**Day 19:**

**Task 1: Generics and Type Safety**

**Create a generic Pair class that holds two objects of different types, and write a method to return a reversed version of the pair.**

public class Pair<T, U> {

private T first;

private U second;

public Pair(T first, U second) {

this.first = first;

this.second = second;

}

public T getFirst() {

return first;

}

public U getSecond() {

return second;

}

public Pair<U, T> reverse() {

return new Pair<>(second, first);

}

@Override

public String toString() {

return "Pair{" +

"first=" + first +

", second=" + second +

'}';

} public static void main(String[] args) {

Pair<String, Integer> pair = new Pair<>("Hello", 123);

System.out.println("Original pair: " + pair);

Pair<Integer, String> reversedPair = pair.reverse();

System.out.println("Reversed pair: " + reversedPair);

}

}

**Task 2: Generic Classes and Methods**

**Implement a generic method that swaps the positions of two elements in an array, regardless of their type, and demonstrate its usage with different object types.**

import java.util.Arrays;

public class GenericSwap {

public static <T> void swap(T[] array, int index1, int index2) {

if (array == null || index1 < 0 || index2 < 0 || index1 >= array.length || index2 >= array.length) {

throw new IllegalArgumentException("Invalid index or array is null");

}

T temp = array[index1];

array[index1] = array[index2];

array[index2] = temp;

}

public static void main(String[] args) {

Integer[] intArray = {1, 2, 3, 4, 5};

System.out.println("Original Integer array: " + Arrays.toString(intArray));

swap(intArray, 1, 3);

System.out.println("Swapped Integer array: " + Arrays.toString(intArray));

String[] strArray = {"apple", "banana", "cherry", "date"};

System.out.println("Original String array: " + Arrays.toString(strArray));

swap(strArray, 0, 2);

System.out.println("Swapped String array: " + Arrays.toString(strArray));

Double[] doubleArray = {1.1, 2.2, 3.3, 4.4};

System.out.println("Original Double array: " + Arrays.toString(doubleArray));

swap(doubleArray, 2, 3);

System.out.println("Swapped Double array: " + Arrays.toString(doubleArray));

}

}

**Task 3: Reflection API**

**Use reflection to inspect a class's methods, fields, and constructors, and modify the access level of a private field, setting its value during runtime**

class SampleClass {

private String privateField = "Initial Value";

public int publicField;

public SampleClass() {

}

public SampleClass(int publicField) {

this.publicField = publicField;

}

public void publicMethod() {

System.out.println("Public method called");

}

private void privateMethod() {

System.out.println("Private method called");

}

}

public class ReflectionExample {

public static void main(String[] args) {

try {

Class<SampleClass> sampleClassClass = SampleClass.class;

System.out.println("Methods:");

Method[] methods = sampleClassClass.getDeclaredMethods();

for (Method method : methods) {

System.out.println(" - " + method);

}

System.out.println("\nFields:");

Field[] fields = sampleClassClass.getDeclaredFields();

for (Field field : fields) {

System.out.println(" - " + field);

}

System.out.println("\nConstructors:");

Constructor<?>[] constructors = sampleClassClass.getConstructors();

for (Constructor<?> constructor : constructors) {

System.out.println(" - " + constructor);

}

SampleClass sampleInstance = new SampleClass();

Field privateField = sampleClassClass.getDeclaredField("privateField");

privateField.setAccessible(true);

privateField.set(sampleInstance, "New Value");

String fieldValue = (String) privateField.get(sampleInstance);

System.out.println("\nModified privateField value: " + fieldValue);

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Task 4: Lambda Expressions**

**Implement a Comparator for a Person class using a lambda expression, and sort a list of Person objects by their age..**

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;

}

@Override

public String toString() {

return "Person{name='" + name + "', age=" + age + "}";

}

}

public class LambdaComparatorExample {

public static void main(String[] args) {

List<Person> people = new ArrayList<>();

people.add(new Person("Alice", 30));

people.add(new Person("Bob", 25));

people.add(new Person("Charlie", 35));

Print the list before sorting

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

people.forEach(System.out::println);

people.sort(Comparator.comparingInt(Person::getAge));

System.out.println("\nAfter sorting by age:");

people.forEach(System.out::println);

}

}

**Task 5: Functional Interfaces**

**Create a method that accepts functions as parameters using Predicate, Function, Consumer, and Supplier interfaces to operate on a Person object.**

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 void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

@Override

public String toString() {

return "Person{name='" + name + "', age=" + age + "}";

}

}

public class FunctionalInterfacesExample {

public static boolean checkPerson(Person person, Predicate<Person> predicate) {

return predicate.test(person);

}

public static String transformPerson(Person person, Function<Person, String> function) {

return function.apply(person);

}

public static void processPerson(Person person, Consumer<Person> consumer) {

consumer.accept(person);

}

public static Person createPerson(Supplier<Person> supplier) {

return supplier.get();

}

public static void main(String[] args) {

Person person = new Person("Alice", 30);

boolean isAdult = checkPerson(person, p -> p.getAge() >= 18);

System.out.println("Is adult: " + isAdult);

String name = transformPerson(person, Person::getName);

System.out.println("Person's name: " + name);

processPerson(person, p -> System.out.println("Processing person: " + p));

Person newPerson = createPerson(() -> new Person("Bob", 25));

System.out.println("New Person: " + newPerson);

}

}

**Day 20:**

**Task 1: Java IO Basics**

**Write a program that reads a text file and counts the frequency of each word using FileReader and FileWriter.**

public class WordFrequencyCounter {

public static void main(String[] args) {

String inputFilePath = "input.txt";

String outputFilePath = "output.txt";

Map<String, Integer> wordCountMap = new HashMap<>();

try (BufferedReader reader = new BufferedReader(new FileReader(inputFilePath))) {

String line;

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

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

for (String word : words) {

word = word.toLowerCase().replaceAll("[^a-zA-Z]", "");

if (!word.isEmpty()) {

wordCountMap.put(word, wordCountMap.getOrDefault(word, 0) + 1);

}

}

}

} catch (IOException e) {

e.printStackTrace();

}

try (BufferedWriter writer = new BufferedWriter(new FileWriter(outputFilePath))) {

for (Map.Entry<String, Integer> entry : wordCountMap.entrySet()) {

writer.write(entry.getKey() + ": " + entry.getValue());

writer.newLine();

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Task 2: Serialization and Deserialization**

**Serialize a custom object to a file and then deserialize it back to recover the object state.**

import java.io.\*;

class Person implements Serializable {

private static final long serialVersionUID = 1L;

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public String getName() {

return name;

} public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

@Override

public String toString() {

return "Person{name='" + name + "', age=" + age + "}";

}

}

public class SerializationExample {

public static void main(String[] args) {

String filePath = "person.ser";

Person person = new Person("Alice", 30);

serializePerson(person, filePath);

Person deserializedPerson = deserializePerson(filePath);

System.out.println("Deserialized Person: " + deserializedPerson);

}

public static void serializePerson(Person person, String filePath) {

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(filePath))) {

oos.writeObject(person);

System.out.println("Person object has been serialized to " + filePath);

} catch (IOException e) {

e.printStackTrace();

}

}

public static Person deserializePerson(String filePath) {

Person person = null;

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(filePath))) {

person = (Person) ois.readObject();

System.out.println("Person object has been deserialized from " + filePath);

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

return person;

}

}

**Task 3: New IO (NIO)**

**Use NIO Channels and Buffers to read content from a file and write to another file.**

import java.io.IOException;

import java.nio.ByteBuffer;

import java.nio.channels.FileChannel;

import java.nio.file.Path;

import java.nio.file.StandardOpenOption;

public class NIOFileCopy {

public static void main(String[] args) {

Path inputFilePath = Path.of("input.txt");

Path outputFilePath = Path.of("output.txt");

try (FileChannel inputFileChannel = FileChannel.open(inputFilePath, StandardOpenOption.READ);

FileChannel outputFileChannel = FileChannel.open(outputFilePath, StandardOpenOption.WRITE, StandardOpenOption.CREATE)) {

ByteBuffer buffer = ByteBuffer.allocate(1024);

while (inputFileChannel.read(buffer) > 0) {

buffer.flip();

outputFileChannel.write(buffer);

buffer.clear();

} System.out.println("File copy completed successfully.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

Task 4: Java Networking

Write a simple HTTP client that connects to a URL, sends a request, and displays the response headers and body.

public class SimpleHttpClient {

public static void main(String[] args) {

String urlString = "http://www.example.com";

try {

URL url = new URL(urlString);

HttpURLConnection connection = (HttpURLConnection) url.openConnection();

connection.setRequestMethod("GET");

int responseCode = connection.getResponseCode();

System.out.println("Response Code: " + responseCode);

System.out.println("Response Headers:");

Map<String, List<String>> headers = connection.getHeaderFields();

for (Map.Entry<String, List<String>> header : headers.entrySet()) {

System.out.println(header.getKey() + ": " + header.getValue());

}

System.out.println("Response Body:");

try (BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()))) {

String inputLine;

StringBuilder response = new StringBuilder();

while ((inputLine = in.readLine()) != null) {

response.append(inputLine).append("\n");

}

System.out.println(response.toString());

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Task 5: Java Networking and Serialization**

**Develop a basic TCP client and server application where the client sends a serialized object with 2 numbers and operation to be performed on them to the server, and the server computes the result and sends it back to the client. for eg, we could send 2, 2, "+" which would mean 2 + 2**

public class TCPServer {

public static void main(String[] args) {

int port = 12345;

try {

ServerSocket serverSocket = new ServerSocket(port);

System.out.println("Server listening on port " + port);

while (true) {

Socket clientSocket = serverSocket.accept();

System.out.println("Client connected: " + clientSocket.getInetAddress().getHostAddress());

ObjectInputStream inputStream = new ObjectInputStream(clientSocket.getInputStream());

ObjectOutputStream outputStream = new ObjectOutputStream(clientSocket.getOutputStream());

try {

CalculationRequest request = (CalculationRequest) inputStream.readObject();

int result;

switch (request.getOperation()) {

case "+":

result = request.getNum1() + request.getNum2();

break;

case "-":

result = request.getNum1() - request.getNum2();

break;

case "\*":

result = request.getNum1() \* request.getNum2();

break;

case "/":

result = request.getNum1() / request.getNum2();

break;

default:

throw new IllegalArgumentException("Invalid operation");

}

outputStream.writeInt(result);

outputStream.flush();

System.out.println("Result sent to client: " + result);

} catch (ClassNotFoundException e) {

e.printStackTrace();

} finally {

inputStream.close();

outputStream.close();

clientSocket.close();

}

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Task 6: Java 8 Date and Time API**

**Write a program that calculates the number of days between two dates input by the user.**

public class DaysBetweenDates {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the first date (yyyy-MM-dd): ");

String firstDateString = scanner.nextLine();

System.out.print("Enter the second date (yyyy-MM-dd): ");

String secondDateString = scanner.nextLine();

scanner.close();

LocalDate firstDate = LocalDate.parse(firstDateString);

LocalDate secondDate = LocalDate.parse(secondDateString);

long daysBetween = ChronoUnit.DAYS.between(firstDate, secondDate); System.out.println("Number of days between " + firstDate + " and " + secondDate + " is: " + daysBetween);

}

}

**Task 7: Timezone**

**Create a timezone converter that takes a time in one timezone and converts it to another timezone.**

public class TimezoneConverter {

public static void main(String[] args) {

ZoneId sourceZone = ZoneId.of("America/New\_York"); // Change to source timezone

ZoneId targetZone = ZoneId.of("Asia/Tokyo"); // Change to target timezone

LocalDateTime sourceTime = LocalDateTime.of(2024, Month.JUNE, 11, 9, 0);

ZonedDateTime sourceZonedDateTime = sourceTime.atZone(sourceZone);

ZonedDateTime targetZonedDateTime = sourceZonedDateTime.withZoneSameInstant(targetZone);

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

String formattedTargetTime = formatter.format(targetZonedDateTime);

System.out.println("Converted time in " + targetZone + ": " + formattedTargetTime);

}

}

**Day 21:**

**Task 1: Establishing Database Connections**

**Write a Java program that connects to a SQLite database and prints out the connection object to confirm successful connection.**

public class SQLiteConnectionExample {

public static void main(String[] args) {

String url = "jdbc:sqlite:/path/to/your/database.db";

try {

Class.forName("org.sqlite.JDBC");

Connection connection = DriverManager.getConnection(url);

System.out.println("Connected to the SQLite database.");

connection.close();

} catch (ClassNotFoundException e) {

System.err.println("SQLite JDBC driver not found.");

e.printStackTrace();

} catch (SQLException e) {

System.err.println("Failed to connect to the SQLite database.");

e.printStackTrace();

}

}

}

**Task 2: SQL Queries using JDBC**

**Create a table 'User' with a following schema 'User ID' and 'Password' stored as hash format (note you have research on how to generate hash from a string), accept "User ID" and "Password" as input and check in the table if they match to confirm whether user access is allowed or not.**

public class UserAuthentication {

static final String url = "jdbc:sqlite:sample.db";

public static void main(String[] args) {

try (Connection connection = DriverManager.getConnection(url)) {

createTable(connection);

String userId = "test\_user";

String password = "test\_password";

String hashedPassword = generateHash(password);

boolean authenticated = authenticateUser(connection, userId, hashedPassword);

if (authenticated) {

System.out.println("User access allowed.");

} else {

System.out.println("User access denied.");

}

} catch (SQLException e) {

e.printStackTrace();

}

}

private static void createTable(Connection connection) throws SQLException {

String createTableSQL = "CREATE TABLE IF NOT EXISTS User (userId TEXT PRIMARY KEY, passwordHash TEXT)";

try (Statement statement = connection.createStatement()) {

statement.execute(createTableSQL);

}

}

private static String generateHash(String input) {

try {

MessageDigest md = MessageDigest.getInstance("SHA-256");

byte[] hashBytes = md.digest(input.getBytes());

StringBuilder sb = new StringBuilder();

for (byte b : hashBytes) {

sb.append(String.format("%02x", b));

}

return sb.toString();

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

return null;

}

}

private static boolean authenticateUser(Connection connection, String userId, String hashedPassword) throws SQLException {

String selectSQL = "SELECT \* FROM User WHERE userId = ? AND passwordHash = ?";

try (PreparedStatement preparedStatement = connection.prepareStatement(selectSQL)) {

preparedStatement.setString(1, userId);

preparedStatement.setString(2, hashedPassword);

ResultSet resultSet = preparedStatement.executeQuery();

return resultSet.next();

}

}

}

**Task 3: PreparedStatement**

**Modify the SELECT query program to use PreparedStatement to parameterize the query and prevent SQL injection.**

public class UserAuthenticationWithPreparedStatement {

static final String url = "jdbc:sqlite:sample.db";

public static void main(String[] args) {

try (Connection connection = DriverManager.getConnection(url)) {

createTable(connection);

String userId = "test\_user";

String password = "test\_password";

String hashedPassword = generateHash(password);

boolean authenticated = authenticateUser(connection, userId, hashedPassword);

if (authenticated) {

System.out.println("User access allowed.");

} else {

System.out.println("User access denied.");

}

} catch (SQLException e) {

e.printStackTrace();

}

}

private static void createTable(Connection connection) throws SQLException {

String createTableSQL = "CREATE TABLE IF NOT EXISTS User (userId TEXT PRIMARY KEY, passwordHash TEXT)";

try (Statement statement = connection.createStatement()) {

statement.execute(createTableSQL);

}

}

private static String generateHash(String input) {

try {

MessageDigest md = MessageDigest.getInstance("SHA-256");

byte[] hashBytes = md.digest(input.getBytes());

StringBuilder sb = new StringBuilder();

for (byte b : hashBytes) {

sb.append(String.format("%02x", b));

}

return sb.toString();

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

return null;

}

}

private static boolean authenticateUser(Connection connection, String userId, String hashedPassword) throws SQLException {

String selectSQL = "SELECT \* FROM User WHERE userId = ? AND passwordHash = ?";

try (PreparedStatement preparedStatement = connection.prepareStatement(selectSQL)) {

preparedStatement.setString(1, userId);

preparedStatement.setString(2, hashedPassword);

ResultSet resultSet = preparedStatement.executeQuery();

return resultSet.next();

}

}

}

**Day 22:**

**Task 1: Write a set of JUnit tests for a given class with simple mathematical operations (add, subtract, multiply, divide) using the basic @Test annotation**

public class MathOperations {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

public int multiply(int a, int b) {

return a \* b;

}

public double divide(int a, int b) {

if (b == 0) {

throw new IllegalArgumentException("Division by zero is not allowed.");

}

return (double) a / b;

}

}

public class MathOperationsTest {

@Test

public void testAdd() {

MathOperations math = new MathOperations();

assertEquals(5, math.add(2, 3));

assertEquals(-1, math.add(-2, 1));

}

@Test

public void testSubtract() {

MathOperations math = new MathOperations();

assertEquals(3, math.subtract(5, 2));

assertEquals(7, math.subtract(10, 3));

}

@Test

public void testMultiply() {

MathOperations math = new MathOperations();

assertEquals(6, math.multiply(2, 3));

assertEquals(-15, math.multiply(5, -3));

}@Test

public void testDivide() {

MathOperations math = new MathOperations();

assertEquals(2.5, math.divide(5, 2), 0.001);

assertEquals(3, math.divide(10, 3), 0.001);

}

@Test(expected = IllegalArgumentException.class)

public void testDivideByZero() {

MathOperations math = new MathOperations();

math.divide(5, 0);

}

}

**Task 2: Extend the above JUnit tests to use @Before, @After, @BeforeClass, and @AfterClass annotations to manage test setup and teardown.**

public class MathOperationsTest {

private static MathOperations math;

@BeforeClass

public static void setUpClass() {

System.out.println("Setting up MathOperationsTest class...");

math = new MathOperations();

}

@AfterClass

public static void tearDownClass() {

System.out.println("Tearing down MathOperationsTest class...");

math = null;

}

@Before

public void setUp() {

System.out.println("Setting up test...");

}

@After

public void tearDown() {

System.out.println("Tearing down test...");

}

@Test

public void testAdd() {

assertEquals(5, math.add(2, 3));

assertEquals(-1, math.add(-2, 1));

}

@Test

public void testSubtract() {

assertEquals(3, math.subtract(5, 2));

assertEquals(7, math.subtract(10, 3));

}

@Test

public void testMultiply() {

assertEquals(6, math.multiply(2, 3));

assertEquals(-15, math.multiply(5, -3));

}

@Test

public void testDivide() {

assertEquals(2.5, math.divide(5, 2), 0.001);

assertEquals(3, math.divide(10, 3), 0.001);

}

@Test(expected = IllegalArgumentException.class)

public void testDivideByZero() {

math.divide(5, 0);

}

}

**Task 3: Create test cases with assertEquals, assertTrue, and assertFalse to validate the correctness of a custom String utility class.**

public class StringUtils {

public static boolean isNullOrEmpty(String str) {

return str == null || str.isEmpty();

}

public static boolean isPalindrome(String str) {

if (isNullOrEmpty(str)) {

return false;

}

str = str.toLowerCase();

int left = 0;

int right = str.length() - 1;

while (left < right) {

if (str.charAt(left++) != str.charAt(right--)) {

return false;

}

}

return true;

}

}

public class StringUtilsTest {

@Test

public void testIsNullOrEmpty() {

assertTrue(StringUtils.isNullOrEmpty(null));

assertTrue(StringUtils.isNullOrEmpty(""));

assertFalse(StringUtils.isNullOrEmpty("Hello"));

}

@Test

public void testIsPalindrome() {

assertTrue(StringUtils.isPalindrome("radar"));

assertTrue(StringUtils.isPalindrome("level"));

assertFalse(StringUtils.isPalindrome("hello"));

assertFalse(StringUtils.isPalindrome(null));

assertFalse(StringUtils.isPalindrome(""));

}

}

**Task 4: Research and present a comparison of different garbage collection algorithms (Serial, Parallel, CMS, G1, ZGC) in Java.**

public class GarbageCollectionDemo {

public static void main(String[] args) {

for (int i = 0; i < 100000; i++) {

new GarbageObject();

}

System.gc();

}

static class GarbageObject {

private byte[] data = new byte[1024];

}

}