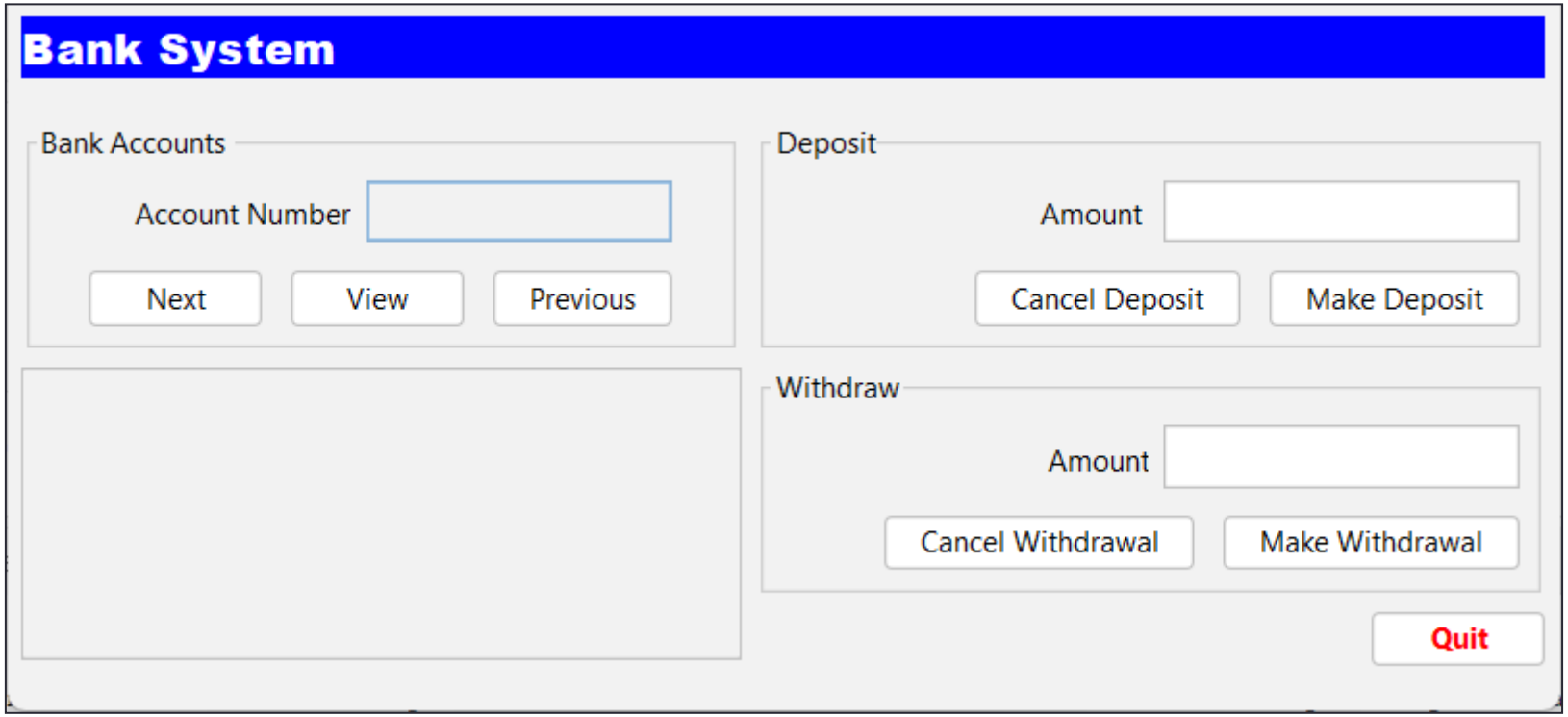
INTRODUCTION TO JAVA GUI – PART 1

Swing Package

Overview

- Using the Bank System example, we'll build a simple Graphical User Interface (GUI) application which will demonstrate the following components:
  - `JFrame`
  - `JPanel`
  - `Jbutton`
  - `JTextField`
  - `Jlabel`
  - `JTextArea`
  - Event Handling
  - User Defined Methods

Bank System GUI



JFRAME

JFrame

Desktop GUI applications are built on a JFrame class

- Which we need to add to our Project

Combine the Controller and Boundary into the JFrame class

- I.e. the JFrame class will also be the main class
- This is why we do not create a Java Frontend Application in NetBeans

Like console application the main method will be static

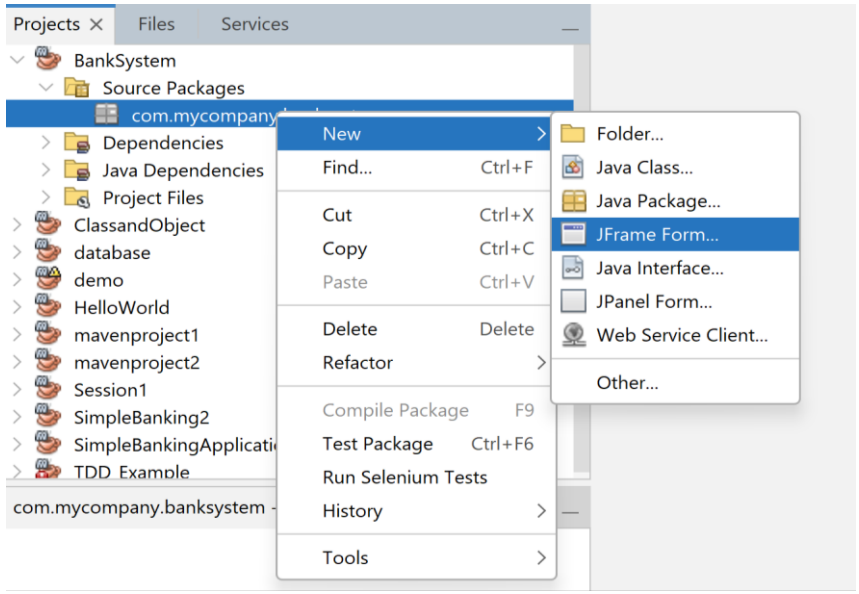
- But unlike console application, its purpose is to invoke the class constructor to load and display the GUI

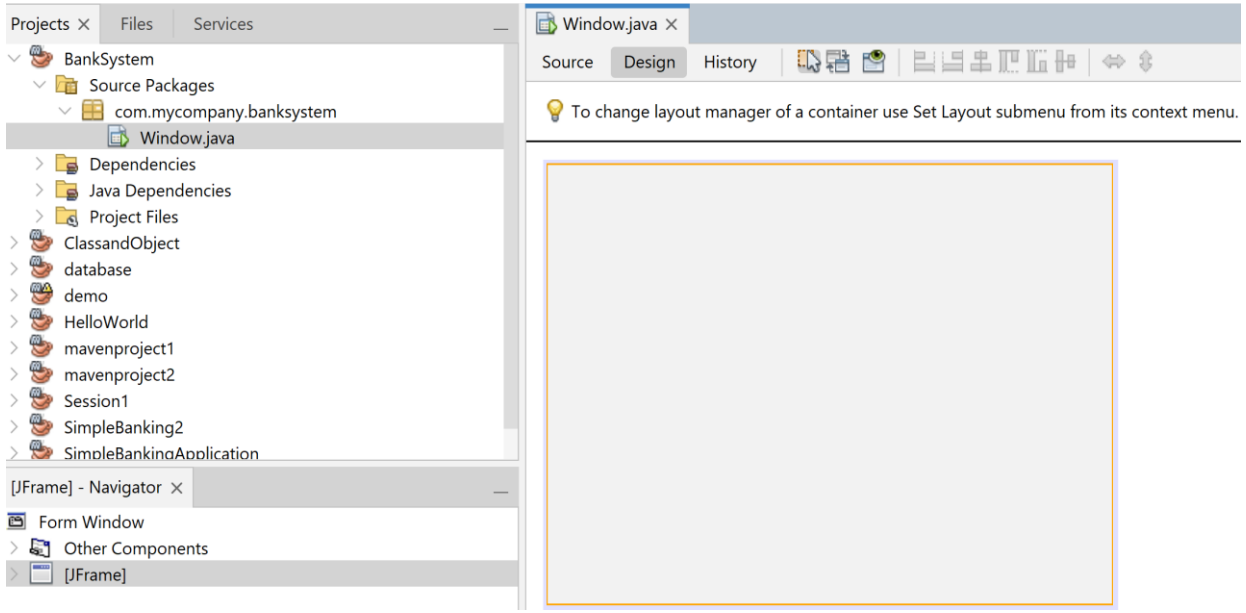
As constructor methods cannot be static

- Any calls from the constructor are to non-static methods, using non static variables
- This only static entity in the JFrame will be the main method

Creating a JFrame

- Create New Project as before
- Delete the Main class
- Add new JFrame Form to the named package
- Name the Class, e.g. Window, Main, BankSystem, etc
- An empty frame will be displayed in **Design** view
- Set this JFrame class as new main class via Project Properties→Run→Main Class

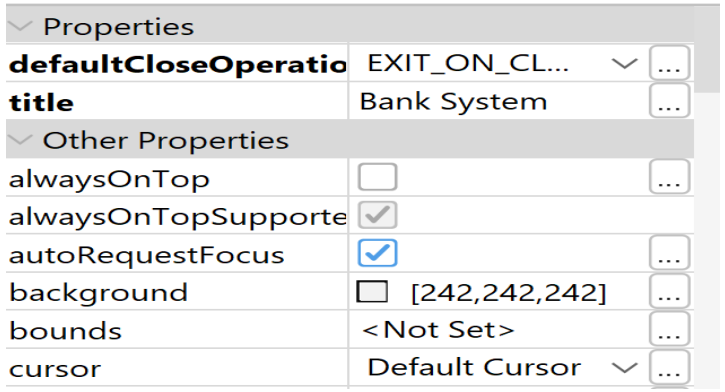




Design and Source Views

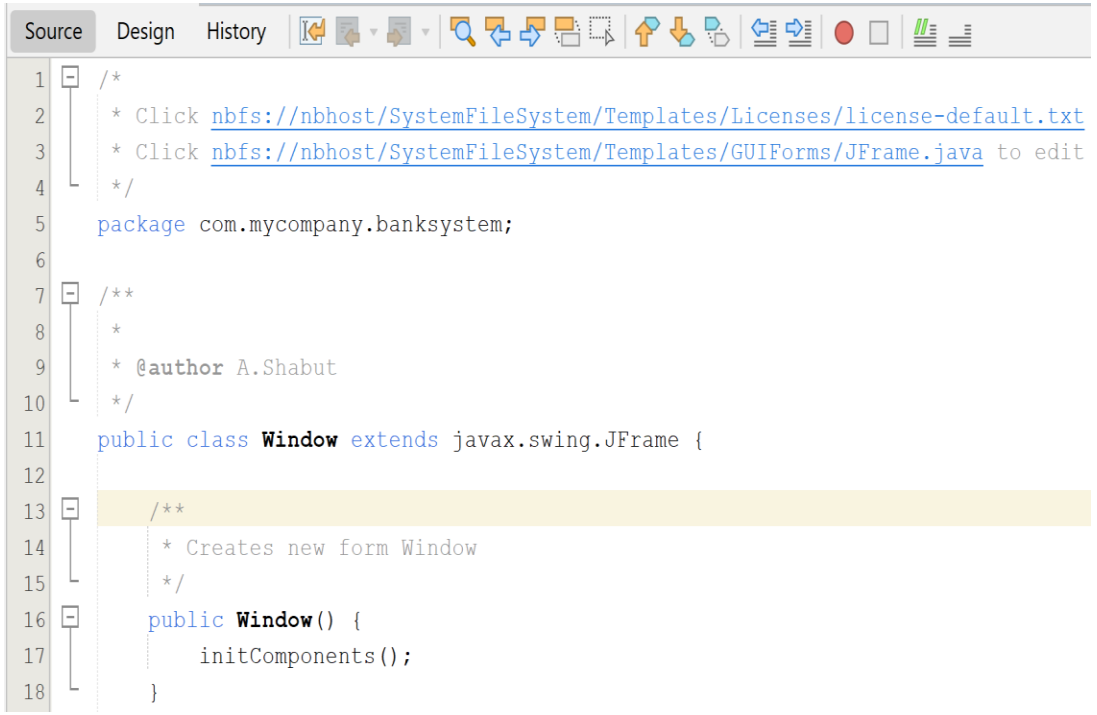
Design View

- Displays the pallete and properties panes
  - On the right-hand side of NetBeans
- The Pallete allows new components to be added to the JFrame form
- The Properties pane display attributes for currently selected component
  - For a JFrame we often set the **title** property



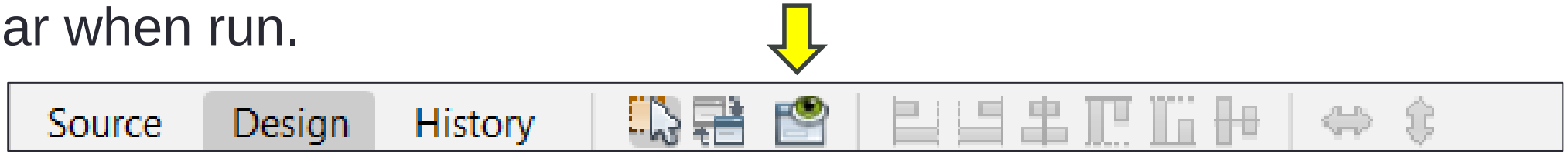
Source View

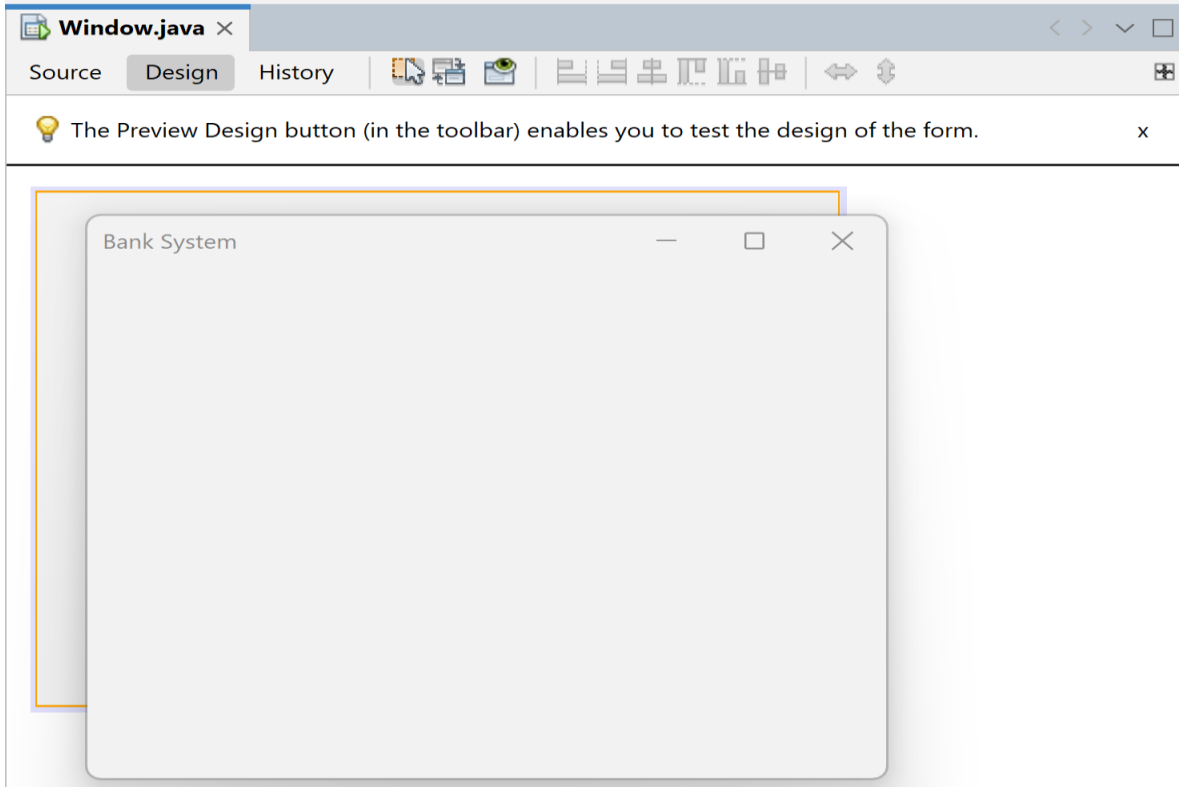
- Displayed by clicking the Source tab above the Form
- Allows code to be entered



Preview

- The eye icon above the form allows a preview of how the JFrame will appear when run.





Auto-generated code

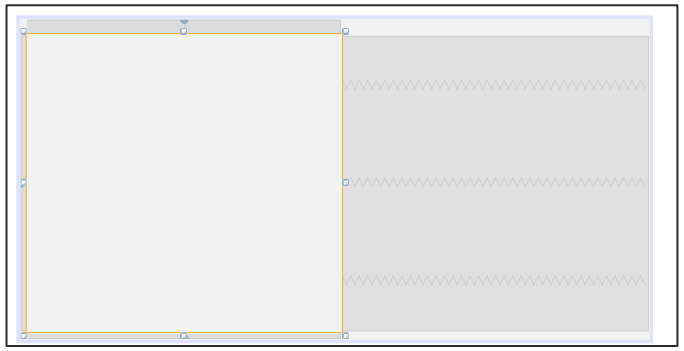
- The image to the right is an example of the code generated automatically
  - As we add and remove components using the design view, this code will be regenerated
  - It is important that you do not manually modify this code!
- The designer manages two files for each form
  - An XML document with the extension .form that describes the structure of the form
  - The Java code – part of which is generated from the XML document

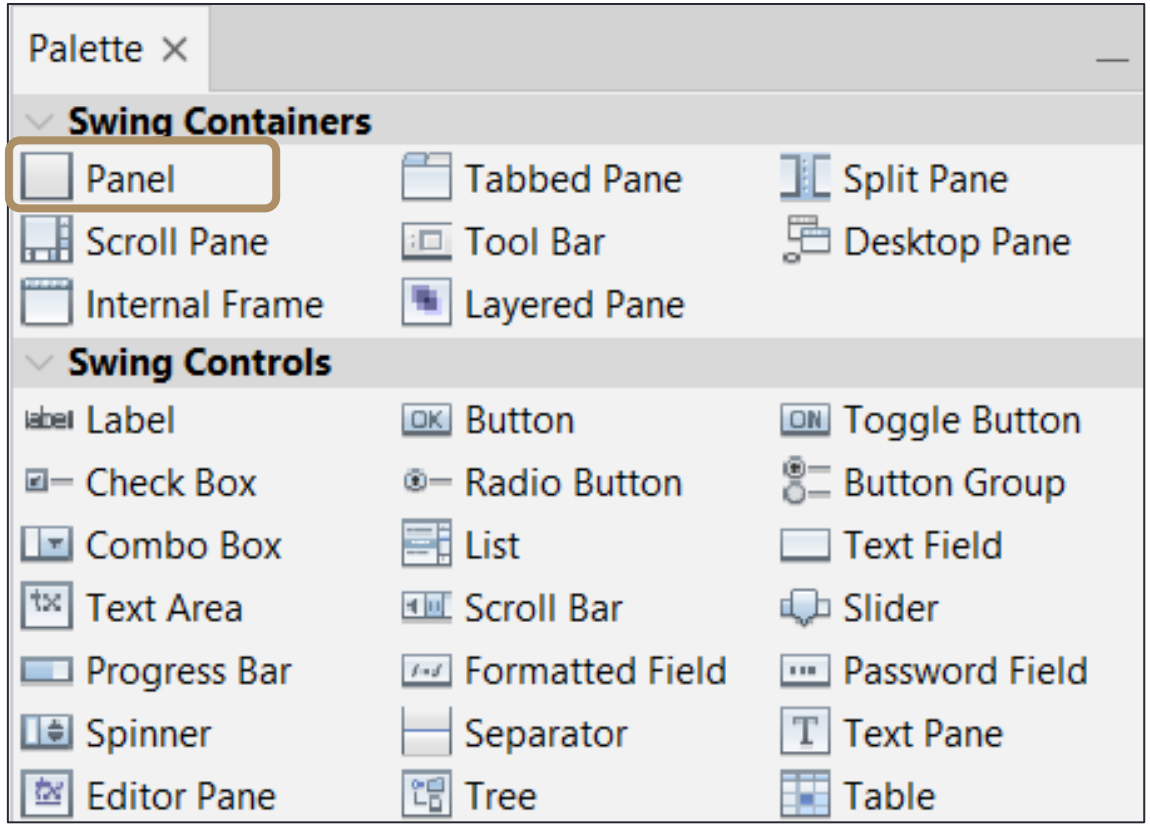
```
private void initComponents() {

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    setTitle("Bank System");

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 400, Short.MAX_VALUE)
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 300, Short.MAX_VALUE)
    );

    pack();
}// </editor-fold>
```

JPANEL

JPanel

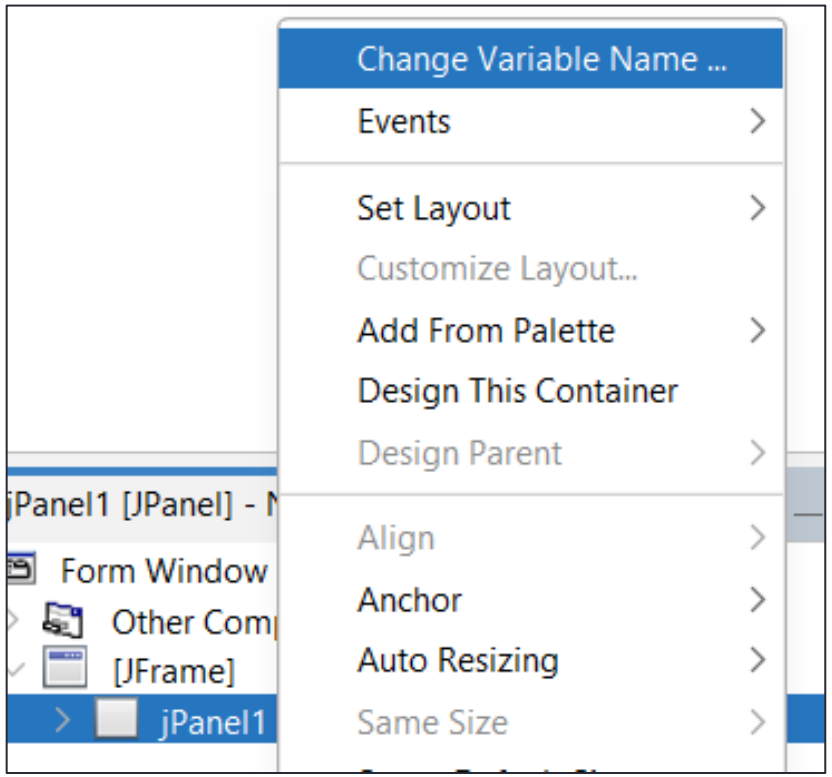
- A JPanel is a generic, rectangular containment component
  - Used to group and organise other components
- To add a JPanel use the Pallete Pane
  - Click on the panel icon
  - Then click on the form



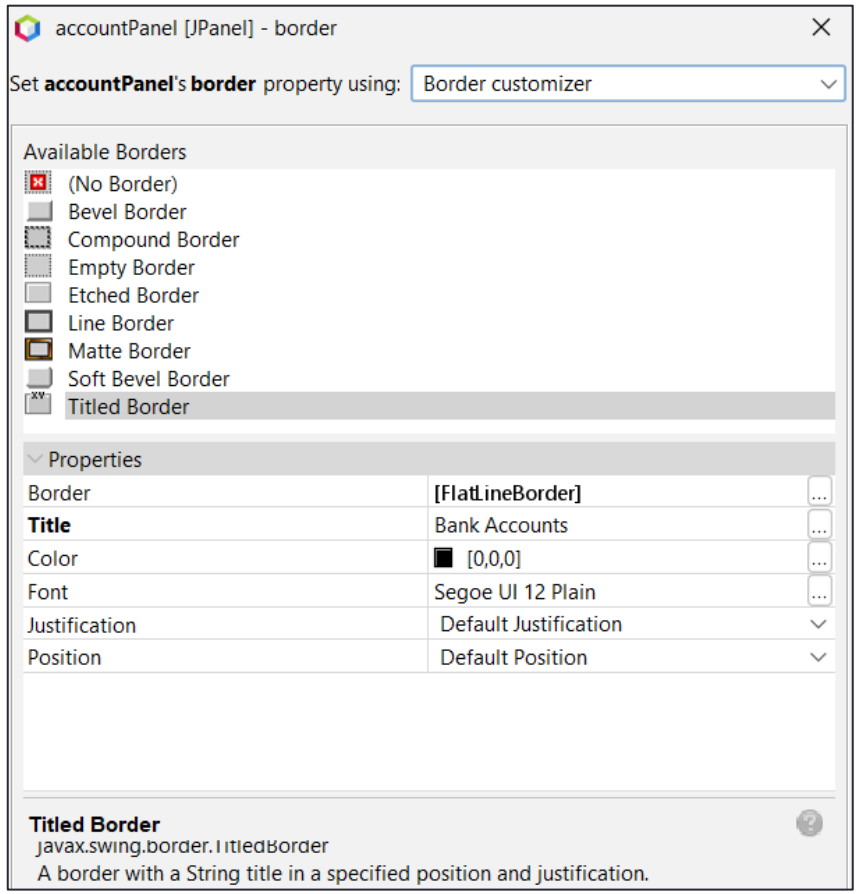


JPanel

- Unlike JFrame do **rename** a panel to reflect its purpose via the Pane
  - Right Click on Panel in Navigator pane
  - Select Change Variable Name
  - Enter new name, e.g., `accountPanel`



- Also, unlike `JFrame`, it does not have a title bar, but a titled border can be applied via the properties pane



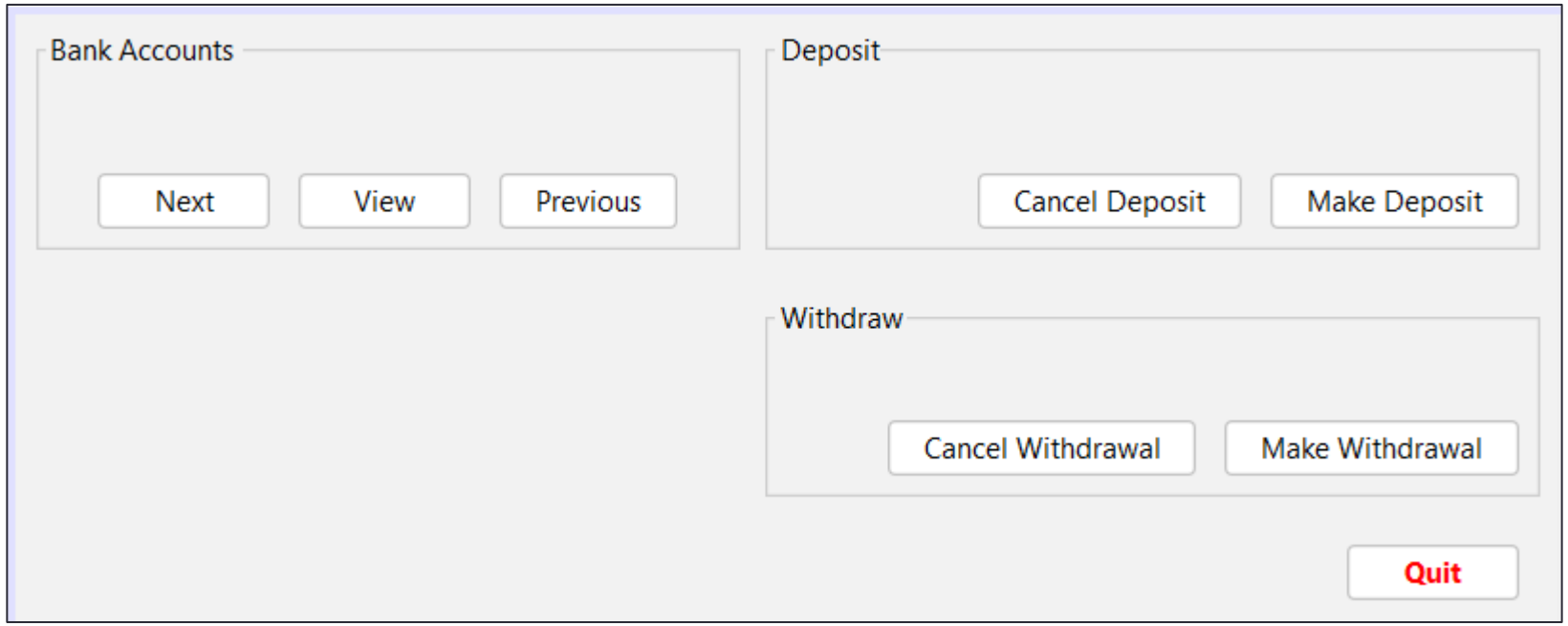
Three panels

- The form will have three panels
  1. `accountPanel` will allow user to cycle through bank accounts
  2. `depositPanel` will allow user to make deposit
  3. `withdrawPanel` will allow user to make a withdrawal

JBUTTON

JButton

- JButtons are used by the user to initiate some action, which in Java is known as an event
- Buttons are created via the Pallete
  - The text displayed by the Button and how the button looks are set via the Properties Pane
  - Change the name of the Button through the navigator pane, e.g. `nextButton`



Event Handling

- Event handling refers to the code we want to run when the user clicks a Button
  - Such code is placed into an Event Handler method which listens for a click event on the button
  - To create the event Handler method simply double click a button, e.g., `quitButton`, in Design View

```
25      @SuppressWarnings("unchecked")
26      Generated Code
172
        private void quitButtonActionPerformed(java.awt.event.ActionEvent evt) {
174         // TODO add your handling code here:
175     }
```

Event Handler

Event Handler Method

- Generally clicking a button is similar to a user making a choice from a command line menu
- I.e. we redirect program flow to the corresponding user defined method
- E.g. Event handler method for Quit Button will invoke the quit() method

Invoked method needs input

- Code to get and validate the input(s) can be coded directly in the event handler
- Before invoking relevant method

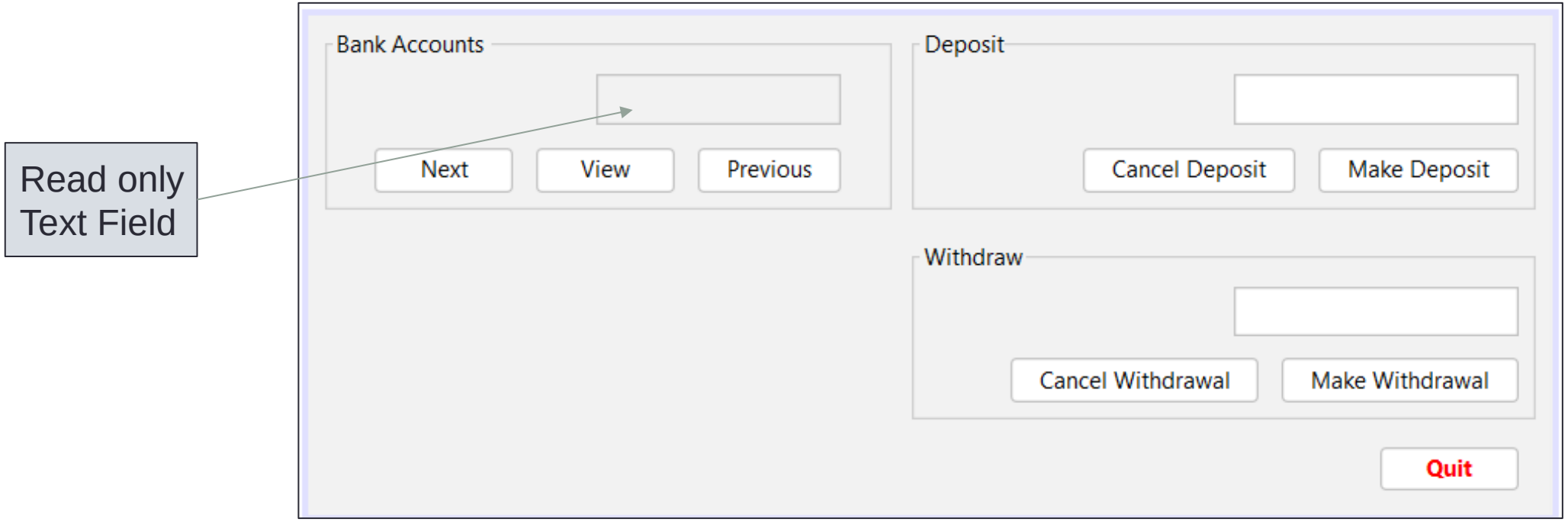
Simple form action

- If clicking a Button does a simple action on the form
- For example, clearing some component
- Then that can be directly coded in the event handler

JTEXTFIELD

JTextField

- JTextFields are created from the pallete
  - Name is set using Navigator Panel, e.g. `depositTextField`
  - Display attributes set using the Properties panel, e.g. font face/colour/size, text, editable, etc



Main attributes of a Text Field

For Output

- Text field can display a single line of data
- If you do not want user to edit text field, make it read only
  - By setting its **Editable** property to false, i.e. uncheck
- To set value in code use .setText

```
depositTextfield.setText("0");
```

- To clear text in Textfield, set to empty string

```
depositTextfield.setText=("");
```

For input

- Text field can accept single line of data
- All data read from a text field is of the string type and obtained using .getText()

```
String value =
depositTextfield.getText();
```

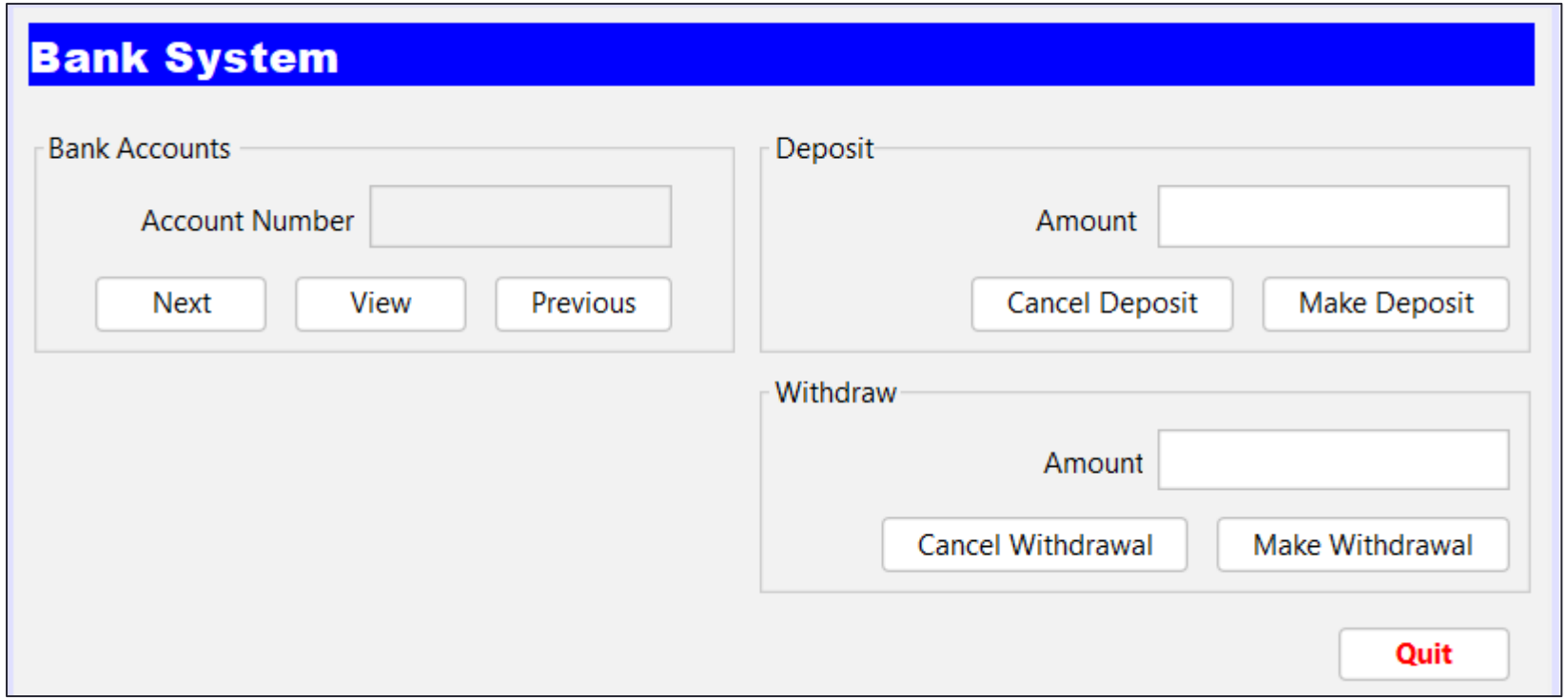
- And most likely need to be validated
  - Before processing the input
- Need to convert numeric values

```
int amount = Integer.parseInt(value);
double amount =Double.parseDouble(value);
```

JLABEL

JLabel

- JLabel can be used to display text or a picture
  - Created from palate pane
  - Appearance changed using Properties pane
- If the content of a label is to be changed by code, then rename it using Navigator pane
  - Otherwise, if content will not be changed, then no need to rename

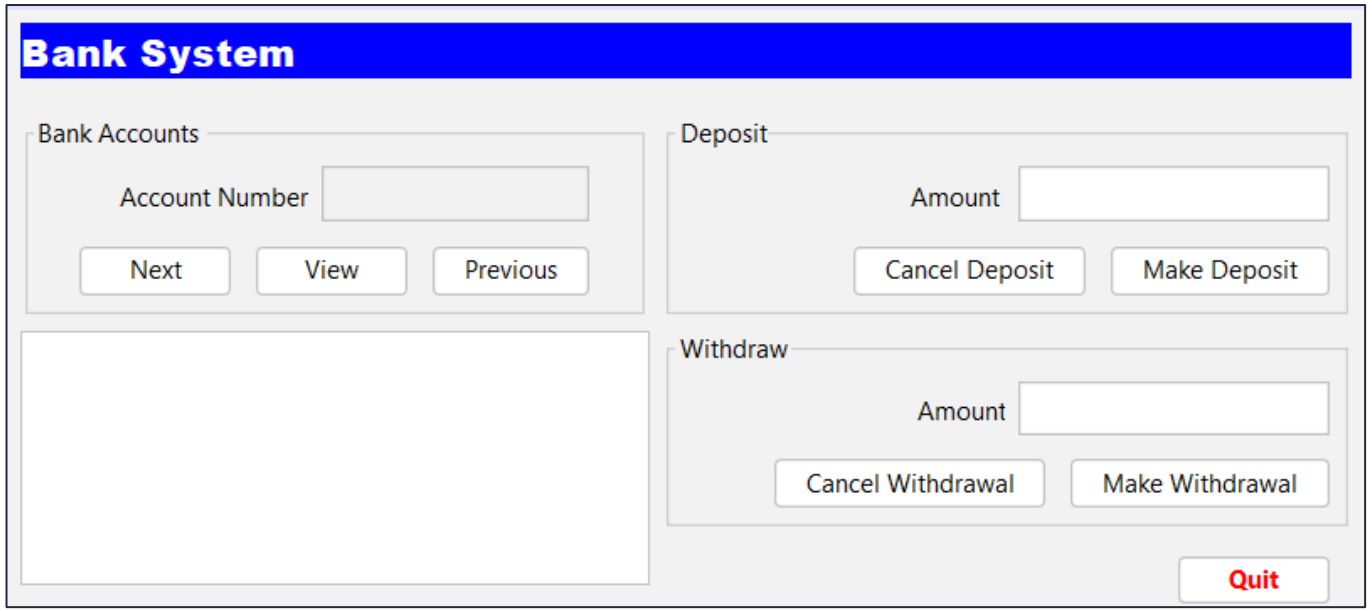


- If changing background colour of JLabel
  - Remember to check the Opaque property

JTEXTAREA

JTextArea

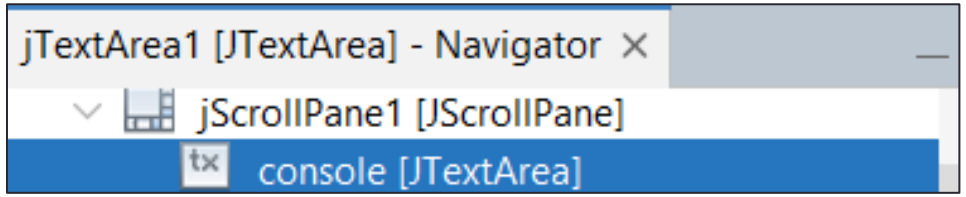
- JTextArea is mainly used to display multiline output
  - I.e. the GUI equivalent of NetBeans output window for Console applications
  - Accordingly, we normal name it `console`



More about JTextArea

Automatic Scrollbars

- When a JTextArea is created
  - NetBeans create a ScrollPane
  - And embeds the TextArea into it



Properties

- Make font face a fixed width font
- Make font size something easy to read
- Uncheck Editable to make read only

Displaying output

- To overwrite text area, use setText(), e.g. `console.setText("")` clears text area
- To append data to text area, use appendText(), e.g. `console.appendText("Name: ")`

LOADING DATA

Data Class

- Simplest way to add Data class to project
  - Create new Data Class file
  - Copy and paste code from another project where data class is used

```
2    * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-de
3    * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java
4    */
5   package com.mycompany.banksystem;
6
7   /**
8    *
9    * @author A.Shabut
10   */
11  public class BankAccount {
12      // Class level variables for account information
        private String accountName;
        private String accountNumber;
15      private double accountBalance;
16
17      // Constructor to initialize account details
18      public BankAccount(String name, String number, double balance) {
19          this.accountName = name;
20          this.accountNumber = number;
21          this.accountBalance = balance;
22      }
23
```

Class level Variable

- Due to the autogenerated code for GUI components
  - Class Level declarations go at the bottom of the main class

```
321         //class level object variables that can be used by different methods
            private ArrayList<BankAccount> accountList = new ArrayList<>();
            private final String DELIMITER = ",";
            private int index=0;
325
326     }//end of main class
327
```

- Note
  - No longer do we need to use the static keyword
  - No need for a scanner object to read command line input
  - Instead, we need an int index variable to keep track of which account in ArrayList has been selected by user

LoadData

- Write the loadData method, same as in Console Application
  - Remember no need for static keyword to be used
  - Best locate user defined methods above class level declarations

```
270     /**...3 lines */
273     public void loadData() {
274         // Create some initial accounts and add them to the list
275         accountList.add( new BankAccount("H Kane", "0123456", 0) );
276         accountList.add( new BankAccount("E Hayes", "1234567", 500) );
277         accountList.add( new BankAccount("B Mead", "2345678", 1000) );
278         accountList.add( new BankAccount("L Bronze", "3456789", 5000) );
279         accountList.add( new BankAccount("P Foden", "4567890", 200) );
280     }
281
```

Invoking loadData

- Invoke loadData() method in main class constructor method before the call to initComponents. Select supress warning if warned about overloaded call

```
9   /**
10   *
11   * @author A.Shabut
12   */
13  public class Window extends javax.swing.JFrame {
14
15      /**
16       * Creates new form Window
17       */
18      public Window() {
            loadData();
20          initComponents();
21      }
```

VIEW ACCOUNT

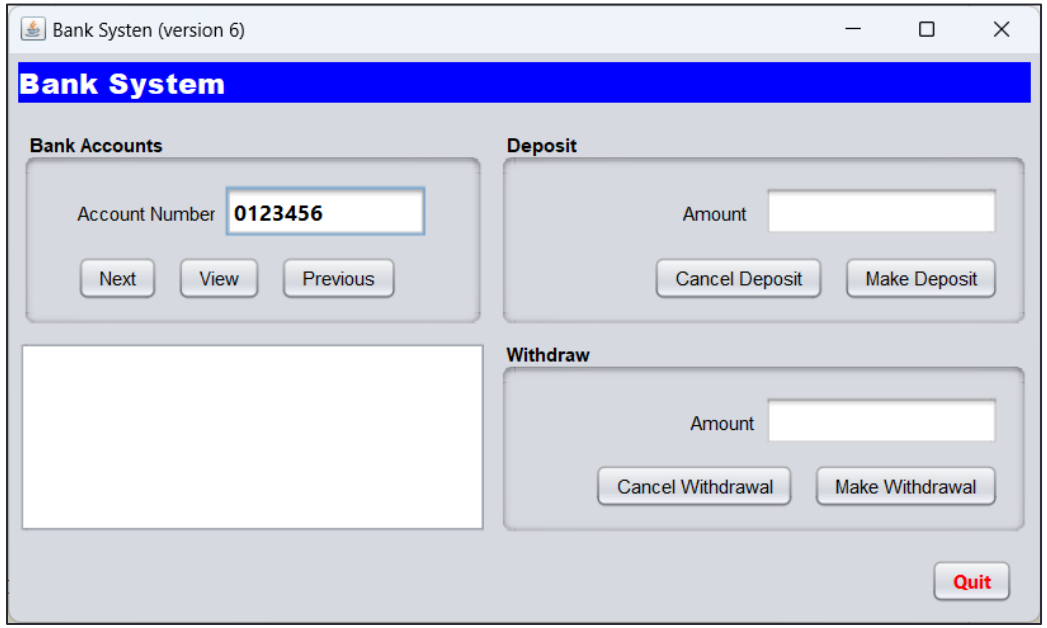
Process

- Will use accountPanel components and Text Area.
- numberTextField will display account number first item in ArrayList
- User clicks view Button to see account details in text area
- To get next account user clicks next button
- To get previous account user clicks previous button

Set first item

- In Constructor after initComponents() call enter code to
  - Initialise index to zero
  - Get account number of first item in arraylist
  - Display number in text field

```
18  public Window() {
        loadData();
20      initComponents();
21      // Set index to zero and display account number
22      index = 0;
        numberTextField.setText(accountList.get(index).getAccountNumber());
24  }
```



View Account Information

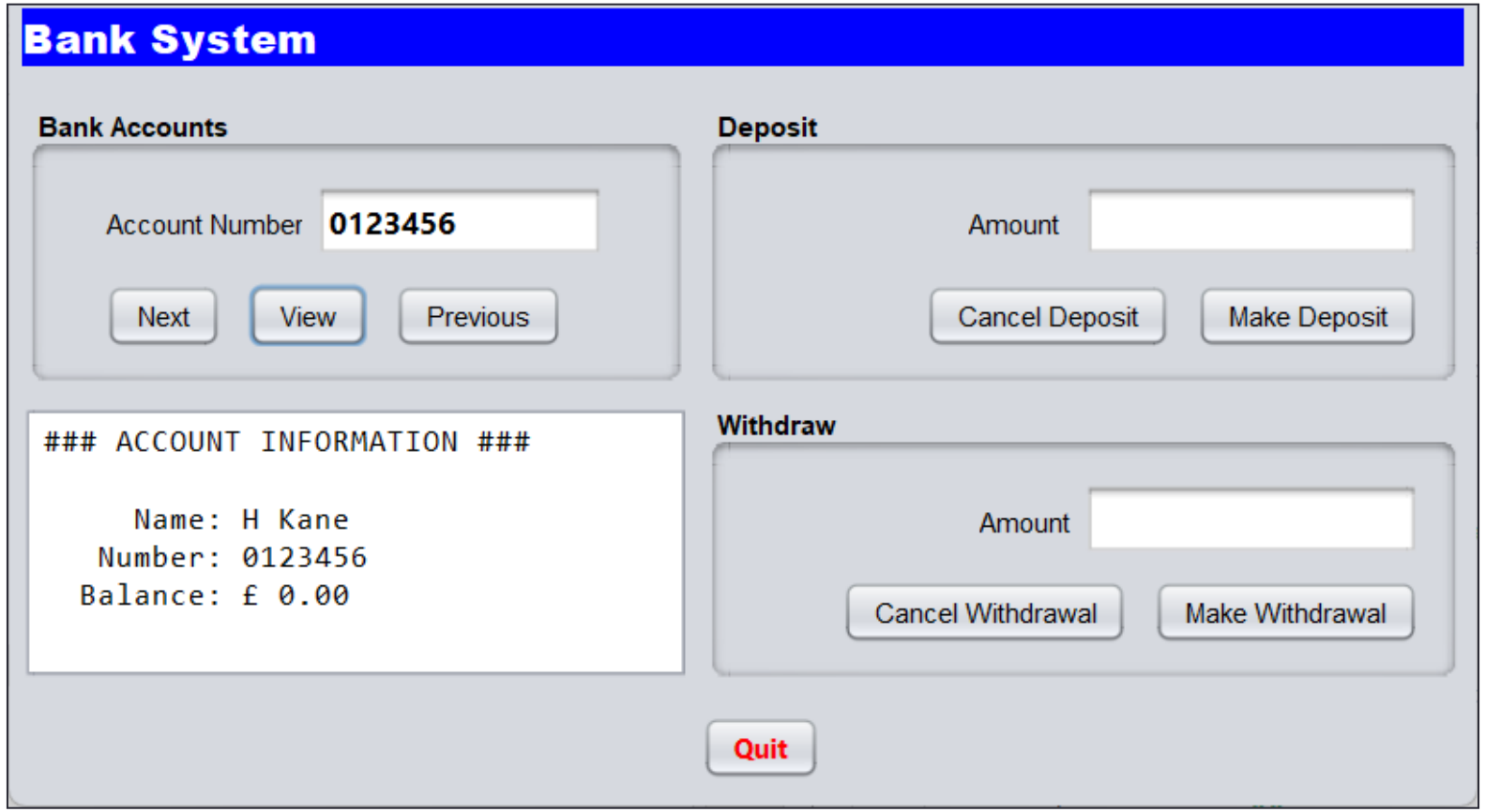
- Write a new viewInformation() method which displays details of current account in the text area

```java
    public void viewInformation() {
        //Report variable
        String report = "";
        // Construct output in report variable
        report+="### Account Information ###\n\n";
        report+= String.format("%10s %s %n", "Name:", accountList.get(index).getAccountName());
        report+= String.format("%10s %s %n", "Number:", accountList.get(index).getAccountNumber());
        report+= String.format("%10s %s %n", "Balance:", accountList.get(index).getAccountBalance());

        // display to text area
        console.setText(report);
    }
```

- The viewInformation() method is invoked by the event handler for the view button

```java
    private void viewButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        viewInformation();
    }
```

View Runtime



Next Button

- When user clicks the Next Button, the index will be incremented
  - This has the effect of making the next item in the ArrayList to be current object
  - For which account number will be displayed and text area cleared

```java
    private void nextButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        if (index < (accountList.size() - 1)) {
            index++;
        } else {
            index = 0;
        }
        numberTextField.setText(accountList.get(index).getAccountNumber());
        console.setText("");

    }
```

Previous Button

- When user clicks the Previous Button, the index will be decremented
  - This has the effect of making the previous item in the ArrayList to be current object
  - For which account number will be displayed and text area cleared

```java
    private void previousButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        if (index > 0) {
            index--;
        } else {
            index = accountList.size() - 1;
        }
        numberTextField.setText(accountList.get(index).getAccountNumber());
        console.setText("");

    }
```

MAKE DEPOSIT

Deposit Method

- With the event handler dealing with validation
- The deposit method just makes the calculation
- Update bank account balance and informs user

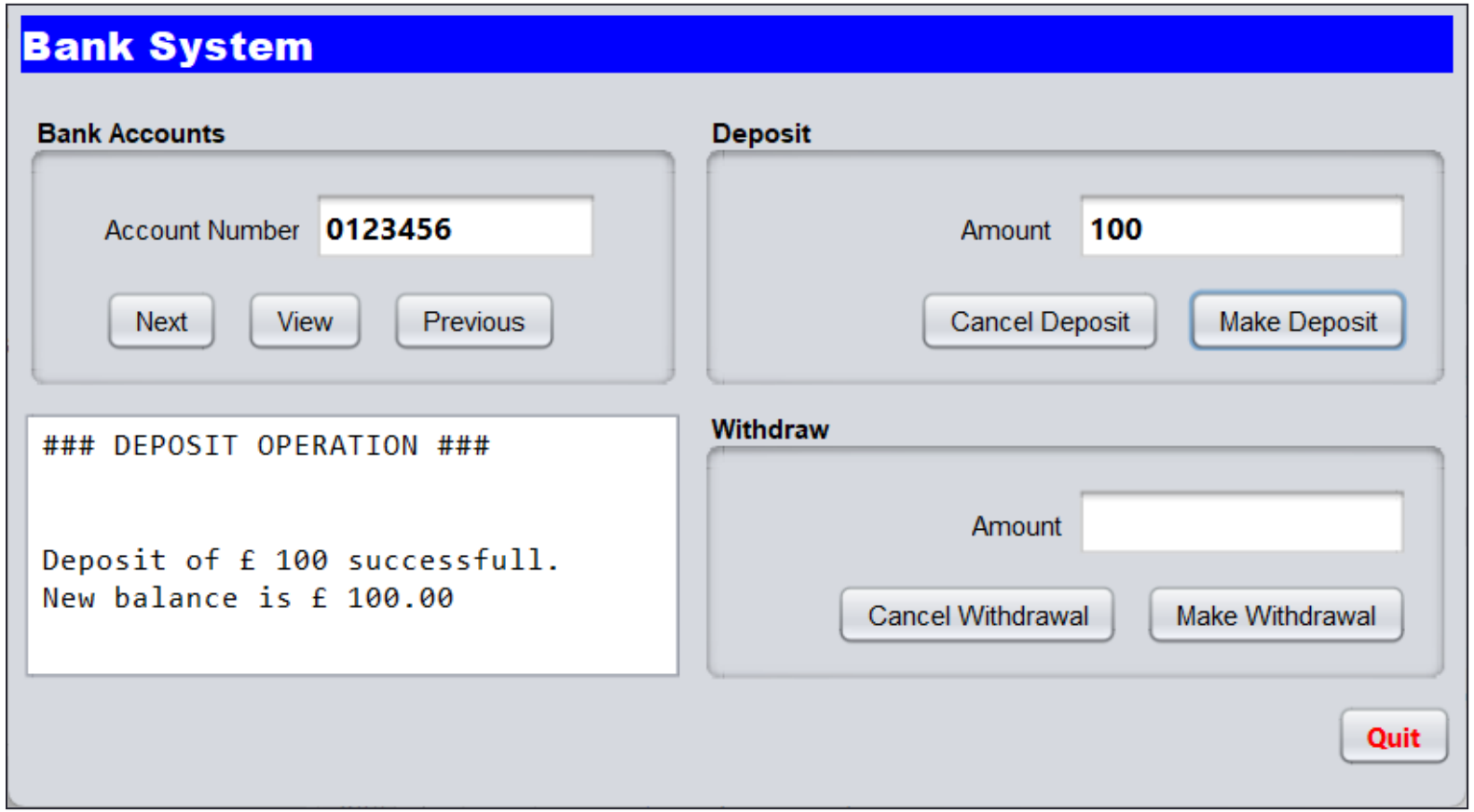
```
    private void deposit(int amount) {
        double balance = accountList.get(index).getAccountBalance() + amount;
        accountList.get(index).setAccountBalance(balance);

        console.append(String.format("%nDeposit of £ %d successful. %nNew balance is %s %n", amount,
                accountList.get(index).getFormattedBalance()));
    }
```

Deposit Button Event Handler

- Event Handler will get input from deposit text field

```
318     private void depositButtonActionPerformed(java.awt.event.ActionEvent evt) {
319         // TODO add your handling code here:
320         console.setText("### Deposit Operation ###\n");
321         int amount = Integer.parseInt(depositTextField.getText());
322         if (amount != -1) {
323             deposit(amount);
324         }
325     }
```

Deposit Runtime



Cancel Deposit

- Clears Text field
- invokes viewInformation method

```
private void cancelDepositButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    depositTextField.setText("");
    viewInformation();
}
```

MAKE WITHDRAWAL

QUIT

Quit Method

- Quit Method will inform user of save errors and program will terminate
  - On NetBeans output window, so messages are retained after GUI closes
- Invoke save method
- Terminate application using `System.exit(0)`

Event Handlers

- Two event handlers are required, for events:
  - Quit button clicked
  - Exit shortcut clicked of form title bar

- Quit Button event handler

```
333
    private void quitButtonActionPerformed(java.awt.event.ActionEvent evt) {
335     //invoke quit method
336     quit();
337 }
```

Tasks

- Code the Bank System example
- Implement the withdraw transaction
- Improve the GUI