Day 9 -Aditya - 13th June 2025

|  |
| --- |
| Inheritance, Polymorphism, Encapsulation, Abstraction, Interfaces, Exception Handling, |
| Collections Framework intro, Streams, File I/O, Multithreading overview |

Exceptions:

class -----> class ======> extends ====> no multiple inheritance

interface ----> class =====> Implements

interface ---> interface ====> extends

Exceptions: --> unexpected situations

2 categoried

- Checked ---> compile time

- Unchecked ---> Runtime time..

pre defined exception classes

FileNotFound

ArrayIndexoutofBounds exception

NullPointer Exception

Arthematic Exception ...

user defined Exceptions

try catch blocks

try{

}catch(Exception ex){

}catch(ArthematicException ex1{

}

finally{

sout(" ");

}

Task 1:

public class Main {

public static void main(String[] args) {

int[] myNumbers = {1, 2, 3};

System.out.println(myNumbers[10]);

}

}

public class Main {

public static void main(String[] args) {

int[] myNumbers = {1, 2, 3};

System.*out*.println(myNumbers[10]);

}

}

Output:

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 3

at Main.main(Main.java:4)

Process finished with exit code 1

Task 2:

public class Main {

public static void main(String[] args) {

try {

int[] myNumbers = {1, 2, 3};

System.out.println(myNumbers[10]);

} catch (Exception e) {

System.out.println("Something went wrong.");

}

}

}

public class Practice {

public static void main(String[] args) {

try {

int[] myNumbers = {1, 2, 3};

System.*out*.println(myNumbers[10]);

} catch (Exception e) {

System.*out*.println("Something went wrong.");

}

}

}

Output:

"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea\_rt.jar=54756" -Dfile.encoding=UTF-8 -classpath C:\Users\adishum\IdeaProjects\ATLAS\_DAY\_9\out\production\ATLAS\_DAY\_9 Practice

Picked up JAVA\_TOOL\_OPTIONS: -Dlog4j2.formatMsgNoLookups=true

Something went wrong.

Process finished with exit code 0

Task 3:

public class Main {

public static void main(String[] args) {

try {

int[] myNumbers = {1, 2, 3};

System.out.println(myNumbers[10]);

} catch (Exception e) {

System.out.println("Something went wrong.");

} finally {

System.out.println("I'm from finally block.");

}

}

}

public class Main {

public static void main(String[] args) {

try {

int[] myNumbers = {1, 2, 3};

System.*out*.println(myNumbers[10]);

} catch (Exception e) {

System.*out*.println("Something went wrong.");

} finally {

System.*out*.println("I'm from finally block.");

}

}

}

Output:

Something went wrong.

I'm from finally block.

Process finished with exit code 0

Task 4:

from the above code from task 3 .. try accessing the 2 or the 3rd index and check the out..

By 10.23 plz finish till task 4..

public class Main {

public static void main(String[] args) {

int[] myNumbers = {1, 2, 3}; // Valid indexes are 0, 1, and 2

// Accessing index 2 (3rd element)

try {

System.*out*.println("Accessing index 2:");

System.*out*.println(myNumbers[2]); // This will work (value is 3)

} catch (Exception e) {

System.*out*.println("Something went wrong.");

} finally {

System.*out*.println("I'm from finally block.");

}

// Accessing index 3 (4th element - doesn't exist)

try {

System.*out*.println("\nAccessing index 3:");

System.*out*.println(myNumbers[3]); // This will throw an exception

} catch (Exception e) {

System.*out*.println("Something went wrong.");

} finally {

System.*out*.println("I'm from finally block.");

}

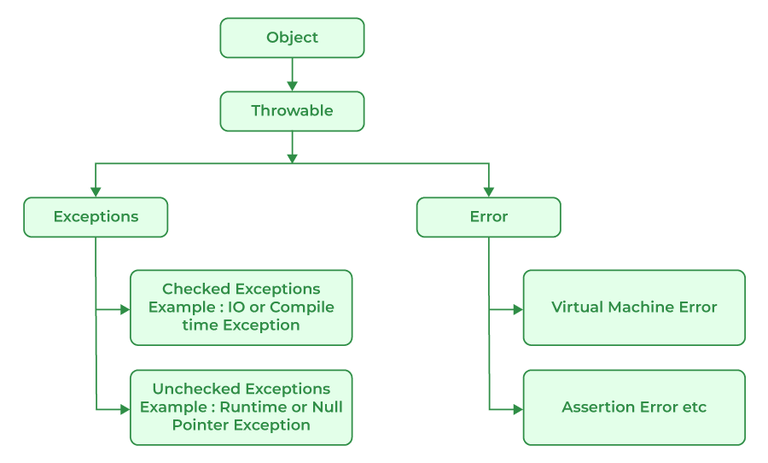
}

}

Output:

Accessing index 3:

Something went wrong.

I'm from finally block.

Task 5

What is the output of the below code… give your reason for the output

public class ExcepTest {

public static void main(String args[]) {

try {

int a[] = new int[2];

int b = 0;

int c = 1/b;

System.out.println("Access element three :" + a[3]);

}

catch (ArithmeticException e) {

System.out.println("ArithmeticException thrown :" + e);

}

catch (ArrayIndexOutOfBoundsException e) {

System.out.println("ArrayIndexOutOfBoundsException thrown :" + e);

}catch (Exception e) {

System.out.println("Exception thrown :" + e);

}

System.out.println("Out of the block");

}

}

public class ExcepTest {

public static void main(String args[]) {

try {

int a[] = new int[2]; // Creates array of size 2 (indexes 0 and 1)

int b = 0; // b is initialized to 0

int c = 1/b; // Division by zero will occur here!

System.*out*.println("Access element three :" + a[3]); // This line won't execute

}

catch (ArithmeticException e) {

System.*out*.println("ArithmeticException thrown :" + e);

}

catch (ArrayIndexOutOfBoundsException e) {

System.*out*.println("ArrayIndexOutOfBoundsException thrown :" + e);

}

catch (Exception e) {

System.*out*.println("Exception thrown :" + e);

}

System.*out*.println("Out of the block");

}

}

Output:

ArithmeticException thrown :java.lang.ArithmeticException: / by zero

Out of the block

Reason for this output:

Program Flow:

* Array 'a' is created with size 2
* Variable 'b' is set to 0
* Program attempts to divide 1 by 0 (1/b)
* ArithmeticException is thrown immediately
* The array access statement is never reached

Exception Handling:

The ArithmeticException is caught by the first catch block

Other catch blocks are skipped

Task 5:

Try with Multiple catch blocks …. Execute the below code snippet n display the out .. along with reason..

public class ExcepTest {

public static void main(String args[]) {

try {

int a[] = new int[2];

int b = 0; ⇒ change 0 to 10 and check..

int c = 1/b;

System.out.println("Access element three :" + a[3]);

}

catch (ArrayIndexOutOfBoundsException e) {

System.out.println("ArrayIndexOutOfBoundsException thrown :" + e);

}catch (Exception e) {

System.out.println("Exception thrown :" + e.getMessage());

}

System.out.println("Out of the block");

}

}

public class ExcepTest {

public static void main(String args[]) {

// Scenario 1: with b = 0

try {

int a[] = new int[2];

int b = 0;

int c = 1/b; // ArithmeticException occurs here

System.*out*.println("Access element three :" + a[3]); // Never reached

} catch (ArrayIndexOutOfBoundsException e) {

System.*out*.println("ArrayIndexOutOfBoundsException thrown :" + e);

} catch (Exception e) {

System.*out*.println("Exception thrown :" + e.getMessage());

}

System.*out*.println("Out of the block");

}

}

Output with b = 0:

Exception thrown :/ by zero

Out of the block

public class ExcepTest {

public static void main(String args[]) {

// Scenario 1: with b = 0

try {

int a[] = new int[2];

int b = 10;

int c = 1/b; // ArithmeticException occurs here

System.*out*.println("Access element three :" + a[3]); // Never reached

} catch (ArrayIndexOutOfBoundsException e) {

System.*out*.println("ArrayIndexOutOfBoundsException thrown :" + e);

} catch (Exception e) {

System.*out*.println("Exception thrown :" + e.getMessage());

}

System.*out*.println("Out of the block");

}

}

Output with b = 10:

ArrayIndexOutOfBoundsException thrown :java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 2

Out of the block

Task 6:

Nested try blocks

public class ExcepTest {

public static void main(String args[]) {

try {

int a[] = new int[2];

System.out.println("Access element three :" + a[2]); // try with a[0] or a[1] ===> and check if control goes to inner try block..

try {

int b = 0;

int c = 1/b;

}catch(Exception e) {

System.out.println("Exception thrown: " + e);

}

System.out.println("Access element three :" + a[3]);

}

catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Exception thrown: " + e);

}

System.out.println("Out of the block");

}

}

public class ExcepTest {

public static void main(String args[]) {

try {

int a[] = new int[2];

System.*out*.println("Access element three :" + a[2]); // ArrayIndexOutOfBoundsException

try {

int b = 0;

int c = 1/b; // ArithmeticException (never reached in original code)

} catch(Exception e) {

System.*out*.println("Exception thrown: " + e);

}

System.*out*.println("Access element three :" + a[3]);

} catch (ArrayIndexOutOfBoundsException e) {

System.*out*.println("Exception thrown: " + e);

}

System.*out*.println("Out of the block");

}

}

With a[2]:

Exception thrown: java.lang.ArrayIndexOutOfBoundsException: Index 2 out of bounds for length 2

Out of the block

5 min

11.05 to 11.10

—-----------------------------------------------------------------------------------------

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

Throw and Throws:

Void divide() throws Excepion{

throw …….

}

Void arr2() throws ArrayIndexout….. {

}

Void arr3() throws ArrayIndexout….. {

}

Void method3() throws Exception{

}

psvmain() {

Try{

divide();

arr2();

}catch(ArrayInde…. eobj1){

}catch(Exception eobj2) {

}Finally{

}

}

Task 7:

// Demonstrating how to throw an exception

class MyClass {

static void fun() throws IllegalAccessException

{

System.out.println("Inside fun(). ");

throw new IllegalAccessException("demo exception by fun method");

}

public static void main(String args[])

{

try {

fun();

//method2(); → arrayindex…

//Method3() —> file not found….

}

catch (IllegalAccessException e) {

System.out.println("Caught in main." + e.getMessage());

}

}

}

public class ExceptionDemo {

// Method that throws Exception

static void divide() throws Exception {

int result = 5/0;

throw new Exception("Division by zero error");

}

// Method that throws ArrayIndexOutOfBoundsException

static void arr2() throws ArrayIndexOutOfBoundsException {

int[] arr = new int[2];

System.*out*.println(arr[3]); // This will throw the exception

}

// Another array method

static void arr3() throws ArrayIndexOutOfBoundsException {

int[] arr = new int[3];

System.*out*.println(arr[4]); // This will throw the exception

}

// Method that throws general Exception

static void method3() throws Exception {

throw new Exception("Generic exception from method3");

}

public static void main(String[] args) {

try {

*divide*(); // This will throw Exception

*arr2*(); // This won't be executed due to previous exception

} catch(ArrayIndexOutOfBoundsException eobj1) {

System.*out*.println("Array index error: " + eobj1.getMessage());

} catch(Exception eobj2) {

System.*out*.println("General error: " + eobj2.getMessage());

} finally {

System.*out*.println("Finally block executed");

}

}

}

General error: / by zero

Finally block executed

Task 8:

Custom exceptions: // user defined exception:

// A Class that represents user-defined exception

class MyException extends Exception {

public MyException(String m) {

super(m);

}

}

// A Class that uses the above MyException

public class setText {

public static void main(String args[]) {

try {

// Throw an object of user-defined exception

throw new MyException("This is a custom exception");

}

catch (MyException ex) {

System.out.println("Caught");

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

}

}

}

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

—----------------------------------------------------------------------------------------

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

Task 9:

import java.util.ArrayList;

class Main {

public static void main (String[] args) {

// Creating an ArrayList

ArrayList<Integer> a = new ArrayList<Integer>();

// Adding Element in ArrayList

a.add(1);

a.add(2);

a.add(3);

// Printing ArrayList

System.out.println(a);

}

}

import java.util.ArrayList;

public class ArrayListDemo {

public static void main(String[] args) {

// Creating ArrayList

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

// Adding elements

fruits.add("Apple");

fruits.add("Banana");

fruits.add("Orange");

// Printing ArrayList

System.*out*.println("Initial List: " + fruits);

// Adding element at specific index

fruits.add(1, "Mango");

System.*out*.println("After adding Mango at index 1: " + fruits);

// Removing element

fruits.remove("Banana");

System.*out*.println("After removing Banana: " + fruits);

// Checking if element exists

System.*out*.println("Contains Apple? " + fruits.contains("Apple"));

// Getting element at index

System.*out*.println("Element at index 1: " + fruits.get(1));

// Size of ArrayList

System.*out*.println("Size of ArrayList: " + fruits.size());

}

}

Output:

Initial List: [Apple, Banana, Orange]

After adding Mango at index 1: [Apple, Mango, Banana, Orange]

After removing Banana: [Apple, Mango, Orange]

Contains Apple? true

Element at index 1: Mango

Size of ArrayList: 3

Note: You can also create a generic ArrayList

Important Features of ArrayList in Java

ArrayList inherits AbstractList class and implements the List interface.

ArrayList is initialized by size. However, the size is increased automatically if the collection grows or shrinks if the objects are removed from the collection.

Java ArrayList allows us to randomly access the list.

ArrayList can not be used for primitive types, like int, char, etc. We need a wrapper class for such cases.

ArrayList in Java can be seen as a vector in C++.

ArrayList is not Synchronized. Its equivalent synchronized class in Java is Vector.

Task 10

Find the output of the be code snippet..

// Addition, Deletion and Updation of Element

import java.util.\*;

class Main {

public static void main(String args[]){

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

al.add("Prasunamba");

al.add("Meher");

System.out.println("Orignal List : "+al);

al.add(1, "Hello");

System.out.println("After Adding element at index 1 : "+ al);

al.remove(0);

System.out.println("Element removed from index 0 : "+ al);

al.remove("Prasunamba");

System.out.println("Element Prasunamba removed : "+ al);

al.set(0, "K");

System.out.println("List after updation of value : "+al);

}

}

Output:

Original List : [Prasunamba, Meher]

After Adding element at index 1 : [Prasunamba, Hello, Meher]

Element removed from index 0 : [Hello, Meher]

Element Prasunamba removed : [Hello, Meher]

List after updation of value : [K, Meher]

The user wants me to provide answers for tasks 11-20, keeping the original structure as is but providing my answers below each task. Let me go through each one:

Task 11: Inner class output  
Task 12: Private inner class  
Task 13: Static inner class  
Task 14: Inner class method accessing outer variable  
Task 15: Static inner class with static variable  
Task 16: Home task about inheritance  
Task 17: Java 8 features  
Task 18: forEach example output  
Task 19: Link reference  
Task 20: Collections demo output

I need to keep their original formatting and structure intact, then provide my answers.

Hide thinking

Task 11:  
What is the output of the below code snippet.. Explain ..  
class OuterClass {  
int x = 10;  
class InnerClass {  
int y = 5;  
}  
}

public class Main {  
public static void main(String[] args) {  
OuterClass myOuter = new OuterClass();  
OuterClass.InnerClass myInner = myOuter.new InnerClass();  
System.out.println(myInner.y + myOuter.x);  
}  
}

**Answer:**

15

**Explanation:**

* OuterClass has variable x = 10
* InnerClass has variable y = 5
* We create an outer class object and then create inner class object using myOuter.new InnerClass()
* The output is myInner.y (5) + myOuter.x (10) = 15

Task 12:  
Use the above code and make the inner class as private and see the output..  
Ex: private class InnerClass {

**Answer:**

class OuterClass {

int x = 10;

private class InnerClass {

int y = 5;

}

}

public class Main {

public static void main(String[] args) {

OuterClass myOuter = new OuterClass();

OuterClass.InnerClass myInner = myOuter.new InnerClass(); // Compilation Error

System.out.println(myInner.y + myOuter.x);

}

}

**Output:**

Compilation Error: OuterClass.InnerClass has private access in OuterClass

**Explanation:** Private inner classes cannot be accessed from outside the outer class. Only the outer class can create instances of private inner classes.

Task 13:  
Use the above code Task 011 and make the inner class static … see the output and explain..  
Ex: static class InnerClass {

**Answer:**

class OuterClass {

int x = 10;

static class InnerClass {

int y = 5;

}

}

public class Main {

public static void main(String[] args) {

OuterClass myOuter = new OuterClass();

OuterClass.InnerClass myInner = new OuterClass.InnerClass(); // Changed syntax

System.out.println(myInner.y + myOuter.x);

}

}

**Output:**

15

**Explanation:**

* Static inner classes don't need an instance of outer class to be created
* We can create static inner class using new OuterClass.InnerClass() instead of myOuter.new InnerClass()
* Static inner classes cannot access non-static members of outer class directly

Task 14: Use the above code Task 11 and create a method in innerclass and return the outer class variable  
class OuterClass{  
Int x = 50;  
Class InnerClass {  
Public int innerMethod() {  
Return x;  
}  
}  
}

Public class DriverClass {  
psvm(){  
OuterClass myOuter = new OuterClass();  
OuterClass.InnerClass myInner = myOuter.new InnerClass();  
System.out.println(myInner.innerMethod());  
}  
}

**Answer:**

class OuterClass{

int x = 50;

class InnerClass {

public int innerMethod() {

return x; // Inner class can access outer class variables

}

}

}

public class DriverClass {

public static void main(String[] args){

OuterClass myOuter = new OuterClass();

OuterClass.InnerClass myInner = myOuter.new InnerClass();

System.out.println(myInner.innerMethod());

}

}

**Output:**

50

**Explanation:** Inner classes have access to all members (including private) of the outer class. The innerMethod() returns the outer class variable x directly.

Task 15:  
Give reason or the code..  
class OuterClass {  
int x = 10;  
static class InnerClass {  
static int y = 5;  
}  
}  
public class Main {  
public static void main(String[] args) {  
OuterClass.InnerClass myInner = new OuterClass.InnerClass();  
System.out.println(myInner.y);

}  
}

**Answer:**

5

**Explanation:**

* This code demonstrates static nested class with static variable
* Static inner class InnerClass has static variable y = 5
* We create instance of static inner class using new OuterClass.InnerClass()
* We access static variable y through the instance, which prints 5
* **Note:** Static variables can be accessed through class name as well: OuterClass.InnerClass.y

—----------------------------------------------------------------

Practice oops:  
Task 16: ⇒ Home task..  
Inheritance, Abstraction concepts..  
Classes customer/ person , employee, Manager … 2 variables in each class  
Driver class – display all the variables… toString();  
Hint : use getter and setters..  
Person (parent) —---> Employee (child) —--> Manager (sub Child)

**Answer:**

class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

// Getters and Setters

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 + "}";

}

}

class Employee extends Person {

private String empId;

private double salary;

public Employee(String name, int age, String empId, double salary) {

super(name, age);

this.empId = empId;

this.salary = salary;

}

// Getters and Setters

public String getEmpId() { return empId; }

public void setEmpId(String empId) { this.empId = empId; }

public double getSalary() { return salary; }

public void setSalary(double salary) { this.salary = salary; }

@Override

public String toString() {

return "Employee{" + super.toString() + ", empId='" + empId + "', salary=" + salary + "}";

}

}

class Manager extends Employee {

private String department;

private int teamSize;

public Manager(String name, int age, String empId, double salary, String department, int teamSize) {

super(name, age, empId, salary);

this.department = department;

this.teamSize = teamSize;

}

// Getters and Setters

public String getDepartment() { return department; }

public void setDepartment(String department) { this.department = department; }

public int getTeamSize() { return teamSize; }

public void setTeamSize(int teamSize) { this.teamSize = teamSize; }

@Override

public String toString() {

return "Manager{" + super.toString() + ", department='" + department + "', teamSize=" + teamSize + "}";

}

}

public class DriverClass {

public static void main(String[] args) {

Person person = new Person("John", 25);

Employee employee = new Employee("Alice", 30, "E001", 50000);

Manager manager = new Manager("Bob", 35, "M001", 80000, "IT", 10);

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

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

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

}

}

Task 17:  
What are the features of Java 8?

Java 8 Features:

Lambda Exp..  
Interfaces can have body (default and static)  
Foreach  
Collection Api  
Stream Api  
Java io improvements  
Functional Interfaces  
Method references  
Time ApI

**Answer:**  
**Java 8 Key Features:**

1. **Lambda Expressions** - Enable functional programming
2. **Default Methods in Interfaces** - Interfaces can have method implementations
3. **Static Methods in Interfaces** - Interfaces can have static methods
4. **forEach() Method** - Enhanced iteration over collections
5. **Stream API** - Functional-style operations on collections
6. **Functional Interfaces** - Interfaces with single abstract method (@FunctionalInterface)
7. **Method References** - Shorthand for lambda expressions (::)
8. **Optional Class** - Handle null values gracefully
9. **New Date/Time API** - LocalDate, LocalTime, LocalDateTime
10. **Collectors Class** - Utility methods for Stream operations
11. **Parallel Arrays** - Support for parallel operations on arrays
12. **Base64 Encoding/Decoding** - Built-in support

Task 18:  
Run the code and give reason for the output..  
import java.util.ArrayList;  
import java.util.Iterator;  
import java.util.List;  
import java.util.function.Consumer;  
import java.lang.Integer;

public class Java8ForEachExample {

public static void main(String[] args) {

List<Integer> myList = new ArrayList<Integer>();  
for(int i=0; i<10; i++) myList.add(i);

Iterator<Integer> it = myList.iterator();  
while(it.hasNext()){  
Integer i = it.next();  
System.out.println("Iterator Value::"+i);  
}

myList.forEach(new Consumer<Integer>() {

public void accept(Integer t) {  
System.out.println("forEach anonymous class Value::"+t);  
}

});

//traversing with Consumer interface implementation  
MyConsumer action = new MyConsumer();  
myList.forEach(action);

}

}

//Consumer implementation that can be reused  
class MyConsumer implements Consumer<Integer>{

public void accept(Integer t) {  
System.out.println("Consumer impl Value::"+t);  
}  
}

**Answer:**

Iterator Value::0

Iterator Value::1

Iterator Value::2

Iterator Value::3

Iterator Value::4

Iterator Value::5

Iterator Value::6

Iterator Value::7

Iterator Value::8

Iterator Value::9

forEach anonymous class Value::0

forEach anonymous class Value::1

forEach anonymous class Value::2

forEach anonymous class Value::3

forEach anonymous class Value::4

forEach anonymous class Value::5

forEach anonymous class Value::6

forEach anonymous class Value::7

forEach anonymous class Value::8

forEach anonymous class Value::9

Consumer impl Value::0

Consumer impl Value::1

Consumer impl Value::2

Consumer impl Value::3

Consumer impl Value::4

Consumer impl Value::5

Consumer impl Value::6

Consumer impl Value::7

Consumer impl Value::8

Consumer impl Value::9

**Explanation:**

1. **First block:** Traditional Iterator pattern iterates through list 0-9
2. **Second block:** forEach with anonymous Consumer implementation iterates 0-9
3. **Third block:** forEach with separate Consumer class implementation iterates 0-9  
   **Reason:** Three different ways to iterate same list, demonstrating Java 8 forEach functionality vs traditional iteration.

Task 19:  
https://www.digitalocean.com/community/tutorials/java-8-features-with-examples  
Plz go through this link for Java 8 features..

Task 20:  
Collection Interfaces:

import java.util.\*;  
public class CollectionsDemo {

public static void main(String[] args) {  
// ArrayList  
List a1 = new ArrayList();  
a1.add("Zara");  
a1.add("Mahnaz");  
a1.add("Ayan");  
System.out.println(" ArrayList Elements");  
System.out.print("\t" + a1);

// LinkedList

List l1 = new LinkedList();

l1.add("Zara");

l1.add("Mahnaz");

l1.add("Ayan");

System.out.println();

System.out.println(" LinkedList Elements");

System.out.print("\t" + l1);

// HashSet

Set s1 = new HashSet();

s1.add("Zara");

s1.add("Mahnaz");

s1.add("Ayan");

System.out.println();

System.out.println(" Set Elements");

System.out.print("\t" + s1);

// HashMap

Map m1 = new HashMap();

m1.put("Zara", "8");

m1.put("Mahnaz", "31");

m1.put("Ayan", "12");

m1.put("Daisy", "14");

System.out.println();

System.out.println(" Map Elements");

System.out.print("\t" + m1);

}  
}

**Answer:**

ArrayList Elements

[Zara, Mahnaz, Ayan]

LinkedList Elements

[Zara, Mahnaz, Ayan]

Set Elements

[Ayan, Mahnaz, Zara]

Map Elements

{Ayan=12, Daisy=14, Mahnaz=31, Zara=8}

**Explanation:**

* **ArrayList & LinkedList:** Maintain insertion order, allow duplicates
* **HashSet:** No duplicates, no guaranteed order (may vary in different runs)
* **HashMap:** Key-value pairs, no guaranteed order, keys are unique
* **Note:** HashSet and HashMap order may differ in actual execution due to hashing

The user has provided a comprehensive Java 8 Stream API example code. This code demonstrates the difference between traditional Java 7 approaches and Java 8 Stream API approaches for various operations like:

1. Counting empty strings
2. Counting strings of specific length
3. Filtering non-empty strings
4. Joining strings with a separator
5. Getting squares of distinct numbers
6. Finding max, min, sum, and average of numbers
7. Generating random numbers
8. Parallel processing

The user wants me to analyze this code and provide the output and explanation in the same format as before - keeping their input structure and adding my answer below.

Let me trace through the code:

1. Java 7 approach:
   * List: [abc, , bc, efg, abcd, , jkl]
   * Empty strings: 2 (two empty strings)
   * Strings of length 3: 2 (abc, efg)
   * Filtered list: [abc, bc, efg, abcd, jkl] (removed empty strings)
   * Merged string: abc, bc, efg, abcd, jkl
   * Numbers: [3, 2, 2, 3, 7, 3, 5]
   * Squares list: [9, 4, 49, 25] (distinct squares)
   * Integers: [1,2,13,4,15,6,17,8,19]
   * Max: 19, Min: 1, Sum: 85, Average: 9 (85/9 = 9 in integer division)
   * Random numbers: 10 random integers
2. Java 8 approach:
   * Same operations but using Stream API
   * Results should be the same
   * Random numbers will be sorted
   * Parallel processing example at the end

Let me provide a comprehensive answer.

Hide thinking

Streams:

Stream was introduced in Java 8, the Stream API is used to process collections of objects. A stream in Java is a sequence of objects that supports various methods that can be pipelined to produce the desired result.

import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.IntSummaryStatistics;  
import java.util.List;  
import java.util.Random;  
import java.util.stream.Collectors;  
import java.util.Map;

public class Java8Tester {

public static void main(String args[]) {  
System.out.println("Using Java 7: ");

// Count empty strings

List<String> strings = Arrays.asList("abc", "", "bc", "efg", "abcd","", "jkl");

System.out.println("List: " +strings);

long count = getCountEmptyStringUsingJava7(strings);

System.out.println("Empty Strings: " + count);

count = getCountLength3UsingJava7(strings);

System.out.println("Strings of length 3: " + count);

//Eliminate empty string

List<String> filtered = deleteEmptyStringsUsingJava7(strings);

System.out.println("Filtered List: " + filtered);

//Eliminate empty string and join using comma.

String mergedString = getMergedStringUsingJava7(strings,", ");

System.out.println("Merged String: " + mergedString);

List<Integer> numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);

//get list of square of distinct numbers

List<Integer> squaresList = getSquares(numbers);

System.out.println("Squares List: " + squaresList);

List<Integer> integers = Arrays.asList(1,2,13,4,15,6,17,8,19);

System.out.println("List: " +integers);

System.out.println("Highest number in List : " + getMax(integers));

System.out.println("Lowest number in List : " + getMin(integers));

System.out.println("Sum of all numbers : " + getSum(integers));

System.out.println("Average of all numbers : " + getAverage(integers));

System.out.println("Random Numbers: ");

//print ten random numbers

Random random = new Random();

for(int i = 0; i < 10; i++) {

System.out.println(random.nextInt());

}

System.out.println("Using Java 8: ");

System.out.println("List: " +strings);

count = strings.stream().filter(string->string.isEmpty()).count();

System.out.println("Empty Strings: " + count);

count = strings.stream().filter(string -> string.length() == 3).count();

System.out.println("Strings of length 3: " + count);

filtered = strings.stream().filter(string ->!string.isEmpty()).collect(Collectors.toList());

System.out.println("Filtered List: " + filtered);

mergedString = strings.stream().filter(string ->!string.isEmpty()).collect(Collectors.joining(", "));

System.out.println("Merged String: " + mergedString);

squaresList = numbers.stream().map( i ->i\*i).distinct().collect(Collectors.toList());

System.out.println("Squares List: " + squaresList);

System.out.println("List: " +integers);

IntSummaryStatistics stats = integers.stream().mapToInt((x) ->x).summaryStatistics();

System.out.println("Highest number in List : " + stats.getMax());

System.out.println("Lowest number in List : " + stats.getMin());

System.out.println("Sum of all numbers : " + stats.getSum());

System.out.println("Average of all numbers : " + stats.getAverage());

System.out.println("Random Numbers: ");

random.ints().limit(10).sorted().forEach(System.out::println);

//parallel processing

count = strings.parallelStream().filter(string -> string.isEmpty()).count();

System.out.println("Empty Strings: " + count);

}

private static int getCountEmptyStringUsingJava7(List<String> strings) {  
int count = 0;

for(String string: strings) {

if(string.isEmpty()) {

count++;

}

}

return count;

}

private static int getCountLength3UsingJava7(List<String> strings) {  
int count = 0;

for(String string: strings) {

if(string.length() == 3) {

count++;

}

}

return count;

}

private static List<String> deleteEmptyStringsUsingJava7(List<String> strings) {  
List<String> filteredList = new ArrayList<String>();

for(String string: strings) {

if(!string.isEmpty()) {

filteredList.add(string);

}

}

return filteredList;

}

private static String getMergedStringUsingJava7(List<String> strings, String separator) {  
StringBuilder stringBuilder = new StringBuilder();

for(String string: strings) {

if(!string.isEmpty()) {

stringBuilder.append(string);

stringBuilder.append(separator);

}

}

String mergedString = stringBuilder.toString();

return mergedString.substring(0, mergedString.length()-2);

}

private static List<Integer> getSquares(List<Integer> numbers) {  
List<Integer> squaresList = new ArrayList<Integer>();

for(Integer number: numbers) {

Integer square = new Integer(number.intValue() \* number.intValue());

if(!squaresList.contains(square)) {

squaresList.add(square);

}

}

return squaresList;

}

private static int getMax(List<Integer> numbers) {  
int max = numbers.get(0);

for(int i = 1;i < numbers.size();i++) {

Integer number = numbers.get(i);

if(number.intValue() > max) {

max = number.intValue();

}

}

return max;

}

private static int getMin(List<Integer> numbers) {  
int min = numbers.get(0);

for(int i= 1;i < numbers.size();i++) {

Integer number = numbers.get(i);

if(number.intValue() < min) {

min = number.intValue();

}

}

return min;

}

private static int getSum(List numbers) {  
int sum = (int)(numbers.get(0));

for(int i = 1;i < numbers.size();i++) {

sum += (int)numbers.get(i);

}

return sum;

}

private static int getAverage(List<Integer> numbers) {  
return getSum(numbers) / numbers.size();  
}  
}

**Answer:**

Using Java 7:

List: [abc, , bc, efg, abcd, , jkl]

Empty Strings: 2

Strings of length 3: 2

Filtered List: [abc, bc, efg, abcd, jkl]

Merged String: abc, bc, efg, abcd, jkl

Squares List: [9, 4, 49, 25]

List: [1, 2, 13, 4, 15, 6, 17, 8, 19]

Highest number in List : 19

Lowest number in List : 1

Sum of all numbers : 85

Average of all numbers : 9

Random Numbers:

[10 random integers - will vary each run]

Using Java 8:

List: [abc, , bc, efg, abcd, , jkl]

Empty Strings: 2

Strings of length 3: 2

Filtered List: [abc, bc, efg, abcd, jkl]

Merged String: abc, bc, efg, abcd, jkl

Squares List: [9, 4, 49, 25]

List: [1, 2, 13, 4, 15, 6, 17, 8, 19]

Highest number in List : 19

Lowest number in List : 1

Sum of all numbers : 85

Average of all numbers : 9.444444444444445

Random Numbers:

[10 sorted random integers - will vary each run]

Empty Strings: 2

**Explanation:**

**Java 7 Traditional Approach:**

* Uses explicit loops, manual counting, and ArrayList manipulation
* More verbose code with multiple helper methods
* Average calculation uses integer division (85/9 = 9)

**Java 8 Stream API Approach:**

* Uses functional programming with method chaining
* **filter()** - filters elements based on condition
* **count()** - counts filtered elements
* **collect()** - collects results into collections
* **Collectors.toList()** - converts stream to List
* **Collectors.joining()** - joins strings with delimiter
* **map()** - transforms each element (i -> i\*i)
* **distinct()** - removes duplicates
* **IntSummaryStatistics** - provides comprehensive statistics
* **limit()** - limits stream to specified number of elements
* **sorted()** - sorts the stream
* **forEach()** - performs action on each element
* **Method Reference (::)** - System.out::println
* **parallelStream()** - enables parallel processing

**Key Differences:**

1. **Code Conciseness:** Stream API reduces boilerplate code significantly
2. **Readability:** Functional approach is more readable and expressive
3. **Performance:** Parallel streams can improve performance for large datasets
4. **Immutability:** Streams don't modify original collections
5. **Lazy Evaluation:** Operations are executed only when terminal operation is called

**Stream Operations:**

* **Intermediate Operations:** filter, map, distinct, sorted, limit (return Stream)
* **Terminal Operations:** count, collect, forEach, summaryStatistics (return result)