Assignment 3

Question 1

You are developing a Java program for a simple library management system using an ArrayList to manage books in the library. Each book should have attributes including the title, author, and ISBN number. Implement a Book class with methods to perform the following actions:

1. **Allow users to add a new book to the library.**
2. **Allow users to remove a book from the library by providing its ISBN number.**
3. **Enable users to search for a book by title and display its details.**
4. **Display the list of all books currently in the library.**

```
import java.util.*;

class Book {
    String title;
    String author;
    int ISBN;

    Book(String title, String author, int ISBN) {
        this.title = title;
        this.author = author;
        this.ISBN = ISBN;
    }

    public String toString() {
        return "\nTitle: " + title + "\nAuthor: " + author +
"\nISBN: " + ISBN;
    }
}

public class LibArrayList {

    static void displayList(Collection<?> list) {
        for (Object obj : list) {
            System.out.println(obj);
        }
    }

    static Book findBookByTitle(List<Book> list, String bookTitle) {
        for (Book book : list) {
            if (book.title.equalsIgnoreCase(bookTitle)) {
                return book;
            }
        }
        return null;
    }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Book> list1 = new ArrayList<>();
        int choice;

        do {
            System.out.println("\n\nChoose your option");
            System.out.println("1. Add new book");
            System.out.println("2. Remove a book by ISBN");
            System.out.println("3. Search for book by Title");
            System.out.println("4. Display list of all books");
            System.out.println("5. To quit");
            System.out.println("Choice: ");
            choice = sc.nextInt();
            sc.nextLine();

            switch (choice) {
                case 1:
                    System.out.println("\n\nEnter the total number
of books you want to add: ");
                    int num = sc.nextInt();
                    sc.nextLine();
                    for (int i = 0; i < num; i++) {
                        System.out.print("\nEnter book title: ");
                        String title = sc.nextLine();

                        System.out.print("Enter book author: ");
                        String author = sc.nextLine();

                        System.out.print("Enter ISBN number: ");
                        int isbn = sc.nextInt();
                        sc.nextLine();

                        list1.add(new Book(title, author, isbn));
                    }
                    break;

                case 2:
                    System.out.println("\nEnter the ISBN of book you
wish to remove: ");
                    int removeISBN = sc.nextInt();
                    sc.nextLine();
                    Book toRemove = null;
                    for (Book book : list1) {
                        if (book.ISBN == removeISBN) {
                            toRemove = book;
                            break;
```

```
                        }
                    }
                    if (toRemove != null) {
                        list1.remove(toRemove);
                    } else {
                        System.out.println("\nBook with ISBN " +
removeISBN + " not found.");
                    }
                    break;

                case 3:
                    System.out.println("\nEnter title of book you
want to search for: ");
                    String searchBook = sc.nextLine();
                    Book findBook = findBookByTitle(list1,
searchBook);
                    if (findBook != null) {
                        System.out.println(findBook);
                    } else {
                        System.out.println("\nBook not found");
                    }
                    break;

                case 4:
                    System.out.println("List of books:");
                    displayList(list1);
                    break;

                case 5:
                    System.out.println("Exit");
                    break;

                default:
                    System.out.println("Invalid choice");
                    break;
            }
        } while (choice != 5);

        sc.close();
    }
}
```

```
PS D:\CDAC\Java\AssignmentNo3> javac LibArrayList.java
PS D:\CDAC\Java\AssignmentNo3> java LibArrayList


Choose your option
1. Add new book
2. Remove a book by ISBN
3. Search for book by Title
4. Display list of all books
5. To quit
Choice:
1


Enter the total number of books you want to add:
5

Enter book title: 1948
Enter book author: George Orwell
Enter ISBN number: 1234

Enter book title: The Great Gatsby
Enter book author: F. Scott Fitzgerald
Enter ISBN number: 5678

Enter book title: Pride and Prejudice
Enter book author: Jane Austen
Enter ISBN number: 2244

Enter book title: The Lantern of Lost Memories
Enter book author: Sanaka Hiiragi
Enter ISBN number: 6745
```

```
Enter book title: White Nights
Enter book author: Fyodor Dostoevsky
Enter ISBN number: 2489


Choose your option
1. Add new book
2. Remove a book by ISBN
3. Search for book by Title
4. Display list of all books
5. To quit
Choice:
4
List of books:

Title: 1948
Author: George Orwell
ISBN: 1234

Title: The Great Gatsby
Author: F. Scott Fitzgerald
ISBN: 5678

Title: Pride and Prejudice
Author: Jane Austen
ISBN: 2244

Title: The Lantern of Lost Memories
Author: Sanaka Hiiragi
ISBN: 6745
```

```
Title: White Nights
Author: Fyodor Dostoevsky
ISBN: 2489


Choose your option
1. Add new book
2. Remove a book by ISBN
3. Search for book by Title
4. Display list of all books
5. To quit
Choice:
2

Enter the ISBN of book you wish to remove:
2233

Book with ISBN 2233 not found.


Choose your option
1. Add new book
2. Remove a book by ISBN
3. Search for book by Title
4. Display list of all books
5. To quit
Choice:
3
```

```
Enter title of book you want to search for:
The Lantern of Lost Memories

Title: The Lantern of Lost Memories
Author: Sanaka Hiiragi
ISBN: 6745


Choose your option
1. Add new book
2. Remove a book by ISBN
3. Search for book by Title
4. Display list of all books
5. To quit
Choice:
5
Exit
```

Question 2

You are developing a Java program to manage an online shopping cart. Implement code to handle the following built-in exceptions:

1. **ArrayIndexOutOfBoundsException**
2. **NumberFormatException**
3. **ArithmeticException**

```
import java.util.Scanner;

public class ShoppingCart {
    public static void main(String[] args) {
        try {
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter number of items in cart: ");
            int no = sc.nextInt();
            sc.nextLine();
            String[] cart = new String[no];
            int i = 0;
            while (i < no) {
                System.out.println("\nItem " + (i + 1) + ": ");
                String item = sc.nextLine();

                cart[i] = item;
                i++;
            }

            System.out.println("\nItems in cart: ");
            for (int j = 0; j < no; j++) {
                System.out.println(cart[j]);
            }

            System.out.println("\n\nEnter total quantity (in
numbers): ");
            String quantity = sc.nextLine();
            int stringToInt = Integer.parseInt(quantity);
            System.out.println("Quantity: " + stringToInt);

            System.out.println("\n\nEnter total items: ");
            int num1 = sc.nextInt();
            System.out.println("Number of people: ");
            int num2 = sc.nextInt();
            int res = num1 / num2;
            System.out.println("Items per person: " + res);

            sc.close();

        } catch (ArrayIndexOutOfBoundsException arre) {
```

```
            System.out.println("\nException: " + arre);
        } catch (NumberFormatException ne) {
            System.out.println("\nException: " + ne);
        } catch (ArithmeticException ae) {
            System.out.println("\nException: " + ae);
        }
    }
}
```

```
PS D:\CDAC\Java\AssignmentNo3> javac ShoppingCart.java
PS D:\CDAC\Java\AssignmentNo3> java ShoppingCart
Enter number of items in cart:
6

Item 1:
Slingback Heels

Item 2:
Novels

Item 3:
Tea Set

Item 4:
Shampoo

Item 5:
Fur Cloak

Item 6:
Phone

Items in cart:
Slingback Heels
Novels
Tea Set
Shampoo
Fur Cloak
Phone
```

```
Enter total quantity (in numbers):
6
Quantity: 6


Enter total items:
6
Number of people:
2
Items per person: 3
```

Question 3

You are developing a Java application for managing vehicles in a rental service. Create a class named Vehicle with attributes for make, model, and year. Implement a constructor that initializes these fields. Next, design a subclass named Car with an additional attribute for numberOfDoors. Provide two constructors for the Car class: one that accepts all fields including make, model, year, and numberOfDoors, and another that accepts only make and model, chaining to the superclass constructor. Validate the input in each constructor and provide appropriate getter and setter methods to access and modify the attributes of the Vehicle class. Ensure proper usage of inheritance principles.

```java
class Vehicle {
    private String make;
    private String model;
    private int year;

    Vehicle(String make, String model, int year) {
        if (make == null) {
            this.make = null;
        } else {
            this.make = make;
        }

        if (model == null) {
            this.model = null;
        } else {
            this.model = model;
        }

        if (year < 1990 || year > 2025) {
            this.year = 0;
        } else {
            this.year = year;
        }
    }
```

```
    public String getMake() {
        return make;
    }

    public void setMake(String make) {
        if (make == null) {
            this.make = null;
        } else {
            this.make = make;
        }
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        if (model == null) {
            this.model = null;
        } else {
            this.model = model;
        }
    }

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        if (year < 1990 || year > 2025) {
            this.year = 0;
        } else {
            this.year = year;
        }
    }

    public String toString() {
        return "Make: " + make + "\nModel: " + model + "\nYear: " +
year + "\n";
    }
}

class Car extends Vehicle {
    int noOfDoors;

    Car(String make, String model, int year, int noOfDoors) {
        super(make, model, year);
        if (noOfDoors < 2 || noOfDoors > 5) {
```

```
            this.noOfDoors = 0;
        } else {
            this.noOfDoors = noOfDoors;
        }
    }

    Car(String make, String model) {
        super(make, model, 0);
    }

    public int getNoOfDoors() {
        return noOfDoors;
    }

    public void setNoOfDoors(int noOfDoors) {
        if (noOfDoors < 2 || noOfDoors > 5) {
            this.noOfDoors = 0;
        } else {
            this.noOfDoors = noOfDoors;
        }
    }

    public String toString() {
        return "Make: " + getMake() + "\nModel: " + getModel() +
"\nYear: " + getYear() + "\nNumber of Doors: "
                + getNoOfDoors() + "\n";
    }
}

public class Question3 {
    public static void main(String[] args) {
        Vehicle v = new Vehicle("Toyota", "Yaris Cross", 2020);
        System.out.println(v.toString());

        Car c1 = new Car("Hyundai", "Creta");
        System.out.println(c1.toString());

        Car c2 = new Car("Toyota", "Crown", 1979, 4);
        System.out.println(c2.toString());

        Car c3 = new Car("Mahindra", "Thar", 2010, 8);
        System.out.println(c3.toString());
    }
}
```

```
PS D:\CDAC\Java\AssignmentNo3> javac Question3.java
PS D:\CDAC\Java\AssignmentNo3> java Question3
Make: Toyota
Model: Yaris Cross
Year: 2020

Make: Hyundai
Model: Creta
Year: 0
Number of Doors: 0

Make: Toyota
Model: Crown
Year: 0
Number of Doors: 4

Make: Mahindra
Model: Thar
Year: 2010
Number of Doors: 0
```

Question 4

Create a Java program to manage an employee database for a company using an ArrayList. Each employee should have attributes such as employee name, employee ID, and department. Implement an Employee class with methods to perform the following operations:

1. **Add a new employee to the database.**
2. **Update an employee's department using their employee ID.**
3. **Remove an employee from the database using their employee ID.**
4. **Display the list of all employees along with their details.**

```java
import java.util.*;

class Employee {
    String name;
    int empID;
    String department;

    Employee(String name, int empID, String department) {
        this.name = name;
        this.empID = empID;
        this.department = department;
    }

    public String toString() {
```

```
        return "\nName: " + name + "\nEmployee ID: " + empID +
"\nDepartment: " + department;
    }
}

public class EmpArrayList {
    static void displayList(Collection<?> list) {
        for (Object obj : list) {
            System.out.println(obj);
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Employee> list1 = new ArrayList<>();
        int choice;

        do {
            System.out.println("\n\nChoose your option");
            System.out.println("1. Add new employee");
            System.out.println("2. Update employee department by
ID");
            System.out.println("3. Remove employee by ID");
            System.out.println("4. Display list of all employees");
            System.out.println("5. To quit");
            System.out.println("Choice: ");
            choice = sc.nextInt();
            sc.nextLine();

            switch (choice) {
                case 1:
                    System.out.println("\n\nEnter the total number
of employee details you want to add: ");
                    int num = sc.nextInt();
                    sc.nextLine();
                    for (int i = 0; i < num; i++) {
                        System.out.print("\nEnter employee name: ");
                        String name = sc.nextLine();

                        System.out.print("Enter employee ID: ");
                        int empID = sc.nextInt();
                        sc.nextLine();

                        System.out.print("Enter department: ");
                        String department = sc.nextLine();

                        list1.add(new Employee(name, empID,
department));
```

```
                    }
                    break;

                case 2:
                    System.out.println("\nEnter the ID of the
employee whose department you wish to update: ");
                    int updateID = sc.nextInt();
                    sc.nextLine();
                    System.out.println("Enter the new department
name: ");
                    String updateDepartment = sc.nextLine();
                    boolean findID = false;
                    for (Employee emp : list1) {
                        if (emp.empID == updateID) {
                            emp.department = updateDepartment;
                            findID = true;
                            break;
                        }
                    }
                    if (findID == true) {
                        System.out.println("\nDepartment updated
successfully");
                    } else {
                        System.out.println("\nEmployee with ID " +
updateID + " not found.");
                    }
                    break;

                case 3:
                    System.out.println("\nEnter the ID of employee
you wish to remove: ");
                    int removeID = sc.nextInt();
                    sc.nextLine();
                    Employee toRemove = null;
                    for (Employee emp : list1) {
                        if (emp.empID == removeID) {
                            toRemove = emp;
                            break;
                        }
                    }
                    if (toRemove != null) {
                        list1.remove(toRemove);
                        System.out.println("Employee removed
successfully!");
                    } else {
                        System.out.println("\nEmployee with ID " +
removeID + " not found.");
                    }
```

```
                    break;

                case 4:
                    System.out.println("List of employees:");
                    displayList(list1);
                    break;

                case 5:
                    System.out.println("Exit");
                    break;

                default:
                    System.out.println("Invalid choice");
                    break;
            }
        } while (choice != 5);

        sc.close();
    }
}
```

```
PS D:\CDAC\Java\AssignmentNo3> javac EmpArrayList.java
PS D:\CDAC\Java\AssignmentNo3> java EmpArrayList


Choose your option
1. Add new employee
2. Update employee department by ID
3. Remove employee by ID
4. Display list of all employees
5. To quit
Choice:
1


Enter the total number of employee details you want to add:
3

Enter employee name: Aileen
Enter employee ID: 1510
Enter department: Admin

Enter employee name: Dante
Enter employee ID: 1204
Enter department: IT

Enter employee name: Sameul
Enter employee ID: 2710
Enter department: Marketing
```

```
Choose your option
1. Add new employee
2. Update employee department by ID
3. Remove employee by ID
4. Display list of all employees
5. To quit
Choice:
2

Enter the ID of the employee whose department you wish to update:
1510
Enter the new department name:
Research

Department updated successfully
```

```
Choose your option
1. Add new employee
2. Update employee department by ID
3. Remove employee by ID
4. Display list of all employees
5. To quit
Choice:
4
List of employees:

Name: Aileen
Employee ID: 1510
Department: Research

Name: Dante
Employee ID: 1204
Department: IT

Name: Sameul
Employee ID: 2710
Department: Marketing
```

```
Choose your option
1. Add new employee
2. Update employee department by ID
3. Remove employee by ID
4. Display list of all employees
5. To quit
Choice:
3

Enter the ID of employee you wish to remove:
4564

Employee with ID 4564 not found.


Choose your option
1. Add new employee
2. Update employee department by ID
3. Remove employee by ID
4. Display list of all employees
5. To quit
Choice:
5
Exit
```

Question 5

You are developing a Java program to manage user inputs in a ticket booking system. Implement code to handle the following built-in exceptions:

1. **InputMismatchException**
2. **IllegalArgumentException**
3. **IndexOutOfBoundsException**

```java
import java.util.InputMismatchException;
import java.util.Scanner;

public class TicketBooking {

    public static void setAge(int age) {
        if (age < 18) {
            throw new IllegalArgumentException();
        }
    }

    public static void main(String[] args) {
        try {
```

```
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter phone number: ");
            sc.nextLong();
            sc.nextLine();
            System.out.println("Enter movie name: ");
            sc.nextLine();

            System.out.println("\nEnter age: ");
            int age = sc.nextInt();
            setAge(age);

            int maxSeat = 5;

            System.out.println("\nMaximum number of tickets that can
be booked: " + maxSeat);
            System.out.println("\nEnter number of tickets to be
booked: ");
            int max = sc.nextInt();
            sc.nextLine();

            if (max > maxSeat) {
                throw new ArrayIndexOutOfBoundsException("Cannot
book more than " + maxSeat + " tickets.");
            }

            String[] ticket = new String[maxSeat];
            int i = 0;
            while (i < max) {
                System.out.println("\nName " + (i + 1) + ": ");
                String name = sc.nextLine();
                ticket[i] = name;
                i++;
            }

            System.out.println("\nNames in ticket: ");
            for (int j = 0; j < max; j++) {
                System.out.println(ticket[j]);
            }

            sc.close();
        } catch (InputMismatchException ie) {
            System.out.println("\nException: " + ie);
        } catch (IllegalArgumentException ie) {
            System.out.println("\nException: " + ie);
        } catch (ArrayIndexOutOfBoundsException ae) {
            System.out.println("\nException: " + ae);
        }
    }
```

```
}
```

```
Enter phone number:
4459882763
Enter movie name:
Nezha 2

Enter age:
20

Maximum number of tickets that can be booked: 5

Enter number of tickets to be booked:
6

Exception: java.lang.ArrayIndexOutOfBoundsException: Cannot book more than 5 tickets.
```

Question 6

You are creating a Java program to represent different types of employees in a company. Implement a class named Employee with attributes for name, employeeID, and department. Provide a constructor to initialize these fields. Next, create a subclass named Manager with an additional attribute for numberOfTeams managed. The Manager class should have two constructors: one that accepts all fields, and another that accepts only name and employeeID, chaining to the superclass constructor. Include methods to validate input data and implement getter and setter methods for the attributes in the Employee class. Demonstrate the inheritance hierarchy by creating instances of both classes and displaying their details.

```
class Employee {
    private String name;
    private int employeeID;
    private String department;

    Employee(String name, int employeeID, String department) {
        if (name == null || name.isEmpty()) {
            System.out.println("\nName cannot be empty");
            this.name = "N/A";
        } else {
            this.name = name;
        }

        if (employeeID < 0) {
            System.out.println("\nEmployee ID cannot be empty");
            this.employeeID = 0;
        } else {
            this.employeeID = employeeID;
        }
```

```
        if (department == null || department.isEmpty()) {
            System.out.println("\nDepartment cannot be empty");
            this.department = "N/A";
        } else {
            this.department = department;
        }
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        if (name == null || name.isEmpty()) {
            System.out.println("\nName cannot be empty");
            this.name = "N/A";
        } else {
            this.name = name;
        }
    }

    public int getEmployeeID() {
        return employeeID;
    }

    public void setEmployeeID(int employeeID) {
        if (employeeID < 1) {
            System.out.println("\nEmployee ID cannot be empty");
            this.employeeID = 0;
        } else {
            this.employeeID = employeeID;
        }
    }

    public String getDepartment() {
        return department;
    }

    public void setDepartment(String department) {
        if (department == null || department.isEmpty()) {
            System.out.println("\nDepartment cannot be empty");
            this.department = "N/A";
        } else {
            this.department = department;
        }
    }
```

```
    public void display() {
        System.out.println("\nName: " + name + "\nEmployee ID: " +
employeeID + "\nDepartment: " + department);
    }
}

class Manager extends Employee {
    int numberOfTeams;

    Manager(String name, int employeeID, String department, int
numberOfTeams) {
        super(name, employeeID, department);
        if (numberOfTeams < 1) {
            System.out.println("\nNumber of teams cannot be empty");
            this.numberOfTeams = 0;
        } else {
            this.numberOfTeams = numberOfTeams;
        }
    }

    Manager(String name, int employeeID) {
        super(name, employeeID, "N/A");
    }

    public int getNumberOfTeams() {
        return numberOfTeams;
    }

    public void setNumberOfTeams(int numberOfTeams) {
        if (numberOfTeams < 1) {
            System.out.println("\nNumber of teams cannot be empty");
            this.numberOfTeams = 0;
        } else {
            this.numberOfTeams = numberOfTeams;
        }
    }

    public void display() {
        System.out.println(
                "\nName: " + getName() + "\nEmployee ID: " +
getEmployeeID() + "\nDepartment: " + getDepartment()
                        + "\nNumber of Teams managed: " +
getNumberOfTeams());
    }
}

public class Question6 {
    public static void main(String[] args) {
```

```
        Employee e1 = new Employee("Aliza", 1510, "Research");
        e1.display();
        Employee e2 = new Employee("Aliza", 0, "");
        e2.display();

        Manager m1 = new Manager("Dante", 1204);
        m1.display();
        Manager m2 = new Manager("Sebastian", 3004, "Design and
Development", 2);
        m2.display();
    }
}
```

```
PS D:\CDAC\Java\AssignmentNo3> javac Question6.java
PS D:\CDAC\Java\AssignmentNo3> java Question6

Name: Aliza
Employee ID: 1510
Department: Research

Department cannot be empty

Name: Aliza
Employee ID: 0
Department: N/A

Name: Dante
Employee ID: 1204
Department: N/A
Number of Teams managed: 0

Name: Sebastian
Employee ID: 3004
Department: Design and Development
Number of Teams managed: 2
```

Question 7

Develop a Java program for a simple banking system using an ArrayList to manage bank accounts. Each bank account should have attributes including the name of the account holder, account number, and initial balance. Implement the BankAccount class with methods to perform the following actions: (Using array list)

- **Allow users to deposit money into their account.**
- **Allow users to withdraw money from their account if they have sufficient balance.**
- **Display the current balance of the account.**

- **Enable users to transfer money from one account to another, provided they have sufficient balance.**

```
import java.util.*;

class Bank {
    String name;
    int accountNo;
    long initialBalance;

    Bank(String name, int accountNo, long initialBalance) {
        this.name = name;
        this.accountNo = accountNo;
        this.initialBalance = initialBalance;
    }

    public String toString() {
        return "\nName: " + name + "\nAccount Number: " + accountNo
+ "\nBalance: " + initialBalance;
    }
}

public class BankArrayList {
    static void displayAccounts(Collection<?> list) {
        for (Object obj : list) {
            System.out.println(obj);
        }
    }

    public static void main(String[] args) {
        String name;
        int accountNo;
        long initialBalance;
        int choice;

        Scanner sc = new Scanner(System.in);
        List<Bank> list1 = new ArrayList<>();

        list1.add(new Bank("Aliza", 225690, 20000));
        list1.add(new Bank("Dante", 225321, 200000));
        list1.add(new Bank("Vernon", 225753, 2500000));
        list1.add(new Bank("Aileen", 225009, 50000));
        list1.add(new Bank("Samuel", 225886, 650000));
        list1.add(new Bank("Jun", 225343, 4000000));
        list1.add(new Bank("Dino", 225105, 2350000));
        list1.add(new Bank("Woozi", 225014, 5600000));

        // System.out.println("List of all Bank Account Details");
        // displayAccounts(list1);
```

```
        do {
            System.out.println("\n\nChoose your option");
            System.out.println("1. Deposit money by Account
Number");
            System.out.println("2. Withdraw money by Account
Number");
            System.out.println("3. View account details by Account
Number");
            System.out.println("4. Transfer money by Account
Number");
            System.out.println("5. To quit");
            System.out.println("Choice: ");
            choice = sc.nextInt();
            sc.nextLine();

            switch (choice) {
                case 1:
                    System.out.println("\nEnter the account number
to deposit money: ");
                    int depositAccount = sc.nextInt();
                    sc.nextLine();
                    System.out.println("\nEnter the amount to be
deposited: ");
                    long depositAmount = sc.nextLong();
                    sc.nextLine();

                    boolean found1 = false;
                    for (Bank b : list1) {
                        if (b.accountNo == depositAccount) {
                            b.initialBalance = b.initialBalance +
depositAmount;
                            System.out.println("\nDeposit
successful!");
                            found1 = true;
                            break;
                        }
                    }
                    if (found1 == false) {
                        System.out.println("\nBank account with " +
depositAccount + " not found.");
                    }
                    break;

                case 2:
                    System.out.println("\nEnter the account number
to withdraw money: ");
                    int withdrawAccount = sc.nextInt();
```

```
                    sc.nextLine();
                    System.out.println("\nEnter the amount to be
withdrawn ");
                    long withdrawAmount = sc.nextLong();
                    sc.nextLine();

                    boolean found2 = false;
                    for (Bank b : list1) {
                        if (b.accountNo == withdrawAccount) {
                            found2 = true;
                            if (b.initialBalance > withdrawAmount) {
                                b.initialBalance = b.initialBalance
- withdrawAmount;
                                System.out.println("\nWithdraw
successful!");
                            } else {
                                System.out.println("Insufficient
balance to withdraw money.");
                            }
                            break;
                        }
                    }
                    if (found2 == false) {
                        System.out.println("\nBank account with " +
withdrawAccount + " not found.");
                    }
                    break;

                case 3:
                    System.out.println("\nEnter the account number
to view balance: ");
                    int viewAccount = sc.nextInt();
                    sc.nextLine();

                    boolean found3 = false;
                    for (Bank b : list1) {
                        if (b.accountNo == viewAccount) {
                            System.out.println("\nAccount Details:
");
                            System.out.println(b);
                            found3 = true;
                            break;
                        }
                    }
                    if (found3 == false) {
                        System.out.println("\nBank account with " +
viewAccount + " not found.");
                    }
```

```
                    break;

                case 4:
                    System.out.println("\nEnter the account number
from which to transfer money: ");
                    int fromAccount = sc.nextInt();
                    sc.nextLine();
                    System.out.println("\nEnter the account number
to which you want to transfer money: ");
                    int toAccount = sc.nextInt();
                    sc.nextLine();
                    System.out.println("\nEnter the amount to be
transfered: ");
                    long transferAmount = sc.nextLong();
                    sc.nextLine();

                    boolean found4 = false;
                    for (Bank b : list1) {
                        if (b.accountNo == fromAccount) {
                            found4 = true;
                            if (b.initialBalance > transferAmount) {
                                b.initialBalance = b.initialBalance
- transferAmount;
                                System.out.println("\nTransfer
successful!");
                                System.out.println("Money
transferred to bank account " + toAccount);
                            } else {
                                System.out.println("Insufficient
balance to transfer money.");
                            }
                            break;
                        }
                    }
                    if (found4 == false) {
                        System.out.println("\nBank account with " +
fromAccount + " not found.");
                    }
                    break;

                case 5:
                    System.out.println("Exit");
                    break;

                default:
                    System.out.println("Invalid choice");
                    break;
            }
```

```
        } while (choice != 5);
        sc.close();
    }
}
```

```
PS D:\CDAC\Java\AssignmentNo3> javac BankArrayList.java
PS D:\CDAC\Java\AssignmentNo3> java BankArrayList


Choose your option
1. Deposit money by Account Number
2. Withdraw money by Account Number
3. View account details by Account Number
4. Transfer money by Account Number
5. To quit
Choice:
1

Enter the account number to deposit money:
225105

Enter the amount to be deposited:
50000

Deposit successful!
```

```
Choose your option
1. Deposit money by Account Number
2. Withdraw money by Account Number
3. View account details by Account Number
4. Transfer money by Account Number
5. To quit
Choice:
3

Enter the account number to view balance:
225105

Account Details:

Name: Dino
Account Number: 225105
Balance: 2400000
```

```
Choose your option
1. Deposit money by Account Number
2. Withdraw money by Account Number
3. View account details by Account Number
4. Transfer money by Account Number
5. To quit
Choice:
2
```

```
Enter the account number to withdraw money:
225009

Enter the amount to be withdrawn
300000
Insufficient balance to withdraw money.


Choose your option
1. Deposit money by Account Number
2. Withdraw money by Account Number
3. View account details by Account Number
4. Transfer money by Account Number
5. To quit
Choice:
4

Enter the account number from which to transfer money:
225014

Enter the account number to which you want to transfer money:
779113

Enter the amount to be transfered:
60000

Transfer successful!
Money transferred to bank account 779113
```

```
Choose your option
1. Deposit money by Account Number
2. Withdraw money by Account Number
3. View account details by Account Number
4. Transfer money by Account Number
5. To quit
Choice:
3

Enter the account number to view balance:
225014

Account Details:

Name: Woozi
Account Number: 225014
Balance: 5540000
```

```
Choose your option
1. Deposit money by Account Number
2. Withdraw money by Account Number
3. View account details by Account Number
4. Transfer money by Account Number
5. To quit
Choice:
5
Exit
```

Question 8

You're developing error handling for a Java program that manages payments in an e-commerce application. Write code to handle the following three, specific payment gateway errors:

- **Timeout error: Implement exception handling to catch situations where the payment gateway response times out.**
- **Invalid card details: Handle exceptions arising from attempts to process payments with invalid card information.**
- **Insufficient funds: Implement exception handling to manage cases where users attempt to make payments without sufficient funds in their account.**

```java
import java.util.Scanner;

class TimeOutException extends Exception {
    TimeOutException(String msg) {
        super(msg);
    }
```

```
}

class InvalidCardException extends Exception {
    InvalidCardException(String msg) {
        super(msg);
    }
}

class InsufficientFundException extends Exception {
    InsufficientFundException(String msg) {
        super(msg);
    }
}

public class ECommerce {
    public static void main(String[] args) {
        try {
            long cardNo;
            int cvv, expiryMonth, expiryYear;
            double amount, balance;
            long timeOut = 2L * 60000 * 1000000;
            Scanner sc = new Scanner(System.in);

            System.out.println("Please enter card details for
processing your payment -----");

            long start = System.nanoTime();

            System.out.println("Enter card number: ");
            cardNo = sc.nextLong();
            System.out.println("Enter CVV: ");
            cvv = sc.nextInt();
            System.out.println("Enter card expiry month: ");
            expiryMonth = sc.nextInt();
            System.out.println("Enter card expiry year: ");
            expiryYear = sc.nextInt();
            if (cardNo < 0000000000000001 || cardNo >
9999999999999999l) {
                throw new InvalidCardException("Card details are
invalid.");
            } else if (cvv < 001 || cvv > 999) {
                throw new InvalidCardException("Card details are
invalid.");
            } else if (expiryMonth < 01 || expiryMonth > 12) {
                throw new InvalidCardException("Card details are
invalid.");
            } else if (expiryYear < 25) {
```

```
                throw new InvalidCardException("Card details are
invalid.");
            }

            System.out.println("\nEnter balance: ");
            balance = sc.nextInt();
            System.out.println("Enter amount: ");
            amount = sc.nextInt();
            if (balance < amount) {
                throw new InsufficientFundException("You don't have
sufficient balance to complete the payment.");
            }

            long finish = System.nanoTime();
            long timeElapsed = finish - start;

            if (timeElapsed > timeOut) {
                throw new TimeOutException("Payment gateway timeout.
Took too long to complete payment.");
            }

            System.out.println("Payment done!");

        } catch (TimeOutException te) {
            System.out.println(te);
        } catch (InvalidCardException ie) {
            System.out.println(ie);
        } catch (InsufficientFundException fe) {
            System.out.println(fe);
        }
    }
}
```

```
PS D:\CDAC\Java\AssignmentNo3> javac ECommerce.java
PS D:\CDAC\Java\AssignmentNo3> java ECommerce
Please enter card details for processing your payment -----
Enter card number:
1234789045672684
Enter CVV:
457
Enter card expiry month:
08
Enter card expiry year:
28

Enter balance:
2000000
Enter amount:
25000
Payment done!
```

```
PS D:\CDAC\Java\AssignmentNo3> javac ECommerce.java
PS D:\CDAC\Java\AssignmentNo3> java ECommerce
Please enter card details for processing your payment -----
Enter card number:
1234567856782345
Enter CVV:
123
Enter card expiry month:
12
Enter card expiry year:
24
InvalidCardException: Card details are invalid.
```

```
PS D:\CDAC\Java\AssignmentNo3> javac ECommerce.java
PS D:\CDAC\Java\AssignmentNo3> java ECommerce
Please enter card details for processing your payment -----
Enter card number:
1234000066883355
Enter CVV:
158
Enter card expiry month:
11
Enter card expiry year:
30

Enter balance:
55000
Enter amount:
5000000
InsufficientFundException: You don't have sufficient balance to complete the payment.
```

```
PS D:\CDAC\Java\AssignmentNo3> javac ECommerce.java
PS D:\CDAC\Java\AssignmentNo3> java ECommerce
Please enter card details for processing your payment -----
Enter card number:
1234123456785678
Enter CVV:
142
Enter card expiry month:
11
Enter card expiry year:
30

Enter balance:
250000
Enter amount:
2789
TimeOutException: Payment gateway timeout. Took too long to complete payment.
```

Question 9

You're developing a Java program to manage individuals within an educational institution. Create a class named "Person" with attributes for the name and age of an individual. Implement a constructor that accepts both fields as arguments. Next, design a subclass named "Student" with an additional attribute for the grade level. Provide two constructors for the Student class: one that accepts all fields including name, age, and grade level, and another that accepts only the name and age, chaining to the superclass constructor. Ensure that your program demonstrates proper inheritance principles, and validate the constructors to ensure valid data is provided during object instantiation, also implement appropriate getter and setter methods to access and modify the attributes of Person class.

```java
class Person {
    private String name;
    private int age;

    Person(String name, int age) {
        if (name == null || name.isEmpty()) {
            System.out.println("\nName cannot be null");
        } else {
            this.name = name;
        }
        if (age < 6 || age > 120) {
            System.out.println("Invalid age");
        } else {
            this.age = age;
        }
    }

    public String getName() {
        return name;
```

```
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public void display() {
        System.out.println("Name: " + name + "\nAge: " + age +
"\n");
    }
}

class Student extends Person {
    double grade;

    Student(String name, int age, double grade) {
        super(name, age);
        if (grade < 0 || grade > 10) {
            this.grade = 0;
        } else {
            this.grade = grade;
        }
    }

    Student(String name, int age) {
        super(name, age);
    }

    public double getGrade() {
        return grade;
    }

    public void setGrade(int grade) {
        this.grade = grade;
    }

    public void display() {
        System.out.println("\nName: " + getName() + "\nAge: " +
getAge() + "\nGrade: " + getGrade() + "\n");
    }
```

```
}

public class Question9 {
    public static void main(String[] args) {
        Person p = new Person("Aliza", 22);
        p.display();

        Student s1 = new Student(null, 0);
        s1.display();

        Student s2 = new Student("Dante", 24, 9.5);
        s2.display();
    }
}
```

```
PS D:\CDAC\Java\AssignmentNo3> javac Question9.java
PS D:\CDAC\Java\AssignmentNo3> java Question9
Name: Aliza
Age: 22


Name cannot be null
Invalid age

Name: null
Age: 0
Grade: 0.0


Name: Dante
Age: 24
Grade: 9.5
```

Question 10

Create a class Student with:

- **int studentId, String name, double grade.**
- **A constructor to initialize these fields.**
- **Methods:**
  updateGrade(double newGrade): Updates the grade, but should not accept negative values (handle using exception handling).
  display(): Prints student details.

```
import java.util.Scanner;

class NegativeGradeException extends Exception {
    NegativeGradeException(String msg) {
```

```
        super(msg);
    }
}

class GradeOutOfRangeException extends Exception {
    GradeOutOfRangeException(String msg) {
        super(msg);
    }
}

class Student {
    int studentId;
    String name;
    double grade;

    Student(int studentId, String name, double grade) {
        this.studentId = studentId;
        this.name = name;
        this.grade = grade;
    }

    public void updateGrade(double grade) throws
NegativeGradeException, GradeOutOfRangeException {
        if (grade < 0) {
            throw new NegativeGradeException("Negative grade not
accepted");
        } else if (grade > 10) {
            throw new GradeOutOfRangeException("Grades must be
between 0 - 10");
        }
        this.grade = grade;

    }

    public void display() {
        System.out.println("ID: " + studentId + "\nName: " + name +
"\nGrade: " + grade);
    }
}

public class Question10 {
    public static void main(String[] args) {
        try {
            Scanner sc = new Scanner(System.in);
            Student s = new Student(1510, "Aliza", 8.5);
            s.display();
            System.out.println("\nEnter your updated grades: ");
            double updateGrade = sc.nextDouble();
```

```
            s.updateGrade(updateGrade);
            System.out.println("\nDetails after update: ");
            s.display();

        } catch (NegativeGradeException ne) {
            System.out.println(ne);
        } catch (GradeOutOfRangeException ge) {
            System.out.println(ge);
        }
    }
}
```

```
PS D:\CDAC\Java\AssignmentNo3> javac Question10.java
PS D:\CDAC\Java\AssignmentNo3> java Question10
ID: 1510
Name: Aliza
Grade: 8.5

Enter your updated grades:
9.8

Details after update:
ID: 1510
Name: Aliza
Grade: 9.8
```

```
PS D:\CDAC\Java\AssignmentNo3> javac Question10.java
PS D:\CDAC\Java\AssignmentNo3> java Question10
ID: 1510
Name: Aliza
Grade: 8.5

Enter your updated grades:
15
GradeOutOfRangeException: Grades must be between 0 - 10
PS D:\CDAC\Java\AssignmentNo3> java Question10
ID: 1510
Name: Aliza
Grade: 8.5

Enter your updated grades:
-2.5
NegativeGradeException: Negative grade not accepted
```

Question 11

Write a Java program that reads a string from the user and attempts to convert it to an integer using Integer.parseInt(). If the input is not a valid integer, handle the NumberFormatException. Additionally, handle NullPointerException if the input is null. Use a finally block to print "Conversion attempt completed."

```java
import java.util.Scanner;

public class Question11 {
    public static void main(String[] args) {
        try {
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter any string: ");
            String str = sc.nextLine();
            int intVal = Integer.parseInt(str);
            System.out.println("Converted integer: " + intVal);

            System.out.println();

            String str2 = null;
            System.out.println("String 2 length: " + str2.length());

        } catch (NumberFormatException ne) {
            System.out.println(ne);
        } catch (NullPointerException pe) {
            System.out.println(pe);
        } finally {
            System.out.println("\nConversion attempt completed!");
        }
    }
}
```

```
PS D:\CDAC\Java\AssignmentNo3> javac Question11.java
PS D:\CDAC\Java\AssignmentNo3> java Question11
Enter any string:
12345678
Converted integer: 12345678

java.lang.NullPointerException: Cannot invoke "String.length()" because "<local4>" is null

Conversion attempt completed!
```

Question 12

Patient Management System

You are managing a patient database for a hospital. Each patient has a unique patient ID, a name, a diagnosis, and the number of days admitted. You need to implement a solution using appropriate Java collection classes to efficiently perform the following operations:

a. **Add a new patient to the database.**
b. **Remove a patient from the database.**
c. **Find all patients with a specific diagnosis.**
d. **Find all patients admitted for more than a given number of days.**

```
import java.util.*;

class Patient {
    int patientID;
    String name;
    String diagnosis;
    int noOfDaysAdmitted;

    Patient(int patientID, String name, String diagnosis, int
noOfDaysAdmitted) {
        this.patientID = patientID;
        this.name = name;
        this.diagnosis = diagnosis;
        this.noOfDaysAdmitted = noOfDaysAdmitted;
    }

    public String toString() {
        return "\nPatient ID: " + patientID + "\nName: " + name +
"\nDiagnosis: " + diagnosis
                + "\nNumber of days admitted: "
                + noOfDaysAdmitted;
    }
}

public class PatientArrayList {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Patient> list1 = new ArrayList<>();
        int choice;

        do {
            System.out.println("\n\nChoose your option");
            System.out.println("1. Add patient");
            System.out.println("2. Remove patient");
            System.out.println("3. Find all patient with specific
diagnosis");
```

```
            System.out.println("4. Find all patient admitted for
more than given number of days");
            System.out.println("5. To quit");
            System.out.println("Choice: ");
            choice = sc.nextInt();
            sc.nextLine();

            switch (choice) {
                case 1:
                    System.out.println("\n\nEnter the total number
of patient details you want to add: ");
                    int num = sc.nextInt();
                    sc.nextLine();
                    for (int i = 0; i < num; i++) {
                        System.out.print("\nEnter patient ID: ");
                        int patientID = sc.nextInt();
                        sc.nextLine();

                        System.out.print("Enter patient name: ");
                        String name = sc.nextLine();

                        System.out.print("Enter diagnosis: ");
                        String diagnosis = sc.nextLine();

                        System.out.print("Enter number of days
admitted: ");
                        int daysAdmitted = sc.nextInt();
                        sc.nextLine();

                        list1.add(new Patient(patientID, name,
diagnosis, daysAdmitted));
                    }
                    break;

                case 2:
                    System.out.println("\nEnter the ID of patient
you wish to remove: ");
                    int removeID = sc.nextInt();
                    sc.nextLine();
                    Patient toRemove = null;
                    for (Patient pat : list1) {
                        if (pat.patientID == removeID) {
                            toRemove = pat;
                            break;
                        }
                    }
                    if (toRemove != null) {
                        list1.remove(toRemove);
```

```
                        System.out.println("Patient removed
successfully!");
                    } else {
                        System.out.println("\nPatient with ID " +
removeID + " not found.");
                    }
                    break;

                case 3:
                    System.out.println("\nEnter the diagnosis to
find all patients: ");
                    String findDiagnosis = sc.nextLine();
                    boolean findPatient = false;
                    for (Patient pat : list1) {
                        if
(pat.diagnosis.equalsIgnoreCase(findDiagnosis)) {
                            System.out.println(pat);
                            findPatient = true;
                        }
                    }
                    if (findPatient == false) {
                        System.out.println("\nPatient with " +
findDiagnosis + " not found.");
                    }
                    break;

                case 4:
                    System.out.println("\nEnter number of days to
filter patients: ");
                    int days = sc.nextInt();
                    sc.nextLine();
                    boolean patientAdmitted = false;
                    System.out.println("\nPatient admitted for more
than " + days + " days");
                    for (Patient pat : list1) {
                        if (pat.noOfDaysAdmitted > days) {
                            System.out.println(pat);
                            patientAdmitted = true;
                        }
                    }
                    if (patientAdmitted == false) {
                        System.out.println("\nNo patient admitted
more than " + days + " days");
                    }
                    break;

                case 5:
                    System.out.println("Exit");
```

```
                    break;

                default:
                    System.out.println("Invalid choice");
                    break;
            }
        } while (choice != 5);
        sc.close();
    }
}
```

```
PS D:\CDAC\Java\AssignmentNo3> javac PatientArrayList.java
PS D:\CDAC\Java\AssignmentNo3> java PatientArrayList


Choose your option
1. Add patient
2. Remove patient
3. Find all patient with specific diagnosis
4. FInd all patient admitted for more than given number of days
5. To quit
Choice:
1


Enter the total number of patient details you want to add:
5
```

```
Enter patient ID: 1102
Enter patient name: Alice
Enter diagnosis: Food Poisoning
Enter number of days admitted: 2

Enter patient ID: 4539
Enter patient name: Bob
Enter diagnosis: Jaundice
Enter number of days admitted: 10

Enter patient ID: 3376
Enter patient name: Charles
Enter diagnosis: Typhoid
Enter number of days admitted: 6
```

```
Enter patient ID: 8807
Enter patient name: Dolly
Enter diagnosis: Dengue
Enter number of days admitted: 20

Enter patient ID: 4460
Enter patient name: Evan
Enter diagnosis: Jaundice
Enter number of days admitted: 8


Choose your option
1. Add patient
2. Remove patient
3. Find all patient with specific diagnosis
4. FInd all patient admitted for more than given number of days
5. To quit
Choice:
2

Enter the ID of patient you wish to remove:
8807
Patient removed successfully!
```

```
Choose your option
1. Add patient
2. Remove patient
3. Find all patient with specific diagnosis
4. FInd all patient admitted for more than given number of days
5. To quit
Choice:
3

Enter the diagnosis to find all patients:
Jaundice

Patient ID: 4539
Name: Bob
Diagnosis: Jaundice
Number of days admitted: 10

Patient ID: 4460
Name: Evan
Diagnosis: Jaundice
Number of days admitted: 8
```

```
Choose your option
1. Add patient
2. Remove patient
3. Find all patient with specific diagnosis
4. FInd all patient admitted for more than given number of days
5. To quit
Choice:
4

Enter number of days to filter patients:
7

Patient admitted for more than 7 days

Patient ID: 4539
Name: Bob
Diagnosis: Jaundice
Number of days admitted: 10

Patient ID: 4460
Name: Evan
Diagnosis: Jaundice
Number of days admitted: 8
```

```
Choose your option
1. Add patient
2. Remove patient
3. Find all patient with specific diagnosis
4. FInd all patient admitted for more than given number of days
5. To quit
Choice:
5
Exit
```