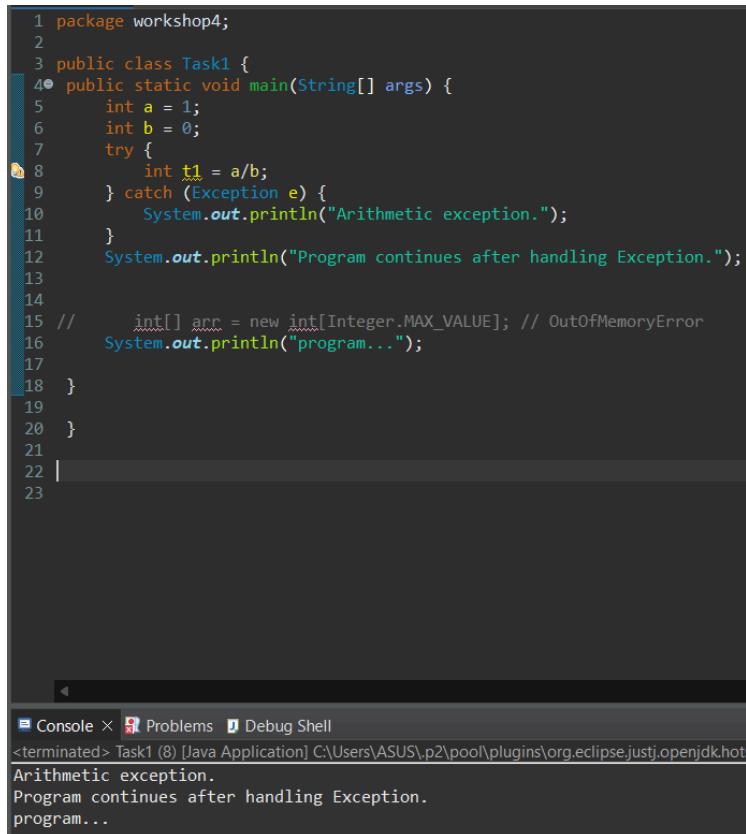


# Workshop 4

## Exception Handling

1. Write a Java program to demonstrate the difference between an Exception and an Error.



The screenshot shows the Eclipse IDE interface. The top half displays a Java code editor with the following code:

```
1 package workshop4;
2
3 public class Task1 {
4     public static void main(String[] args) {
5         int a = 1;
6         int b = 0;
7         try {
8             int t1 = a/b;
9         } catch (Exception e) {
10             System.out.println("Arithmetic exception.");
11         }
12         System.out.println("Program continues after handling Exception.");
13
14
15 //     int[] arr = new int[Integer.MAX_VALUE]; // OutOfMemoryError
16 //     System.out.println("program...");
17
18 }
19
20 }
21
22 |
23
```

The bottom half shows the Eclipse Console view with the following output:

```
Console × Problems Debug Shell
<terminated> Task1 (8) [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjdk.hots
Arithmetic exception.
Program continues after handling Exception.
program...
```

2. Create a program that demonstrates ArithmeticException by dividing a number by zero and handle it using try-catch.

The screenshot shows the Eclipse IDE interface. On the left is the Java code editor with the following content:

```
1 package workshop4;
2
3 public class Task2 {
4     public static void main(String[] args) {
5         int a = 10;
6         int b = 0;
7         try {
8             int c = a/b;
9             System.out.println("Result: " +c);
10        } catch (ArithmaticException e) {
11            System.out.println("Number cannot divide by zero.");
12        } finally {
13            System.out.println("Finally block executed.");
14        }
15    }
16 }
```

On the right is the Eclipse Console window, which displays the output of the program's execution:

```
<terminated> Task2 (5) [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.open
Number cannot divide by zero.
Finally block executed.
```

3. Write a Java program to show the complete Exception Hierarchy using comments and examples.

```

1 package workshop4;
2
3 import java.io.FileReader;[]
4
5 public class Task3 {
6
7     public static void main(String[] args) {
8
9         //RuntimeException (Unchecked)
10        //ArithmeticeException
11        try {
12            int a = 10 / 0;
13        } catch (ArithmeticeException e) {
14            System.out.println("ArithmeticeException");
15        }
16
17        //NullPointerException
18        try {
19            String name = null;
20            System.out.println(name.length());
21        } catch (NullPointerException e) {
22            System.out.println("NullPointerException");
23        }
24
25        // ArrayIndexOutOfBoundsException
26        try {
27            int[] arr = {1,2,3,4,5,6};
28            System.out.println(arr[8]);
29        } catch (ArrayIndexOutOfBoundsException e) {
30            System.out.println("ArrayIndexOutOfBoundsException");
31        }
32
33        //Checked Exception
34        try {
35            FileReader f = new FileReader("test.txt"); // FileNotFoundException
36        } catch (FileNotFoundException e) {
37            System.out.println("File not found");
38        }
39    }
40
41
42}
43
44    // Causes StackOverflowError (Error)
45 //    static void infiniteMethod() {
46 //        infiniteMethod();
47    }
48
49

```

The screenshot shows an IDE interface with a code editor and a terminal window. The code editor contains the Java code from the previous block. The terminal window, titled 'Console', displays the following output:

```

Console × Problems Debug Shell
<terminated> Task3 (5) [Java Application] C:\Users\ASUS\p2\pool\p
ArithmeticeException
NullPointerException
ArrayIndexOutOfBoundsException
File not found

```

4. Develop a program to handle multiple types of exceptions (ArithmeticeException, ArrayIndexOutOfBoundsException, and NullPointerException) in a single try block.

```
1 package workshop4;
2
3 public class Task4 {
4
5     public static void main(String[] args) {
6         int a = 10;
7         int b = 0;
8         String name = null;
9         int[] numbers = {1, 2, 3};
10
11     try {
12         int c = a/b;
13         System.out.println("Result: "+c);
14
15         System.out.println(numbers[5]);
16
17         System.out.println(name.length());
18     }
19     catch (ArithmaticException e) {
20         System.out.println("ArithmaticException occurred: Division by zero");
21     }
22     catch (ArrayIndexOutOfBoundsException e) {
23         System.out.println("ArrayIndexOutOfBoundsException occurred: Invalid index");
24     }
25     catch (NullPointerException e) {
26         System.out.println("NullPointerException occurred: Null object access");
27     }
28     finally {
29         System.out.println("program.....");
30     }
31     }
32     }
33 }
34 }
35 }
```

Console X Problems Debug Shell

<terminated> Task4 (5) [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64

ArithmaticException occurred: Division by zero

program.....

5. Write a program that demonstrates checked and unchecked exceptions with suitable examples.

```
1 package workshop4;
2
3 import java.io.FileReader;[]
4
5
6 public class Task5 {
7
8     public static void main(String[] args) {
9
10         // UNCHECKED EXCEPTION
11
12         try {
13             int a = 10 / 0;    // ArithmeticException
14         } catch (ArithmetiException e) {
15             System.out.println("Unchecked Exception caught: Division by zero");
16         }
17
18         // | CHECKED EXCEPTION
19         try {
20             FileReader fr = new FileReader("test.txt"); // FileNotFoundException
21         } catch (FileNotFoundException e) {
22             System.out.println("Checked Exception caught: File not found");
23         }
24
25         System.out.println("Program executed successfully");
26     }
27 }
28
29
```

The screenshot shows the Eclipse IDE interface. The top part displays the Java code for 'Task5'. The bottom part shows the 'Console' tab with the following output:

```
<terminated> Task5 (5) [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full
Unchecked Exception caught: Division by zero
Checked Exception caught: File not found
Program executed successfully
```

6. Create a program that reads an integer from the user and throws a custom exception if the number is negative.

```
1 package workshop4;
2
3 import java.util.Scanner;
4
5 class NegativeNumberException extends Exception {
6     public NegativeNumberException(String message) {
7         super(message);
8     }
9 }
10
11 public class Task6 {
12
13     public static void main(String[] args) {
14
15         Scanner sc = new Scanner(System.in);
16
17         try {
18             System.out.print("Enter an integer: ");
19             int num = sc.nextInt();
20
21             if (num < 0) {
22                 throw new NegativeNumberException("Number cannot be negative");
23             }
24
25         }
26         catch (NegativeNumberException e) {
27             System.out.println("Custom Exception caught: " + e.getMessage());
28         }
29         finally {
30             sc.close();
31         }
32     }
33 }
```

Console X Problems Debug Shell  
<terminated> Task6 (4) [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.jdt.openjdk.hotspot.jre.full.  
Enter an integer: -1  
Custom Exception caught: Number cannot be negative

7. Write a Java program using multiple catch blocks to handle different exceptions separately.

```
1 package workshop4;
2
3 public class Task7 {
4
5     public static void main(String[] args) {
6
7         int a = 10;
8         int b = 0;
9         int arr[] = {1, 2, 3, 4};
10
11         try {
12             System.out.println(arr[4]);
13         }
14         catch (ArrayIndexOutOfBoundsException aie) {
15             System.out.println("Unable to access index.");
16         }
17
18         try {
19             int result = a / b;
20             System.out.println("Result is: " + result);
21         } catch (ArithmaticException ae) {
22             System.out.println("Arithmatic Exception: Division by zero.");
23         }
24
25     }
26 }
```

Console X Problems Debug Shell  
<terminated> Task7 (3) [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.jdt.openjdk.hotspot.jre.full.  
Unable to access index.  
Arithmatic Exception: Division by zero.

8. Create a program demonstrating nested try-catch blocks.

The screenshot shows a Java code editor and a terminal window. The code editor displays a Java file named Task8.java with the following content:

```
1 package workshop4;
2
3 public class Task8 {
4     public static void main(String[] args) {
5         try {
6             try {
7                 int a = 10 / 0;
8             }
9             catch (ArithmaticException e) {
10                 System.out.println("Inner catch");
11             }
12         }
13         catch (Exception e) {
14             System.out.println("Outer catch");
15         }
16     }
17 }
18
```

The terminal window below the code editor shows the output of the program:

```
Console × Problems Debug Shell
<terminated> Task8 (3) [Java Application] C:\Users\ASUS\p2\pool\plugins
Inner catch
```

9. Write a Java program that uses the throw keyword to manually throw an exception when invalid input is entered.

```
1 package workshop4;
2
3 public class Task9 {
4
5     public static void main(String[] args) {
6         int age = -2;
7
8         try {
9             if (age < 0) {
10                 throw new IllegalArgumentException("Age cannot be negative!");
11             }
12             else {
13                 System.out.println("Your age is: " + age);
14             }
15         } catch (IllegalArgumentException e) {
16             System.out.println("Exception caught: " + e.getMessage());
17         }
18
19         System.out.println("Program continues...");
20     }
21 }
22
23
```

The screenshot shows the Eclipse IDE interface. The code editor contains the Java code for Task9. The console output window shows the program's execution. It prints 'Exception caught: Age cannot be negative!' followed by 'Program continues...'. The status bar at the bottom indicates the application is terminated.

10. Develop a Java program that uses the throws keyword in a method and handle the exception in the calling method.

```
1 package workshop4;
2
3 public class Task10 {
4
5     static int divide(int a, int b) throws ArithmeticException {
6         return a / b;
7     }
8
9     public static void main(String[] args) {
10
11         try {
12             int result = divide(10, 0);
13             System.out.println("Result: " + result);
14         } catch (ArithmeticException e) {
15             System.out.println("Exception caught: Cannot divide by zero!");
16         }
17
18         System.out.println("Program continues...");
19     }
20 }
21
22
```

The screenshot shows the Eclipse IDE interface. The code editor contains the Java code for Task10. The console output window shows the program's execution. It prints 'Exception caught: Cannot divide by zero!' followed by 'Program continues...'. The status bar at the bottom indicates the application is terminated.

11. Write a Java program to demonstrate the use of the finally block in exception handling.

The screenshot shows the Eclipse IDE interface. On the left is the code editor with Task11.java. On the right is the 'Console' view showing the application's output.

```
1 package workshop4;
2
3 public class Task11 {
4     public static void main(String[] args) {
5         try {
6             int a = 10 / 0;
7         } catch (ArithmaticException e) {
8             System.out.println("Exception");
9         } finally {
10            System.out.println("Finally block always executes");
11        }
12    }
13 }
14
```

Console output:

```
<terminated> Task11 (2) [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjpa.core\lib\openjpa-2.3.2\lib\openjpa-pool.jar
Exception
Finally block always executes
```

12. Create a program to show that the finally block executes even when return is used inside a try or catch block.

The screenshot shows the Eclipse IDE interface. On the left is the code editor with Task12.java. On the right is the 'Console' view showing the application's output.

```
1 package workshop4;
2
3 public class Task12{
4     static int test() {
5         try {
6             return 10;
7         } finally {
8             System.out.println("Finally executed even with return");
9         }
10    }
11
12    public static void main(String[] args) {
13        System.out.println("Returned value: " + test());
14    }
15 }
```

Console output:

```
<terminated> Task12 (2) [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjpa.core\lib\openjpa-2.3.2\lib\openjpa-pool.jar
Finally executed even with return
Returned value: 10
```

13. Write a program to demonstrate try-with-resources by reading data from a file safely.

```
1 package workshop4;
2
3 import java.io.BufferedReader;
4
5 public class Task13 {
6     public static void main(String[] args) {
7         try (BufferedReader br = new BufferedReader(new FileReader("test.txt"))) {
8             System.out.println(br.readLine());
9         } catch (Exception e) {
10             System.out.println("Error reading file");
11         }
12     }
13 }
14
15
16
```

Console X Problems Debug Shell  
<terminated> Task13 (2) [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64 Error reading file

14. Develop a Java program to handle multiple exceptions using multi-catch syntax (catch(Exception1 | Exception2 e)).

```
1 package workshop4;
2
3 public class Task14 {
4     public static void main(String[] args) {
5         try {
6             int[] arr = {1,2,3,4,5};
7             System.out.println(arr[5]);
8             int result = 10 / 0;
9         } catch (ArithmaticException | ArrayIndexOutOfBoundsException e) {
10             System.out.println("Exception caught: " + e);
11         }
12     }
13 }
14
15
```

Console X Problems Debug Shell  
<terminated> Task14 (2) [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64 Exception caught: java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5

15. Write a program that shows how an exception is propagated from one method to another using throws.

```
1 package workshop4;
2
3 public class Task15 {
4     static void method1() throws ArithmeticException {
5         int a = 10 / 0;
6     }
7
8     static void method2() throws ArithmeticException {
9         method1();
10    }
11
12    public static void main(String[] args) {
13        try {
14            method2();
15        } catch (ArithmeticException e) {
16            System.out.println("Exception caught in main: " + e);
17        }
18    }
19 }
```

Console × Problems Debug Shell  
<terminated> Task15 (1) [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjdk.hots  
Exception caught in main: java.lang.ArithmetricException: / by zero

16. Create a program to demonstrate the concept of rethrowing an exception.

```
1 package workshop4;
2
3 public class Task16 {
4     public static void main(String[] args) {
5         try {
6             try {
7                 int a = 10 / 0;
8             } catch (ArithmetricException e) {
9                 System.out.println("Inner catch: " + e);
10                throw e;
11            }
12        } catch (ArithmetricException e) {
13            System.out.println("Outer catch: Exception rethrown: " + e);
14        }
15    }
16 }
```

Console × Problems Debug Shell  
<terminated> Task16 (1) [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjdk.hots  
Inner catch: java.lang.ArithmetricException: / by zero  
Outer catch: Exception rethrown: java.lang.ArithmetricException: / by zero

17. Write a Java program to demonstrate the difference between checked and unchecked exceptions by creating your own custom checked exception.

```
1 package workshop4;
2
3 class MyCheckedException extends Exception {
4     MyCheckedException(String msg) { super(msg); }
5 }
6
7 public class Task17 {
8     static void test() throws MyCheckedException {
9         throw new MyCheckedException("This is a custom checked exception");
10    }
11
12    public static void main(String[] args) {
13        try {
14            test();
15        } catch (MyCheckedException e) {
16            System.out.println("Exception caught: " + e.getMessage());
17        }
18    }
19 }
```

Console × Problems Debug Shell  
<terminated> Task17 (1) [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.fu  
Exception caught: This is a custom checked exception

18. Create a class InvalidAgeException and write a program that throws this exception if a person's age is below 18.

```
1 package workshop4;
2
3 class InvalidAgeException extends Exception {
4     InvalidAgeException(String msg) { super(msg); }
5 }
6
7 public class Task18 {
8     public static void main(String[] args) {
9         int age = 16;
10        try {
11            if (age < 18) throw new InvalidAgeException("Age is below 18");
12        } catch (InvalidAgeException e) {
13            System.out.println("Exception: " + e.getMessage());
14        }
15    }
16 }
17
18
```

Console × Problems Debug Shell  
<terminated> Task18 [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.fu  
Exception: Age is below 18

19. Write a Java program to handle user input errors gracefully using exception handling.

The screenshot shows the Eclipse IDE interface. The code editor contains the following Java code:

```
1 package workshop4;
2
3 import java.util.Scanner;
4
5 public class Task19 {
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8         try {
9             System.out.print("Enter an integer: ");
10            int num = sc.nextInt();
11            System.out.println("You entered: " + num);
12        } catch (Exception e) {
13            System.out.println("Invalid input! Please enter a valid integer.");
14        }
15        sc.close();
16    }
17 }
```

The console output shows the program's execution:

```
Console X Problems Debug Shell
<terminated> Task19 [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64
Enter an integer: s
Invalid input! Please enter a valid integer.
```

20. Develop a Java program to demonstrate exception chaining (new Exception("msg", cause)).

The screenshot shows the Eclipse IDE interface. The code editor contains the following Java code:

```
1 package workshop4;
2
3 public class Task20 {
4     public static void main(String[] args) {
5         try {
6             try {
7                 int a = 10 / 0;
8             } catch (ArithmaticException e) {
9                 throw new Exception("New exception caused by division error", e);
10            }
11        } catch (Exception e) {
12            System.out.println("Caught: " + e);
13            System.out.println("Cause: " + e.getCause());
14        }
15    }
16 }
```

The console output shows the program's execution:

```
Console X Problems Debug Shell
<terminated> Task20 [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64
Caught: java.lang.Exception: New exception caused by division error
Cause: java.lang.ArithmaticException: / by zero
```

21.Create a program to demonstrate how finally is used for closing resources like files or database connections.

The screenshot shows the Eclipse IDE interface. On the left is the code editor with the following Java code:

```
1 package workshop4;
2
3 import java.io.FileReader;
4
5 public class Task21 {
6     public static void main(String[] args) {
7         FileReader fr = null;
8         try {
9             fr = new FileReader("test.txt");
10        } catch (Exception e) {
11            System.out.println(e);
12        } finally {
13            System.out.println("Finally: Closing resources if needed");
14        }
15    }
16 }
17
18
```

On the right is the Console view, which displays the following output:

```
Console × Problems Debug Shell
<terminated> Task21 [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.2
java.io.FileNotFoundException: test.txt (The system cannot find the file specified)
Finally: Closing resources if needed
```

22.Write a program that takes a filename from the user, reads its content, and handles FileNotFoundException.

The screenshot shows the Eclipse IDE interface. On the left is the code editor with the following Java code:

```
1 package workshop4;
2
3 import java.io.FileReader;
4
5 public class Task22 {
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8         System.out.print("Enter filename: ");
9         String filename = sc.nextLine();
10        try {
11            FileReader fr = new FileReader(filename);
12            System.out.println("File opened successfully!");
13            fr.close();
14        } catch (Exception e) {
15            System.out.println("File not found: " + e);
16        }
17        sc.close();
18    }
19 }
20
```

On the right is the Console view, which displays the following output:

```
Console × Problems Debug Shell
<terminated> Task22 [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.2
Enter filename: ssss
File not found: java.io.FileNotFoundException: ssss (The system cannot find the file specified)
```

23. Write a program that demonstrates how to create and handle multiple custom exceptions in a single program.

```
1 package workshop4;
2
3 import java.util.Scanner;
4 class LowBalanceException extends Exception {
5     LowBalanceException(String msg) {
6         super(msg);
7     }
8 }
9 class InvalidPinException extends Exception {
10    InvalidPinException(String msg) {
11        super(msg);
12    }
13 }
14
15 public class Task23{
16
17    public static void main(String[] args) {
18
19        Scanner sc = new Scanner(System.in);
20
21        System.out.print("Enter your account balance: ");
22        int balance = sc.nextInt();
23
24        System.out.print("Enter your PIN: ");
25        int pin = sc.nextInt();
26
27        try {
28            if (balance < 500) {
29                throw new LowBalanceException("Insufficient balance!");
30            }
31        } catch (LowBalanceException e) {
32            System.out.println("Caught Exception: " + e.getMessage());
33        }
34
35
36
37        try {
38            if (pin != 1234) {
39                throw new InvalidPinException("PIN is incorrect!");
40            }
41
42            System.out.println("Transaction successful!");
43        } catch (InvalidPinException e) {
44            System.out.println("Caught Exception: " + e.getMessage());
45        }
46
47
48        System.out.println("Program continues...");
49        sc.close();
50    }
51 }
52
```

Console × Problems Debug Shell  
<terminated> Task23 [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjdk.h  
Caught Exception: Insufficient balance!  
Caught Exception: PIN is incorrect!  
Program continues...

24.Create a Java program to demonstrate how exception handling improves program reliability and flow control.

```
1 package workshop4;
2
3 public class Task24 {
4     public static void main(String[] args) {
5         try {
6             int result = 10 / 0;
7         } catch (Exception e) {
8             System.out.println("Exception handled, program continues safely");
9         }
10        System.out.println("Application still running...");
11    }
12 }
```

Console × Problems Debug Shell  
<terminated> Task24 [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.  
Exception handled, program continues safely  
Application still running...

25.Develop a Java program that demonstrates the difference between throw and throws with examples in different methods.

```
1 package workshop4;
2
3 public class Task25 {
4
5     static void testThrows() throws ArithmeticException {
6         int a = 10 / 0;
7     }
8
9     static void testThrow() {
10         throw new ArithmeticException("Manually thrown exception");
11     }
12
13     public static void main(String[] args) {
14         try {
15             testThrows();
16         } catch (Exception e) {
17             System.out.println("throws example: " + e);
18         }
19
20         try {
21             testThrow();
22         } catch (Exception e) {
23             System.out.println("throw example: " + e);
24         }
25     }
26 }
```

Console × Problems Debug Shell  
<terminated> Task24 [Java Application] C:\Users\ASUS\p2\pool\plugins\org.eclipse.justj.openjdk.h  
Exception handled, program continues safely  
Application still running...

