Q1. Sort a list of students by roll number (ascending) using Comparable.

Create a Student class with fields: rollNo, name, and marks. Implement the Comparable interface to sort students by their roll numbers.

A.

```
package day9;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

class Student implements Comparable<Student> {
    int rollNo;
    String name;
    int marks;
    public Student(int rollNo, String name, int marks) {
        this.rollNo = rollNo;
        this.name = name;
        this.marks = marks;
    }
    public int compareTo(Student other) {
        return Integer.compare(this.rollNo, other.rollNo);
    }
    public String toString() {
        return rollNo + " " + name + " " + marks;
    }
}
public class SortStudentsByRollNumber {
    public static void main(String[] args) {
```

```
        List<Student> list = new ArrayList<Student>();
        list.add(new Student(3, "John", 85));
        list.add(new Student(1, "Alice", 90));
        list.add(new Student(2, "Bob", 75));
        System.out.println("Before Sorting:");
        for (Student s : list) {
            System.out.println(s);
        }
        Collections.sort(list);
        System.out.println("\nAfter Sorting by Roll Number:");
        for (Student s : list) {
            System.out.println(s);
        }
    }
}
```

Q2. Create a Product class and sort products by price using Comparable.

Implement Comparable<Product> and sort a list of products using Collections.sort().

A.

```
package day9;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
class Product implements Comparable<Product> {
    String name;
    double price;
```

```
    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }
    public int compareTo(Product other) {
        return Double.compare(this.price, other.price);
    }
    public String toString() {
        return name + " - $" + price;
    }
}
public class SortProductsByPrice {
    public static void main(String[] args) {
        List<Product> list = new ArrayList<Product>();
        list.add(new Product("Laptop", 1200.0));
        list.add(new Product("Mouse", 25.5));
        list.add(new Product("Keyboard", 75.0));
        System.out.println("Before Sorting:");
        for (Product p : list) {
            System.out.println(p);
        }
        Collections.sort(list);
        System.out.println("After Sorting by Price:");
        for (Product p : list) {
            System.out.println(p);
        }
```

```
    }
}
```

Q3. Create an Employee class and sort by name using Comparable.

Use the compareTo() method to sort alphabetically by employee names.

A.

```
package day9;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
class Employee implements Comparable<Employee> {
    String name;
    double salary;
    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }
    public int compareTo(Employee other) {
        return this.name.compareTo(other.name);
    }
    public String toString() {
        return name + " - " + salary;
    }
}
public class SortEmployeesByName {
    public static void main(String[] args) {
        List<Employee> list = new ArrayList<Employee>();
        list.add(new Employee("John", 50000));
```

```
        list.add(new Employee("Alice", 60000));
        list.add(new Employee("Bob", 45000));
        System.out.println("Before Sorting:");
        for (Employee e : list) {
            System.out.println(e);
        }
        System.out.println("After Sorting by Name:");
        Collections.sort(list);
        for (Employee e : list) {
            System.out.println(e);
        }
    }
}
```

Q4. Sort a list of Book objects by bookId in descending order using Comparable.

Hint: Override compareTo() to return the reverse order.

A.

```
package day9;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
class Book implements Comparable<Book> {
    int bookId;
    String title;
    public Book(int bookId, String title) {
        this.bookId = bookId;
        this.title = title;
    }
```

```
    public int compareTo(Book other) {
        return Integer.compare(other.bookId, this.bookId);
    }
    public String toString() {
        return bookId + " - " + title;
    }
}
public class SortBooksByIdDescending {
    public static void main(String[] args) {
        List<Book> list = new ArrayList<Book>();
        list.add(new Book(103, "Java Basics"));
        list.add(new Book(101, "Algorithms"));
        list.add(new Book(102, "Data Structures"));
        System.out.println("Before Sorting:");
        for (Book b : list) {
            System.out.println(b);
        }
        System.out.println("After Sorting by Book ID (Descending):");
        Collections.sort(list);
        for (Book b : list) {
            System.out.println(b);
        }
    }
}
```

Q5. Implement a program that sorts a list of custom objects using Comparable, and displays them before and after sorting.

A.

```
package day9;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
class Item implements Comparable<Item> {
    int id;
    String name;
    public Item(int id, String name) {
        this.id = id;
        this.name = name;
    }
    public int compareTo(Item other) {
        return Integer.compare(this.id, other.id);
    }
    public String toString() {
        return id + " - " + name;
    }
}
public class SortCustomObjectsUsingComparable {
    public static void main(String[] args) {
        List<Item> list = new ArrayList<Item>();
        list.add(new Item(3, "Pen"));
        list.add(new Item(1, "Notebook"));
        list.add(new Item(2, "Eraser"));
```

```
        System.out.println("Before Sorting:");
        for (Item i : list) {
            System.out.println(i);
        }
        Collections.sort(list);
        System.out.println("After Sorting:");
        for (Item i : list) {
            System.out.println(i);
        }
    }
}
```

Q6. Sort a list of students by marks (descending) using Comparator.

Create a Comparator class or use a lambda expression to sort by marks.

A.
```
package day9;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
class StudentMarks {
    String name;
    int marks;
    public StudentMarks(String name, int marks) {
        this.name = name;
        this.marks = marks;
    }
```

```
    public String toString() {
        return name + " - " + marks;
    }
}
public class SortStudentsByMarksDescending {
    public static void main(String[] args) {
        List<StudentMarks> list = new ArrayList<StudentMarks>();
        list.add(new StudentMarks("John", 85));
        list.add(new StudentMarks("Alice", 92));
        list.add(new StudentMarks("Bob", 78));
        System.out.println("Before Sorting:");
        for (StudentMarks s : list) {
            System.out.println(s);
        }
        Collections.sort(list, (s1, s2) -> Integer.compare(s2.marks,
s1.marks));
        System.out.println("After Sorting by Marks (Descending):");
        for (StudentMarks s : list) {
            System.out.println(s);
        }
    }
}
```

Q7. Create multiple sorting strategies for a Product class.

Implement comparators to sort by:

Price ascending

Price descending

Name alphabetically

A. package day9;

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
class ProductSort {
    String name;
    double price;
    public ProductSort(String name, double price) {
        this.name = name;
        this.price = price;
    }
    public String toString() {
        return name + " - $" + price;
    }
}
public class MultipleSortingStrategiesForProduct {
    public static void main(String[] args) {
        List<ProductSort> list = new ArrayList<ProductSort>();
        list.add(new ProductSort("Laptop", 1200.0));
        list.add(new ProductSort("Mouse", 25.5));
        list.add(new ProductSort("Keyboard", 75.0));
        System.out.println("Sort by Price Ascending:");
        Collections.sort(list, Comparator.comparingDouble(p ->
p.price));
        for (ProductSort p : list) {
            System.out.println(p);
```

```
        }

        System.out.println("Sort by Price Descending:");
        Collections.sort(list, Comparator.comparingDouble((ProductSort
p) -> p.price).reversed());
        for (ProductSort p : list) {
            System.out.println(p);
        }
        System.out.println("Sort by Name Alphabetically:");
        Collections.sort(list, Comparator.comparing(p -> p.name));
        for (ProductSort p : list) {
            System.out.println(p);
        }
    }
}
```

Q8. Sort Employee objects by joining date using Comparator.

Use Comparator to sort employees based on LocalDate or Date.

A.

```
package day9;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
class EmployeeDate {
    String name;
    LocalDate joiningDate;
```

```
    public EmployeeDate(String name, LocalDate joiningDate) {
        this.name = name;
        this.joiningDate = joiningDate;
    }
    public String toString() {
        return name + " - " + joiningDate;
    }
}
public class SortEmployeesByJoiningDate {
    public static void main(String[] args) {
        List<EmployeeDate> list = new ArrayList<EmployeeDate>();
        list.add(new EmployeeDate("John", LocalDate.of(2022, 5, 10)));
        list.add(new EmployeeDate("Alice", LocalDate.of(2020, 3,
15)));
        list.add(new EmployeeDate("Bob", LocalDate.of(2021, 8, 1)));
        System.out.println("Before Sorting:");
        for (EmployeeDate e : list) {
            System.out.println(e);
        }
        Collections.sort(list, Comparator.comparing(e -> e.joiningDate));
        System.out.println("After Sorting by Joining Date:");
        for (EmployeeDate e : list) {
            System.out.println(e);
        }
    }
}
```

Q9. Write a program that sorts a list of cities by population using Comparator.

A.

```
package day9;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
class City {
    String name;
    int population;
    public City(String name, int population) {
        this.name = name;
        this.population = population;
    }
    public String toString() {
        return name + " - " + population;
    }
}
public class SortCitiesByPopulation {
    public static void main(String[] args) {
        List<City> list = new ArrayList<City>();
        list.add(new City("New York", 8419000));
        list.add(new City("Los Angeles", 3980000));
        list.add(new City("Chicago", 2716000));
        System.out.println("Before Sorting:");
        for (City c : list) {
            System.out.println(c);
```

```
        }
        Collections.sort(list, Comparator.comparingInt(c ->
c.population));
        System.out.println("After Sorting by Population:");
        for (City c : list) {
            System.out.println(c);
        }
    }
}
```

Q10. Use an anonymous inner class to sort a list of strings by length.

A.

```
package day9;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
public class SortStringsByLengthUsingAnonymousClass {
    public static void main(String[] args) {
        List<String> list = new ArrayList<String>();
        list.add("Apple");
        list.add("Banana");
        list.add("Kiwi");
        list.add("Strawberry");
        System.out.println("Before Sorting:");
        for (String s : list) {
            System.out.println(s);
        }
```

```
        Collections.sort(list, new Comparator<String>() {
            public int compare(String s1, String s2) {
                return Integer.compare(s1.length(), s2.length());
            }
        });
        System.out.println("After Sorting by Length:");
        for (String s : list) {
            System.out.println(s);
        }
    }
}
```

Q11. Create a program where:

Student implements Comparable to sort by name

Use Comparator to sort by marks

Demonstrate both sorting techniques in the same program.

A.

```
package day9;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
class StudentData implements Comparable<StudentData> {
    String name;
    int marks;
    public StudentData(String name, int marks) {
        this.name = name;
        this.marks = marks;
```

```
    }

    public int compareTo(StudentData other) {
        return this.name.compareTo(other.name);
    }
    public String toString() {
        return name + " - " + marks;
    }
}
public class SortStudentByNameAndMarks {
    public static void main(String[] args) {
        List<StudentData> list = new ArrayList<StudentData>();
        list.add(new StudentData("Neeva Sharma", 85));
        list.add(new StudentData("Reeva Sharma", 90));
        list.add(new StudentData("Seeva Sharma", 75));

        System.out.println("Sorted by Name (Comparable):");
        Collections.sort(list);
        for (StudentData s : list) {
            System.out.println(s);
        }
        System.out.println("Sorted by Marks (Comparator):");
        Collections.sort(list, Comparator.comparingInt(s -> s.marks));
        for (StudentData s : list) {
            System.out.println(s);
        }
```

```
    }
}
```

Q12. Sort a list of Book objects using both Comparable (by ID) and Comparator (by title, then author).

A.

```
package day9;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
class BookInfo implements Comparable<BookInfo> {
    int id;
    String title;
    String author;
    public BookInfo(int id, String title, String author) {
        this.id = id;
        this.title = title;
        this.author = author;
    }
    public int compareTo(BookInfo other) {
        return Integer.compare(this.id, other.id);
    }
    public String toString() {
        return id + " - " + title + " - " + author;
    }
}
public class SortBooksByIdTitleAuthor {
    public static void main(String[] args) {
```

```
        List<BookInfo> list = new ArrayList<BookInfo>();
        list.add(new BookInfo(103, "Java Programming", "Neeva Sharma"));
        list.add(new BookInfo(101, "Algorithms", "Reeva Sharma"));
        list.add(new BookInfo(102, "Data Structures", "Seeva Sharma"));
        System.out.println("Sorted by ID (Comparable):");
        Collections.sort(list);
        for (BookInfo b : list) {
            System.out.println(b);
        }

        System.out.println("Sorted by Title, then Author (Comparator):");
        Collections.sort(list, Comparator.comparing((BookInfo b) -> b.title).thenComparing(b -> b.author));
        for (BookInfo b : list) {
            System.out.println(b);
        }
    }
}
```

Q13. Write a menu-driven program to sort Employee objects by name, salary, or department using Comparator.

A.

```
package day9;
import java.util.*;
class EmployeeSort {
    String name;
    double salary;
```

```
    String department;
    public EmployeeSort(String name, double salary, String
department) {
        this.name = name;
        this.salary = salary;
        this.department = department;
    }
    public String toString() {
        return name + " - " + salary + " - " + department;
    }
}
public class MenuDrivenSortEmployees {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<EmployeeSort> list = new ArrayList<EmployeeSort>();
        list.add(new EmployeeSort("Neeva Sharma", 60000, "HR"));
        list.add(new EmployeeSort("Reeva Sharma", 50000, "IT"));
        list.add(new EmployeeSort("Seeva Sharma", 70000, "Finance"));
        System.out.println("Choose sorting option:");
        System.out.println("1. By Name");
        System.out.println("2. By Salary");
        System.out.println("3. By Department");
        int choice = sc.nextInt();
        switch (choice) {
            case 1:
                Collections.sort(list, Comparator.comparing(e -> e.name));
                break;
```

```
            case 2:
                Collections.sort(list, Comparator.comparingDouble(e ->
e.salary));
                break;
            case 3:
                Collections.sort(list, Comparator.comparing(e ->
e.department));
                break;
            default:
                System.out.println("Invalid choice");
                sc.close();
                return;
        }
        for (EmployeeSort e : list) {
            System.out.println(e);
        }
        sc.close();
    }
}
```

Q14. Use Comparator.comparing() with method references to sort objects in Java 8+.

A.

```
package day9;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
```

```
class Employee {
    String name;
    double salary;
    String department;
    public Employee(String name, double salary, String department) {
        this.name = name;
        this.salary = salary;
        this.department = department;
    }
    public String getName() {
        return name;
    }
    public double getSalary() {
        return salary;
    }
    public String getDepartment() {
        return department;
    }
    public String toString() {
        return name + " - " + salary + " - " + department;
    }
}
public class SortUsingComparatorMethodReference {
    public static void main(String[] args) {
        List<Employee> list = new ArrayList<Employee>();
        list.add(new Employee("Neeva Sharma", 60000, "HR"));
```

```
        list.add(new Employee("Reeva Sharma", 55000, "IT"));
        list.add(new Employee("Seeva Sharma", 70000, "Finance"));
        System.out.println("Sorted by Name:");
        Collections.sort(list,
Comparator.comparing(Employee::getName));
        for (Employee e : list) {
            System.out.println(e);
        }
        System.out.println("Sorted by Salary:");
        Collections.sort(list,
Comparator.comparingDouble(Employee::getSalary));
        for (Employee e : list) {
            System.out.println(e);
        }
        System.out.println("Sorted by Department:");
        Collections.sort(list,
Comparator.comparing(Employee::getDepartment));
        for (Employee e : list) {
            System.out.println(e);
        }
    }
}
```

Q15. Use TreeSet with a custom comparator to sort a list of persons by age.

A.

```
package day9;
import java.util.Comparator;
import java.util.TreeSet;
```

```
class Person {
    String name;
    int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String toString() {
        return name + " - " + age;
    }
}
public class SortPersonsByAgeUsingTreeSet {
    public static void main(String[] args) {
        TreeSet<Person> set = new
TreeSet<Person>(Comparator.comparingInt(p -> p.age));
        set.add(new Person("Neeva Sharma", 30));
        set.add(new Person("Reeva Sharma", 25));
        set.add(new Person("Seeva Sharma", 28));
        for (Person p : set) {
            System.out.println(p);
        }
    }
}
```

Q1. Create and Write to a File

Write a Java program to create a file named student.txt and write 5 lines of student names using FileWriter.

A. package day9;

```
import java.io.FileWriter;
import java.io.IOException;
public class CreateAndWriteToFile {
    public static void main(String[] args) {
        try {
            FileWriter writer = new FileWriter("student.txt");
            writer.write("Neeva Sharma\n");
            writer.write("Reeva Sharma\n");
            writer.write("Seeva Sharma\n");
            writer.write("John Doe\n");
            writer.write("Jane Doe\n");
            writer.close();
            System.out.println("File written successfully");
        } catch (IOException e) {
            System.out.println("Error writing file: " + e.getMessage());
        }
    }
}
```

Q2. Read from a File

Write a program to read the contents of student.txt and display them line by line using BufferedReader.

A. package day9;

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
```

```
public class ReadFromFile {
    public static void main(String[] args) {
        try {
            BufferedReader reader = new BufferedReader(new
FileReader("student.txt"));
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
            reader.close();
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

Q3. Append Data to a File

Write a Java program to append a new student name to the existing student.txt file without overwriting existing data.

A.

```
package day9;
import java.io.FileWriter;
import java.io.IOException;
public class AppendDataToFile {
    public static void main(String[] args) {
        try {
            FileWriter writer = new FileWriter("student.txt", true);
            writer.write("New Student\n");
            writer.close();
```

```
            System.out.println("Data appended successfully");
        } catch (IOException e) {
            System.out.println("Error appending data: " +
e.getMessage());
        }
    }
}
```

Q4. Count Words and Lines

Write a program to count the number of words and lines in a given text file notes.txt.

A.
```
package day9;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class CountWordsAndLines {
    public static void main(String[] args) {
        int lines = 0;
        int words = 0;
        try {
            BufferedReader reader = new BufferedReader(new
FileReader("notes.txt"));
            String line;
            while ((line = reader.readLine()) != null) {
                lines++;
                String[] wordArray = line.trim().split("\\s+");
                if (!line.trim().isEmpty()) {
                    words += wordArray.length;
```

```
            }
        }
        reader.close();
        System.out.println("Number of lines: " + lines);
        System.out.println("Number of words: " + words);
    } catch (IOException e) {
        System.out.println("Error reading file: " + e.getMessage());
    }
  }
}
```

Q5. Copy Contents from One File to Another

Write a program to read from source.txt and write the same content into destination.txt.

A.

```
package day9;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
public class CopyContentsFromOneFileToAnother {
   public static void main(String[] args) {
      try {
         BufferedReader reader = new BufferedReader(new
FileReader("source.txt"));
         FileWriter writer = new FileWriter("destination.txt");

         String line;
         while ((line = reader.readLine()) != null) {
```

```
            writer.write(line);
            writer.write(System.lineSeparator());
        }
        reader.close();
        writer.close();
        System.out.println("File copied successfully");
    } catch (IOException e) {
        System.out.println("Error copying file: " + e.getMessage());
    }
  }
}
```

Q6. Check if a File Exists and Display Properties

Create a program to check if report.txt exists. If it does, display its:

- Absolute path
- File name
- Writable (true/false)
- Readable (true/false)
- File size in bytes

A.
```
package day9;
import java.io.File;
public class CheckFileExistsAndDisplayProperties {
   public static void main(String[] args) {
      File file = new File("report.txt");
      if (file.exists()) {
         System.out.println("Absolute Path: " +
file.getAbsolutePath());
         System.out.println("File Name: " + file.getName());
```

```
            System.out.println("Writable: " + file.canWrite());
            System.out.println("Readable: " + file.canRead());
            System.out.println("File Size (bytes): " + file.length());
        } else {
            System.out.println("File report.txt does not exist");
        }
    }
}
```

Q7. Create a File and Accept User Input

Accept input from the user (using Scanner) and write the input to a file named userinput.txt.

A.

```
package day9;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
public class CreateFileAndAcceptUserInput {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter text to write to file:");
        String input = sc.nextLine();
        try {
            FileWriter writer = new FileWriter("userinput.txt");
            writer.write(input);
            writer.close();
            System.out.println("Input written to file");
        } catch (IOException e) {
            System.out.println("Error writing file: " + e.getMessage());
```

```
        }
        sc.close();
    }
}
```

Q8. Reverse File Content

Write a program to read a file data.txt and create another file reversed.txt containing the lines in reverse order.

A.

```
package day9;
import java.io.*;
import java.util.*;
public class ReverseFileContent {
    public static void main(String[] args) {
        try {
            BufferedReader reader = new BufferedReader(new
FileReader("data.txt"));
            List<String> lines = new ArrayList<String>();
            String line;
            while ((line = reader.readLine()) != null) {
                lines.add(line);
            }
            reader.close();
            Collections.reverse(lines);

            FileWriter writer = new FileWriter("reversed.txt");
            for (String l : lines) {
                writer.write(l);
                writer.write(System.lineSeparator());
```

```
            }
            writer.close();
            System.out.println("File reversed successfully");
        } catch (IOException e) {
            System.out.println("Error processing file: " + e.getMessage());
        }
    }
}
```

Q9. Store Objects in a File using Serialization

Create a Student class with id, name, and marks. Serialize one object and save it in a file named student.ser.

A.
```
package day9;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.Serializable;
class Student implements Serializable {
    int id;
    String name;
    int marks;
    public Student(int id, String name, int marks) {
        this.id = id;
        this.name = name;
        this.marks = marks;
    }

    public String toString() {
```

```
        return id + " - " + name + " - " + marks;
    }
}
public class SerializeStudent {
    public static void main(String[] args) {
        Student s = new Student(1, "Neeva Sharma", 95);
        try {
            FileOutputStream fos = new FileOutputStream("student.ser");
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(s);
            oos.close();
            fos.close();
            System.out.println("Student object serialized");
        } catch (IOException e) {
            System.out.println("Error serializing object: " +
e.getMessage());
        }
    }
}
```

Q10. Read Serialized Object from File

Deserialize the student.ser file and display the object's content on the console.

A. package day9;

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
public class DeserializeStudent {
```

```
    public static void main(String[] args) {
        try {
            FileInputStream fis = new FileInputStream("student.ser");
            ObjectInputStream ois = new ObjectInputStream(fis);
            Student s = (Student) ois.readObject();
            ois.close();
            fis.close();
            System.out.println("Deserialized Student:");
            System.out.println(s);
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error deserializing object: " +
e.getMessage());
        }
    }
}
```

Q11. Print All Files in a Directory

Write a program to list all files (not directories) inside a folder path given by the user.

A.
```
package day9;
import java.io.File;
import java.util.Scanner;
public class PrintAllFilesInDirectory {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter folder path: ");
        String path = sc.nextLine();
        sc.close();
```

```
        File folder = new File(path);
        if (folder.exists() && folder.isDirectory()) {
            File[] files = folder.listFiles();
            System.out.println("Files in directory:");
            for (File file : files) {
                if (file.isFile()) {
                    System.out.println(file.getName());
                }
            }
        } else {
            System.out.println("Invalid directory path");
        }
    }
}
```

Q12. Delete a File

Write a program to delete a file (given by file name) if it exists.

A.

```
package day9;
import java.io.File;
import java.util.Scanner;
public class DeleteFile {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter file name to delete: ");
        String filename = sc.nextLine();
        sc.close();
        File file = new File(filename);
```

```
        if (file.exists()) {
            if (file.delete()) {
                System.out.println(filename + " deleted successfully");
            } else {
                System.out.println("Failed to delete " + filename);
            }
        } else {
            System.out.println("File does not exist");
        }
    }
}
```

Q13. Word Search in a File

Ask the user to enter a word and check whether it exists in the file notes.txt.

A.
```
package day9;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;
public class WordSearchInFile {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter word to search: ");
        String word = sc.nextLine();
        sc.close();
        boolean found = false;
        try {
```

```
            BufferedReader reader = new BufferedReader(new
FileReader("notes.txt"));
            String line;
            while ((line = reader.readLine()) != null) {
                if (line.contains(word)) {
                    found = true;
                    break;
                }
            }
            reader.close();
            if (found) {
                System.out.println("Word '" + word + "' found in file.");
            } else {
                System.out.println("Word '" + word + "' not found in file.");
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

Q14. Replace a Word in a File

Read content from story.txt, replace all occurrences of the word "Java" with "Python", and write the updated content to updated_story.txt

A. package day9;

import java.io.BufferedReader;

import java.io.FileReader;

```
import java.io.FileWriter;
import java.io.IOException;
public class ReplaceWordInFile {
    public static void main(String[] args) {
        try {
            BufferedReader reader = new BufferedReader(new
FileReader("story.txt"));
            StringBuilder content = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                content.append(line.replace("Java",
"Python")).append(System.lineSeparator());
            }
            reader.close();
            FileWriter writer = new FileWriter("updated_story.txt");
            writer.write(content.toString());
            writer.close();
            System.out.println("Word replaced and new file created
successfully");
        } catch (IOException e) {
            System.out.println("Error processing file: " + e.getMessage());
        }
    }
}
```