K.R. MANGALAM UNIVERSITY

THE COMPLETE WORLD OF EDUCATION

JAVA PROGRAMMING LAB

(ENCS251)

K.R. MANGALAM UNIVERSITY

For

Bachelor Of Technology

In

Computer Science Engineering

Submitted By

Submitted To

Mrs. Lucky Verma

Assist. Professor

INDEX

Practical 01 : Java Application that manages a small library system.

Problem Statement : In this assignment, you will work with arrays and strings in Java. You will develop a java application that manages a small library system, which includes storing book titles and managing the borrowing system. The system should use arrays to store the book titles and track the status of each book (borrowed or available). Additionally, you will implement string manipulations to handle various book-related operations.

Code : 1-

```
import java.util.Arrays;
class LibrarySystem {
private String[] bookTitles = new String[100];
private boolean[] bookStatus = new boolean [100];
private int bookCount = 0;
// Method to add a book if it's not already in the library
public void addBook(String title) {
if (getBookIndex(title) != -1) {
System.out.println("Book already exists: + title);
return;
}
if (bookCount == bookTitles.length) expandCapacity();
bookTitles [bookCount] = title;
bookStatus [bookCount] = true;
bookCount++;
System.out.println("Book added: " + title);
}
private void expandCapacity() {
bookTitles = Arrays.copyOf (bookTitles, bookTitles.length * 2);
bookStatus = Arrays.copyOf(bookStatus, bookStatus.length * 2);
}
// Method to borrow a book if available
public void borrowBook(String title) {
int index = getBookIndex(title);
if (index == -1) {
System.out.println("Book not found: + title);
} else if (bookStatus [index]) {
bookStatus [index] = false;
System.out.println("You've borrowed: " + title);
} else {
System.out.println("Book already borrowed: + title);
}
}
// Method to return a borrowed book
public void returnBook(String title) {
int index getBookIndex(title);
if (index == -1) {
System.out.println("Book not found: " + title);
} else if (!bookStatus [index]) {
bookStatus [index] = true;
System.out.println("You've returned: + title);
} else {
System.out.println("Book is already available: " + title);
}
// Method to search for a book by title
public String searchBook(String title) {
int index getBookIndex(title);
```

```
return index == -1? "Book not found": bookTitles [index] + (bookStatus [index] ?
}
private int getBookIndex(String title) {
for
(int i = 0; i < bookCount; i++) {
if (bookTitles[i].equalsIgnoreCase (title)) return i;
}
return -1;
}
// Methods to get all, available, and borrowed books
public String[] getAllBooks() {
return Arrays.copyOfRange(bookTitles,, bookCount);
}
public String[] getAvailableBooks() {
return getBooksByStatus (true);
}
public String[] getBorrowedBooks() {
return getBooksByStatus (false);
}
private String[] getBooksByStatus (boolean status) {
return Arrays.stream(bookTitles,, bookCount)
.filter(title -> bookStatus [getBookIndex(title)) == status)
.toArray(String[]::new);
}
}
public class Main {
public static void main(String[] args) {
LibrarySystem library = new LibrarySystem();
library.addBook("Java Programming");
library.borrowBook("Java Programming");
library.returnBook ("Java Programming");
System.out.println(library.searchBook("Java Programming"));
System.out.println(Arrays.toString(library.getAvailableBooks()));
}
 }
```

Output :

```
Book added: Java Programming
You've borrowed: Java Programming
You've returned: Java Programming
Java Programming - Available
[Java Programming]


...Program finished with exit code 0
Press ENTER to exit console.
```

Practical 02 : A Java Application to manage a simple bank account system.

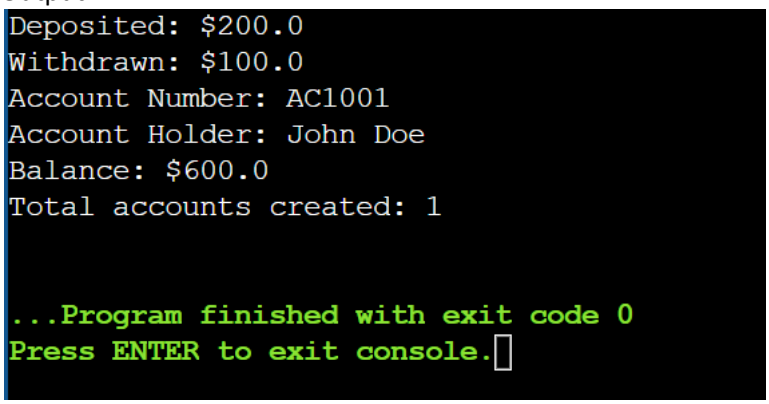
Problem Statement : In this assignment, you will create a Java application to manage a simple bank account system. You will implement classes to represent bank accounts and customers, utilizing concepts such as data members, member methods, constructors, method overloading, static members, and the 'this' keyword. The system should allow users to create accounts, deposit and withdraw funds, and display account information.

Code :

```
class BankAccount {
// Data Members
private String accountNumber;
private String accountHolderName;
private double balance;
// Static data member to track the number of accounts
private static int accountCount = 0;
// Default Constructor
public BankAccount() {
this.accountNumber = "AC" + (++accountCount); // Automatically generated account number
this.accountHolderName = "Unknown";
this.balance = 0.0;
}
// Parameterized Constructor
public BankAccount(String accountNumber, String accountHolderName, double initialBalance) {
this.accountNumber = accountNumber;
this.accountHolderName = accountHolderName:
this.accountHolderName = accountHolderName;
this.balance initialBalance;
accountCount++;
}
// Copy Constructor
public BankAccount (BankAccount other) {
this.accountNumber = other.accountNumber;
this.accountHolderName other.accountHolderName;
this.balance other.balance;
accountCount++;
}
// Method to deposit money into the account
public void deposit (double amount) {
if (amount > 0) {
this.balance += amount;
System.out.println("Deposited: $" + amount);
} else {
System.out.println("Deposit amount must be positive.");
}
}
// Method to withdraw money from the account with balance check
public void withdraw(double amount) {
if (amount > 0 && amount <= this.balance) {
this.balance amount;
System.out.println("Withdrawn: $" + amount);
} else if (amount > this.balance) {
System.out.println("Insufficient funds for withdrawal.");
} else {
System.out.println("Withdrawal amount must be positive.");
}
}
```

```
// Method to display account information
public void displayAccountInfo() {
System.out.println("Account Number: " + this.accountNumber);
System.out.println("Account Holder: " + this.account HolderName);
System.out.println("Balance: $" + this.balance);
}
// Static method to get the total number of accounts created
public static int getAccountCount() {
return accountCount;
}
}
public class Main {
public static void main(String[] args) {
// Deposit and withdraw some money
account.deposit(200);
// Deposit $200
account.withdraw(100);
// Withdraw $100
// Display account information
account.displayAccountInfo();
// Display total number of accounts created
System.out.println("Total accounts created: " + BankAccount.getAccountCount());
}
}
```

Output :



Practical 03: Java class hierarchy for a simple educational institution system.

Problem Statement : In this assignment, you will explore inheritance and access modifiers in Java by creating a class hierarchy for a simple educational institution system. The system will include classes for persons, students, and faculty members. You will implement various types of inheritance and utilize access modifiers to control access to class members. The goal is to demonstrate your understanding of derived and superclass relationships, the 'super' keyword, and different inheritance types.

Code :

```
class Person {
protected String name;
protected int age;
private String address;
public Person(String name, int age, String address) {
this.name = name;
this.age age;
this.address = address;
}
// Display method
public void displayInfo() {
System.out.println("Name: System.out.println("Age: + age);
System.out.println("Address: " + address);
}
}
class Student extends Person {
private String studentId;
private String course;
// Constructor
public Student(String name, int age, String address, String studentId, String course) {
super (name, age, address); // Call to superclass constructor
this.studentId = studentId;
this.course = course;
}
// Display student-specific information
public void displayStudentInfo() {
super.displayInfo(); // Call to superclass method System.out.println("Student ID: + studentId);
System.out.println("Course: " + course);
}
}
class Faculty extends Person {
private String facultyId;
private String department;
// Constructor
public Faculty (String name, int age, String address, String facultyId, String department) {
super(name, age, address); // Call to superclass constructor
this.facultyId = facultyId;
this.department = department;
}
// Display faculty-specific information
public void displayFacultyInfo() {
public void displayFacultyInfo() {
super.displayInfo(); // Call to superclass method
System.out.println("Faculty ID: " + facultyId);
System.out.println("Department: + department);
}
}
class TeachingAssistant extends Student {
```

```
private String facultyAdvisor;

public TeachingAssistant(String name, int age, String address, String studentId, String course
super(name, age, address, studentId, course); // Call to superclass (Student) constructo
this.facultyAdvisor facultyAdvisor;
}
// Display TA-specific information
public void displayTAInfo() {
super.displayStudentInfo(); // Call to superclass (Student) method
System.out.println("Faculty Advisor: + facultyAdvisor);
}
}
class Main {
public static void main(String[] args) {
Person person = new Person("Alice", 45, "123 Maple St.");
Student student = new Student("Bob", 20, "456 Oak St.", "S123", "Computer Science");
Faculty faculty = new Faculty ("Dr. Smith", 50, "789 Pine St.", "F456", "Engineering");
TeachingAssistant ta = new TeachingAssistant("Charlie", 22, "456 Oak St.", "S124", "Comput
System.out.println("Person Info:"); person.displayInfo();
System.out.println("\nStudent Info: ");
student.displayStudentInfo();
System.out.println("\nFaculty Info:");
faculty.displayFacultyInfo();
System.out.println("\nTeaching Assistant Info:");
ta.displayTAInfo();
}
}
```

Output:

```
Person Info:
Name: Alice
Age: 45
Address: 123 Maple St.

Student Info:
Name: Bob
Age: 20
Address: 456 Oak St.
Student ID: S123
Course: Computer Science

Faculty Info:
Name: Dr. Smith
Age: 50
Address: 789 Pine St.
Faculty ID: F456
Department: Engineering

Teaching Assistant Info:
Name: Charlie
Age: 22
Address: 456 Oak St.
Student ID: S124
```

Practical 04: A system for managing different types of vehicles in a rental service.

Problem Statement : In this assignment, you will explore the concepts of polymorphism in Java, both static (method overloading) and dynamic (method overriding). Additionally, you will implement final classes and methods, and learn about resource management using the 'finalize' keyword and garbage collection. You will create a system for managing different types of vehicles in a rental service, demonstrating these concepts.

Code :

```
abstract class Vehicle {
private String licensePlate;
private String model;
// Constructor
public Vehicle(String licensePlate, String model) {
this.licensePlate licensePlate;
this.model = model;
}
// Method to display vehicle information (dynamic polymorphism)
public void displayInfo() {
System.out.println("License Plate: " + licensePlate);
System.out.println("Model: " + model);
}
// Overloaded methods to demonstrate static polymorphism
public void rent() {
System.out.println("Vehicle rented.");
}
public void rent(int days) {
System.out.println("Vehicle rented for + days + days.");
}
// Finalize method to demonstrate garbage collection
@Override
protected void finalize() throws Throwable {
System.out.println("Vehicle object with license plate + licensePlate is being garba
}
}
class Car extends Vehicle {
private int seatingCapacity;
public Car(String licensePlate, String model, int seatingCapacity) {
super(licensePlate, model);
this.seatingCapacity seatingCapacity;
}
}
}
final class Motorcycle extends Vehicle {
private boolean hasSidecar;
public Motorcycle(String licensePlate, String model, boolean hasSidecar) {
super(licensePlate, model);
this.hasSidecar hasSidecar;
}
@Override
public void displayInfo() {
super.displayInfo();
System.out.println("Sidecar: "+ (hasSidecar? "Yes": "No"));
}
}
class ElectricCar extends Car {
private final int batteryCapacity; // Final data member
```

```
public ElectricCar (String licensePlate, String model, int seatingCapacity, int batteryCapacity
super (licensePlate, model, seatingCapacity);
this.batteryCapacity = batteryCapacity;
}
@Override
public void displayInfo() {
super.displayInfo();
System.out.println("Battery Capacity:" + batteryCapacity + kWh");
}
// Method demonstrating the use of 'final' keyword
public final void chargeBattery() {
System.out.println("ElectricCar is being charged.");
}
public class Main {
public static void main(String[] args) {
Vehicle car = new Car ("ABC123", "Toyota Camry", 5);
ElectricCar("EV456", "Tesla Model 3", 5, 75);
System.out.println("Car Info:");
car.displayInfo();
car.rent();
car.rent(3);
System.out.println("\nMotorcycle Info:");
motorcycle.displayInfo();
System.out.println("\nElectricCar Info:");
electricCar.displayInfo();
electriccar = null; // Eligible for garbage collection
System.gc(); // Request garbage collection to demonstrate finalize
}
}
```

Output ;

```
Car Info:
License Plate: ABC123
Model: Toyota Camry
Seating Capacity: 5
Vehicle rented.
Vehicle rented for 3 days.

Motorcycle Info:
License Plate: XYZ789
Model: Harley Davidson
Sidecar: Yes

ElectricCar Info:
License Plate: EV456
Model: Tesla Model 3
Seating Capacity: 5
Battery Capacity: 75 kWh


...Program finished with exit code 0
Press ENTER to exit console.
```

Practical 05: A Java Application for a basic shape drawing application.

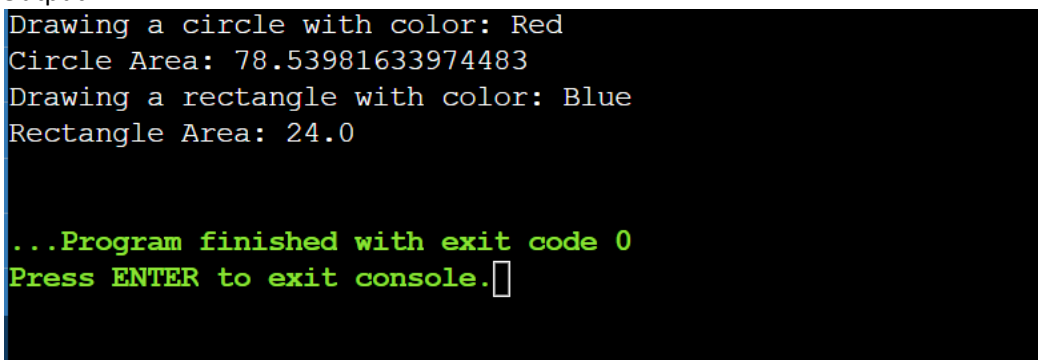
Problem Statement : In this assignment, you will explore the concepts of abstract classes and interfaces in Java, along with creating and managing packages. You will develop a software system for a basic shape drawing application. This system will utilize abstract classes for general shape properties, interfaces for drawable behavior, and packages for organizing the codebase. The goal is to demonstrate your understanding of abstraction, interfaces, and code organization using packages.

Code :

```
-public class Main {
// Abstract Shape class
public abstract static class Shape { private String color;
public Shape(String color) { this.color = color;
}
public String getColor() {
return color;
}
public void setColor(String color) { this.color = color;
}
public abstract void draw(); // Abstract method for drawing
public abstract double calculateArea(); // Abstract method for calculating area
}
// Drawable interface
public interface Drawable {
void draw();
}
// Circle class extending Shape and implementing Drawable
public static class Circle extends Shape implements Drawable {
private double radius;
public Circle(String color, double radius) {
super(color);
this.radius = radius;
}
@Override
public void draw() {
System.out.println("Drawing a circle with color: + getColor());
}
@Override
public double calculateArea() {
return Math.PI radius radius;
}
}
// Rectangle class extending Shape and implementing Drawable
public static class Rectangle extends Shape implements Drawable {
private double length;
private double width;
public Rectangle(String color, double length, double width) {
super(color);
this.length = length;
this.width = width;
}
@Override
public void draw() {
System.out.println("Drawing a rectangle with color: + getColor());
}
@Override
public double calculateArea() {
```

```
return length width;
}
}
// Main method to test the classes
public static void main(String[] args) {
Shape circle = new Circle("Red", 5.0);
Shape rectangle = new Rectangle("Blue", 4.0, 6.0);
circle.draw();
System.out.println("Circle Area: " + circle.calculateArea());
rectangle.draw();
System.out.println("Rectangle Area: + rectangle.calculateArea());
}
}
```

Output :



Practical 06 : A Java multithreaded application that simulates a banking system.

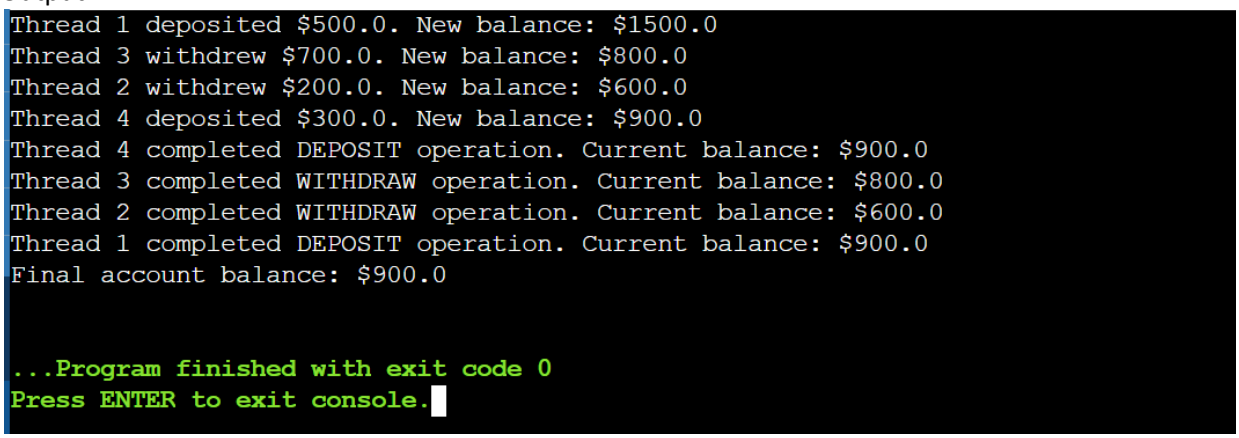
Problem Statement : In this assignment, you will delve into exception handling, multithreaded programming, and wrapper classes in Java. You will create a multithreaded application that simulates a banking system, demonstrating the use of exception handling for error management, thread synchronization, and the use of wrapper classes for data manipulation. The system should handle various operations such as account creation, deposit, withdrawal, and balance checking.

Code :

```
// BankAccount.java
class BankAccount {
private String accountNumber;
private Double balance; // Using Double wrapper class to demonstrate autoboxing/unboxing
public BankAccount(String accountNumber, double balance) {
this.accountNumber accountNumber;
this.balance = balance;
}
public synchronized void deposit(double amount) {
balance + amount;
System.out.println(Thread.currentThread().getName() + " deposited $" + amount + New balance: $"
}
public synchronized void withdraw (double amount) throws Insufficient Funds Exception {
if (balance amount) {
throw new Insufficient Funds Exception("Insufficient funds for withdrawal of $" + amount);
}
balance amount;
System.out.println(Thread.currentThread().getName() + withdrew $" + amount+". New balance: $"
}
public double getBalance() {
return balance;
}
}
// Insufficient Funds Exception.java
class Insufficient Funds Exception extends Exception {
public Insufficient Funds Exception(String message) {
super (message);
}
}
// OperationType.java
enum OperationType {
DEPOSIT,
WITHDRAW
}
// BankOperation.java
class BankOperation implements Runnable {
private BankAccount account;
private double amount;
private OperationType type;
public BankOperation (BankAccount account, double amount, OperationType type) {
this.account = account;
this.amount = amount;
this.type type;
}
@Override
public void run() {
try {
switch (type) {
```

```
case DEPOSIT:
account.deposit(amount);
break;
case WITHDRAW:
account.withdraw(amount);
break;
}
} catch (Insufficient FundsException e) {
System.out.println(Thread.currentThread().getName() + failed to withdraw $" + amount +
} finally
{
System.out.println(Thread.currentThread().getName()+ completed + type + operation. Curren
}
}
}
// Main.java
public class Main {
public static void main(String[] args) {
BankAccount account = new BankAccount("12345678", 1000.0);
Thread t1 new Thread(new BankOperation (account, 500.0, OperationType.DEPOSIT), "Thread 1");
Thread t2 = new Thread(new BankOperation (account, 200.0, OperationType.WITHDRAW), "Thread 2");
Thread t3= new Thread(new BankOperation (account, 700.0, OperationType. WITHDRAW), "Thread 3");
Thread t4 new Thread(new BankOperation(account, 300.0, OperationType. DEPOSIT), "Thread 4");
// Start threads
t1.start();
t2.start();
t3.start();
try {
t1.join();
t3.join();
t4.join();
}
catch (InterruptedException e)
{
System.out.println("Thread interrupted: + e.getMessage());
}
System.out.println("Final account balance: $" + account.getBalance());
}
}
```

Output :



Practical 07 : A file management system that supports operations such as reading, writing, copying, and navigating files and directories.

Problem Statement : In this assignment, you will explore file handling in Java, focusing on reading from and writing to files using various I/O streams. You will develop a file management system that supports operations such as reading, writing, copying, and navigating files and directories. This system will utilize different types of streams and classes for efficient file handling, including buffered streams, character and byte streams, and random access files.

Code :

```
import java.io.*;
import java.nio.file.*;
import java.util.Arrays;
public class FileManager {
private File file;
public FileManager (String filePath) {
this.file= new File(filePath);
}
// Method to read file using BufferedReader
public void readFile() {
try (BufferedReader br = new BufferedReader(new FileReader(file))) {
String line;
while ((line = br.readLine()) != null) {
System.out.println(line);
}
} catch (IOException e) {
System.err.println("Error reading file: + e.getMessage());
}
}
// Method to write content to file using BufferedWriter
public void writeFile(String content) {
try (BufferedWriter bw = new BufferedWriter(new FileWriter (file, true))) {
bw.write(content);
bw.newLine();
} catch (IOException e) {
System.err.println("Error writing file: + e.getMessage());
}
}
// Method to copy file content to another file
public void copyFile(String destinationPath) {
try (FileInputStream fis = new FileInputStream(file);
FileOutputStream fos = new FileOutputStream(destinationPath)) {
byte[] buffer = new byte[1024];
int length;
while ((length = fis.read(buffer)) > 0) {
fos.write(buffer,, length);
}
System.out.println("File copied to " + destinationPath);
} catch (IOException e) {
System.err.println("Error copying file: + e.getMessage());
}
}
// Random access write operation
public void randomAccessWrite(String content, long position) {
try (RandomAccessFile raf = new RandomAccessFile(file, "rw")) {
raf.seek(position);
raf.writeUTF(content);
```

```
} catch (IOException e) {
System.err.println("Error in random access write: " + e.getMessage());
}
}
// Random access read operation
public void randomAccessRead(long position, int size) {
try (RandomAccessFile raf new RandomAccess File(file, "r")) {
raf.seek(position);
byte[] bytes = new byte[size];
raf.read(bytes);
System.out.println("Data at position " + position + ": + new String(bytes));
} catch (IOException e) {
System.err.println("Error in random access read: + e.getMessage());
}
}
// List directory contents
public void listDirectoryContents() {
if (file.isDirectory()) {
System.out.println("Directory contents:");
Arrays.stream(file.listFiles()).forEach(f -> System.out.println(f.getName()));
} else {
System.out.println("Not a directory.");
raf.seek(position);
byte[] bytes = new byte[size];
raf.read(bytes);
System.out.println("Data at position " + position + ": + new String(bytes));
} catch (IOException e) {
System.err.println("Error in random access read: + e.getMessage());
}
}
// List directory contents
public void listDirectoryContents() {
if (file.isDirectory()) {
System.out.println("Directory contents:");
Arrays.stream(file.listFiles()).forEach(f -> System.out.println(f.getName()));
} else {
System.out.println("Not a directory.");
}
}
// Print file attributes
public void printFileAttributes() {
try {
System.out.println("File size: + Files.size(file.toPath()));
System.out.println("Readable: + Files.isReadable(file.toPath()));
System.out.println("Writable: + Files.isWritable(file.toPath()));
System.out.println("Executable: + Files.isExecutable (file.toPath()));
} catch (IOException e) {
System.err.println("Error getting attributes: + e.getMessage());
}
}
// Change file permissions
public void changeFilePermissions(boolean readable, boolean writable, boolean executable) {
file.setReadable (readable);
file.setWritable (writable);
file.setExecutable (executable);
System.out.println("Permissions changed: Readable=" + readable + ", Writable=" + writab:
}
```

```
public static void main(String[] args) {
FileManager manager = new FileManager("example.txt");
// Writing to file
manager.writeFile("Hello, World!");
// Reading from file manager.readFile();
// Copying file manager.copyFile("example_copy.txt");
// Listing directory contents manager.listDirectoryContents();
// Displaying file attributes manager.printFileAttributes();
// Changing file permissions manager.changeFilePermissions (true, true, false);
```

```
File copied to example_copy.txt
Data at position 5: RandomData
Directory contents:
example.txt
example_copy.txt
File size: 123 bytes
Readable: true
Writable: true
Executable: false
Permissions changed: Readable=true, Writable=true, Executable=false
```

Practical 08 : A contact management system that utilizes different data structures like Lists, Sets, Queues, and Maps.

Problem Statement : In this assignment, you will explore the Java Collections Framework, focusing on the usage of various collection interfaces and classes. You will develop a contact management system that utilizes different data structures like Lists, Sets, Queues, and Maps. The system should support operations such as adding, removing, and searching contacts, as well as sorting and iterating over the collections.

Code :

```
import java.util.*;
public class ContactManager {
private List<Contact> contactList = new ArrayList<>();
private Set<Contact> contactSet = new HashSet<>();
private Queue<Contact> contactQueue = new PriorityQueue<>(Comparator.comparing (Contact
private Map<String, Contact> contactMap = new HashMap<>();
// Add a contact to the system
public void addContact (Contact contact) {
contactList.add(contact);
contactSet.add(contact);
contactQueue.offer(contact);
contactMap.put(contact.getName(), contact);
}
// Remove a contact by name
public void removeContact(String name) {
contactList.removeIf (c -> c.getName().equals(name));
contactSet.removeIf (c -> c.getName().equals(name));
contactQueue.removeIf(c -> c.getName().equals(name));
contactMap.remove(name);
}
// Search for a contact by name
public Contact searchContactByName (String name) {
return contactMap.get(name);
}
// Sort contacts by name
public void sortContactsByName() {
contactList.sort(Comparator.comparing (Contact::getName));
}
// Display all contacts
public void displayAllContacts() {
contactList.forEach(System.out::println);
}
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
ContactManager manager = new ContactManager();
// Adding sample contacts
manager.addContact(new Contact("Alice", "1234567890", "alice@example.com"));
manager.addContact(new Contact("Bob", "2345678901", "bob@example.com"));
// Simple console interaction for managing contacts
while (true) {
System.out.println("\n1. Add 2. Remove 3. Search 4. Display 5. Sort 6. Exit");
int option = scanner.nextInt();
scanner.nextLine(); // Consume newline
if (option == 1) {
System.out.print("Enter name: ");
String name = scanner.nextLine();
System.out.print("Enter phone: ");
String phone = scanner.nextLine();
```

```
System.out.print("Enter email: ");
String email = scanner.nextLine();
manager.addContact(new Contact (name, phone, email));
} else if (option == 2) {
System.out.print("Enter name to remove: ");
manager.removeContact(scanner.nextLine());
} else if (option == 3) {
System.out.print("Enter name to search: ");
Contact contact = manager.searchContactByName(scanner.nextLine());
if (contact != null) {
System.out.println(contact);
} else {
System.out.println("Contact not found.");
}
} else if (option == 4) {
manager.displayAllContacts();
} else if (option == 5) {
manager.sortContactsByName();
System.out.println("Contacts sorted.");
} else if (option == 6) break;
}
}
}
@Override
public String toString() {
return "Name: "+ name + Phone: + phone + Email: + email;
}
@Override
public boolean equals (Object obj) {
if (this == obj) return true;
if (obj == null || getClass() != obj.getClass()) return false;
Contact contact = (Contact) obj;
return Objects.equals(name, contact.name);
}
@Override
public int hashCode() {
return Objects.hash(name);
}
}
```

Output :

```
1. Add 2. Remove 3. Search 4. Display 5. Sort 6. Exit
1. Enter name: Alice
Enter phone: 1234567890
Enter email: alice@example.com

1. Add 2. Remove 3. Search 4. Display 5. Sort 6. Exit
4
Name: Alice, Phone: 1234567890, Email: alice@example.com
Name: Bob, Phone: 2345678901, Email: bob@example.com
```