Coding Task 6

Service Implementation

---

6. Create class named FinanceApp with main method in app Trigger all the methods in service implementation class by user choose operation from the following menu.

1. **Add User.**
2. **Add expense.**
3. **Delete User.**
4. **Delete expense.**
5. **Update expense**

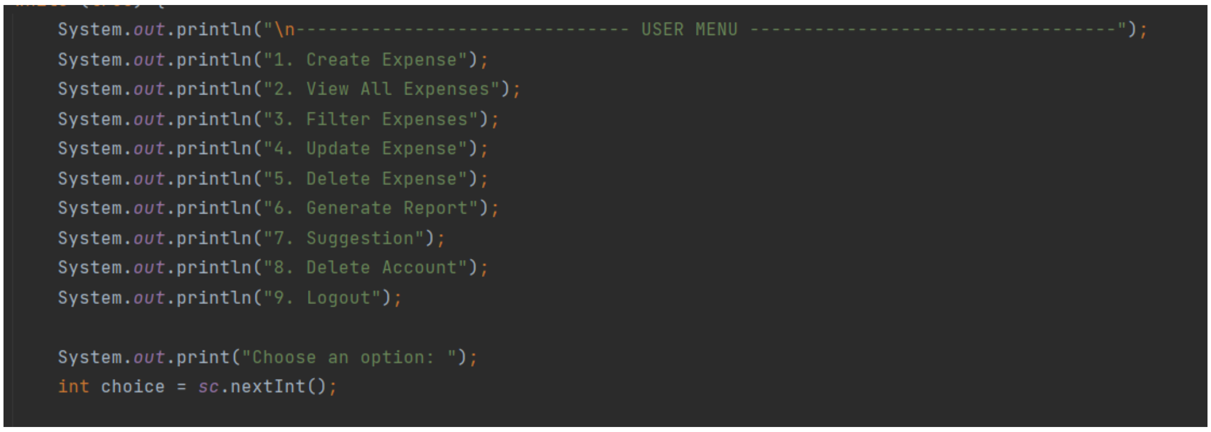
The FinanceApp class serves as the **main entry point** for the Finance Management System. It allows users to **interact** with the system via a console-based menu. This menu provides various options to perform operations such as adding users, managing expenses, viewing reports, and deleting data. All service methods are triggered based on user input, with necessary validations and exception handling in place for a seamless and robust user experience.

Explanation of Each Method & Menu Option

The app handles two types of users:

- **Admin (Developer-controlled)**
- **Normal Users (Registered Users)**

Once a user logs in or registers, the userActions() method offers the following functionalities:

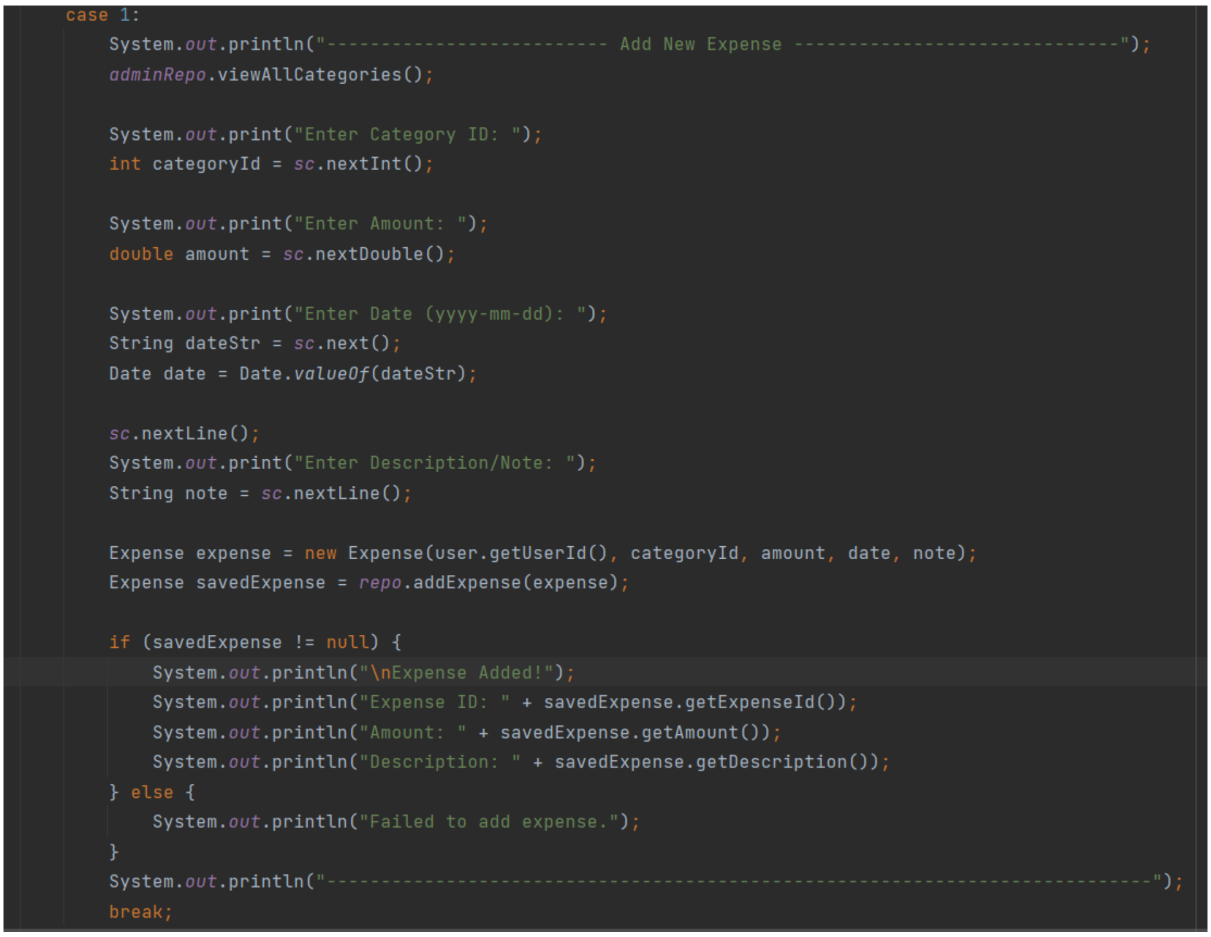


```
    System.out.println("\n----------------------------- USER MENU ---------------------------------");
    System.out.println("1. Create Expense");
    System.out.println("2. View All Expenses");
    System.out.println("3. Filter Expenses");
    System.out.println("4. Update Expense");
    System.out.println("5. Delete Expense");
    System.out.println("6. Generate Report");
    System.out.println("7. Suggestion");
    System.out.println("8. Delete Account");
    System.out.println("9. Logout");

    System.out.print("Choose an option: ");
    int choice = sc.nextInt();
```

---

1. Create Expense

- Allows the user to **add a new expense** by providing:
  - Category ID (with options listed from admin)
  - Amount
  - Date
  - Description
- The expense is saved into the database and confirmation is shown.

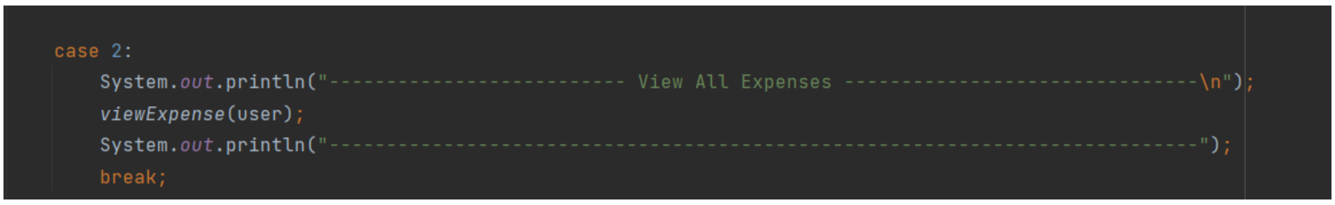


```java
case 1:
    System.out.println("------------------------- Add New Expense -----------------------------");
    adminRepo.viewAllCategories();

    System.out.print("Enter Category ID: ");
    int categoryId = sc.nextInt();

    System.out.print("Enter Amount: ");
    double amount = sc.nextDouble();

    System.out.print("Enter Date (yyyy-mm-dd): ");
    String dateStr = sc.next();
    Date date = Date.valueOf(dateStr);

    sc.nextLine();
    System.out.print("Enter Description/Note: ");
    String note = sc.nextLine();

    Expense expense = new Expense(user.getUserId(), categoryId, amount, date, note);
    Expense savedExpense = repo.addExpense(expense);

    if (savedExpense != null) {
        System.out.println("\nExpense Added!");
        System.out.println("Expense ID: " + savedExpense.getExpenseId());
        System.out.println("Amount: " + savedExpense.getAmount());
        System.out.println("Description: " + savedExpense.getDescription());
    } else {
        System.out.println("Failed to add expense.");
    }
    System.out.println("-----------------------------------------------------------------------------");
    break;
```

---

2. View All Expenses

- Displays a **list of all expenses** associated with the logged-in user.
- Data shown includes:Expense ID, Category, Amount, Date, Description

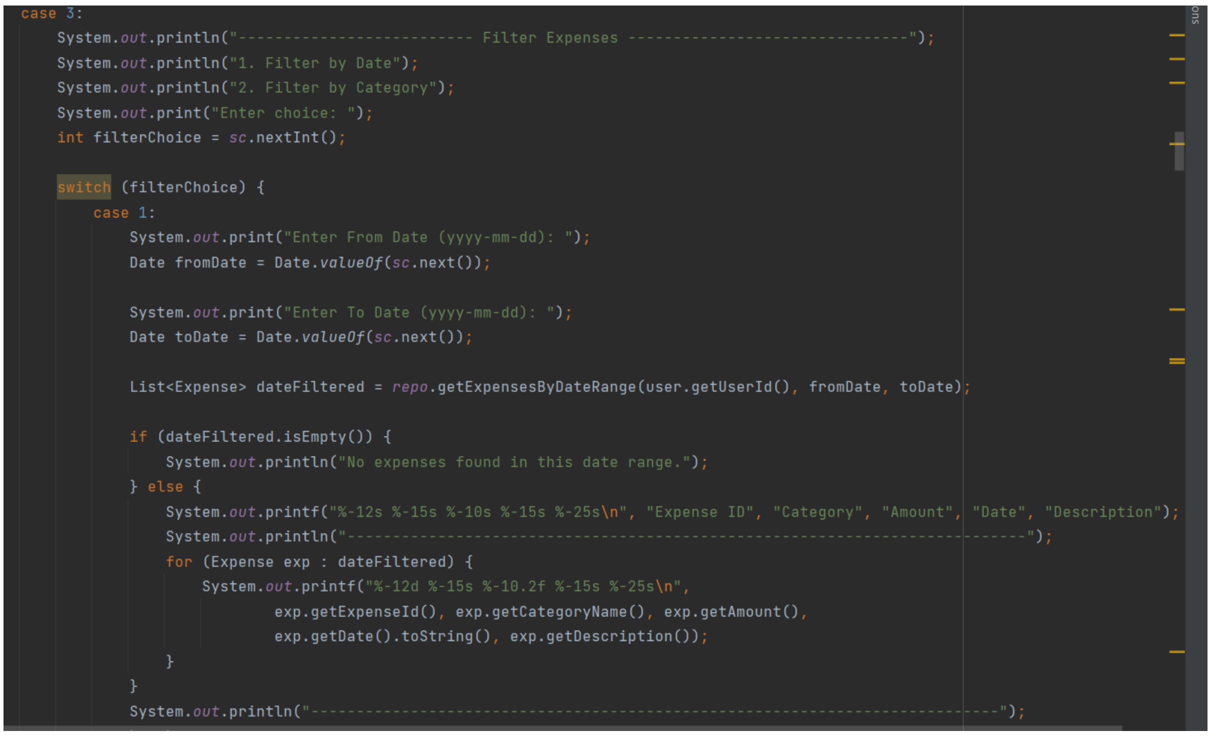


```java
case 2:
    System.out.println("------------------------- View All Expenses ------------------------------\n");
    viewExpense(user);
    System.out.println("------------------------------------------------------------------------------");
    break;
```

3. Filter Expenses

Provides two options:

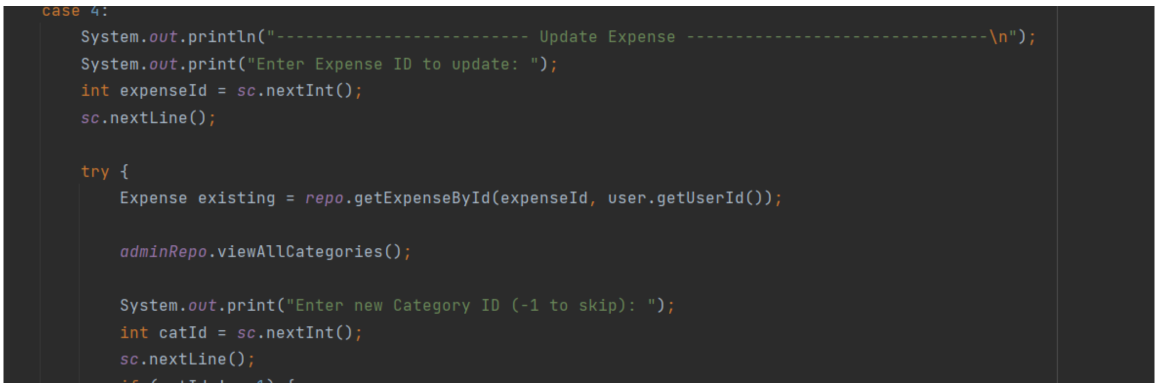
- **Filter by Date**: User inputs start and end date to view expenses in that range.
- **Filter by Category**: User inputs Category ID to view related expenses.



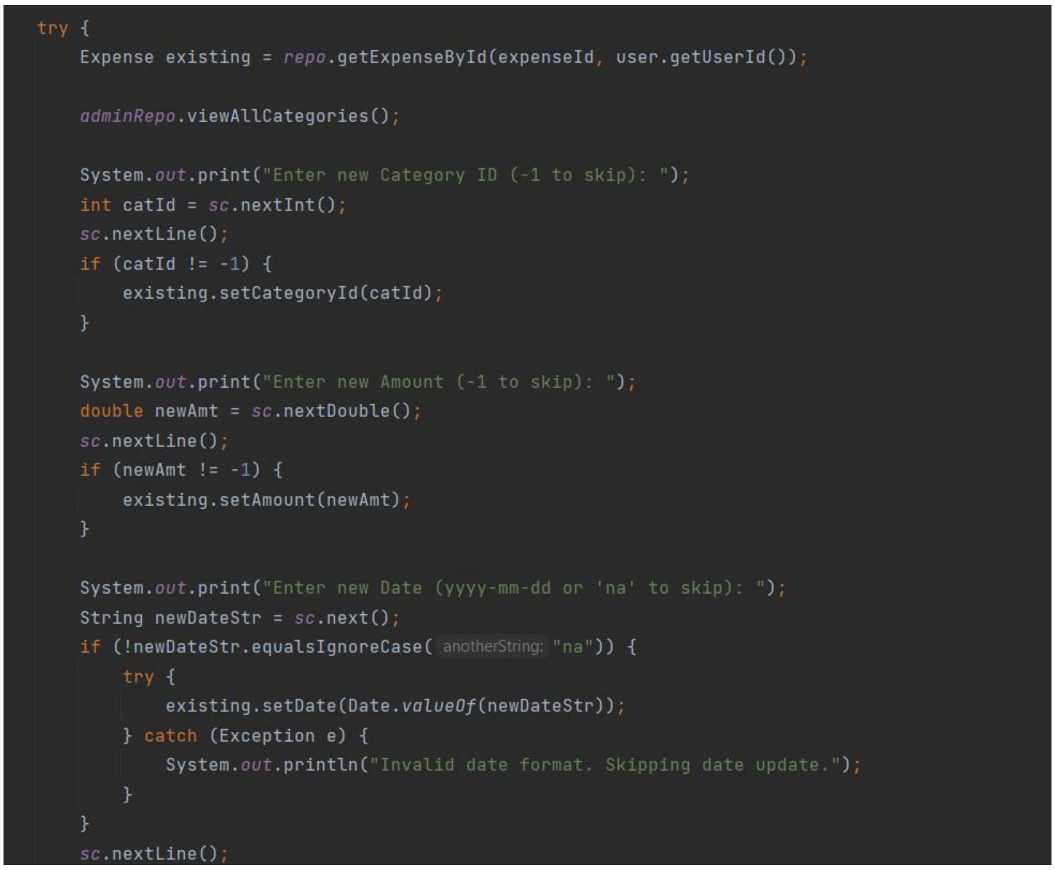
```java
case 3:
    System.out.println("------------------------ Filter Expenses -----------------------------");
    System.out.println("1. Filter by Date");
    System.out.println("2. Filter by Category");
    System.out.print("Enter choice: ");
    int filterChoice = sc.nextInt();

    switch (filterChoice) {
        case 1:
            System.out.print("Enter From Date (yyyy-mm-dd): ");
            Date fromDate = Date.valueOf(sc.next());

            System.out.print("Enter To Date (yyyy-mm-dd): ");
            Date toDate = Date.valueOf(sc.next());

            List<Expense> dateFiltered = repo.getExpensesByDateRange(user.getUserId(), fromDate, toDate);

            if (dateFiltered.isEmpty()) {
                System.out.println("No expenses found in this date range.");
            } else {
                System.out.printf("%-12s %-15s %-10s %-15s %-25s\n", "Expense ID", "Category", "Amount", "Date", "Description");
                System.out.println("-------------------------------------------------------------------------");
                for (Expense exp : dateFiltered) {
                    System.out.printf("%-12d %-15s %-10.2f %-15s %-25s\n",
                            exp.getExpenseId(), exp.getCategoryName(), exp.getAmount(),
                            exp.getDate().toString(), exp.getDescription());
                }
            }
            System.out.println("-------------------------------------------------------------------------");
```

4. Update Expense

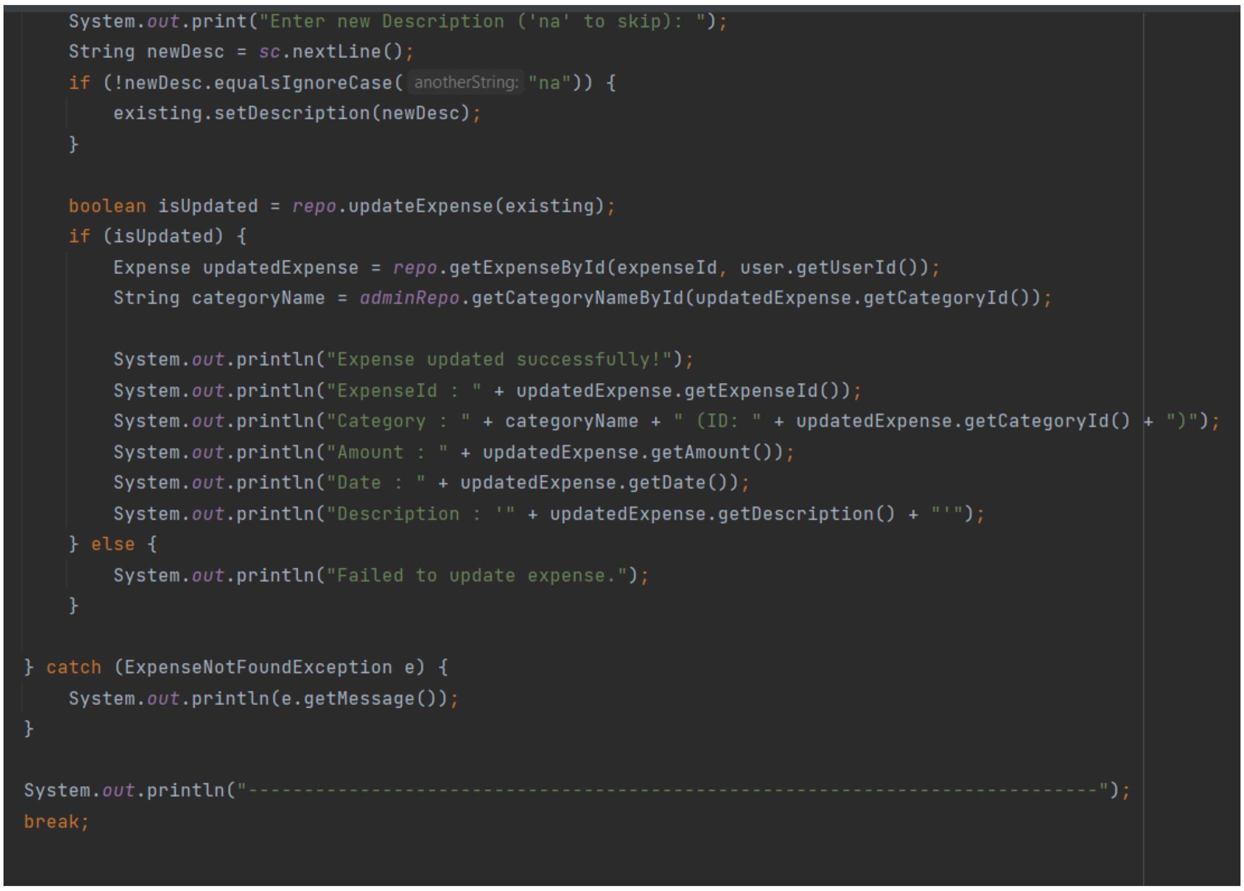
- Allows users to **modify an existing expense** by:
  - Entering Expense ID
  - Updating category, amount, date, and description (optional fields can be skipped using -1 or na)
- Exception handled: ExpenseNotFoundException if invalid ID.



```java
case 4:
    System.out.println("------------------------- Update Expense -----------------------------\n");
    System.out.print("Enter Expense ID to update: ");
    int expenseId = sc.nextInt();
    sc.nextLine();

    try {
        Expense existing = repo.getExpenseById(expenseId, user.getUserId());

        adminRepo.viewAllCategories();

        System.out.print("Enter new Category ID (-1 to skip): ");
        int catId = sc.nextInt();
        sc.nextLine();
```



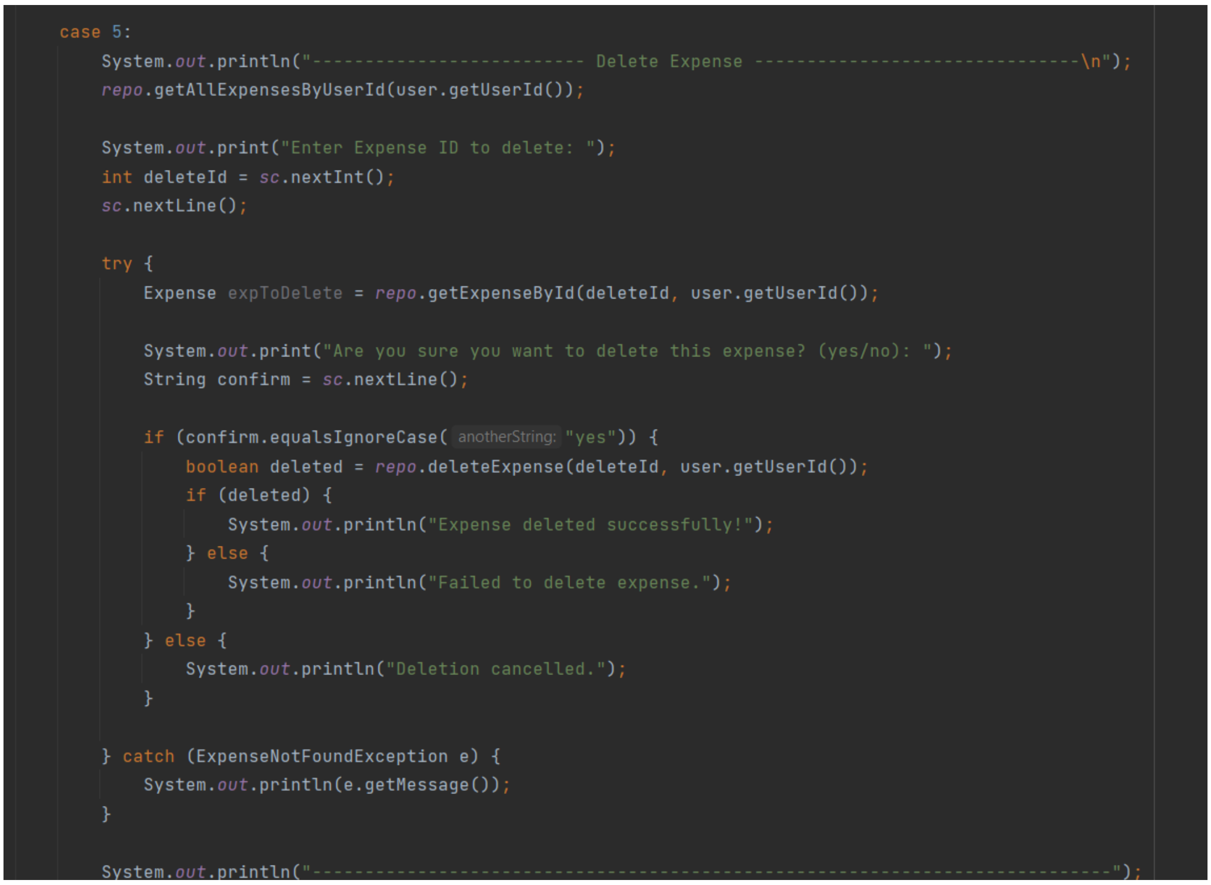
```java
try {
    Expense existing = repo.getExpenseById(expenseId, user.getUserId());

    adminRepo.viewAllCategories();

    System.out.print("Enter new Category ID (-1 to skip): ");
    int catId = sc.nextInt();
    sc.nextLine();
    if (catId != -1) {
        existing.setCategoryId(catId);
    }

    System.out.print("Enter new Amount (-1 to skip): ");
    double newAmt = sc.nextDouble();
    sc.nextLine();
    if (newAmt != -1) {
        existing.setAmount(newAmt);
    }

    System.out.print("Enter new Date (yyyy-mm-dd or 'na' to skip): ");
    String newDateStr = sc.next();
    if (!newDateStr.equalsIgnoreCase( anotherString: "na")) {
        try {
            existing.setDate(Date.valueOf(newDateStr));
        } catch (Exception e) {
            System.out.println("Invalid date format. Skipping date update.");
        }
    }
    sc.nextLine();
```



```java
    System.out.print("Enter new Description ('na' to skip): ");
    String newDesc = sc.nextLine();
    if (!newDesc.equalsIgnoreCase( anotherString: "na")) {
        existing.setDescription(newDesc);
    }

    boolean isUpdated = repo.updateExpense(existing);
    if (isUpdated) {
        Expense updatedExpense = repo.getExpenseById(expenseId, user.getUserId());
        String categoryName = adminRepo.getCategoryNameById(updatedExpense.getCategoryId());

        System.out.println("Expense updated successfully!");
        System.out.println("ExpenseId : " + updatedExpense.getExpenseId());
        System.out.println("Category : " + categoryName + " (ID: " + updatedExpense.getCategoryId() + ")");
        System.out.println("Amount : " + updatedExpense.getAmount());
        System.out.println("Date : " + updatedExpense.getDate());
        System.out.println("Description : '" + updatedExpense.getDescription() + "'");
    } else {
        System.out.println("Failed to update expense.");
    }

} catch (ExpenseNotFoundException e) {
    System.out.println(e.getMessage());
}

System.out.println("----------------------------------------------------------------------------");
break;
```

5. Delete Expense

- Deletes an expense by asking:
  - Expense ID
  - Confirmation (yes/no)
- Handles ExpenseNotFoundException if the expense doesn't exist.



```
case 5:
    System.out.println("------------------------- Delete Expense ------------------------------\n");
    repo.getAllExpensesByUserId(user.getUserId());

    System.out.print("Enter Expense ID to delete: ");
    int deleteId = sc.nextInt();
    sc.nextLine();

    try {
        Expense expToDelete = repo.getExpenseById(deleteId, user.getUserId());

        System.out.print("Are you sure you want to delete this expense? (yes/no): ");
        String confirm = sc.nextLine();

        if (confirm.equalsIgnoreCase( anotherString: "yes")) {
            boolean deleted = repo.deleteExpense(deleteId, user.getUserId());
            if (deleted) {
                System.out.println("Expense deleted successfully!");
            } else {
                System.out.println("Failed to delete expense.");
            }
        } else {
            System.out.println("Deletion cancelled.");
        }

    } catch (ExpenseNotFoundException e) {
        System.out.println(e.getMessage());
    }

    System.out.println("---------------------------------------------------------------------------");
```

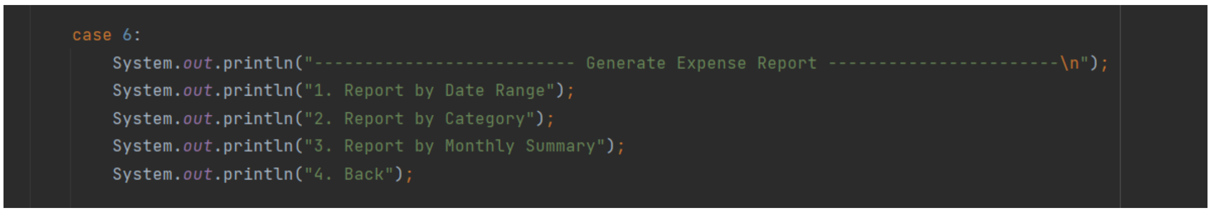
---

6. Generate Report

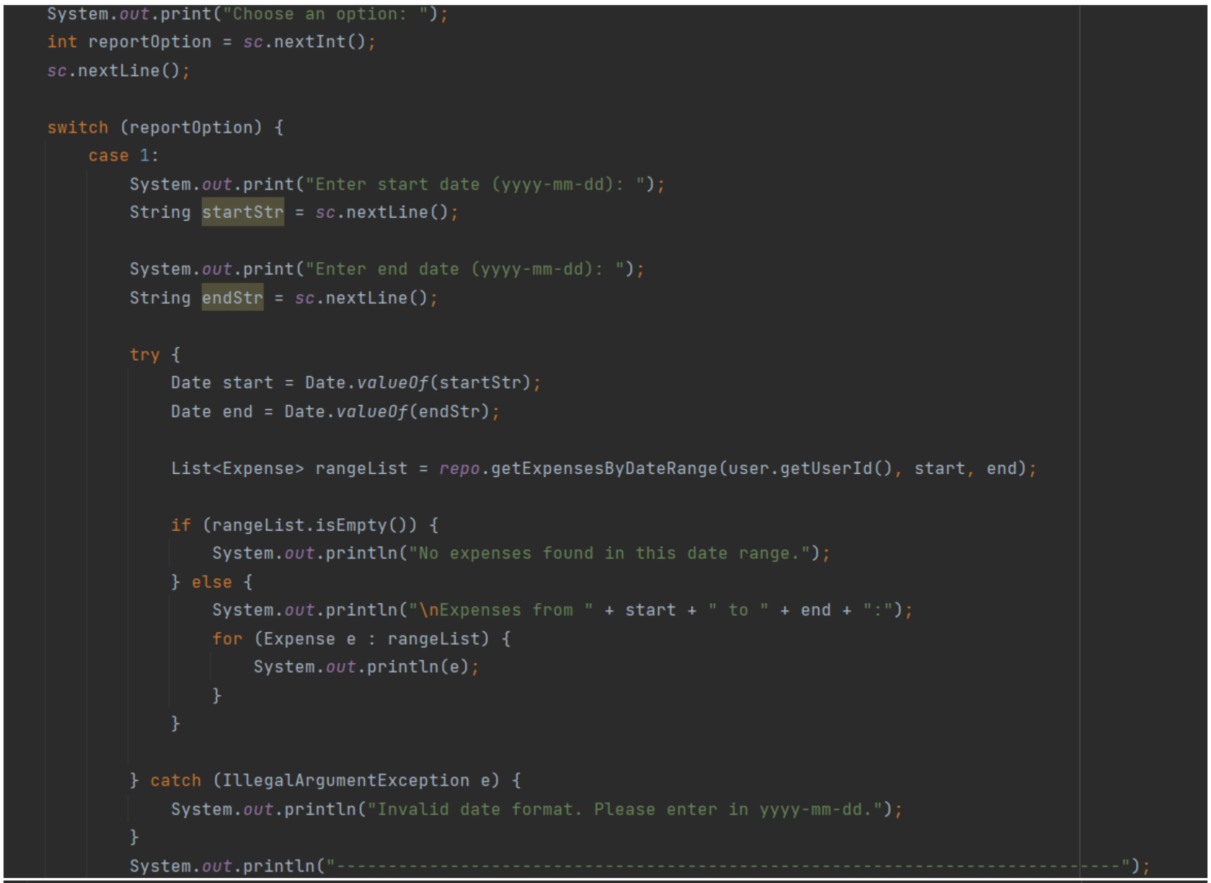
Offers 3 types of reports:

1. **By Date Range**: Lists expenses between two dates.
2. **By Category**: Category-wise totals between dates.
3. **Monthly Summary**: Shows total expense per month in a given range.

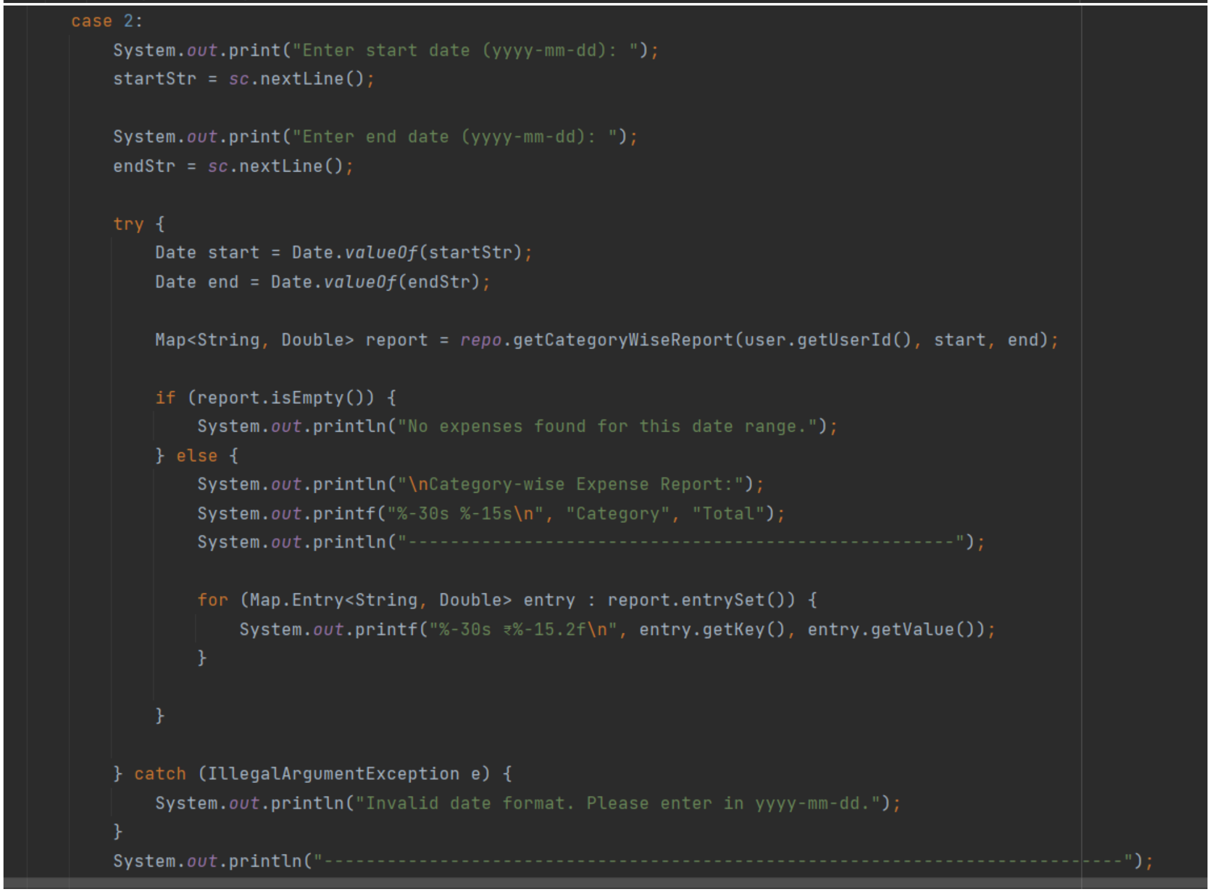
Each report provides valuable insights for budgeting and financial planning.



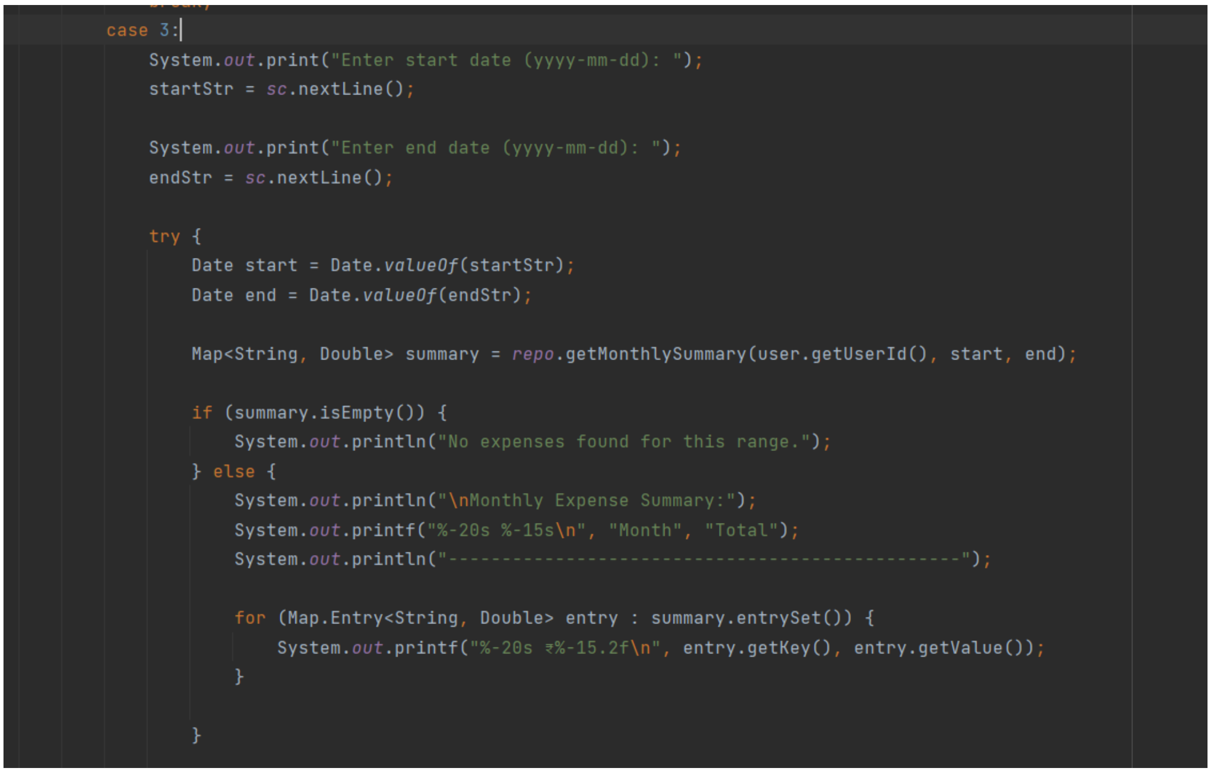
```
case 6:
    System.out.println("------------------------- Generate Expense Report ----------------------\n");
    System.out.println("1. Report by Date Range");
    System.out.println("2. Report by Category");
    System.out.println("3. Report by Monthly Summary");
    System.out.println("4. Back");
```



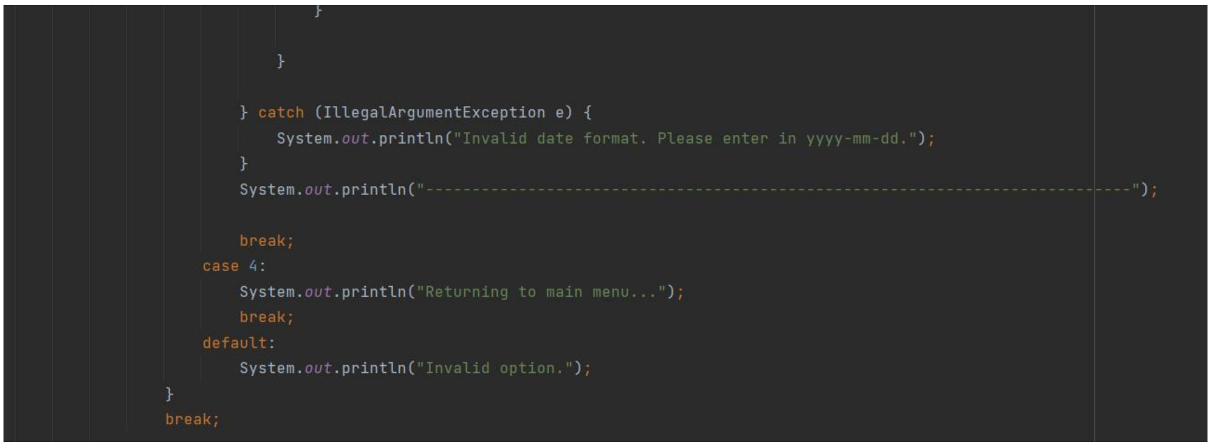
```java
System.out.print("Choose an option: ");
int reportOption = sc.nextInt();
sc.nextLine();

switch (reportOption) {
    case 1:
        System.out.print("Enter start date (yyyy-mm-dd): ");
        String startStr = sc.nextLine();

        System.out.print("Enter end date (yyyy-mm-dd): ");
        String endStr = sc.nextLine();

        try {
            Date start = Date.valueOf(startStr);
            Date end = Date.valueOf(endStr);

            List<Expense> rangeList = repo.getExpensesByDateRange(user.getUserId(), start, end);

            if (rangeList.isEmpty()) {
                System.out.println("No expenses found in this date range.");
            } else {
                System.out.println("\nExpenses from " + start + " to " + end + ":");
                for (Expense e : rangeList) {
                    System.out.println(e);
                }
            }

        } catch (IllegalArgumentException e) {
            Svstem.out.println("Invalid date format. Please enter in yyyy-mm-dd.");
        }
        System.out.println("-----------------------------------------------------------------------------");
```



```java
    case 2:
        System.out.print("Enter start date (yyyy-mm-dd): ");
        startStr = sc.nextLine();

        System.out.print("Enter end date (yyyy-mm-dd): ");
        endStr = sc.nextLine();

        try {
            Date start = Date.valueOf(startStr);
            Date end = Date.valueOf(endStr);

            Map<String, Double> report = repo.getCategoryWiseReport(user.getUserId(), start, end);

            if (report.isEmpty()) {
                System.out.println("No expenses found for this date range.");
            } else {
                System.out.println("\nCategory-wise Expense Report:");
                System.out.printf("%-30s %-15s\n", "Category", "Total");
                System.out.println("-----------------------------------------------------");

                for (Map.Entry<String, Double> entry : report.entrySet()) {
                    System.out.printf("%-30s ₹%-15.2f\n", entry.getKey(), entry.getValue());
                }

            }

        } catch (IllegalArgumentException e) {
            System.out.println("Invalid date format. Please enter in yyyy-mm-dd.");
        }
        System.out.println("------------------------------------------------------------------------------");
```



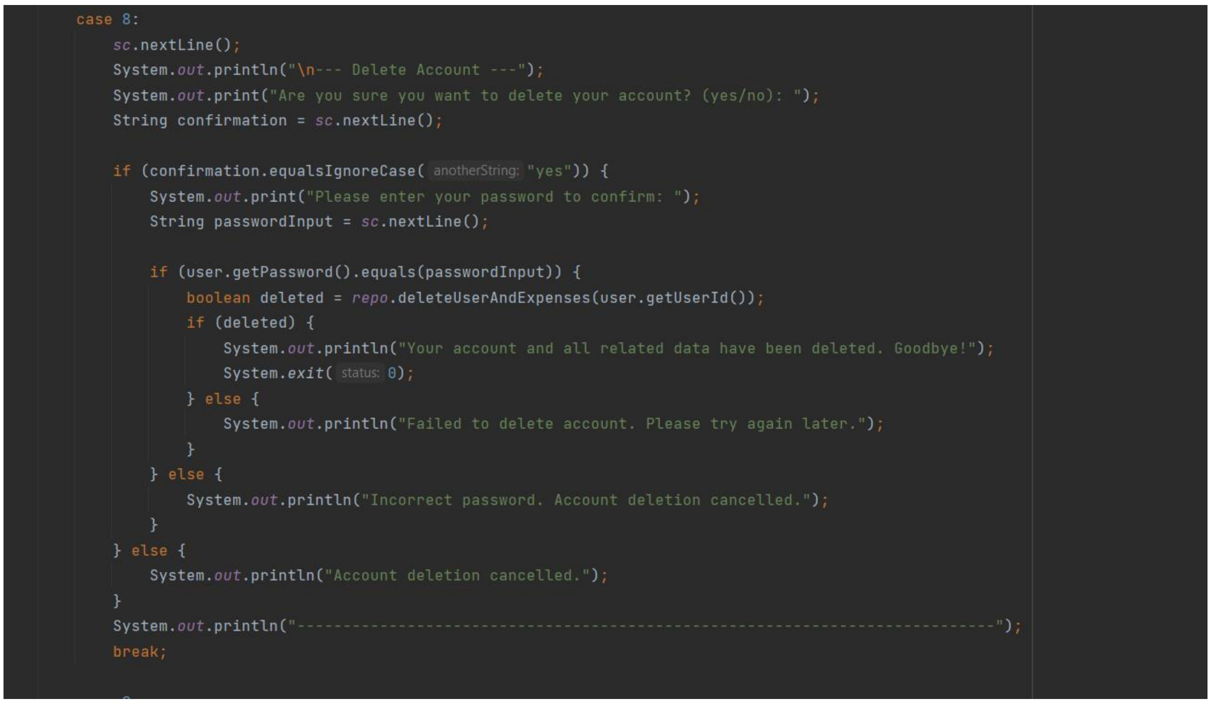
```
        case 3:
            System.out.print("Enter start date (yyyy-mm-dd): ");
            startStr = sc.nextLine();

            System.out.print("Enter end date (yyyy-mm-dd): ");
            endStr = sc.nextLine();

            try {
                Date start = Date.valueOf(startStr);
                Date end = Date.valueOf(endStr);

                Map<String, Double> summary = repo.getMonthlySummary(user.getUserId(), start, end);

                if (summary.isEmpty()) {
                    System.out.println("No expenses found for this range.");
                } else {
                    System.out.println("\nMonthly Expense Summary:");
                    System.out.printf("%-20s %-15s\n", "Month", "Total");
                    System.out.println("-----------------------------------------------");

                    for (Map.Entry<String, Double> entry : summary.entrySet()) {
                        System.out.printf("%-20s ₹%-15.2f\n", entry.getKey(), entry.getValue());
                    }

                }
```



```
                    }

                }

            } catch (IllegalArgumentException e) {
                System.out.println("Invalid date format. Please enter in yyyy-mm-dd.");
            }
            System.out.println("----------------------------------------------------------------------------");

            break;
        case 4:
            System.out.println("Returning to main menu...");
            break;
        default:
            System.out.println("Invalid option.");
    }
    break;
```

8. Delete Account

- Deletes both:
  - **User account**
  - **All related expenses**
- Requires password confirmation and user consent.
- Ensures full data wipeout upon deletion.

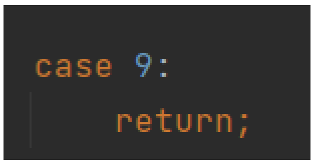


```java
case 8:
    sc.nextLine();
    System.out.println("\n--- Delete Account ---");
    System.out.print("Are you sure you want to delete your account? (yes/no): ");
    String confirmation = sc.nextLine();

    if (confirmation.equalsIgnoreCase( anotherString: "yes")) {
        System.out.print("Please enter your password to confirm: ");
        String passwordInput = sc.nextLine();

        if (user.getPassword().equals(passwordInput)) {
            boolean deleted = repo.deleteUserAndExpenses(user.getUserId());
            if (deleted) {
                System.out.println("Your account and all related data have been deleted. Goodbye!");
                System.exit( status: 0);
            } else {
                System.out.println("Failed to delete account. Please try again later.");
            }
        } else {
            System.out.println("Incorrect password. Account deletion cancelled.");
        }
    } else {
        System.out.println("Account deletion cancelled.");
    }
    System.out.println("---------------------------------------------------------------------------");
    break;
```

---

9. Logout

- Exits the user menu and returns to the main login/register screen.



```java
case 9:
    return;
```

Conclusion

The FinanceApp class efficiently ties together all services, models, and utilities of the system through an intuitive command-line interface. By offering CRUD operations on users and expenses, along with filters, reports, and custom suggestions, it ensures a complete end-to-end user experience. Robust validation and exception handling further enhance the system's usability and stability. This task marks the integration of all components into a functioning and user-friendly application.
