Problem: Develop a library management system to handle book lending operations using generic classes, collections, file I/O, and exception handling.
1. Create a Book class with the following fields: ○ ISBN (String) ○ Title (String) ○ Author (String) ○ IsAvailable (boolean)
2. Create a Member class with the following fields: ○ Member ID (String) ○ Name(String) ○ Borrowed Books (ArrayList)
3. Create a generic Library class where T represents any type of member. The class should: ○ Use acollection (HashMap>) to store books by ISBN and members by their ID. ○ Implement methods to lend a book to a member, return a book, and display available books. ○ Add file I/O functionality to load and save book and member data from/to files. ■ Books file format: ISBN, Title, Author, IsAvailable ■ Members file format: Member ID, Name, Borrowed Books (ISBNs)
4. Handle exceptions for invalid operations, such as borrowing an unavailable book or returning a book not borrowed by the member. Requirements: ● void loadBooksFromFile(String fileName)– Load book data from a file. ● void loadMembersFromFile(String fileName)– Load member data from a file. ● void saveLibraryData(String booksFileName, String membersFileName)– Save updated book and member data to separate files. In the main method: ● Load book and member data from files. ● Lend at least 2 books to a member, return one, and display the updated availability. ● Save the updated data back to files.
5. Create a readme file for the system
Ans
Code Implementation

1. Book Class

```
public class Book {
    private String isbn;
    private String title;
    private String author;
    private boolean isAvailable;
    public Book(String isbn, String title, String author, boolean
isAvailable) {
        this.isbn = isbn;
        this.title = title;
        this.author = author;
        this.isAvailable = isAvailable;
    }
    public String getIsbn() {
        return isbn;
    }

    public String getTitle() {
        return title;
```

```
    }
    public String getAuthor() {
        return author;
    }

    public boolean isAvailable() {
        return isAvailable;
    }
    public void setAvailable(boolean available) {
    isAvailable = available;
    }

    @Override
    public String toString() {
        return isbn + ", " + title + ", " + author + ", " +
(isAvailable ? "Available" : "Not Available");
    }
}
```

Member Class

```
java
Copy code
import java.util.ArrayList;

public class Member {
    private String memberId;
    private String name;
    private ArrayList<Book> borrowedBooks;

    public Member(String memberId, String name) {
        this.memberId = memberId;
        this.name = name;
        this.borrowedBooks = new ArrayList<>();
    }

    public String getMemberId() {
        return memberId;
    }
```

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

    public ArrayList<Book> getBorrowedBooks() {
        return borrowedBooks;
    }

    public void borrowBook(Book book) {
        borrowedBooks.add(book);
    }

    public void returnBook(Book book) {
        borrowedBooks.remove(book);
    }
}
```

Library class

```
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;

public class Library<T> {
    private HashMap<String, Book> books = new HashMap<>();
    private HashMap<String, T> members = new HashMap<>();

    public void addBook(Book book) {
        books.put(book.getIsbn(), book);
    }

    public void addMember(String id, T member) {
        members.put(id, member);
    }

    public void lendBook(String isbn, String memberId) throws
Exception {
        Book book = books.get(isbn);
        Member member = (Member) members.get(memberId);
```

```
        if (book == null || member == null) throw new Exception("Book
or Member not found!");
        if (!book.isAvailable()) throw new Exception("Book is not
available!");

        book.setAvailable(false);
        member.borrowBook(book);
    }

    public void returnBook(String isbn, String memberId) throws
Exception {
        Book book = books.get(isbn);
        Member member = (Member) members.get(memberId);

        if (book == null || member == null ||
!member.getBorrowedBooks().contains(book))
            throw new Exception("Invalid return operation!");

        book.setAvailable(true);
        member.returnBook(book);
    }

    public void displayAvailableBooks() {
        books.values().stream()
                .filter(Book::isAvailable)
                .forEach(System.out::println);
    }

    public void loadBooksFromFile(String fileName) throws IOException
{
        try (BufferedReader br = new BufferedReader(new
FileReader(fileName))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] parts = line.split(", ");
                addBook(new Book(parts[0], parts[1], parts[2],
Boolean.parseBoolean(parts[3])));
            }
```

```
        }
    }

    public void loadMembersFromFile(String fileName) throws
IOException {
        try (BufferedReader br = new BufferedReader(new
FileReader(fileName))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] parts = line.split(", ");
                Member member = new Member(parts[0], parts[1]);
                String[] borrowedBooks = parts[2].split(";");
                for (String isbn : borrowedBooks) {
                    if (books.containsKey(isbn)) {
                        member.borrowBook(books.get(isbn));
                    }
                }
                addMember(parts[0], (T) member);
            }
        }
    }

    public void saveLibraryData(String booksFileName, String
membersFileName) throws IOException {
        try (BufferedWriter bw = new BufferedWriter(new
FileWriter(booksFileName))) {
            for (Book book : books.values()) {
                bw.write(book.toString());
                bw.newLine();
            }
        }

        try (BufferedWriter bw = new BufferedWriter(new
FileWriter(membersFileName))) {
            for (T memberObj : members.values()) {
                Member member = (Member) memberObj;
                String borrowedIsbns = String.join(";",
member.getBorrowedBooks().stream()
```

```
                        .map(Book::getIsbn)
                        .toArray(String[]::new));
                bw.write(member.getMemberId() + ", " +
member.getName() + ", " + borrowedIsbns);
                bw.newLine();
            }
        }
    }
}
 LibraryApp (Main Method)
public class LibraryApp {
    public static void main(String[] args) {
        try {
            Library<Member> library = new Library<>();

            library.loadBooksFromFile("books.txt");
            library.loadMembersFromFile("members.txt");

            library.lendBook("ISBN123", "M002");
            library.lendBook("ISBN456", "M002");

            library.returnBook("ISBN123", "M002");
            library.displayAvailableBooks();

            library.saveLibraryData("updated_books.txt",
"updated_members.txt");
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }
}
```

README: Library Management System

Library Management System

Description

A Java-based system to manage library book lending operations, including member management, book availability, and data persistence using file I/O.

Features

- Display all available books in the library.
- Lend a book to a member by ISBN.
- Return a borrowed book by ISBN.
- Add new books and members to the system.
- Save and load library data to/from files.
- Handle exceptions for invalid operations (e.g., unavailable books, invalid returns).

How to Compile and Run

1. Save all the provided class files (Book.java, Member.java, Library.java, and LibraryApp.java) in the same directory.

Compile:
bash
Copy code
javac *.java

2.

Run:
bash
Copy code
java LibraryApp

3.

Usage

1. The program will load books and members from specified files (books.txt and members.txt).
2. Follow the menu options to:

- Lend books to members.
- Return books.
- Display available books.

3. Updated data will be saved to updated_books.txt and updated_members.txt upon exiting.

Example Input Format

Books File (books.txt):

vbnet
Copy code
ISBN123, The Great Gatsby, F. Scott Fitzgerald, true
ISBN456, 1984, George Orwell, true
ISBN789, To Kill a Mockingbird, Harper Lee, false

Members File (members.txt):

Copy code
M001, John Doe, ISBN789
M002, Jane Smith,

Example Console Interaction

mathematica
Copy code
Enter ISBN of the book to lend: ISBN123
Enter Member ID: M002

Console Output:

css
Copy code
Book successfully lent to Member ID M002.

Authors

- **Code by Abhigyan and Arnav :)**

Car Booking system
Code Implementation

1. Car Class

java
Copy code

```
public class Car {
    private String licensePlate;
    private String model;
    private boolean isAvailable;

    public Car(String licensePlate, String model, boolean isAvailable)
{
        this.licensePlate = licensePlate;
        this.model = model;
        this.isAvailable = isAvailable;
    }

    public String getLicensePlate() {
        return licensePlate;
    }

    public String getModel() {
        return model;
    }

    public boolean isAvailable() {
        return isAvailable;
    }

    public void setAvailable(boolean available) {
        isAvailable = available;
    }
```

```
    @Override
    public String toString() {
        return licensePlate + ", " + model + ", " + (isAvailable ?
"Available" : "Not Available");
    }
}
```

CarRentalSystem Class

```
java
Copy code
import java.util.ArrayList;

public class CarRentalSystem {
    private ArrayList<Car> cars;

    public CarRentalSystem() {
        cars = new ArrayList<>();
    }

    public void addCar(Car car) {
        cars.add(car);
    }

    public void displayAllCars() {
        for (Car car : cars) {
            System.out.println(car);
        }
    }

    public void displayAvailableCars() {
        for (Car car : cars) {
            if (car.isAvailable()) {
                System.out.println(car);
            }
        }
    }

    public void rentCar(String licensePlate) {
```

```
        for (Car car : cars) {
            if (car.getLicensePlate().equalsIgnoreCase(licensePlate))
{
                if (car.isAvailable()) {
                    car.setAvailable(false);
                    System.out.println("Car " + licensePlate + " has
been rented.");
                    return;
                } else {
                    System.out.println("Car " + licensePlate + " is
already rented.");
                    return;
                }
            }
        }
        System.out.println("Car with license plate " + licensePlate +
" not found.");
    }

    public void returnCar(String licensePlate) {
        for (Car car : cars) {
            if (car.getLicensePlate().equalsIgnoreCase(licensePlate))
{
                if (!car.isAvailable()) {
                    car.setAvailable(true);
                    System.out.println("Car " + licensePlate + " has
been returned.");
                    return;
                } else {
                    System.out.println("Car " + licensePlate + " was
not rented.");
                    return;
                }
            }
        }
        System.out.println("Car with license plate " + licensePlate +
" not found.");
    }
```

```
}
```

CarRentalApp (Main Method)

```
java
Copy code
import java.util.Scanner;

public class CarRentalApp {
    public static void main(String[] args) {
        CarRentalSystem carRentalSystem = new CarRentalSystem();
        Scanner scanner = new Scanner(System.in);

        // Adding some initial cars
        carRentalSystem.addCar(new Car("MH1234", "Toyota Camry",
true));
        carRentalSystem.addCar(new Car("DL5678", "Honda Civic",
true));
        carRentalSystem.addCar(new Car("KA9101", "Ford Mustang",
false));

        while (true) {
            System.out.println("\n--- Car Rental System ---");
            System.out.println("1. Display all cars");
            System.out.println("2. Rent a car");
            System.out.println("3. Display available cars");
            System.out.println("4. Return a car");
            System.out.println("5. Add a new car");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    carRentalSystem.displayAllCars();
                    break;
                case 2:
```

```
                    System.out.print("Enter the license plate of the
car to rent: ");
                    String rentLicense = scanner.nextLine();
                    carRentalSystem.rentCar(rentLicense);
                    break;
                case 3:
                    carRentalSystem.displayAvailableCars();
                    break;
                case 4:
                    System.out.print("Enter the license plate of the
car to return: ");
                    String returnLicense = scanner.nextLine();
                    carRentalSystem.returnCar(returnLicense);
                    break;
                case 5:
                    System.out.print("Enter the license plate of the
new car: ");
                    String newLicense = scanner.nextLine();
                    System.out.print("Enter the model of the new car:
");
                    String newModel = scanner.nextLine();
                    carRentalSystem.addCar(new Car(newLicense,
newModel, true));
                    System.out.println("New car added successfully.");
                    break;
                case 6:
                    System.out.println("Exiting the system.
Goodbye!");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice. Please try
again.");
            }
        }
    }
}
```

README: Car Booking System

Car Booking System

Description

A Java-based application to manage car bookings, allowing users to view, rent, return, and add cars.

Features

- Display all cars in the system.
- Rent a car by its license plate.
- Return a rented car using its license plate.
- Display all available cars for rent.
- Add new cars to the system.

How to Compile and Run

1. Save all the provided class files (Car.java, CarRentalSystem.java, and CarRentalApp.java) in the same directory.

Compile:
bash
Copy code
javac *.java

2.

Run:
bash
Copy code
java CarRentalApp

3.

Usage

- Follow the menu options displayed in the console:

1. Display all cars.
2. Rent a car.
3. Display available cars.
4. Return a car.
5. Add a new car.
6. Exit the program.

Example Input Format

When prompted:

```
css
Copy code
Enter the license plate of the car to rent: MH1234
```

The system processes the input and updates the car's availability.

Authors

- Code by Abhigyan and Arnav :)

  Question 2
  Problem: Design and implement a multithreaded stock trading simulation system, with real-time trading updates and file-based transaction logging.
  1. Create a Stock class with the following fields: ○ Stock Symbol (String) ○ Company Name (String) ○ Price (double)
  2. Methods: ○ void updatePrice(double newPrice)– Update the price of the stock. ○ void logTransaction(String logFileName, String transactionDetails)– Log buy/sell transactions to a file, including stock symbol, price, and timestamp. 3. Create a Trader class that implements Runnable. Each trader will: ○ Buy and sell stocks in real-time. Each transaction should be logged to a file. ○ Simulate buying a stock by lowering its availability and selling it by increasing its availability. ○ Ensure synchronized access to stock price updates to avoid race conditions.
  4. Create a StockMarket class to manage a collection of stocks (HashMap). It should: ○ Provide methods for traders to buy and

sell stocks. ○ Use multithreading to simulate multiple traders trading concurrently. ○ Log all transactions in real-time to a file, ensuring that the logs are properly synchronized.
5. Ensure proper synchronization between multiple traders so that stock prices and transaction logs are consistent and free from race conditions. Requirements: ● Multithreading: Simulate at least 3 traders buying and selling stocks concurrently. ● File I/O: Log all transactions to a file with proper synchronization using file streams (BufferedWriter, etc.). ● Synchronization: Ensure thread-safe access to stock prices and the transaction log. In the main method: ● Create at least 5 different stocks and simulate trading by 3 traders. ● Verify that all stock transactions are correctly logged to a file.
Answer

1. Stock Class

java

Copy code

```
import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;


public class Stock {

    private String stockSymbol;

    private String companyName;

    private double price;
```

```
    public Stock(String stockSymbol, String companyName, double price)
{
        this.stockSymbol = stockSymbol;
        this.companyName = companyName;
        this.price = price;
    }

    public synchronized void updatePrice(double newPrice) {
        this.price = newPrice;
    }

    public synchronized double getPrice() {
        return price;
    }

    public String getStockSymbol() {
        return stockSymbol;
    }

    public String getCompanyName() {
        return companyName;
    }
```

```
    public void logTransaction(String logFileName, String
transactionDetails) {

        synchronized (Stock.class) {

            try (BufferedWriter writer = new BufferedWriter(new
FileWriter(logFileName, true))) {

                String timestamp =
LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss"));

                writer.write(timestamp + " - " + transactionDetails);

                writer.newLine();

            } catch (IOException e) {

                System.err.println("Error writing to log file: " +
e.getMessage());

            }

        }

    }

}
```

2. Trader Class

java

Copy code

```
import java.util.Random;


public class Trader implements Runnable {

    private String name;
```

```
    private StockMarket stockMarket;

    private String logFileName;


    public Trader(String name, StockMarket stockMarket, String
logFileName) {

        this.name = name;

        this.stockMarket = stockMarket;

        this.logFileName = logFileName;

    }


    @Override

    public void run() {

        Random random = new Random();

        String[] stockSymbols = stockMarket.getStockSymbols();


        for (int i = 0; i < 10; i++) {

            String stockSymbol =
stockSymbols[random.nextInt(stockSymbols.length)];

            double transactionAmount = 50 + random.nextDouble() * 100;
// Random price between 50 and 150

            boolean isBuying = random.nextBoolean();


            if (isBuying) {
```

```
                stockMarket.buyStock(stockSymbol, transactionAmount,
name, logFileName);
            } else {
                stockMarket.sellStock(stockSymbol, transactionAmount,
name, logFileName);
            }

            try {
                Thread.sleep(500); // Simulate delay
            } catch (InterruptedException e) {
                System.err.println("Trader interrupted: " +
e.getMessage());
            }
        }
    }
}
```

---

3. StockMarket Class

java

Copy code

```
import java.util.HashMap;
import java.util.Set;
```

```
public class StockMarket {

    private HashMap<String, Stock> stocks = new HashMap<>();


    public void addStock(Stock stock) {

        stocks.put(stock.getStockSymbol(), stock);

    }


    public String[] getStockSymbols() {

        Set<String> keys = stocks.keySet();

        return keys.toArray(new String[0]);

    }


    public synchronized void buyStock(String stockSymbol, double
price, String traderName, String logFileName) {

        Stock stock = stocks.get(stockSymbol);

        if (stock != null) {

            stock.updatePrice(stock.getPrice() + price * 0.01); //
Increase price by 1% of the transaction

            String transactionDetails = "BUY: Trader " + traderName +
" bought " + stockSymbol + " for " + price;

            stock.logTransaction(logFileName, transactionDetails);

            System.out.println(transactionDetails);

        }

    }
```

```
    public synchronized void sellStock(String stockSymbol, double
price, String traderName, String logFileName) {

        Stock stock = stocks.get(stockSymbol);

        if (stock != null) {

            stock.updatePrice(stock.getPrice() - price * 0.01); //
Decrease price by 1% of the transaction

            String transactionDetails = "SELL: Trader " + traderName +
" sold " + stockSymbol + " for " + price;

            stock.logTransaction(logFileName, transactionDetails);

            System.out.println(transactionDetails);

        }

    }

}
```

---

4. Main Method (Simulation)

```
java

Copy code

public class StockTradingApp {

    public static void main(String[] args) {

        String logFileName = "stock_transactions.log";


        // Initialize StockMarket and add stocks
```

```
        StockMarket stockMarket = new StockMarket();

        stockMarket.addStock(new Stock("AAPL", "Apple Inc.", 150.00));

        stockMarket.addStock(new Stock("GOOGL", "Alphabet Inc.",
2800.00));

        stockMarket.addStock(new Stock("AMZN", "Amazon.com, Inc.",
3400.00));

        stockMarket.addStock(new Stock("TSLA", "Tesla, Inc.",
1000.00));

        stockMarket.addStock(new Stock("MSFT", "Microsoft
Corporation", 299.00));



        // Create and start trader threads

        Thread trader1 = new Thread(new Trader("Trader1", stockMarket,
logFileName));

        Thread trader2 = new Thread(new Trader("Trader2", stockMarket,
logFileName));

        Thread trader3 = new Thread(new Trader("Trader3", stockMarket,
logFileName));



        trader1.start();

        trader2.start();

        trader3.start();



        try {

            trader1.join();

            trader2.join();
```

```
            trader3.join();
        } catch (InterruptedException e) {
            System.err.println("Main thread interrupted: " +
e.getMessage());
        }

        System.out.println("Stock trading simulation complete. Check "
+ logFileName + " for transaction logs.");
    }
}
```

Readme File

Stock Trading Simulation System

Description

A Java-based multithreaded system that simulates real-time stock trading, enabling traders to buy and sell stocks with synchronized updates to stock prices and transaction logs.

Features

- **Stock Management**:
  - Update stock prices in real-time.
  - Maintain synchronized access to stock data to avoid race conditions.
- **Trader Functionality**:
  - Simulate multiple traders buying and selling stocks concurrently.
  - Automatically log transactions with timestamps.
- **Transaction Logging**:
  - Log all buy and sell operations to a file.
  - Ensure thread-safe logging using synchronized file access.
- **Stock Market**:

- Manage a collection of stocks.
- Allow multiple traders to trade stocks in parallel.

How to Compile and Run

1. Save all the provided class files (Stock.java, Trader.java, StockMarket.java, and StockTradingApp.java) in the same directory.

Compile:
bash
Copy code
javac *.java

2.

Run:
bash
Copy code
java StockTradingApp

3.

Usage

1. The application creates a collection of stocks and simulates trading by three traders.
2. Transactions are logged in real-time to stock_transactions.log.
3. At the end of the simulation, check the console for trading activity and the log file for detailed transaction records.

Example Input and Output

Console Output:

makefile

Copy code

BUY: Trader Trader1 bought AAPL for 125.67

SELL: Trader Trader2 sold TSLA for 98.76

BUY: Trader Trader3 bought GOOGL for 100.25

Transaction Log (stock_transactions.log):

yaml

Copy code

2024-12-15 12:30:01 - BUY: Trader Trader1 bought AAPL for 125.67

2024-12-15 12:30:02 - SELL: Trader Trader2 sold TSLA for 98.76

2024-12-15 12:30:03 - BUY: Trader Trader3 bought GOOGL for 100.25

Authors

- Code by Abhigyan and Arnav:)

  Problem: **Movie Rental System**
  Movie Class

java

Copy code

```
public class Movie {

    private String movieId;

    private String title;

    private String genre;

    private boolean isAvailable;
```

```
    public Movie(String movieId, String title, String genre, boolean
isAvailable) {
        this.movieId = movieId;
        this.title = title;
        this.genre = genre;
        this.isAvailable = isAvailable;
    }

    public String getMovieId() {
        return movieId;
    }

    public String getTitle() {
        return title;
    }

    public String getGenre() {
        return genre;
    }

    public boolean isAvailable() {
        return isAvailable;
    }
```

```
    public void setAvailable(boolean available) {
        isAvailable = available;
    }

    @Override
    public String toString() {
        return movieId + ", " + title + ", " + genre + ", " +
(isAvailable ? "Available" : "Not Available");
    }
}
```

. Customer Class

java

Copy code

```
import java.util.ArrayList;

public class Customer {
    private String customerId;
    private String name;
    private ArrayList<Movie> rentedMovies;

    public Customer(String customerId, String name) {
        this.customerId = customerId;
```

```
        this.name = name;
        this.rentedMovies = new ArrayList<>();
    }

    public String getCustomerId() {
        return customerId;
    }

    public String getName() {
        return name;
    }

    public ArrayList<Movie> getRentedMovies() {
        return rentedMovies;
    }

    public void rentMovie(Movie movie) {
        rentedMovies.add(movie);
    }

    public void returnMovie(Movie movie) {
        rentedMovies.remove(movie);
    }
```

```
}
```

. MovieRentalStore Class

java

Copy code

```
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;


public class MovieRentalStore<T> {
    private HashMap<String, Movie> movies = new HashMap<>();
    private HashMap<String, T> customers = new HashMap<>();

    public void addMovie(Movie movie) {
        movies.put(movie.getMovieId(), movie);
    }

    public void addCustomer(String id, T customer) {
        customers.put(id, customer);
    }

    public void rentMovie(String movieId, String customerId) throws
Exception {
        Movie movie = movies.get(movieId);
```

```
        Customer customer = (Customer) customers.get(customerId);

        if (movie == null || customer == null) {
            throw new Exception("Movie or Customer not found!");
        }

        if (!movie.isAvailable()) {
            throw new Exception("Movie is not available!");
        }

        movie.setAvailable(false);
        customer.rentMovie(movie);
        System.out.println("Movie " + movie.getTitle() + " rented to
Customer " + customer.getName());
    }

    public void returnMovie(String movieId, String customerId) throws
Exception {
        Movie movie = movies.get(movieId);
        Customer customer = (Customer) customers.get(customerId);

        if (movie == null || customer == null ||
!customer.getRentedMovies().contains(movie)) {
            throw new Exception("Invalid return operation!");
```

```
        }


        movie.setAvailable(true);

        customer.returnMovie(movie);

        System.out.println("Movie " + movie.getTitle() + " returned by
Customer " + customer.getName());

    }


    public void displayAvailableMovies() {

        movies.values().stream()

                .filter(Movie::isAvailable)

                .forEach(System.out::println);

    }


    public void loadMoviesFromFile(String fileName) throws IOException
{

        try (BufferedReader br = new BufferedReader(new
FileReader(fileName))) {

            String line;

            while ((line = br.readLine()) != null) {

                String[] parts = line.split(", ");

                addMovie(new Movie(parts[0], parts[1], parts[2],
Boolean.parseBoolean(parts[3])));

            }
```

```
        }
    }


    public void loadCustomersFromFile(String fileName) throws
IOException {
        try (BufferedReader br = new BufferedReader(new
FileReader(fileName))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] parts = line.split(", ");
                Customer customer = new Customer(parts[0], parts[1]);
                String[] rentedMovies = parts[2].split(";");
                for (String movieId : rentedMovies) {
                    Movie movie = movies.get(movieId);
                    if (movie != null) {
                        customer.rentMovie(movie);
                    }
                }
                addCustomer(parts[0], (T) customer);
            }
        }
    }
```

```
    public void saveRentalData(String moviesFileName, String
customersFileName) throws IOException {

        try (BufferedWriter bw = new BufferedWriter(new
FileWriter(moviesFileName))) {

            for (Movie movie : movies.values()) {

                bw.write(movie.toString());

                bw.newLine();

            }

        }


        try (BufferedWriter bw = new BufferedWriter(new
FileWriter(customersFileName))) {

            for (T customerObj : customers.values()) {

                Customer customer = (Customer) customerObj;

                String rentedMovieIds = String.join(";",

                        customer.getRentedMovies().stream()

                                .map(Movie::getMovieId)

                                .toArray(String[]::new));

                bw.write(customer.getCustomerId() + ", " +
customer.getName() + ", " + rentedMovieIds);

                bw.newLine();

            }

        }

    }
```

```
}
```

. Main Method (MovieRentalApp)

java

Copy code

```
import java.util.Scanner;


public class MovieRentalApp {
    public static void main(String[] args) {
        MovieRentalStore<Customer> store = new MovieRentalStore<>();
        Scanner scanner = new Scanner(System.in);


        try {
            // Load initial data
            store.loadMoviesFromFile("movies.txt");
            store.loadCustomersFromFile("customers.txt");
        } catch (IOException e) {
            System.err.println("Error loading data: " +
e.getMessage());
        }


        while (true) {
            System.out.println("\n--- Movie Rental System ---");
            System.out.println("1. Display Available Movies");
```

```
            System.out.println("2. Add New Movie");
            System.out.println("3. Add New Customer");
            System.out.println("4. Rent a Movie");
            System.out.println("5. Return a Movie");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            try {
                switch (choice) {
                    case 1:
                        store.displayAvailableMovies();
                        break;
                    case 2:
                        System.out.print("Enter Movie ID: ");
                        String movieId = scanner.nextLine();
                        System.out.print("Enter Title: ");
                        String title = scanner.nextLine();
                        System.out.print("Enter Genre: ");
                        String genre = scanner.nextLine();
                        store.addMovie(new Movie(movieId, title,
genre, true));
```

```
                        System.out.println("New movie added
successfully.");
                        break;
                    case 3:
                        System.out.print("Enter Customer ID: ");
                        String customerId = scanner.nextLine();
                        System.out.print("Enter Name: ");
                        String name = scanner.nextLine();
                        store.addCustomer(customerId, new
Customer(customerId, name));
                        System.out.println("New customer added
successfully.");
                        break;
                    case 4:
                        System.out.print("Enter Movie ID to rent: ");
                        String rentMovieId = scanner.nextLine();
                        System.out.print("Enter Customer ID: ");
                        String rentCustomerId = scanner.nextLine();
                        store.rentMovie(rentMovieId, rentCustomerId);
                        break;
                    case 5:
                        System.out.print("Enter Movie ID to return:
");
                        String returnMovieId = scanner.nextLine();
```

```
                        System.out.print("Enter Customer ID: ");
                        String returnCustomerId = scanner.nextLine();
                        store.returnMovie(returnMovieId,
returnCustomerId);
                        break;
                    case 6:
                        store.saveRentalData("updated_movies.txt",
"updated_customers.txt");
                        System.out.println("Data saved successfully.
Exiting...");
                        scanner.close();
                        return;
                    default:
                        System.out.println("Invalid choice. Please try
again.");
                }
            } catch (Exception e) {
                System.err.println(e.getMessage());
            }
        }
    }
}
```

Movie Rental System

Description

A Java-based application to manage movie rentals for a rental store, allowing users to add movies, manage customer accounts, and perform rental operations.

Features

- **Movie Management**:
  - Add new movies to the inventory.
  - Display all available movies for rent.
  - Load and save movie data to/from files.
- **Customer Management**:
  - Add new customers to the system.
  - Track movies rented by customers.
  - Load and save customer data to/from files.
- **Rental Operations**:
  - Rent a movie to a customer.
  - Return a rented movie to the inventory.
  - Handle invalid operations such as renting unavailable movies or returning movies not rented.
- **File I/O**:
  - Load movies and customer data from files (movies.txt, customers.txt).
  - Save updated data to files (updated_movies.txt, updated_customers.txt).

How to Compile and Run

1. Save all the provided class files (Movie.java, Customer.java, MovieRentalStore.java, and MovieRentalApp.java) in the same directory.

Compile:
bash
Copy code
javac *.java

2.

Run:
bash
Copy code
java MovieRentalApp

3.

Usage

- Follow the menu options displayed in the console:
    1. Display all available movies.
    2. Add a new movie.
    3. Add a new customer.
    4. Rent a movie.
    5. Return a movie.
    6. Exit the program.

Example Input and Output

Example Console Interaction:

markdown

Copy code

1. Display Available Movies

2. Add New Movie

3. Add New Customer

4. Rent a Movie

5. Return a Movie

6. Exit

Enter your choice: 4

Enter Movie ID to rent: M001

Enter Customer ID: C002

Movie Inception rented to Customer Jane Smith.

Example Log:

Movies File (movies.txt):
arduino
Copy code
M001, Inception, Sci-Fi, true

M002, Titanic, Romance, true

M003, The Godfather, Crime, false

-

Customers File (customers.txt):
Copy code
C001, John Doe, M003

C002, Jane Smith,

-
- **Updated Files after running the program:**
    - updated_movies.txt
    - updated_customers.txt

Authors

- Code by Abhigyan and Arnav :)