Use of the try-catch-finally block to handle exceptions that occur during the transfer. Throwing an exception when the transfer amount is greater than the available balance in the customer's account. Write a JAVA code for the same

```java
public class excep {
private double balance;
public excep(double initialBalance) {
balance = initialBalance;
}
public void transfer(double amount, excep recipient) throws
InsufficientFundsException {
try {
if (amount > balance) {
throw new InsufficientFundsException("Transfer amount exceeds available balance");
} else {
balance -= amount;
recipient.balance += amount;
System.out.println("Transfer successful!");
}
} catch (InsufficientFundsException e) {
System.out.println("Transfer failed: " + e.getMessage());
throw e;
}
}
public static void main(String[] args) {
excep account1 = new excep(1000.0);
excep account2 = new excep(500.0);
try {
System.out.println("Acc1 balance before transfer: " + account1.balance);
System.out.println("Acc2 balance before transfer: " + account2.balance);
account1.transfer(500.0, account2);
// account2.transfer(1000.0, account1);
} catch (InsufficientFundsException e) {
// Handle the exception here
}
finally
{
System.out.println("Acc1 balance after transfer: " + account1.balance);
System.out.println("Acc2 balance after transfer: " + account2.balance);
}
}
}
class InsufficientFundsException extends Exception {
public InsufficientFundsException(String message) {
super(message);
}
}
```

Implementation of Generic programming.

```java
import java.util.ArrayList;
import java.util.NoSuchElementException;
public class Stack<T> {
private ArrayList<T> items;
public Stack() {
items = new ArrayList<T>();
}
public void push(T item) {
items.add(item);
}
public T pop() {
if (isEmpty()) {
throw new NoSuchElementException("Stack is empty");
}
return items.remove(items.size() - 1);
}
public T peek() {
if (isEmpty()) {
throw new NoSuchElementException("Stack is empty");
}
return items.get(items.size() - 1);
}
public boolean isEmpty() {
return items.isEmpty();
}
public int size() {
return items.size();
}
public static void main(String[] args) {
Stack<Integer> intStack = new Stack<Integer>();
intStack.push(20);
intStack.push(30);
intStack.push(23);
System.out.println("Top element: " + intStack.peek());
System.out.println("Size of stack: " + intStack.size());
while (!intStack.isEmpty()) {
System.out.println(intStack.pop());
}
}
}
```

Implementation of Multithreaded program:

```java
import java.util.Random;
public class th {
public static void main(String[] args) {
Random random = new Random();
NumberGenerator numberGenerator = new NumberGenerator(random);
SquareCalculator squareCalculator = new SquareCalculator();
CubeCalculator cubeCalculator = new CubeCalculator();
Thread generatorThread = new Thread(numberGenerator);
Thread squareThread = new Thread(squareCalculator);
Thread cubeThread = new Thread(cubeCalculator);
generatorThread.start();
squareThread.start();
cubeThread.start();
}
}
class NumberGenerator implements Runnable {
private final Random random;
public NumberGenerator(Random random) {
this.random = random;
}
public void run() {
while (true) {
int number = random.nextInt(10);
if (number % 2 == 0) {
```

```java
SquareCalculator.handleNumber(number);
} else {
CubeCalculator.handleNumber(number);
}
try {
Thread.sleep(1000);
} catch (InterruptedException e) {
e.printStackTrace();
}
}
}
}
class SquareCalculator implements Runnable {
public static synchronized void handleNumber(int number) {
System.out.println("Received an even number: " + number);
int square = number * number;
System.out.println("Square of the number: " + square);
}
public void run() {
// This thread doesn't need to do anything, as the handleNumber() method
// is static and synchronized, so it can be called from any thread.
}
}
class CubeCalculator implements Runnable {
public static synchronized void handleNumber(int number) {
System.out.println("Received an odd number: " + number);
int cube = number * number * number;
System.out.println("Cube of the number: " + cube);
}
public void run() {
// This thread doesn't need to do anything, as the handleNumber() method
// is static and synchronized, so it can be called from any thread.
}
}
```