Question 1: Write a Java method that takes two integers as input and returns their sum. Ensure your method is public and static.

Code:

```
public class Calculator{
    public static int sum(int a,int b){
        return a+b;
    }
    public static void main(String[] args){
        System.out.println(sum(5,3));
        System.out.println(sum(-1,10));
        System.out.println(sum(0,0));


    }
}
```

Output:

```
8
9
0
```

Question 2: Implement a Java method that checks if a given integer is even or odd. The method should return "Even" for even numbers and "Odd" for odd numbers.

Code:

```
public class NumberChecker {
```

```
            public static String checkEvenOdd(int number){
                return (number%2==0?"Even":"Odd");
            }
            public static void main(String[] args) {
                System.out.println(checkEvenOdd(4));
                System.out.println(checkEvenOdd(7));
                System.out.println(checkEvenOdd(0));
            }
        }
```

Output:

```
Even
Odd
Even
```

Question 3: Create a Java method that takes an array of integers and returns the largest number found in the array.

Code:

```
public class ArrayUtils {
    public static int findLargest(int[] numbers){
        int max = Integer.MIN_VALUE;
        for(int i=0;i<numbers.length;i++){
            if(numbers[i]>max){
                max = numbers[i];
            }
        }
        return max;
    }
    public static void main(String[] args){
        System.out.println(findLargest(new int[]{1,5,2,9,3}));//Output: 9
        System.out.println(findLargest(new int[]{-10,-5,-2}));//output: -2
        System.out.println(findLargest(new int[]{7}));
    }
}
```

Output:

```
9
-2
7
```

Question 4: Write a Java method that takes a String and returns its length.

Code:

```
public class StringHelper {
    public static int getStringLength(String str){
        return str.length();
    }
    public static void main(String[] args) {
        System.out.println(getStringLength("hello"));//output: 5
        System.out.println(getStringLength(""));//Output: 0
        System.out.println(getStringLength("Java is fun"));//Output: 11
    }
}
```

Output:

```
5
0
11
```

Question 5: Implement a Java method 'reverseString' that takes a string as input and returns a new string with the characters in reverse order. Do not use built-in reverse functions for String or StringBuilder/StringBuffer.

Code:

```
public class StringManipulator {
    public static String reverseString(String str){
        String stringReverse="";
        for(int i=str.length()-1;i>=0;i--){
            stringReverse+=str.charAt(i);
        }
        return "\""+stringReverse+"\"";
    }
    public static void main(String[] args) {
        System.out.println(reverseString("hello"));
        System.out.println(reverseString("Java"));
        System.out.println(reverseString(""));
        System.out.println(reverseString("a"));
    }
}
```

Output:

```
"olleh"
"avaJ"
""
"a"
```

Question 6: Write a Java method that calculates the Nth Fibonacci number. The Fibonacci sequence starts with 0 and 1. The method should handle N >= 0

Code:

```
public class FibonacciCalculator {
    public static int fibonacci(int n){
        if(n<=1) return n;
        int a=0,b=1;
        for(int i=2;i<=n;i++){
            int c = a+b;
            a=b;
            b=c;
        }
        return b;
    }
    public static void main(String[] args) {
        System.out.println(fibonacci(0));
        System.out.println(fibonacci(1));
        System.out.println(fibonacci(6));
        System.out.println(fibonacci(10));


    }
}
```

Output:

```
0
1
8
55
```

Question 7: Given a 'List of strings, write a Java method that removes all duplicate strings from the list and returns a new 'List' containing only unique strings, maintaining their original order of appearance.

Code:

```
import java.util.List;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Set;

import java.util.LinkedHashSet;

public class ListUtils {

    public static List<String> removeDuplicates(List<String> list){

        return new ArrayList<>(new LinkedHashSet<>(list));

    }

    public static void main(String[] args) {

        System.out.println(removeDuplicates(Arrays.asList("apple",
"banana", "apple", "orange")));

        System.out.println(removeDuplicates(Arrays.asList("a", "b",
"c")));

        System.out.println(removeDuplicates(Arrays.asList("red",
"blue", "red", "green", "blue")));

        System.out.println(removeDuplicates(Arrays.asList()));

    }

}
```

Output:

```
[apple, banana, orange]

[a, b, c]
```

```
[red, blue, green]

[]
```

Question 8: Design a 'BankAccount' class with `private' fields for 'accountNumber` (String), `accountHolderName` (String), and `balance` (double). Provide a constructor to initialize these fields. Implement 'public' getter methods for all fields and 'deposit(double amount)` and `withdraw(double amount) methods. The 'withdraw' method should only allow withdrawal if `balance` is sufficient, otherwise it should return 'false'. Both 'deposit' and 'withdraw` methods should prevent negative amounts and return `true` on success.

Code:

```
public class BankAccount {

private String accountNumber;

private String accountHolderName;

private double balance;

public BankAccount(String accountNumber, String
accountHolderName, double initialBalance) {

    this.accountNumber = accountNumber;

    this.accountHolderName = accountHolderName;

    this.balance = initialBalance;

}

// Getters

public String getAccountNumber() {

    return accountNumber;
```

```
}
public String getAccountHolderName() {
    return accountHolderName;
}
public double getBalance() {
    return balance;
}
// Methods
public boolean deposit (double amount) {
    if(amount<=0) return false;
    balance+=amount;
    return true;
}
public boolean withdraw(double amount) {
    if(amount<=0 | amount>balance) return false;
    balance-=amount;
    return true;
}
public static void main(String[] args){
    BankAccount b1 = new BankAccount("123","John Doe",100.0);
    System.out.println(b1.deposit(50.0));
    BankAccount b2 = new BankAccount("123","John Doe",100.0);
    System.out.println(b2.withdraw(30.0));
    BankAccount b3 = new BankAccount("123","John Doe",100.0);
    System.out.println(b3.withdraw(120.0));
    BankAccount b4 = new BankAccount("123","John Doe",100.0);
    System.out.println(b4.deposit(-20.0));
    BankAccount b5 = new BankAccount("123","John Doe",100.0);
    System.out.println(b5.withdraw(-10.0));

}
}
```

Output:

```
true
true
false
false
false
```

Question 9: Create a custom checked exception named 'Insufficient FundsException'. Then, implement a 'Payment Processor' class with a method 'processPayment(double amount)` that simulates a payment transaction. This method should `throw` `Insufficient Funds Exception` if the `amount' exceeds a predefined 'MAX_PAYMENT_LIMIT` (e.g., 1000.0). Also, demonstrate how to catch this exception in a 'main' method.

Code:

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

public class PaymentProcessor {
private static final double MAX_PAYMENT_LIMIT = 1000.0;
public void processPayment(double amount) throws
InsufficientFundsException {
    if(amount>MAX_PAYMENT_LIMIT){
        throw new
InsufficientFundsException("InsufficientFundsException caught and
handled.");
    }

}
public static void main(String[] args) {
PaymentProcessor processor = new PaymentProcessor();
try {
processor.processPayment (500.0);
System.out.println("Payment 1: Successfully processed.");
} catch (InsufficientFundsException e) {
System.out.println("Payment 1: Error - "+ e.getMessage());
}
try {
processor.processPayment(1200.0);
System.out.println("Payment 2: Successfully processed.");
} catch (InsufficientFundsException e) {
System.out.println("Payment 2: Error - "+ e.getMessage());
}
}
}
```

Question 10: Implement a simple multi-threaded counter. Create a class 'SharedCounter` with an integer field 'count' initialized to 0. Add a synchronized method `increment()` that increments 'count'. Then, create two threads that each call 'increment()` 10000 times. The main method should wait for both threads to complete and then print the final value of 'count`.

The expected final count should be 20000.

Code:

```
public class SharedCounter {
private int count = 0;
public synchronized void increment() {
   count++;
}
public int getCount() {
return count;
}
public static void main(String[] args) throws InterruptedException {
```

```
SharedCounter counter = new SharedCounter();
Runnable task = () -> {
for (int i = 0;i < 10000; i++) {
counter.increment();
}
};
Thread thread1 = new Thread(task);
Thread thread2 = new Thread(task);
thread1.start();
thread2.start();
thread1.join(); // Wait for thread1 to finish
thread2.join(); // Wait for thread2 to finish
System.out.println("Final count: "+ counter.getCount());
}
}
```

Output:

```
Final count: 20000
```