K.R. Mangalam University
School of Engineering & Technology

Fundamentals Of Java Programming
(ENCA203)
Capstone Assignment - 5

Submitted by:

Name: RAKESH G

Roll No: 2401201064

Class: BCA (AI & DS)

Submitted to:

Dr. Manish Kumar

GitHub Repository:
https://github.com/rakesh4407/Java_Assignment_Rakesh

1. Banking Application for Account Management

Code:

RAKESH_LAB_MANUAL.cpp ● BankingApplication.java 3 ● ManagementSystem.java 5 ● ResultManager.java 2

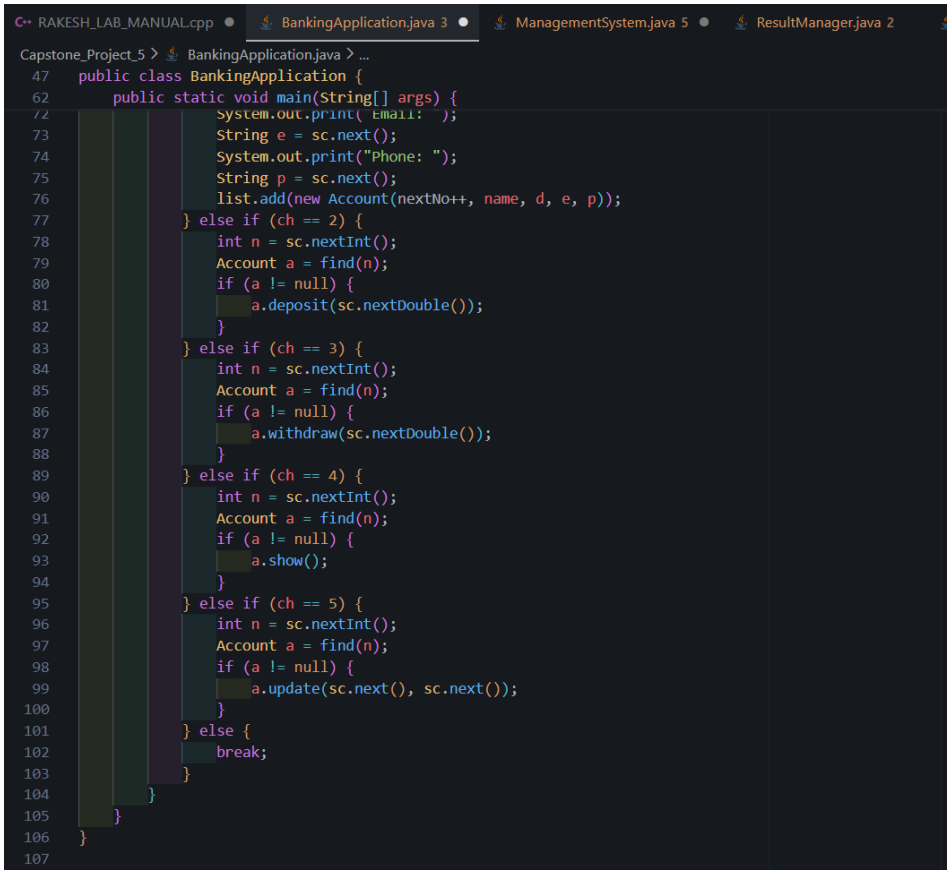
Capstone_Project_5 > BankingApplication.java > ...

```java
// PROJECT 1 - Banking Application
// Java Assingment 5
// RAKESH G - 2401201064

import java.util.*;

class Account {

    int accNo;
    String name, email, phone;    Variable email is never read
    double balance;

    Account(int accNo, String name, double bal, String email, String phone) {
        this.accNo = accNo;
        this.name = name;
        this.balance = bal;
        this.email = email;
        this.phone = phone;
    }

    void deposit(double amt) {
        if (amt > 0) {
            balance += amt;
        }else {
            System.out.println("Invalid amount");
        }
    }

    void withdraw(double amt) {
        if (amt > 0 && amt <= balance) {
            balance -= amt;
        }else {
            System.out.println("Insufficient balance");
        }
    }

    void update(String email, String phone) {
        this.email = email;
```

RAKESH_LAB_MANUAL.cpp ● BankingApplication.java 3 ● ManagementSystem.java 5 ● ResultManager.java 2

Capstone_Project_5 > BankingApplication.java > ...

```java
class Account {
    void update(String email, String phone) {
        this.phone = phone;
    }

    void show() {
        System.out.println(accNo + " | " + name + " | " + balance);
    }
}

public class BankingApplication {

    static Scanner sc = new Scanner(System.in);
    static ArrayList<Account> list = new ArrayList<>();
    static int nextNo = 1001;

    static Account find(int n) {
        for (Account a : list) {
            if (a.accNo == n) {
                return a;
            }
        }
        return null;
    }

    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        while (true) {
            System.out.println("1.Create 2.Deposit 3.Withdraw 4.View 5.Update 6.Exit");
            int ch = sc.nextInt();

            if (ch == 1) {    Replace chain of ifs with switch
                System.out.print("Name: ");
                String name = sc.next();
                System.out.print("Deposit: ");
                double d = sc.nextDouble();
                System.out.print("Email: ");
                String e = sc.next();
```


RAKESH_LAB_MANUAL.cpp
BankingApplication.java 3
ManagementSystem.java 5
ResultManager.java 2
Capstone_Project_5 > BankingApplication.java > ...
47 public class BankingApplication {
62 public static void main(String[] args) {
System.out.print("Email: ");
String e = sc.next();
System.out.print("Phone: ");
String p = sc.next();
list.add(new Account(nextNo++, name, d, e, p));
} else if (ch == 2) {
int n = sc.nextInt();
Account a = find(n);
if (a != null) {
a.deposit(sc.nextDouble());
}
} else if (ch == 3) {
int n = sc.nextInt();
Account a = find(n);
if (a != null) {
a.withdraw(sc.nextDouble());
}
} else if (ch == 4) {
int n = sc.nextInt();
Account a = find(n);
if (a != null) {
a.show();
}
} else if (ch == 5) {
int n = sc.nextInt();
Account a = find(n);
if (a != null) {
a.update(sc.next(), sc.next());
}
} else {
break;
}
}
}
}
107

Output:

```
Welcome to the Banking Application!
1. Create a new account
2. Deposit money
3. Withdraw money
4. View account details
5. Update contact details
6. Exit
Enter your choice: 1
Enter account holder name: RAKESH
Enter initial deposit amount: 5000
Enter email address: Rakesh64@gmail.com
Enter phone number: 2401201064
Account created successfully with Account Number: 1001

Welcome to the Banking Application!
1. Create a new account
2. Deposit money
3. Withdraw money
4. View account details
5. Update contact details
6. Exit
Enter your choice: 2
Enter account number: 1001
Enter deposit amount: 4000
Deposit successful. New balance: 9000.0

Welcome to the Banking Application!
1. Create a new account
2. Deposit money
3. Withdraw money
4. View account details
5. Update contact details
6. Exit
Enter your choice: 3
Enter account number: 1001
Enter withdrawal amount: 1000
Withdrawal successful. New balance: 8000.0
```

```
Welcome to the Banking Application!
1. Create a new account
2. Deposit money
3. Withdraw money
4. View account details
5. Update contact details
6. Exit
Enter your choice: 4
Enter account number: 1001
---- Account Details ----
Account Number : 1001
Holder Name    : RAKESH
Balance        : 8000.0
Email          : Rakesh64@gmail.com
Phone Number   : 2401201064
------------------------

Welcome to the Banking Application!
1. Create a new account
2. Deposit money
3. Withdraw money
4. View account details
5. Update contact details
6. Exit
Enter your choice: 6
Thank you for using the Banking Application. Goodbye!
```

Explanation:

This project is based on the basic concepts of Java programming, including classes, objects, arrays/ArrayList, control structures, and string handling. The main class in this project is the **Account** class, which contains important details such as account number, account holder name, balance, email, and phone number. The account number is automatically generated using a static variable so that every new account gets a unique number starting from 1001. The Account class provides essential operations such as deposit, withdraw, update contact details, and display account details. The deposit and withdraw methods include validation to make sure that only positive amounts are accepted and that the user does not withdraw more than what is available in the balance.
The **BankingApplication** class acts as the user interface where an ArrayList is used to store multiple accounts. A menu-driven system using loops and switch case statements allows the user to easily navigate through different banking operations like creating an account, depositing money, withdrawing money, viewing their details, and updating their contact information. The Scanner class is used to take user input. The program uses exception handling to avoid crashes when the user enters invalid input. All data is encapsulated properly to follow Object-Oriented Programming principles. The project successfully demonstrates the use of Java features like classes, objects, methods, control flow, loops, strings, and ArrayList. This fulfils the expected CO1 outcomes and creates a complete functional mini banking system.

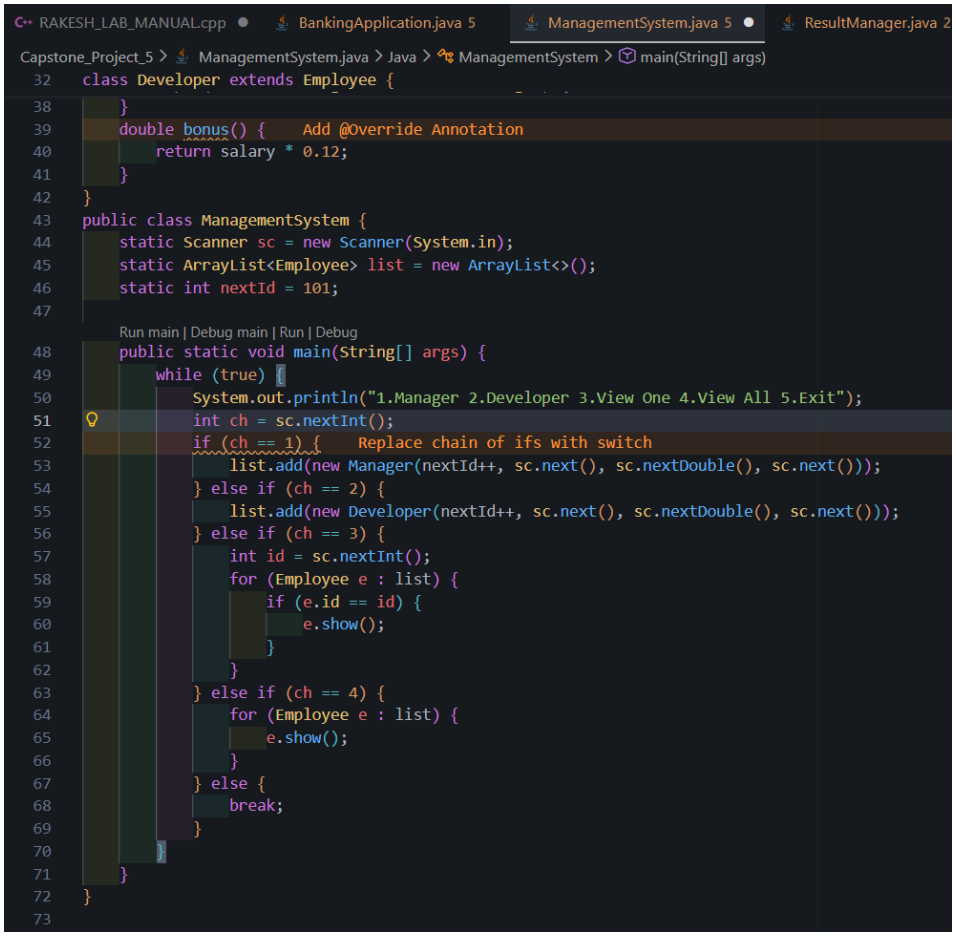
2. Employee Management System

Code:

RAKESH_LAB_MANUAL.cpp ● BankingApplication.java 5 ManagementSystem.java 5 ● Re

Capstone_Project_5 > ManagementSystem.java > Java > ManagementSystem > main(String[] args)

```java
// PROJECT 2 - Employee Management System
// Java Assingment 5
// RAKESH G - 2401201064
import java.util.*;
class Employee {
    int id;
    String name;
    double salary;
    Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
    double bonus() {
        return salary * 0.10;
    }
    void show() {
        System.out.println(id + " | " + name + " | " + bonus());
    }
}
class Manager extends Employee {
    String dept;    Variable dept is never read

    Manager(int id, String n, double s, String d) {
        super(id, n, s);
        dept = d;
    }
    double bonus() {    Add @Override Annotation
        return salary * 0.15;
    }
}
class Developer extends Employee {
    String lang;    Variable lang is never read

    Developer(int id, String n, double s, String l) {
        super(id, n, s);
        lang = l;
    }
```


RAKESH_LAB_MANUAL.cpp
BankingApplication.java 5
ManagementSystem.java 5
ResultManager.java 2
Capstone_Project_5 > ManagementSystem.java > Java > ManagementSystem > main(String[] args)
class Developer extends Employee {
}
double bonus() { Add @Override Annotation
return salary * 0.12;
}
}
public class ManagementSystem {
static Scanner sc = new Scanner(System.in);
static ArrayList<Employee> list = new ArrayList<>();
static int nextId = 101;
Run main | Debug main | Run | Debug
public static void main(String[] args) {
while (true) {
System.out.println("1.Manager 2.Developer 3.View One 4.View All 5.Exit");
int ch = sc.nextInt();
if (ch == 1) { Replace chain of ifs with switch
list.add(new Manager(nextId++, sc.next(), sc.nextDouble(), sc.next()));
} else if (ch == 2) {
list.add(new Developer(nextId++, sc.next(), sc.nextDouble(), sc.next()));
} else if (ch == 3) {
int id = sc.nextInt();
for (Employee e : list) {
if (e.id == id) {
e.show();
}
}
} else if (ch == 4) {
for (Employee e : list) {
e.show();
}
} else {
break;
}
}
}
}

Output:

```
Welcome to the Employee Management System!
1. Add Manager
2. Add Developer
3. Display Employee Details
4. Display All Employees
5. Exit
Enter your choice: 1
Enter Manager name: RAKESH
Enter salary: 90000
Enter department: IT
Manager added successfully with ID: 101

Welcome to the Employee Management System!
1. Add Manager
2. Add Developer
3. Display Employee Details
4. Display All Employees
5. Exit
Enter your choice: 2
Enter Developer name: Mani
Enter salary: 20000
Enter programming language: Java
Developer added successfully with ID: 102

Welcome to the Employee Management System!
1. Add Manager
2. Add Developer
3. Display Employee Details
4. Display All Employees
5. Exit
Enter your choice: 4
Manager -> ID: 101, Name: RAKESH, Department: IT, Salary: 90000.0, Bonus: 13500.0
Developer -> ID: 102, Name: Mani, Language: Java, Salary: 20000.0, Bonus: 2400.0

Welcome to the Employee Management System!
1. Add Manager
2. Add Developer
3. Display Employee Details
4. Display All Employees
5. Exit
Enter your choice: 5
Exiting Management System. Goodbye!

d:\RAKESH\VSC\Capstone_Project_5>
```

Explanation:

This project focuses on the concepts of **inheritance, polymorphism, method overriding, and object-oriented design**. The base class **Employee** contains common attributes like employee ID, name, and salary. It also includes a method called calculateBonus, which returns a default bonus amount (10% of salary) and a method displayDetails for printing the employee information. To demonstrate inheritance, two derived classes are created: **Manager** and **Developer**. Both classes extend the Employee class and add their own special fields: Manager has a department, while Developer has a programming language.

Both subclasses override the calculateBonus method to provide role-based bonus structures—Managers receive 15% while Developers receive 12%. This demonstrates **runtime polymorphism** because the correct bonus method is chosen at execution time depending on the object type. The main class **ManagementSystem** uses an ArrayList of type Employee, allowing both Manager and Developer objects to be stored in the same structure. A menu-based interface allows the user to add employees, search for an employee, and display all employees. This project clearly shows the use of inheritance, super(), method overriding, polymorphism, and encapsulation. These concepts are essential for CO2 learning outcomes and help in creating a well-structured employee management application.**Management System** uses an Array List of type Employee, allowing both Manager and Developer objects to be stored in the same structure. A menu-based interface allows the user to add employees, search for an employee, and display all employees. This project clearly shows the use of inheritance, super (), method overriding, polymorphism, and encapsulation. These concepts are essential for CO2 learning outcomes and help in creating a well-structured employee management application.

3. Student Result Management System

Code:

RAKESH_LAB_MANUAL.cpp ● BankingApplication.java ManagementSystem.java 5 ResultManager.java 2 ●

Capstone_Project_5 > ResultManager.java > ...

```java
// PROJECT 3 - Student Result System
// Java Assingment 5
// RAKESH G - 2401201064

import java.util.*;

class InvalidMarksException extends Exception {

    InvalidMarksException(String msg) {
        super(msg);
    }
}
class Student {
    int roll;
    String name;
    int[] m = new int[3];

    Student(int r, String n, int[] marks) throws InvalidMarksException {
        roll = r;
        name = n;
        m = marks;
        for (int x : m) {
            if (x < 0 || x > 100) {
                throw new InvalidMarksException("Invalid mark: " + x);
            }
        }
    }

    void show() {
        System.out.println(roll + " | " + name + " | " + (m[0] + m[1] + m[2]) / 3.0);
    }
}
public class ResultManager {

    static Scanner sc = new Scanner(System.in);
    static ArrayList<Student> list = new ArrayList<>();

    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        while (true) {
            System.out.println("1.Add 2.View 3.Exit");
            int ch = sc.nextInt();

            if (ch == 1) {    Replace chain of ifs with switch
                try {
                    int r = sc.nextInt();
                    String n = sc.next();
                    int[] marks = {sc.nextInt(), sc.nextInt(), sc.nextInt()};
                    list.add(new Student(r, n, marks));
                } catch (Exception e) {    Can be replaced with multicatch or several catch clauses
                    System.out.println(e);
                }
            } else if (ch == 2) {
                int r = sc.nextInt();
                for (Student s : list) {
                    if (s.roll == r) {
                        s.show();
                    }
                }
            } else {
                break;
            }
        }
    }
}
```

Output:

```
===== Student Result Management System =====
1. Add Student
2. Show Student Details
3. Exit
Enter your choice: 1
Enter Roll Number: 64
Enter Student Name: Rakesh
Enter marks for subject 1: 98
Enter marks for subject 2: 99
Enter marks for subject 3: 96
Student added successfully.

===== Student Result Management System =====
1. Add Student
2. Show Student Details
3. Exit
Enter your choice: 2
Enter Roll Number to search: 64
------ Student Result ------
Roll Number : 64
Name        : Rakesh
Marks       : 98, 99, 96
Average     : 97.67
Status      : Pass
----------------------------

===== Student Result Management System =====
1. Add Student
2. Show Student Details
3. Exit
Enter your choice: 3
Exiting Result Manager. Goodbye!

d:\RAKESH\VSC\Capstone_Project_5>
```

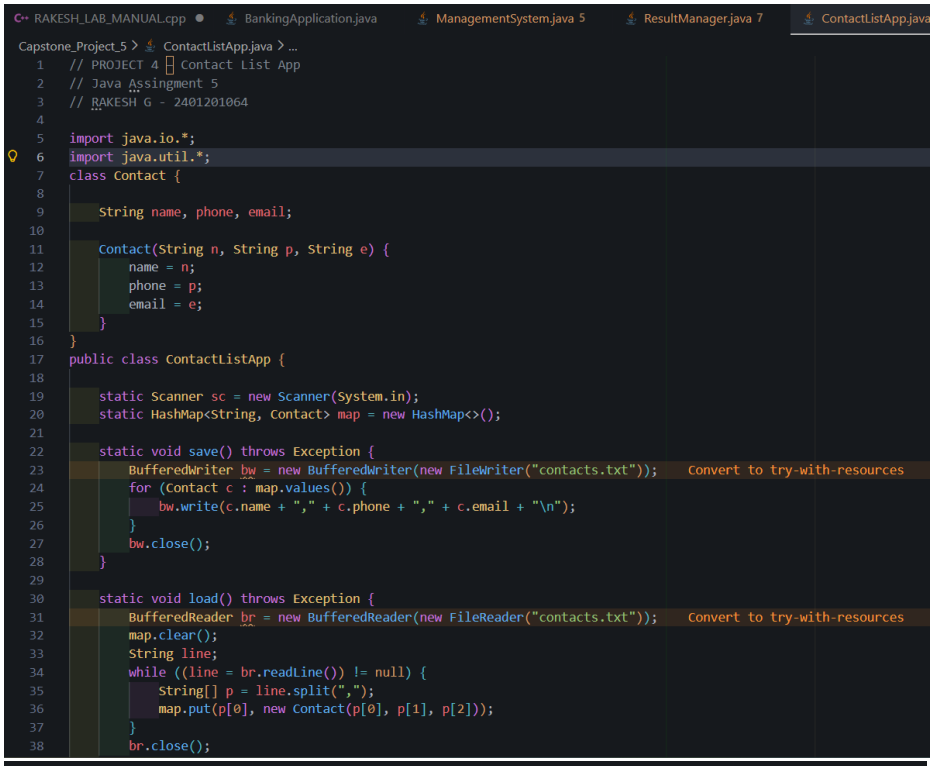
Explanation:

This project highlights **exception handling**, custom exceptions, input validation, and robust program design. The **Student** class stores the roll number, name, and marks of three subjects inside an integer array. A custom exception class named **InvalidMarksException** is created by extending the Exception class. This exception is thrown when the marks provided are not within the range 0–100. The Student class contains a validateMarks method that checks each mark and throws the custom exception if the value is invalid. It also includes methods to calculate average, determine pass/fail status, and display complete student results.

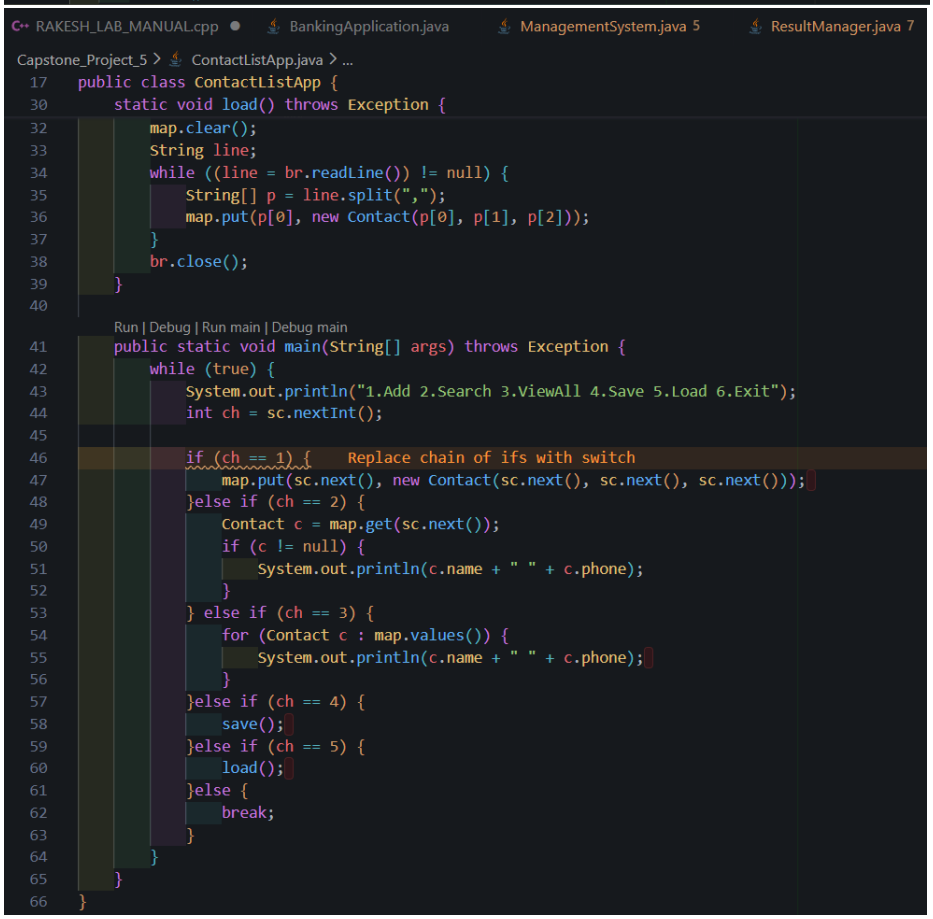
The **ResultManager** class uses an ArrayList of Student objects to manage multiple students. A menu-driven interface allows the user to add a new student or search for a student by roll number. The addStudent method uses try-catch blocks to handle custom exceptions, built-in exceptions like NumberFormatException, and general errors. This ensures the program does not crash even if the user enters incorrect data. When invalid marks are entered, the custom exception displays an informative message and returns control to the main menu. This project successfully shows the use of try-catch, throw, throws, custom exceptions, and input validation. It demonstrates CO4 learning outcomes through practical implementation of Java exception handling.

4. Contact List Application

Code:



```
// PROJECT 4 - Contact List App
// Java Assingment 5
// RAKESH G - 2401201064

import java.io.*;
import java.util.*;
class Contact {

    String name, phone, email;

    Contact(String n, String p, String e) {
        name = n;
        phone = p;
        email = e;
    }
}
public class ContactListApp {

    static Scanner sc = new Scanner(System.in);
    static HashMap<String, Contact> map = new HashMap<>();

    static void save() throws Exception {
        BufferedWriter bw = new BufferedWriter(new FileWriter("contacts.txt"));    Convert to try-with-resources
        for (Contact c : map.values()) {
            bw.write(c.name + "," + c.phone + "," + c.email + "\n");
        }
        bw.close();
    }

    static void load() throws Exception {
        BufferedReader br = new BufferedReader(new FileReader("contacts.txt"));    Convert to try-with-resources
        map.clear();
        String line;
        while ((line = br.readLine()) != null) {
            String[] p = line.split(",");
            map.put(p[0], new Contact(p[0], p[1], p[2]));
        }
        br.close();
```



```
public class ContactListApp {
    static void load() throws Exception {
        map.clear();
        String line;
        while ((line = br.readLine()) != null) {
            String[] p = line.split(",");
            map.put(p[0], new Contact(p[0], p[1], p[2]));
        }
        br.close();
    }

    Run | Debug | Run main | Debug main
    public static void main(String[] args) throws Exception {
        while (true) {
            System.out.println("1.Add 2.Search 3.ViewAll 4.Save 5.Load 6.Exit");
            int ch = sc.nextInt();

            if (ch == 1) {    Replace chain of ifs with switch
                map.put(sc.next(), new Contact(sc.next(), sc.next(), sc.next()));
            }else if (ch == 2) {
                Contact c = map.get(sc.next());
                if (c != null) {
                    System.out.println(c.name + " " + c.phone);
                }
            } else if (ch == 3) {
                for (Contact c : map.values()) {
                    System.out.println(c.name + " " + c.phone);
                }
            }else if (ch == 4) {
                save();
            }else if (ch == 5) {
                load();
            }else {
                break;
            }
        }
    }
}
```

Output:

```
Welcome to the Contact List Application!
1. Add a new contact
2. Remove an existing contact
3. Search for a contact by name
4. Display all contacts
5. Save contacts to file
6. Load contacts from file
7. Exit
Enter your choice: 1
Enter contact name: Rakesh
Enter phone number: 2401201064
Enter email: Rakesh@gmail.com
Contact added successfully.

Welcome to the Contact List Application!
1. Add a new contact
2. Remove an existing contact
3. Search for a contact by name
4. Display all contacts
5. Save contacts to file
6. Load contacts from file
7. Exit
Enter your choice: 4
Name : Rakesh
Phone: 2401201064
Email: Rakesh@gmail.com
------------------

Welcome to the Contact List Application!
1. Add a new contact
2. Remove an existing contact
3. Search for a contact by name
4. Display all contacts
5. Save contacts to file
6. Load contacts from file
7. Exit
Enter your choice: 7
Exiting Contact List App. Goodbye!

d:\RAKESH\VSC\Capstone Project 5>
```

Explanation:

This project demonstrates the usage of **Java Collections Framework and File Handling**. A Contact class is created with name, phone number, and email attributes. The main class **ContactListApp** uses a **HashMap** to store contacts where the key is the contact's name in lowercase. HashMap provides fast searching, adding, and removing, making it ideal for a contact list application. The program includes functions to add a contact, delete a contact, search a contact, and display all contacts.
One of the main features is saving and loading contacts using File Handling. The program uses **BufferedWriter** and **BufferedReader** to write and read contact information from a text file in a simple comma-separated format. When saved, each contact is stored as a line in the file, and when loading, the file is read and converted back into Contact objects. The program includes proper error handling for file-related issues. A menu-based interface using loops and switch case statements allows the user to perform all actions easily. This project achieves CO6 outcomes by demonstrating real-world use of Collections, file I/O, string manipulation, and data persistence between sessions.

5. Library Management System

Code:

RAKESH_LAB_MANUAL.cpp ● ContactListApp.java 6 LibraryApp.java 1 ●

Capstone_Project_5 > LibraryApp.java > ...

```java
// PROJECT 5 - Library Management System
// Java Assingment 5
// RAKESH G - 2401201064
import java.util.*;
class Book {
    String title, author;
    boolean available = true;
    Book(String t, String a) {
        title = t;
        author = a;
    }
    void show() {
        System.out.println(title + " | " + author + " | " + (available ? "Available" : "Taken"));
    }
}
public class LibraryApp {
    static Scanner sc = new Scanner(System.in);
    static ArrayList<Book> list = new ArrayList<>();
    static Book search(String t) {
        for (Book b : list) {
            if (b.title.equalsIgnoreCase(t)) {
                return b;
            }
        }
        return null;
    }
    static void searchByAuthor(String a) {
        for (Book b : list) {
            if (b.author.equalsIgnoreCase(a)) {
                b.show();
            }
        }
    }
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        while (true) {
            System.out.println("1.Add 2.SearchTitle 3.SearchAuthor 4.Borrow 5.Return 6.ViewAll 7.Exit");
            int ch = sc.nextInt();
```

RAKESH_LAB_MANUAL.cpp ● ContactListApp.java 6 LibraryApp.java 1 ●

Capstone_Project_5 > LibraryApp.java > ...

```java
public class LibraryApp {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        while (true) {
            System.out.println("1.Add 2.SearchTitle 3.SearchAuthor 4.Borrow 5.Return 6.ViewAll 7.Exit");
            int ch = sc.nextInt();

            if (ch == 1) {    Replace chain of ifs with switch
                list.add(new Book(sc.next(), sc.next()));
            }else if (ch == 2) {
                Book b = search(sc.next());
                if (b != null) {
                    b.show();
                }
            } else if (ch == 3) {
                searchByAuthor(sc.next());
            }else if (ch == 4) {
                Book b = search(sc.next());
                if (b != null) {
                    b.available = false;
                }
            } else if (ch == 5) {
                Book b = search(sc.next());
                if (b != null) {
                    b.available = true;
                }
            } else if (ch == 6) {
                for (Book b : list) {
                    b.show();
                }
            }else {
                break;
            }
        }
    }
}
```

Output:

```
Welcome to the Library Management System!
1. Add a new book
2. Search for a book by title
3. Search for books by author
4. Borrow a book
5. Return a book
6. Display all books
7. Exit
Enter your choice: 1
Enter book title: Rock Star
Enter book author: Rakesh
Book added successfully.

Welcome to the Library Management System!
1. Add a new book
2. Search for a book by title
3. Search for books by author
4. Borrow a book
5. Return a book
6. Display all books
7. Exit
Enter your choice: 3
Enter author: Rakesh
Title : Rock Star
Author: Rakesh
Status: Available
------------------

Welcome to the Library Management System!
1. Add a new book
2. Search for a book by title
3. Search for books by author
4. Borrow a book
5. Return a book
6. Display all books
7. Exit
Enter your choice: 7
Exiting Library App. Goodbye!

d:\RAKESH\VSC\Capstone_Project_5>
```

Explanation:

This project focuses on **Object-Oriented Programming and method overloading**. A Book class is created with attributes like title, author, and availability status. The LibraryApp class maintains a collection of books using an ArrayList. The user can add new books, borrow books, return books, search for books, and view all books. Borrow and return methods change the availability status and display appropriate messages.
The main concept shown in this project is **method overloading**. Two versions of the searchBook method are created—one searches by title and returns a single Book, while the other searches by author and returns a list of books by that author. This helps in demonstrating how the same method name can be used with different parameters to perform different operations. A simple menu-driven program uses loops and switch case statements to provide the user interface. The system displays all the books with their status so the user can easily understand which books are available. This project effectively demonstrates the use of classes, objects, strings, arrays/collections, method overloading, and control structures. It meets the expected CO2 outcomes related to OOP and method overloading.