Practice problem sets

1. **NullPointerException**

   Create a Java class **Person** with attributes **name** and **age**. Then, create a method **printAge** that prints the age of the person. In the main method, create a Person object without initializing it. Call the printAge method on this object and handle the resulting **NullPointerException** by printing an appropriate error message.

2. **InvalidAgeException**

   Design a custom exception called **InvalidAgeException**. Create a method called **validateAge** that takes an integer age as input and throws the **InvalidAgeException** if the age is **less than 0 or greater than 120**. In the main method, take age as user input and use the validateAge method to check if it's valid. Handle the custom exception by printing an appropriate error message.

3. **Bank Account Exception**

   Create a class **BankAccount** with attributes **accountNumber** and **balance**. Implement a method called **withdraw** that takes a double value as the amount to withdraw. If the withdrawal amount is greater than the balance, throw a custom exception called **InsufficientFundsException**. In the main method, create a BankAccount object with an initial balance. Prompt the user to enter the withdrawal amount and handle the custom exception by printing an appropriate error message.

4. **Email Validation**

   Create a method called **validateEmail** that takes a string **email address** as input and checks if it's a valid email format. The valid email format should have the following criteria:

   - It must contain exactly one '@' symbol.
   - The domain part must contain at least one '.' symbol.
   - The domain and username should not contain spaces.

   If the email address does not meet these criteria, throw a custom exception called **InvalidEmailException**. In the main method, take the email address as user input and use the validateEmail method to check if it's valid. Handle the custom exception by

printing an appropriate error message.

5.

2a. You have a class called "Book" with attributes: title, author, and numberOfAvailableCopies, which indicates how many copies of the book are available in the store. When someone buys a book, the available copies decrease by 1. If the available copies reach zero, the "buyBook" method in the "Book" class should raise a custom exception called "BookOutOfStock" with the message: "[BookOutOfStockException] The value of numberOfAvailableCopies is 0."
Now, implement this functionality by adding code to the specified portion below.

```
import java.util.ArrayList;
import java.util.List;
class Book {
   String title; String author;
   int numberOfAvailableCopies;

   public Book(String title, String author, int numberOfAvailableCopies) {
      this.title = title; this.author = author;
      this.numberOfAvailableCopies = numberOfAvailableCopies;
   }

   void buyBook(){// Complete the method}
}
```

6.

2. Consider the following code:

```
class Math{
public int divide(int a, int b){
    return a/b;
}
}
public class test {
   public static void main(String[] args) {
       Math m = new Math();
       int n = 4;
       int d = 0;
       System.out.println(m.divide(n,d));
       System.out.println("Complete"); //this line should be always
executed
   }
}
```

a) Write the necessary codes **within the main method** so that no exception is thrown. Also make sure "Complete" is always printed. **[3]**

b) Write a new user defined exception called '**DivbyZero**' as a subtype of '**ArithmeticException**'. **[5]**

- The '**divide()**' should check if the second parameter is '**0**' or not.
- If the second parameter is '**0**', the method should throw '**DivbyZero**' exception with a message stating "**The denominator can not be zero!**".
- The '**divide()**' method should have proper use of '**throw**' and '**throws**'.