Day 12 - 27th June 2025

**Task 1:**

**What do you understand about data structures?**

A data structure is a specialized format for organizing, storing, and managing data in a computer so that it can be accessed and modified efficiently. It's like a container that stores data in a specific way to serve particular purposes.

Data structures help in:

* Managing large amounts of data
* Making data manipulation more efficient
* Organizing data in a way that suits specific operations
* Improving program performance and efficiency

**Task 2:**

**What are the types of data structures**

Data structures can be broadly classified into two categories:

Primitive Data Structures:

1. Integer
2. Float
3. Character
4. Boolean
5. Double
6. String

Non-Primitive Data Structures:

A. Linear Data Structures:

1. Arrays
2. Linked Lists
3. Stacks
4. Queues

B. Non-Linear Data Structures:

1. Trees
2. Graphs
3. Hash Tables

C. Dynamic Data Structures:

1. Dynamic Arrays
2. Dynamic Lists

Each type has specific characteristics and use cases:

* Arrays: Fixed-size sequential collection
* Linked Lists: Sequential elements connected via pointers
* Stacks: LIFO (Last In First Out) structure
* Queues: FIFO (First In First Out) structure
* Trees: Hierarchical structure with parent-child relationships
* Graphs: Network structure with nodes and edges
* Hash Tables: Key-value pair structure for fast data retrieval

The choice of data structure depends on:

Type of data to be stored is decided based on:

1. Cost of operations
2. Memory usage
3. Ease of implementation

**Task 3:**

**What all operations can we do in Data structures?**

In Data Structures, we can perform several fundamental operations. Here are the main operations:

Insertion Operations:

* Adding elements to the data structure
* Can be done at beginning, end, or specific position
* Example: insert(element), push(), append()

Deletion Operations:

* Removing elements from the data structure
* Can delete from beginning, end, or specific position
* Example: delete(element), pop(), remove()

Traversal Operations:

* Visiting each element in the data structure
* Processing elements one by one
* Example: iterate through array, traverse linked list

Searching Operations:

* Finding specific elements in the data structure
* Linear search, binary search, etc.
* Example: search(key), find(element)

Sorting Operations:

* Arranging elements in a specific order
* Ascending or descending order
* Example: bubble sort, quick sort, merge sort

Updating Operations:

* Modifying existing elements
* Changing values at specific positions
* Example: update(position, newValue)

Merging Operations:

* Combining two data structures
* Creating a new structure from existing ones
* Example: merge two sorted arrays

Splitting Operations:

* Dividing data structure into smaller parts
* Creating sub-structures
* Example: splitting an array into two

Access Operations:

* Retrieving elements from specific positions
* Reading values without modification
* Example: get(index), peek()
* These operations vary in efficiency (time complexity) depending on:

Type of data structure used is decided based on:

Size of data

Implementation method

Position of element being operated on

Understanding these operations helps in:

Choosing right data structure for specific needs:

Optimizing program performance

Efficient data management

Problem-solving in programming

**Task 4:**

**What are static and dynamic arrays? Explain or summarize key points in a table like**

**Size, performance, memory, flexibility, limitations**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Static Arrays** | **Dynamic Arrays** |
| Size | • Fixed size at declaration  • Cannot be changed during runtime  • Size must be known at compile time | • Variable size  • Can grow or shrink during runtime  • Automatically resizes as needed |
| Performance | • Faster access time (O(1))  • No resizing overhead  • Better cache utilization  • Direct memory access | • Slightly slower due to resizing operations  • Amortized time complexity for insertion  • Additional overhead for size management |
| Memory | • Fixed memory allocation  • More memory efficient  • No extra space overhead | • Additional memory overhead  • Usually allocates more space than needed  • Typically doubles size when full |
| Flexibility | • Less flexible  • Cannot accommodate more elements than initial size  • Better for fixed-size data | • Highly flexible  • Can adapt to varying data sizes  • Good for unknown or changing data sizes |
| Limitations | • Risk of overflow  • Wastage of space if underutilized  • Cannot shrink to save memory | • Memory fragmentation possible  • Performance cost during resizing  • More complex memory management |

**8-bit representation of capital 'A':**

1. First, find ASCII value of 'A' = 65

2. Convert 65 to binary:

65 ÷ 2 = 32 (Remainder: 1)

32 ÷ 2 = 16 (Remainder: 0)

16 ÷ 2 = 8 (Remainder: 0)

8 ÷ 2 = 4 (Remainder: 0)

4 ÷ 2 = 2 (Remainder: 0)

2 ÷ 2 = 1 (Remainder: 0)

1 ÷ 2 = 0 (Remainder: 1)

Reading from bottom to top: 1000001

3. For 8-bit representation, add leading zeros:

01000001

Therefore, capital 'A' in 8-bit binary = 01000001

Quick check:

- 7th bit (64) = 1

- 0th bit (1) = 1

- All other bits = 0

- Total = 64 + 1 = 65 (ASCII value of 'A')

**Task 5:**

**What is the binary value of a?**

Hint ascii value is 97

Step 1: Convert 97 to binary

97 ÷ 2 = 48 (Remainder: 1)

48 ÷ 2 = 24 (Remainder: 0)

24 ÷ 2 = 12 (Remainder: 0)

12 ÷ 2 = 6 (Remainder: 0)

6 ÷ 2 = 3 (Remainder: 0)

3 ÷ 2 = 1 (Remainder: 1)

1 ÷ 2 = 0 (Remainder: 1)

Reading from top to bottom

8-bit representation of 'a' = 01100001

**Task 6:**

**Types of Computer memory with examples.. Explain ..**

1. Primary Memory (Main Memory)

• RAM (Random Access Memory)

Volatile memory

Examples: DDR3, DDR4, DDR5

Used for temporary data storage

• ROM (Read Only Memory)

Non-volatile memory

Examples: PROM, EPROM, EEPROM

Stores permanent instructions

1. Secondary Memory

• Hard Disk Drives (HDD)

Mechanical storage

Large capacity, slower access

• Solid State Drives (SSD)

Flash-based storage

Faster than HDD, more expensive

• Optical Drives

CD-ROM, DVD, Blu-ray

Removable storage media

1. Cache Memory

• L1 Cache

Smallest, fastest

Located closest to CPU

• L2 Cache

Larger than L1, slightly slower

• L3 Cache

Largest cache, shared between cores

Registers

1. CPU Registers

Fastest memory

Very small capacity

Used for immediate processing

1. Flash Memory

• USB drives

• Memory cards

• NAND flash storage

Key Characteristics:

• Speed: Registers > Cache > RAM > SSD > HDD

• Cost: Higher speed = Higher cost

• Capacity: Higher speed = Lower capacity

• Volatility: RAM is volatile, ROM/Secondary memory is non-volatile

Data Structures

Data structures in Java.docx in docs to study – for reference..

The user has provided a Java programming assignment with file handling and streams tasks. They want me to keep the original structure and add answers below every question. Let me go through each task and provide solutions/corrections.

Let me analyze each task:

Task 1: There are syntax errors in the code - Import should be import (lowercase), and there are issues with the byte array declaration.

Task 2: This is a reading bytes program that should work with the file created in Task 1.

Task 3: Taking user input and writing to file.

Task 4: Character writing using FileWriter - there are syntax errors (Class should be class, missing public).

Task 5: Character reading using FileReader - similar syntax errors.

Task 6: Copy bytes from one file to another - incomplete code that needs to be fixed.

Task 7: Merge two files - syntax errors.

Task 8: Rename file - syntax errors.

Task 9-20: Lambda expressions and streams examples.

Let me provide corrected code for each task.

Hide thinking

**Day 11 - 21st June 2025**

File Handling

Streams

**========================================================================  
File handling:**

**Task 1:**

Run the below code and see the file with the given name created or not..  
Run it again with "I like India" instead of "I love India".. And see the file …

Import java.io.\*;

Import java.io.IOException;

public class WriteByte

{

public static void main(String args[])

{

File f1=new File("FileName01.txt"); \\ to create new file

FileOutputStream outfile = null;

byte Text[] = {'I',' ',''L','O','V','E',' ','I','N','D','I','A'};

try

{

outfile = new FileOutputStream(f1);

outfile.write(Text);

}

catch(IOException e)

{

System.out.println(e);

System.exit(-1);

}

System.out.println("Write Byte");

System.out.println("Thank You...!!!");

}

}

**Answer:**

import java.io.\*;

import java.io.IOException;

public class WriteByte

{

public static void main(String args[])

{

File f1=new File("FileName01.txt"); // to create new file

FileOutputStream outfile = null;

byte Text[] = {'I',' ','L','O','V','E',' ','I','N','D','I','A'};

try

{

outfile = new FileOutputStream(f1);

outfile.write(Text);

outfile.close();

}

catch(IOException e)

{

System.out.println(e);

System.exit(-1);

}

System.out.println("Write Byte");

System.out.println("Thank You...!!!");

}

}

// For "I like India" version:

// byte Text[] = {'I',' ','L','I','K','E',' ','I','N','D','I','A'};

**Task 2:**

Try this code to see the output …  
Write a program which reads byte from file.

import java.io.\*;

public class ReadingByte

{

public static void main(String args[])

{

FileInputStream infile = null;

int b;

try

{

infile = new FileInputStream("FileName01.txt");

while((b = infile.read()) != -1)

{

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

}

infile.close();

}

catch(IOException e)

{

System.out.println("Sorry..!! File Not Found...!!!");

}

}

}

**Answer:**

import java.io.\*;

public class ReadingByte

{

public static void main(String args[])

{

FileInputStream infile = null;

int b;

try

{

infile = new FileInputStream("FileName01.txt");

while((b = infile.read()) != -1)

{

System.out.print((char)b); // Changed to print instead of println for better output

}

infile.close();

}

catch(IOException e)

{

System.out.println("Sorry..!! File Not Found...!!!");

}

}

}

**Task 3:**

Taking input from the user and writing on the file…  
Create a file and see the output…

import java.io.\*;

import java.util.\*;

public class WriteByte\_1

{

public static void main(String args[]) {

FileOutputStream outfile = null;

//String s=args[0]; // to input string from command line Scanner sc=new Scanner(System.in);

String s=sc.nextLine();

byte b1[] = s.getBytes();

try

{

outfile = new FileOutputStream("FileName02.txt");

outfile.write(b1);

}

catch(IOException e)

{

System.out.println(e);

System.exit(-1);

}

System.out.println("Write Byte");

System.out.println("Thank You...!!!");

}

}

**Answer:**

import java.io.\*;

import java.util.\*;

public class WriteByte\_1

{

public static void main(String args[]) {

FileOutputStream outfile = null;

Scanner sc=new Scanner(System.in);

System.out.print("Enter text to write to file: ");

String s=sc.nextLine();

byte b1[] = s.getBytes();

try

{

outfile = new FileOutputStream("FileName02.txt");

outfile.write(b1);

outfile.close();

}

catch(IOException e)

{

System.out.println(e);

System.exit(-1);

}

System.out.println("Write Byte");

System.out.println("Thank You...!!!");

sc.close();

}

}

**=========================================  
Reading/writing characters**

FileReader and FileWriter

**Task 4:**

Write a program which creates file and writes character into that file.

import java.io.\*;

Class CharacterWrite {

public static void main(String args[]) {

File f1=new File("FileName03.txt");

FileWriter fw = null;

try {

fw=new FileWriter(f1);

fw.write("ahmedabad \n");

fw.write(" baroda \n");

fw.close();

}

catch(FileNotFoundException e)

{

System.out.println("Sorry..!! File Not Found...!!!");

}

catch(IOException e)

{

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

}

System.out.println(" write operation done!!");

}

}

**Answer:**

import java.io.\*;

public class CharacterWrite {

public static void main(String args[]) {

File f1=new File("FileName03.txt");

FileWriter fw = null;

try {

fw=new FileWriter(f1);

fw.write("ahmedabad \n");

fw.write(" baroda \n");

fw.close();

}

catch(FileNotFoundException e)

{

System.out.println("Sorry..!! File Not Found...!!!");

}

catch(IOException e)

{

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

}

System.out.println(" write operation done!!");

}

}

**Task 5:**

Write a program which reads character from file.

import java.io.\*;

Class Readchar

{

public static void main(String args[])

{

FileReader fr =null;

try

{

fr = new FileReader("FileName03.txt");

int ch;

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

{

System.out.print((char)ch);

}

System.out.println("Reading complete");

fr.close();

}

catch(FileNotFoundException e)

{

System.out.println("Sorry..!! File Not Found...!!!");

}

catch(IOException e)

{

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

}

}

}

**Answer:**

import java.io.\*;

public class Readchar

{

public static void main(String args[])

{

FileReader fr =null;

try

{

fr = new FileReader("FileName03.txt");

int ch;

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

{

System.out.print((char)ch);

}

System.out.println("\nReading complete");

fr.close();

}

catch(FileNotFoundException e)

{

System.out.println("Sorry..!! File Not Found...!!!");

}

catch(IOException e)

{

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

}

}

}

**Task 6:**

Write a program to read one byte at a time from a file and copy it into another file immediately.

import java.io.\*;

Class CopyByte

{

public static void main(String args[])

{

try

{

byte b=0;

FileInputStream infile = new FileInputStream("NewFile01.txt");

FileOutputStream outfile = new FileOutputStream("NewFile05.txt");

Initialize byteread here….

while(byteread != -1)

{

b = (byte)infile.read();

outfile.write(b);

}

System.out.println("Byte Copied From in.txt to out.txt FIle ");

}

catch(FileNotFoundException e)

{

System.out.println("Sorry..!! File Not Found...!!!");

}

catch(IOException e)

{

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

}

}

}

**Answer:**

import java.io.\*;

public class CopyByte

{

public static void main(String args[])

{

try

{

int byteread = 0; // Initialize byteread here

FileInputStream infile = new FileInputStream("NewFile01.txt");

FileOutputStream outfile = new FileOutputStream("NewFile05.txt");

while((byteread = infile.read()) != -1)

{

outfile.write(byteread);

}

infile.close();

outfile.close();

System.out.println("Byte Copied From NewFile01.txt to NewFile05.txt File ");

}

catch(FileNotFoundException e)

{

System.out.println("Sorry..!! File Not Found...!!!");

}

catch(IOException e)

{

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

}

}

}

**Task 7:**

Merging two files to 3rd file..

Write a program to merge two files in third file.

import java.io.\*;

classFileMergeDemo

{

public static void main(String args[])

{

try

{

FileInputStream file1 = new FileInputStream("NewFile01.txt");

FileInputStream file2 = new FileInputStream("NewFile02.txt");

SequenceInputStream file3 = new SequenceInputStream(file1, file2);

BufferedInputStream br1 = new BufferedInputStream(file3);

BufferedOutputStream br2 = new BufferedOutputStream(System.out);

int ch;

while((ch = br1.read())!=-1)

{

br2.write((char)ch);

}

br1.close();

br2.close();

file1.close();

file2.close();

System.out.println("Merge Two File Sucessfully ");

}

catch(IOException e)

{

System.out.println("Sorry..!! File Not Found...!!!");

}

}

}

**Answer:**

import java.io.\*;

public class FileMergeDemo

{

public static void main(String args[])

{

try

{

FileInputStream file1 = new FileInputStream("NewFile01.txt");

FileInputStream file2 = new FileInputStream("NewFile02.txt");

SequenceInputStream file3 = new SequenceInputStream(file1, file2);

BufferedInputStream br1 = new BufferedInputStream(file3);

BufferedOutputStream br2 = new BufferedOutputStream(new FileOutputStream("MergedFile.txt"));

int ch;

while((ch = br1.read())!=-1)

{

br2.write(ch);

}

br1.close();

br2.close();

file1.close();

file2.close();

System.out.println("Merge Two File Successfully ");

}

catch(IOException e)

{

System.out.println("Sorry..!! File Not Found...!!!");

}

}

}

**Task 8:**

Write an application to rename a file. Use the renameTo() method of File to accomplish this task. The first command line argument is the old filename and the second is the newfilename.

import java.io.\*;

classFileRenameDemo

{

public static void main(String args[])

{

File f1 = new File(args[0]);

File f2 = new File(args[1]);

f1.renameTo(f2);

System.out.println("Rename File " +f1+" To "+f2+" Sucessfully "); }

}

**Answer:**

import java.io.\*;

public class FileRenameDemo

{

public static void main(String args[])

{

if(args.length != 2) {

System.out.println("Usage: java FileRenameDemo <oldfilename> <newfilename>");

return;

}

File f1 = new File(args[0]);

File f2 = new File(args[1]);

if(f1.exists()) {

boolean renamed = f1.renameTo(f2);

if(renamed) {

System.out.println("Rename File " + args[0] + " To " + args[1] + " Successfully ");

} else {

System.out.println("Failed to rename file");

}

} else {

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

}

}

}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
Streams  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Lambda Expressions:

**Task 9**

import java.lang.FunctionalInterface;

// this is functional interface

@FunctionalInterface

interface MyInterface{

// abstract method

double getPiValue();

}

public class Main {

public static void main( String[] args ) {

// declare a reference to MyInterface

MyInterface ref;

// lambda expression

ref = () -> 3.1415;

System.out.println("Value of Pi = " + ref.getPiValue());

}

}

**Answer:**

import java.lang.FunctionalInterface;

// this is functional interface

@FunctionalInterface

interface MyInterface{

// abstract method

double getPiValue();

}

public class Main {

public static void main( String[] args ) {

// declare a reference to MyInterface

MyInterface ref;

// lambda expression

ref = () -> 3.1415;

System.out.println("Value of Pi = " + ref.getPiValue());

}

}

// Output: Value of Pi = 3.1415

**Task 10:**

Write a code to reverse a string..

**Answer:**

@FunctionalInterface

interface StringReverser {

String reverse(String str);

}

public class ReverseString {

public static void main(String[] args) {

StringReverser reverser = (str) -> {

StringBuilder sb = new StringBuilder(str);

return sb.reverse().toString();

};

System.out.println("Reversed: " + reverser.reverse("Hello World"));

}

}

// Output: Reversed: dlroW olleH

**Task 11:**

@FunctionalInterface

interface MyInterface {

// abstract method

String reverse(String n);

}

public class Main {

public static void main( String[] args ) {

// declare a reference to MyInterface

// assign a lambda expression to the reference

MyInterface ref = (str) -> {

String result = "";

for (int i = str.length()-1; i >= 0 ; i--)

result += str.charAt(i);

return result;

};

// call the method of the interface

System.out.println("Lambda reversed = " + ref.reverse("Lambda"));

}

}

**Answer:**

@FunctionalInterface

interface MyInterface {

// abstract method

String reverse(String n);

}

public class Main {

public static void main( String[] args ) {

// declare a reference to MyInterface

// assign a lambda expression to the reference

MyInterface ref = (str) -> {

String result = "";

for (int i = str.length()-1; i >= 0 ; i--)

result += str.charAt(i);

return result;

};

// call the method of the interface

System.out.println("Lambda reversed = " + ref.reverse("Lambda"));

}

}

// Output: Lambda reversed = adbmaL

**Task 12:**

Wap to create an arraylist with 5 friends names..

**Answer:**

import java.util.ArrayList;

import java.util.List;

public class FriendsList {

public static void main(String[] args) {

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

friends.add("Alice");

friends.add("Bob");

friends.add("Charlie");

friends.add("Diana");

friends.add("Edward");

System.out.println("Friends list: " + friends);

// Display using forEach

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

}

}

**Task 13:**

Wap to create a List of 5 friends names (first name and last name)

**Answer:**

import java.util.ArrayList;

import java.util.List;

public class FullNamesList {

public static void main(String[] args) {

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

fullNames.add("Alice Johnson");

fullNames.add("Bob Smith");

fullNames.add("Charlie Brown");

fullNames.add("Diana Prince");

fullNames.add("Edward Norton");

System.out.println("Full names list:");

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

}

}

**Task 14:**

import java.util.ArrayList;

import java.util.List;

public class StreamMain {

// create an object of list using ArrayList

static List<String> places = new ArrayList<>();

// preparing our data

public static List getPlaces(){

// add places and country to the list

places.add("Nepal, Kathmandu");

places.add("Nepal, Pokhara");

places.add("India, Delhi");

places.add("USA, New York");

places.add("Africa, Nigeria");

return places;

}

public static void main( String[] args ) {

List<String> myPlaces = getPlaces();

System.out.println("Places from Nepal:");

// Filter places from Nepal

myPlaces.stream()

.filter((p) -> p.startsWith("Nepal"))

.map((p) -> p.toUpperCase())

.sorted()

.forEach((p) -> System.out.println(p));

}

}

**Answer:**

import java.util.ArrayList;

import java.util.List;

public class StreamMain {

// create an object of list using ArrayList

static List<String> places = new ArrayList<>();

// preparing our data

public static List<String> getPlaces(){

// add places and country to the list

places.add("Nepal, Kathmandu");

places.add("Nepal, Pokhara");

places.add("India, Delhi");

places.add("USA, New York");

places.add("Africa, Nigeria");

return places;

}

public static void main( String[] args ) {

List<String> myPlaces = getPlaces();

System.out.println("Places from Nepal:");

// Filter places from Nepal

myPlaces.stream()

.filter((p) -> p.startsWith("Nepal"))

.map((p) -> p.toUpperCase())

.sorted()

.forEach((p) -> System.out.println(p));

}

}

/\* Output:

Places from Nepal:

NEPAL, KATHMANDU

NEPAL, POKHARA

\*/

**Task 15:**

Collect : Terminal operator..

Wap to accept or create a list of 5 integers and display the squares of each ..

**Answer:**

import java.util.\*;

import java.util.stream.Collectors;

public class SquareNumbers {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

List<Integer> squareOfNums = numbers.stream()

.map(num -> num \* num)

.collect(Collectors.toList());

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

System.out.println("Squares: " + squareOfNums);

}

}

/\* Output:

Original numbers: [1, 2, 3, 4, 5]

Squares: [1, 4, 9, 16, 25]

\*/

**Task 16**

Write a code to create an array list and filter the values which are odd numbers and display them..

**Answer:**

import java.util.\*;

import java.util.stream.Collectors;

public class FilterOddNumbers {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

List<Integer> oddNumbers = numbers.stream()

.filter(num -> num % 2 != 0)

.collect(Collectors.toList());

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

System.out.println("Odd numbers: " + oddNumbers);

}

}

/\* Output:

Original numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Odd numbers: [1, 3, 5, 7, 9]

\*/

**Task 17:**

Wap to create an array list to remove duplicate values from the List.

**Answer:**

import java.util.\*;

import java.util.stream.Collectors;

public class RemoveDuplicates {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 2, 4, 3, 5, 1, 6);

List<Integer> removeDups = numbers.stream()

.distinct()

.collect(Collectors.toList());

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

System.out.println("After removing duplicates: " + removeDups);

}

}

/\* Output:

Original numbers: [1, 2, 3, 2, 4, 3, 5, 1, 6]

After removing duplicates: [1, 2, 3, 4, 5, 6]

\*/

**Task 18:**

Wap to run a loop / iterate() and limit it to 20 values (1 to 20)  
While displaying use for each to limit till 10 numbers.

**Answer:**

import java.util.stream.Stream;

public class StreamIterate {

public static void main(String[] args) {

Stream<Integer> nums = Stream

.iterate(1, n -> n + 1)

.limit(20);

System.out.println("First 10 numbers from 1-20:");

nums

.limit(10)

.forEach(System.out::println);

}

}

/\* Output:

First 10 numbers from 1-20:

1

2

3

4

5

6

7

8

9

10

\*/

**Task 19:**

Wap to create an array List skip 15 numbers and print the output using foreach loop

**Answer:**

import java.util.stream.Stream;

public class StreamSkip {

public static void main(String[] args) {

Stream<Integer> nums = Stream

.iterate(1, n -> n + 1)

.limit(20);

System.out.println("Numbers after skipping first 15:");

nums.skip(15)

.forEach(System.out::println);

}

}

/\* Output:

Numbers after skipping first 15:

16

17

18

19

20

\*/

**Task 20**

import java.util.Arrays;

import java.util.List;

import java.util.Optional;

public class ReduceExample {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

Optional<Integer> sum = numbers.stream().reduce((x, y) -> x + y);

System.out.println("Sum of all elements: " + sum.orElse(0));

Optional<Integer> max = numbers.stream().reduce(Integer::max);

System.out.println("Maximum element: " + max.orElse(0));

List<String> strings = Arrays.asList("Hello", " ", "world", "!");

Optional<String> concatenatedString = strings.stream().reduce((x, y) -> x + y);

System.out.println("Concatenated string: " + concatenatedString.orElse(""));

}

}

**Answer:**

import java.util.Arrays;

import java.util.List;

import java.util.Optional;

public class ReduceExample {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

Optional<Integer> sum = numbers.stream().reduce((x, y) -> x + y);

System.out.println("Sum of all elements: " + sum.orElse(0));

Optional<Integer> max = numbers.stream().reduce(Integer::max);

System.out.println("Maximum element: " + max.orElse(0));

List<String> strings = Arrays.asList("Hello", " ", "world", "!");

Optional<String> concatenatedString = strings.stream().reduce((x, y) -> x + y);

System.out.println("Concatenated string: " + concatenatedString.orElse(""));

}

}

/\* Output:

Sum of all elements: 15

Maximum element: 5

Concatenated string: Hello world!

\*/

When to use reduce and when to use collect..

**Reduce** will be used if you are expecting a single result from the stream (eg min, max , sum, product…)

**Collect** will be used if you are expecting a list of values… (list, set, map)

**Home tasks:**

==================================================

**File Handling add ons**

**Buffered reader and writer — for large files to be read.**

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class ReadFileExample {

public static void main(String[] args) {

try (BufferedReader br = new BufferedReader(new FileReader("largefile.txt"))) {

String line;

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

System.out.println(line);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

non buffered

import java.io.FileReader;

import java.io.IOException;

public class NonBufferedReaderExample {

public static void main(String[] args) {

try (FileReader fr = new FileReader("largefile.txt")) {

int ch;

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

System.out.print((char) ch);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

Buffered Writer

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

public class BufferedWriterDemo {

public static void main(String[] args) {

String filePath = "example.txt";

String content = "Hello, World!\nThis is a BufferedWriter example.";

// Initialize BufferedWriter with a FileWriter

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

// Write content to the file

writer.write(content);

System.out.println("Content written to file.");

} catch (IOException e) {

System.err.println("An error occurred: " + e.getMessage());

}

}

}

Add ons — Java 8 features nio

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.nio.file.StandardCopyOption;

public class ImageCopier {

public static void main(String[] args) {

// Define the source and destination file paths

Path sourcePath = Paths.get("path/to/your/source\_image.jpg"); // Replace with your source image path

Path destinationPath = Paths.get("path/to/your/destination\_image.jpg"); // Replace with your desired destination path

try {

// Copy the file

// REPLACE\_EXISTING option overwrites the destination file if it already exists

Files.copy(sourcePath, destinationPath, StandardCopyOption.REPLACE\_EXISTING);

System.out.println("Image copied successfully from " + sourcePath + " to " + destinationPath);

} catch (IOException e) {

System.err.println("Error copying image: " + e.getMessage());

e.printStackTrace();

}

}

}

========================================================================

Info Box

Data structures - a few codes - for reference

https://drive.google.com/drive/folders/1OXACrFwF5hQ2WVSHh2gP1EZDF\_PcEy\_U?usp=sharing

Java core recording — only for beginners

https://drive.google.com/drive/folders/1j8dVg\_RjjnDAayAbCxZFWMewCH99WHnq?usp=sharing

Next week's topics: plz be prepared..

Data Structure  
Section Overview, What Is A Data Structure?, How Computers Store Data, Data Structures In Different Languages, Operations On Data Structures,  
Data Structures: Arrays  
Arrays Introduction, Static vs Dynamic Arrays, Implementing An Array, Strings and Arrays, Exercise: Reverse A String, Solution: Reverse A String, Exercise: Merge Sorted Arrays, Solution: Merge Sorted Arrays,  
Hash Tables  
Hash Tables Introduction, Hash Function, Hash Collisions, Hash Tables In Different Languages, Exercise: Implement A Hash Table, Solution: Implement A Hash Table, keys(), Extra: keys() Without Collision, Hash Tables VS Arrays,  
Linked List  
Linked Lists Introduction, What Is A Linked List?, Exercise: Imposter Syndrome, Exercise: Why Linked Lists?, Solution: Why Linked Lists?, What Is A Pointer?, Our First Linked List, Solution: append(), Solution: prepend(), Node Class, insert(), Solution: insert(), Solution: remove(), Doubly Linked Lists, Exercise: Doubly Linked Lists, Solution: Doubly Linked Lists, Singly VS Doubly Linked Lists,

========================================================================