EXCEPTIOS...!

Exception

```
try {
        System.out.println("Inside try Block");
        data = num / 0;
} catch (ArithmeticException e) {
        System.out.println("Inside Catch Block");
        System.out.println(e.getMessage());
} finally {
        System.out.println("Inside Finally Block");
}
```

An Exception is an **abnormal condition** that occur in a code at **run time**.

– Exception is a RUN TIME ERROR.

Eg, Zero division error on Integers.

• A Java **exception is an object** that describes an exceptional condition (that is, error) that occurred in a piece of code during **run time**. If the exception is NOT handled in a **catch block**, the program will crash.

How Java handle exception

1. When an Exception occurs, JVM detects it and create corresponding Exception class objects, for eg, if Zero division on integers, JVM creates object of class ArithmeticException.

2. Then the JVM halts the normal execution and ***throws*** *the created object to the method it created.*

3. JVM first check, if the function created exception has a matching ***catch*** *block implemented, handle it there, so that prgram wont crash, as given below.*

```
try {
        System.out.println("Inside try Block");
        data = num / 0;
} catch (ArithmeticException e) {
        System.out.println("Inside Catch Block");
        System.out.println(e.getMessage());
}
```

4. If a matching ***catch*** *block is* ***not available****, JVM passes (propogates) the exception object to the method that called it. (previous method in the Call stack).*

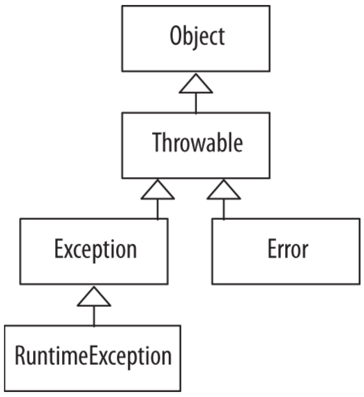
5. if still matching catch is not avaiable, propogate to the next function in the Call stack, this continue untill it unwinds the call stack, that till reached main(). This is called ***exception propogation****.*

6. If the matching catch is not found in Call stack, program terminates.

Example.

```
public static int devideProc(int num, int den) {
        return num / den;
       /* ArithmeticException object will be created
               and propogated/thrown to caller main() */
}
public static void main(String[] args) {
     try {
        int n = devideProc(5,0);
    } catch(ArithmeticException e) {
        /* ArithmeticException  object will be trapped/handled
                  here in matching catch block – program will NOT crash */
        System.out.println(e.getMessage());
    }
}
```

Exception Class Hirarchy



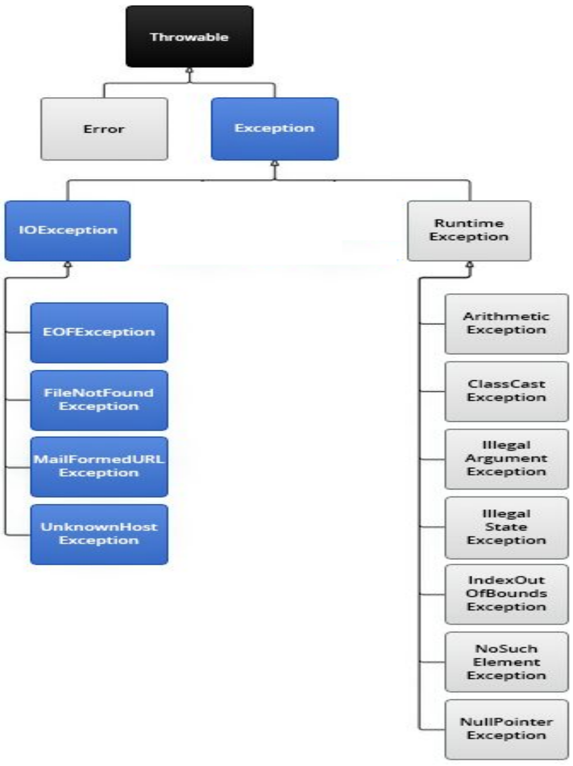
Object: The root class of the Java class hierarchy; every class in Java inherits from Object.

Throwable: The superclass of all errors and exceptions in Java.

Exception: Represents exceptions that are conditions a program might want to catch and handle, such as I/O errors.

RuntimeException: A subclass of Exception for exceptions that occur during the program's runtime, typically due to programming errors like NullPointerException.

Error: Represents serious issues beyond the program's control, often related to the JVM environment, such as OutOfMemoryError. ***It can not be handled,*** *Results in program crash*.



Type Of Exceptions

1. **Checked Exception –** Checks at Compile time.
   If not handled, will get compilation error.
2. **Unchecked Exception –** Checks at Runtime,
   If not handled, will get runtime error, and program ends.

Checked exceptions	**Unchecked exceptions**
• Checked at compile time.(COMPILE TIME EXCEPTIONS)	• NOT checked at compile time.(RUN TIME EXCEPTINS)
• Not sub class of RunTimeException	• Sub class of RunTimeException
• The method must either handle the exception or it must specify the exception using *throws* keyword.	• It is NOT needed to handle or catch these exceptions
• Shows compile error if checked exception is not handled.	• DO NOT Show compile error if exception is not handled. But shows run-time error.
• E.g. *ClassNotFoundException, IOException*	• Eg. *ArithmeticException, ArrayIndexOutOfBoundsException*

Exception handling fundamentals

5 Key words

1. **try**
2. **catch**
3. **finally**
4. **throws**
5. **throw**

try
catch
throw
throws
finally
Java exception handling is managed via FIVE keywords:

1. try

In exception handling, the try block is used to define a section of code where exceptions might occur.

It ensures that if an exception occurs, the program can transfer control to a corresponding catch block to handle the exception, preventing the program from crashing.

2. catch

In exception handling, the catch block is used to handle exceptions that are thrown from a corresponding try block.

When an exception occurs in the try block, the catch block is used to define what should happen in response to that specific type of exception.

The catch block handles the exception, allowing the program to continue running smoothly after an error is detected in the try block.

3. finally

In exception handling, the finally block is used to define code that will always be executed, whether an exception is thrown or not.

To ensure that certain code runs no matter what happens during the try-catch execution., fo eg, certain cleanup tasks such as closing resources.

Format:

```
try {
   // Code that may throw an exception
} catch (ExceptionType e) {
   // Code to handle the exception
} finally {
   // Code that always executes
}
```

Example:

```
try {
   int[] numbers = {1, 2, 3};
   System.out.println(numbers[5]);
   // This will throw ArrayIndexOutOfBoundsException
} catch (ArrayIndexOutOfBoundsException e) {
   System.out.println("Exception caught: " + e.getMessage());
} finally {
   System.out.println("This block is always executed.");
}
```

4. throws

is used in a method declaration when the method **chooses not to handle potential exceptions itself** within the method.

It indicates that the method may throw one or more exceptions during execution.

It informs the compiler and the caller that they need to be aware of these exceptions and be prepared to handle them:

Either by using a try-catch block to catch the exception.
Or by letting the exception propagate further up the call stack,
possibly to another caller.

This approach is typically used with checked exceptions, which must be either caught or declared in the method signature.

Example:

```
class ThrowsTest {
        public static int divisionTest(int a, int b) throws ArithmeticException {
                return a/b;
        }
        public static void main(String[] args) {
                try {
                        divisionTest(5,0);
                } catch (ArithmeticException e) {
                        System.out.println(e.getMessage());
                }
        }
}
```

5. throw
In Java, the throw keyword is used to explicitly throw an exception from a method or a block of code.

Once the exception is thrown, the program doesn't proceed with normal execution until the exception is either handled by an appropriate catch block in the call stack, or the program terminates if the exception remains unhandled.

```
class ThrowTest {
        public static int divisionTest(int a, int b) {
                if (b == 0) {
                        throw new ArithmeticException("Zero Division Error");
                }
                return a/b;
        }
        public static void main(String[] args) {
                try {
                        divisionTest(5,0);
                } catch (ArithmeticException e) {
                        System.out.println(e.getMessage());
                }
        }
}
```

Creating User defined Exception

In java we can create custome or user defined exception. For eg, the requirement is to create an exception, say InvalidAgeException, when the entered age is below 18 or a negative value etc. The steps involved are

Example:

```
// Custom checked exception by extending Exception class
class InvalidAgeException extends Exception {
    // Constructor with a custom message
    public InvalidAgeException(String message) {
        super(message);
    }
}

public class TestCustomException {
    // Method that throws the custom checked exception
    public static void checkAge(int age) throws InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("Age less than 18 is not allowed.");
        } else {
            System.out.println("Valid age");
```

```
                }
            }

            public static void main(String[] args) {
                try {
                    // This will throw the custom InvalidAgeException
                    checkAge(16);
                } catch (InvalidAgeException e) {
                    // Handle the custom exception
                    System.out.println("Caught Exception: " + e.getMessage());
                }
            }
        }
```

Multiple Catch clauses

In some cases, more than one exception could be raised by a single block of code.
For eg,

```
String num1 = "one", String den = "0";
int num1 = Integer.parseInt(num);
int den1 = Integer.parseInt(den);
int div = num/den;
```

Above code snippet can generate NumberFormatException and ArithemticException.

To handle this type of situation, you can specify two or more catch clauses, each catching a different type of exception.

When an exception is thrown, each catch statement is inspected in order, and the first one whose type matches that of the exception is executed.

After one catch statement executes, the others are bypassed, and execution continues after the try / catch block.

Exception Order in Catch
Subclasses must come before any of their superclasses

This is because a catch statement that uses a superclass will catch exceptions of that type plus any of its subclasses.

Thus, a subclass would never be reached if it came after its superclass. Further, in Java, unreachable code is an error, and result in COmpilation Error

```
class MultipleCatch {
        static void testMultipleCatch(String num, String den) {
                try {
                        int num1 = Integer.parseInt(num);
                        int den1 = Integer.parseInt(den);
                        int res = num1/den1;
                } catch (NumberFormatException e1) {
                        System.out.println(e1.getMessage());
                }catch (ArithmeticException e2) {
                        System.out.println(e2.getMessage());
                }
        }
        public static void main(String[] args) {
                // pass string values
                testMultipleCatch("10", "0");
        }
}
```

Order of Exception in Catch

Subclasses** must come **first** before any of their **superclasses

This is because a catch statement that uses a superclass will catch exceptions of that type plus any of its subclasses.

*Thus, a subclass would never be reached if it came after its superclass. Further, in Java, unreachable code is an error, and result in **Compilation Error***

Nested try block

The try statement can be nested within another try.

When entering a try statement, the context of that exception is added to a **stack**.

If an inner try lacks a handler for a specific exception, the stack is checked, and the next try's handlers are inspected.

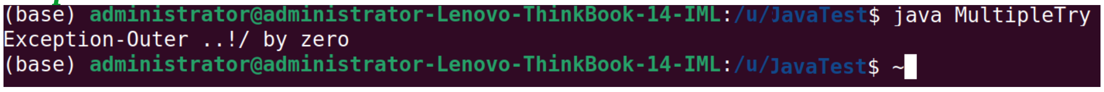
This process continues until a matching catch is found,
or all nested try blocks are checked.

If no catch matches, the Java runtime system handles the exception and program terminates.

```
/*
 * If an inner try lacks a handler for a specific exception,
   the stack is checked, and the next try's handlers are inspected.
   here res = num1/den1; is not handled in inner try,
   but eventually its get handled in outer exception...!
 */
class MultipleTry {
static void testMultipleTry(String num, String den) {
        try {
                int num1 = Integer.parseInt(num);
                int den1 = Integer.parseInt(den);
                try {
                        int res = num1/den1;
                }catch (NullPointerException e2) {
                        System.out.println("Exception-Inner ..! "+ e2.getMessage());
                }
        } catch (Exception e1) {
                System.out.println("Exception-Outer ..!" + e1.getMessage());
        }
}

public static void main(String[] args) {
                testMultipleTry("10", "0");
        }
}
```

Output:



```
(base) administrator@administrator-Lenovo-ThinkBook-14-IML:/u/JavaTest$ java MultipleTry
Exception-Outer ..!/ by zero
(base) administrator@administrator-Lenovo-ThinkBook-14-IML:/u/JavaTest$ ~
```

Note: int res = num1/den1; is **not handled in inner try**, but eventually its get handled in **outer exception...**!

<u>University exam questions</u>

1. Differentiate between checked and unchecked exceptions in Java with examples.?
  Ans: *refer "Type Of Exceptions" in this doc*

2. Demonstrate the significance of the keywords 'try', 'catch', 'finally', 'throw' and 'throws' in exception handling of Java with appropriate examples.
  Ans: refer "Exception handling fundamentals" in this doc.

3. What is exception? List any four exception classes in Java.
  Ans*: An Exception is an abnormal condition that occur in a code at run time.*
  – Exception is a RUN TIME ERROR.
  Eg, Zero division error on Integers.

  • A Java exception is an object that describes an exceptional condition (that is, error) that occurred in a piece of code during run time. If the exception is NOT handled in a catch block, the program will crash.

  Classes : IOException, FileNotFoundException – these are checked exceptions
  ArithmeticException, NumerFormatException – these are unchecked exception

4. Briefly explain various exception handling keywords in Java, with examples.
  Ans: refer "Exception handling fundamentals" in this doc.

5. Explain in detail how exception handling mechanism used in Java using BufferedReader class

Ans

1. FileNotFound Exception – in opening a file, that is creting a reader object. This will be throwned, if the file does not exist.

2. IOException: in reading data using reader object. This is because, input source can result in various I/O errors (e.g., file not found, stream closed, network issues, etc.).

```
        class BufferedReaderTest {
                void readFile(String filename) {
                        FileReader freader = null;
                        try {
                                freader = new FileReader(filename);
                        } catch (FileNotFoundException e) {
                                e.printStackTrace();
                        }
                        BufferedReader reader = new BufferedReader(freader);
                        try {
                                String str = reader.readLine();
```

```
                    } catch (IOException e) {
                            e.printStackTrace();
                    }
            }
    }
    public class Lab_eception {
            public static void main(String[] args) {
                    BufferedReaderTest brt = new BufferedReaderTest();
                    brt.readFile("input.txt");
            }
    }
```

6. How user defined eception is created ?. ==> important question

Ans*: Refer "Creating User defined Exception" in this doc*

7. Write a Java program to read characters from the console using 'throw' and 'throws

Ans:

```
    import java.io.*;
    class ConsoleReader {
            static void readFunction() throws IOException{
                    InputStreamReader irs = new InputStreamReader(System.in);
                    BufferedReader reader = new BufferedReader(irs);
                    System.out.print("Enter a string :");
                    String str = reader.readLine();
                    if(str.isEmpty() || str.length() <= 0) {
                            throw new IOException("Exception:/read empty
    string...!");
                    }
            }
            public static void main(String[] args) {
                    try {
                            readFunction();
                    } catch (IOException e) {
                            System.out.println(e.getMessage());
                    }
            }
    }
```

***8.* Explain the scenario under which the following three exceptions occur, NumberFormatException, ArithmeticException, and ArrayIndexOutOfBoundsException.**

Ans:

ArithmeticException:

```
        int x = 5/0;
```

NumberFormatException

```
        String strNumber = "one";
        int digit = Integer.parseInt( strNumber);
```

ArrayIndexOutOfBoundsException:

```
        int[] arr = new int[3];
        arr[5] = 100;
```

9. What is an exception? Why it needs to be handled?

Ans: refer question 3

If exception is not handles, program will crash for unchecked exception.

If exception is not handles, compilation error for checked exception.

10: illustrate order of exception in multiple catch blocks ?

Ans: refer "Multiple Catch clauses" in this doc.

University exam questions – Module 3

1. Describe various methods of reading data from the keyboard with appropriate examples in Java

Ans:

1. Scanner class

```
package java.utils
/*------Class 1: Scanner class------*/
Scanner scanner = new Scanner(System.in);
System.out.print("Enter your name: ");
// Read a line of text from the console
String name = scanner.nextLine();
// Display the input received
System.out.println("Hello, " + name + "!");
/*-------------------------------------*/
```

2. BufferedReader class

```
package java.io
/*------Class 2: BufferedReader class------*/
InputStreamReader Isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(Isr);
System.out.print("Enter your name: ");
try {
name = br.readLine();
System.out.println("Hello, " + name + "!");
} catch (IOException e) {
System.out.println(e.getMessage());
}
/*----------------------------------------------*/
```

3. DataInputStream class

```
package java.io
/*------Class 3: DataInputStream class------*/
DataInputStream dis = new DataInputStream(System.in);
System.out.print("Enter your name: ");
try {
name = dis.readLine();
} catch (IOException e) {
System.out.println(e.getMessage());
}
System.out.println("Hello, " + name + "!");
/*-------------------------------------------------*/
```

4. InputStreamReader class

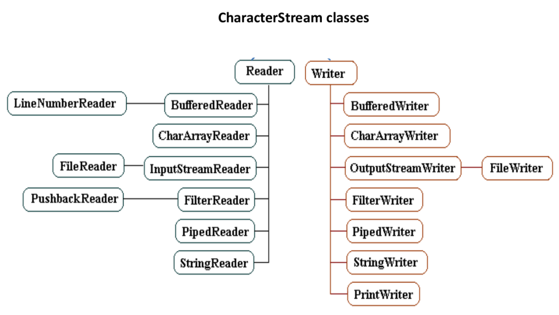
```
package java.io
/*------Class 3: DataInputStream class------*/
InputStreamReader Isr1 = new InputStreamReader(System.in);
char[] c = new char[20];
try {
Isr1.read(c);
System.out.println("Hello, " + new String(c));
} catch (IOException e) {
```

```
System.out.println(e.getMessage());
}
/*-----------------------------------------------*/
```

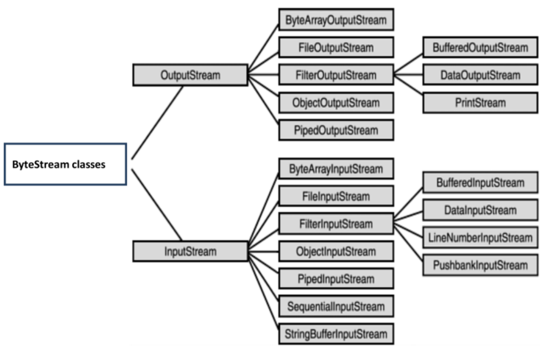
2. Demonstrate the significance of the keywords 'try', 'catch', 'finally', 'throw' and 'throws' in exception handling of Java with appropriate examples.

Ans: refer "Exception handling fundamentals" in this doc.

3. Write a note on byte stream and character stream related classes.



<u>Byte Stream Classes</u>



***Definition: Byte streams** are used to perform input and output of 8-bit bytes. They are suitable for handling raw binary data such as image files, audio files, or any other binary file formats.*

***Definition: Character streams** are used to handle 16-bit Unicode characters, making them suitable for handling text data. Character streams are intended for text-based files such as .txt, .xml, or .html.*

4. Write a Java program that accepts N integers through console and compute their average.

```
import java.util.Scanner;
public class CalcAvg {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the number of integers (N): ");
      int N = scanner.nextInt();
      int sum = 0;
      for (int i = 1; i <= N; i++) {
         System.out.print("Enter integer " + i + ": ");
         int num = scanner.nextInt();
         sum += num;
      }
      double average = (double) sum / N;
      System.out.println("The average is: " + average);
      scanner.close();
   }
}
```

5. Develop a java package named primepackage, with a class Prime containing a static method that check whether a number is prime or not and returns that information. Import this package in another class and use to check a number is

prime or not.

Ans

1. create a folder primepackage in woring folder.

2. create a class Prime.java inside   primepackage folder.

```
package primepackage;
public class Prime {
        public static boolean isPrime(int N) {
                boolean retVal = false;
                if ( N<=1 ) {
                        return false;
                }
                if (N==2) {
                        return true;
                }
                for (int i=2; i<=N/2; i++) {
                        if (N % i == 0) {
                                return false;
                        }
                }
                return true;
        }
}
```

2. create a Test class in working folder as below importing primepackage.

```
import primepackage.*;
public class PrimeTest {
public static void main(String[] s) {
           int N = 5;
                if (true == Prime.isPrime(N)) {
                        System.out.println(N + " is Prime");
                } else {
                        System.out.println(N + " is NOT Prime");
                }
        }
}
```

6. Model a Java class in such a manner that it is restricted to have only one instance throughout the program in which it is used.

This is a good question.

*This is a **Design pattern** called **Singleton**, where only one instance is created for all users.*

```
class Singleton {
   private static Singleton instance;
   private Singleton() {
      System.out.println("Singleton Constructor...!");
   }
   public static Singleton getInstance() {
         if (instance == null) {
                instance = new Singleton();
         }
         return instance;
   }
   public void printMessage() {
      System.out.println("printMessage() hashcode : " + this.hashCode());
```

```
    }
}
public class SingleTonTest {
        public static void main(String[] args) {
        Singleton singleton1 = Singleton.getInstance();
        Singleton singleton2 = Singleton.getInstance();
        singleton1.printMessage();
        singleton2.printMessage();
        }
}
```

Output:

```
        Singleton Constructor...!
        printMessage() hashcode : 312116338
        printMessage() hashcode : 312116338
```

7. Can a class in Java implement more than one interfaces, if yes what is the syntax used?

Eg, Multiple Inheritance in java

```
interface Animal {
 void eat();
}
interface Bird {
void fly();
}
class Bat implements Animal, Bird {
        // Implementing abstract method from Animal interface
        public void eat() {
        System.out.println("Bat eats insects.");
        }
        // Implementing abstract method from Bird interface
        public void fly() {
        System.out.println("Bat can fly.");}
        }
public class Main {
public static void main(String[] args) {
        Bat bat = new Bat();
        bat.eat(); // From Animal interface
        bat.fly(); // From Bird interface
}
}
```

8. Write a Java program to create a new file named 'MyFile.txt' and write the statement "This is the University Exam for OODP. This a program to illustrate the use of files." into the file with each sentence in the statement representing a new line in the file

```
import java.io.*;
public class MyFileTest {
        public static void main(String[] args) {
                BufferedWriter bw = null;
                try {
                        bw = new BufferedWriter( new FileWriter("MyFile.txt"));
                        bw.write("This is the University Exam for OODP.");
                        bw.write("\n");
                        bw.write("This a program to illustrate the use of files.");
                        bw.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```

9. Define two user defined exception EvenNumberException' and 'OddNumberException'. Write a Java class which has a method which checks whether a given number if even or not. The method throws 'EvenNumberException' or 'OddNumberException' if the number is even or odd respectively. Illustrate the handling of the exception with suitable sequence of codes.

```
class EvenNumberException extends Exception {
        EvenNumberException (String str) {
                super(str);
        }
        public String getMsg() {
                return "EvenNumberException".toString();
        }
}
class OddNumberException extends Exception {
        OddNumberException (String str) {
                super(str);
        }
        public String getMsg() {
                return "OddNumberException".toString();
        }
}
public class NumberExcpetionTest {
        public static void checkNumber(int N)
                        throws EvenNumberException, OddNumberException {
                if (N % 2 == 0) {
                        throw new EvenNumberException("Even number...!");
                } else {
                        throw new OddNumberException("Odd number...!");
                }
        }
        public static void main(String[] args) {
                int N = 100;
                try {
                        checkNumber(N);
                } catch(EvenNumberException e) {
                        System.out.println(e.getMessage());
                } catch(OddNumberException e) {
                        System.out.println(e.getMessage());
                }
                N = 99;
                try {
                        checkNumber(N);
                } catch(EvenNumberException e) {
                        System.out.println(e.getMessage());
                } catch(OddNumberException e) {
                        System.out.println(e.getMessage());
                }
        }
}
```

Output:

```
        Even number...!
        Odd number...!
```

10. Explain the scenario under which the following three exceptions occur, NumberFormatException, ArithmeticException, and ArrayIndexOutOfBoundsException.

```
public class ExcetionTest {
        public static void main(String[] args) {
                // NumberFormatException
                String num = "one";
                try {
                        int n = Integer.parseInt(num);
                } catch(NumberFormatException e) {
                        System.out.println(e.getMessage());
                }
                // ArithmeticException
                int a = 10, b = 0;
                try {
                        int n = a/b;
                } catch(ArithmeticException e) {
                        System.out.println(e.getMessage());
                }
                // ArrayIndexOutOfBoundsException
                try {
                        int[] arr = new int[3];
                        arr[0] = 0;
                        arr[1] = 1;
                        arr[2] = 2;
                        arr[3] = 3;
                } catch(ArrayIndexOutOfBoundsException e) {
                        System.out.println(e.getMessage());
                }
        }
}
```

Output:

```
        For input string: "one"
        / by zero
        Index 3 out of bounds for length 3
```