Static Analysis:

BankHelp:

Issues Found and Recommendations

1. Missing Exception Handling Detail

Issue:

- Exceptions such as MalformedURLException and IOException are caught, but the handling is limited to printing stack traces to System.out. This approach is not user-friendly or robust.

Recommendation:

- Log the exceptions using a logging framework such as java.util.logging or SLF4J. Optionally, provide user feedback through the UI to indicate that an error occurred.

Example:

java

Copy code

```
catch (MalformedURLException e) {
  Logger.getLogger(HtmlPane.class.getName()).log(Level.SEVERE, "Invalid URL", e);
  JOptionPane.showMessageDialog(null, "Invalid file path: " + e.getMessage());
}
catch (IOException e) {
  Logger.getLogger(HtmlPane.class.getName()).log(Level.SEVERE, "Error loading file", e);
  JOptionPane.showMessageDialog(null, "Error loading file: " + e.getMessage());
}
```

---

2. Hardcoded String Concatenation for File URLs

Issue:

- The code manually concatenates "file:" with the file path. This can lead to issues if the path contains spaces or special characters.

Recommendation:

- Use the File.toURI().toURL() method, which safely converts file paths to URLs.

Example:

java

Copy code

```
URL url = f.toURI().toURL();
```

---

3. UI Responsiveness in the linkActivated Method

Issue:

- The linkActivated method directly manipulates the UI cursor and potentially blocks the Event Dispatch Thread (EDT) while loading the page.

Recommendation:

- Perform heavy tasks like loading a URL in a background thread using SwingWorker to avoid freezing the UI.

Example:

java

Copy code

```
SwingWorker<Void, Void> worker = new SwingWorker<>() {
  @Override
  protected Void doInBackground() throws Exception {
    html.setPage(u);
    return null;
  }

  @Override
  protected void done() {
    html.setCursor(cursor);
  }
};
worker.execute();
```

---

4. Inefficient Repainting

Issue:

- Explicit calls to parent.repaint() in PageLoader may lead to unnecessary repaints.

Recommendation:

- Avoid manual repainting unless absolutely necessary, as setPage() already handles rendering updates.

---

5. Lack of Null Check for the filename Parameter

Issue:

- The HtmlPane constructor assumes filename is valid, which may result in a NullPointerException.

Recommendation:

- Add a null check for filename and throw an IllegalArgumentException if it is null or invalid.

Example:

java

Copy code

```
if (filename == null || filename.isEmpty()) {
  throw new IllegalArgumentException("Filename must not be null or empty");
}
```

---

6. Inner Class Access to HtmlPane Attributes

Issue:

- The PageLoader inner class directly accesses outer class attributes (url, cursor), making it tightly coupled.

Recommendation:

- Refactor PageLoader to pass required attributes explicitly in the constructor, improving encapsulation.

---

7. Lack of Thread Safety

Issue:

- PageLoader manipulates the url variable, which may lead to race conditions if accessed concurrently.

Recommendation:

- Synchronize access to shared resources or avoid shared mutable state altogether.

---

8. No Validation of User-Provided URLs

Issue:

- No validation of URLs before loading them. Malicious or invalid URLs can cause runtime errors or expose the application to security risks.

Recommendation:

- Validate and sanitize URLs before passing them to setPage().

Example:

java

Copy code

```java
if (!url.getProtocol().equals("file")) {
  throw new IllegalArgumentException("Only file URLs are supported");
}
```

---

9. Missing JavaDoc Comments

Issue:

- The methods and classes lack JavaDoc comments, making it harder to understand the purpose and usage of the code.

Recommendation:

- Add descriptive comments for all public methods and classes.

---

10. Resource Management

Issue:

- The HtmlPane class opens resources (e.g., file streams or network connections) but does not explicitly close them.

Recommendation:

- Use try-with-resources for resource management.

Summary of Recommendations		
Category	**Issues Found**	**Priority**
Exception Handling	Inadequate handling of exceptions (e.g., printing to System.out)	High
URL Handling	Manual URL string concatenation is prone to errors	High
UI Responsiveness	EDT is blocked during page loading	High
Input Validation	No validation for filename or URLs	High

Code Readability	Lack of JavaDoc and meaningful comments	Medium
Resource Management	No explicit resource closing for HtmlPane	Medium
Thread Safety	Potential race conditions in shared variables	Medium
Encapsulation	Tight coupling of PageLoader with HtmlPane attributes	Low

Bank System:

Issues Found and Recommendations

1. Lack of Mocking and Dependency Injection

Issue:

- The BankSystem instance depends directly on JDesktopPane and other components, making it difficult to isolate the system under test (SUT) from its dependencies.

Recommendation:

- Use mocking frameworks like Mockito to mock the BankSystem and its dependencies, isolating the tests and ensuring independence from UI behavior.

Example:

java

BankSystem mockBankSystem = Mockito.mock(BankSystem.class);

---

2. Redundant Assertions

Issue:

- Tests like testGetAccountNo and testChangeLookAndFeel only contain assertions like assertTrue(true), which do not actually verify any behavior.

Recommendation:

- Replace placeholder assertions with verifications of specific behavior. If behavior cannot be tested directly, consider integration testing or GUI automation tools.

Example:

- For testGetAccountNo, verify if the correct account number is fetched and displayed.

---

3. Testing quitApp Behavior

Issue:

- The testQuitApp and testActionPerformed_quitApp tests assert that the application is no longer visible using bankSystem.isVisible(). However, this depends on the isVisible() implementation in BankSystem.

Recommendation:

- Instead of testing visibility directly, mock System.exit() or verify whether cleanup actions are performed before quitting.

Example:

java

Copy code

// Use a flag to verify quit behavior

assertTrue("quitApp should set appropriate flags or perform cleanup", bankSystem.hasQuit());

---

4. Hardcoded Mock Data in testFindRec

Issue:

- Hardcoding mock data in the records array for testFindRec couples the test logic to the data structure.

Recommendation:

- Use a test helper or mock data loader to abstract away mock data initialization.

Example:

java

Copy code

```
TestHelper.populateMockData(bankSystem.records, "12345", "John Doe", "1000");
```

---

5. Insufficient Coverage for UI Interaction

Issue:

- The tests simulate UI interactions but do not validate the internal state changes of the BankSystem.

Recommendation:

- Verify state changes in the BankSystem object after UI interactions.

Example:

```
assertEquals("Current balance should be updated", expectedBalance,
bankSystem.getBalance(accountNumber));
```

---

6. Lack of Assertions for GUI Changes

Issue:

- Tests for GUI-related methods like changeLookAndFeel do not verify that the Look and Feel actually changed.

Recommendation:

- Use assertions to validate GUI changes if feasible, or utilize GUI testing tools like FEST or AssertJ Swing for GUI validation.

---

7. Lack of Cleanup After Tests

Issue:

- The desktop pane and other components are not reset after tests, potentially causing interference between tests.

Recommendation:

- Use an @After method to reset shared state after each test.

Example:

```
public void tearDown() {

  desktopPane.removeAll();

}
```

---

8. Test for populateArray Depends on External File

Issue:

- The testPopulateArray depends on the existence and contents of the Bank.dat file, making the test non-deterministic.

Recommendation:

- Mock file operations to provide predictable test data without relying on external files.

Example:

java

Copy code

```
File mockFile = new File("mockBank.dat");

bankSystem.populateArray(mockFile);
```

---

9. Lack of Edge Case Testing

Issue:

- Tests do not handle edge cases, such as empty or null account numbers in testFindRec, or invalid indices in testChangeLookAndFeel.

Recommendation:

- Add tests for invalid inputs and edge cases.

Example:

```
@Test(expected = IllegalArgumentException.class)

public void testFindRec_withNullAccountNumber() {

  bankSystem.findRec(null);

}
```

---

10. Verbose Repeated Logic for Verifying Open Frames

Issue:

- Each test manually iterates through frames to check for specific window titles.

Recommendation:

- Extract this repeated logic into a reusable helper method.

Example:

```
private boolean isFrameOpened(String title) {

  return Arrays.stream(desktopPane.getAllFrames())

          .anyMatch(frame -> frame.getTitle().equalsIgnoreCase(title));

}
```

Summary of Recommendations

Category	**Issues Found**	**Priority**
Dependency Isolation	Lack of mocking and direct dependency on UI components	High
Assertion Quality	Redundant or insufficient assertions	High

GUI Testing	Lack of validation for GUI changes	High
Test Data Isolation	Hardcoded mock data and external file dependencies	High
Cleanup and Teardown	Shared state not reset after tests	Medium
Code Reusability	Repeated frame-checking logic	Medium
Edge Case Coverage	Tests do not handle invalid inputs or edge cases	Medium

Delete Customer:

Issues Identified and Recommendations

---

1. Error Handling

- **Issue**:
    - Generic catch (Exception ex) blocks are used in several places. This practice can obscure the specific issue being encountered.
- **Recommendation**:
    - Catch specific exceptions (e.g., IOException, ArrayIndexOutOfBoundsException) to provide better diagnostic information and avoid masking unrelated issues.
    - Add proper logging or user feedback to handle exceptions gracefully.

---

2. File Handling

- **Issue**:
  - File streams (FileInputStream, DataInputStream) are not reliably closed in case of exceptions.
  - Use of readUTF() assumes proper encoding but does not handle unexpected file content.
- **Recommendation**:
  - Use a try-with-resources block to ensure resources are closed automatically.
  - Validate the file content before attempting to read to prevent runtime crashes.

---

3. Array Handling

- **Issue**:
  - Fixed-size array (records[500][6]) is used to hold records. This approach lacks scalability and error handling for overflows.
- **Recommendation**:
  - Use a List<String[]> instead of a fixed array. This allows dynamic growth and better memory management.
  - Perform bounds checking before accessing or updating array elements.

---

4. Thread Safety

- **Issue**:
  - The records array and total variable are accessed and modified without synchronization. This can lead to data inconsistency if accessed by multiple threads.
- **Recommendation**:
  - Use proper synchronization mechanisms (e.g., synchronized blocks) if the class is expected to be accessed in a multithreaded environment.

---

5. UI Layout

- **Issue**:
  - null layout is used for the panel, leading to hardcoded positions. This is not scalable or adaptable to different screen resolutions.
- **Recommendation**:
  - Use a LayoutManager (e.g., GridBagLayout, GroupLayout) for a responsive and maintainable UI.

---

6. Input Validation

- **Issue**:
  - Only a numeric validation is applied to the account number field, but other fields lack robust validation.
- **Recommendation**:
  - Add validations for account name, balance, and other fields to ensure data consistency.

---

7. Magic Numbers and Constants

- **Issue**:
  - Hardcoded values (e.g., 500 for array size, 6 for record fields, component positions) reduce readability and maintainability.
- **Recommendation**:
  - Define these values as constants (e.g., private static final int MAX_RECORDS = 500;).

---

8. Code Duplication

- **Issue**:
  - Repeated logic for file handling and array processing can lead to maintenance issues.
- **Recommendation**:

  - Refactor common logic into utility methods (e.g., a method for reading/writing files).

---

9. Event Listeners

- **Issue**:
  - The KeyListener implementation allows only numeric input but does not account for edge cases like pasting invalid text.
- **Recommendation**:
  - Use input verifiers or DocumentFilter for robust input validation.

---

10. User Feedback

- **Issue**:
  - Error messages are generic, and no logging is performed.
- **Recommendation**:
  - Improve user feedback by providing specific error messages and add logging for debugging purposes.

---

11. Java Swing Best Practices

- **Issue**:
  - JOptionPane is used excessively for feedback, which can interrupt the user experience.
- **Recommendation**:
  - Consider using a dedicated status bar or non-blocking notifications for user feedback.

Proposed Refactoring Priorities

1. **Use List<String[]> and remove fixed array implementation.**
2. **Implement try-with-resources for file handling.**
3. **Refactor UI layout using LayoutManager.**

4. **Improve input validation with DocumentFilter.**
5. **Add proper exception handling and logging mechanisms.**

Deposit Money :

Issues and Recommendations:		
Issue	**Description**	**Recommendation**
1. Hardcoded File Name	The file name Bank.dat is hardcoded multiple times.	Use a constant or configuration file for the file name to avoid hardcoding.
2. No Validation for Deposit Amount	There's no check to ensure the deposit amount is positive or a valid number.	Add validation to check if the deposit amount is a positive number before processing.

3. Potential Array Out of Bounds	The populateArray() method loads records into a 2D array, which may overflow if more than 500 records are added.	Use dynamic data structures such as ArrayList instead of a fixed-size array.
4. curr Variable Usage	The curr variable is assigned but its purpose isn't clear in terms of handling the balance. It directly uses curr + deposit without validation.	Implement proper handling and checks for balance calculations to avoid inconsistencies or errors in deposit operations.
5. File I/O Exceptions Not Properly Handled	The catch block in populateArray() and editFile() does not properly handle potential file I/O errors.	Include specific handling for IOException with proper error messages and logging.

6. Excessive FocusListener and KeyListener Logic	There are redundant listeners for validating numeric inputs (account number and deposit amount), which can be simplified.	Consider using a common method for validating numeric input fields or custom input verifiers to reduce code duplication.
7. Data Integrity Issues	When updating records, there is no mechanism to ensure data integrity (e.g., race conditions in concurrent scenarios).	Add synchronization mechanisms or use database transactions to ensure data integrity if multiple users or threads may access the records.

8. Inefficient Data Storage	The records are stored in memory, and the entire array is rewritten to the file after each edit, which can be inefficient for large datasets.	Consider using a database or a more efficient storage mechanism to handle large datasets more effectively.
9. Lack of Error Handling for User Actions	User actions such as canceling or saving might not handle errors well (e.g., failing to read/write to a file).	Implement better error handling for user actions, including file access errors, to improve user experience.
10. Unused editRec Method	The method editRec() is being called to save user changes, but it's not clear if the changes are successful after the update.	Ensure that after editing, the user interface reflects the updated data, and provide feedback regarding success/failure.

11. btnEnable Method	This method disables certain fields, but it could be better structured to handle all UI states based on actions.	Create distinct enable/disable methods for different states of the application (e.g., for editing, saving, etc.).
12. Missing Data Validation for User Input	Inputs such as account number and deposit amount should be validated (e.g., ensure no empty values, account exists).	Add validation checks on the user input before processing it to ensure no empty or invalid data is submitted.

Find Account :

Issue	**Description**	**Recommendation**

1. Hardcoded File Name	The file name "Bank.dat" is hardcoded multiple times in the code.	Use a constant or configuration file for the file name to avoid hardcoding and make the code more flexible.
2. No Validation for Account Number	There's no validation to ensure the account number exists in the records before searching.	Add validation to check if the account number exists and provide meaningful error messages if not found.
3. Potential Array Out of Bounds	The records[][] array is statically sized to 500, which may lead to issues if more records are added.	Consider using dynamic data structures like ArrayList<String[]> for better scalability and flexibility.

4. File Handling Without Closing Resources	The DataInputStream and FileInputStream are not always properly closed in populateArray() in case of an exception.	Ensure proper closing of file streams in a finally block to prevent resource leaks.
5. Lack of Error Handling in populateArray()	The catch block in populateArray() does not properly handle all potential exceptions, such as IOException.	Add more detailed error handling for file I/O exceptions and ensure proper feedback is given to the user.
6. Inconsistent Text Field Behavior	The text field txtNo is restricted to numeric input, but txtBal and other fields are not.	Implement consistent validation across all text fields where applicable (e.g., for numeric fields like balance).

7. Data Integrity Issues	The array records[][] is stored in memory, which can be prone to data loss if the program crashes or the system shuts down unexpectedly.	Implement persistent storage mechanisms (e.g., using a database or writing to a file) with proper backup and recovery strategies.
8. Inefficient Data Search	Searching for records involves iterating through the entire array each time a search is performed, which could be inefficient for large datasets.	Use a more efficient data structure, such as a HashMap<String, String[]>, to allow faster lookup times by account number.

9. Lack of Proper User Feedback	When no record is found for the account number, the UI provides no clear feedback about the absence of the account.	Provide more user-friendly messages and feedback, such as a pop-up message that clearly states the account was not found.
10. Unnecessary btnEnable() Method	The method btnEnable() disables the account number field and the search button, which might not be necessary for all cases.	Review the purpose of the btnEnable() method, and consider reworking it to handle all possible user actions and states.

11. Unused Exception Handling	In the catch block inside populateArray(), the exception is silently handled without providing feedback or logging.	Add logging to track errors and inform the user about what went wrong.
12. Missing Data Validation for User Input	User input (e.g., account number) is not properly validated before searching.	Add validation checks to ensure that the account number field is not empty and that it contains only valid numeric values.

Find Account name :

Issue	**Description**	**Recommendation**

1. Hardcoded File Name	The file name "Bank.dat" is hardcoded multiple times.	Use a constant or configuration file to define the file name, making the code more flexible and maintainable.
2. No Validation for Name Input	There is no validation to ensure the name input is not empty before initiating a search.	Add validation to check if the name field is not empty before performing the search operation.
3. Potential Array Out of Bounds	The records[][] array is statically sized to 500, which may lead to issues if more records are added.	Consider using dynamic data structures like ArrayList<String[]> to handle an unknown number of records more efficiently.

4. Inefficient Data Search	The method findRec() searches the entire array for the matching name, which could be inefficient for large datasets.	Use a more efficient data structure, such as a HashMap<String, String[]>, to allow faster lookups by customer name.
5. Lack of Proper Resource Management	The DataInputStream and FileInputStream are not always properly closed in case of an exception in populateArray().	Ensure proper closing of file streams in a finally block to prevent resource leaks.

6. Inconsistent Text Field Behavior	The text field txtNo is disabled, which prevents the user from seeing the account number associated with the name they search for.	Consider making the txtNo field visible but uneditable, so the user can view the account number without modifying it.
7. Missing Error Handling for User Actions	The user actions like finding a record or clearing text do not have proper error handling for file access or internal logic errors.	Add error handling for file access failures and other possible issues that could arise while interacting with the user interface.
8. Data Integrity Issues	The records[][] array is stored in memory, and there's no backup mechanism in case of system failure.	Implement persistent storage (e.g., database) or periodic saving of in-memory data to avoid data loss.

9. Lack of Confirmation for Successful Search	When a user successfully finds a record, there’s no confirmation message or feedback indicating success.	Provide a success message or confirmation dialog after successfully finding a record.
10. Unnecessary btnEnable() Method	The btnEnable() method disables the name field and the search button, but this might not be necessary.	Review and refactor the btnEnable() method to handle UI states more intuitively and only disable elements when necessary.

11. No Handling for Duplicate Records	There is no check for duplicate records when searching for an account by name. If multiple customers have the same name, only the first match is found.	Implement checks to handle cases where multiple customers share the same name, and provide a way to list or distinguish between them.
12. Lack of Proper Data Feedback	The search function does not provide feedback about the number of records found or any other relevant data about the search results.	Improve feedback after searching, such as showing how many records match the search criteria or indicating no matches were found.

FindName :

Issue	**Description**	**Recommendation**

1. Hardcoded File Name	The file name "Bank.dat" is hardcoded multiple times in the code.	Use a constant or configuration file to define the file name, making the code more flexible and maintainable.
2. No Validation for Name Input	There is no validation to ensure the name input is not empty before initiating a search.	Add validation to check if the name field is not empty before performing the search operation.
3. Potential Array Out of Bounds	The records[][] array is statically sized to 500, which may cause issues if more records are added.	Consider using dynamic data structures like ArrayList<String[]> to handle an unknown number of records efficiently.

4. Inefficient Data Search	The method findRec() searches the entire array for the matching name, which can be slow for large datasets.	Use a more efficient data structure, such as a HashMap<String, String[]>, to allow faster lookups by customer name.
5. Lack of Proper Resource Management	The DataInputStream and FileInputStream are not always properly closed in case of an exception in populateArray().	Ensure proper closing of file streams in a finally block to prevent resource leaks.
6. Inconsistent Text Field Behavior	The text field txtNo is disabled, preventing the user from seeing the account number.	Consider making the txtNo field visible but uneditable so the user can view the account number without modifying it.

7. Missing Error Handling for User Actions	The user actions like finding a record or clearing text do not have proper error handling for file access or internal logic errors.	Add error handling for file access failures and other possible issues that may arise during user interaction.
8. Data Integrity Issues	The records[][] array is stored in memory and may be lost if the application crashes.	Consider using persistent storage (e.g., a database) or periodic saving of in-memory data to avoid data loss.
9. Lack of Feedback for Successful Search	When a user successfully finds a record, there’s no feedback or message to confirm the search was successful.	Provide feedback after finding a record, such as a confirmation dialog or a success message.

10. Unnecessary btnEnable() Method	The btnEnable() method disables the name field and the search button, but it might not be needed.	Review and refactor the btnEnable() method to handle UI state more intuitively and disable elements only when necessary.
11. No Handling for Duplicate Records	There is no check for duplicate records when searching by name. If multiple customers share the same name, only the first match is found.	Implement handling for multiple records with the same name, such as displaying a list of matches or allowing the user to select from multiple results.

12. Lack of Detailed Feedback	The search function doesn’t provide details about the search results or whether multiple matches were found.	Improve feedback after a search by displaying how many records matched or showing a list of matching records.

NewAccount :

Issue	**Description**	**Recommendation**
1. Hardcoded File Name	The file name "Bank.dat" is hardcoded in multiple locations.	Use a constant or configuration file for the file name, allowing for more flexible management of the file path.

2. Array Size Limitation	The records[][] and saves[][] arrays are statically sized to 500, which could limit the number of accounts stored.	Consider using dynamic data structures like ArrayList<String[]> to handle a potentially unlimited number of records.
3. Lack of Input Validation for Deposit	While numeric validation is performed on the deposit field, there is no check for negative values or zero.	Add validation to ensure that the deposit amount is a positive value greater than zero.

4. Inconsistent Data Handling	The code loads records from the file and stores them in memory (records array) but doesn't update the file when new records are added.	After adding a new record, the records array should be saved back to the file, or the array should be handled dynamically to avoid overwriting previous records.
5. Inefficient Data Search	The method findRec() searches the entire records[] array linearly to check if an account number exists. This approach may become slow with large datasets.	Consider using a HashSet<String> or HashMap<String, String[]> to quickly check for duplicates by account number.

6. Lack of Feedback for Duplicate Account	When an account number already exists, the code just clears the form. There’s no way for the user to see which account number was duplicated.	Provide a clearer error message when an account number already exists, including details on the duplicated account.
7. Poor Resource Management	The FileInputStream and DataInputStream are opened but not always closed in case of an exception, which could lead to resource leaks.	Ensure proper closing of file streams in a finally block to prevent resource leaks, especially in populateArray() and saveFile().

8. Lack of Date Validation	The selected date is stored but isn't validated to check if the month/day/year combination is correct.	Add validation to ensure that the selected month, day, and year combination forms a valid date.
9. Error Handling in saveFile()	The saveFile() method catches IOException, but the message displayed is generic. It would be better to show more specific details about the error.	Improve error handling by displaying more detailed error messages, such as the specific IO exception or file access issue.

10. No Confirmation of Saving Record	After saving the account, the user is shown a message, but there’s no confirmation or visual cue to show the account was successfully saved, and the UI could be clearer.	Improve the user interface by providing visual feedback (e.g., highlighting the new account fields) and confirming successful saving.
11. Hardcoded Date Ranges	The years in the date combobox are hardcoded from 2000 to 2015.	Make the year range dynamic, based on the current year, so that users can select future years.

12. Lack of Internationalization	The labels and buttons are hardcoded in English, and if you need to translate the application in the future, this could cause issues.	Consider using Java's ResourceBundle for internationalization, allowing you to easily translate the application.
13. Inefficient Handling of Multiple Accounts	records[][] is loaded into memory every time an account is saved. This could lead to inefficiencies when dealing with large files.	Instead of loading all records into memory on every save, only load the data when needed and append the new record directly to the file.

14. Redundant Code in saveArray()	The saveArray() method copies the data to the saves[][] array before writing it to the file. This can be simplified.	Directly save the new record to the file without the intermediary step of storing it in the saves[][] array, reducing complexity.

ViewCustomer:

Issue	**Description**	**Recommendation**
1. Hardcoded File Name	The file name "Bank.dat" is hardcoded in multiple locations.	Use a constant or configuration file for the file name, allowing for more flexible management of the file path.

2. Static Array Size	The rows[][] and rowData[][] arrays are statically sized to 500 and 4, respectively. This can lead to inefficiencies and wasted memory.	Consider using dynamic data structures like ArrayList<String[]> for better flexibility when handling the data.
3. No Dynamic Table Updates	The table is populated once when the window is opened and never updated after that. If records change (e.g., new records added), the table will not reflect these changes.	Consider adding a method to refresh the table after records are updated or allow for real-time updates when records are added/removed.

4. Inefficient File Reading	The file reading process in populateArray() assumes the file has a fixed format and loops indefinitely until an exception is thrown. This can lead to performance issues or incorrect data handling.	Use a more robust way of reading the file, like reading until the end of the file, or using BufferedReader to process lines more efficiently.
5. Lack of Error Handling for File I/O	The exception handling in populateArray() is broad and doesn't give any detailed information about what went wrong with the file reading process.	Improve error handling by catching specific exceptions (e.g., FileNotFoundException, EOFException) and displaying more detailed error messages.

6. Fixed Column Size	The column sizes for the table are hardcoded. This can cause issues if the content length changes or if the window is resized.	Consider using TableColumnModel with auto-resizing capabilities or set the column width dynamically based on the content.
7. Memory Inefficiency with File Handling	The entire file is read into memory at once and stored in an array, even if it's not all needed at once.	Implement pagination or load records in chunks to avoid reading all data into memory at once.
8. Poor Resource Management	The DataInputStream and FileInputStream streams are not always closed properly, especially when an exception occurs.	Ensure proper closing of file streams in a finally block or use a try-with-resources statement to handle resources automatically.

9. No Feedback for Empty Records	If the records file is empty, the user is only shown a message without providing further options or actions to take.	Provide a more user-friendly feedback option, such as a button or action to add records or a clearer message prompting the user to add records.
10. Limited Date Display	The date is shown in a concatenated format (Month, Day, Year), but this might not be user-friendly or sufficient for other uses.	Consider displaying the date in a more formal format, such as MM/dd/yyyy, or allowing users to choose the format.

11. Inconsistent Exception Handling	While exceptions are caught when closing streams, the overall structure of exception handling is inconsistent.	Consolidate exception handling to ensure proper cleanup and informative messages for the user. Use specific exception types for better diagnosis.
12. Lack of Sorting	The table rows are displayed as they are read from the file without any sorting. This could be inconvenient for users looking for specific data.	Implement sorting functionality for the table based on columns (e.g., Account No., Customer Name, etc.).

13. Potential Data Integrity Issues	If new records are added outside of this window, they won't be reflected in the current table view unless the window is refreshed.	Provide a mechanism to refresh the table data, such as a "Refresh" button or a listener that updates the table when changes are made to the underlying data.
14. No Search Functionality	Users cannot search for specific customer records in the table.	Add a search feature to allow users to filter or search records based on customer name, account number, or other criteria.

Viewone :

Issue	**Description**	**Recommendation**

1. Hardcoded File Name	The file name "Bank.dat" is hardcoded in the FileInputStream.	Use a constant or read the file name from a configuration file to improve flexibility.
2. Static Array Size	The records[][] array is statically sized to 500, which can lead to wasted memory if there are fewer records.	Use dynamic data structures like ArrayList<String[]> for better memory usage and flexibility when handling records.

3. Inefficient File Reading	The file reading process assumes that the file contains exactly 6 fields per record, and it reads indefinitely until an exception is thrown. This could be inefficient and lead to unexpected issues.	Use BufferedReader or a more structured approach to reading the file. Consider reading until the end of the file or using try-with-resources for better management.
4. Lack of Error Handling for File I/O	If there is an error reading the file, no specific error message is provided.	Improve error handling by catching and displaying specific exceptions like FileNotFoundException or EOFException, providing more information to the user.

5. No Check for Empty File	If the file is empty, the records array is not populated, but no message is shown about this until the user interacts with the navigation buttons.	Display an initial message if the file is empty to notify the user right away that no records exist.
6. No Data Validation	There is no validation to ensure that the data being read from the file matches the expected format (e.g., 6 fields per record).	Validate the data read from the file to ensure its consistency, preventing potential crashes or errors when accessing undefined array elements.

7. Repetitive Record Navigation	When navigating between records, the actionPerformed() method contains repetitive logic for checking the boundaries (first, back, next, last).	Refactor the navigation logic to avoid redundancy, such as creating helper methods for boundary checks and record display updates.
8. No Data Sorting	The records are displayed in the order they are read from the file. There is no sorting mechanism available.	Implement sorting functionality for the records, allowing users to view them in a sorted order (e.g., by account number or customer name).

9. Lack of Synchronization	If multiple instances of ViewOne are opened or if changes are made to the file outside this window, the view will not be updated.	Implement a method to refresh the view or listen for changes in the file and update the displayed records accordingly.
10. No Search or Filter Options	The user can only navigate through the records sequentially, without the ability to search or filter records.	Add search functionality to allow users to find specific records by account number, name, or other criteria.
11. No Confirmation for Record Navigation	When navigating between records, there's no confirmation or warning when the user reaches the first or last record.	Provide a message when the user reaches the first or last record, informing them that no more records are available.

12. Memory Inefficiency with File Handling	The entire file is read into memory at once, even if it's not all needed.	Implement pagination or lazy loading to avoid loading all records at once, especially for large files.
13. No Feedback for Empty Records	If the file is empty, only after the user tries to navigate is the message displayed.	Show a clear message when the window is first opened, indicating that no records exist. This improves user experience by setting expectations early.
14. Lack of Responsiveness	The window and controls are fixed in size, which might cause layout issues if the window is resized.	Make the window and controls more responsive by using layout managers that adapt to resizing, such as GridBagLayout or BoxLayout.

15. Potential UI Clutter	The labels and text fields could benefit from better layout management to avoid the UI looking cluttered.	Consider using a more flexible layout manager (like GridBagLayout) to improve the arrangement of the components and make the UI more organized.

Withdraw money :

Issue	**Description**	**Recommendation**
1. Hardcoded File Name	The file name "Bank.dat" is hardcoded in the FileInputStream.	Use a constant or read the file name from a configuration file to improve flexibility.

2. Static Array Size	The records[][] array is statically sized to 500, which may waste memory if there are fewer records.	Use dynamic data structures like ArrayList<String[]> for better memory usage and flexibility when handling records.
3. Inefficient File Reading	The file reading process assumes that the file contains exactly 6 fields per record, and it reads indefinitely until an exception is thrown.	Use BufferedReader or a more structured approach to reading the file. Consider reading until the end of the file or using try-with-resources for better resource management.

4. Lack of Error Handling for File I/O	If there is an error reading the file, no specific error message is provided.	Improve error handling by catching and displaying specific exceptions like FileNotFoundException or EOFException, providing more information to the user.
5. No Check for Empty File	If the file is empty, the records array is not populated, but no message is shown about this until the user interacts with the navigation buttons.	Display an initial message if the file is empty to notify the user right away that no records exist.

6. No Data Validation	There is no validation to ensure that the data being read from the file matches the expected format (e.g., 6 fields per record).	Validate the data read from the file to ensure its consistency, preventing potential crashes or errors when accessing undefined array elements.
7. Repetitive Record Navigation	The actionPerformed() method contains repetitive logic for checking the boundaries (first, back, next, last).	Refactor the navigation logic to avoid redundancy, such as creating helper methods for boundary checks and record display updates.

8. No Data Sorting	The records are displayed in the order they are read from the file. There is no sorting mechanism available.	Implement sorting functionality for the records, allowing users to view them in a sorted order (e.g., by account number or customer name).
9. Lack of Synchronization	If multiple instances of WithdrawMoney are opened or if changes are made to the file outside this window, the view will not be updated.	Implement a method to refresh the view or listen for changes in the file and update the displayed records accordingly.

10. No Search or Filter Options	The user can only navigate through the records sequentially, without the ability to search or filter records.	Add search functionality to allow users to find specific records by account number, name, or other criteria.
11. No Confirmation for Record Navigation	When navigating between records, there's no confirmation or warning when the user reaches the first or last record.	Provide a message when the user reaches the first or last record, informing them that no more records are available.
12. Memory Inefficiency with File Handling	The entire file is read into memory at once, even if it's not all needed.	Implement pagination or lazy loading to avoid loading all records at once, especially for large files.

13. No Feedback for Empty Records	If the file is empty, only after the user tries to navigate is the message displayed.	Show a clear message when the window is first opened, indicating that no records exist. This improves user experience by setting expectations early.
14. Lack of Responsiveness	The window and controls are fixed in size, which might cause layout issues if the window is resized.	Make the window and controls more responsive by using layout managers that adapt to resizing, such as GridBagLayout or BoxLayout.

15. Potential UI Clutter	The labels and text fields could benefit from better layout management to avoid the UI looking cluttered.	Consider using a more flexible layout manager (like GridBagLayout) to improve the arrangement of the components and make the UI more organized.