Day 9 - 13th June 2025

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

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 {

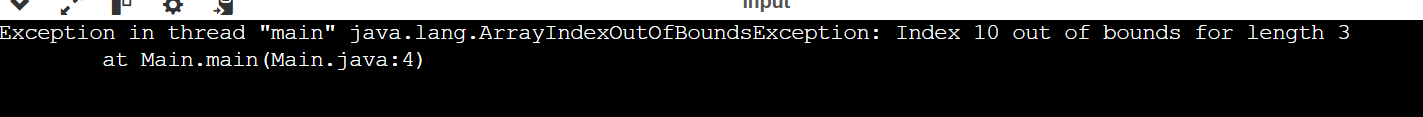
public static void main(String[] args) {

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

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

}

}



**Task 2**

public class Main {

public static void main(String[] args) {

try {

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

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

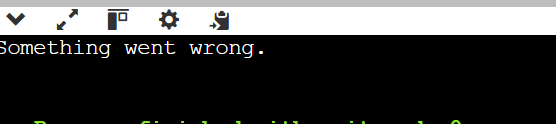
} catch (Exception e) {

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

}

}

}



**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.");

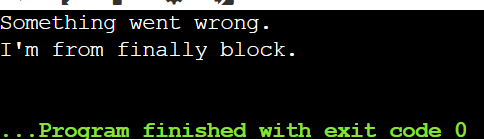
} finally {

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

}

}

}



**Task 4**

public class Main {

public static void main(String[] args) {

try {

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

System.out.println(myNumbers[2]);

System.out.println(myNumbers[3]);

} catch (Exception e) {

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

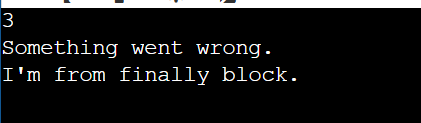
} finally {

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

}

}

}



**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;

int c = 1/b;

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

}

catch (ArrayIndexOutOfBoundsException e) {

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

}catch (Exception e) {

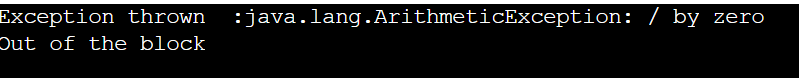
System.out.println("Exception thrown :" + e);

}

System.out.println("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);

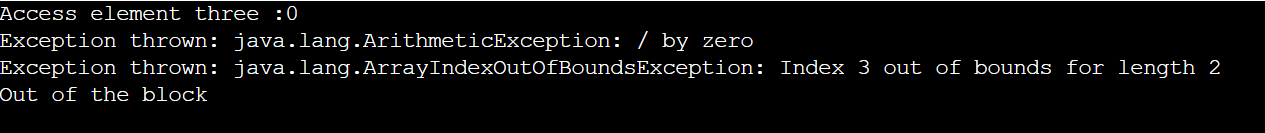
}

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

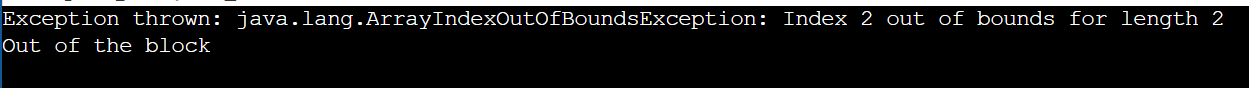
}

}

Output a[0] & a[1]



Output a[2]



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:

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….

}

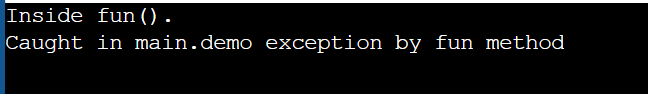
catch (IllegalAccessException e) {

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

}

}

}



Task 8

// User-defined exception class

class MyException extends Exception {

public MyException(String message) {

super(message); // Call parent class constructor with message

}

}

public class task008 {

// Method that throws custom exception based on condition

static void checkAge(int age) throws MyException {

if (age < 18) {

throw new MyException("Age is below 18 – Not eligible to vote.");

} else {

System.out.println("You are eligible to vote!");

}

}

public static void main(String[] args) {

try {

checkAge(16); // This will throw the custom exception

} catch (MyException e) {

System.out.println("Caught custom exception:");

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

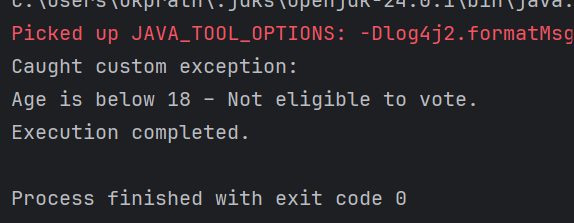
} finally {

System.out.println("Execution completed.");

}

}

}



**Add on Task 1**

class Geeks {

public static void main(String[] args)

{

int n = 10;

int m = 0;

try {

// Code that may throw an exception

int ans = n / m;

System.out.println("Answer: " + ans);

}

catch (ArithmeticException e) {

// Handling the exception

System.out.println(

"Error: Division by zero is not allowed!");

}

finally {

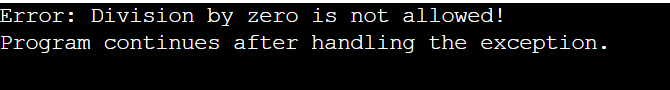
System.out.println(

"Program continues after handling the exception.");

}

}

}



**Add on Task 2** 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) {

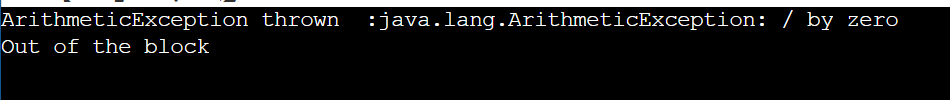
System.out.println("Exception thrown :" + e);

}

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

}

}



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);

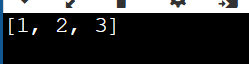
a.add(3);

// Printing ArrayList

System.out.println(a);

}

}



Task 10

Find the output of the be code snippet..

// Addition, Deletion and Updation of Element

import java.util.\*;

class task010 {

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");

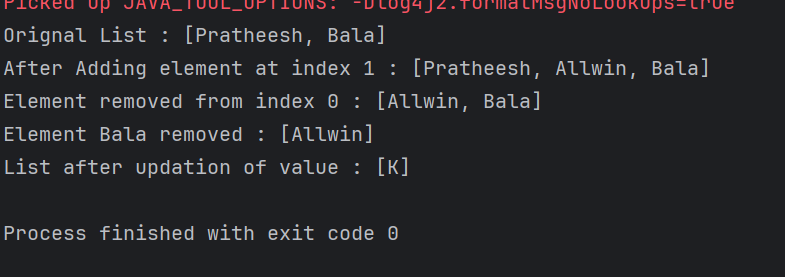
System.out.println("Element Prasunamba removed : "+ al);

al.set(0, "K");

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

}

}



Task 11

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);

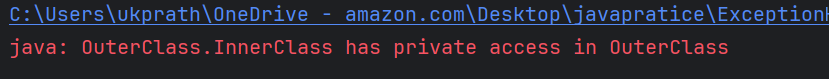
}

}

**Output : 15**

Task 12:

Make the innerclass private and check the output or error?



Task 13

Use the above code Task 011 and make the inner class static … see the output and explain..

class OuterClasses {

int x = 10;

static class InnerClass {

int y = 5;

}

}

public class task013 {

public static void main(String[] args) {

OuterClasses myOuter = new OuterClasses();

OuterClasses.InnerClass myInner = new OuterClasses.InnerClass();

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

}

}

Output : 15

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 {

public static void main(String[] args) {

OuterClass myOuter = new OuterClass();

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

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

}

}

Output : 50

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);

}

}

Output : 15

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

Task 16:

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)

// Abstract base class

abstract class Person {

private String name;

private int age;

public Person() {}

public Person(String name, int age) {

this.name = name;

this.age = age;

}

// Getters & 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; }

// toString

public String toString() {

return "Name: " + name + ", Age: " + age;

}

}

// Customer class

class Customer extends Person {

private String customerId;

private String address;

public Customer(String name, int age, String customerId, String address) {

super(name, age);

this.customerId = customerId;

this.address = address;

}

public String getCustomerId() { return customerId; }

public void setCustomerId(String customerId) { this.customerId = customerId; }

public String getAddress() { return address; }

public void setAddress(String address) { this.address = address; }

@Override

public String toString() {

return super.toString() + ", Customer ID: " + customerId + ", Address: " + address;

}

}

// Employee class

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;

}

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 super.toString() + ", Employee ID: " + empId + ", Salary: $" + salary;

}

}

// Manager class extends Employee

class Manager extends Employee {

private String dept;

private int teamSize;

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

super(name, age, empId, salary);

this.dept = dept;

this.teamSize = teamSize;

}

public String getDept() { return dept; }

public void setDept(String dept) { this.dept = dept; }

public int getTeamSize() { return teamSize; }

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

@Override

public String toString() {

return super.toString() + ", Department: " + dept + ", Team Size: " + teamSize;

}

}

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

Task 17:

What are the features of Java 8?

* **Lambda Expressions**: Concise functional code using ->.
* **Functional Interfaces**: Single-method interfaces.
* **Introduced and Improved APIs:**
  1. **Stream API**: Efficient Data Manipulation.
  2. **Date/Time API**: Robust Date and Time Handling.
  3. **Collection API Improvements**: Enhanced Methods for Collections (e.g., removeIf, replaceAll).
  4. **Concurrency API Improvements**: New classes for parallel processing (e.g., CompletableFuture).
* **Optional Class**: Handle null values safely.
* **forEach() Method in Iterable Interface**: Executes an action for each element in a Collection.
* **Default Methods**: Evolve interfaces without breaking compatibility.
* **Static Methods**: Allows adding methods with default implementations to interfaces.
* **Method References**: Refer to methods easily.

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

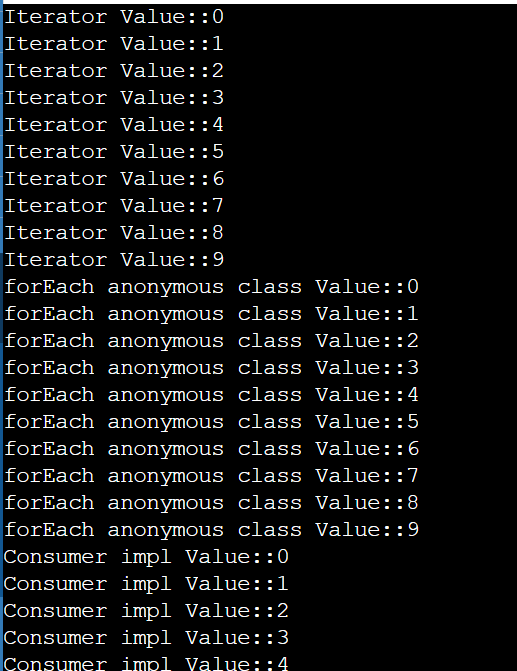
class MyConsumer implements Consumer<Integer>{

public void accept(Integer t) {

System.out.println("Consumer impl Value::"+t);

}

}



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);

}

}

