```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ExpenseTracker tracker = new ExpenseTracker
();


        DatabaseManager.createTables();

        while (true) {
            System.out.println("1. Add Expense\n2.
View Expenses\n3. Total Expenses\n4. Delete Expense\n
5. Save Expenses to File\n6. Load Expense from File
File\n7. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();

            switch (choice) {

                case 1:
                    System.out.print("Enter expense
description: ");
                    scanner.nextLine();
                    String description = scanner.
nextLine();
                    System.out.print("Enter expense
amount: ");
                    double amount = scanner.
nextDouble();
                    tracker.addExpense(new Expense(
description, amount));
                    break;

                case 2:
                    tracker.displayExpenses();
                    break;
                case 3:
                    System.out.println("Total
Expenses: Rs" + tracker.getTotalExpenses());
```

```
32                      break;
33                  case 4:
34                      System.out.print("Enter index of
   expense to delete: ");
35                      int index = scanner.nextInt();
36                      tracker.deleteExpense(index);
37                      break;
38                  case 5:
39                      System.out.println("Enter file
   name to save expenses: ");
40                      String saveFileName = scanner.
   next();
41                      tracker.saveExpensesToFile(
   saveFileName);
42                      break;
43                  case 6:
44                      System.out.println("Enter file
   name to load expenses: ");
45                      String loadFileName = scanner.
   next();
46                      tracker.loadExpensesFromFile(
   loadFileName);
47                      break;
48                  case 7:
49                      System.out.println("Exiting...");
50                      System.exit(0);
51                  default:
52                      System.out.println("Invalid
   choice");
53              }
54          }
55      }
56 }
57
```

```java
import java.io.Serializable;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

class Expense implements Serializable {
    private String description;
    private double amount;
    private LocalDateTime dateTime;


    private static final DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

    public Expense(String description, double amount) {
        this.description = description;
        this.amount = amount;
        this.dateTime = LocalDateTime.now(); // Set current system time
    }

    public String getDescription() {
        return description;
    }

    public double getAmount() {
        return amount;
    }

    public LocalDateTime getDateTime() {
        return dateTime;
    }

    @Override
    public String toString() {
        return "Expense{" +
                "description='" + description + '\'' +
                ", amount= Rs" + amount +
                ", dateTime=" + dateTime.format(
```

```
36 formatter) + // Format the LocalDateTime
37                     '}';
38     }
39 }
40
```

```java
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

    public class DailyExpense {
    private Date date;
    private List<Double> expenses;

    public DailyExpense(Date date) {
        this.date = date;
        this.expenses = new ArrayList<>();
    }

    public Date getDate() {
        return date;
    }

    public List<Double> getExpenses() {
        return expenses;
    }

    public void addExpense(double expense) {
        expenses.add(expense);
    }
}

```

```
 1
 2 public interface ExpenseManager {
 3     void addExpense(Expense expense);
 4
 5     void deleteExpense(int index);
 6
 7     double getTotalExpenses();
 8
 9     void displayExpenses();
10
11     void saveExpensesToFile(String fileName);
12
13     void loadExpensesFromFile(String fileName);
14 }
15
16
```

```java
import java.util.ArrayList;
import java.io.*;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class ExpenseTracker implements ExpenseManager {
    private ArrayList<Expense> expenses;

    public ExpenseTracker() {
        expenses = new ArrayList<>();
    }

    public void addExpense(Expense expense) {
        expenses.add(expense);
    }

    public void deleteExpense(int index) {
        if (index >= 0 && index < expenses.size()) {
            expenses.remove(index);
            System.out.println("Expense deleted successfully.");
        } else {
            System.out.println("Invalid expense index.");
        }
    }

    public double getTotalExpenses() {
        double total = 0;
        for (Expense expense : expenses) {
            total += expense.getAmount();
        }
        return total;
    }

    public void displayExpenses() {
        System.out.println("Expenses:");
        for (Expense expense : expenses) {
            System.out.println(expense);
```

```
39          }
40      }
41      //  public void saveExpensesToFile(String
    fileName) {
42      //      try (ObjectOutputStream oos = new
    ObjectOutputStream(new FileOutputStream(fileName))) {
43      //          oos.writeObject(expenses);
44      //        System.out.println("Expenses saved to
    file successfully.");
45      //      } catch (IOException e) {
46      //          System.out.println("Error saving
    expenses to file: " + e.getMessage());
47      //      }
48      // }
49
50
51      public void saveExpensesToFile(String fileName) {
52          try (BufferedWriter writer = new
    BufferedWriter(new FileWriter(fileName))) {
53              for (Expense expense : expenses) {
54                  writer.write(expense.getDescription
    () + "," + expense.getAmount());
55                  writer.newLine();
56              }
57              System.out.println("Expenses saved to
    file successfully.");
58          } catch (IOException e) {
59              System.out.println("Error saving expenses
     to file: " + e.getMessage());
60          }
61      }
62
63      @Override
64      public void loadExpensesFromFile(String fileName
    ) {
65          try (ObjectInputStream ois = new
    ObjectInputStream(new FileInputStream(fileName))) {
66              expenses = (ArrayList<Expense>) ois.
    readObject();
67              System.out.println("Expenses loaded from
    file successfully.");
```

```
68          } catch (IOException |
    ClassNotFoundException e) {
69              System.out.println("Error loading
    expenses from file: " + e.getMessage());
70          }
71      }
72  }
73
```

```
 1 import java.sql.*;
 2
 3 public class DatabaseManager {
 4     static final String DB_URL = "jdbc:mysql://
   localhost/java_rud";
 5     static final String USER = "java_rud";
 6     static final String PASS = "@qwertyuiop1407";
 7
 8     public static Connection getConnection() throws
   SQLException {
 9         return DriverManager.getConnection(DB_URL,
   USER, PASS);
10     }
11
12     public static void createTables() {
13         try (Connection conn = getConnection();
14              Statement stmt = conn.createStatement
   ()) {
15
16             String createExpenseTableSQL = "CREATE
   TABLE IF NOT EXISTS Expense (" +
17                     "id INT AUTO_INCREMENT PRIMARY
   KEY," +
18                     "description VARCHAR(255) NOT
   NULL," +
19                     "amount DECIMAL(10, 2) NOT NULL"
    +
20                     ")";
21             stmt.executeUpdate(createExpenseTableSQL
   );
22             System.out.println("Expense table created
    successfully");
23
24             String createAnotherTableSQL = "CREATE
   TABLE IF NOT EXISTS AnotherTable (" +
25                     "id INT AUTO_INCREMENT PRIMARY
   KEY," +
26                     "expense_id INT," +
27                     "FOREIGN KEY (expense_id)
   REFERENCES Expense(id)" +
28                     ")";
```

```
29              stmt.executeUpdate(createAnotherTableSQL
    );
30              System.out.println("AnotherTable created
    successfully");
31          } catch (SQLException e) {
32              e.printStackTrace();
33          }
34      }
35  }
36
```

```
 1 import javax.swing.*;
 2 import javax.swing.border.EmptyBorder;
 3 import java.awt.*;
 4 import java.io.*;
 5 import java.text.ParseException;
 6 import java.text.SimpleDateFormat;
 7 import java.util.ArrayList;
 8 import java.util.Date;
 9 import java.util.List;
10
11 public class ExpenseTrackerGUI extends JFrame {
12
13
14     private JTextField expenseField;
15     private JTextField descriptionField;
16     private JTextArea expenseList;
17     private JTextField totalExpenseField;
18     private JTextField totalMonthlyExpenseField;
19     private JTextField dailyExpenseField;
20     private JTextArea dailyExpenseList;
21     private List<ExpenseEntry> expenses;
22     private List<Double> dailyExpenses;
23
24     public ExpenseTrackerGUI() {
25         setTitle("Expense Tracker");
26         setSize(600, 400);
27         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE
   );
28         setLocationRelativeTo(null);
29
30         expenses = new ArrayList<>();
31         dailyExpenses = new ArrayList<>();
32
33         JLabel titleLabel = new JLabel("Expense
   Tracker");
34         titleLabel.setFont(new Font("Cambria", Font.
   BOLD, 30));
35         titleLabel.setForeground(Color.BLACK);
36
37         JPanel titlePanel = new JPanel(new FlowLayout
   (FlowLayout.CENTER));
```

```
38          titlePanel.setBackground(Color.WHITE);
39          titlePanel.setBorder(new EmptyBorder(10, 10,
    10, 10));
40          titlePanel.add(titleLabel);
41
42          JPanel topPanel = new JPanel(new BorderLayout
    ());
43          topPanel.setBackground(Color.WHITE);
44          topPanel.setBorder(new EmptyBorder(0, 10, 10
    , 10));
45          JPanel expenseInputPanel = new JPanel(new
    FlowLayout());
46          expenseInputPanel.setBackground(Color.WHITE);
47          expenseInputPanel.setBorder(new EmptyBorder(5
    , 5, 5, 5));
48          JLabel expenseLabel = new JLabel("Enter
    Expense:");
49          expenseField = new JTextField(10);
50          JLabel descriptionLabel = new JLabel("
    Description:");
51          descriptionField = new JTextField(10);
52          JButton addButton = new JButton("Add Expense"
    );
53          JButton viewButton = new JButton("View
    Expenses");
54          JButton deleteButton = new JButton("Delete
    Expense");
55          JButton saveButton = new JButton("Save to
    File");
56          expenseInputPanel.add(expenseLabel);
57          expenseInputPanel.add(expenseField);
58          expenseInputPanel.add(descriptionLabel);
59          expenseInputPanel.add(descriptionField);
60          expenseInputPanel.add(addButton);
61          expenseInputPanel.add(viewButton);
62          expenseInputPanel.add(deleteButton);
63          expenseInputPanel.add(saveButton);
64          topPanel.add(titlePanel, BorderLayout.NORTH);
65          topPanel.add(expenseInputPanel, BorderLayout.
    CENTER);
66
```

```
67          JPanel middlePanel = new JPanel(new
    BorderLayout());
68          middlePanel.setBackground(Color.WHITE);
69          JPanel expenseListPanel = new JPanel(new
    BorderLayout());
70          expenseListPanel.setBackground(Color.WHITE);
71          expenseListPanel.setBorder(new EmptyBorder(5
    , 5, 5, 5));
72          JLabel expenseListLabel = new JLabel("
    Expense List");
73          expenseList = new JTextArea(10, 40);
74          expenseList.setFont(new Font("Cambria", Font
    .PLAIN, 40));
75          JScrollPane scrollPane = new JScrollPane(
    expenseList);
76          expenseListPanel.add(expenseListLabel,
    BorderLayout.NORTH);
77          expenseListPanel.add(scrollPane,
    BorderLayout.CENTER);
78          middlePanel.add(expenseListPanel,
    BorderLayout.CENTER);
79
80          JPanel bottomPanel = new JPanel(new
    BorderLayout());
81          bottomPanel.setBackground(Color.WHITE);
82          JPanel totalExpensePanel = new JPanel(new
    FlowLayout());
83          totalExpensePanel.setBackground(Color.WHITE
    );
84          totalExpensePanel.setBorder(new EmptyBorder(
    5, 5, 5, 5));
85          JLabel totalExpenseLabel = new JLabel("Total
     Expense:");
86          totalExpenseLabel.setForeground(Color.BLACK
    );
87          totalExpenseField = new JTextField(10);
88          totalExpenseField.setEditable(false);
89          JButton calculateButton = new JButton("
    Calculate Total");
90          totalExpensePanel.add(totalExpenseLabel);
91          totalExpensePanel.add(totalExpenseField);
```

```
 92          totalExpensePanel.add(calculateButton);
 93          bottomPanel.add(totalExpensePanel,
    BorderLayout.NORTH);
 94
 95          // Panel for total monthly expense
 96          JPanel monthlyExpensePanel = new JPanel(new
    FlowLayout());
 97          monthlyExpensePanel.setBackground(Color.
    WHITE);
 98          monthlyExpensePanel.setBorder(new
    EmptyBorder(5, 5, 5, 5));
 99          JLabel totalMonthlyExpenseLabel = new JLabel
    ("Total Monthly Expense:");
100          totalMonthlyExpenseLabel.setForeground(Color
    .BLACK);
101          totalMonthlyExpenseField = new JTextField(10
    );
102          totalMonthlyExpenseField.setEditable(false);
103          JButton calculateMonthlyButton = new JButton
    ("Calculate Monthly Total");
104          monthlyExpensePanel.add(
    totalMonthlyExpenseLabel);
105          monthlyExpensePanel.add(
    totalMonthlyExpenseField);
106          monthlyExpensePanel.add(
    calculateMonthlyButton);
107          bottomPanel.add(monthlyExpensePanel,
    BorderLayout.CENTER);
108
109          // Panel for managing daily expenses
110          JPanel dailyExpensePanel = new JPanel(new
    BorderLayout());
111          dailyExpensePanel.setBackground(Color.WHITE
    );
112          dailyExpensePanel.setBorder(new EmptyBorder(
    5, 5, 5, 5));
113          JLabel dailyExpenseLabel = new JLabel("Enter
     Daily Expense:");
114          dailyExpenseField = new JTextField(10);
115          dailyExpenseList = new JTextArea(10, 20);
116          dailyExpenseList.setEditable(false);
```

```
117         JScrollPane dailyScrollPane = new
    JScrollPane(dailyExpenseList);
118         JButton addDailyButton = new JButton("Add
    Daily Expense");
119 
120         JButton viewDailyButton = new JButton("View
    Daily Expenses");
121         JPanel dailyInputPanel = new JPanel(new
    FlowLayout());
122         dailyInputPanel.add(dailyExpenseLabel);
123         dailyInputPanel.add(dailyExpenseField);
124         dailyInputPanel.add(addDailyButton);
125         dailyInputPanel.add(viewDailyButton);
126         dailyExpensePanel.add(dailyInputPanel,
    BorderLayout.NORTH);
127         dailyExpensePanel.add(dailyScrollPane,
    BorderLayout.CENTER);
128         bottomPanel.add(dailyExpensePanel,
    BorderLayout.SOUTH);
129 
130         setLayout(new BorderLayout());
131         add(topPanel, BorderLayout.NORTH);
132         add(middlePanel, BorderLayout.CENTER);
133         add(bottomPanel, BorderLayout.SOUTH);
134 
135         addButton.addActionListener(e -> {
136             try {
137                 addExpense();
138             } catch (InvalidExpenseAmountException
     | InvalidDescriptionException ex) {
139                 JOptionPane.showMessageDialog(this,
    ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE
    );
140             }
141         });
142 
143         JButton loadButton = new JButton("Load from
    File");
144         expenseInputPanel.add(loadButton);
145 
146         viewButton.addActionListener(e ->
```

```
146 viewExpenses());
147         deleteButton.addActionListener(e ->
    deleteExpense());
148         saveButton.addActionListener(e -> saveToFile
    ());
149         calculateButton.addActionListener(e ->
    calculateTotalExpense());
150         calculateMonthlyButton.addActionListener(e
     -> calculateTotalMonthlyExpense());
151         addDailyButton.addActionListener(e ->
    addDailyExpense());
152         viewDailyButton.addActionListener(e ->
    viewDailyExpenses());
153         loadButton.addActionListener(e ->
    loadFromFile());
154     }
155
156     private void addExpense() throws
    InvalidExpenseAmountException,
    InvalidDescriptionException {
157         String expenseStr = expenseField.getText().
    trim();
158         String description = descriptionField.
    getText().trim();
159         if (!expenseStr.isEmpty()) {
160             try {
161                 double expenseAmount = Double.
    parseDouble(expenseStr);
162                 if (expenseAmount <= 0) {
163                     throw new
    InvalidExpenseAmountException("Expense amount must
    be a positive number.");
164                 }
165
166                 if (!validateDescription(description
    )) {
167                     throw new
    InvalidDescriptionException("Description should not
    contain numerical values.");
168                 }
169
```

```
170                 Date datetime = new Date();
171                 ExpenseEntry entry = new
    ExpenseEntry(expenseAmount, description, datetime);
172                 expenses.add(entry);
173                 updateExpenseList();
174                 expenseField.setText("");
175                 descriptionField.setText("");
176             } catch (NumberFormatException ex) {
177                 throw new
    InvalidExpenseAmountException("Invalid expense
    amount format.");
178             }
179         } else {
180             throw new InvalidExpenseAmountException(
    "Please enter an expense amount.");
181         }
182     }
183 
184     private boolean validateDescription(String
    description) {
185         return !description.matches(".*\\d.*");
186     }
187 
188     private void viewExpenses() {
189         updateExpenseList();
190     }
191 
192     private void updateExpenseList() {
193         expenseList.setText("");
194         SimpleDateFormat dateFormat = new
    SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
195         Font font = new Font("Cambria", Font.PLAIN,
    20); // Define the font
196         expenseList.setFont(font); // Set the font
    for the expense list
197         for (ExpenseEntry entry : expenses) {
198             String dateTimeStr = dateFormat.format(
    entry.getDateTime());
199             expenseList.append("Date-Time: " +
    dateTimeStr + ", Rs:" + entry.getAmount() + ",
    Description: " + entry.getDescription() + "\n");
```

```
200         }
201     }
202 
203 
204     private void deleteExpense() {
205         String indexStr = JOptionPane.
    showInputDialog(this, "Enter index of expense to
    delete:");
206         if (indexStr != null && !indexStr.isEmpty
    ()) {
207             try {
208                 int index = Integer.parseInt(
    indexStr);
209                 if (index >= 0 && index < expenses.
    size()) {
210                     expenses.remove(index);
211                     updateExpenseList();
212                 } else {
213                     JOptionPane.showMessageDialog(
    this, "Invalid index.", "Error", JOptionPane.
    ERROR_MESSAGE);
214                 }
215             } catch (NumberFormatException ex) {
216                 JOptionPane.showMessageDialog(this,
    "Invalid index.", "Error", JOptionPane.ERROR_MESSAGE
    );
217             }
218         }
219     }
220 
221     private void saveToFile() {
222         JFileChooser fileChooser = new JFileChooser
    ();
223         fileChooser.setDialogTitle("Save Expenses");
224         fileChooser.setCurrentDirectory(new File(
    System.getProperty("user.home") + File.separator + "
    Downloads"));
225 
226         int userSelection = fileChooser.
    showSaveDialog(this);
227         if (userSelection == JFileChooser.
```

```
227 APPROVE_OPTION) {
228             File fileToSave = fileChooser.
    getSelectedFile();
229             String filePath = fileToSave.
    getAbsolutePath();
230 
231             try {
232                 FileWriter writer = new FileWriter(
    filePath);
233                 SimpleDateFormat dateFormat = new
    SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
234                 for (ExpenseEntry entry : expenses
    ) {
235                     String dateTimeStr = dateFormat.
    format(entry.getDateTime());
236                     writer.write(entry.getAmount
    () + "," + entry.getDescription() + "," +
    dateTimeStr + "\n");
237                 }
238                 writer.close();
239                 JOptionPane.showMessageDialog(this,
    "Expenses saved to file successfully: " + fileToSave
    .getName(), "Success", JOptionPane.
    INFORMATION_MESSAGE);
240             } catch (IOException ex) {
241                 JOptionPane.showMessageDialog(this,
    "Error saving expenses to file.", "Error",
    JOptionPane.ERROR_MESSAGE);
242             }
243         }
244     }
245 
246 
247 
248 
249 
250     private void loadFromFile() {
251         JFileChooser fileChooser = new JFileChooser(
    new File(System.getProperty("user.home") + File.
    separator + "Downloads" + File.separator + "Java
    Project Expense"));
```

```
252         fileChooser.setDialogTitle("Load Expenses");
253 
254         int userSelection = fileChooser.
    showOpenDialog(this);
255         if (userSelection == JFileChooser.
    APPROVE_OPTION) {
256             File fileToLoad = fileChooser.
    getSelectedFile();
257             String filePath = fileToLoad.
    getAbsolutePath();
258 
259             try {
260                 BufferedReader reader = new
    BufferedReader(new FileReader(filePath));
261                 String line;
262                 SimpleDateFormat dateFormat = new
    SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
263                 while ((line = reader.readLine
    ()) != null) {
264                     String[] parts = line.split(","
    );
265                     if (parts.length == 3) {
266                         double amount = Double.
    parseDouble(parts[0]);
267                         String description = parts[1
    ];
268                         Date datetime = dateFormat.
    parse(parts[2]);
269                         ExpenseEntry entry = new
    ExpenseEntry(amount, description, datetime);
270                         expenses.add(entry);
271                     }
272                 }
273                 reader.close();
274                 updateExpenseList(); // Update the
    display after loading
275                 JOptionPane.showMessageDialog(this,
    "Expenses loaded from file successfully.", "Success"
    , JOptionPane.INFORMATION_MESSAGE);
276             } catch (IOException |
    NumberFormatException | ParseException ex) {
```

```
277                 JOptionPane.showMessageDialog(this,
    "Error loading expenses from file.", "Error",
    JOptionPane.ERROR_MESSAGE);
278             }
279         }
280     }
281 
282 
283 
284     private void calculateTotalExpense() {
285         double total = 0;
286         for (ExpenseEntry entry : expenses) {
287             total += entry.getAmount();
288         }
289         totalExpenseField.setText(String.valueOf(
    total));
290     }
291 
292     private void calculateTotalMonthlyExpense() {
293         double totalMonthlyExpense = 0;
294         java.util.Calendar cal = java.util.Calendar.
    getInstance();
295         int currentMonth = cal.get(java.util.
    Calendar.MONTH);
296         int currentYear = cal.get(java.util.Calendar
    .YEAR);
297 
298         for (ExpenseEntry entry : expenses) {
299             cal.setTime(entry.getDateTime());
300             int expenseMonth = cal.get(java.util.
    Calendar.MONTH);
301             int expenseYear = cal.get(java.util.
    Calendar.YEAR);
302 
303             if (expenseMonth == currentMonth &&
    expenseYear == currentYear) {
304                 totalMonthlyExpense += entry.
    getAmount();
305             }
306         }
307         totalMonthlyExpenseField.setText(String.
```

```
307 valueOf(totalMonthlyExpense));
308     }
309 
310     private void addDailyExpense() {
311         String expenseStr = dailyExpenseField.
    getText().trim();
312         if (!expenseStr.isEmpty()) {
313             try {
314                 double expenseAmount = Double.
    parseDouble(expenseStr);
315                 dailyExpenses.add(expenseAmount);
316                 updateDailyExpenseList();
317                 dailyExpenseField.setText("");
318             } catch (NumberFormatException ex) {
319                 JOptionPane.showMessageDialog(this,
    "Invalid expense amount.", "Error", JOptionPane.
    ERROR_MESSAGE);
320             }
321         } else {
322             JOptionPane.showMessageDialog(this, "
    Please enter a daily expense amount.", "Error",
    JOptionPane.ERROR_MESSAGE);
323         }
324     }
325 
326     private void viewDailyExpenses() {
327         updateDailyExpenseList();
328     }
329 
330     private void updateDailyExpenseList() {
331         dailyExpenseList.setText("");
332         SimpleDateFormat dateFormat = new
    SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
333         for (ExpenseEntry entry : expenses) {
334             String dateTimeStr = dateFormat.format(
    entry.getDateTime());
335             dailyExpenseList.append("Date-Time: " +
    dateTimeStr + ", Rs:" + entry.getAmount() + "\n");
336         }
337     }
338 
```

```
339     public static void main(String[] args) {
340         SwingUtilities.invokeLater(() -> new
    ExpenseTrackerGUI().setVisible(true));
341     }
342 
343     private static class ExpenseEntry {
344         private double amount;
345         private String description;
346         private Date datetime;
347 
348         public ExpenseEntry(double amount, String
    description, Date datetime) {
349             this.amount = amount;
350             this.description = description;
351             this.datetime = datetime;
352         }
353 
354         public double getAmount() {
355             return amount;
356         }
357 
358         public String getDescription() {
359             return description;
360         }
361 
362         public Date getDateTime() {
363             return datetime;
364         }
365     }
366 
367     private static class
    InvalidExpenseAmountException extends Exception {
368         public InvalidExpenseAmountException(String
    message) {
369             super(message);
370         }
371     }
372 
373     private static class InvalidDescriptionException
     extends Exception {
374         public InvalidDescriptionException(String
```

```
374 message) {
375                 super(message);
376             }
377         }
378 }
```