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.

package assignment9;

import java.util.\*;

class Student implements Comparable<Student> {

int rollNo;

String name;

double marks;

Student(int rollNo, String name, double marks) {

this.rollNo = rollNo;

this.name = name;

this.marks = marks;

}

public int compareTo(Student other) {

return this.rollNo - other.rollNo;

}

public String toString() {

return rollNo + " " + name + " " + marks;

}

}

public class sortstudents {

public static void main(String[] args) {

List<Student> students = new ArrayList<>();

students.add(new Student(3, "Ravi", 85.5));

students.add(new Student(1, "Priya", 90.0));

students.add(new Student(2, "Amit", 78.0));

Collections.*sort*(students);

for (Student s : students) {

System.***out***.println(s);

}

}

}

Output: 1 Priya 90.0

2 Amit 78.0

3 Ravi 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().

package assignment9;

import java.util.\*;

class Product implements Comparable<Product>{

String name;

double price;

Product(String name,double price){

this.name=name;

this.price=price;

}

public int compareTo(Product p){

return Double.*compare*(this.price,p.price);

}

public String toString(){

return name+" "+price;

}

}

public class sortproducts{

public static void main(String[] args){

List<Product> products=new ArrayList<>();

products.add(new Product("Pen",10.5));

products.add(new Product("Book",55.0));

products.add(new Product("Laptop",35000.0));

Collections.*sort*(products);

for(Product p:products)

System.***out***.println(p);

}

}

Output: Pen 10.5

Book 55.0

Laptop 35000.0

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

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

package assignment9;

import java.util.\*;

class Employee implements Comparable<Employee>{

String name;

int id;

Employee(String name,int id){

this.name=name;

this.id=id;

}

public int compareTo(Employee e){

return this.name.compareTo(e.name);

}

public String toString(){

return id+" "+name;

}

}

public class sortemp{

public static void main(String[] args){

List<Employee> employees=new ArrayList<>();

employees.add(new Employee("Ravi",101));

employees.add(new Employee("Priya",102));

employees.add(new Employee("Amit",103));

Collections.*sort*(employees);

for(Employee e:employees)

System.***out***.println(e);

}

}

Output: 103 Amit

102 Priya

101 Ravi

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

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

package assignment9;

import java.util.\*;

class Book implements Comparable<Book>{

int bookId;

String title;

Book(int bookId,String title){

this.bookId=bookId;

this.title=title;

}

public int compareTo(Book b){

return b.bookId - this.bookId;

}

public String toString(){

return bookId+" "+title;

}

}

public class sortbooks{

public static void main(String[] args){

List<Book> books=new ArrayList<>();

books.add(new Book(3,"Java"));

books.add(new Book(1,"Python"));

books.add(new Book(2,"C++"));

Collections.*sort*(books);

for(Book b:books)

System.***out***.println(b);

}

}

Output: 3 Java

2 C++

1 Python

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

package assignment9;

import java.util.\*;

class Item implements Comparable<Item>{

int id;

String name;

Item(int id,String name){

this.id=id;

this.name=name;

}

public int compareTo(Item other){

return this.id - other.id;

}

public String toString(){

return id+" "+name;

}

}

public class sortcustomobj{

public static void main(String[] args){

List<Item> items=new ArrayList<>();

items.add(new Item(3,"Pen"));

items.add(new Item(1,"Book"));

items.add(new Item(2,"Laptop"));

System.***out***.println("Before sorting:");

for(Item i:items) System.***out***.println(i);

Collections.*sort*(items);

System.***out***.println("After sorting:");

for(Item i:items) System.***out***.println(i);

}

}

Output: Before sorting:

3 Pen

1 Book

2 Laptop

After sorting:

1 Book

2 Laptop

3 Pen

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

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

package assignment9;

import java.util.\*;

class Student{

String name;

double marks;

Student(String name,double marks){

this.name=name;

this.marks=marks;

}

public String toString(){

return name+" "+marks;

}

}

public class sortstudbymarks{

public static void main(String[] args){

List<Student> students=new ArrayList<>();

students.add(new Student("Ravi",85.5));

students.add(new Student("Priya",90.0));

students.add(new Student("Amit",78.0));

students.sort((s1,s2) -> Double.*compare*(s2.marks,s1.marks));

for(Student s:students)

System.***out***.println(s);

}

}

Output: Priya 90.0

Ravi 85.5

Amit 78.0

Q7. Create multiple sorting strategies for a Product class.

Implement comparators to sort by:

Price ascending

Price descending

Name alphabetically

package assignment9;

import java.util.\*;

class Product{

String name;

double price;

Product(String name,double price){

this.name=name;

this.price=price;

}

public String toString(){

return name+" "+price;

}

}

class PriceAscComparator implements Comparator<Product>{

public int compare(Product p1,Product p2){

return Double.*compare*(p1.price,p2.price);

}

}

class PriceDescComparator implements Comparator<Product>{

public int compare(Product p1,Product p2){

return Double.*compare*(p2.price,p1.price);

}

}

class NameComparator implements Comparator<Product>{

public int compare(Product p1,Product p2){

return p1.name.compareTo(p2.name);

}

}

public class prodsortdemo{

public static void main(String[] args){

List<Product> products=new ArrayList<>();

products.add(new Product("Pen",10.5));

products.add(new Product("Book",55.0));

products.add(new Product("Laptop",35000.0));

Collections.*sort*(products,new PriceAscComparator());

System.***out***.println("Price Ascending:");

products.forEach(System.***out***::println);

Collections.*sort*(products,new PriceDescComparator());

System.***out***.println("Price Descending:");

products.forEach(System.***out***::println);

Collections.*sort*(products,new NameComparator());

System.***out***.println("Name Alphabetical:");

products.forEach(System.***out***::println);

}

}

Output: Price Ascending:

Pen 10.5

Book 55.0

Laptop 35000.0

Price Descending:

Laptop 35000.0

Book 55.0

Pen 10.5

Name Alphabetical:

Book 55.0

Laptop 35000.0

Pen 10.5

Q8. Sort Employee objects by joining date using Comparator.

Use Comparator to sort employees based on LocalDate or Date.

package assignment9;

import java.util.\*;

import java.time.\*;

class Employee5{

String name;

LocalDate joiningDate;

Employee5(String name,LocalDate joiningDate){

this.name=name;

this.joiningDate=joiningDate;

}

public String toString(){

return name+" "+joiningDate;

}

}

public class sortempbydate{

public static void main(String[] args){

List<Employee5> employees=new ArrayList<>();

employees.add(new Employee5("Ravi",LocalDate.*of*(2022,5,10)));

employees.add(new Employee5("Priya",LocalDate.*of*(2020,3,15)));

employees.add(new Employee5("Amit",LocalDate.*of*(2021,8,25)));

employees.sort(Comparator.*comparing*(e -> e.joiningDate));

for(Employee5 e:employees)

System.***out***.println(e);

}

}

Output: Priya 2020-03-15

Amit 2021-08-25

Ravi 2022-05-10

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

package assignment9;

import java.util.\*;

class City{

String name;

int population;

City(String name,int population){

this.name=name;

this.population=population;

}

public String toString(){

return name+" "+population;

}

}

public class sortcitybypopulation{

public static void main(String[] args){

List<City> cities=new ArrayList<>();

cities.add(new City("Mumbai",20000000));

cities.add(new City("Delhi",18000000));

cities.add(new City("Bangalore",12000000));

cities.sort(Comparator.*comparingInt*(c -> c.population));

for(City c:cities)

System.***out***.println(c);

}

}

Output: Bangalore 12000000

Delhi 18000000

Mumbai 20000000

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

package assignment9;

import java.util.\*;

public class sortstr\_bylen{

public static void main(String[] args){

List<String> strings = new ArrayList<>(Arrays.*asList*("Apple", "Banana", "Kiwi", "Orange"));

Collections.*sort*(strings, new Comparator<String>(){

public int compare(String s1, String s2){

return s1.length() - s2.length();

}

});

for(String s : strings)

System.***out***.println(s);

}

}

Output: Kiwi

Apple

Banana

Orange

Q11. Create a program where:

Student implements Comparable to sort by name

Use Comparator to sort by marks

Demonstrate both sorting techniques in the same program.

package assignment9;

import java.util.\*;

class Student implements Comparable<Student>{

String name;

double marks;

Student(String name,double marks){

this.name=name;

this.marks=marks;

}

public int compareTo(Student s){

return this.name.compareTo(s.name);

}

public String toString(){

return name+" "+marks;

}

}

public class studsort{

public static void main(String[] args){

List<Student> students=new ArrayList<>();

students.add(new Student("Ravi",85.5));

students.add(new Student("Priya",90.0));

students.add(new Student("Amit",78.0));

System.***out***.println("Sort by name:");

Collections.*sort*(students);

for(Student s:students)

System.***out***.println(s);

System.***out***.println("Sort by marks:");

students.sort(Comparator.*comparingDouble*(s -> s.marks));

for(Student s:students)

System.***out***.println(s);

}

}

Output: Sort by name:

Amit 78.0

Priya 90.0

Ravi 85.5

Sort by marks:

Amit 78.0

Ravi 85.5

Priya 90.0

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

package assignment9;

import java.util.\*;

class Book implements Comparable<Book>{

int id;

String title, author;

Book(int id,String title,String author){

this.id=id;

this.title=title;

this.author=author;

}

public int compareTo(Book b){

return this.id - b.id;

}

public String toString(){

return id+" "+title+" "+author;

}

}

class TitleAuthorComparator implements Comparator<Book>{

public int compare(Book b1, Book b2){

int res = b1.title.compareTo(b2.title);

if(res == 0)

return b1.author.compareTo(b2.author);

else

return res;

}

}

public class booksortd{

public static void main(String[] args){

List<Book> books = new ArrayList<>();

books.add(new Book(3,"Java","Alice"));

books.add(new Book(1,"Python","Bob"));

books.add(new Book(2,"Java","Charlie"));

System.***out***.println("Sort by ID (Comparable):");

Collections.*sort*(books);

for(Book b : books)

System.***out***.println(b);

System.***out***.println("Sort by Title then Author (Comparator):");

Collections.*sort*(books, new TitleAuthorComparator());

for(Book b : books)

System.***out***.println(b);

}

}

Output: Sort by ID (Comparable):

1 Python Bob

2 Java Charlie

3 Java Alice

Sort by Title then Author (Comparator):

3 Java Alice

2 Java Charlie

1 Python Bob

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

package assignment9;

import java.util.\*;

class Emp1{

String name, department;

double salary;

Emp1(String name, String department, double salary){

this.name = name;

this.department = department;

this.salary = salary;

}

public String toString(){

return name + " " + department + " " + salary;

}

}

public class empsort\_menu{

public static void main(String[] args){

List<Emp1> employees = new ArrayList<>();

employees.add(new Emp1("Ravi", "IT", 50000));

employees.add(new Emp1("Priya", "HR", 60000));

employees.add(new Emp1("Amit", "Finance", 55000));

Scanner sc = new Scanner(System.***in***);

while(true){

System.***out***.println("\nSort by: 1-Name 2-Salary 3-Department 4-Exit");

int choice = sc.nextInt();

if(choice == 4) break;

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;

default:

System.***out***.println("Invalid choice");

continue;

}

for(Emp1 e : employees)

System.***out***.println(e);

}

sc.close();

}

}

Output: Sort by: 1-Name 2-Salary 3-Department 4-Exit

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

package assignment9;

import java.util.\*;

class Emp{

String name;

double salary;

Emp(String name, double salary){

this.name = name;

this.salary = salary;

}

public String getName(){

return name;

}

public double getSalary(){

return salary;

}

public String toString(){

return name + " " + salary;

}

}

public class sortwith\_method{

public static void main(String[] args){

List<Emp> list = new ArrayList<>();

list.add(new Emp("Ravi", 50000));

list.add(new Emp("Priya", 60000));

list.add(new Emp("Amit", 55000));

System.***out***.println("Sort by name:");

list.sort(Comparator.*comparing*(Emp::getName));

list.forEach(System.***out***::println);

System.***out***.println("Sort by salary:");

list.sort(Comparator.*comparingDouble*(Emp::getSalary));

list.forEach(System.***out***::println);

}

}

Output: Sort by name:

Amit 55000.0

Priya 60000.0

Ravi 50000.0

Sort by salary:

Ravi 50000.0

Amit 55000.0

Priya 60000.0

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

package assignment9;

import java.util.\*;

class Person7{

String name;

int age;

Person7(String name, int age){

this.name = name;

this.age = age;

}

public String toString(){

return name + " " + age;

}

}

public class treecustom\_comp{

public static void main(String[] args){

Comparator<Person7> ageComparator = new Comparator<Person7>(){

public int compare(Person7 p1, Person7 p2){

return p1.age - p2.age;

}

};

TreeSet<Person7> persons = new TreeSet<>(ageComparator);

persons.add(new Person7("Ravi", 25));

persons.add(new Person7("Priya", 30));

persons.add(new Person7("Amit", 22));

for(Person7 p : persons)

System.***out***.println(p);

}

}

Output: Amit 22

Ravi 25

Priya 30

**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.

package assignment9;

import java.io.FileWriter;

import java.io.IOException;

public class writestudfile {

public static void main(String[] args) {

try {

FileWriter writer = new FileWriter("student.txt");

writer.write("John\n");

writer.write("Alice\n");

writer.write("Bob\n");

writer.write("Emma\n");

writer.write("David\n");

writer.close();

System.***out***.println("student.txt created and data written successfully.");

} catch (IOException e) {

System.***out***.println("An error occurred.");

e.printStackTrace();

}

}

}

Output: student.txt created and data written successfully.

**Q2. Read from a File**

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

package assignment9;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class readstudfile {

public static void main(String[] args) {

try {

BufferedReader reader = new BufferedReader(new FileReader("student.txt"));

String line;

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

System.***out***.println(line);

}

reader.close();

} catch (IOException e) {

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

e.printStackTrace();

}

}

}

Output: John

Alice

Bob

Emma

David

**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.

package assignment9;

import java.io.FileWriter;

import java.io.IOException;

public class appendstudfile {

public static void main(String[] args) {

try {

FileWriter writer = new FileWriter("student.txt", true);

writer.write("Sophia\n");

writer.close();

System.***out***.println("New student name appended successfully.");

} catch (IOException e) {

System.***out***.println("An error occurred while appending.");

e.printStackTrace();

}

}

}

Output: New student name appended successfully.

**Q4. Count Words and Lines**

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

package assignment9;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class countwordlines {

public static void main(String[] args) {

int lineCount = 0;

int wordCount = 0;

try {

BufferedReader reader = new BufferedReader(new FileReader("Student.txt"));

String line;

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

lineCount++;

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

if (!line.trim().isEmpty()) {

wordCount += words.length;

}

}

reader.close();

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: Number of lines: 6

Number of words: 6

**Q5. Copy Contents from One File to Another**

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

package assignment9;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

public class copyfile {

public static void main(String[] args) {

try {

BufferedReader reader = new BufferedReader(new FileReader("Student.txt"));

BufferedWriter writer = new BufferedWriter(new FileWriter("employee.txt"));

String line;

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

writer.write(line);

writer.newLine();

}

reader.close();

writer.close();

System.***out***.println("File copied successfully.");

} catch (IOException e) {

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

e.printStackTrace();

}

}

}

Output: File copied successfully.

**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

package assignment9;

import java.io.File;

public class fileproperties {

public static void main(String[] args) {

File file = new File("Student.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 in bytes: " + file.length());

} else {

System.***out***.println("The file does not exist.");

}

}

}

Output: File exists.

Absolute path: C:\Users\raviy\eclipse-workspace\Java\_practice\Student.txt

File name: Student.txt

Writable: true

Readable: true

File size in bytes: 33

**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.

package assignment9;

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***);

try {

FileWriter writer = new FileWriter("userinput.txt");

System.***out***.println("Enter text to write into userinput.txt (type 'exit' to stop):");

while (true) {

String input = sc.nextLine();

if (input.equalsIgnoreCase("exit")) {

break;

}

writer.write(input + "\n");

}

writer.close();

System.***out***.println("Data written to userinput.txt successfully.");

} catch (IOException e) {

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

e.printStackTrace();

}

sc.close();

}

}

Output: Enter text to write into userinput.txt (type 'exit' to stop):

hello welcome to java

hi

exit

Data written to userinput.txt successfully.

**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.

package assignment9;

import java.io.\*;

import java.util.\*;

public class reversefilecontent {

public static void main(String[] args) {

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

try {

BufferedReader r = new BufferedReader(new FileReader("userinput.txt"));

String line;

while ((line = r.readLine()) != null) lines.add(line);

r.close();

Collections.*reverse*(lines);

BufferedWriter w = new BufferedWriter(new FileWriter("reversed.txt"));

for (String l : lines) {

w.write(l);

w.newLine();

}

w.close();

System.***out***.println("File reversed successfully.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

Output: File reversed successfully.

**Q9. Store Objects in a File using Serialization**

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

package assignment9;

import java.io.\*;

class Studentj implements Serializable {

int id;

String name;

double marks;

Studentj(int id, String name, double marks) {

this.id = id;

this.name = name;

this.marks = marks;

}

}

public class serializestudent {

public static void main(String[] args) {

Studentj s1 = new Studentj(101, "John", 85.5);

try {

FileOutputStream fileOut = new FileOutputStream("employee.ser");

ObjectOutputStream out = new ObjectOutputStream(fileOut);

out.writeObject(s1);

out.close();

fileOut.close();

System.***out***.println("Student object serialized and saved in student.ser");

} catch (IOException e) {

e.printStackTrace();

}

}

}

Output: Student object serialized and saved in student.ser

**Q10. Read Serialized Object from File**

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

package assignment9;

import java.io.\*;

class Studenta implements Serializable {

int id;

String name;

double marks;

Studenta(int id, String name, double marks) {

this.id = id;

this.name = name;

this.marks = marks;

}

}

public class deserializestudent {

public static void main(String[] args) {

try {

FileInputStream fileIn = new FileInputStream("employeee.ser");

ObjectInputStream in = new ObjectInputStream(fileIn);

Studenta s = (Studenta) in.readObject();

in.close();

fileIn.close();

System.***out***.println("ID: " + s.id);

System.***out***.println("Name: " + s.name);

System.***out***.println("Marks: " + s.marks);

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

}

}

**Q11. Print All Files in a Directory**

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

package assignment9;

import java.io.File;

import java.util.Scanner;

public class listfiles {

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

System.***out***.print("Enter folder path: ");

String path = sc.nextLine();

File folder = new File(path);

if (folder.exists() && folder.isDirectory()) {

File[] files = folder.listFiles();

System.***out***.println("Files in the folder:");

for (File f : files) {

if (f.isFile()) {

System.***out***.println(f.getName());

}

}

} else {

System.***out***.println("Invalid folder path.");

}

sc.close();

}

}

Output: Enter folder path: C:\Users\raviy\OneDrive\Desktop\Wipro\ppts\Day9

Files in the folder:

29\_Java\_File\_Handling.pptx

30\_Java\_Comparator\_Comparable.pptx

Day9\_assignment.docx

**Q12. Delete a File**

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

package assignment9;

import java.io.File;

import java.util.Scanner;

public class deletefile {

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

System.***out***.print("Enter file name to delete: ");

String fileName = sc.nextLine();

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.");

}

sc.close();

}

}

Output: Enter file name to delete: sample.txt

File deleted successfully.

**Q13. Word Search in a File**

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

package assignment9;

import java.io.\*;

import java.util.Scanner;

public class wordsearchinfile {

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

System.***out***.print("Enter word to search: ");

String word = sc.nextLine();

boolean found = false;

try {

BufferedReader reader = new BufferedReader(new FileReader("Student.txt"));

String line;

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

if (line.contains(word)) {

found = true;

break;

}

}

reader.close();

if (found) {

System.***out***.println("Word found in file.");

} else {

System.***out***.println("Word not found in file.");

}

} catch (IOException e) {

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

e.printStackTrace();

}

sc.close();

}

}

Output: Enter word to search: id

Word found in file.

**Q14. Replace a Word in a File**

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

package assignment9;

import java.io.\*;

public class replacewordinfile {

public static void main(String[] args) {

try {

BufferedReader reader = new BufferedReader(new FileReader("Student.txt"));

BufferedWriter writer = new BufferedWriter(new FileWriter("sample.txt"));

String line;

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

line = line.replace("Java", "Python");

writer.write(line);

writer.newLine();

}

reader.close();

writer.close();

System.***out***.println("Replacement done. Updated file: sample.txt");

} catch (IOException e) {

e.printStackTrace();

}

}

}

Output: Replacement done. Updated file: sample.txt