static void main(String[] args) {
        try {
            Scanner sc = new Scanner(System.in);
            System.out.print("Write to file: ");
            new FileOutputStream("io.txt").write(sc.nextLine().getBytes());

            FileInputStream fi = new FileInputStream("io.txt");
            int i; while((i=fi.read()) != -1) System.out.print((char)i);
        } catch(Exception e) { System.out.println(e); }
    }
}
```

4. Producer-Consumer (Multithreading)

Theory: One thread "produces" data while another "consumes" it. They use wait() and notify() to talk to each other so they don't crash.

Java

```
import java.util.*;

class Store {
    Queue<Integer> q = new LinkedList<>();
    synchronized void add(int v) throws Exception {
        while(q.size()==3) wait();
        q.add(v); System.out.println("Produced: "+v); notify();
    }
```

```
    synchronized void get() throws Exception {
        while(q.isEmpty()) wait();
        System.out.println("Consumed: "+q.poll()); notify();
    }
}
public class ProducerConsumerExample {
    public static void main(String[] args) {
        Store s = new Store();
        new Thread(() -> { for(int i=1; i<4; i++) try{ s.add(i); }catch(Exception e){} }).start();
        new Thread(() -> { for(int i=1; i<4; i++) try{ s.get(); }catch(Exception e){} }).start();
    }
}
```

5. Thread Synchronization

Theory: synchronized is like a lock. It makes sure only one person (thread) uses the shared data at a time to prevent errors.

Java

```
class MathOp {
    synchronized void calc(String n, int x) {
        System.out.println(n + " Result: " + (x * x));
    }
}
class SynchDemo implements Runnable {
    String n; int v; MathOp m;
    SynchDemo(String name, int val, MathOp op) {
        n=name; v=val; m=op; new Thread(this).start();
    }
    public void run() { m.calc(n, v); }
    public static void main(String[] args) {
        MathOp op = new MathOp();
        new SynchDemo("T1", 5, op); new SynchDemo("T2", 10, op);
    }
}
```

6. JDBC Database Connectivity

Theory: This allows Java to talk to a Database (like MySQL). You send a name from Java, and

it saves it in a table.

Java

```
import java.sql.*;
import java.util.Scanner;

class jdbcDemo {
    public static void main(String[] args) {
        try {
            Scanner sc = new Scanner(System.in);
            System.out.print("Enter Name to save: "); String n = sc.next();
            Connection c = DriverManager.getConnection("jdbc:mysql://localhost:3306/db", "root",
"pass");
            PreparedStatement p = c.prepareStatement("insert into stu values(?, 20)");
            p.setString(1, n); p.executeUpdate();
            ResultSet r = c.createStatement().executeQuery("select * from stu");
            while(r.next()) System.out.println(r.getString(1));
        } catch(Exception e) { System.out.println("Error: " + e); }
    }
}
```

7. GUI Calculator

Theory: Creates a visual window with buttons. It uses GridLayout for the buttons and ActionListener to perform math when you click.

Java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class CalculatorGUI implements ActionListener {
    JFrame f = new JFrame(); JTextField t = new JTextField();
    double n1; char op;
    CalculatorGUI() {
```

```
        f.setLayout(new BorderLayout()); f.add(t, "North");
        JPanel p = new JPanel(new GridLayout(4,4));
        String[] b = {"7","8","9","/", "4","5","6","*", "1","2","3","-", "0","C","=","+"};
        for(String s:b) { JButton btn=new JButton(s); btn.addActionListener(this); p.add(btn); }
        f.add(p); f.setSize(250,300); f.setVisible(true); f.setDefaultCloseOperation(3);
    }
    public void actionPerformed(ActionEvent e) {
        try {
            String s = e.getActionCommand();
            if(s.charAt(0)>='0' && s.charAt(0)<='9') t.setText(t.getText()+s);
            else if(s.equals("C")) t.setText("");
            else if(s.equals("=")) {
                double n2 = Double.parseDouble(t.getText());
                if(op=='+') t.setText(""+(n1+n2)); if(op=='-') t.setText(""+(n1-n2));
                if(op=='*') t.setText(""+(n1*n2)); if(op=='/') t.setText(""+(n1/n2));
            } else { n1 = Double.parseDouble(t.getText()); op = s.charAt(0); t.setText(""); }
        } catch(Exception ex) { t.setText("Error"); }
    }
    public static void main(String[] args) { new CalculatorGUI(); }
}
```

PART 3: Execution Instructions (Beginner Friendly)

Step 1: Creating the Files in NetBeans

1. Inside your TermWork project, right-click the **Source Packages** folder.
2. Select **New** > **Java Class**.
3. **Crucial:** Name the class exactly as shown in the code (e.g., StreamDemo or CalculatorGUI).
4. Delete everything in the NetBeans file and paste the code from above.

Step 2: How to Run

- **Method:** Right-click inside the code editor window and click **Run File**. (Do NOT use the green play button at the top, as it only runs the "Main" file).

Step 3: Handling Inputs

- **Keyboard Input:** When you run the program, look at the **Output** window at the bottom of NetBeans. Click inside that window to type your choices or text.
- **File Input (Auto-Creation):** For programs 1, 2, and 3, the code **automatically creates** the file (like f.txt or io.txt) for you. You don't need to create them manually!
    - *Where is the file?* Go to the **Files** tab (next to the Projects tab) in NetBeans. Open the

TermWork folder, and you will see your created text files there.

Step 4: Special Note for JDBC (Program 6)

- You must right-click the **Libraries** folder in your project.
- Select **Add JAR/Folder**.
- Locate and add your mysql-connector-java.jar file. If you don't do this, the program will show a "Driver Not Found" error.