Day 9 - 11th June 2025

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

Exceptions:

Types Exceptions:

Doc 18 exception Handling in Docs tro Study

Task 1:

What do you understand by exceptions?

An exception is an unwanted or unexpected event that occurs during program execution

It disrupts the normal flow of the program's instructions. It’s a mechanism to handle runtime errors.

Task 2:

What are the categories of Exceptions do we have in Java? What are they?

Checked Exceptions:

* Checked at compile time
* Must be explicitly handled in code
* Examples: IOException, SQLException
* Typically external to the program (file handling, database operations)

Unchecked Exceptions (Runtime):

* Not checked at compile time
* Examples: NullPointerException, ArithmeticException
* Usually due to programming errors

Errors:

* Serious problems
* Generally not handled by programs
* Examples: OutOfMemoryError, StackOverflowError

Task 3:

Can you try the below code snippet and let me know which kind of exception is this

// Java program to demonstrates handling

// the exception using try-catch block

import java.io.\*;

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!");

        }

catch (ArithmeticException e) {

            // Handling the exception

            System.out.println(

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

        }

        finally {

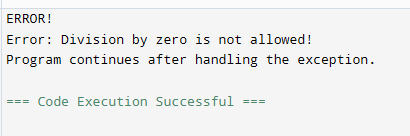
            System.out.println(

                "Program continues after handling the exception.");

        }

    }

}



This code demonstrates an example of an ArithmeticException, which is an Unchecked (Runtime) Exception. The code attempts to divide 10 by 0 (n/m where n=10 and m=0)Division by zero is mathematically undefined, This operation throws an ArithmeticException.

Try block throws and error and catch block handled the exception by printing the error message and finally block Executes regardless of exception occurrence, Provides confirmation that program continues.

Task 4:

List of checked and unchecked exceptions.

**Checked Exceptions:**

* IOException (Input/Output exceptions like file reading/writing errors)
* SQLException (Database related exceptions)
* FileNotFoundException (File not found exception)
* ClassNotFoundException (Class not found exception)
* CloneNotSupportedException
* InterruptedException
* ReflectiveOperationException
* IllegalAccessException
* InstantiationException
* NoSuchFieldException
* NoSuchMethodException

**Unchecked Exceptions:**

* NullPointerException (Accessing a null reference)
* ArrayIndexOutOfBoundsException (Accessing an array with an invalid index)
* ArithmeticException (Arithmetic errors like division by zero)
* ClassCastException (Invalid type casting)
* IllegalArgumentException (Illegal argument passed to a method)
* IllegalStateException (Method called in an inappropriate state)
* IndexOutOfBoundsException
* NegativeArraySizeException
* NumberFormatException
* SecurityException
* StringIndexOutOfBoundsException
* TypeNotPresentException
* UnsupportedOperationException
* EnumConstantNotPresentException
* IllegalMonitorStateException
* IllegalThreadStateExceptio

Task 4:

List of checked and unchecked exceptions.

10.45 to 10.49

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) {

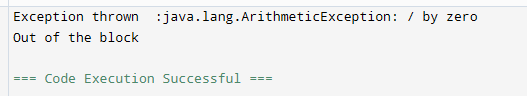
          System.out.println("Exception thrown  :" + e);

      }

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

   }

}



In this code, there are two exceptions: ArithmeticException (division by zero: 1/b where b=0) and ArrayIndexOutOfBoundsException (accessing a[3] in an array of size 2). The ArithmeticException occurs first at int c = 1/b; and is caught by the general Exception catch block, so the code never reaches the array access that would cause ArrayIndexOutOfBoundsException. Therefore, the output shows the ArithmeticException message followed by "Out of the block".

The order of exception occurrence and handling determines the output, demonstrating how only the first exception is caught and handled.

Task 6:

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");

   }

}

When the code executes int c = 1/b; (where b=0), it immediately throws an ArithmeticException

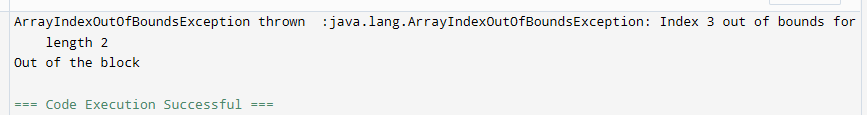
The code execution stops at this point and jumps to the first matching catch block (ArithmeticException)

The line System.out.println("Access element three :" + a[3]); is never executed because the exception occurred before reaching this line.

After executing the catch block, the program continues and prints "Out of the block"

The code never reaches the ArrayIndexOutOfBoundsException or the general Exception catch blocks because the ArithmeticException was caught and handled by its specific catch block first.

After changing b value to 2



Task 7:

In the below code we are having use multiple catch in a single statement: find the output and try to understand the code..

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 | ArithmeticException e) {

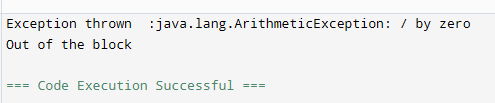
         System.out.println("Exception thrown  :" + e);

      }

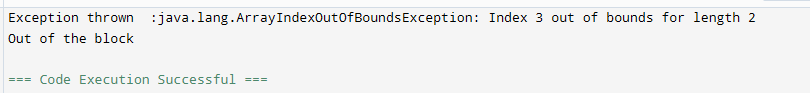
      System.out.println("Out of the block");

   }

}



Same code after changing b value to 2



Task 008:

public class ExcepTest {

   public static void main(String args[]) {

      try {

         int a[] = new int[2];

         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) {

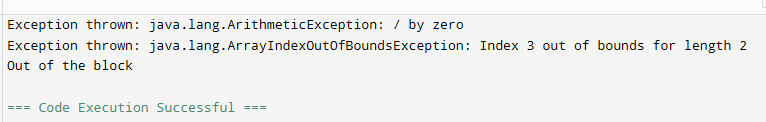
         System.out.println("Exception thrown: " + e);

      }

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

   }

}



Task 009

// Demonstrating how to throw an exception

class MyClass {

    static void fun() throws IllegalAccessException

    {

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

        throw new IllegalAccessException("demo");

    }

    public static void main(String args[])

    {

        try {

            fun();

//method2();   → arrayindex…

//Method3()  —> file not found….

        }

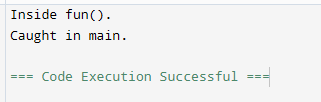
        catch (IllegalAccessException e) {

            System.out.println("Caught in main.");

        }

    }

}



Task 011

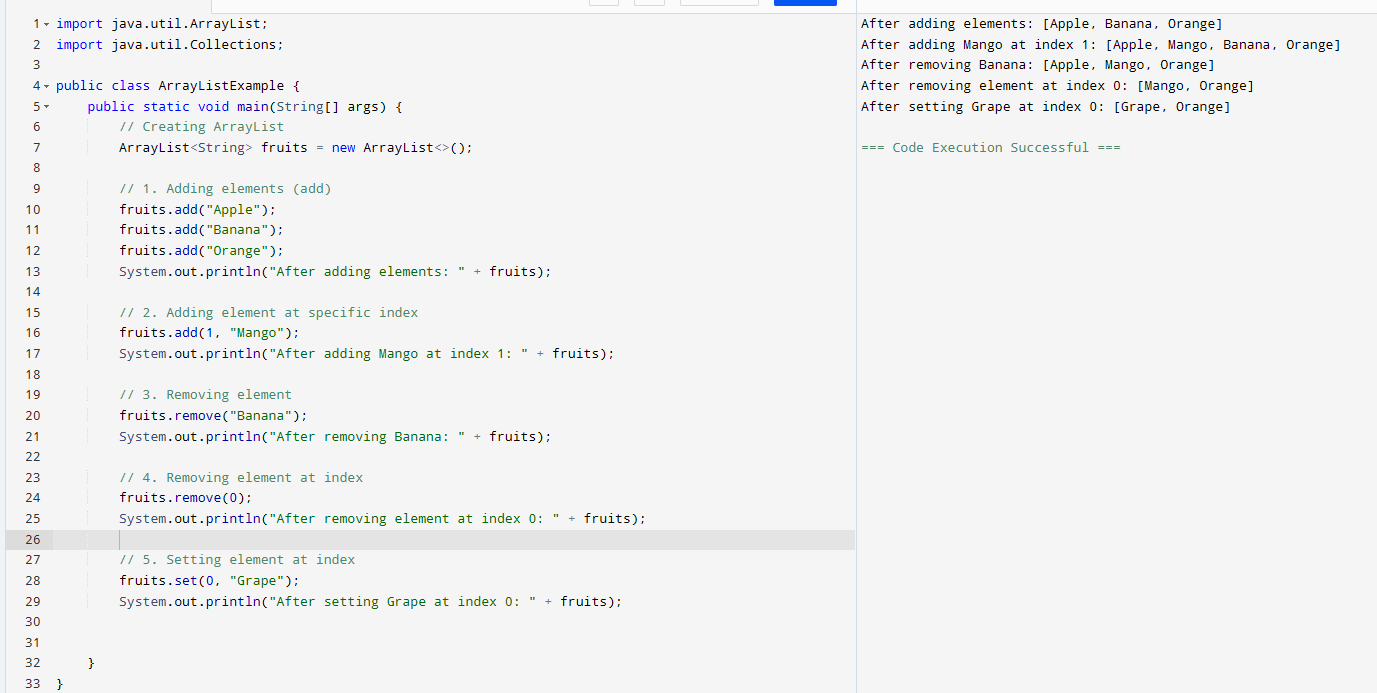
Wap to create an array list to display 10 elements using for loop.



Task 012

Find the output of the be code snippet..

// Addition, Deletion and Updation of Element



Task 013

Run the code and see hope the user defined exception works..

User defined Exception:

// A Class that represents user-defined exception

class Customer extends Exception {// predefined class Exception

    public Customer(String m) { // constructor with parameters

        super(m); // parent class constructor

    }

}

// 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) {

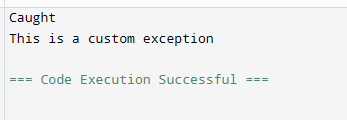
            System.out.println("Caught");

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

        }

    }

}



Task 014

Inheritance

Classes customer, employee, Manager  … 2 variables in each class

Driver class – display all the variables… toString();

Hint : use getter and setters..

public class InheritanceDemo {

// Person class (Parent class)

static class Person {

private String name;

private int age;

// Constructor

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

}

}

// Employee class (inherits from Person)

static class Employee extends Person {

private String department;

private double salary;

// Constructor

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

super(name, age);

this.department = department;

this.salary = salary;

}

// Getters and Setters

public String getDepartment() {

return department;

}

public void setDepartment(String department) {

this.department = department;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

@Override

public String toString() {

return super.toString() + ", Department: " + department + ", Salary: $" + salary;

}

}

// Manager class (inherits from Employee)

static class Manager extends Employee {

private int teamSize;

private String projectName;

// Constructor

public Manager(String name, int age, String department, double salary,

int teamSize, String projectName) {

super(name, age, department, salary);

this.teamSize = teamSize;

this.projectName = projectName;

}

// Getters and Setters

public int getTeamSize() {

return teamSize;

}

public void setTeamSize(int teamSize) {

this.teamSize = teamSize;

}

public String getProjectName() {

return projectName;

}

public void setProjectName(String projectName) {

this.projectName = projectName;

}

@Override

public String toString() {

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

", Project Name: " + projectName;

}

}

// Main method

public static void main(String[] args) {

// Create objects of each class

Person person = new Person("John Doe", 30);

Employee employee = new Employee("Jane Smith", 25, "IT", 50000.0);

Manager manager = new Manager("Mike Johnson", 35, "Development", 80000.0,

10, "Project Alpha");

// Display information using toString()

System.out.println("Person Details:");

System.out.println(person);

System.out.println("\nEmployee Details:");

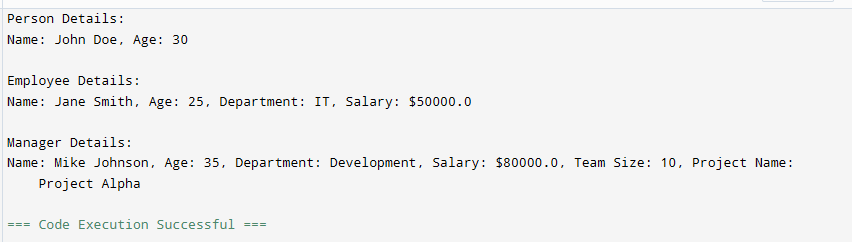
System.out.println(employee);

System.out.println("\nManager Details:");

System.out.println(manager);

}

}



Inner classes

Task 015

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) {

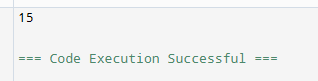
    OuterClass myOuter = new OuterClass();

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

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

  }

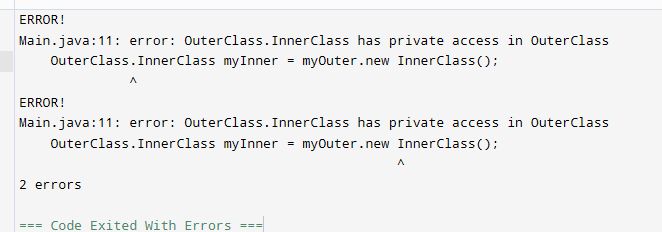
}



Task 016

Use the above code and make the inner class as private and see the output..

Ex: Private  class InnerClass {



Task 017

Use the above code Task 015 and make the inner class static … see the output and explain..

Ex: static class InnerClass {



Task 018

Use the above code Task 015 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());

