Java Exception Handling Questions & Solutions

# Question 1: Null Pointer Exception Handling

Write a method getStringLength that takes a string and returns its length. Handle the NullPointerException if the input string is null and return an appropriate message.

Code:

public static Object getStringLength(String str) {  
 try {  
 return str.length();  
 } catch (NullPointerException e) {  
 return "Error: Null value passed";  
 }  
}

Sample Output:

getStringLength("Hello") → 5  
getStringLength(null) → Error: Null value passed

# Question 2: Arithmetic and Array Exception Handling

Write a method calculateAverage that takes an array of integers and returns the average. Handle exceptions for division by zero (empty array) and array access errors (index out of bounds).

Code:

public static Object calculateAverage(int[] arr) {  
 try {  
 int sum = 0;  
 for (int i = 0; i < arr.length; i++) {  
 sum += arr[i];  
 }  
 return sum / arr.length;  
 } catch (ArithmeticException e) {  
 return "Error: Division by zero";  
 } catch (ArrayIndexOutOfBoundsException e) {  
 return "Error: Index out of bounds";  
 }  
}

Sample Output:

calculateAverage(new int[]{10, 20, 30}) → 20  
calculateAverage(new int[]{}) → Error: Division by zero

# Question 3: Custom Exception for Age Validation

Write a method checkAge that takes an integer representing a person's age and throws a custom InvalidAgeException if the age is less than 18. Otherwise, it returns a message saying the person is eligible.

Code:

class InvalidAgeException extends Exception {  
 public InvalidAgeException(String message) {  
 super(message);  
 }  
}  
  
public static String checkAge(int age) throws InvalidAgeException {  
 if (age < 18) {  
 throw new InvalidAgeException("InvalidAgeException: Age must be 18 or above");  
 }  
 return "Eligible";  
}

Sample Output:

checkAge(20) → Eligible  
checkAge(15) → InvalidAgeException: Age must be 18 or above

# Question 4: Input Validation and Exception Handling

Write a method parseAndMultiply that takes a string input, converts it to an integer, and multiplies it by a constant (e.g., 10). Handle exceptions for invalid inputs (e.g., non-numeric values) and return an appropriate error message.

Code:

public static Object parseAndMultiply(String input) {  
 try {  
 int num = Integer.parseInt(input);  
 return num \* 10;  
 } catch (NumberFormatException e) {  
 return "Error: Invalid input";  
 }  
}

Sample Output:

parseAndMultiply("5") → 50  
parseAndMultiply("abc") → Error: Invalid input

# Question 5: Nested Try-Catch Blocks

Write a method nestedExceptionHandling that contains nested try-catch blocks to handle multiple exceptions in a single operation: dividing two numbers and accessing an array element at a specific index.

Code:

public static void nestedExceptionHandling(String a, String b, int[] arr, int index) {  
 try {  
 try {  
 int num1 = Integer.parseInt(a);  
 int num2 = Integer.parseInt(b);  
 int result = num1 / num2;  
 System.out.print(result + " and ");  
 } catch (ArithmeticException e) {  
 System.out.println("Error: Division by zero");  
 return;  
 }  
  
 try {  
 System.out.println("element: " + arr[index]);  
 } catch (ArrayIndexOutOfBoundsException e) {  
 System.out.println("Error: Index out of bounds");  
 }  
  
 } catch (NumberFormatException e) {  
 System.out.println("Error: Invalid input format");  
 }  
}

Sample Output:

nestedExceptionHandling("20", "5", new int[]{1,2,3,4}, 1) → 4 and element: 2  
nestedExceptionHandling("10", "0", new int[]{1,2,3}, 5) → Error: Division by zero