Prepared by: **Chala Urgessa** , Neusoft Institute Technology

---

White Box Testing (白盒测试)

White Box Testing, also known as structural testing (结构测试), focuses on examining the internal workings of a program. Unlike black box testing, which only examines the program's outputs, white box testing allows us to see "inside" the program, understanding how it operates.

Path Testing (路径测试)

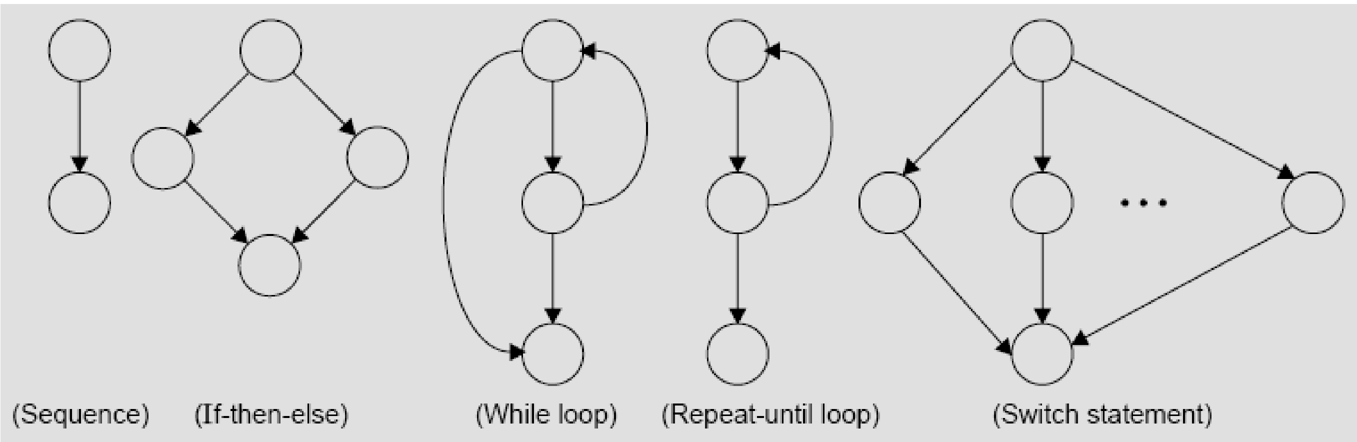
Path Testing is a technique where specific paths through a program are chosen and tested. This method ensures that all possible paths are covered, leading to a thorough testing process.

This involves: Creating paths that cover every part of the program. Finding test cases that will execute each path.

Flow Graph (流图)

A Flow Graph offers a graphical representation of a program's control flow, with nodes representing actions or decisions, and edges showing the flow from one action to the next. This helps in visualizing and understanding the program's structure.

Example:




```
        Fig1:- The basic construct of the flow graph
```

Example 1: Quadratic Equation (示例 1: 二次方程)

Consider determining the nature of roots for a quadratic equation. Inputs are three positive integers (a, b, c) within the range [0,100]. The outputs are categorized as:

> [Not a quadratic equation (不是二次方程); Real roots (实根); Imaginary roots (虚根); Equal roots (相等的根)]

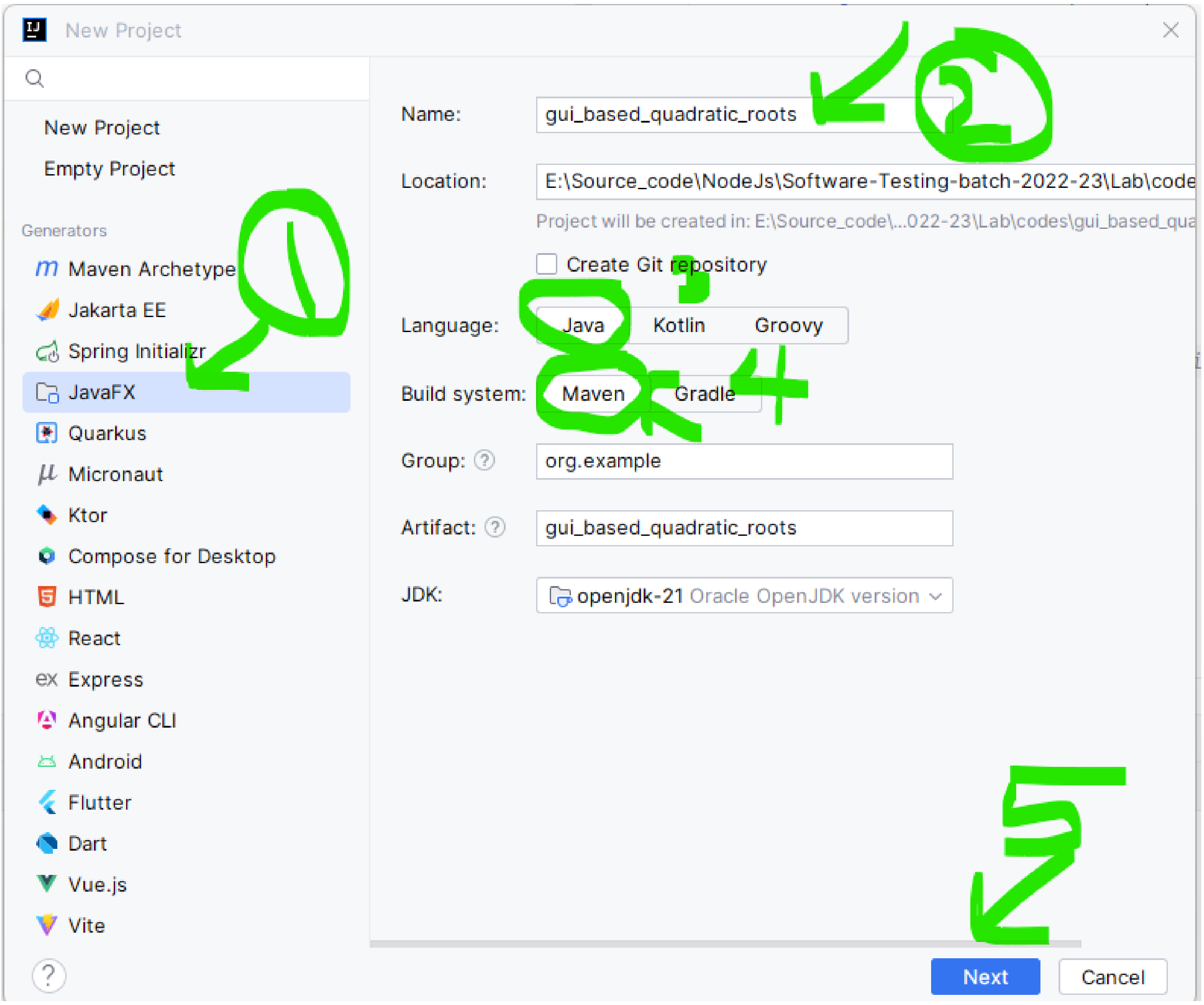
Your tasks include drawing the **flow graph**, **DD path graph**, and finding **independent paths** from the DD Path graph.

Solution: Start With Implementation (解决方案: 从实现开始)

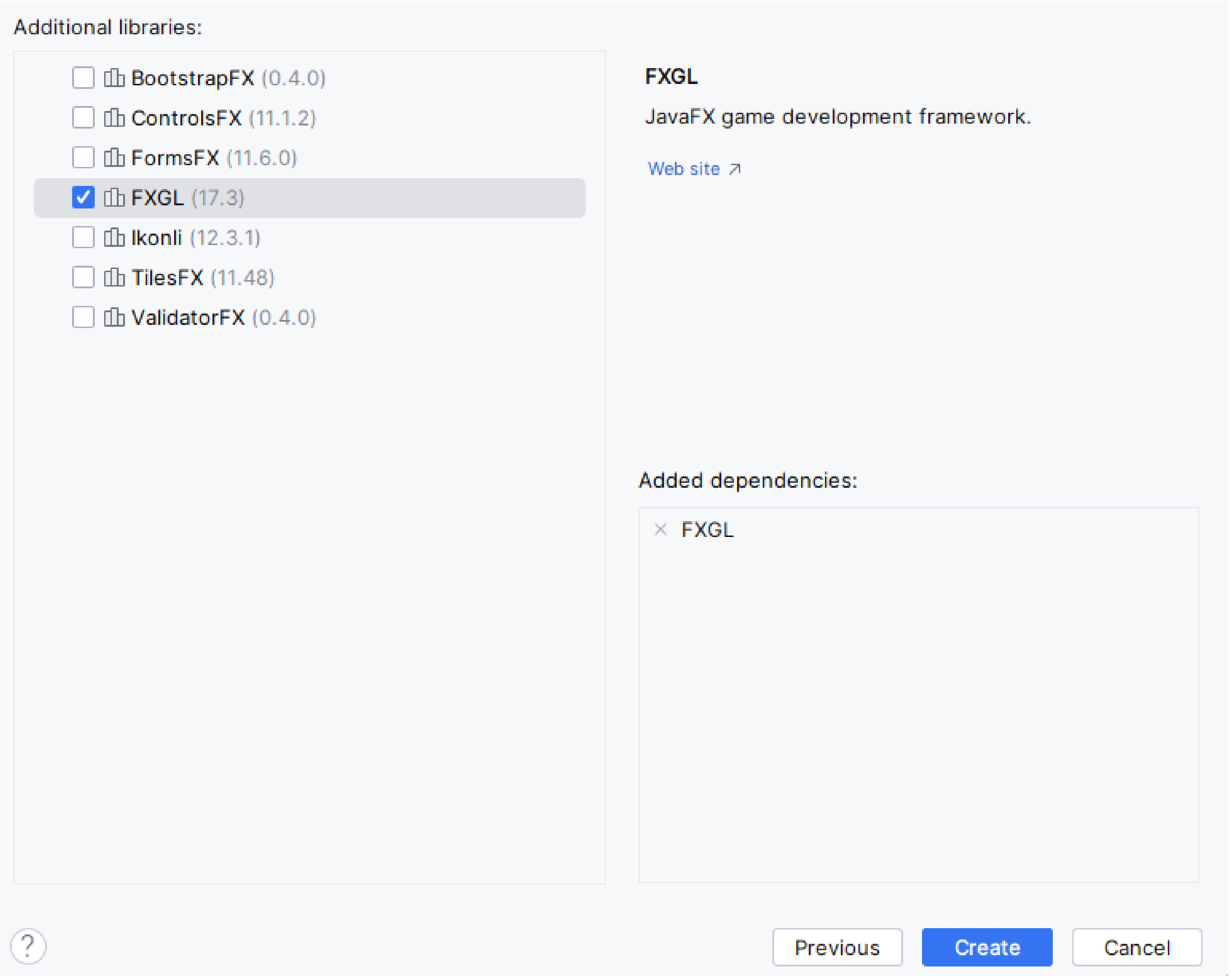
Let's use JavaFX for the GUI implementation, providing a graphical interface for our solution. This approach makes interaction easier and allows for a more intuitive understanding of the program's behavior.

Steps:

1. Create a New Project:



> - **Note:** Remember to include FXGL for additional features.

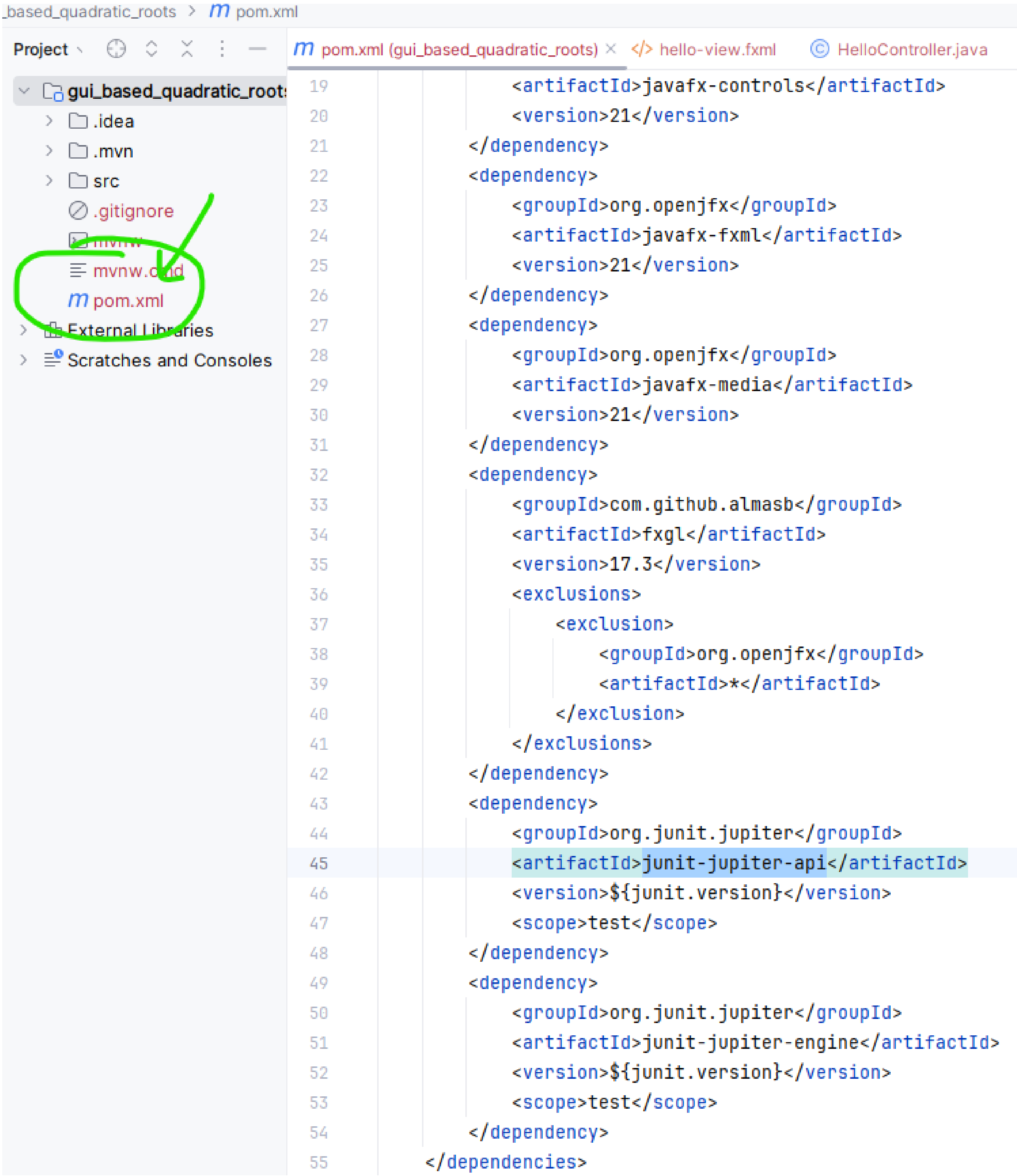


2. **Click 'Create'** to create a new JavaFX project

---

3. **Adding Dependencies:** Open the `pom.xml` file and ensure that the following dependencies are included:

> - `javafx-controls` - This module provides classes for JavaFX UI controls, including basic elements like buttons, text fields, labels, and more complex components like tables and trees.
> - `javafx-fxml`: This module enables your application to load FXML files at runtime, separating the UI design from the logic.
> - `junit-jupiter-api`: Part of JUnit 5 (also known as Jupiter), this library is used for writing and running tests in Java applications. It provides annotations and assertions for defining tests and verifying outcomes.

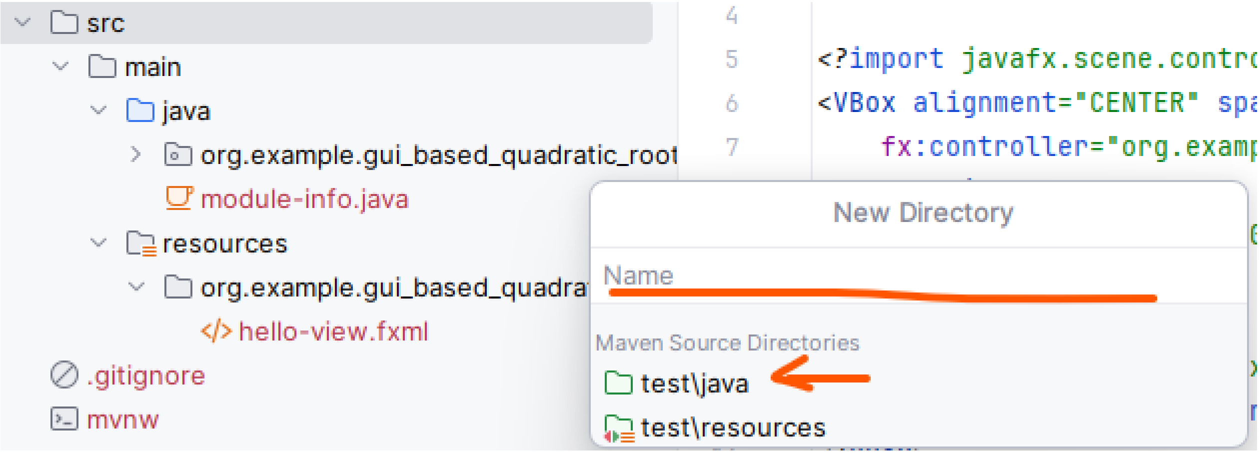


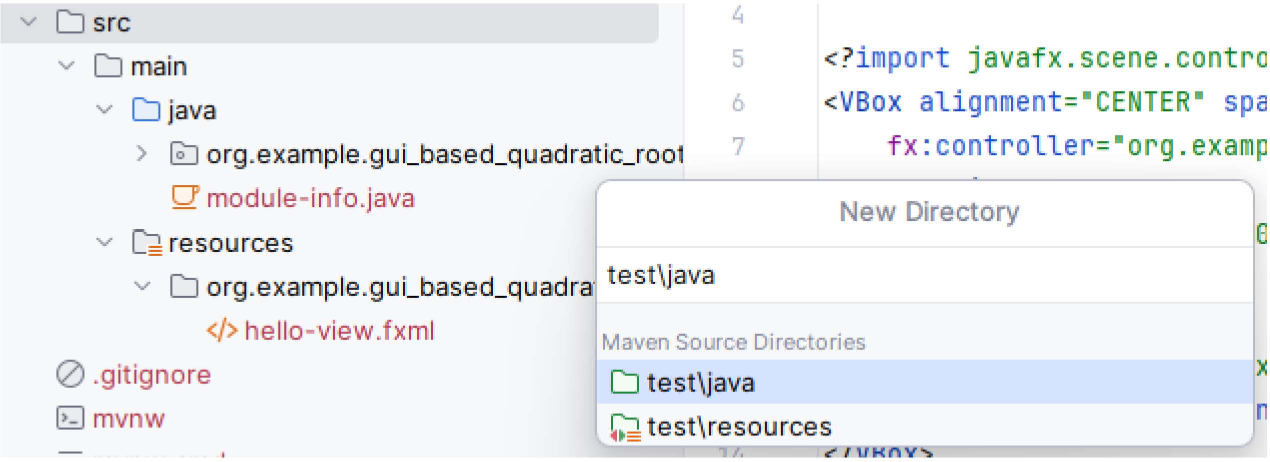
```xml
<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>21</version>
</dependency>
<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-fxml</artifactId>
    <version>21</version>
```

```xml
        </dependency>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-media</artifactId>
            <version>21</version>
        </dependency>
        <dependency>
            <groupId>com.github.almasb</groupId>
            <artifactId>fxgl</artifactId>
            <version>17.3</version>
            <exclusions>
                <exclusion>
                    <groupId>org.openjfx</groupId>
                    <artifactId>*</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
```

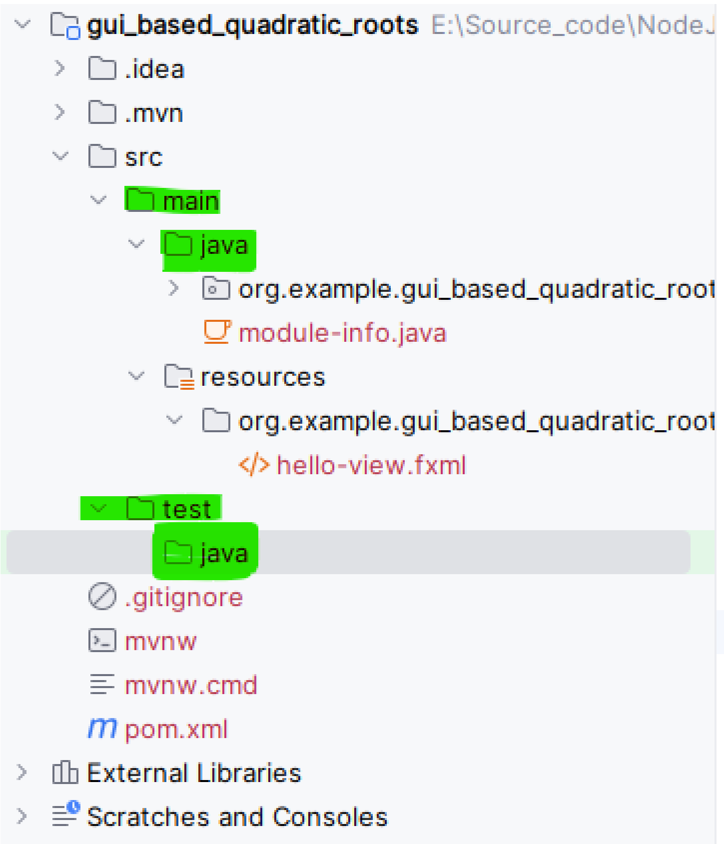
Step 4. **Setting Up Testing Environment:** Inside the `src` directory, create a new folder by navigating to and selecting the `test/java` directory to incorporate it into your project. Follow these steps: `Right-click` on `Src`, choose `New`, and then `left-click` on `Directory`.

> Write click on test\java



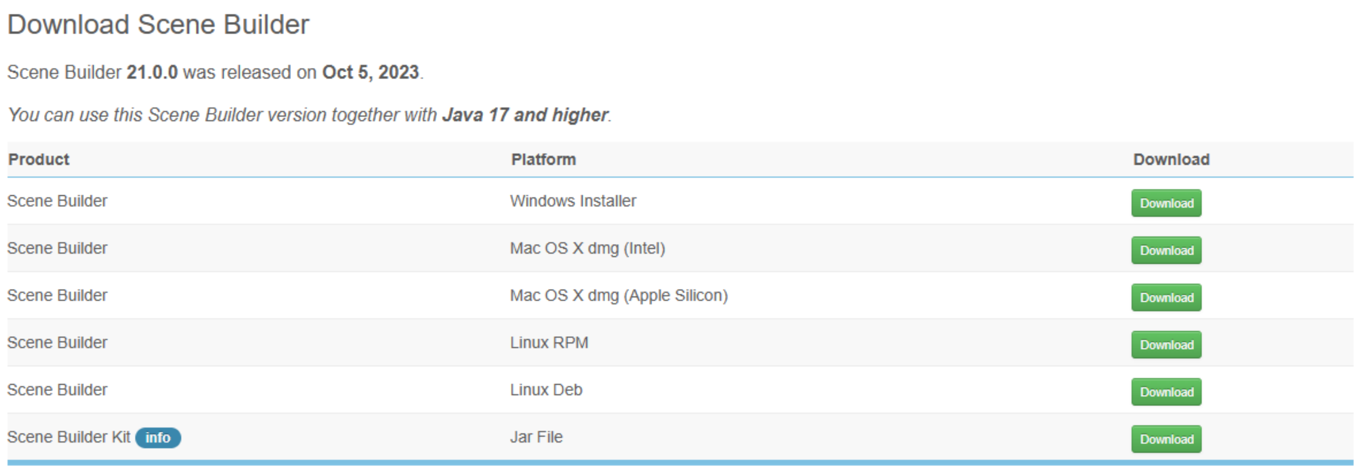


> Click **Enter** key to create a folder.



---

5. **Scene Builder Setup:** Download and install Scene Builder from Gluon.



Download Scene Builder

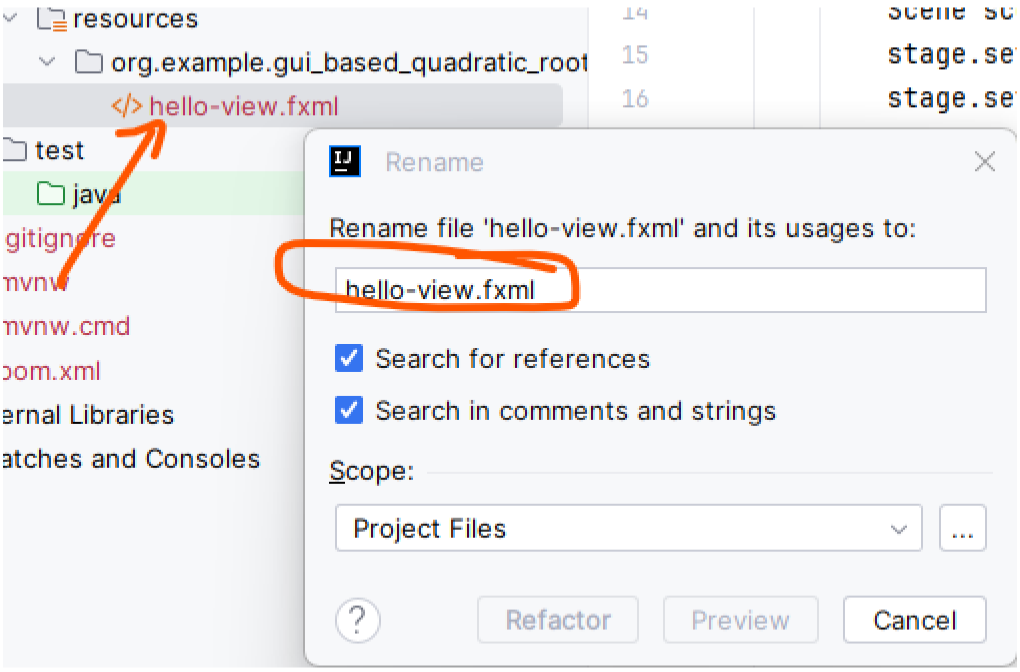
Scene Builder **21.0.0** was released on **Oct 5, 2023**.

*You can use this Scene Builder version together with **Java 17 and higher**.*

Product	Platform	Download
Scene Builder	Windows Installer	Download
Scene Builder	Mac OS X dmg (Intel)	Download
Scene Builder	Mac OS X dmg (Apple Silicon)	Download
Scene Builder	Linux RPM	Download
Scene Builder	Linux Deb	Download
Scene Builder Kit info	Jar File	Download

---

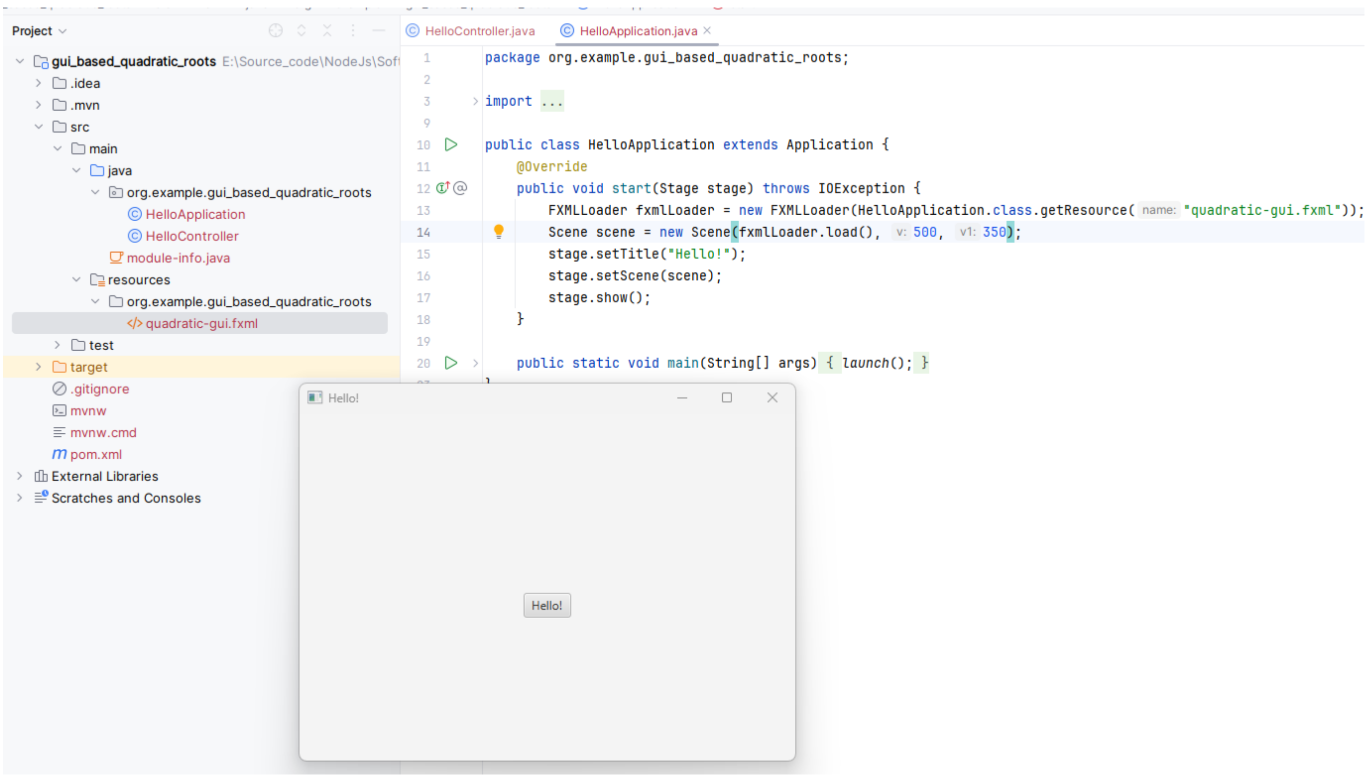
6. **Code Implementation:** Within the `src/resources` directory, rename the `.fxml` to match your UI layout.



- After renaming the file, navigate to the `src/main/java/package directory`, open `HelloApplication.java`, and execute the Java code to launch your GUI.

```java
public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(
            HelloApplication.class.getResource("quadratic-gui.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 500, 350);
        stage.setTitle("Hello!");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch();
```

```
        }
    }
```



Congratulations on setting up your first GUI! Continue by designing your UI with Scene Builder, focusing on making it user-friendly and intuitive.

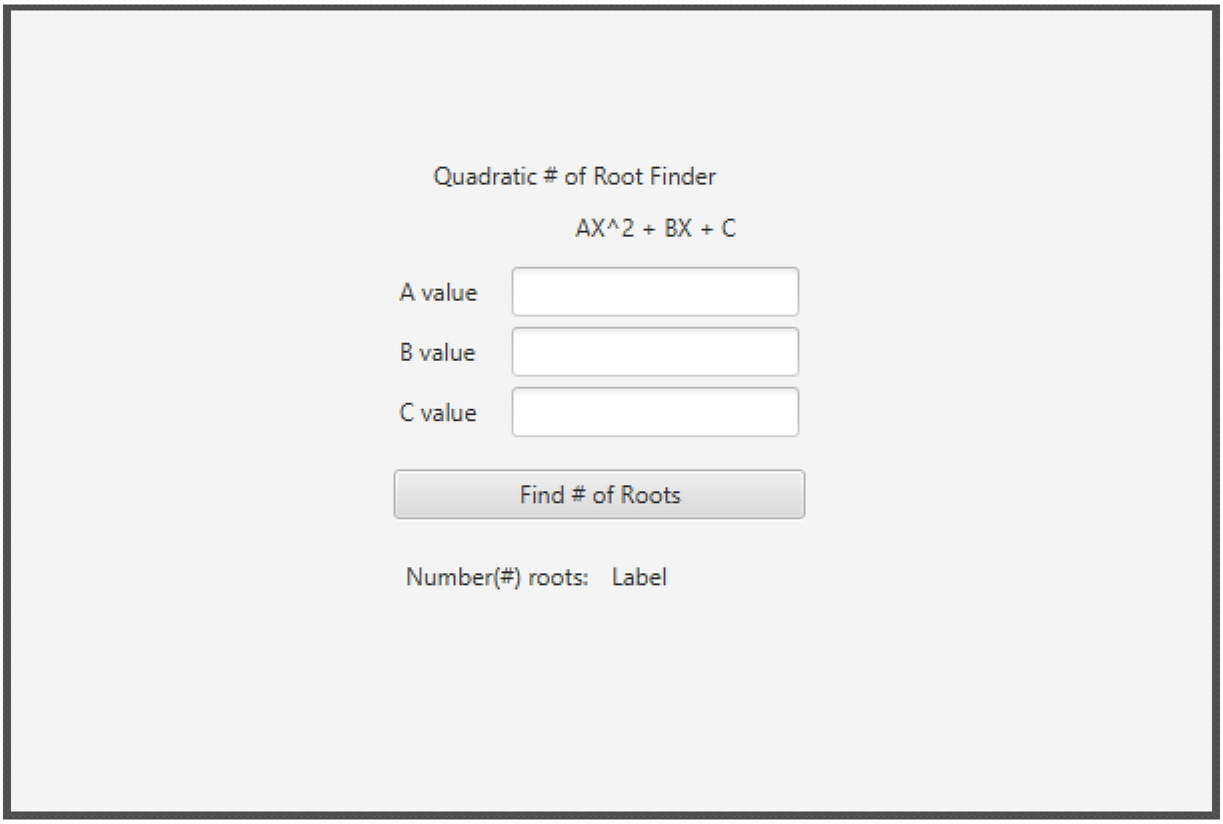
After launching your GUI with a "Hello" button, let's proceed to design the user interface (UI). Open quadratic-gui.fxml in Scene Builder to start. We'll use the AnchorPane layout for our first UI design.

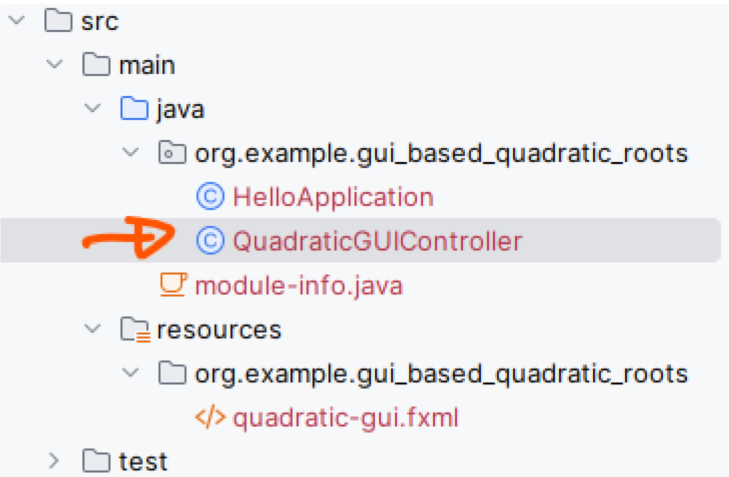
Choosing the Right JavaFX Layout Pane Selecting the appropriate JavaFX layout pane depends on your application's UI needs. Here’s a simplified guide:

1. **Simple, Linear Layouts (简单线性布局)**: `HBox` or `VBox` are perfect for rows of buttons or columns of text fields.
2. **Flexible Grids (灵活的网格)**: Use `GridPane` for layouts needing a grid arrangement, like forms.
3. **Dynamic Wrapping (动态包装)**: `FlowPane` allows elements to flow and wrap naturally, useful for adaptive layouts.
4. **Overlaying Elements (元素叠加)**: `StackPane` enables stacking components, ideal for overlays.
5. **Responsive Design (响应式设计)**: `AnchorPane` provides control for positioning elements, great for responsive UIs.
6. **Traditional Web Layout (传统网页布局)**: `BorderPane` suits web-style layouts with headers, footers, and content areas.
7. **Uniformly Sized Tiles (统一大小的瓦片)**: `TilePane` organizes items into a uniform grid.
8. **Scrollable Content (可滚动内容)**: Use `ScrollPane` for content exceeding display area.
9. **Divisible Areas (可分区域)**: `SplitPane` lets users adjust space between components.
10. **Tabulated Content (分页内容)**: `TabPane` organizes content into tabs for categorized information.

Designing your UI carefully will enhance the user experience by making the application intuitive and easy to navigate.

Example UI Implementation (quadratic-gui.fxml) Here's an example of how your quadratic-gui.fxml could look.





`quadratic-gui.fxml`

```
<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
    prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/21"
    xmlns:fx="http://javafx.com/fxml/1"
    fx:controller="org.example.gui_based_quadratic_roots.QuadraticGUIController">

    <children>
      <GridPane layoutX="194.0" layoutY="125.0">
        <columnConstraints>
          <ColumnConstraints hgrow="SOMETIMES" maxWidth="95.0" minWidth="10.0" prefWidth="56.0" />
          <ColumnConstraints hgrow="SOMETIMES" maxWidth="144.0" minWidth="10.0" prefWidth="144.0" />
        </columnConstraints>
        <rowConstraints>
          <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
          <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
          <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
        </rowConstraints>
         <children>
              <Label text="A value" />
              <Label text="B value" GridPane.rowIndex="1" />
            <Label text="C value" GridPane.rowIndex="2" />
              <TextField fx:id="txtA_input" GridPane.columnIndex="1" />
              <TextField fx:id="txtB_input" prefHeight="25.0" prefWidth="134.0" GridPane.columnIndex="1"
                  GridPane.rowIndex="1" />
            <TextField fx:id="txtC_input" GridPane.columnIndex="1" GridPane.rowIndex="2" />
         </children>
      </GridPane>
        <Button fx:id="onFIndRootButtonClick" layoutX="191.0" layoutY="229.0" mnemonicParsing="false"
            onAction="#onFIndRootButtonClick" prefHeight="25.0" prefWidth="206.0" text="Find # of Roots" />

      <Label layoutX="211.0" layoutY="74.0" text="Quadratic # of Root Finder" />
      <Label layoutX="282.0" layoutY="100.0" text="AX^2 + BX + C" />
      <Label layoutX="197.0" layoutY="274.0" text="Number(#) roots:" />
      <Label fx:id="lblOutput" layoutX="300.0" layoutY="274.0" />
    </children>
</AnchorPane>
```

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.RowConstraints?>

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
    prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/21"
    xmlns:fx="http://javafx.com/fxml/1"

fx:controller="org.example.gui_based_quadratic_roots.QuadraticGUIController">
```

```
    <children>
      <GridPane layoutX="194.0" layoutY="125.0">
        <columnConstraints>
          <ColumnConstraints hgrow="SOMETIMES" maxWidth="95.0"
minWidth="10.0" prefWidth="56.0" />
          <ColumnConstraints hgrow="SOMETIMES" maxWidth="144.0"
minWidth="10.0" prefWidth="144.0" />
        </columnConstraints>
        <rowConstraints>
          <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
          <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
          <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
        </rowConstraints>
         <children>
              <Label text="A value" />
              <Label text="B value" GridPane.rowIndex="1" />
            <Label text="C value" GridPane.rowIndex="2" />
              <TextField fx:id="txtA_input" GridPane.columnIndex="1" />
              <TextField fx:id="txtB_input" prefHeight="25.0"
prefWidth="134.0" GridPane.columnIndex="1"
                  GridPane.rowIndex="1" />
            <TextField fx:id="txtC_input" GridPane.columnIndex="1"
GridPane.rowIndex="2" />
         </children>
      </GridPane>
        <Button fx:id="onFIndRootButtonClick" layoutX="191.0"
layoutY="229.0" mnemonicParsing="false"
            onAction="#onFIndRootButtonClick" prefHeight="25.0"
prefWidth="206.0" text="Find # of Roots" />

      <Label layoutX="211.0" layoutY="74.0" text="Quadratic # of Root
Finder" />
      <Label layoutX="282.0" layoutY="100.0" text="AX^2 + BX + C" />
      <Label layoutX="197.0" layoutY="274.0" text="Number(#) roots:" />
      <Label fx:id="lblOutput" layoutX="300.0" layoutY="274.0" />
    </children>
</AnchorPane>
```

Implementing HelloApplication.java and QuadraticGUIController.java.

HelloApplication.java

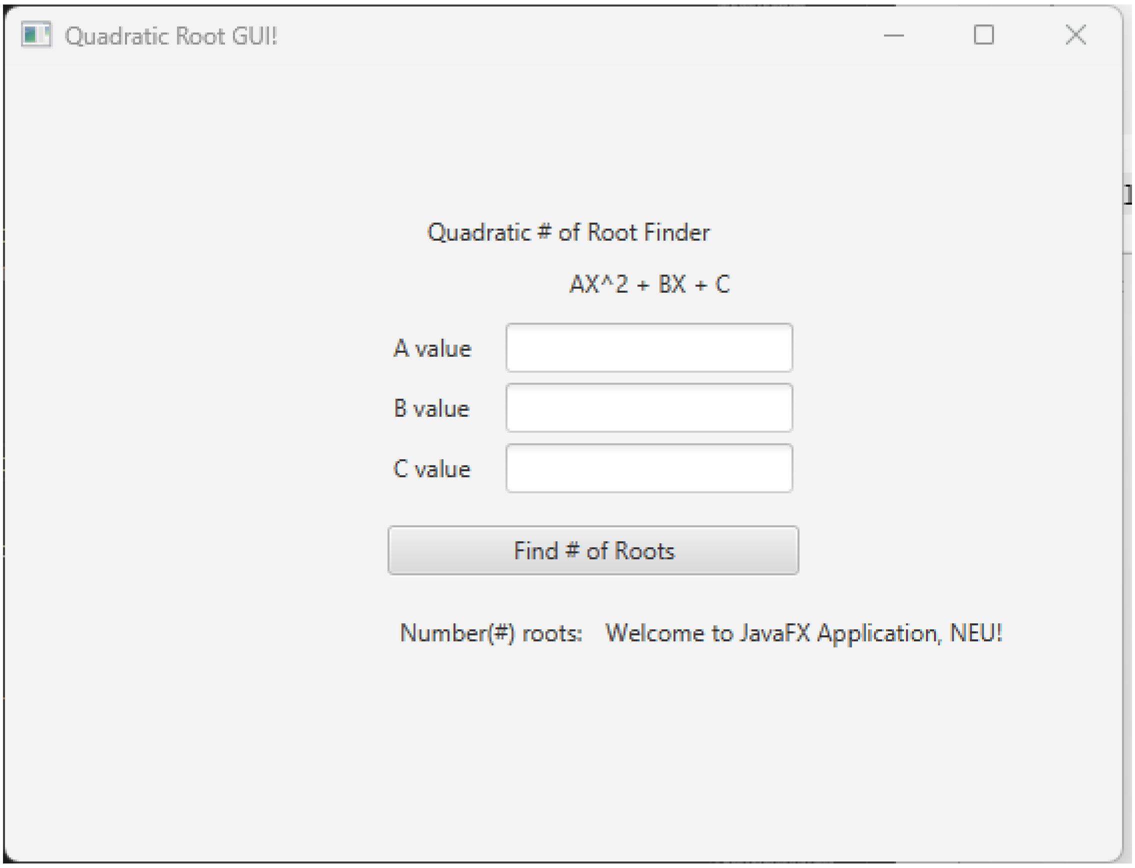
```
package org.example.gui_based_quadratic_roots;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
// import javafx.scene.layout.AnchorPane;
```

```
import javafx.stage.Stage;

import java.io.IOException;

public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        // Create an FXMLLoader instance to load the FXML file.
        // getClass().getResource("quadratic-gui.fxml") locates the FXML
file
        // in the same directory as the HelloApplication class.
        FXMLLoader loader = new
FXMLLoader(getClass().getResource("quadratic-gui.fxml"));

        // loader.setRoot(mew AnchorPage)  // if the root is not mentioned
inside fxml - start with fx.root

        // No need to manually set an AnchorPane as root here because the
root element is defined in the FXML file.
        // The load() method processes the FXML file and automatically
sets up the root element.
        // This root element is what we've defined inside the FXML with
<fx:root>.
        Parent root = loader.load();

        // Create a new Scene with the root element loaded from the FXML
file.
        // The dimensions are set to 500x350, but these can be adjusted as
needed.
        Scene scene = new Scene(root, 500, 350);

        // Set the title of the application window.
        stage.setTitle("Quadratic Root GUI!");

        // Set the scene for the stage. The scene contains all the visual
elements.
        stage.setScene(scene);

        // Display the application window.
        stage.show();
    }

    public static void main(String[] args) {
        // Launch the JavaFX application. This method creates an instance
of
        // HelloApplication and calls the start method on the JavaFX
Application Thread.
        launch();
    }
}
```

QuadraticGUIController.java

```
package org.example.gui_based_quadratic_roots;

import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;

public class QuadraticGUIController {
    @FXML
    private Label lblOutput;

    @FXML
    private TextField txtA_input;
    @FXML
    private TextField txtB_input;
    @FXML
    private TextField txtC_input;

    @FXML
    protected void onFIndRootButtonClick() {
        lblOutput.setText("Welcome to JavaFX Application, NEU!");
    }
}
```

Execute `HelloApplication.java` to **test** your GUI



It's time for the implementation of the `onFindRootButtonClick()` method of Example 1.

Algorithm for Quadratic Equation Roots Calculation (二次方程根的计算算法)

Input (输入) Coefficients `(a,b,c)` within the range `[1,100]`.
Output (输出) # of Roots and Roots of the quadratic equation, or a message indicating no real roots or invalid input.

1. Inputs a, b, c
2. variables isValid=0, d, D
3. if ((a >= 0) && (a <=100) && (b >= 0) && (b <=100) && (c >= 0) && (c <=100))
4. isValid = 1

5. **if (a == 0)**
6. isValid = -1
7. **end if**
8. end if
9. if (isValid == 1)
10. d = b^2 - 4ac

11. **if (d ==0)**
12. return `The roots are equal r: {root}`
13. **else if(d > 0)**
14. D = sqrt(d)
15. return `2 real roots r1: {root} and r2: {root}`

16. **else**
17. D = sqrt(-d)

18. return `2 imaginary roots r1: {root} and r2: {root}`
19. **end else if**
20. else if (validInput == -1)
21. return `The values do not constitute a Quadratic equation`

22. else
23. return `The inputs belong to invalid range`
24. end else if |

Implementing `onFindRootButtonClick()` in QuadraticGUIController.java

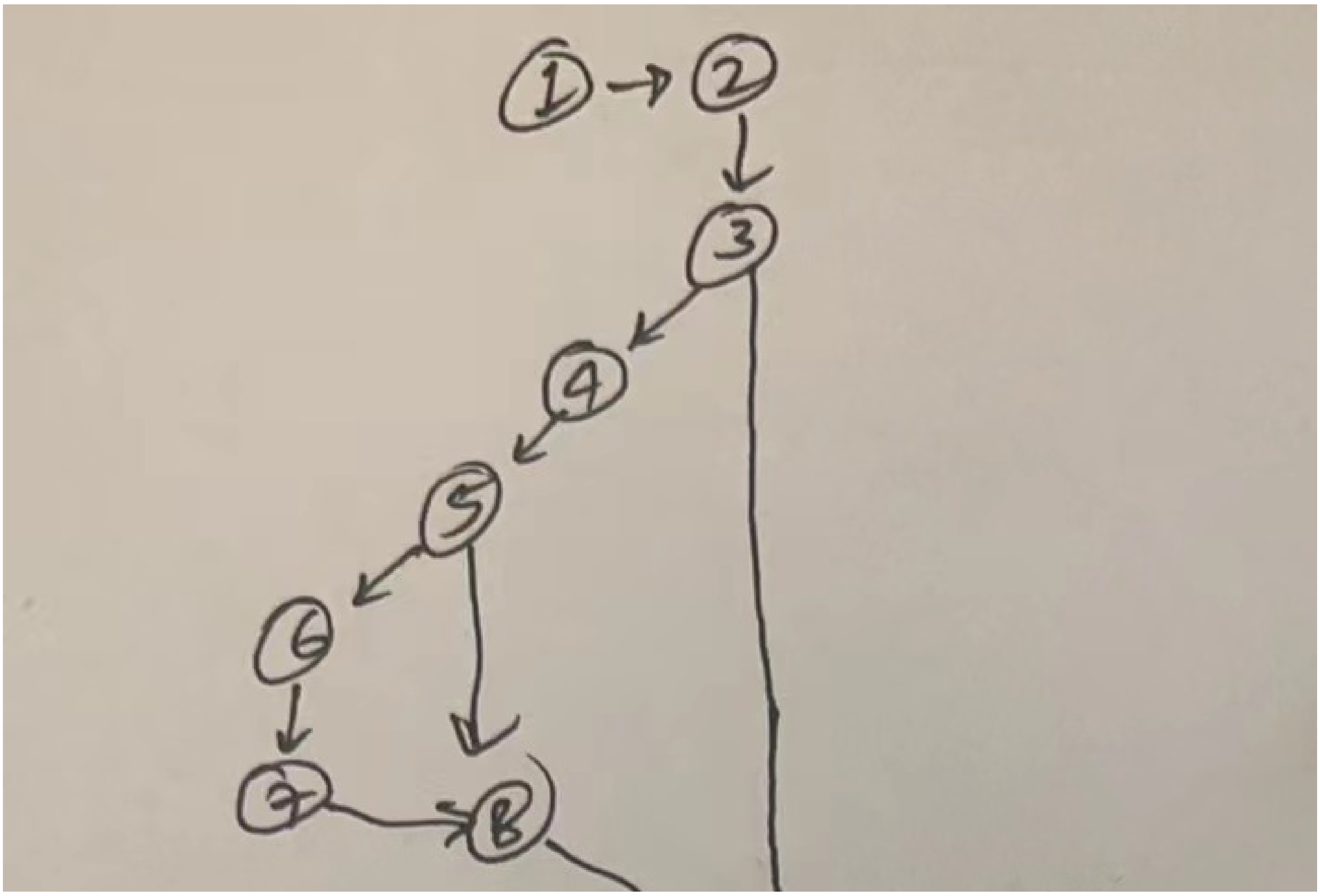
```
public class QuadraticGUIController {
    @FXML
    private Label lblOutput;

    @FXML
    private TextField txtA_input;
    @FXML
    private TextField txtB_input;
    @FXML
```

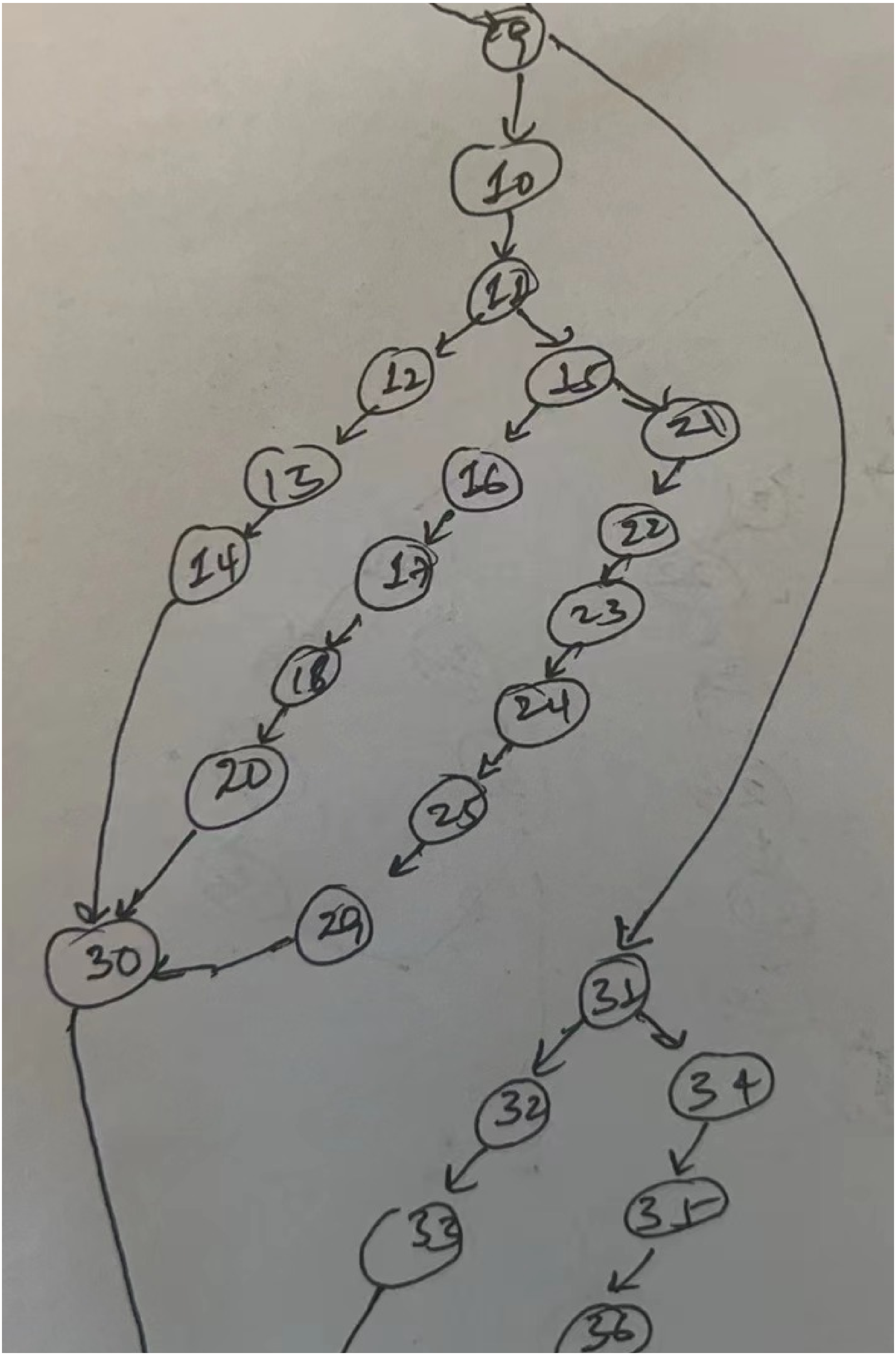
```
    private TextField txtC_input;

    @FXML
    protected void onFIndRootButtonClick(){

        String a = txtA_input.getText();
        String b = txtB_input.getText();
        String c = txtC_input.getText();

        String result = findRoots(a, b, c);

        lblOutput.setText(result);

    }

    public String findRoots(String a, String b, String c) throws
NumberFormatException {
    try {

        double A = Double.parseDouble(a);
        double B = Double.parseDouble(b);
        double C = Double.parseDouble(c);

        // Implementation


    } catch (NumberFormatException e) {
        // Raise when coonverting from String to Double / Integer
        return "The input must be number";
    }
    }

    public String quadraticRoots(double a, double b, double c) {
      // Implementation
    }

}
```

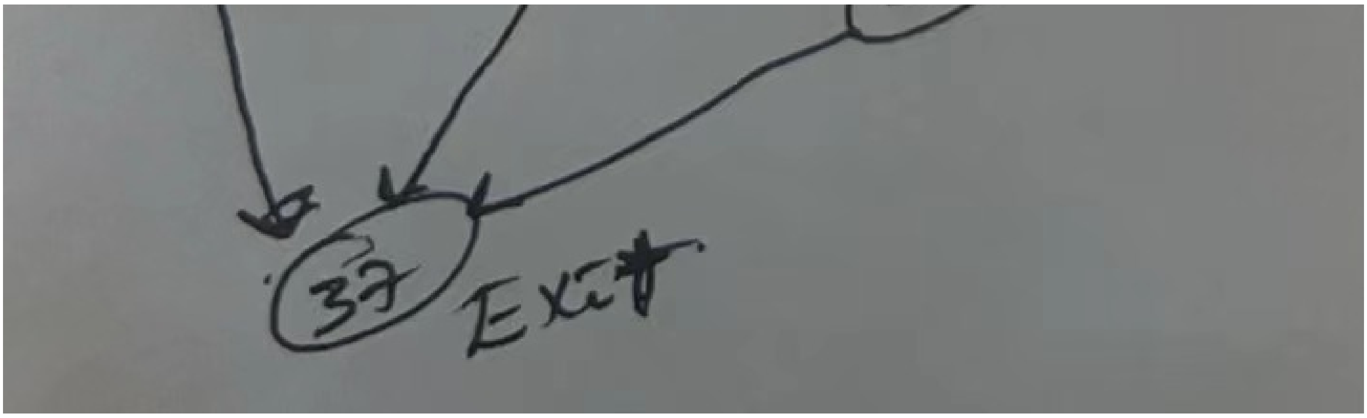
```
int isValid = 0 // int a, b, c;
double d, D;
if ((a >= 0) && (a <= 100) && (b >= 0) && (b <= 100) && (c >= 0) && (c <= 100)) {
 isValid = 1;
 if (a == 0) {
     isValid = -1;
 }
}
if (isValid == 1) {
     d = b * b - 4 * a * c;
     if (d == 0) {
```

```
        return "One Real Root, r = " + String.format("%.2f", -b /
(2*a));
    }
    else if (d > 0) {
    D = Math.sqrt(d);
        return "2 Real Roots, r1 = " + String.format("%.2f", (-b + D)
/ (2 / a)) +
            " and r2 = " + String.format("%.2f", (-b - D) / (2 / a));
    }
    else {
        D = Math.sqrt(-1 * d) / (2 * a);
        return "2 Imaginary Roots, r1 = " + String.format("%.2f", -1*b
/ (2 * a)) + " + " +
            String.format("%.2f", D) + "i and r2 = " +
            String.format("%.2f", -1*b / (2 * a)) + " - " +
            String.format("%.2f", D) + "i";
    }
  }
  else if (isValid == -1) {
    return "The Equation is not a quadratic equation";
}
else {
    return "The input is out of range";
}
```

Draw a program **flow graph:** Visualize the control flow of the above logic using a flow graph. This graphical representation helps in understanding the possible execution paths through the program.




9
10
11
12
15
13
16
21
14
17
22
23
18
24
20
25
29
30
31
32
34
33
35
36



Decision-to-Decision 决策到决策路径 (DD) Path:

A Decision-to-Decision path in software testing is a control flow path from one decision point to another, excluding loops. It's used in analyzing and understanding the logical flow of a program for creating test cases that cover different paths, enhancing test coverage.

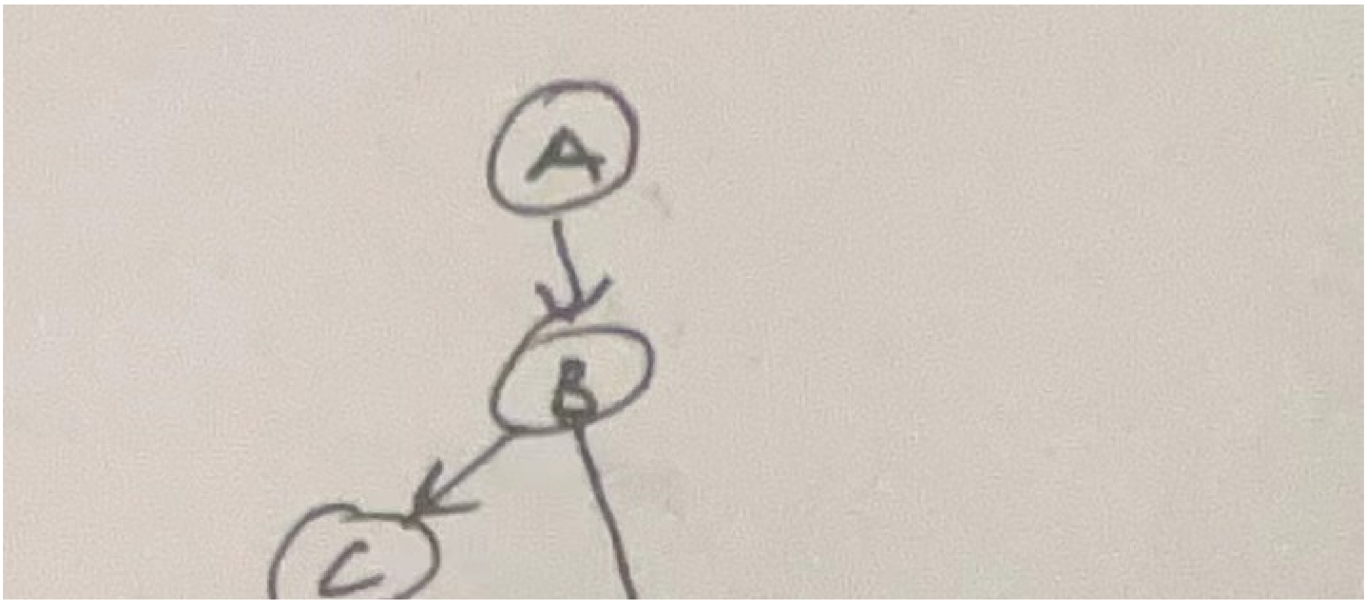
DD paths are crucial for identifying the ***minimum set of paths*** that need to be tested to ensure that all decision points are covered. It's particularly useful in ***cyclomatic complexity***, a metric used to measure the complexity of a program based on the control flow graph's decision points.

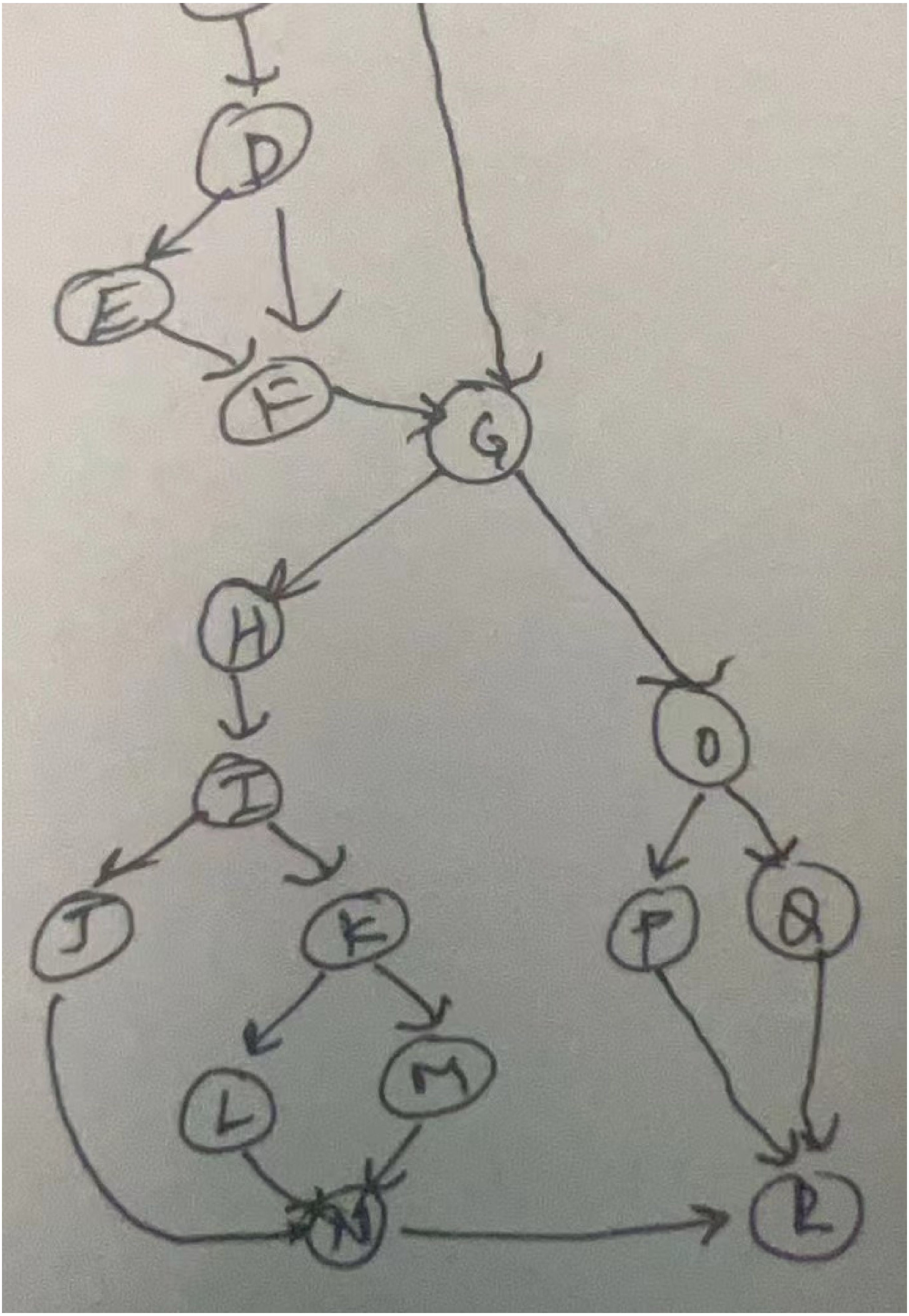
Decision Points and Control Flow *Decision Points:* In a program, decision points are locations where the control flow can diverge, meaning the program can follow one of two or more paths based on some condition (e.g., if-else statements, switch cases). Control Flow Path: The sequence of execution from one point in a program to another.

Excluding Loops *Loops:* Constructs like for, while, and do-while that repeatedly execute a block of code as long as a specified condition remains true.

Excluding Loops in DD Paths: When defining or analyzing DD paths, we typically focus on the paths that flow directly from one decision point to the next without reiterating the parts of the code within loops. This exclusion is because loops can theoretically extend the path infinitely (or as long as the loop's condition is met), complicating the analysis of control flow for test coverage.

Decision-to-Desicion (DD) graph



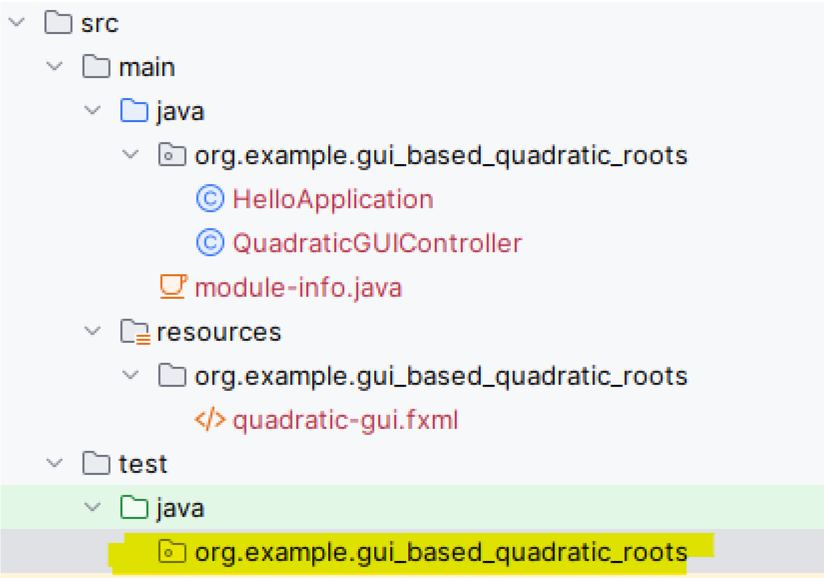

D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R

The Mapping table for DD path graph is:

Flow graph nodes	DD path graph corresponding node	Remarks
1 to 2	A	Sequential node
3	B	Decision node
4	C	Intermediate node
5	D	Decision node
6, 7	E	Sequential node
8	F	Two edges are combined here
9	G	Two edges are combined and decision node
10	H	Intermediate node
11	I	Decision node
12, 13, 14	J	Sequential node
15	K	Decision node
16, 17, 18, 20	L	Sequential node
21, 22, 23, 24, 25, 29	M	Sequential node
30	N	Three edges are combined
31	O	Decision node
32, 33	P	Sequential node
34, 35, 36	Q	Sequential node
37	N	Three edges are combined here with exit node

Independent Paths: (i) ABGOQRS (ii) ABGOPRS (iii) ABCDFGOQRS (iv) ABCDEFGOPRS (v) ABGHIJNRS (vi) ABGHIKLNRS (vi) ABGHIKMNRS

It's time to create a test package




```java
package org.example.gui_based_quadratic_roots;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class QuadraticGUIControllerTest {

    @Test
    public void path1() {

    }
}
```

Class Activity Consider a program given in Fig.8.20 for the classification of a triangle. Its input is a triple of positive integers (say a,b,c) from the interval [1,100]. The output may be

> [Scalene, Isosceles, Equilateral, Not a triangle].

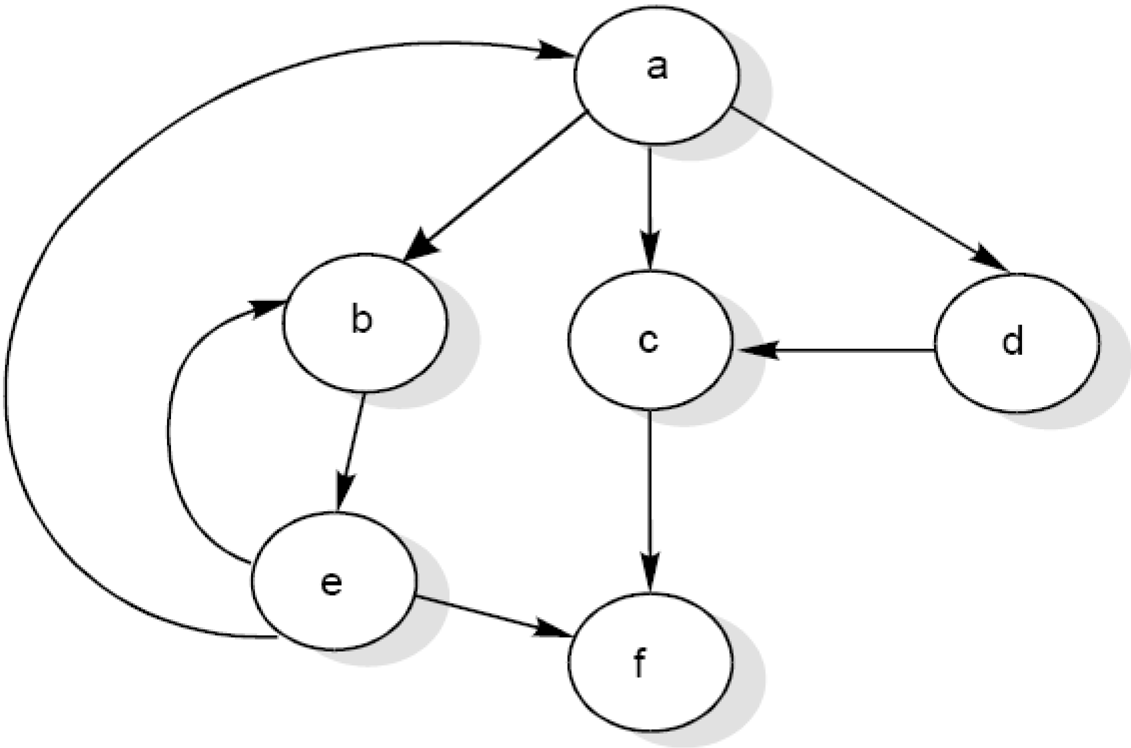
Draw the flow graph & DD Path graph. Also find the independent paths from the DD Path graph.

Cyclomatic Complexity (环路复杂度)

McCabe's cyclomatic metric V(G) = e – n + 2P, measures the number of linearly independent paths through a program's source code. This metric is crucial for determining the thoroughness of the test cases developed based on the control flow graph of the program.

> - P is the number of disconnected components or graphs, typically 1 for a cohesive, single-component program.

For example, a flow graph shown in in Fig below with entry node ‘a’ and exit node ‘f’.



The value of cyclomatic complexity can be calculated as : ***V(G) = 9 – 6 + 2 = 5*** Here `e = 9`, `n = 6` and `P = 1` There will be five independent paths for the flow graph illustrated in Fig. 21. `Path 1 : a c f` `Path 2 : a b e f` `Path 3 : a d c f` `Path 4 : a b e a c f or a b e a b e f` `Path 5 : a b e b e f`

> Note: ***P*** is the number of `disconnected subgraphs` or `connected components` within the **control flow graph** of the program. For most single, self-contained programs, `P` is typically equal to **1**, indicating one connected component that is the entire program itself.
>
> - Therefore, for a single, unified program that doesn't consist of separate disconnected parts, you would use ***P=1*** when calculating cyclomatic complexity using McCabe's formula.

Several properties of cyclomatic complexity are stated below:

> 1. V(G) ≥1 (V(G) 不小于 1)
> 2. V (G) is the maximum number of independent paths in graph G. (独立路径的最大数量)
> 3. Inserting & deleting functional statements to G does not affect V(G). (增删非决策语句不影响V(G))
> 4. G has only one path if and only if V(G)=1. (当 V(G)=1 时，只有一条路径)
> 5. Inserting a new row in G increases V(G) by unity. (每增加一个决策，V(G)增加1)
> 6. V(G) depends only on the decision structure of G. (仅取决于决策结构)

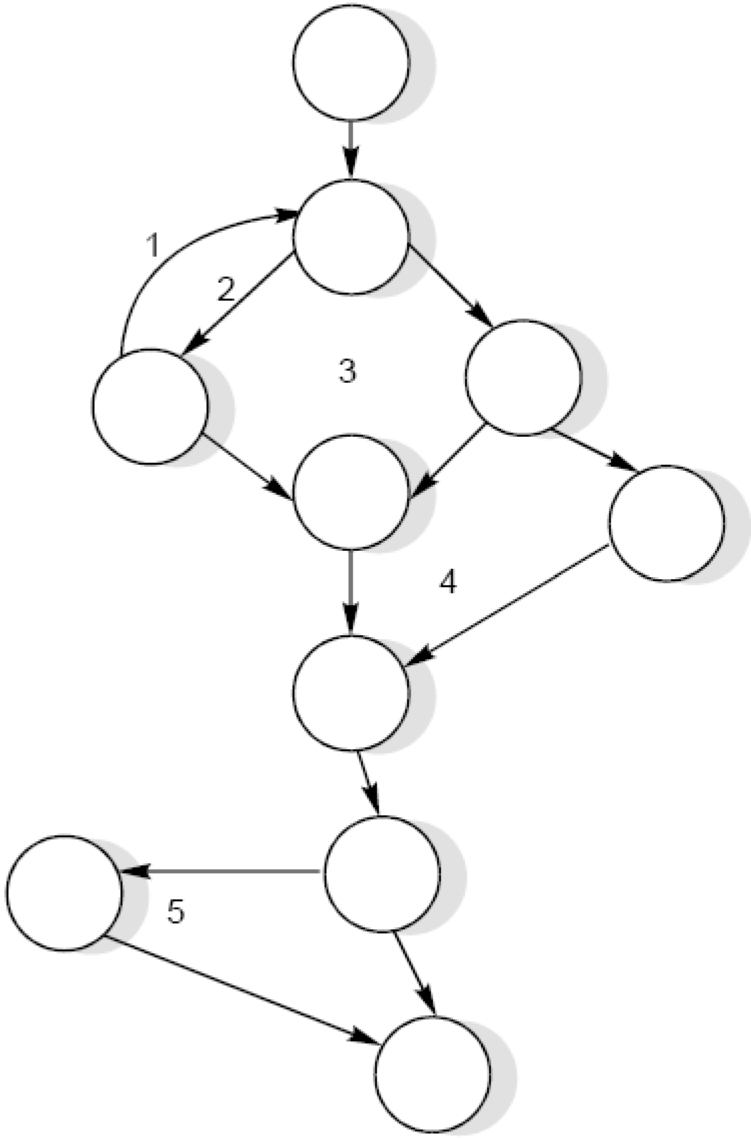
Two alternate methods are available for the complexity calculations.

1. Cyclomatic complexity V(G) of a flow graph G is equal to the number of predicate (decision) nodes plus one.

   > V(G)= $\prod$ +1 Where $\prod$ is the number of predicate nodes contained in the flow graph G.

2. Cyclomatic complexity is equal to the number of regions of the flow graph.

Example 3 Consider a flow graph given in Fig below and calculate the cyclomatic complexity by all three



methods.

Solution Cyclomatic complexity can be calculated by any of the three methods.

1. V(G) = e − n + 2P = 13 − 10 + 2 = 5
2. V(G) = π + 1 = 4 + 1 = 5
3. V(G) = number of regions = 5 Therefore, complexity value of a flow graph in Fig. above is 5.

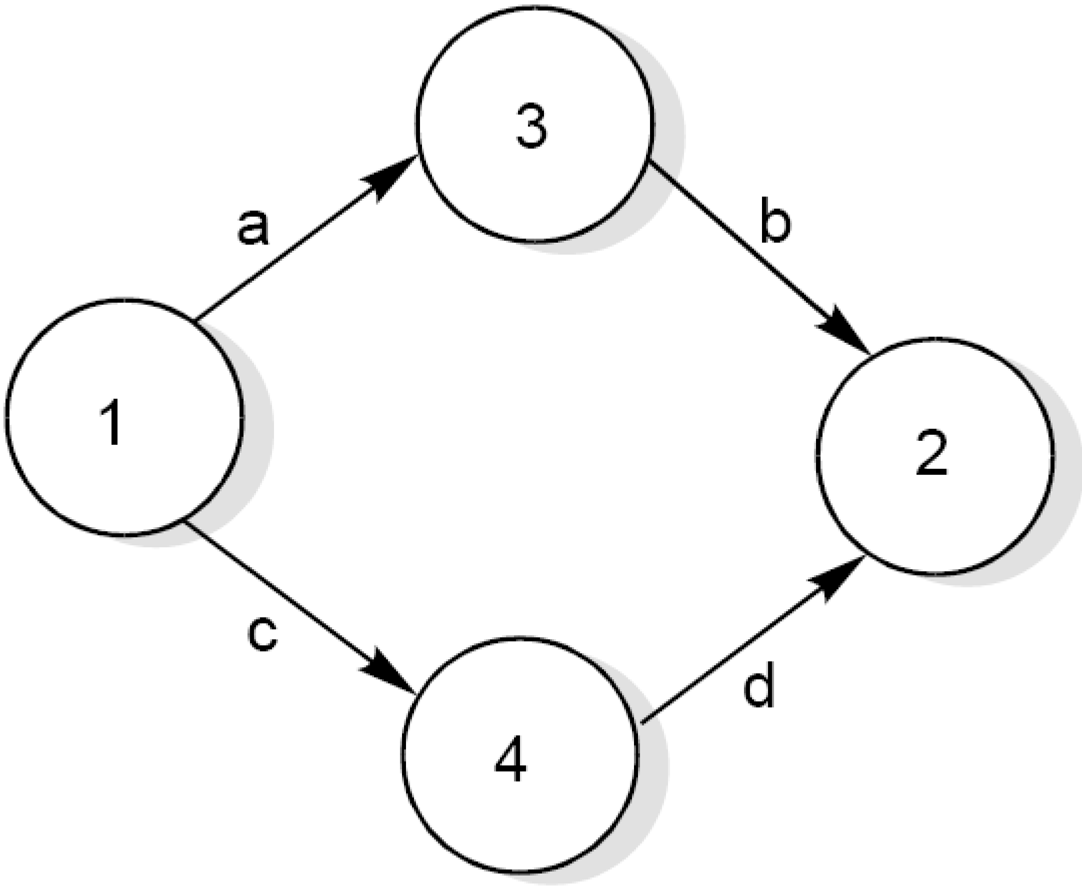
Example Consider the quadratic equation problem given with its DD Path graph. Find the cyclomatic complexity:

Solution Number of nodes (n) = 19 Number of edges (e) = 24 (i) V(G) = e − n + 2P = 24 − 19 + 2 = 7 (ii) V(G) = π + 1 = 6 + 1 = 7 (iii) V(G) = Number of regions = 7 Hence cyclomatic complexity is 7 meaning thereby, seven independent paths in the DD Path graph.

Example: Class Activity Consider the classification of triangle problem given. Find the cyclomatic complexity.

Graph Matrices

A graph matrix is a square matrix with one row and one column for every node in the graph. The size of the matrix (i.e., the number of rows and columns) is equal to the number of nodes in the flow graph. Some examples of graphs and associated matrices are shown in fig. _.

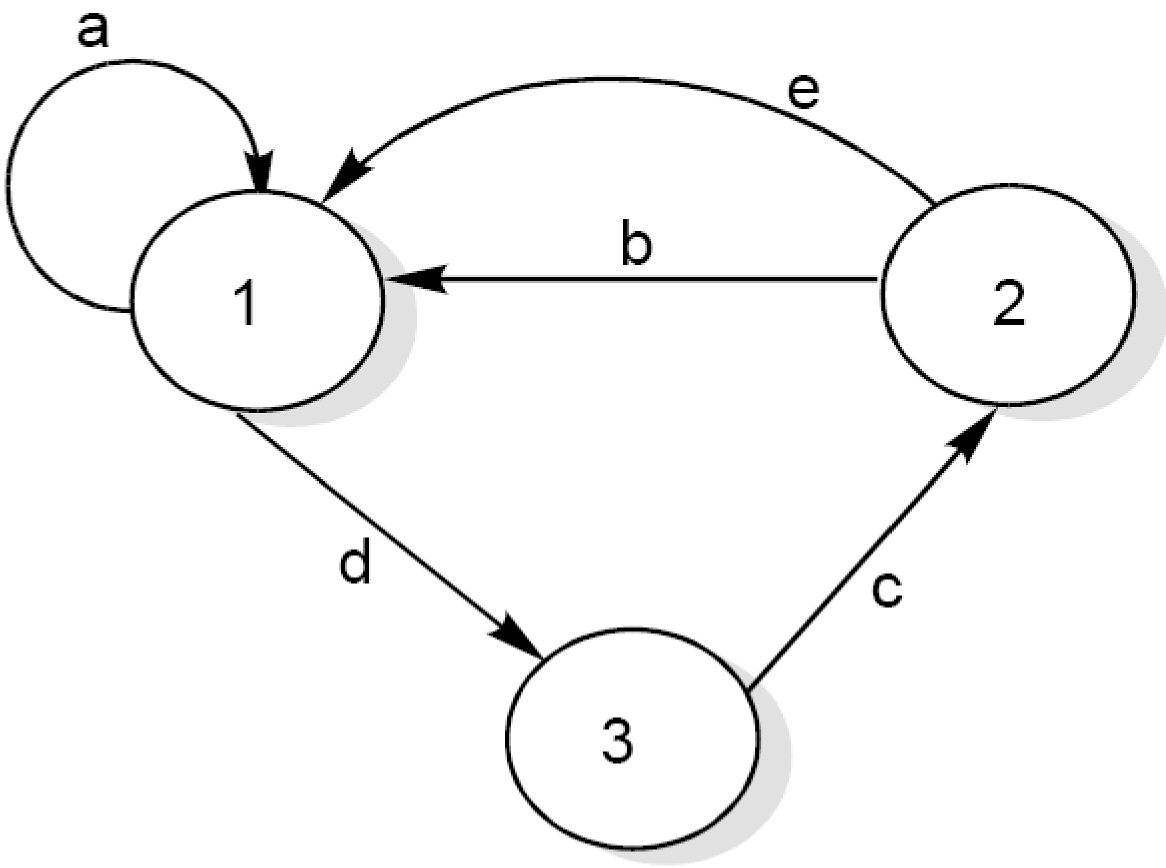


Flow graph

	1	2	3	4
1			a	c
2				
3		b		
4		d		

Graph Matrix

Fig. 24 (a): Flow graph and graph matrices

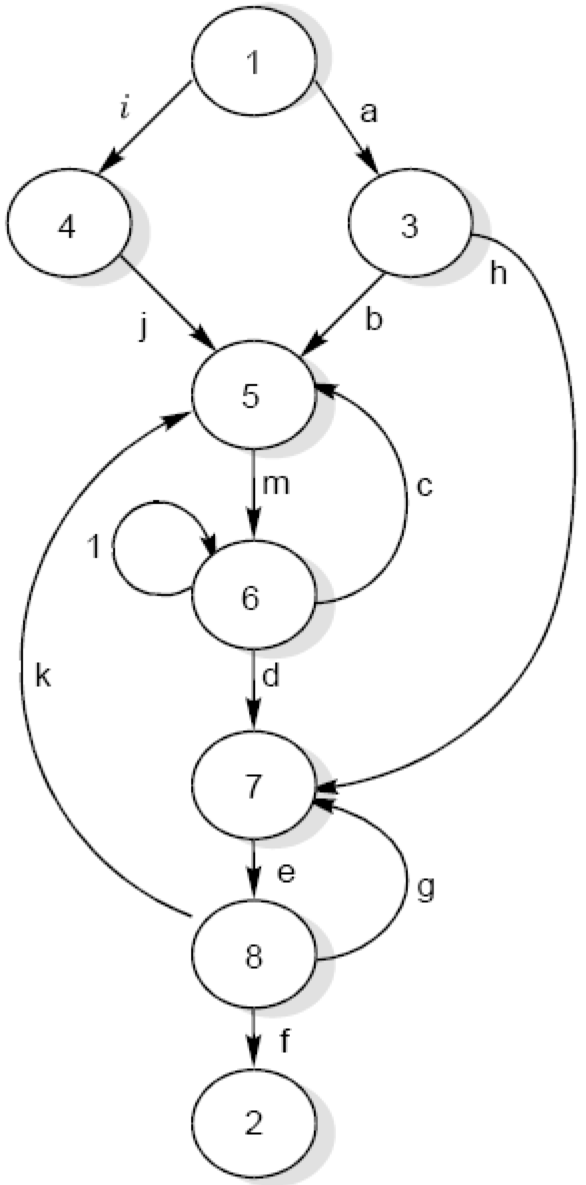


Flow graph

	1	2	3
1	a		d
2	b + e		
3		c	

Graph Matrix

Fig. 24 (b): Flow graph and graph matrices



Flow graph

	1	2	3	4	5	6	7	8
1			a	i				
2								
3					b		h	
4					j			
5						m		
6					c	l	d	
7								e
8		f			k		g	

Graph Matrix

Fig. 24 (c): Flow graph and graph matrices

	1	2	3	4	5	6	7	8	Connections
1			1	1					2 – 1 = 1
2									
3					1		1		2 – 1 = 1
4					1				1 – 1 = 0
5						1			1 – 1 = 0
6					1	1	1		3 – 1 = 2
7								1	1 – 1 = 0
8		1			1		1		3 – 1 = 2
									6 + 1 = 7

Fig. 25 : Connection matrix of flow graph shown in Fig. 24 (c)

Data Flow Testing

Data flow testing is another from of structural testing. It has nothing to do with data flow diagrams.

> i. It focuses on the points at which variables receive values ii. The points at which these values are used or referenced.

> - It detects improper use of data values (data flow anomalies) due to coding errors.

Early data flow analyses often cantered on a set of faults that are known as define / reference anomalies.

> - A variable that is defined but never used (referenced)
> - A variable that is used but never defined
> - A variable that is defined twice before it is used.

Definitions

The definitions refer to a program P that has a program graph G(P) and a set of program variables V. The G(P) has a single entry node and a single exit node. The set of all paths in P is PATHS(P)

> **(i) Defining Node**: Node `n ∈ G(P)` is a defining node of the variable `v ∈ V`, written as **`DEF (v, n)`**, if the value of the variable `v` is defined at the statement fragment corresponding to node `n`.

> **(ii) Usage Node**: Node `n ∈ G(P)` is a usage node of the variable `v ∈ V`, written as **`USE (v, n)`**, if the value of the variable `v` is used at statement fragment corresponding to node `n`. A usage node `USE (v, n)` is a predicate use (denote as `p`) if statement `n` is a predicate statement otherwise `USE (v, n)` is a computation use (denoted as `c`).

> **(iii) Definition use**: A definition use path with respect to a variable **`v (denoted du-path)`** is a path in `PATHS(P)` such that, for some `v ∈ V`, there are define and usage nodes `DEF(v, m)` and `USE(v, n)` such that `m` and `n` are ***initial*** and ***final nodes*** of the path.

> **(iv) Definition clear** : A definition clear path with respect to a variable **`v (denoted dc-path)`** is a definition use path in `PATHS(P)` with **initial** and **final** nodes `DEF(v, m)` and `USE(v, n)`, such that no other node in the path is a defining node of `v`.

> - The **du-paths** and **dc-paths** describe the flow of data across source statements from points at which the values are **defined** to points at which the values are **used**.
> - The **du-paths** that are ***not definition clear*** are potential trouble spots.

Hence, our objective is to find all du-paths and then identity those du-paths which are not dc-paths. We may like to generate specific test cases for du-paths that are not dc-paths.

> ***Steps for data flow testing:*** Step 1: Draw the program flow graph Step 2: Find the DD path graph Step 3: Prepare a table for define/usse status of all variable Step 4: Find all du-paths Step 5: Identify-du pathss that are not dc paths

---

Example

Consider the program of the determination of the nature of roots of a quadratic equation. Its input is a triple of positive integers (say a,b,c) and values for each of these may be from interval [0,100]. The program is given in **Fig. 19.** The output may have one of the option given below:

> (i) Not a quadratic program (ii) real roots (iii) imaginary roots (iv) equal roots (v) invalid inputs

Find all **du-paths** and identify those du-paths that are **definition clear**.

Class Activity Consider the program given in Fig. 20 for the classification of a triangle. Its input is a triple of positive integers (say a,b,c) from the interval [1,100]. The output may be:

> [Scalene, Isosceles, Equilateral, Not a triangle, Invalid inputs].

Find all du-paths and identify those du-paths that are definition clear.

Solution Step I: The program flow graph is given in Fig. 19 (a). The variables used in the program are a,b,c,d, validinput, D.

Step II: DD Path graph is given in Fig. 19(b). The cyclomatic complexity of this graph is 7 indicating there are seven independent paths.

Step III: Define/use nodes for all variables are given below:

Variable	Defined at node	Used at node
a	param - 1	3, 5, 10, 12, 16, 19, 20
b	param - 1	3, 10, 12, 16, 20
c	param - 1	3, 10
d	10	11, 14, 15, 16, 17
D	15, 20	16, 20, 21
isValid	1, 4, 6	9, 23

Step IV: The du-paths are identified and are named by their beginning and ending nodes using Fig. 19 (a).

Variable	Path (beginning, end) nodes	Definition clear ?
a	param - 1, 3	Yes
a	param - 1, 5	Yes