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.

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

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

    public int getRollNo() {
        return rollNo;
    }

    public String getName() {
        return name;
    }

    public double getMarks() {
        return marks;
    }

    public int compareTo(Student other) {
        return Integer.compare(this.rollNo, other.rollNo);
    }

    public String toString() {
        return "RollNo: " + rollNo + ", Name: " + name + ", Marks: " + marks;
    }
}

public class StudentSortDemo {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();

        students.add(new Student(3, "Ajay", 85.5));
        students.add(new Student(1, "vinay", 90.0));
        students.add(new Student(2, "vijay", 78.3));

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

        Collections.sort(students); // Uses compareTo()

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

Output:

```
Before Sorting:
RollNo: 3, Name: Ajay, Marks: 85.5
RollNo: 1, Name: vinay, Marks: 90.0
RollNo: 2, Name: vijay, Marks: 78.3

After Sorting by Roll Number (Ascending):
RollNo: 1, Name: vinay, Marks: 90.0
RollNo: 2, Name: vijay, Marks: 78.3
RollNo: 3, Name: Ajay, Marks: 85.5
```

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

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

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

class Product implements Comparable<Product> {
    private String name;
    private double price;

    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }
    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }


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

    public String toString() {
        return name + "- $" + price;
    }
}

public class ProductSortDemo {
    public static void main(String[] args) {
        List<Product> products = new ArrayList<>();

        products.add(new Product("Laptop", 1200.50));
        products.add(new Product("Smartphone", 799.99));
        products.add(new Product("Tablet", 450.00));
        products.add(new Product("Monitor", 300.00));

        System.out.println("Before Sorting:");
        for (Product p : products) {
            System.out.println(p);
        }

        Collections.sort(products);

        System.out.println("\nAfter Sorting by Price (Ascending):");
        for (Product p : products) {
            System.out.println(p);
        }
    }
}
```

Output:

```
Before Sorting:
Laptop- 60000
Smartphone- 15000
Tablet- 10000
Monitor 25000

After Sorting by Price (Ascending):
Tablet- 10000
Smartphone- 15000
Monitor 25000
Laptop- 60000
```

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

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

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
class Employee implements Comparable<Employee> {
    private int id;
    private String name;
    private double salary;

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

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public double getSalary() {
        return salary;
    }
    public int compareTo(Employee other) {
        return this.name.compareTo(other.name);
    }

    public String toString() {
        return id + "- " + name + "- $" + salary;
    }
}

public class EmployeeSortByName {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee(101, “ajay", 50000));
        employees.add(new Employee(103, "vinay", 60000));
        employees.add(new Employee(102, "rahul", 55000));
        employees.add(new Employee(104, "uday", 70000));

        System.out.println("Before Sorting:");
        for (Employee e : employees) {
            System.out.println(e);
        }
        Collections.sort(employees);
        System.out.println("\nAfter Sorting by Name (Alphabetically):");
        for (Employee e : employees) {
            System.out.println(e);
        }
    }
}
```

Output:

```
Before Sorting:
101- ajay- 50000
103- vinay- 60000
102- rahul- 55000
104- uday- 70000

After Sorting by Name (Alphabetically):
101- ajay- 50000
102- rahul- 55000
104- uday- 70000
103- vinay- 60000
```

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

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

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

class Book implements Comparable<Book> {
    private int bookId;
    private String title;
    private String author;

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

    public int getBookId() {
        return bookId;
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

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

    public String toString() {
        return bookId + "- " + title + " by " + author;
    }
}

public class BookSortDescending {
    public static void main(String[] args) {
        List<Book> books = new ArrayList<>();

        books.add(new Book(101, "Java Basics", "John Smith"));
        books.add(new Book(105, "Data Structures", "Alice Brown"));
        books.add(new Book(103, "Algorithms", "Bob White"));
        books.add(new Book(110, "Database Systems", "Clara Green"));

        System.out.println("Before Sorting:");
        for (Book b : books) {
            System.out.println(b);
        }

        Collections.sort(books); // Uses compareTo()

        System.out.println("\nAfter Sorting by Book ID (Descending):");
        for (Book b : books) {
            System.out.println(b);
        }
    }
}
```

Output:

```
Before Sorting:
101- Java Basics by John Smith
105- Data Structures by Alice Brown
103- Algorithms by Bob White
110- Database Systems by Clara Green

After Sorting by Book ID (Descending):
110- Database Systems by Clara Green
105- Data Structures by Alice Brown
103- Algorithms by Bob White
101- Java Basics by John Smith
```

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

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

class Student implements Comparable<Student> {
    private int rollNo;
    private String name;
    private double marks;

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

    public int getRollNo() {
        return rollNo;
    }

    public String getName() {
        return name;
    }

    public double getMarks() {
        return marks;
    }
    public int compareTo(Student other) {
        return Double.compare(this.marks, other.marks);
    }

    public String toString() {
        return rollNo + "- " + name + "- " + marks;
    }
}

public class ComparableSortDemo {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();

        students.add(new Student(101, "Ajay", 85.5));
        students.add(new Student(102, "vijay", 92.0));
        students.add(new Student(103, "vinay", 78.2));
        students.add(new Student(104, "rahul", 88.8));

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

        Collections.sort(students); // Sort using Comparable

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

Output:

```
Before Sorting:
101- Ajay- 85.5
102- vijay- 92.0
103- vinay- 78.2
104- rahul- 88.8

After Sorting (by Marks Ascending):
103- vinay- 78.2
101- Ajay- 85.5
104- rahul- 88.8
102- vijay- 92.0
```

Q6. 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 {
    private int rollNo;
    private String name;
    private double marks;

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

    public int getRollNo() {
        return rollNo;
    }

    public String getName() {
        return name;
    }

    public double getMarks() {
        return marks;
    }

    public String toString() {
        return rollNo + "- " + name + "- " + marks;
    }
}

class MarksDescendingComparator implements Comparator<Student> {
    public int compare(Student s1, Student s2) {
        return Double.compare(s2.getMarks(), s1.getMarks()); // reverse order
    }
}

public class ComparatorSortDemo {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();
        students.add(new Student(101, "Ajay", 85.5));
        students.add(new Student(102, "vijay", 92.0));
        students.add(new Student(103, "vinay", 78.2));
        students.add(new Student(104, "rahul", 88.8));

        System.out.println("Before Sorting:");
        students.forEach(System.out::println);
        Collections.sort(students, new MarksDescendingComparator());

        System.out.println("\nAfter Sorting (Descending by Marks- Comparator Class):");
        students.forEach(System.out::println);
        students.sort((s1, s2)-> Double.compare(s2.getMarks(), s1.getMarks()));

        System.out.println("\nAfter Sorting (Descending by Marks- Lambda):");
        students.forEach(System.out::println);
    }
}
```

Output:

```
Before Sorting:
101- Ajay- 85.5
102- vijay- 92.0
103- vinay- 78.2
104- rahul- 88.8

After Sorting (Descending by Marks- Comparator Class):
102- vijay- 92.0
104- rahul- 88.8
101- Ajay- 85.5
```

```
103- vinay- 78.2

After Sorting (Descending by Marks- Lambda):
102- vijay- 92.0
104- rahul- 88.8
101- Ajay- 85.5
103- vinay- 78.2
```

Q7. Create multiple sorting strategies for a Product class.

Implement comparators to sort by:

Price ascending

Price descending

Name alphabetically

```
import java.util.*;

class Product {
   private String name;
   private double price;

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

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

```
    public double getPrice() {
        return price;
    }
    public String toString() {
        return name + "- $" + price;
    }
}

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

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

class NameAlphabeticalComparator implements Comparator<Product> {

    public int compare(Product p1, Product p2) {
        return p1.getName().compareToIgnoreCase(p2.getName());
    }
}

public class ProductSortDemo {
    public static void main(String[] args) {
        List<Product> products = new ArrayList<>();
        products.add(new Product("Laptop", 1200.50));
        products.add(new Product("Phone", 800.00));
        products.add(new Product("Tablet", 400.75));
        products.add(new Product("Monitor", 300.25));

        System.out.println("Original List:");
        products.forEach(System.out::println);

        Collections.sort(products, new PriceAscendingComparator());
        System.out.println("\nSorted by Price (Ascending):");
        products.forEach(System.out::println);

        Collections.sort(products, new PriceDescendingComparator());
        System.out.println("\nSorted by Price (Descending):");
        products.forEach(System.out::println);

        Collections.sort(products, new NameAlphabeticalComparator());
        System.out.println("\nSorted by Name (Alphabetical):");
        products.forEach(System.out::println);

        products.sort((p1, p2) -> Double.compare(p1.getPrice(), p2.getPrice()));
        System.out.println("\nSorted by Price (Ascending- Lambda):");
        products.forEach(System.out::println);
    }
}
```

Output:

```
Original List:
Laptop- $1200.5
```

```
Phone- $800.0
Tablet- $400.75
Monitor- $300.25

Sorted by Price (Ascending):
Monitor- $300.25
Tablet- $400.75
Phone- $800.0
Laptop- $1200.5

Sorted by Price (Descending):
Laptop- $1200.5
Phone- $800.0
Tablet- $400.75
Monitor- $300.25

Sorted by Name (Alphabetical):
Laptop- $1200.5
Monitor- $300.25
Phone- $800.0
Tablet- $400.75

Sorted by Price (Ascending- Lambda):
Monitor- $300.25
Tablet- $400.75
Phone- $800.0
Laptop- $1200.5
```

Q8. Sort Employee objects by joining date using Comparator.

Use Comparator to sort employees based on LocalDate or Date.

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

class Employee {
    private String name;
    private LocalDate joiningDate;

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

    public String getName() {
        return name;
    }

    public LocalDate getJoiningDate() {
        return joiningDate;
    }
    public String toString() {
        return name + " (Joined: " + joiningDate + ")";
    }
}

public class EmployeeSortByDate {
```

```java
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee("Alice", LocalDate.of(2020, 5, 10)));
        employees.add(new Employee("Bob", LocalDate.of(2018, 3, 25)));
        employees.add(new Employee("Charlie", LocalDate.of(2022, 8, 15)));
        employees.add(new Employee("David", LocalDate.of(2019, 11, 1)));

        System.out.println("Original List:");
        employees.forEach(System.out::println);



        employees.sort(Comparator.comparing(Employee::getJoiningDate));
        System.out.println("\nSorted by Joining Date (Oldest First):");
        employees.forEach(System.out::println);

        employees.sort(Comparator.comparing(Employee::getJoiningDate).reversed());
        System.out.println("\nSorted by Joining Date (Most Recent First):");
        employees.forEach(System.out::println);
    }
}
```

Output:

```
Original List:
Alice (Joined: 2020-05-10)
Bob (Joined: 2018-03-25)
Charlie (Joined: 2022-08-15)
David (Joined: 2019-11-01)

Sorted by Joining Date (Oldest First):
Bob (Joined: 2018-03-25)
```

```
David (Joined: 2019-11-01)
Alice (Joined: 2020-05-10)
Charlie (Joined: 2022-08-15)

Sorted by Joining Date (Most Recent First):
Charlie (Joined: 2022-08-15)
Alice (Joined: 2020-05-10)
David (Joined: 2019-11-01)
Bob (Joined: 2018-03-25)
```

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

```
import java.util.*;

class City {
    private String name;
    private int population;

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

    public String getName() {
        return name;
    }

    public int getPopulation() {
        return population;
    }

    public String toString() {
        return name + " (Population: " + population + ")";
    }
}

public class CitySortByPopulation {
    public static void main(String[] args) {
        List<City> cities = new ArrayList<>();
        cities.add(new City("New York", 8419600));
        cities.add(new City("Tokyo", 13929286));
        cities.add(new City("Mumbai", 20411000));
        cities.add(new City("Paris", 2148327));
        cities.add(new City("London", 8982000));

        System.out.println("Original List:");
        cities.forEach(System.out::println);
        cities.sort(Comparator.comparingInt(City::getPopulation));
        System.out.println("\nSorted by Population (Ascending):");
        cities.forEach(System.out::println);



        cities.sort(Comparator.comparingInt(City::getPopulation).reversed());
        System.out.println("\nSorted by Population (Descending):");
        cities.forEach(System.out::println);
    }
}
```

Output:

```
Original List:
New York (Population: 8419600)
Tokyo (Population: 13929286)
Mumbai (Population: 20411000)
Paris (Population: 2148327)
London (Population: 8982000)

Sorted by Population (Ascending):
Paris (Population: 2148327)
New York (Population: 8419600)
London (Population: 8982000)
Tokyo (Population: 13929286)
Mumbai (Population: 20411000)

Sorted by Population (Descending):
Mumbai (Population: 20411000)
Tokyo (Population: 13929286)
London (Population: 8982000)
New York (Population: 8419600)
Paris (Population: 2148327)
```

Q10. Sort a list of Book objects using both Comparable (by ID) and Comparator (by title, then author).

```java
import java.util.*;

class Book implements Comparable<Book> {
    private int bookId;
    private String title;
    private String author;

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

    public int getBookId() {
        return bookId;
    }
    public String getTitle() {
        return title;
    }
    public String getAuthor() {
        return author;
    }

    public int compareTo(Book other) {

        return Integer.compare(this.bookId, other.bookId);
    }

    public String toString() {
        return "BookID: " + bookId + ", Title: " + title + ", Author: " + author;
    }
}

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

        System.out.println("Original List:");
        books.forEach(System.out::println);

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



        Comparator<Book> byTitleThenAuthor = Comparator
                .comparing(Book::getTitle)
                .thenComparing(Book::getAuthor);

        books.sort(byTitleThenAuthor);
        System.out.println("\nSorted by Title, then Author (Comparator):");
        books.forEach(System.out::println);
    }
}
```

Output:

```
Original List:
BookID: 102, Title: Java Programming, Author: James Gosling
BookID: 101, Title: Data Structures, Author: Mark Allen
BookID: 103, Title: Algorithms, Author: Thomas Cormen
```

```
BookID: 104, Title: Java Programming, Author: Herbert Schildt

Sorted by Book ID (Comparable):
BookID: 101, Title: Data Structures, Author: Mark Allen
BookID: 102, Title: Java Programming, Author: James Gosling
BookID: 103, Title: Algorithms, Author: Thomas Cormen
BookID: 104, Title: Java Programming, Author: Herbert Schildt

Sorted by Title, then Author (Comparator):
BookID: 103, Title: Algorithms, Author: Thomas Cormen
BookID: 101, Title: Data Structures, Author: Mark Allen
BookID: 104, Title: Java Programming, Author: Herbert Schildt
BookID: 102, Title: Java Programming, Author: James Gosling
```

Q11. Write a menu-driven program to sort Employee objects by name, salary, or department using Comparator.

```
import java.util.*;

class Employee {
    private String name;
    private double salary;
    private 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 String.format("Name: %-10s Salary: %.2f Department: %s",
                name, salary, department);
    }
}

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

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

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

```
public class EmployeeSorter {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee("Alice", 60000, "HR"));
        employees.add(new Employee("Bob", 75000, "IT"));
        employees.add(new Employee("Charlie", 50000, "Finance"));
        employees.add(new Employee("David", 70000, "IT"));
        employees.add(new Employee("Eve", 55000, "HR"));

        Scanner sc = new Scanner(System.in);
        int choice;

        do {
            System.out.println("\n--- Employee Sorting Menu ---");
            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("Enter your choice: ");
            choice = sc.nextInt();

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

        sc.close();
    }
}
```

Output:

```
--- Employee Sorting Menu---
1. Sort by Name
2. Sort by Salary
3. Sort by Department
4. Exit
Enter your choice: 1

Sorted by Name:
Name: Alice     Salary: 60000.00 Department: HR
```

```
Name: Bob       Salary: 75000.00 Department: IT

Name: Charlie   Salary: 50000.00 Department: Finance

Name: David     Salary: 70000.00 Department: IT

Name: Eve       Salary: 55000.00 Department: HR


--- Employee Sorting Menu---

1. Sort by Name

2. Sort by Salary

3. Sort by Department

4. Exit

Enter your choice: 2


Sorted by Salary:

Name: Charlie   Salary: 50000.00 Department: Finance

Name: Eve       Salary: 55000.00 Department: HR

Name: Alice     Salary: 60000.00 Department: HR

Name: David     Salary: 70000.00 Department: IT

Name: Bob       Salary: 75000.00 Department: IT


--- Employee Sorting Menu---

1. Sort by Name

2. Sort by Salary

3. Sort by Department

4. Exit

Enter your choice: 3


Sorted by Department:

Name: Charlie   Salary: 50000.00 Department: Finance

Name: Alice     Salary: 60000.00 Department: HR
```

```
Name: Eve       Salary: 55000.00 Department: HR
Name: Bob       Salary: 75000.00 Department: IT
Name: David     Salary: 70000.00 Department: IT

--- Employee Sorting Menu ---
1. Sort by Name
2. Sort by Salary
3. Sort by Department
4. Exit
Enter your choice: 4
Exiting...
```

Q12. Use TreeSet with a custom comparator to sort a list of persons by age.

```
import java.util.*;

class Person {
    private String name;
    private int age;

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

    public String getName() { return name; }
    public int getAge() { return age; }
    public String toString() {
        return String.format("%s (%d years)", name, age);
    }
}

public class PersonAgeSort {
    public static void main(String[] args) {
        Comparator<Person> ageComparator = new Comparator<>() {

            public int compare(Person p1, Person p2) {
                return Integer.compare(p1.getAge(), p2.getAge());
            }
        };

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


        persons.add(new Person("Alice", 30));
        persons.add(new Person("Bob", 25));
        persons.add(new Person("Charlie", 35));
        persons.add(new Person("David", 28));
        persons.add(new Person("Eve", 25)); // Duplicate age will be considered equal → skipped

        System.out.println("Persons sorted by age:");
        for (Person p : persons) {
            System.out.println(p);
        }
    }
}
```

```
Output:

Persons sorted by age:

Bob (25 years)

David (28 years)

Alice (30 years)

Charlie (35 years)
```

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.

```
import java.io.FileWriter;

import java.io.IOException;


public class WriteStudentFile {

  public static void main(String[] args) {

    String fileName = "student.txt";


    try (FileWriter writer = new FileWriter(fileName)) {

      writer.write("Ajay\n");

      writer.write("vijay\n");

      writer.write("vinay\n");

      writer.write("uday\n");

      writer.write("rahul\n");


      System.out.println("Successfully wrote student names to " + fileName);

    } catch (IOException e) {

      System.out.println("An error occurred while writing to the file.");

      e.printStackTrace();
```

```
        }
    }
}
```

Output:

```
Successfully wrote student names to student.txt

Ajay

vijay

vinay

uday

rahul
```

Q2. 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 ReadStudentFile {

    public static void main(String[] args) {

        String fileName = "student.txt";


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

            String line;

            System.out.println("Contents of " + fileName + ":");


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

                System.out.println(line);
            }

        } catch (IOException e) {
            System.out.println("An error occurred while reading the file.");
            e.printStackTrace();
        }
    }
}
```

Output:

```
Contents of student.txt:
Ajay
vijay
vinay
uday
rahul
```

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.

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

public class AppendStudentFile {
    public static void main(String[] args) {
        String fileName = "student.txt";

        try (FileWriter writer = new FileWriter(fileName, true)) {
            writer.write("anil\n");
            System.out.println("Successfully appended new student to " + fileName);
        } catch (IOException e) {
            System.out.println("An error occurred while appending to the file.");
            e.printStackTrace();
        }
    }
}
```

Output:

```
Ajay
vijay
vinay
uday
rahul
anil
```

Q4. 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 CountWordsLines {
    public static void main(String[] args) {
        String fileName = "notes.txt";
        int lineCount = 0;
        int wordCount = 0;

        try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
            String line;
            while ((line = br.readLine()) != null) {
                lineCount++;


                String[] words = line.trim().split("\\s+");


                if (!line.trim().isEmpty()) {
                    wordCount += words.length;
                }
            }

            System.out.println("Number of lines: " + lineCount);
            System.out.println("Number of words: " + wordCount);

        } catch (IOException e) {
            System.out.println("An error occurred while reading the file.");
            e.printStackTrace();
        }
    }
}
```

Output:

If notes.txt contains

```
Hello world
This is a sample text file
It has multiple lines
```

Then output will be

```
Number of lines: 3
Number of words: 10
```

Q5. 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.FileReader;

import java.io.FileWriter;

import java.io.IOException;


public class CopyFile {

    public static void main(String[] args) {

        String sourceFile = "source.txt";

        String destinationFile = "destination.txt";


        try (

            FileReader fr = new FileReader(sourceFile);

            FileWriter fw = new FileWriter(destinationFile)

        ) {

            int ch;

            while ((ch = fr.read()) !=-1) {

                fw.write(ch); // write character to destination

            }

            System.out.println("File copied successfully from " + sourceFile + " to " + destinationFile);

        } catch (IOException e) {

            System.out.println("An error occurred while copying the file.");

            e.printStackTrace();

        }

    }

}
```

Output:

If source.txt contains

Hello World

This is a test file.

Then after running the program, the destination.txt contains:

Hello World

This is a test file.

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

```
import java.io.File;

public class FileInfo {
    public static void main(String[] args) {
        File file = new File("report.txt");

        if (file.exists()) {
            System.out.println("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 does not exist.");
    }
  }
}
```

```
Output:
If report.txt exists:
File exists!
Absolute path: C:\Users\ Documents\report.txt
File name: report.txt
Writable: true
Readable: true
File size (bytes): 256
```

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.

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

public class UserInputToFile {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String fileName = "userinput.txt";

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

```
        try (FileWriter writer = new FileWriter(fileName)) {
            while (true) {
                String line = sc.nextLine();
                if (line.equalsIgnoreCase("exit")) {
                    break;
                }
                writer.write(line + System.lineSeparator());
            }
            System.out.println("User input saved to " + fileName);
        } catch (IOException e) {
            System.out.println("An error occurred while writing to the file.");
            e.printStackTrace();
        }

        sc.close();
    }
}
```

Output:

Example console run:

```
Enter text (type 'exit' to finish):
Hello
This is a test.
exit
User input saved to userinput.txt
```

Then userinput.txt file will contain:

```
Hello
This is a test.
```

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.

```
import java.io.*;

import java.util.*;


public class ReverseFileLines {

  public static void main(String[] args) {

    String inputFile = "data.txt";

    String outputFile = "reversed.txt";


    List<String> lines = new ArrayList<>();


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

      String line;

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

        lines.add(line);

      }

    } catch (IOException e) {

      System.out.println("Error reading from " + inputFile);

      e.printStackTrace();

      return;

    }



    Collections.reverse(lines);


    try (BufferedWriter bw = new BufferedWriter(new FileWriter(outputFile))) {
            for (String reversedLine : lines) {
                bw.write(reversedLine);
                bw.newLine();
            }
            System.out.println("Reversed lines written to " + outputFile);
        } catch (IOException e) {
            System.out.println("Error writing to " + outputFile);
            e.printStackTrace();
        }
    }
}
```

Output:

If data.txt contains:

```
Line 1
Line 2
Line 3
```

After running the program, reversed.txt will contain:

```
Line 3
Line 2
Line 1
```

Q9. 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 ListFilesInFolder {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter folder path: ");
        String folderPath = sc.nextLine();

        File folder = new File(folderPath);

        if (folder.exists() && folder.isDirectory()) {
            File[] files = folder.listFiles();

            if (files != null) {
                System.out.println("Files in folder \"" + folderPath + "\":");
                boolean foundFile = false;
                for (File file : files) {
                    if (file.isFile()) {
                        System.out.println(file.getName());
                        foundFile = true;
                    }
                }
                if (!foundFile) {
                    System.out.println("No files found in the directory.");
                }
            } else {
                System.out.println("Unable to access files in the directory.");
            }
        } else {
            System.out.println("The path is not a valid directory.");
        }

        sc.close();
    }
```

```
}
```

Output:

```
Enter folder path: /home/user/Documents
Files in folder "/home/user/Documents":
report.pdf
notes.txt
image.jpg
todo.txt
```

Q10. 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 DeleteFile {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the filename to delete: ");
        String fileName = sc.nextLine();

        File file = new File(fileName);

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

        sc.close();
    }
}
```

Output:

```
Enter the filename to delete: example.txt
File 'example.txt' deleted successfully.
```

Q11. 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 WordSearchInFile {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the word to search: ");
        String wordToFind = sc.nextLine();

        String fileName = "notes.txt";
        boolean found = false;

        try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
            String line;
            while ((line = br.readLine()) != null) {
                if (line.toLowerCase().contains(wordToFind.toLowerCase())) {
                    found = true;
                    break;
                }
            }
        } catch (IOException e) {
            System.out.println("An error occurred while reading the file.");
            e.printStackTrace();
            sc.close();
            return;
        }

        if (found) {
            System.out.println("The word \"" + wordToFind + "\" exists in the file.");
        } else {
            System.out.println("The word \"" + wordToFind + "\" does NOT exist in the file.");
        }

        sc.close();
    }
}
```

Output:

```
Enter the word to search: hello
The word "hello" exists in the file.
```

Q12. 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 ReplaceWordInFile {
    public static void main(String[] args) {
        String inputFile = "story.txt";
        String outputFile = "updated_story.txt";


        try (
            BufferedReader br = new BufferedReader(new FileReader(inputFile));
            BufferedWriter bw = new BufferedWriter(new FileWriter(outputFile))
        ) {
            String line;
            while ((line = br.readLine()) != null) {
                String updatedLine = line.replace("Java", "Python");
                bw.write(updatedLine);
                bw.newLine();
            }
            System.out.println("All occurrences of \"Java\" replaced with \"Python\" in " +
outputFile);
        } catch (IOException e) {
            System.out.println("An error occurred during file processing.");
            e.printStackTrace();
        }
    }
}
```

Output:

If you run the program with a story.txt containing:

```
Java is a popular programming language.
Many developers enjoy Java because of its portability.
Learning Java opens many career opportunities.
```

The console output will be:

```
All occurrences of "Java" replaced with "Python" in updated_story.txt
```

And the file updated_story.txt will contain:

```
Python is a popular programming language.
Many developers enjoy Python because of its portability.
Learning Python opens many career opportunities.
```