

Лекция 12а

Приложения с обработка на файлове

Основни теми

- Обработка на файлове с последователен **достъп и произволен достъп**.
- **Model- View- Controller** дизайн на приложение.
- Файлово приложение при **обработка на транзакции**- основни функции и реализация

7.1 Model View Controller (MVC) design

7.1.1 Сериализирано писане на обекти във файл с последователен достъп
чрез MVC design на приложението

7.1.2 Сериализирано четене на обекти във файл с последователен достъп

7.2 Обработка на файл с произволен достъп

7.2.1 Създаване на файл с произволен достъп

7.2.2 Писане във файл с произволен достъп

7.2.3 Четене от файл с произволен достъп

7.3 Пример- обработка на транзакции

Задачи

Литература:

Java How to Program, 7 Edition, глава 14

7.1 Model View Controller (MVC) design

MVC е модел предложен от Хероx в публикации от 1980-те.

Основната идея е разделяне на логиката на три **независими** компоненти:

- **Model** (бизнес логиката)
- **View** (представяне, графичен интерфейс)
- **Controller** (обработка на заявки).

В клиент- сървер приложения и в частност приложенията свързани с обработка на файл (*на практика един и същи файл може да се обработва от няколко клиента едновременно*):

- **Бизнес логиката** *се свързва с обработката на данните на приложението* (клиенти, продукти, поръчки и пр.).
- **Представянето** *се свързва с извеждането на данни и общуването с потребителя* (изграждане на потребителски интерфейс, видове контроли и разположението им, шрифтове, цветове и пр.),
- **Обработката на заявки** *е това, което свързва заедно компонентите на бизнес логиката.*



7.1.1 Използване на MVC за обработка на файл с последователен достъп

- Създаваме графично приложение илюстриращо **MVC**
 - `BankUI.java` (*presentation*)
 - `AccountRecord.java` (*model*)
 - `CreateSequentialFile.java` (*controller*)
- Създава файл с последователен достъп, в който се пишат **сериализирано AccountRecord** обекти като се използва графичния интерфейс **BankUI**



```
1 // Fig. 17.5: BankUI.java
2 // A reusable GUI for the examples in this chapter.
3 package com.deitel.jhtp5.ch17;
4
5 import java.awt.*;
6 import javax.swing.*;
7
8 public class BankUI extends JPanel {
9
10     // label text for GUI
11     protected final static String names[] = { "Account number",
12         "First name", "Last name", "Balance", "Transaction Amount" };
13
14     // GUI components; protected for future subclass access
15     protected JLabel labels[];
16     protected JTextField fields[];
17     protected JButton doTask1, doTask2;
18     protected JPanel innerPanelCenter, innerPanelSouth;
19
20     protected int size; // number of text fields in GUI
21
22     // constants representing text fields in GUI
23     public static final int ACCOUNT = 0, FIRSTNAME = 1, LASTNAME = 2,
24         BALANCE = 3, TRANSACTION = 4;
25
```

Компилира се в пакет за
многократна употреба

Bank GUI компонентата е
обща за останалите
приложения

Компонента по
представяне на
логиката

Това са бутони, чиито действия ще
се задават от конкретното
приложение
View компонентата не се обвързва с
конкретното ѝ приложение



```
26 // Set up GUI. Constructor argument size determines the number of
27 // rows of GUI components.
28 public BankUI( int mySize )
29 {
30     size = mySize;
31     labels = new JLabel[ size ];
32     fields = new JTextField[ size ];
33
34     // create labels
35     for ( int count = 0; count < labels.length; count++ )
36         labels[ count ] = new JLabel( names[ count ] );
37
38     // create text fields
39     for ( int count = 0; count < fields.length; count++ )
40         fields[ count ] = new JTextField();
41
42     // create panel to lay out labels and fields
43     innerPanelCenter = new JPanel();
44     innerPanelCenter.setLayout( new GridLayout( size, 2 ) );
45
46     // attach labels and fields to innerPanelCenter
47     for ( int count = 0; count < size; count++ ) {
48         innerPanelCenter.add( labels[ count ] );
49         innerPanelCenter.add( fields[ count ] );
50     }
51 }
```

Създават се обекти на
основните компоненти на
интерфейса без да се обвързват
с конкретното им използване



```
52 // create generic buttons; no labels or event handlers
53 doTask1 = new JButton();
54 doTask2 = new JButton();
55
56 // create panel to lay out buttons and attach buttons
57 innerPanelSouth = new JPanel();
58 innerPanelSouth.add( doTask1 );
59 innerPanelSouth.add( doTask2 );
60
61 // set layout of this container and attach panels to it
62 setLayout( new BorderLayout() );
63 add( innerPanelCenter, BorderLayout.CENTER );
64 add( innerPanelSouth, BorderLayout.SOUTH );
65
66 validate(); // validate layout
67
68 } // end constructor
69
70 // return reference to generic task button doTask1
71 public JButton getDoTask1Button()
72 {
73     return doTask1;
74 }
75
76 // return reference to generic task button doTask2
77 public JButton getDoTask2Button()
78 {
79     return doTask2;
80 }
```

Връща референции
към бутоните с цел
дефиниране на
конкретни свойства и
действия от
Компонентата
Контролер



BankUI.java

Set и Get методи с
цел дефиниране на
конкретни свойства и
действия от
Компонента
Контролер

```
81 // return reference to fields array of JTextFields
82 public JTextField[] getFields()
83 {
84     return fields;
85 }
86
87 // clear content of text fields
88 public void clearFields()
89 {
90     for ( int count = 0; count < size; count++ )
91         fields[ count ].setText( "" );
92 }
93
94 // set text field values; throw IllegalArgumentException if
95 // incorrect number of Strings in argument
96 public void setFieldValues( String strings[] )
97     throws IllegalArgumentException
98 {
99     if ( strings.length != size )
100         throw new IllegalArgumentException( "There must be " +
101             size + " Strings in the array" );
102
103     for ( int count = 0; count < size; count++ )
104         fields[ count ].setText( strings[ count ] );
105 }
106 }
```



BankUI.java

```
107
108 // get array of strings with current text field contents
109 public String[] getFieldValues()
110 {
111     String values[] = new String[ size ];
112
113     for ( int count = 0; count < size; count++ )
114         values[ count ] = fields[ count ].getText();
115
116     return values;
117 }
118
119 } // end class BankUI
```

Get методи с цел
прочитане на
въведените данни и
обработката им от
Компонентата
Контролер



```
1 // Fig. 17.6: AccountRecord.java
2 // A class that represents one record of information.
```

```
3 package com.deitel.jhtp5.ch17;
```

```
4
5 import java.io.Serializable;
```

Компилира се в пакет за многократна
употреба

```
6
7 public class AccountRecord implements Serializable {
```

AccountRecord.java

```
8     private int account;
```

```
9     private String firstName;
```

```
10    private String lastName;
```

```
11    private double balance;
```

Интерфейс `Serializable` позволява `AccountRecord`
обекти да се четат и пишат в потоци данни

```
12
13 // no-argument constructor calls other constructor with default values
```

```
14 public AccountRecord()
```

```
15 {
```

```
16     this( 0, "", "", 0.0 );
```

```
17 }
```

Компонента от
MVC

Бизнес логика

```
18
19 // initialize a record
```

```
20 public AccountRecord( int acct, String first, String last, double bal )
```

```
21 {
```

```
22     setAccount( acct );
```

```
23     setFirstName( first );
```

```
24     setLastName( last );
```

```
25     setBalance( bal );
```

```
26 }
```

```
27
```



```
28 // set account number
29 public void setAccount( int acct )
30 {
31     account = acct;
32 }
33
34 // get account number
35 public int getAccount()
36 {
37     return account;
38 }
39
40 // set first name
41 public void setFirstName( String first )
42 {
43     firstName = first;
44 }
45
46 // get first name
47 public String getFirstName()
48 {
49     return firstName;
50 }
51
```



Резюме

AccountRecord.j
ava

```
52 // set last name
53 public void setLastName( String last )
54 {
55     lastName = last;
56 }
57
58 // get last name
59 public String getLastName()
60 {
61     return lastName;
62 }
63
64 // set balance
65 public void setBalance( double bal )
66 {
67     balance = bal;
68 }
69
70 // get balance
71 public double getBalance()
72 {
73     return balance;
74 }
75
76 } // end class AccountRecord
```



```
1 // Fig. 17.7: CreateSequentialFile.java
2 // Writing objects sequentially to a file with class ObjectOutputStream.
3 import java.io.*;
4 import java.awt.*;
5 import java.awt.event.*;
6 import javax.swing.*;
7
8 import com.deitel.jhtp5.ch17.BankUI;
9 import com.deitel.jhtp5.ch17.AccountRecord;
10
11 public class CreateSequentialFile extends JFrame {
12     private ObjectOutputStream output;
13     private BankUI userInterface;
14     private JButton enterButton, openButton;
15
16     // set up GUI
17     public CreateSequentialFile()
18     {
19         super( "Creating a Sequential File of Objects" );
20
21         // create instance of reusable user interface
22         userInterface = new BankUI( 4 ); // four textfields
23         getContentPane().add( userInterface, BorderLayout.CENTER );
24
25         // configure button doTask1 for use in this program
26         openButton = userInterface.getDoTask1Button();
27         openButton.setText( "Save into File ..." );
```

Импорт на класовете
дефиниращи бизнес логиката
и представянето

Компонента от
MVC

Контролер
(различни
контролери могат
да използват едни и
съща бизнес логика
и представяне!)

Създаване и конкретизиране на представянето-
дефинират се определени свойства на
контролите(бутон) и действията им



CreateSequentialFile
.java

```
28 // register listener to call openFile when button pressed
29 openButton.addActionListener(
30
```

```
31 // anonymous inner class to handle openButton event
32 new ActionListener() {
33
```

```
34 // call openFile when button pressed
35 public void actionPerformed((ActionEvent event) )
36 {
37     openFile();
38 }
39
```

```
40 } // end anonymous inner class
41
```

```
42 ); // end call to addActionListener
43
```

```
44 // configure button doTask2 for use in this program
45 enterButton = userInterface.getDoTask2Button();
```

```
46 enterButton.setText( "Enter" );
47 enterButton.setEnabled( false ); // disable button
48
```

```
49 // register listener to call addRecord when button pressed
50 enterButton.addActionListener(
51
52
```

Това трябва да се програмира
директно- не става с използване
на wizard

Създаване и
конкретизиране на
представянето-
дефинират се
определени
свойства на
контролите(бутон) и
действията им

Това също трябва да се
програмира директно- не става с
използване на wizard



CreateSequentialFile
.java

```
53 // anonymous inner class to handle enterButton event
54 new ActionListener() {
55
56     // call addRecord when button pressed
57     public void actionPerformed((ActionEvent event) )
58     {
59         addRecord();
60     }
61
62 } // end anonymous inner class
63
64 ); // end call to addActionListener
65
66 // register window listener to handle window closing event
67 addWindowListener(
68
69     // anonymous inner class to handle windowClosing event
70     new WindowAdapter() {
71
72         // add current record in GUI to file, then close file
73         public void windowClosing( WindowEvent event )
74         {
75             if ( output != null )
76                 addRecord();
77
78             closeFile();
79         }
80     }
81 );
```

Дефинира действие за
изпълнение при затваряне на
прозореца с потребителския
интерфейс за по-голяма
сигурност




```
80     } // end anonymous inner class
81
82
83 ); // end call to addWindowListener
84
85 setSize( 300, 200 );
86 setVisible( true );
87
88 } // end CreateSequentialFile constructor
89
90 // allow user to specify file name
91 private void openFile()
92 {
93     // display file dialog, so user can choose file to open
94     JFileChooser fileChooser = new JFileChooser();
95     fileChooser.setSelectionMode( JFileChooser.FILES_ONLY );
96
97     int result = fileChooser.showSaveDialog( this );
98
99     // if user clicked Cancel button on dialog, return
100     if ( result == JFileChooser.CANCEL_OPTION )
101         return;
102
103     File fileName = fileChooser.getSelectedFile(); // get selected file
104 }
```

Създава обект JFileChooser с
референция fileChooser

Константата FILES_ONLY
указва избор само на файлове

showSaveDialog
извежда
JFileChooser
диалог Save

Прочитане на името на избрания файл

Излиза, ако е
избран Cancel
бутон в
диалоговия
прозорец



CreateSequentialFile
.java

```
105 // display error if invalid
106 if ( fileName == null || fileName.getName().equals( "" ) )
107     JOptionPane.showMessageDialog( this, "Invalid File Name",
108         "Invalid File Name", JOptionPane.ERROR_MESSAGE );
109
110 else {
111
112     // open file
113     try {
114         output = new ObjectOutputStream(
115             new FileOutputStream( fileName ) );
116
117         openButton.setEnabled( false );
118         enterButton.setEnabled( true );
119     }
120
121     // process exceptions from opening file
122     catch ( IOException ioException ) {
123         JOptionPane.showMessageDialog( this, "Error Opening File",
124             "Error", JOptionPane.ERROR_MESSAGE );
125     }
126
127 } // end else
128
129 } // end method openFile
130
```

Отваря файла с
избраното име



Метод `closeFile`
затваря файла и
прекратява приложението

```
131 // close file and terminate application
132 private void closeFile()
133 {
134     // close file
135     try {
136         output.close();
137         System.exit( 0 );
138     }
139
140     // process exceptions from closing file
141     catch( IOException ioException ) {
142         JOptionPane.showMessageDialog( this, "Error closing file",
143             "Error", JOptionPane.ERROR_MESSAGE );
144         System.exit( 1 );
145     }
146
147 } // end method closeFile
148
149 // add record to file
150 public void addRecord()
151 {
152     int accountNumber = 0;
153     AccountRecord record;
154     String fieldValues[] = userInterface.getFieldValues();
155 }
```

Прочита въведеното в текстовите
полета на графичния интерфейс
(MVC компонента)



CreateSequentialFile
.java

```
156 // if account field value is not empty
157 if ( ! fieldValues[ BankUI.ACCOUNT ].equals( "" ) ) {
158
159     // output values to file
160     try {
161         accountNumber = Integer.parseInt(
162             fieldValues[ BankUI.ACCOUNT ] );
163
164         if ( accountNumber > 0 ) {
165
166             // create new record
167             record = new AccountRecord( accountNumber,
168                 fieldValues[ BankUI.FIRSTNAME ],
169                 fieldValues[ BankUI.LASTNAME ],
170                 Double.parseDouble( fieldValues[ BankUI.BALANCE ] ) );
171
172             // output record and flush buffer
173             output.writeObject( record );
174             output.flush();
175         }
176     }
177     else {
178         JOptionPane.showMessageDialog( this,
179             "Account number must be greater than 0",
180             "Bad account number", JOptionPane.ERROR_MESSAGE );
181     }
182 }
```

Създава
AccountRecord
обект за нов запис

Записва обекта във файла
и за по- сигурно форсира
изчистване на входно
изходния буфер



CreateSequentialFile
.java

```
183         // clear textfields
184         userInterface.clearFields();
185
186     } // end try
187
188     // process invalid account number or balance format
189     catch ( NumberFormatException formatException ) {
190         JOptionPane.showMessageDialog( this,
191             "Bad account number or balance", "Invalid Number Format",
192             JOptionPane.ERROR_MESSAGE );
193     }
194
195     // process exceptions from file output
196     catch ( IOException ioException ) {
197         JOptionPane.showMessageDialog( this, "Error writing to file",
198             "IO Exception", JOptionPane.ERROR_MESSAGE );
199         closeFile();
200     }
201
202 } // end if
203
204 } // end method addRecord
205
```



Резюме

CreateSequentialFile
.java

```
206 public static void main( String args[] )
207 {
208     new CreateSequentialFile();
209 }
210
211 } // end class CreateSequentialFile
```

BankUI графичен
интерфейс



Creating a Sequential File of Objects

Account number

First name

Last name

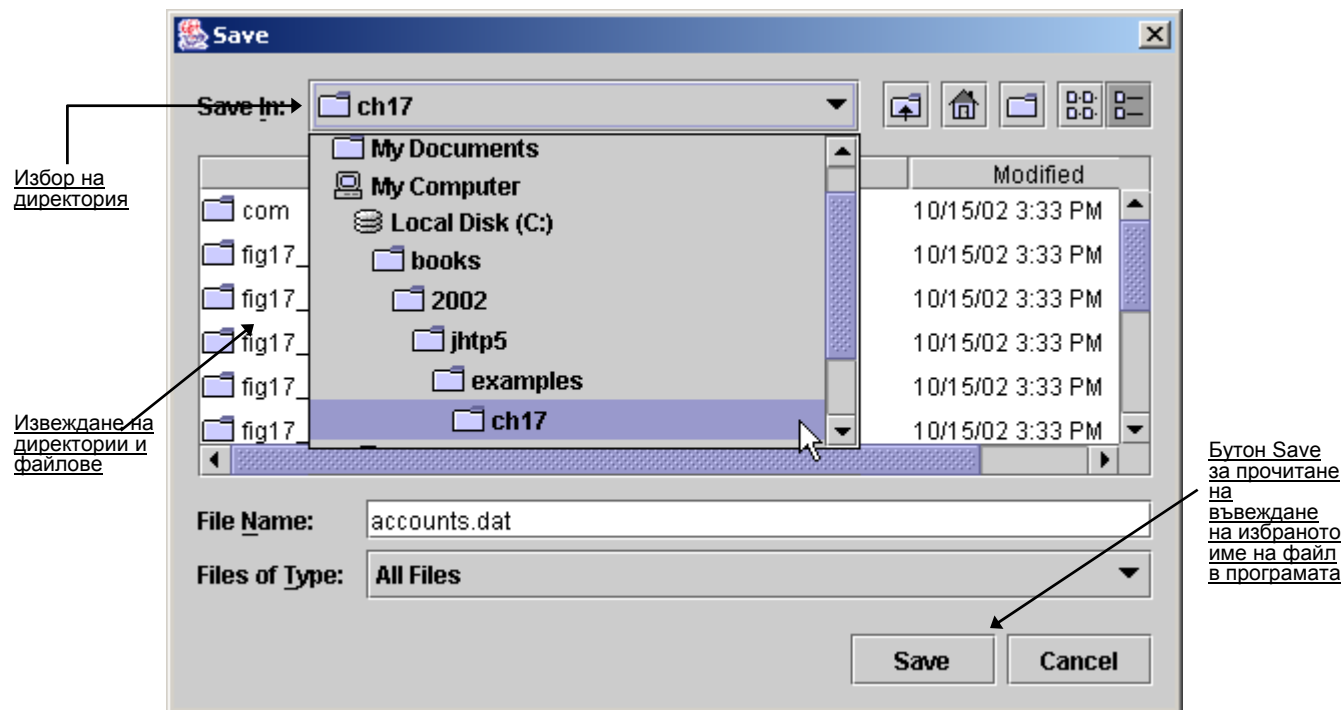
Balance

Save into File ... Enter



Резюме

CreateSequentialFile
.java



Creating a Sequential File of Objects	
Account number	100
First name	Bob
Last name	Jones
Balance	24.98
<div>Save into File ... Enter</div>	



7.1.2 Сериализирано четене на файл с последователен достъп

- Използваме **същите MVC компоненти**
 - Бизнес логика - модел
 - Представяне - view

с **друг контролер** за създаване на ново приложение

Предимства

- Реализира се концепцията за многократно използване на код
- Независимо обновяване на бизнес логиката и графичния интерфейс
- Лесно тестване и много други



Fig. 7.1 Примерни данни за въвеждане във файл

Sample Data			
100	Bob	Jones	24.98
200	Steve	Doe	-345.67
300	Pam	White	0.00
400	Sam	Stone	-42.16
500	Sue	Rich	224.62

ReadSequentialFile
.java

НОВ Контролер

СЪЩЕТЕ бизнес
логика и
представяне

```
1 // Fig. 17.9: ReadSequentialFile.java
2 // This program reads a file of objects sequentially
3 // and displays each record.
4 import java.io.*;
5 import java.awt.*;
6 import java.awt.event.*;
7 import javax.swing.*;
8
9 import com.deitel.jhtp5.ch17.*;
10
11 public class ReadSequentialFile extends JFrame {
12     private ObjectInputStream input;
13     private BankUI userInterface;
14     private JButton nextButton, openButton;
15
16     // Constructor -- initialize the Frame
17     public ReadSequentialFile()
18     {
19         super( "Reading a Sequential File of Objects" );
20
21         // create instance of reusable user interface
22         userInterface = new BankUI( 4 ); // four textfields
23         getContentPane().add( userInterface, BorderLayout.CENTER );
24     }
25 }
```

Създаване на обект от графичния
интерфейс- MVC компонента



```
25 // get reference to generic task button doTask1 from BankUI
26 openButton = userInterface.getDoTask1Button();
27 openButton.setText( "Open File" );
28
29 // register listener to call openFile when button pressed
30 openButton.addActionListener(
31
32     // anonymous inner class to handle openButton event
33     new ActionListener() {
34
35         // close file and terminate application
36         public void actionPerformed((ActionEvent event) )
37         {
38             openFile();
39         }
40
41     } // end anonymous inner class
42
43 ); // end call to addActionListener
44
45 // register window listener for window closing event
46 addWindowListener(
47
48     // anonymous inner class to handle windowClosing event
49     new WindowAdapter() {
50
```

Get метод за
дефиниране на
желаното действие
за бутон – **отваря**
файла



```
51 // close file and terminate application
52 public void windowClosing( WindowEvent event )
53 {
54     if ( input != null )
55         closeFile();
56
57     System.exit( 0 );
58 }
59
60 } // end anonymous inner class
61
62 ); // end call to addWindowListener
63
64 // get reference to generic task button doTask2 from BankUI
65 nextButton = userInterface.getDoTask2Button();
66 nextButton.setText( "Next Record" );
67 nextButton.setEnabled( false );
68
69 // register listener to call readRecord when button pressed
70 nextButton.addActionListener(
71
72     // anonymous inner class to handle nextRecord event
73     new ActionListener() {
74
```

Get метод за
дефиниране на
желаното действие
за бутон – **чете**
следващия запис



```
75 // call readRecord when user clicks nextRecord
76 public void actionPerformed( ActionEvent event )
77 {
78     readRecord();
79 }
80
81 } // end anonymous inner class
82
83 ); // end call to addActionListener
84
85 pack();
86 setSize( 300, 200 );
87 setVisible( true );
88
89 } // end ReadSequentialFile constructor
90
91 // enable user to select file to open
92 private void openFile()
93 {
94     // display file dialog so user can select file to open
95     JFileChooser fileChooser = new JFileChooser();
96     fileChooser.setFileSelectionMode( JFileChooser.FILES_ONLY );
97
98     int result = fileChooser.showOpenDialog( this );
99
```

Създава обект от JFileChooser и го
реферира с fileChooser

Константата FILES_ONLY
указва да се избират само
файлове

Методът showOpenDialog извежда
JFileChooser диалогът Open



ReadSequentialFile.
java

```
100 // if user clicked Cancel button on dialog, return
101 if ( result == JFileChooser.CANCEL_OPTION )
102     return;
103
104 // obtain selected file
105 File fileName = fileChooser.getSelectedFile();
106
107 // display error if file name invalid
108 if ( fileName == null || fileName.getName().equals( "" ) )
109     JOptionPane.showMessageDialog( this, "Invalid File Name",
110         "Invalid File Name", JOptionPane.ERROR_MESSAGE );
111
112 else {
113
114     // open file
115     try {
116         input = new ObjectInputStream(
117             new FileInputStream( fileName ) );
118
119         openButton.setEnabled( false );
120         nextButton.setEnabled( true );
121     }
122
123     // process exceptions opening file
124     catch ( IOException ioException ) {
125         JOptionPane.showMessageDialog( this, "Error Opening File",
126             "Error", JOptionPane.ERROR_MESSAGE );
127     }
128 }
```

Прекратява методът ако е
избран бутон Cancel

Прочита избраното
име на файл

Отваря
файл с
избраното
име



```
128
129     } // end else
130
131 } // end method openFile
132
133 // read record from file
134 public void readRecord()
135 {
136     AccountRecord record;
137
138     // input the values from the file
139     try {
140         record = ( AccountRecord ) input.readObject();
141
142         // create array of Strings to display in GUI
143         String values[] = { String.valueOf( record.getAccount() ),
144                             record.getFirstName(), record.getLastName(),
145                             String.valueOf( record.getBalance() ) };
146
147         // display record contents
148         userInterface.setFieldValues( values );
149     }
150
151     // display message when end-of-file reached
152     catch ( EOFException endOfFileException ) {
153         nextButton.setEnabled( false );
154     }
```

Метод readObject
прочита Object от
ObjectInputStream

Създава масив от String с
инициализиращ списък и
го извежда н аграфичния
интерфейс

Прочетен е край на
потока с данни



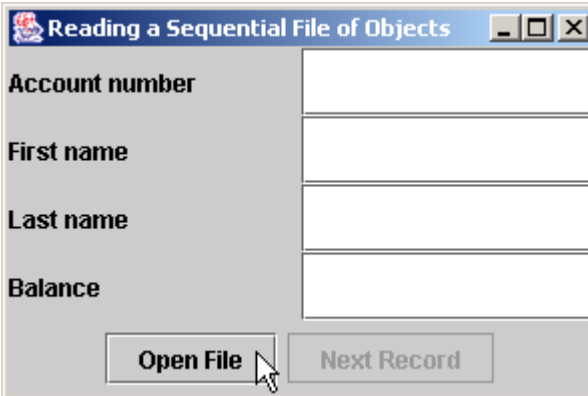
ReadSequentialFile .java

```
155     JOptionPane.showMessageDialog( this, "No more records in file",
156         "End of File", JOptionPane.ERROR_MESSAGE );
157 }
158
159 // display error message if class is not found
160 catch ( ClassNotFoundException classNotFoundException ) {
161     JOptionPane.showMessageDialog( this, "Unable to create object",
162         "Class Not Found", JOptionPane.ERROR_MESSAGE );
163 }
164
165 // display error message if cannot read due to problem with file
166 catch ( IOException ioException ) {
167     JOptionPane.showMessageDialog( this,
168         "Error during read from file",
169         "Read Error", JOptionPane.ERROR_MESSAGE );
170 }
171
172 } // end method readRecord
173
174 // close file and terminate application
175 private void closeFile() ←
176 {
177     // close file and exit
178     try {
179         input.close();
180         System.exit( 0 );
181     }
```

Метод closeFile
затваря файла




```
182
183 // process exception while closing file
184 catch ( IOException ioException ) {
185     JOptionPane.showMessageDialog( this, "Error closing file",
186         "Error", JOptionPane.ERROR_MESSAGE );
187
188     System.exit( 1 );
189 }
190
191 } // end method closeFile
192
193 public static void main( String args[] )
194 {
195     new ReadSequentialFile();
196 }
197
198 } // end class ReadSequentialFile
```



Reading a Sequential File of Objects

Account number

First name

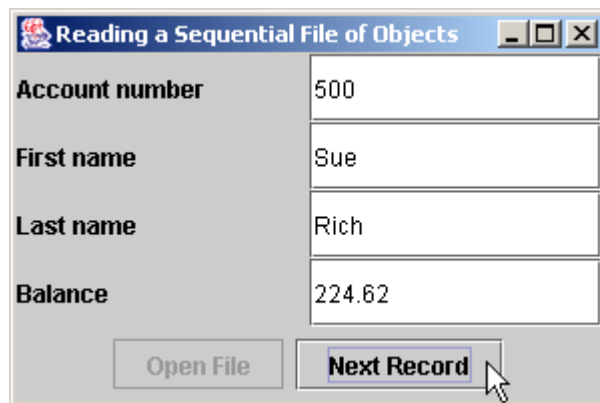
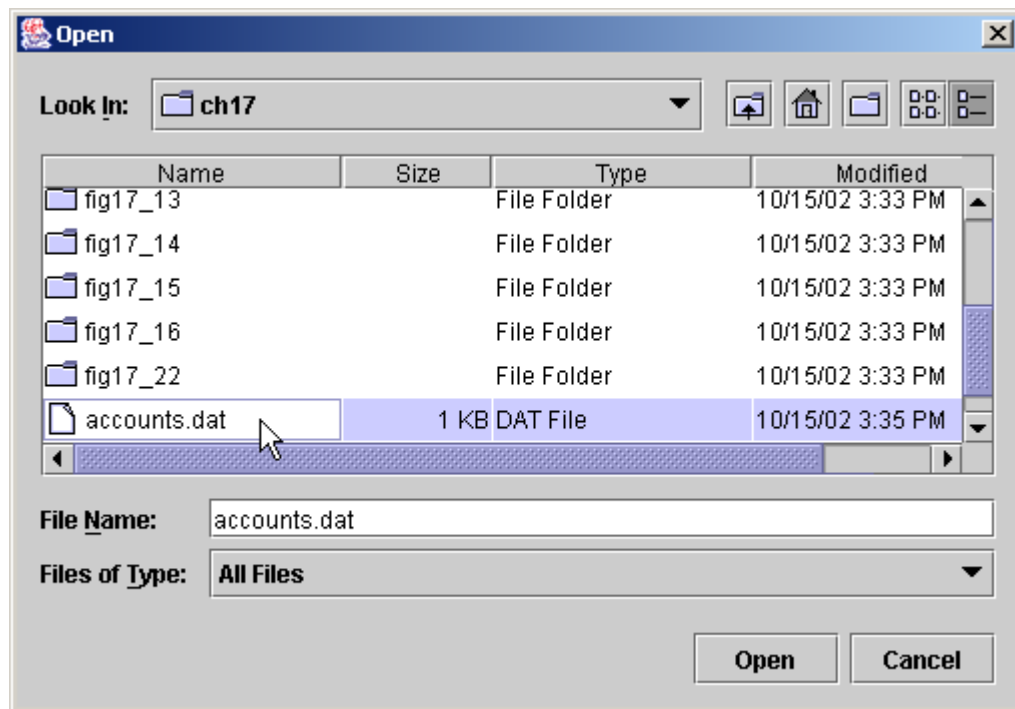
Last name

Balance

Open File Next Record



ReadSequentialFile. java



7.2 Обработка на файл с произволен достъп

- “Директен достъп” приложения

- Записите да са достъпни на момента

Пример- Обработка на транзакции

- Несъвместими с ограниченията на последователен достъп

- Файлове в произволен достъп

- Достъпът до отделни записи е бърз и директен

- Използват фиксирана дължина на записите

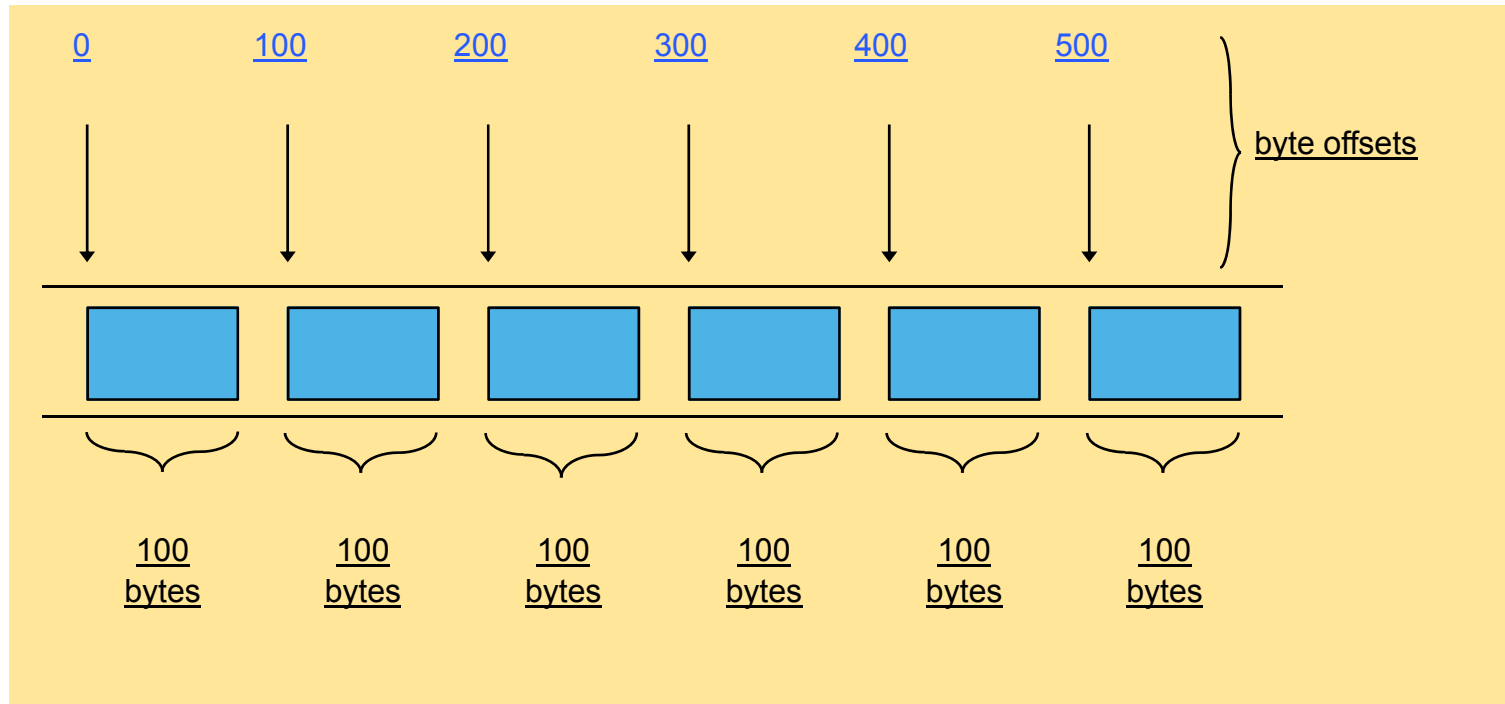
- Позволява лесно да се сметне позицията на запис във файл

- Позволява вмъкване на записи без повреждане на останалите записи

- Fig. 7.2 илюстрира файл с произволен достъп



Fig. 7.2 Java модел на файл с произволен достъп



7.2.1 Създаване на файл с произволен достъп

- `RandomAccessFile` обект
 - Има свойствата на `FileInputStream` и `FileOutputStream` и интерфейсите `DataInput` и `DataOutput`, реализирани в класове `DataInputStream` и `DataOutputStream`
 - класове `DataInputStream` и `DataOutputStream` позволяват четене и писане на място от файла указано от файл указател (пойнтер) за позиция
 - Обработка всички данни като примитивни, масиви от байтове и низове
 - Рядко се пише отделно поле във файл с произволен достъп, обикновено се записва един обект във файла
- Пример- програма за обработка на транзакции



7.2.1 Създаване на файл с произволен достъп

- **RandomAccessFile** обект
 - При **отваряне на файл** с произволен достъп се създава обект от **RandomAccessFile**, който има **указател за текущия запис** за четене или писане и позволява обработката на данните записани във файла като примитивни
 - Пример- при писане на **int** се записват 4 байта във файла, при четене на **double** се прочитат 8 байта от файла (размерът на примитивните данни е фиксиран в JAVA)



7.2.1 Създаване на файл с произволен достъп

- `RandomAccessFile` обект
 - Методи `readInt`, `readDouble`, `readChar` се използват за **четене** на `integer`, `double` и текстови данни от файл
 - Методи `writeInt`, `writeDouble`, `writeChars` се използват за **писане** на `integer`, `double` и текстови данни във файл
 - Режим на отваряне на файл– задава дали файлът се отваря за **четене** (“`r`”), за **четене и писане** (“`rw`”).
Задава се като втори аргумент в конструктора на обект от `RandomAccessFile`



Съвет за по-добро качество 7.2

Отваряйте файл с “r” режим за четене, ако не искате да се променя съдържанието на файла

7.2.1 Създаване на файл с произволен достъп

- MVC модел
 - Допълваме бизнес логиката от предишните примери с методи за четене и писане във файл с произволен достъп
 - Използваме наследственост `RandomAccessAccountRecord`



7.2.1 Създаване на файл с произволен достъп

- `class StringBuffer`— позволява динамична обработка на низове
 - `String` обектите са *immutable*, `StringBuffer` позволява низовете се променят динамично
 - Същата функционалност както `StringBuilder`
 - Използва **капацитет**, аналогично на `StringBuilder`

RandomAccessAccountRecord.j
ava

```
1 // Fig. 17.12: RandomAccessAccountRecord.java
2 // Subclass of AccountRecord for random access file programs.
3 package com.deitel.jhttp5.ch17;
4
5 import java.io.*;
6
7 public class RandomAccessAccountRecord extends AccountRecord {
8
9     public static final int SIZE = 72; // bytes in one record
10
11     // no-argument constructor calls other constructor with default values
12     public RandomAccessAccountRecord()
13     {
14         this( 0, "", "", 0.0 );
15     }
16
17     // initialize a RandomAccessAccountRecord
18     public RandomAccessAccountRecord( int account, String firstName,
19         String lastName, double balance )
20     {
21         super( account, firstName, lastName, balance );
22     }
23 }
```

Произведен клас
на
AccountRecord

Задава размер на запис от
AccountRecord
4(int), 30(char),
30(char), 8(double)



```

24 // read a record from specified RandomAccessFile
25 public void read( RandomAccessFile file ) throws IOException
26 {
27     setAccount( file.readInt() );
28     setFirstName( readName( file ) );
29     setLastName( readName( file ) );
30     setBalance( file.readDouble() );
31 }
32
33 // ensure that name is proper length
34 private String readName( RandomAccessFile file ) throws IOException
35 {
36     char name[] = new char[ 15 ], temp;
37
38     for ( int count = 0; count < name.length; count++ ) {
39         temp = file.readChar();
40         name[ count ] = temp;
41     }
42
43     return new String( name ).replace( '\0', ' ' );
44 }
45

```

Метод read чете полетата
RandomAccessAccountRecord на
RandomAccessFile

Метод readInt
чете едно цяло
число

Метод readDouble
чете double

Метод readChar
чете един символ

Ако прочетения низ е по- къс от 15 символа низът се
допълва със '\0' по подразбиране. Java не извежда '\0'
и затова тези символи се заместват с ' '



```
46 // write a record to specified RandomAccessFile
47 public void write( RandomAccessFile file ) throws IOException
48 {
49     file.writeInt( getAccount() );
50     writeName( file, getFirstName() );
51     writeName( file, getLastName() );
52     file.writeDouble( getBalance() );
53 }
54
55 // write a name to file; maximum of 15 characters
56 private void writeName( RandomAccessFile file, String name )
57     throws IOException
58 {
59     StringBuffer buffer = null;
60
61     if ( name != null )
62         buffer = new StringBuffer( name );
63     else
64         buffer = new StringBuffer( 15 );
65
66     buffer.setLength( 15 );
67     file.writeChars( buffer.toString() );
68 }
69
70 } // end class RandomAccessAccountRecord
```

Метод write пише
един запис в
RandomAccessFile

Метод writeInt
пише един int

Метод writeDouble
пише един double

Метод writeName
пише String във файла

Пишат се точно 15 символа, за да
се запази дължината на записа

Метод writeChars пише String



Пример-
създава файл с
100 празни
записа

```
1 // Fig. 17.13: CreateRandomFile.java
2 // Creates random access file by writing 100 empty records to disk.
3 import java.io.*;
4 import javax.swing.*;
5
6 import com.deitel.jhtp5.ch17.RandomAccessAccountRecord;
7
8 public class CreateRandomFile {
9
10     private static final int NUMBER_RECORDS = 100;
11
12     // enable user to select file to open
13     private void createFile()
14     {
15         // display dialog so user can choose file
16         JFileChooser fileChooser = new JFileChooser();
17         fileChooser.setFileSelectionMode( JFileChooser.FILES_ONLY );
18
19         int result = fileChooser.showSaveDialog( null );
20
21         // if user clicked Cancel button on dialog, return
22         if ( result == JFileChooser.CANCEL_OPTION )
23             return;
24
25         // obtain selected file
26         File fileName = fileChooser.getSelectedFile();
```



```
27 // display error if file name invalid
28 if ( fileName == null || fileName.getName().equals( "" ) )
29     JOptionPane.showMessageDialog( null, "Invalid File Name",
30         "Invalid File Name", JOptionPane.ERROR_MESSAGE );
31
32
33 else {
34
35     // open file
36     try {
37         RandomAccessFile file =
38             new RandomAccessFile( fileName, "rw" );
39
40         RandomAccessAccountRecord blankRecord =
41             new RandomAccessAccountRecord();
42
43         // write 100 blank records
44         for ( int count = 0; count < NUMBER_RECORDS; count++ )
45             blankRecord.write( file );
46
47         file.close(); // close file
48
49         // display message that file was created
50         JOptionPane.showMessageDialog( null, "Created file " +
51             fileName, "Status", JOptionPane.INFORMATION_MESSAGE );
```

Отваря за четене и
писане
RandomAccessFile

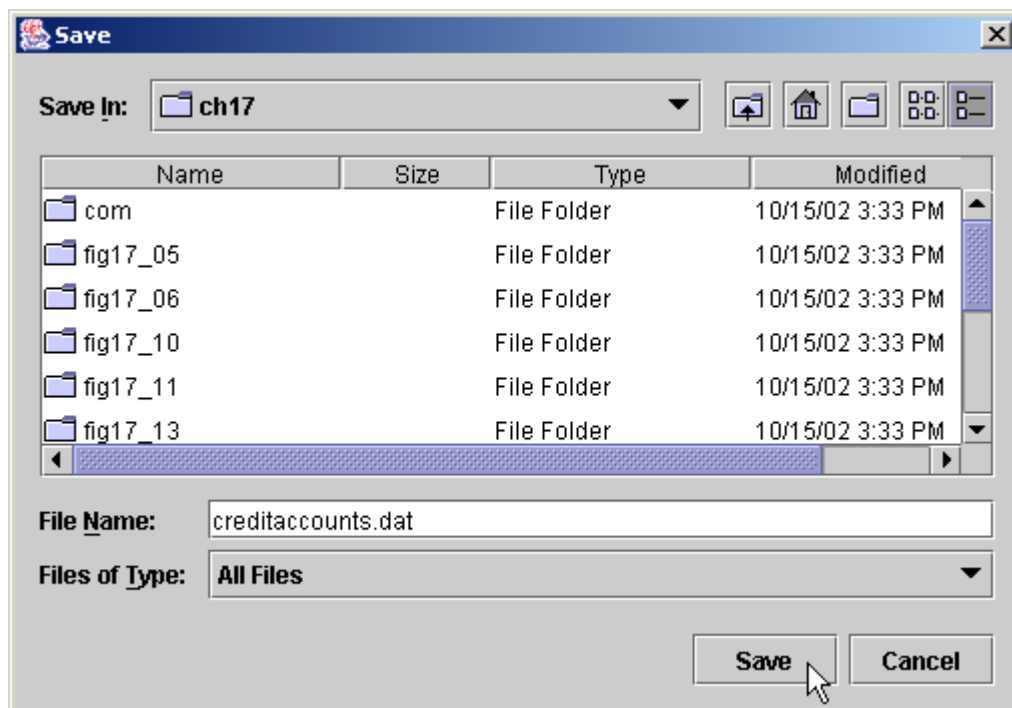
Записва 100 празни
записа



```
52         System.exit( 0 ); // terminate program
53
54     } // end try
55
56     // process exceptions during open, write or close file operations
57     catch ( IOException ioException ) {
58         JOptionPane.showMessageDialog( null, "Error processing file",
59             "Error processing file", JOptionPane.ERROR_MESSAGE );
60
61         System.exit( 1 );
62     }
63
64 } // end else
65
66 } // end method createFile
67
68
69 public static void main( String args[] )
70 {
71     CreateRandomFile application = new CreateRandomFile();
72     application.createFile();
73 }
74
75 } // end class CreateRandomFile
```



CreateRandomFile.
java



7.2.2 Писане във файл с произволен достъп

- `RandomAccessFile` има метод `seek`
 - Задава позиция на запис във файла, осигуряваща достъп за четене или писане
 - Премества указателя на файла в тази позиция



Резюме

WriteRandomFile
.java

Нов контролер-

писане във файл
с произволен
достъп



```
1 // Fig. 17.14: WriteRandomFile.java
2 // This program uses textfields to get information from the user at the
3 // keyboard and writes the information to a random-access file.
4 import java.awt.*;
5 import java.awt.event.*;
6 import java.io.*;
7 import javax.swing.*;
8
9 import com.deitel.jhtp5.ch17.*;
10
11 public class WriteRandomFile extends JFrame {
12     private RandomAccessFile output;
13     private BankUI userInterface;
14     private JButton enterButton, openButton;
15
16     private static final int NUMBER_RECORDS = 100;
17
18     // set up GUI
19     public WriteRandomFile()
20     {
21         super( "Write to random access file" );
22
23         // create instance of reusable user interface BankUI
24         userInterface = new BankUI( 4 ); // four textfields
25         getContentPane().add( userInterface,
26                               BorderLayout.CENTER );
```

```
27 // get reference to generic task button doTask1 in BankUI
28 openButton = userInterface.getDoTask1Button();
29 openButton.setText( "Open..." );
30
31
32 // register listener to call openFile when button pressed
33 openButton.addActionListener(
34
35     // anonymous inner class to handle openButton event
36     new ActionListener() {
37
38         // allow user to select file to open
39         public void actionPerformed((ActionEvent event) )
40         {
41             openFile();
42         }
43
44     } // end anonymous inner class
45
46 ); // end call to addActionListener
47
48 // register window listener for window closing event
49 addWindowListener(
50
```



```
51 // anonymous inner class to handle windowClosing event
52 new WindowAdapter() {
53
54     // add record in GUI, then close file
55     public void windowClosing( WindowEvent event )
56     {
57         if ( output != null )
58             addRecord();
59
60         closeFile();
61     }
62
63 } // end anonymous inner class
64
65 ); // end call to addWindowListener
66
67 // get reference to generic task button doTask2 in BankUI
68 enterButton = userInterface.getDoTask2Button();
69 enterButton.setText( "Enter" );
70 enterButton.setEnabled( false );
71
72 // register listener to call addRecord when button pressed
73 enterButton.addActionListener(
74
```



```
75 // anonymous inner class to handle enterButton event
76 new ActionListener() {
77
78     // add record to file
79     public void actionPerformed((ActionEvent event) )
80     {
81         addRecord();
82     }
83
84 } // end anonymous inner class
85
86 ); // end call to addActionListener
87
88 setSize( 300, 150 );
89 setVisible( true );
90 }
91
92 // enable user to choose file to open
93 private void openFile()
94 {
95     // display file dialog so user can select file
96     JFileChooser fileChooser = new JFileChooser();
97     fileChooser.setFileSelectionMode( JFileChooser.FILES_ONLY );
98
99     int result = fileChooser.showOpenDialog( this );
100
```



WriteRandomFile
.java

Line 117

```
101 // if user clicked Cancel button on dialog, return
102 if ( result == JFileChooser.CANCEL_OPTION )
103     return;
104
105 // obtain selected file
106 File fileName = fileChooser.getSelectedFile();
107
108 // display error if file name invalid
109 if ( fileName == null || fileName.getName().equals( "" ) )
110     JOptionPane.showMessageDialog( this, "Invalid File Name",
111         "Invalid File Name", JOptionPane.ERROR_MESSAGE );
112
113 else {
114
115     // open file
116     try {
117         output = new RandomAccessFile( fileName, "rw" );
118         enterButton.setEnabled( true );
119         openButton.setEnabled( false );
120     }
121
122     // process exception while opening file
123     catch ( IOException ioException ) {
124         JOptionPane.showMessageDialog( this, "File does not exist",
125             "Invalid File Name", JOptionPane.ERROR_MESSAGE );
126     }
```

Отваря за четене и
писане
RandomAccessFile



WriteRandomFile
.java

```
127
128     } // end else
129
130 } // end method openFile
131
132 // close file and terminate application
133 private void closeFile()
134 {
135     // close file and exit
136     try {
137         if ( output != null )
138             output.close();
139
140         System.exit( 0 );
141     }
142
143     // process exception while closing file
144     catch( IOException ioException ) {
145         JOptionPane.showMessageDialog( this, "Error closing file",
146             "Error", JOptionPane.ERROR_MESSAGE );
147
148         System.exit( 1 );
149     }
150
151 } // end method closeFile
152
```



WriteRandomFile
.java

```
153 // add one record to file
154 private void addRecord()
155 {
156     String fields[] = userInterface.getFieldValues();
157
158     // ensure account field has a value
159     if ( ! fields[ BankUI.ACCOUNT ].equals( "" ) ) {
160
161         // output values to file
162         try {
163             int accountNumber =
164                 Integer.parseInt( fields[ ACCOUNT ] );
165
166             if ( accountNumber > 0 && accountNumber <= NUMBER_RECORDS ) {
167                 RandomAccessAccountRecord record
168                     new RandomAccessAccountRecord();
169
170                 record.setAccount( accountNumber );
171                 record.setFirstName( fields[ BankUI.FIRSTNAME ] );
172                 record.setLastName( fields[ BankUI.LASTNAME ] );
173                 record.setBalance( Double.parseDouble(
174                     fields[ BankUI.BALANCE ] ) );
175
176                 output.seek( ( accountNumber - 1 ) *
177                     RandomAccessAccountRecord.SIZE );
178                 record.write( output );
179             }
180         }
181     }
182 }
```

Добавя (променя) запис
за даден номер на
банкова сметка

Премества указателя на
файла в записът с
желания номер на сметка
(спрямо началото на
файла)



```
180
181     else {
182         JOptionPane.showMessageDialog( this,
183             "Account must be between 1 and 100",
184             "Invalid account number", JOptionPane.ERROR_MESSAGE );
185     }
186
187     userInterface.clearFields(); // clear TextFields
188
189 } // end try
190
191 // process improper account number or balance format
192 catch ( NumberFormatException formatException ) {
193     JOptionPane.showMessageDialog( this,
194         "Bad account number or balance",
195         "Invalid Number Format", JOptionPane.ERROR_MESSAGE );
196 }
197
198 // process exceptions while writing to file
199 catch ( IOException ioException ) {
200     JOptionPane.showMessageDialog( this,
201         "Error writing to the file", "IO Exception",
202         JOptionPane.ERROR_MESSAGE );
203     closeFile();
204 }
```



Резюме

WriteRandomFile
.java

```
205
206     } // end if
207
208 } // end method addRecord
209
210 public static void main( String args[] )
211 {
212     new WriteRandomFile();
213 }
214
215 } // end class WriteRandomFile
```



Write to random access file

Account number

First name

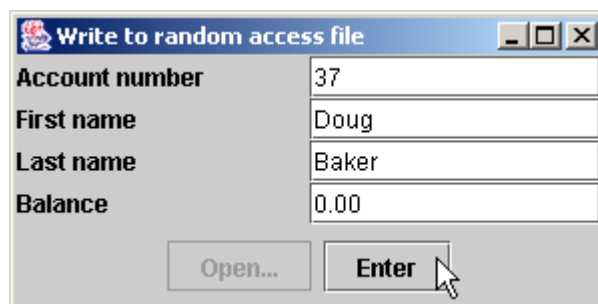
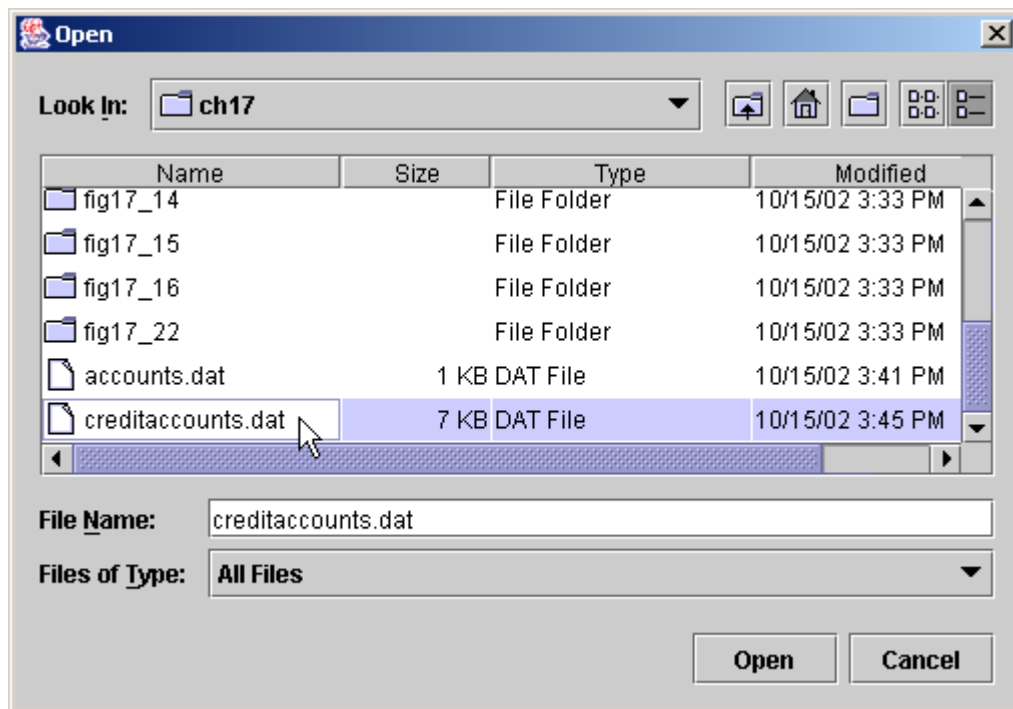
Last name

Balance

Open... Enter



writeRandomFile
.java



7.2.3 Четене от файл с произволен достъп

- Прочитане и извеждане на всички записи от `RandomAccessFile`
- Променяме само контролер компонентата в MVC модела



ReadRandomFile. java

```
1 // Fig. 17.15: ReadRandomFile.java
2 // This program reads a random-access file sequentially and
3 // displays the contents one record at a time in text fields.
4 import java.awt.*;
5 import java.awt.event.*;
6 import java.io.*;
7 import java.text.DecimalFormat;
8 import javax.swing.*;
9
10 import com.deitel.jhtp5.ch17.*;
11
12 public class ReadRandomFile extends JFrame {
13     private BankUI userInterface;
14     private RandomAccessFile input;
15     private JButton nextButton, openButton;
16
17     private static DecimalFormat twoDigits = new DecimalFormat( "0.00" );
18
19     // set up GUI
20     public ReadRandomFile()
21     {
22         super( "Read Client File" );
23
24         // create reusable user interface instance
25         userInterface = new BankUI( 4 ); // four textfields
26         getContentPane().add( userInterface );
```

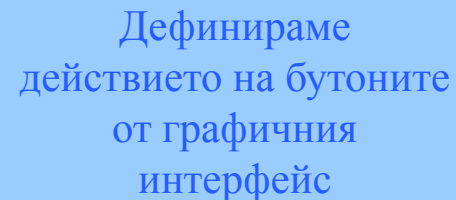


```
27 // configure generic doTask1 button from BankUI
28 openButton = userInterface.getDoTask1Button();
29 openButton.setText( "Open File for Reading..." );
30
31
32 // register listener to call openFile when button pressed
33 openButton.addActionListener(
34
35     // anonymous inner class to handle openButton event
36     new ActionListener() {
37
38         // enable user to select file to open
39         public void actionPerformed((ActionEvent event) )
40         {
41             openFile();
42         }
43
44     } // end anonymous inner class
45
46 ); // end call to addActionListener
47
48 // configure generic doTask2 button from BankUI
49 nextButton = userInterface.getDoTask2Button();
50 nextButton.setText( "Next" );
51 nextButton.setEnabled( false );
```



```
52 // register listener to call readRecord when button pressed
53 nextButton.addActionListener(
54
55     // anonymous inner class to handle nextButton event
56     new ActionListener() {
57
58         // read a record when user clicks nextButton
59         public void actionPerformed( ActionEvent event )
60         {
61             readRecord();
62         }
63
64     } // end anonymous inner class
65
66 ); // end call to addActionListener
67
68 // register listener for window closing event
69 addWindowListener(
70
71     // anonymous inner class to handle windowClosing event
72     new WindowAdapter() {
73
74
```

Дефинираме
действието на бутоните
от графичния
интерфейс




```
75         // close file and terminate application
76         public void windowClosing( WindowEvent event )
77         {
78             closeFile();
79         }
80
81     } // end anonymous inner class
82
83 ); // end call to addWindowListener
84
85 setSize( 300, 150 );
86 setVisible( true );
87
88 } // end constructor
89
90 // enable user to select file to open
91 private void openFile()
92 {
93     // display file dialog so user can select file
94     JFileChooser fileChooser = new JFileChooser();
95     fileChooser.setFileSelectionMode( JFileChooser.FILES_ONLY );
96
97     int result = fileChooser.showOpenDialog( this );
98
```



```
99  // if user clicked Cancel button on dialog, return
100  if ( result == JFileChooser.CANCEL_OPTION )
101      return;
102
103  // obtain selected file
104  File fileName = fileChooser.getSelectedFile();
105
106  // display error is file name invalid
107  if ( fileName == null || fileName.getName().equals( "" ) )
108      JOptionPane.showMessageDialog( this, "Invalid File Name",
109          "Invalid File Name", JOptionPane.ERROR_MESSAGE );
110
111  else {
112
113      // open file
114      try {
115          input = new RandomAccessFile( fileName, "r" );
116          nextButton.setEnabled( true );
117          openButton.setEnabled( false );
118      }
119
120      // catch exception while opening file
121      catch ( IOException ioException ) {
122          JOptionPane.showMessageDialog( this, "File does not exist",
123              "Invalid File Name", JOptionPane.ERROR_MESSAGE );
124      }
```

Отваря за **четене само**
RandomAccessFile



```
125
126     } // end else
127
128 } // end method openFile
129
130 // read one record
131 private void readRecord()
132 {
133     RandomAccessAccountRecord record = new RandomAccessAccountRecord();
134
135     // read a record and display
136     try {
137
138         do {
139             record.read( input );
140         } while ( record.getAccount() == 0 );
141
142         String values[] = { String.valueOf( record.getAccount()
143             record.getFirstName(), record.getLastName(),
144             String.valueOf( record.getBalance() ) );
145         userInterface.setFieldValues( values );
146     }
147 }
```

Чете докато има
валидни банкови
номера, пропуска
празни записи

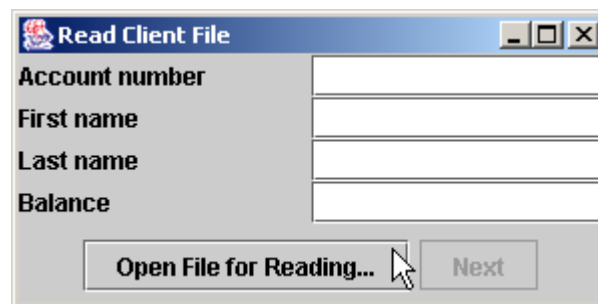


```
148 // close file when end-of-file reached
149 catch ( EOFException eofException ) {
150     JOptionPane.showMessageDialog( this, "No more records",
151         "End-of-file reached", JOptionPane.INFORMATION_MESSAGE );
152     closeFile();
153 }
154
155 // process exceptions from problem with file
156 catch ( IOException ioException ) {
157     JOptionPane.showMessageDialog( this, "Error Reading File",
158         "Error", JOptionPane.ERROR_MESSAGE );
159
160     System.exit( 1 );
161 }
162
163 } // end method readRecord
164
165 // close file and terminate application
166 private void closeFile()
167 {
168     // close file and exit
169     try {
170         if ( input != null )
171             input.close();
172
173         System.exit( 0 );
174     }
```

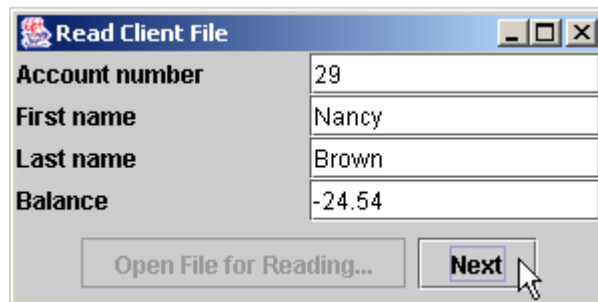
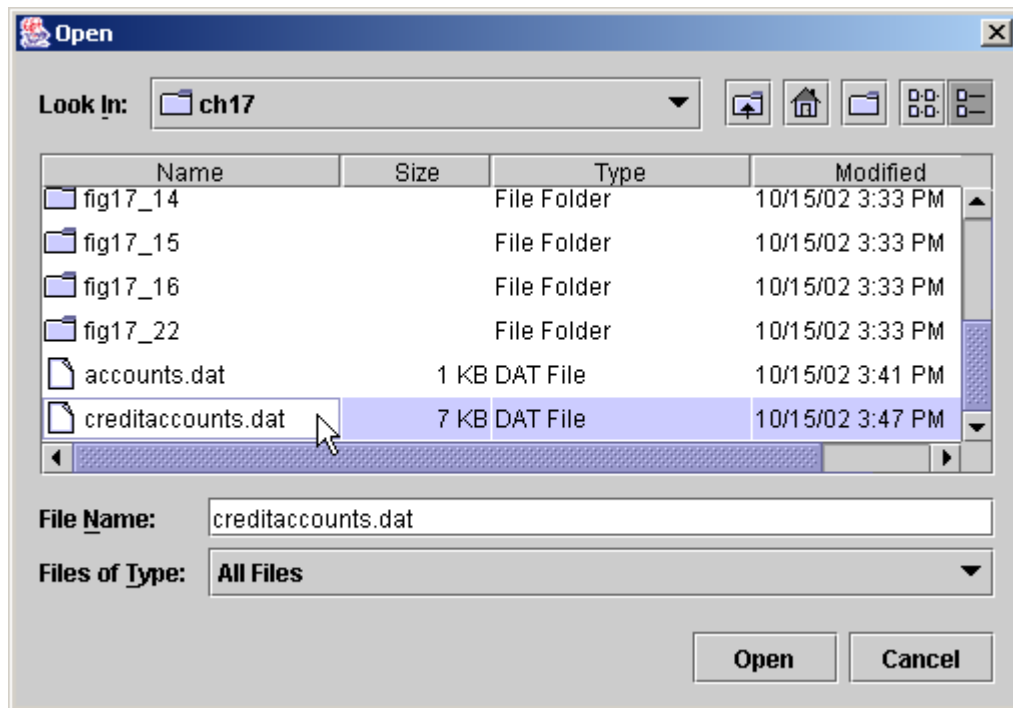
Условие за край на
файл



```
175
176 // process exception closing file
177 catch( IOException ioException ) {
178     JOptionPane.showMessageDialog( this, "Error closing file",
179         "Error", JOptionPane.ERROR_MESSAGE );
180
181     System.exit( 1 );
182 }
183
184 } // end method closeFile
185
186 public static void main( String args[] )
187 {
188     new ReadRandomFile();
189 }
190
191 } // end class ReadRandomFile
```



ReadRandomFile.
java



7.3 Пример- обработка на транзакции

- Substantial transaction-processing system
 - Използва файл с произволен достъп за съхранение и обработка на банкови сметки (извежда съществуващи, променя, добавя и изтрива)



Fig. 7.3 Графичен интерфейс

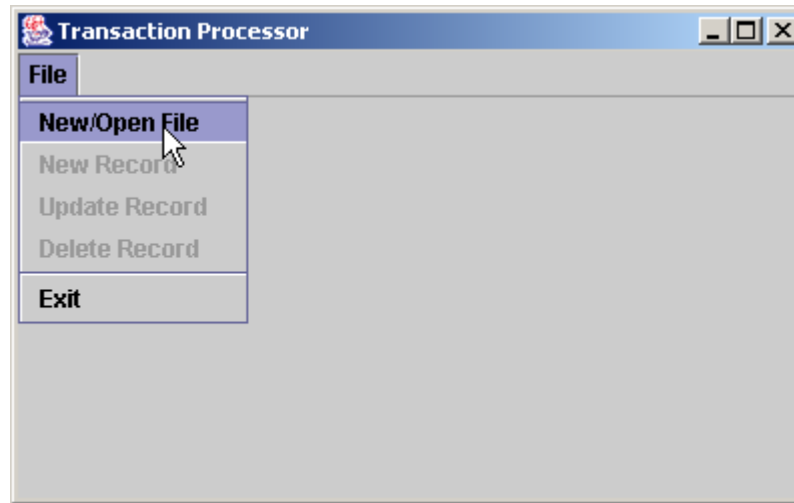


Fig. 7.4 Промяна на запис: зарежда запис за промяна

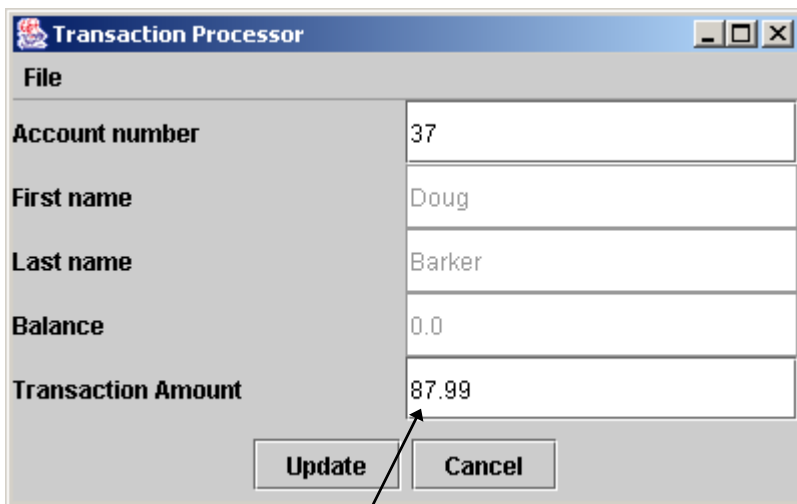
Въвежда номер на сметка и
натиска Enter key за
показване на полетата на
сметката

The window titled "Transaction Processor" has a menu bar with "File". Below it are five input fields: "Account number" (containing "37"), "First name", "Last name", "Balance", and "Transaction Amount". At the bottom are "Update" and "Cancel" buttons.

Етикетът на бутонът е
променен в съответствие с
изпълняваното действие.

The window titled "Transaction Processor" has a menu bar with "File". Below it are five input fields: "Account number" (containing "37"), "First name" (containing "Doug"), "Last name" (containing "Barker"), "Balance" (containing "0.0"), and "Transaction Amount" (containing "Charge(+) or payment (-)"). At the bottom are "Update" and "Cancel" buttons.

Fig. 7.5 Промяна на запис: Въвеждане на транзакция



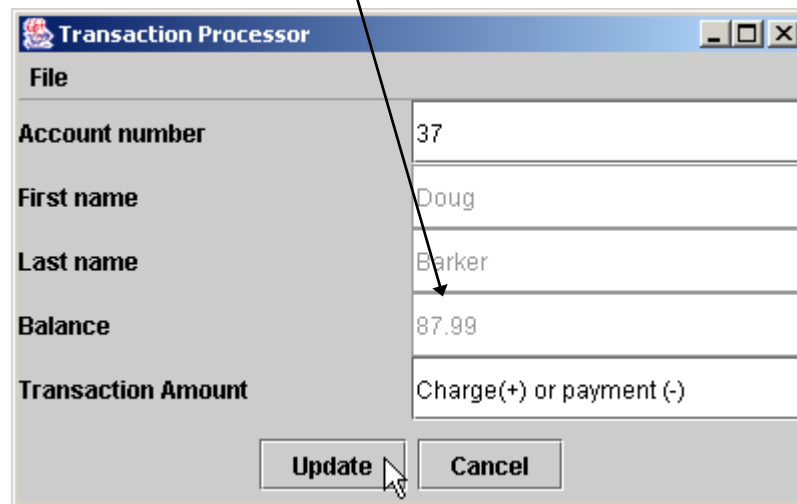
The screenshot shows a window titled "Transaction Processor" with a menu bar containing "File". Below the menu bar are five text input fields: "Account number" (37), "First name" (Doug), "Last name" (Barker), "Balance" (0.0), and "Transaction Amount" (87.99). At the bottom are "Update" and "Cancel" buttons. An arrow points from the text "Въвеждане на нова сума за транзакция." to the "Transaction Amount" field.

File	
Account number	37
First name	Doug
Last name	Barker
Balance	0.0
Transaction Amount	87.99

Update Cancel

Въвеждане на нова сума за транзакция.

След промяната.

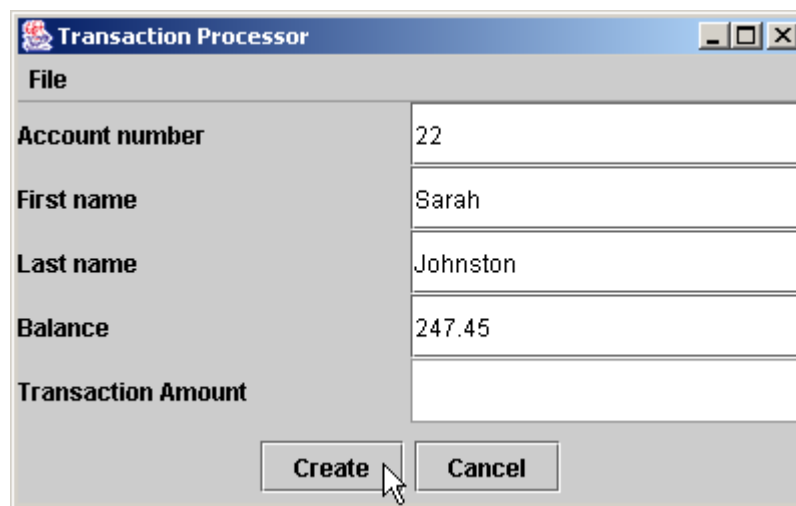


The screenshot shows the same "Transaction Processor" window after an update. The "Balance" field now contains "87.99" and the "Transaction Amount" field contains the text "Charge(+) or payment (-)". An arrow points from the text "След промяната." to the "Balance" field.

File	
Account number	37
First name	Doug
Last name	Barker
Balance	87.99
Transaction Amount	Charge(+) or payment (-)

Update Cancel

Fig. 7.4 Нов запис: Добавяне на запис към файла

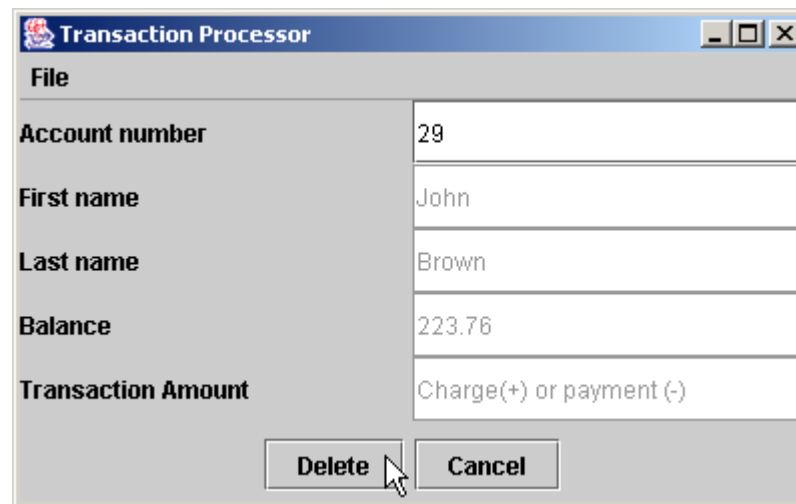


A screenshot of a Java Swing dialog box titled "Transaction Processor". The dialog has a standard Windows-style title bar with a minimize, maximize, and close button. Below the title bar is a menu bar with a single item "File". The main area of the dialog contains five text input fields, each preceded by a label: "Account number" (containing "22"), "First name" (containing "Sarah"), "Last name" (containing "Johnston"), "Balance" (containing "247.45"), and "Transaction Amount" (which is empty). At the bottom of the dialog are two buttons: "Create" and "Cancel". A mouse cursor is pointing at the "Create" button.

File	
Account number	22
First name	Sarah
Last name	Johnston
Balance	247.45
Transaction Amount	

Create Cancel

Fig. 7.4 Изтриване на запис



A screenshot of a Java Swing dialog box titled "Transaction Processor". The dialog has a standard Windows-style title bar with a blue gradient and window control buttons (minimize, maximize, close). Below the title bar is a menu bar with a single item "File". The main area of the dialog contains five text input fields, each with a label to its left: "Account number" (containing "29"), "First name" (containing "John"), "Last name" (containing "Brown"), "Balance" (containing "223.76"), and "Transaction Amount" (containing "Charge(+) or payment (-)"). At the bottom of the dialog are two buttons: "Delete" and "Cancel". A mouse cursor is pointing at the "Delete" button.

File	
Account number	29
First name	John
Last name	Brown
Balance	223.76
Transaction Amount	Charge(+) or payment (-)

Buttons: Delete, Cancel

7.3 Пример- обработка на транзакции

- Операциите по обработка на файла с произволен достъп изтегляме в `class FileEditor` като **част от бизнес логиката** на приложението
- Запис считаме празен, ако номерът на банковата сметка е нула
- Изтриването на запис се симулира с нулиране на номера на банковата сметка
- Добавяне на нов запис става с въвеждане на ненулев номер за банкова сметка на запис със същия пореден номер във файла



```
1 // Fig. 17.22: FileEditor.java
2 // This class declares methods that manipulate bank account
3 // records in a random access file.
4 import java.io.*;
5
6 import com.deitel.jhtp5.ch17.RandomAccessAccountRecord;
7
8 public class FileEditor {
9
10     RandomAccessFile file; // reference to the file
11
12     // open the file
13     public FileEditor( File fileName ) throws IOException
14     {
15         file = new RandomAccessFile( fileName, "rw" );
16     }
17
18     // close the file
19     public void closeFile() throws IOException
20     {
21         if ( file != null )
22             file.close();
23     }
24 }
```

Създава RandomAccessFile
обект по зададено име на файл

Затваря RandomAccessFile
обект



```
25 // get a record from the file
26 public RandomAccessAccountRecord getRecord( int accountNumber )
27     throws IllegalArgumentException, NumberFormatException, IOException
28 {
29     RandomAccessAccountRecord record = new RandomAccessAccountRecord();
30
31     if ( accountNumber < 1 || accountNumber > 100 )
32         throw new IllegalArgumentException( "Out of range" );
33
34     // seek appropriate record in file
35     file.seek( ( accountNumber - 1 ) * RandomAccessAccountRecord.SIZE );
36
37     record.read( file );
38
39     return record;
40
41 } // end method getRecord
42
43 // update record in file
44 public void updateRecord( int accountNumber, String firstName,
45     String lastName, double balance )
46     throws IllegalArgumentException, IOException
47 {
48     RandomAccessAccountRecord record = getRecord( accountNumber );
49     if ( accountNumber == 0 )
50         throw new IllegalArgumentException( "Account does not exist" );
51 }
```

Прочита цял
RandomAccessAccountRecord
запис от файла

Променя
позицията на
файл указателя

Чете запис

Променя запис



Позиционира указателя на файла

Презаписва записа с
НОВ

Създава нов запис,
ако няма въведен
номер за банкова
сметка

Позиционира
указателя на файла



```
52 // seek appropriate record in file
53 file.seek( ( accountNumber - 1 ) * RandomAccessAccountRecord.SIZE );
54
55 record = new RandomAccessAccountRecord( accountNumber,
56     firstName, lastName, balance );
57
58 record.write( file ); // write updated record to file
59
60 } // end method updateRecord
61
62 // add record to file
63 public void newRecord( int accountNumber, String firstName,
64     String lastName, double balance )
65     throws IllegalArgumentException, IOException
66 {
67     RandomAccessAccountRecord record = getRecord( accountNumber );
68
69     if ( record.getAccount() != 0 )
70         throw new IllegalArgumentException( "Account already exists" );
71
72     // seek appropriate record in file
73     file.seek( ( accountNumber - 1 ) * RandomAccessAccountRecord.SIZE );
74
75     record = new RandomAccessAccountRecord( accountNumber,
76         firstName, lastName, balance );
```



```
77     record.write( file );  
78  
79  
80 } // end method newRecord  
81  
82 // delete record from file  
83 public void deleteRecord( int accountNumber )  
84     throws IllegalArgumentException, IOException  
85 {  
86     RandomAccessAccountRecord record = getRecord( accountNumber );  
87  
88     if ( record.getAccount() == 0 )  
89         throw new IllegalArgumentException( "Account does not exist" );  
90  
91     // seek appropriate record in file  
92     file.seek( ( accountNumber - 1 ) * RandomAccessAccountRecord.SIZE );  
93  
94     // create a blank record to write to the file  
95     record = new RandomAccessAccountRecord();  
96     record.write( file );  
97  
98 } // end method deleteRecord  
99  
100 } // end class EditFile
```

Записва новия запис

Изтрива записа

Позиционира
указателя на
файлаИзтрива запис като презаписва запис с нулиран
номер на банкова сметка

TransactionProcessor.java

```
1 // Fig. 17.21: TransactionProcessor.java
2 // A transaction processing program using random-access files.
3 import java.awt.*;
4 import java.awt.event.*;
5 import java.io.*;
6 import java.text.DecimalFormat;
7 import javax.swing.*;
8
9 import com.deitel.jhtp5.ch17.*;
10
11 public class TransactionProcessor extends JFrame {
12
13     private BankUI userInterface;
14     private JMenuItem newItem, updateItem, deleteItem, openItem, exitItem;
15     private JTextField fields[];
16     private JTextField accountField, transactionField;
17     private JButton actionButton, cancelButton;
18     private FileEditor dataFile;
19     private RandomAccessAccountRecord record;
20
21     public TransactionProcessor()
22     {
23         super( "Transaction Processor" );
24     }
```

Добавяне на меню към JFrame

Class FileEditor дефинира
методи за обработка на
записи от файл с
произволен достъп



```
25 // set up desktop, menu bar and File menu
26 userInterface = new BankUI( 5 );
27 getContentPane().add( userInterface );
28 userInterface.setVisible( false );
29
30 // set up the action button
31 actionButton = userInterface.getDoTask1Button();
32 actionButton.setText( "Save Changes" );
33 actionButton.setEnabled( false );
34
35 // register action button listener
36 actionButton.addActionListener(
37
38     new ActionListener() { // anonymous inner class
39
40         public void actionPerformed((ActionEvent event) )
41         {
42             String action = event.getActionCommand();
43             performAction( action );
44
45         } // end method actionPerformed
46
47     } // end anonymous inner class
48
49 ); // end call to addActionListener
50
```

Дефинираме действие на бутон в съответствие с името на бутона



```
51 // set up the cancel button
52 cancelButton = userInterface.getDoTask2Button();
53 cancelButton.setText( "Cancel" );
54 cancelButton.setEnabled( false );
55
56 // register cancel button listener
57 cancelButton.addActionListener(
58
59     new ActionListener() { // anonymous inner class
60
61         // clear the fields
62         public void actionPerformed((ActionEvent event)
63         {
64             userInterface.clearFields();
65         }
66
67     } // end anonymous inner class
68
69 ); // end call to addActionListener
70
71 // set up the listener for the account field
72 fields = userInterface.getFields();
73 accountField = fields[ BankUI.ACCOUNT ];
74 accountField.addActionListener(
75
```

Дефинираме действието на **Cancel**
бутона



```
76     new ActionListener() { // anonymous inner class
77
78         public void actionPerformed((ActionEvent event) )
79         {
80             displayRecord( "0" );
81         }
82
83     } // end anonymous inner class
84
85 ); // end call to addActionListener
86
87 // create reference to the transaction field
88 transactionField = fields[ BankUI.TRANSACTION ];
89
90 // register transaction field listener
91 transactionField.addActionListener(
92
93     new ActionListener() { // anonymous inner class
94
95         // update the GUI fields
96         public void actionPerformed((ActionEvent event) )
97         {
98             displayRecord( transactionField.getText() );
99         }
100
101     } // end anonymous inner class
102
```



```
103 ); // end call to addActionListener
104
105 JMenuBar menuBar = new JMenuBar(); // set up the menu
106 setJMenuBar( menuBar );
107
108 JMenu fileMenu = new JMenu( "File" );
109 menuBar.add( fileMenu );
110
111 // set up menu item for adding a record
112 newItem = new JMenuItem( "New Record" );
113 newItem.setEnabled( false );
114
115 // register new item listener
116 newItem.addActionListener(
117
118     new ActionListener() { // anonymous inner class
119
120         public void actionPerformed((ActionEvent event) )
121         {
122
123             // set up the GUI fields for editing
124             fields[ BankUI.ACCOUNT ].setEnabled( true );
125             fields[ BankUI.FIRSTNAME ].setEnabled( true );
126             fields[ BankUI.LASTNAME ].setEnabled( true );
127             fields[ BankUI.BALANCE ].setEnabled( true );
128             fields[ BankUI.TRANSACTION ].setEnabled( false );
```

Дефиниране на меню и добавяне
на опции към менюто

Дефинираме действието на
ОПЦИТЕ ОТ МЕНЮТО



```
129         actionPerformed( true );
130         actionPerformed( "Create" );
131         cancelButton.setEnabled( true );
132
133         userInterface.clearFields(); // reset the textfields
134
135     } // end method actionPerformed
136
137     } // end anonymous inner class
138
139 ); // end call to addActionListener
140
141 // set up menu item for updating a record
142 updateItem = new JMenuItem( "Update Record" );
143 updateItem.setEnabled( false );
144
145 // register update item listener
146 updateItem.addActionListener(
147
148     new ActionListener() { // anonymous inner class
149
150         public void actionPerformed((ActionEvent event) )
151         {
152
```



```
153         // set up the GUI fields for editing
154         fields[ BankUI.ACCOUNT ].setEnabled( true );
155         fields[ BankUI.FIRSTNAME ].setEnabled( false );
156         fields[ BankUI.LASTNAME ].setEnabled( false );
157         fields[ BankUI.BALANCE ].setEnabled( false );
158         fields[ BankUI.TRANSACTION ].setEnabled( true );
159
160         actionButton.setEnabled( true );
161         actionButton.setText( "Update" );
162         cancelButton.setEnabled( true );
163
164         userInterface.clearFields(); // reset the textfields
165
166     } // end method actionPerformed
167
168 } // end anonymous inner class
169
170 ); // end call to addActionListener
171
172 // set up menu item for deleting a record
173 deleteItem = new JMenuItem( "Delete Record" );
174 deleteItem.setEnabled( false );
175
176 // register delete item listener
177 deleteItem.addActionListener(
178
```




```
179 new ActionListener() { // anonymous inner class
180
181     public void actionPerformed((ActionEvent event)
182     {
183         // set up the GUI fields for editing
184         fields[ BankUI.ACCOUNT ].setEnabled( true );
185         fields[ BankUI.FIRSTNAME ].setEnabled( false );
186         fields[ BankUI.LASTNAME ].setEnabled( false );
187         fields[ BankUI.BALANCE ].setEnabled( false );
188         fields[ BankUI.TRANSACTION ].setEnabled( false );
189
190         actionButton.setEnabled( true );
191         actionButton.setText( "Delete" );
192         cancelButton.setEnabled( true );
193
194         userInterface.clearFields(); // reset the textfields
195
196     } // end method actionPerformed
197
198 } // end anonymous inner class
199
200 ); // end call to addActionListener
201
202 // set up menu item for opening file
203 openItem = new JMenuItem( "New/Open File" );
204
```



```
205 // register open item listener
206 openItem.addActionListener(
207
208     new ActionListener() { // anonymous inner class
209
210         public void actionPerformed((ActionEvent event)
211         {
212             // try to open the file
213             if ( !openFile() )
214                 return;
215
216             // set up the menu items
217             newItem.setEnabled( true );
218             updateItem.setEnabled( true );
219             deleteItem.setEnabled( true );
220             openItem.setEnabled( false );
221
222             // set the interface
223             userInterface.setVisible( true );
224             fields[ BankUI.ACCOUNT ].setEnabled( false );
225             fields[ BankUI.FIRSTNAME ].setEnabled( false );
226             fields[ BankUI.LASTNAME ].setEnabled( false );
227             fields[ BankUI.BALANCE ].setEnabled( false );
228             fields[ BankUI.TRANSACTION ].setEnabled( false );
229
230         } // end method actionPerformed
231
```



```
232     } // end anonymous inner class
233
234 ); // end call to addActionListener
235
236 // set up menu item for exiting program
237 exitItem = new JMenuItem( "Exit" );
238
239 // register exit item listener
240 exitItem.addActionListener(
241
242     new ActionListener() { // anyomus inner class
243
244         public void actionPerformed((ActionEvent event) )
245         {
246             try {
247                 dataFile.closeFile(); // close the file
248             }
249
250             catch ( IOException ioException ) {
251                 JOptionPane.showMessageDialog(
252                     TransactionProcessor.this, "Error closing file",
253                     "IO Error", JOptionPane.ERROR_MESSAGE );
254             }
255
256             finally {
257                 System.exit( 0 ); // exit the program
258             }
259         }
260     }
261 }
```



```
259         } // end method actionPerformed
260
261     } // end anonymous inner class
262
263 ); // end call to addActionListener
264
265 // attach menu items to File menu
266 fileMenu.add( openItem );
267 fileMenu.add( newItem );
268 fileMenu.add( updateItem );
269 fileMenu.add( deleteItem );
270 fileMenu.addSeparator();
271 fileMenu.add( exitItem );
272
273
274 setSize( 400, 250 );
275 setVisible( true );
276
277 } // end constructor
278
279 public static void main( String args[] )
280 {
281     new TransactionProcessor();
282 }
283
```



```
284 // get the file name and open the file
285 private boolean openFile()
286 {
287     // display dialog so user can select file
288     JFileChooser fileChooser = new JFileChooser();
289     fileChooser.setFileSelectionMode( JFileChooser.FILES_ONLY );
290
291     int result = fileChooser.showOpenDialog( this );
292
293     // if user clicked Cancel button on dialog, return
294     if ( result == JFileChooser.CANCEL_OPTION )
295         return false;
296
297     // obtain selected file
298     File fileName = fileChooser.getSelectedFile();
299
300     // display error if file name invalid
301     if ( fileName == null || fileName.getName().equals( "" ) ) {
302         JOptionPane.showMessageDialog( this, "Invalid File Name",
303             "Bad File Name", JOptionPane.ERROR_MESSAGE );
304         return false;
305     }
306
307     try {
308         // call the helper method to open the file
309         dataFile = new FileEditor( fileName );
310     }
```

Създаване на FileEditor обект
по името на файла



```
311 catch( IOException ioException ) {
312     JOptionPane.showMessageDialog( this, "Error Opening File",
313         "IO Error", JOptionPane.ERROR_MESSAGE );
314     return false;
315 }
316
317 return true;
318
319 } // end method openFile
320
321 // create, update or delete the record
322 private void performAction( String action )
323 {
324     try {
325         // get the textfield values
326         String[] values = userInterface.getFieldValues();
327
328         int accountNumber = Integer.parseInt( values[ BankUI.ACCOUNT ] );
329         String firstName = values[ BankUI.FIRSTNAME ];
330         String lastName = values[ BankUI.LASTNAME ];
331         double balance = Double.parseDouble( values[ BankUI.BALANCE ] );
332
333         if ( action.equals( "Create" ) )
334             dataFile.newRecord( accountNumber, // create a new record
335                 firstName, lastName, balance );
336     }
337 }
```

Изпълнява се при
натискане на първия
бутон- действието
се определя от
текста изписан
върху бутона

Създава нов запис



TransactionProcessor
.java

Променя запис

Изтрива запис

```
338     else if ( action.equals( "Update" ) )
339         dataFile.updateRecord( accountNumber, // update record
340             firstName, lastName, balance );
341
342     else if ( action.equals( "Delete" ) )
343         dataFile.deleteRecord( accountNumber ); // delete record
344
345     else
346         JOptionPane.showMessageDialog( this, "Invalid Action",
347             "Error executing action", JOptionPane.ERROR_MESSAGE );
348
349 } // end try
350
351 catch( NumberFormatException format ) {
352     JOptionPane.showMessageDialog( this, "Bad Input",
353         "Number Format Error", JOptionPane.ERROR_MESSAGE );
354 }
355
356 catch( IllegalArgumentException badAccount ) {
357     JOptionPane.showMessageDialog( this, badAccount.getMessage(),
358         "Bad Account Number", JOptionPane.ERROR_MESSAGE );
359 }
360
361 catch( IOException ioException ) {
362     JOptionPane.showMessageDialog( this, "Error writing to the file",
363         "IO Error", JOptionPane.ERROR_MESSAGE );
364 }
```



```
365 } // end method performAction
366
367
368 // input a record in the textfields and update the balance
369 private void displayRecord( String transaction )
370 {
371     try {
372         // get the account number
373         int accountNumber = Integer.parseInt(
374             userInterface.getFieldValues()[ BankUI.ACCOUNT ] );
375
376         // get the associated record
377         RandomAccessAccountRecord record =
378             dataFile.getRecord( accountNumber );
379
380         if ( record.getAccount() == 0 )
381             JOptionPane.showMessageDialog( this, "Record does not exist",
382                 "Bad Account Number", JOptionPane.ERROR_MESSAGE );
383
384         // get the transaction
385         double change = Double.parseDouble( transaction );
386
387         // create a string array to send to the textfields
388         String[] values = { String.valueOf( record.getAccount() ),
389             record.getFirstName(), record.getLastName(),
390             String.valueOf( record.getBalance() + change ),
391             "Charge(+) or payment (-)" };
392     }
```

Извежда запис в
текстовите полета
на графичния
интерфейс




```
392         userInterface.setFieldValues( values );
393
394     } // end try
395
396     catch( NumberFormatException format ) {
397         JOptionPane.showMessageDialog( this, "Bad Input",
398             "Number Format Error", JOptionPane.ERROR_MESSAGE );
399     }
400
401     catch ( IllegalArgumentException badAccount ) {
402         JOptionPane.showMessageDialog( this, badAccount.getMessage(),
403             "Bad Account Number", JOptionPane.ERROR_MESSAGE );
404     }
405
406     catch( IOException ioException ) {
407         JOptionPane.showMessageDialog( this, "Error reading the file",
408             "IO Error", JOptionPane.ERROR_MESSAGE );
409     }
410
411 } // end method displayRecord
412
413 } // end class TransactionProcessor
```



Задачи

Задача 1.

Напишете приложението за обработка на транзакции във вариант да използва текстов интерфейс със стандартен вход и изход.

Задача 2.

Напишете приложение, което обработва транзакции посредством сериализация на данните.

Упътване: Сериализирайте обекта си до масив от байтове и после ги запазете в `RandomAccessFile`.