1. 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.

```
import java.util.*;
class Student implements Comparable<Student> {
    int rollNo;
    String name;
    double marks;

    public Student(int rollNo, String name, double marks) {
        this.rollNo = rollNo;
        this.name = name;
        this.marks = marks;
    }
    @Override
    public int compareTo(Student other) {
        return Integer.compare(this.rollNo, other.rollNo);
    }

    @Override
    public String toString() {
        return "Roll No: " + rollNo + ", Name: " + name + ", Marks: " + marks;
    }
}

public class Main {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();
        students.add(new Student(103, "Alice", 85.5));
        students.add(new Student(101, "Bob", 90.0));
        students.add(new Student(102, "Charlie", 78.2));

        Collections.sort(students);
```

```
        for (Student s : students) {
            System.out.println(s);
        }
    }
}
```

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

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

```
import java.util.*;
class Product implements Comparable<Product> {
    int id;
    String name;
    double price;

    public Product(int id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }

    @Override
    public int compareTo(Product other) {
        return Double.compare(this.price, other.price);
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Price: " + price;
    }
}

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

```
        List<Product> products = new ArrayList<>();
        products.add(new Product(101, "Laptop", 75000.0));
        products.add(new Product(102, "Smartphone", 35000.0));
        products.add(new Product(103, "Tablet", 25000.0));

        Collections.sort(products);

        for (Product p : products) {
            System.out.println(p);
        }
    }
}
```

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

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

```
import java.util.*;
class Employee implements Comparable<Employee> {
    int id;
    String name;
    double salary;

    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }

    @Override
    public int compareTo(Employee other) {
        return this.name.compareTo(other.name);
    }

    @Override
```

```
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}

public class Main {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee(201, "John", 50000));
        employees.add(new Employee(202, "Alice", 60000));
        employees.add(new Employee(203, "Bob", 55000));

        Collections.sort(employees);

        for (Employee e : employees) {
            System.out.println(e);
        }
    }
}
```

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

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

```
import java.util.*;
class Book implements Comparable<Book> {
    int bookId;
    String title;
    String author;

    public Book(int bookId, String title, String author) {
        this.bookId = bookId;
        this.title = title;
        this.author = author;
```

```
    }

    @Override
    public int compareTo(Book other) {
        return Integer.compare(other.bookId, this.bookId); // Descending order
    }

    @Override
    public String toString() {
        return "Book ID: " + bookId + ", Title: " + title + ", Author: " + author;
    }
}

public class Main {
    public static void main(String[] args) {
        List<Book> books = new ArrayList<>();
        books.add(new Book(301, "Java Programming", "James Gosling"));
        books.add(new Book(303, "Python Basics", "Guido van Rossum"));
        books.add(new Book(302, "C++ Guide", "Bjarne Stroustrup"));

        Collections.sort(books);

        for (Book b : books) {
            System.out.println(b);
        }
    }
}
```

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

```
import java.util.*;
class Person implements Comparable<Person> {
    int id;
```

```
    String name;
    int age;

    public Person(int id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    @Override
    public int compareTo(Person other) {
        return Integer.compare(this.age, other.age); // Sort by age (ascending)
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Age: " + age;
    }
}

public class Main {
    public static void main(String[] args) {
        List<Person> people = new ArrayList<>();
        people.add(new Person(1, "Alice", 30));
        people.add(new Person(2, "Bob", 25));
        people.add(new Person(3, "Charlie", 35));

        System.out.println("Before Sorting:");
        for (Person p : people) {
            System.out.println(p);
        }

        Collections.sort(people);
```

```
        System.out.println("\nAfter Sorting (by age):");
        for (Person p : people) {
            System.out.println(p);
        }
    }
}
```

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

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

```
import java.util.*;
class Student {
    int rollNo;
    String name;
    double marks;

    public Student(int rollNo, String name, double marks) {
        this.rollNo = rollNo;
        this.name = name;
        this.marks = marks;
    }

    @Override
    public String toString() {
        return "Roll No: " + rollNo + ", Name: " + name + ", Marks: " + marks;
    }
}

class MarksDescendingComparator implements Comparator<Student> {
    @Override
    public int compare(Student s1, Student s2) {
        return Double.compare(s2.marks, s1.marks); // Descending
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();
        students.add(new Student(101, "Alice", 85.5));
        students.add(new Student(102, "Bob", 92.0));
        students.add(new Student(103, "Charlie", 78.8));

        System.out.println("Before Sorting:");
        for (Student s : students) {
            System.out.println(s);
        }

        Collections.sort(students, new MarksDescendingComparator());

        System.out.println("\nAfter Sorting by Marks (Descending):");
        for (Student s : students) {
            System.out.println(s);
        }
    }
}
```

7. Create multiple sorting strategies for a Product class.

Implement comparators to sort by:

Price ascending

Price descending

Name alphabetically

```
import java.util.*;
class Product {
    int id;
    String name;
    double price;
```

```
    public Product(int id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Price: " + price;
    }
}

class PriceAscendingComparator implements Comparator<Product> {
    @Override
    public int compare(Product p1, Product p2) {
        return Double.compare(p1.price, p2.price);
    }
}

class PriceDescendingComparator implements Comparator<Product> {
    @Override
    public int compare(Product p1, Product p2) {
        return Double.compare(p2.price, p1.price);
    }
}

class NameAlphabeticalComparator implements Comparator<Product> {
    @Override
    public int compare(Product p1, Product p2) {
        return p1.name.compareTo(p2.name);
    }
}

public class Main {
```

```
    public static void main(String[] args) {
        List<Product> products = new ArrayList<>();
        products.add(new Product(1, "Laptop", 75000));
        products.add(new Product(2, "Smartphone", 35000));
        products.add(new Product(3, "Tablet", 25000));

        System.out.println("Original List:");
        for (Product p : products) {
            System.out.println(p);
        }

        Collections.sort(products, new PriceAscendingComparator());
        System.out.println("\nSorted by Price (Ascending):");
        for (Product p : products) {
            System.out.println(p);
        }

        Collections.sort(products, new PriceDescendingComparator());
        System.out.println("\nSorted by Price (Descending):");
        for (Product p : products) {
            System.out.println(p);
        }

        Collections.sort(products, new NameAlphabeticalComparator());
        System.out.println("\nSorted by Name (Alphabetical):");
        for (Product p : products) {
            System.out.println(p);
        }
    }
}
```

8. Sort Employee objects by joining date using Comparator.

Use Comparator to sort employees based on LocalDate or Date.

```
import java.util.*;
import java.time.*;

class Employee {
    int id;
    String name;
    LocalDate joiningDate;

    public Employee(int id, String name, LocalDate joiningDate) {
        this.id = id;
        this.name = name;
        this.joiningDate = joiningDate;
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Joining Date: " + joiningDate;
    }
}

class JoiningDateComparator implements Comparator<Employee> {
    @Override
    public int compare(Employee e1, Employee e2) {
        return e1.joiningDate.compareTo(e2.joiningDate);
    }
}

public class Main {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();

        employees.add(new Employee(101, "Alice", LocalDate.of(2022, 3, 15)));
        employees.add(new Employee(102, "Bob", LocalDate.of(2020, 6, 10)));
        employees.add(new Employee(103, "Charlie", LocalDate.of(2021, 1, 25)));
```

```
        System.out.println("Before Sorting:");
        for (Employee e : employees) {
            System.out.println(e);
        }

        Collections.sort(employees, new JoiningDateComparator());

        System.out.println("\nAfter Sorting by Joining Date (Ascending):");
        for (Employee e : employees) {
            System.out.println(e);
        }
    }
}
```

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

```
import java.util.*;
class City {
    String name;
    int population;

    public City(String name, int population) {
        this.name = name;
        this.population = population;
    }

    @Override
    public String toString() {
        return "City: " + name + ", Population: " + population;
    }
}
```

```
class PopulationComparator implements Comparator<City> {
    @Override
    public int compare(City c1, City c2) {
        return Integer.compare(c1.population, c2.population);
    }
}

public class Main {
    public static void main(String[] args) {
        List<City> cities = new ArrayList<>();
        cities.add(new City("Mumbai", 20400000));
        cities.add(new City("Delhi", 19000000));
        cities.add(new City("Bangalore", 12000000));
        cities.add(new City("Chennai", 10000000));

        System.out.println("Before Sorting:");
        for (City city : cities) {
            System.out.println(city);
        }

        Collections.sort(cities, new PopulationComparator());

        System.out.println("\nAfter Sorting by Population (Ascending):");
        for (City city : cities) {
            System.out.println(city);
        }
    }
}
```

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

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
```

```
        List<String> words = new ArrayList<>();
        words.add("Apple");
        words.add("Banana");
        words.add("Kiwi");
        words.add("Pineapple");
        words.add("Mango");

        System.out.println("Before Sorting:");
        for (String word : words) {
            System.out.println(word);
        }

        Collections.sort(words, new Comparator<String>() {
            @Override
            public int compare(String s1, String s2) {
                return Integer.compare(s1.length(), s2.length()); // Ascending by length
            }
        });

        System.out.println("\nAfter Sorting by Length:");
        for (String word : words) {
            System.out.println(word);
        }
    }
}
```

11. 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.

```
import java.util.*;
class Student implements Comparable<Student> {
    int rollNo;
```

```
    String name;
    double marks;

    public Student(int rollNo, String name, double marks) {
        this.rollNo = rollNo;
        this.name = name;
        this.marks = marks;
    }

    @Override
    public int compareTo(Student other) {
        return this.name.compareTo(other.name); // Sort by name
    }

    @Override
    public String toString() {
        return "Roll No: " + rollNo + ", Name: " + name + ", Marks: " + marks;
    }
}

class MarksComparator implements Comparator<Student> {
    @Override
    public int compare(Student s1, Student s2) {
        return Double.compare(s1.marks, s2.marks); // Ascending by marks
    }
}

public class Main {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();
        students.add(new Student(101, "Alice", 85.5));
        students.add(new Student(102, "Charlie", 92.0));
        students.add(new Student(103, "Bob", 78.8));
```

```
        System.out.println("Original List:");
        for (Student s : students) {
            System.out.println(s);
        }

        Collections.sort(students); // Sort by name
        System.out.println("\nSorted by Name (using Comparable):");
        for (Student s : students) {
            System.out.println(s);
        }

        Collections.sort(students, new MarksComparator()); // Sort by marks
        System.out.println("\nSorted by Marks (using Comparator):");
        for (Student s : students) {
            System.out.println(s);
        }
    }
}
```

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

```
import java.util.*;
class Book implements Comparable<Book> {
    int bookId;
    String title;
    String author;

    public Book(int bookId, String title, String author) {
        this.bookId = bookId;
        this.title = title;
        this.author = author;
    }
```

```
    @Override
    public int compareTo(Book other) {
        return Integer.compare(this.bookId, other.bookId);
    }

    @Override
    public String toString() {
        return "Book ID: " + bookId + ", Title: " + title + ", Author: " + author;
    }
}

class TitleAuthorComparator implements Comparator<Book> {
    @Override
    public int compare(Book b1, Book b2) {
        int titleCompare = b1.title.compareTo(b2.title);
        return titleCompare != 0 ? titleCompare : b1.author.compareTo(b2.author);
    }
}

public class Main {
    public static void main(String[] args) {
        List<Book> books = new ArrayList<>();
        books.add(new Book(103, "Data Structures", "Mark Allen"));
        books.add(new Book(101, "Java Programming", "James Gosling"));
        books.add(new Book(102, "Java Programming", "Herbert Schildt"));
        books.add(new Book(104, "Algorithms", "Robert Sedgewick"));

        System.out.println("Original List:");
        for (Book b : books) {
            System.out.println(b);
        }

        Collections.sort(books);
        System.out.println("\nSorted by Book ID (using Comparable):");
```

```
        for (Book b : books) {
            System.out.println(b);
        }

        Collections.sort(books, new TitleAuthorComparator());
        System.out.println("\nSorted by Title, then Author (using Comparator):");
        for (Book b : books) {
            System.out.println(b);
        }
    }
}
```

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

```
import java.util.*;
import java.util.Scanner;
class Employee {
    int id;
    String name;
    double salary;
    String department;

    public Employee(int id, String name, double salary, String department) {
        this.id = id;
        this.name = name;
        this.salary = salary;
        this.department = department;
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Salary: " + salary + ", Department: " + department;
```

```
    }
}

class NameComparator implements Comparator<Employee> {
    public int compare(Employee e1, Employee e2) {
        return e1.name.compareTo(e2.name);
    }
}

class SalaryComparator implements Comparator<Employee> {
    public int compare(Employee e1, Employee e2) {
        return Double.compare(e1.salary, e2.salary);
    }
}

class DepartmentComparator implements Comparator<Employee> {
    public int compare(Employee e1, Employee e2) {
        return e1.department.compareTo(e2.department);
    }
}

public class Main {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee(101, "Alice", 60000, "HR"));
        employees.add(new Employee(102, "Bob", 75000, "IT"));
        employees.add(new Employee(103, "Charlie", 50000, "Finance"));
        employees.add(new Employee(104, "David", 80000, "IT"));

        Scanner scanner = new Scanner(System.in);
        boolean exit = false;

        while (!exit) {
            System.out.println("\nMenu:");
```

```
        System.out.println("1. Sort by Name");
        System.out.println("2. Sort by Salary");
        System.out.println("3. Sort by Department");
        System.out.println("4. Exit");
        System.out.print("Choose an option: ");

        int choice = scanner.nextInt();

        switch (choice) {
            case 1:
                Collections.sort(employees, new NameComparator());
                System.out.println("\nSorted by Name:");
                break;
            case 2:
                Collections.sort(employees, new SalaryComparator());
                System.out.println("\nSorted by Salary:");
                break;
            case 3:
                Collections.sort(employees, new DepartmentComparator());
                System.out.println("\nSorted by Department:");
                break;
            case 4:
                exit = true;
                System.out.println("Exiting program.");
                continue;
            default:
                System.out.println("Invalid choice.");
                continue;
        }

        for (Employee e : employees) {
            System.out.println(e);
        }
    }
```

```
        scanner.close();
    }
}
```

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

```
import java.util.*;
import java.util.stream.*;
class Employee {
    int id;
    String name;
    double salary;

    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public double getSalary() {
        return salary;
    }
    public int getId() {
        return id;
    }
    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Salary: " + salary;
```

```
    }
}
public class Main {
    public static void main(String[] args) {
        List<Employee> employees = Arrays.asList(
            new Employee(103, "Charlie", 55000),
            new Employee(101, "Alice", 60000),
            new Employee(102, "Bob", 50000)
        );

        System.out.println("Sorted by Name:");
        employees.stream()
                .sorted(Comparator.comparing(Employee::getName))
                .forEach(System.out::println);

        System.out.println("\nSorted by Salary:");
        employees.stream()
                .sorted(Comparator.comparing(Employee::getSalary))
                .forEach(System.out::println);

        System.out.println("\nSorted by ID:");
        employees.stream()
                .sorted(Comparator.comparing(Employee::getId))
                .forEach(System.out::println);
    }
}
```

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

```
import java.util.*;
class Person {
    String name;
    int age;
```

```
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public int getAge() {
        return age;
    }

    @Override
    public String toString() {
        return "Name: " + name + ", Age: " + age;
    }
}

public class Main {
    public static void main(String[] args) {
        Comparator<Person> ageComparator = Comparator.comparingInt(Person::getAge);

        Set<Person> people = new TreeSet<>(ageComparator);

        people.add(new Person("Alice", 30));
        people.add(new Person("Bob", 25));
        people.add(new Person("Charlie", 35));
        people.add(new Person("David", 28));

        for (Person p : people) {
            System.out.println(p);
        }
    }
}
```

1. 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.

```
import java.io.FileWriter;
import java.io.IOException;


public class Main {
   public static void main(String[] args) {
      try {
         FileWriter writer = new FileWriter("student.txt");
         writer.write("Alice\n");
         writer.write("Bob\n");
         writer.write("Charlie\n");
         writer.write("David\n");
         writer.write("Eve\n");
         writer.close();
         System.out.println("Successfully wrote to student.txt");
      }
 catch (IOException e) {
         System.out.println("An error occurred.");
         e.printStackTrace();
      }
   }
}
```

2. Read from a File

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

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

```
public class Main {
    public static void main(String[] args) {
        try (BufferedReader br = new BufferedReader(new FileReader("student.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}
```

3. 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.

```
import java.io.FileWriter;
import java.io.IOException;
public class Main {
    public static void main(String[] args) {
        try (FileWriter writer = new FileWriter("student.txt", true)) { // true for append mode
            writer.write("Frank\n");
            System.out.println("Successfully appended to student.txt");
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}
```

4. Count Words and Lines

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

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

public class Main {
    public static void main(String[] args) {
        int lineCount = 0;
        int wordCount = 0;
        try (BufferedReader br = new BufferedReader(new FileReader("notes.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                lineCount++;
                String[] words = line.trim().split("\\s+");
                if (line.trim().length() > 0) {
                    wordCount += words.length;
                }
            }
            System.out.println("Lines: " + lineCount);
            System.out.println("Words: " + wordCount);
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}
```

5. Copy Contents from One File to Another

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

```
import java.io.*;
public class Main {
    public static void main(String[] args) {
        try (BufferedReader br = new BufferedReader(new FileReader("source.txt"));
```

```
         BufferedWriter bw = new BufferedWriter(new FileWriter("destination.txt"))) {

            String line;
            while ((line = br.readLine()) != null) {
                bw.write(line);
                bw.newLine();
            }
            System.out.println("File copied successfully.");
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}
```

6. 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**

```
import java.io.File;
public class Main {
    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("report.txt does not exist.");
        }
    }
}
```

7. Create a File and Accept User Input

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

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(System.in);
             FileWriter writer = new FileWriter("userinput.txt")) {

            System.out.println("Enter text (type 'exit' to finish):");

            while (true) {
                String input = scanner.nextLine();
                if (input.equalsIgnoreCase("exit")) {
                    break;
                }
                writer.write(input + System.lineSeparator());
            }

            System.out.println("Input saved to userinput.txt");
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
```

```
    }
}
```

8. Reverse File Content

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

```
import java.io.*;
import java.util.*;
public class Main {
    public static void main(String[] args) {
        List<String> lines = new ArrayList<>();

        try (BufferedReader br = new BufferedReader(new FileReader("data.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                lines.add(line);
            }
        } catch (IOException e) {
            System.out.println("Error reading data.txt");
            e.printStackTrace();
            return;
        }

        Collections.reverse(lines);

        try (BufferedWriter bw = new BufferedWriter(new FileWriter("reversed.txt"))) {
            for (String line : lines) {
                bw.write(line);
                bw.newLine();
            }
            System.out.println("Reversed content written to reversed.txt");
        } catch (IOException e) {
```

```
            System.out.println("Error writing reversed.txt");
            e.printStackTrace();
        }
    }
}
```

9. 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.

```
import java.io.*;
class Student implements Serializable {
    int id;
    String name;
    double marks;

    public Student(int id, String name, double marks) {
        this.id = id;
        this.name = name;
        this.marks = marks;
    }
}

public class Main {
    public static void main(String[] args) {
        Student student = new Student(101, "Alice", 89.5);

        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("student.ser"))) {
            oos.writeObject(student);
            System.out.println("Student object serialized to student.ser");
        } catch (IOException e) {
            System.out.println("Serialization error");
            e.printStackTrace();
```

```
        }
    }
}
```

10. Read Serialized Object from File

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

```
import java.io.*;
class Student implements Serializable {
    int id;
    String name;
    double marks;

    public Student(int id, String name, double marks) {
        this.id = id;
        this.name = name;
        this.marks = marks;
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Marks: " + marks;
    }
}

public class Main {
    public static void main(String[] args) {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("student.ser"))) {
            Student student = (Student) ois.readObject();
            System.out.println("Deserialized Student:");
            System.out.println(student);
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Deserialization error");
```

```
            e.printStackTrace();
        }
    }
}
```

11. Print All Files in a Directory

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

```
import java.io.File;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter folder path: ");
        String folderPath = scanner.nextLine();

        File folder = new File(folderPath);

        if (folder.exists() && folder.isDirectory()) {
            File[] files = folder.listFiles(file -> file.isFile());
            if (files != null && files.length > 0) {
                System.out.println("Files in directory:");
                for (File file : files) {
                    System.out.println(file.getName());
                }
            } else {
                System.out.println("No files found in the directory.");
            }
        } else {
            System.out.println("Invalid folder path.");
        }

        scanner.close();
```

```
    }
}
```

12. Delete a File

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

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

13. Word Search in a File

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

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

```
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter word to search: ");
        String word = scanner.nextLine();

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

14. 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

```
import java.io.*;
public class Main {
    public static void main(String[] args) {
```

```
        StringBuilder content = new StringBuilder();

        try (BufferedReader br = new BufferedReader(new FileReader("story.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                content.append(line.replace("Java", "Python")).append(System.lineSeparator());
            }
        } catch (IOException e) {
            System.out.println("Error reading story.txt");
            e.printStackTrace();
            return;
        }

        try (BufferedWriter bw = new BufferedWriter(new FileWriter("updated_story.txt"))) {
            bw.write(content.toString());
            System.out.println("Updated content written to updated_story.txt");
        } catch (IOException e) {
            System.out.println("Error writing updated_story.txt");
            e.printStackTrace();
        }
    }
}
```