JAVA Programming Assignment 04

Name : Aaditya Khot

PRN : 21610051

Batch : S-5

1.Write a program in java to handle below exceptions

a. Divide by Zero
b. Array Index Out Of Bound
c. Number Format
d. Null Pointer

```
public class ExceptionHandlingExample {

    public static void main(String[] args) {
        try {
            // Divide by Zero
            int a = 10;
            int b = 0;
            int c = a / b;
        } catch (ArithmeticException e) {
            System.out.println("Divide by zero exception: " + e.getMessage());
        }

        try {
            // Array Index Out Of Bound
            int[] arr = new int[5];
            arr[10] = 11;
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array index out of bound exception: " + e.getMessage());
        }

        try {
            // Number Format
            String s = "abc";
            int i = Integer.parseInt(s);
        } catch (NumberFormatException e) {
            System.out.println("Number format exception: " + e.getMessage());
        }

        try {
            // Null Pointer
            String str = null;
            int length = str.length();
        } catch (NullPointerException e) {
            System.out.println("Null pointer exception: " + e.getMessage());
        }
    }
}
```

2. Write a program in java to handle custom exception with single try block and multiple catch block.

```
class CustomException extends Exception {
    public CustomException(String message) {
        super(message);
    }
}

public class CustomExceptionHandlingExample {

    public static void main(String[] args) {
        try {
            int age = 17;
            if (age < 18) {
                throw new CustomException("You are not eligible to vote.");
            }
            System.out.println("You are eligible to vote.");
        } catch (CustomException e) {
            System.out.println("Custom exception caught: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Exception caught: " + e.getMessage());
        }
    }
}
```

3. Write a program in java to show the use of finally keyword.

```
public class FinallyExample {

    public static void main(String[] args) {
        try {
            int a = 10;
            int b = 0;
            int c = a / b;
            System.out.println("This code will not be executed.");
        } catch (ArithmeticException e) {
            System.out.println("Divide by zero exception: " + e.getMessage());
        } finally {
            System.out.println("This code will always be executed.");
        }
    }
}
```

4. Write a program in java for handling exceptions with nested try block.

```
public class NestedTryBlockExample {

    public static void main(String[] args) {
        try {
            int[] arr = {1, 2, 3};
            int a = 10;
            int b = 0;
            try {
                int c = a / b; // This will throw an ArithmeticException
            } catch (ArithmeticException e) {
                System.out.println("Divide by zero exception: " + e.getMessage());
            }
            System.out.println("The value at index 5 is " + arr[5]); // This will throw an
ArrayIndexOutOfBoundsException
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array index out of bound exception: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Exception caught: " + e.getMessage());
        } finally {
            System.out.println("This code will always be executed.");
        }
    }
}
```

5. Write a program in java for custom exception to check speed of car on highway, if speed exceeds 120Km/hr then throw a ‘Speed Limit Exceeded’ exception. (use throw)

```
class SpeedLimitExceededException extends Exception {
    public SpeedLimitExceededException(String message) {
        super(message);
    }
}

public class CustomExceptionExample {

    public static void main(String[] args) {
        int speed = 150; // Assume the speed is 150 km/hr
        try {
            if (speed > 120) {
                throw new SpeedLimitExceededException("Speed Limit Exceeded: " + speed + "
km/hr");
            } else {
                System.out.println("The speed is within the limit.");
            }
        } catch (SpeedLimitExceededException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

6. Differentiate in between throw and throws keyword.

In Java, `throw` and `throws` are two keywords that are used to handle exceptions, but they have different meanings and usages.

`throw` is used to explicitly throw an exception from a method or a block of code. It is used when the programmer wants to throw an exception manually, rather than waiting for an exception to occur naturally. For example:

```
if (balance < 0) {
    throw new InsufficientFundsException("Balance is negative");
}
```

In this code snippet, the throw keyword is used to throw a custom InsufficientFundsException if the balance is negative.

throws is used in the method signature to indicate that the method may throw one or more exceptions. It is used to inform the caller of the method that they need to handle or propagate the exceptions that may occur. For example:

```
public void readFile(String fileName) throws FileNotFoundException, IOException {
    // Code to read file
}
```

In this code snippet, the throws keyword is used in the method signature to indicate that the readFile method may throw two exceptions - FileNotFoundException and IOException. The caller of the method needs to handle or propagate these exceptions.

In summary, throw is used to explicitly throw an exception, while throws is used to indicate that a method may throw one or more exceptions.

7. Explain exception handling mechanism.

Exception handling in Java is a mechanism that allows a program to handle unexpected or exceptional situations that occur during the execution of the program. An exception is an event that occurs during the execution of a program and disrupts the normal flow of the program.

When an exception occurs, the program terminates abruptly and the error message is displayed on the console, which is not user-friendly. Exception handling provides a way to handle exceptions gracefully and to recover from them if possible.

The basic mechanism of exception handling in Java involves the following:

Throw an Exception: If an exceptional situation occurs, the program generates an exception using the throw keyword.

Catch an Exception: The program catches the exception using the try-catch block. The try block contains the code that may generate an exception, and the catch block contains the code that handles the exception.

Handle the Exception: If the exception occurs, the program executes the code in the catch block that handles the exception.

Finally Block: A finally block can be added to the try-catch block. The code in the finally block is always executed, regardless of whether an exception occurred or not. This is useful for releasing resources and cleaning up after the program.

Java also provides a hierarchy of exception classes that can be used to handle different types of exceptions. The Throwable class is the root class of the exception hierarchy, and it has two subclasses: Exception and Error. The Exception class represents exceptional conditions that can be handled by the program, while the Error class represents exceptional conditions that cannot be handled by the program.

To handle an exception, the try block is used to enclose the code that might throw an exception. The catch block is used to catch the exception and handle it. Multiple catch blocks can be used to handle different types of exceptions. The finally block is used to release resources, such as files or network connections, that were opened in the try block.

In summary, exception handling in Java is a mechanism that allows a program to handle unexpected or exceptional situations that occur during the execution of the program. It provides a way to handle exceptions gracefully and to recover from them if possible.

8. Write a program in java for handling checked exceptions using throws keyword.

```
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class FileReaderExample {

    public static void main(String[] args) throws IOException {
        File file = new File("example.txt");
        FileReader fr = null;
        try {
            fr = new FileReader(file);
            char[] chars = new char[(int) file.length()];
            fr.read(chars);
            System.out.println(chars);
        } finally {
            if (fr != null) {
                fr.close();
            }
        }
    }
}
```