# Day – 9

## Comparable and comparator

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.

Code:

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 s) {  
 return this.rollNo - s.rollNo;  
 }  
   
 @Override  
 public String toString() {  
 return "RollNo: " + rollNo + ", Name: " + name + ", Marks: " + marks;  
 }  
}  
  
public class SortStudentsByRoll {  
 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", 76.0));  
 students.add(new Student(102, "Charlie", 92.3));  
   
 Collections.sort(students);  
 students.forEach(System.out::println);  
 }  
}

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

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

Code:

import java.util.\*;  
  
class Product implements Comparable<Product> {  
 String name;  
 double price;  
   
 public Product(String name, double price) {  
 this.name = name;  
 this.price = price;  
 }  
   
 @Override  
 public int compareTo(Product p) {  
 return Double.compare(this.price, p.price);  
 }  
   
 @Override  
 public String toString() {  
 return name + " - $" + price;  
 }  
}  
  
public class SortProductsByPrice {  
 public static void main(String[] args) {  
 List<Product> products = new ArrayList<>();  
 products.add(new Product("Laptop", 999.99));  
 products.add(new Product("Phone", 699.99));  
 products.add(new Product("Tablet", 399.99));  
   
 Collections.sort(products);  
 products.forEach(System.out::println);  
 }  
}

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

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

Code:

import java.util.\*;  
  
class Employee implements Comparable<Employee> {  
 String name;  
 int id;  
   
 public Employee(String name, int id) {  
 this.name = name;  
 this.id = id;  
 }  
   
 @Override  
 public int compareTo(Employee e) {  
 return this.name.compareTo(e.name);  
 }  
   
 @Override  
 public String toString() {  
 return "ID: " + id + ", Name: " + name;  
 }  
}  
  
public class SortEmployeesByName {  
 public static void main(String[] args) {  
 List<Employee> employees = new ArrayList<>();  
 employees.add(new Employee("John", 101));  
 employees.add(new Employee("Alice", 102));  
 employees.add(new Employee("Bob", 103));  
   
 Collections.sort(employees);  
 employees.forEach(System.out::println);  
 }  
}

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

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

Code:

import java.util.\*;  
  
class Book implements Comparable<Book> {  
 int bookId;  
 String title;  
   
 public Book(int bookId, String title) {  
 this.bookId = bookId;  
 this.title = title;  
 }  
   
 @Override  
 public int compareTo(Book b) {  
 return b.bookId - this.bookId; // Descending order  
 }  
   
 @Override  
 public String toString() {  
 return "ID: " + bookId + ", Title: " + title;  
 }  
}  
  
public class SortBooksByIdDesc {  
 public static void main(String[] args) {  
 List<Book> books = new ArrayList<>();  
 books.add(new Book(101, "Java Programming"));  
 books.add(new Book(103, "Data Structures"));  
 books.add(new Book(102, "Algorithms"));  
   
 Collections.sort(books);  
 books.forEach(System.out::println);  
 }  
}

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

Code:

import java.util.\*;  
  
class Person implements Comparable<Person> {  
 String name;  
 int age;  
   
 public Person(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
   
 @Override  
 public int compareTo(Person p) {  
 return this.name.compareTo(p.name);  
 }  
   
 @Override  
 public String toString() {  
 return name + " (" + age + ")";  
 }  
}  
  
public class SortCustomObjects {  
 public static void main(String[] args) {  
 List<Person> people = new ArrayList<>();  
 people.add(new Person("John", 25));  
 people.add(new Person("Alice", 30));  
 people.add(new Person("Bob", 22));  
   
 System.out.println("Before sorting:");  
 people.forEach(System.out::println);  
   
 Collections.sort(people);  
   
 System.out.println("\nAfter sorting:");  
 people.forEach(System.out::println);  
 }  
}

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

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

Code:

import java.util.\*;  
  
class Student {  
 String name;  
 double marks;  
   
 public Student(String name, double marks) {  
 this.name = name;  
 this.marks = marks;  
 }  
   
 @Override  
 public String toString() {  
 return name + ": " + marks;  
 }  
}  
  
public class SortStudentsByMarks {  
 public static void main(String[] args) {  
 List<Student> students = new ArrayList<>();  
 students.add(new Student("Alice", 85.5));  
 students.add(new Student("Bob", 76.0));  
 students.add(new Student("Charlie", 92.3));  
   
 // Using lambda expression  
 students.sort((s1, s2) -> Double.compare(s2.marks, s1.marks)); // Descending  
   
 students.forEach(System.out::println);  
 }  
}

7. Create multiple sorting strategies for a Product class.

Implement comparators to sort by:

Price ascending

Price descending

Name alphabetically

Code:

import java.util.\*;  
  
class Product {  
 String name;  
 double price;  
   
 public Product(String name, double price) {  
 this.name = name;  
 this.price = price;  
 }  
   
 @Override  
 public String toString() {  
 return name + " - $" + price;  
 }  
}  
  
public class MultipleProductSorting {  
 public static void main(String[] args) {  
 List<Product> products = new ArrayList<>();  
 products.add(new Product("Laptop", 999.99));  
 products.add(new Product("Phone", 699.99));  
 products.add(new Product("Tablet", 399.99));  
   
 System.out.println("Sorted by Price (Ascending):");  
 products.sort(Comparator.comparingDouble(p -> p.price));  
 products.forEach(System.out::println);  
   
 System.out.println("\nSorted by Price (Descending):");  
 products.sort((p1, p2) -> Double.compare(p2.price, p1.price));  
 products.forEach(System.out::println);  
   
 System.out.println("\nSorted by Name:");  
 products.sort(Comparator.comparing(p -> p.name));  
 products.forEach(System.out::println);  
 }  
}

8. Sort Employee objects by joining date using Comparator.

Use Comparator to sort employees based on LocalDate or Date.

Code:

import java.time.\*;  
import java.util.\*;  
  
class Employee {  
 String name;  
 LocalDate joiningDate;  
   
 public Employee(String name, LocalDate joiningDate) {  
 this.name = name;  
 this.joiningDate = joiningDate;  
 }  
   
 @Override  
 public String toString() {  
 return name + " joined on " + joiningDate;  
 }  
}  
  
public class SortEmployeesByDate {  
 public static void main(String[] args) {  
 List<Employee> employees = new ArrayList<>();  
 employees.add(new Employee("John", LocalDate.of(2020, 5, 15)));  
 employees.add(new Employee("Alice", LocalDate.of(2019, 8, 20)));  
 employees.add(new Employee("Bob", LocalDate.of(2021, 3, 10)));  
   
 employees.sort(Comparator.comparing(e -> e.joiningDate));  
 employees.forEach(System.out::println);  
 }  
}

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

Code:

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 name + " (Population: " + population + ")";  
 }  
}  
  
public class SortCitiesByPopulation {  
 public static void main(String[] args) {  
 List<City> cities = new ArrayList<>();  
 cities.add(new City("New York", 8419000));  
 cities.add(new City("Los Angeles", 3971000));  
 cities.add(new City("Chicago", 2716000));  
   
 cities.sort((c1, c2) -> Integer.compare(c2.population, c1.population)); // Descending  
 cities.forEach(System.out::println);  
 }  
}

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

Code:

import java.util.\*;  
  
public class SortStringsByLength {  
 public static void main(String[] args) {  
 List<String> strings = new ArrayList<>();  
 strings.add("Apple");  
 strings.add("Banana");  
 strings.add("Cherry");  
 strings.add("Date");  
 strings.add("Elderberry");  
   
 Collections.sort(strings, new Comparator<String>() {  
 @Override  
 public int compare(String s1, String s2) {  
 return Integer.compare(s1.length(), s2.length());  
 }  
 });  
   
 strings.forEach(System.out::println);  
 }  
}

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.

Code:

import java.util.\*;  
  
class Student implements Comparable<Student> {  
 String name;  
 double marks;  
   
 public Student(String name, double marks) {  
 this.name = name;  
 this.marks = marks;  
 }  
   
 @Override  
 public int compareTo(Student s) {  
 return this.name.compareTo(s.name);  
 }  
   
 @Override  
 public String toString() {  
 return name + ": " + marks;  
 }  
}  
  
public class StudentSortingDemo {  
 public static void main(String[] args) {  
 List<Student> students = new ArrayList<>();  
 students.add(new Student("Alice", 85.5));  
 students.add(new Student("Bob", 76.0));  
 students.add(new Student("Charlie", 92.3));  
   
 System.out.println("Sorted by Name (Comparable):");  
 Collections.sort(students);  
 students.forEach(System.out::println);  
   
 System.out.println("\nSorted by Marks (Comparator):");  
 students.sort((s1, s2) -> Double.compare(s2.marks, s1.marks));  
 students.forEach(System.out::println);  
 }  
}

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

Code:

import java.util.\*;  
class Book implements Comparable<Book> {  
 int id;  
 String title;  
 String author;  
   
 public Book(int id, String title, String author) {  
 this.id = id;  
 this.title = title;  
 this.author = author;  
 }  
   
 @Override  
 public int compareTo(Book b) {  
 return this.id - b.id;  
 }  
   
 @Override  
 public String toString() {  
 return "ID: " + id + ", Title: " + title + ", Author: " + author;  
 }  
}  
  
public class BookSortingDemo {  
 public static void main(String[] args) {  
 List<Book> books = new ArrayList<>();  
 books.add(new Book(101, "Java", "John"));  
 books.add(new Book(103, "Python", "Alice"));  
 books.add(new Book(102, "C++", "Bob"));  
   
 System.out.println("Sorted by ID (Comparable):");  
 Collections.sort(books);  
 books.forEach(System.out::println);  
   
 System.out.println("\nSorted by Title then Author (Comparator):");  
 books.sort(Comparator.comparing(Book::getTitle)  
 .thenComparing(Book::getAuthor));  
 books.forEach(System.out::println);  
 }  
}

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

Code:

import java.util.\*;  
import java.time.\*;  
  
class Employee {  
 String name;  
 double salary;  
 String department;  
 LocalDate joiningDate;  
   
 public Employee(String name, double salary, String department, LocalDate joiningDate) {  
 this.name = name;  
 this.salary = salary;  
 this.department = department;  
 this.joiningDate = joiningDate;  
 }  
   
 @Override  
 public String toString() {  
 return name + " (Dept: " + department + ", Salary: " + salary + ", Joined: " + joiningDate + ")";  
 }  
}  
  
public class EmployeeSortingMenu {  
 public static void main(String[] args) {  
 List<Employee> employees = new ArrayList<>();  
 employees.add(new Employee("John", 50000, "IT", LocalDate.of(2020, 5, 15)));  
 employees.add(new Employee("Alice", 60000, "HR", LocalDate.of(2019, 8, 20)));  
 employees.add(new Employee("Bob", 55000, "Finance", LocalDate.of(2021, 3, 10)));  
   
 Scanner scanner = new Scanner(System.in);  
 while (true) {  
 System.out.println("\n1. 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 choice: ");  
   
 int choice = scanner.nextInt();  
 scanner.nextLine(); // consume newline  
   
 switch (choice) {  
 case 1:  
 employees.sort(Comparator.comparing(e -> e.name));  
 break;  
 case 2:  
 employees.sort(Comparator.comparingDouble(e -> e.salary));  
 break;  
 case 3:  
 employees.sort(Comparator.comparing(e -> e.department));  
 break;  
 case 4:  
 System.exit(0);  
 default:  
 System.out.println("Invalid choice");  
 }  
   
 System.out.println("\nSorted Employees:");  
 employees.forEach(System.out::println);  
 }  
 }  
}

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

Code:  
import java.util.\*;  
  
class Person {  
 String name;  
 int age;  
   
 public Person(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
   
 public String getName() { return name; }  
 public int getAge() { return age; }  
   
 @Override  
 public String toString() {  
 return name + " (" + age + ")";  
 }  
}  
  
public class MethodReferenceSorting {  
 public static void main(String[] args) {  
 List<Person> people = new ArrayList<>();  
 people.add(new Person("John", 25));  
 people.add(new Person("Alice", 30));  
 people.add(new Person("Bob", 22));  
   
 // Using method references  
 people.sort(Comparator.comparing(Person::getName));  
 System.out.println("Sorted by name:");  
 people.forEach(System.out::println);  
   
 people.sort(Comparator.comparingInt(Person::getAge));  
 System.out.println("\nSorted by age:");  
 people.forEach(System.out::println);  
 }  
}

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

Code:

import java.util.\*;  
  
class Person {  
 String name;  
 int age;  
   
 public Person(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
   
 @Override  
 public String toString() {  
 return name + " (" + age + ")";  
 }  
}  
  
public class TreeSetWithComparator {  
 public static void main(String[] args) {  
 // TreeSet with custom comparator to sort by age  
 TreeSet<Person> people = new TreeSet<>((p1, p2) -> p1.age - p2.age);  
   
 people.add(new Person("John", 25));  
 people.add(new Person("Alice", 30));  
 people.add(new Person("Bob", 22));  
   
 people.forEach(System.out::println);  
 }  
}

## File Operations

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.

Code:

import java.io.\*;  
  
public class CreateAndWriteFile {  
 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");  
 System.out.println("Successfully wrote to the file.");  
 } 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.

Code:

import java.io.\*;  
  
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);  
 }  
 } 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.

Code:

import java.io.\*;  
  
public class AppendToFile {  
 public static void main(String[] args) {  
 try (FileWriter writer = new FileWriter("student.txt", true)) {  
 writer.write("Frank\n");  
 System.out.println("Successfully appended to the file.");  
 } 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.

Code:  
import java.io.\*;  
public class CountWordsAndLines {  
 public static void main(String[] args) {  
 int wordCount = 0;  
 int lineCount = 0;  
   
 try (BufferedReader reader = new BufferedReader(new FileReader("notes.txt"))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 lineCount++;  
 String[] words = line.split("\\s+");  
 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.");  
 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.

Code:

import java.io.\*;  
  
public class CopyFileContents {  
 public static void main(String[] args) {  
 try (BufferedReader reader = new BufferedReader(new FileReader("source.txt"));  
 BufferedWriter writer = new BufferedWriter(new FileWriter("destination.txt"))) {  
   
 String line;  
 while ((line = reader.readLine()) != null) {  
 writer.write(line);  
 writer.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

Code:

import java.io.\*;  
  
public class CheckFileProperties {  
 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 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.

Code:

import java.io.\*;  
import java.util.Scanner;  
  
public class UserInputToFile {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
   
 try (FileWriter writer = new FileWriter("userinput.txt")) {  
 System.out.println("Enter text (type 'exit' to stop):");  
 String input;  
 while (!(input = scanner.nextLine()).equalsIgnoreCase("exit")) {  
 writer.write(input + "\n");  
 }  
 System.out.println("Input saved to file.");  
 } 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.

Code:

import java.io.\*;  
import java.util.\*;  
  
public class ReverseFileContent {  
 public static void main(String[] args) {  
 List<String> lines = new ArrayList<>();  
   
 try (BufferedReader reader = new BufferedReader(new FileReader("data.txt"))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 lines.add(line);  
 }  
   
 Collections.reverse(lines);  
   
 try (BufferedWriter writer = new BufferedWriter(new FileWriter("reversed.txt"))) {  
 for (String reversedLine : lines) {  
 writer.write(reversedLine);  
 writer.newLine();  
 }  
 System.out.println("File reversed successfully.");  
 }  
 } catch (IOException e) {  
 System.out.println("An error occurred.");  
 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.

Code:

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 SerializeStudent {  
 public static void main(String[] args) {  
 Student student = new Student(101, "Alice", 85.5);  
   
 try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("student.ser"))) {  
 oos.writeObject(student);  
 System.out.println("Student object serialized successfully.");  
 } catch (IOException e) {  
 System.out.println("An error occurred.");  
 e.printStackTrace();  
 }  
 }  
}

10. Read Serialized Object from File

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

Code:

import java.io.\*;  
  
public class DeserializeStudent {  
 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("An error occurred.");  
 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.

Code:  
import java.io.\*;  
import java.util.Scanner;  
  
public class ListFilesInDirectory {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Enter directory path: ");  
 String path = scanner.nextLine();  
   
 File directory = new File(path);  
 if (directory.exists() && directory.isDirectory()) {  
 File[] files = directory.listFiles(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 directory.");  
 }  
 } else {  
 System.out.println("Directory does not exist or is not a directory.");  
 }  
 }  
}

12. Delete a File

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

Code:

import java.io.\*;  
import java.util.Scanner;  
  
public class DeleteFile {  
 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("File deleted successfully.");  
 } else {  
 System.out.println("Failed to delete the file.");  
 }  
 } else {  
 System.out.println("File does not exist.");  
 }  
 }  
}

13. Word Search in File

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

Code:

import java.io.\*;  
import java.util.Scanner;  
  
public class WordSearchInFile {  
 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;  
 int lineNumber = 0;  
   
 try (BufferedReader reader = new BufferedReader(new FileReader("notes.txt"))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 lineNumber++;  
 if (line.contains(word)) {  
 System.out.println("Word found at line " + lineNumber + ": " + line);  
 found = true;  
 }  
 }  
   
 if (!found) {  
 System.out.println("Word not found in file.");  
 }  
 } catch (IOException e) {  
 System.out.println("An error occurred.");  
 e.printStackTrace();  
 }  
 }  
}

14. Replace Word in File

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

Code:

import java.io.\*;  
import java.util.\*;  
  
public class ReplaceWordInFile {  
 public static void main(String[] args) {  
 List<String> lines = new ArrayList<>();  
   
 try (BufferedReader reader = new BufferedReader(new FileReader("story.txt"))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 lines.add(line.replace("Java", "Python"));  
 }  
   
 try (BufferedWriter writer = new BufferedWriter(new FileWriter("updated\_story.txt"))) {  
 for (String updatedLine : lines) {  
 writer.write(updatedLine);  
 writer.newLine();  
 }  
 System.out.println("Word replaced successfully.");  
 }  
 } catch (IOException e) {  
 System.out.println("An error occurred.");  
 e.printStackTrace();  
 }  
 }  
}