Task 1

In this task, you are given a Java code snippet that performs division between two integers 'a' and 'b'. Your goal is to handle exceptions that may occur during the division operation. You need to test the code with different inputs for 'a' and 'b' and understand the flow of the program by adding print statements before and after the division operation. Specifically, you need to add catch blocks for two types of exceptions: NumberFormatException and ArithmeticException.

```
static int computeDivision(int a, int b) {
    int res = 0;
    try {
        res = a / b;
    } catch (NumberFormatException ex) {
        System.out.println("NumberFormatException is occurred");
    }
    return res;
}

public static void main(String args[]) {
    int a = 1;
    int b = 0;
    try
    {
        int i = computeDivision(a, b);
    }
    catch (ArithmeticException ex) {
        System.out.println(ex.getMessage()); // Print description of the
exception (here / by zero)
    }
}
```

Task-2

Throw, Throws, and Finally:

1. Finally Block: You need to modify the code from Task-1 by adding a finally block to the try-catch construct. The finally block will contain a print statement. The purpose of the finally block is to execute the code inside it, regardless of whether an exception occurred or not. Observe how the finally block behaves in different scenarios.
2. Now, you are required to write a separate test example to catch the IndexOutOfBoundsException. The IndexOutOfBoundsException is thrown when attempting to access an element at an invalid index in an array or a collection. You need to demonstrate how to handle this exception using try-catch blocks.
3. Define DivisionbyZeroException: You need to define your custom exception class called DivisionbyZeroException that extends the Exception class. The purpose of this exception is to handle cases where the denominator is zero during division. You will implement this exception by providing a constructor that takes a string message as a parameter and calls the constructor of the parent Exception class with the message.

Task-3

Bank Account Class Description:

The BankAccount class represents a simple bank account with basic functionalities to manage the account balance. It is part of the larger Banking System example. The class encapsulates the account details, such as the account number, account holder's name, and the account balance.

Class: BankAccount

Properties:

- `accountNumber` (String): A string representing the account number of the bank account.

- `accountHolder` (String): A string representing the name of the account holder.

- `balance` (double): A double representing the current balance of the bank account.

Constructor:

- `BankAccount(String accountNumber, String accountHolder, double initialBalance)`: Constructs a BankAccount object with the provided account number, account holder's name, and initial

balance. The initial balance is the amount of money available in the account when the account is created.

Public Methods:

- `getAccountNumber() -> String`: Returns the account number associated with the bank account.

- `getAccountHolder() -> String`: Returns the name of the account holder associated with the bank account.

- `getBalance() -> double`: Returns the current balance of the bank account.

- `**deposit(double amount)** -> void`: Accepts a double value representing the amount to deposit and increases the account balance by that amount. It also prints a message to indicate the deposit transaction.

- `**withdraw(double amount)** -> void`: Accepts a double value representing the amount to withdraw. It decreases the account balance by that amount if the account has sufficient funds. If the withdrawal amount exceeds the account balance, it throws an `InsufficientFundsException`. If the withdrawal is successful, it also prints a message to indicate the withdrawal transaction.

Exceptions:

- `InsufficientFundsException`: This is a custom exception class that extends the `Exception`. It is thrown when a withdrawal request is made, and the total balance in the account is less than the amount to be withdrawn.