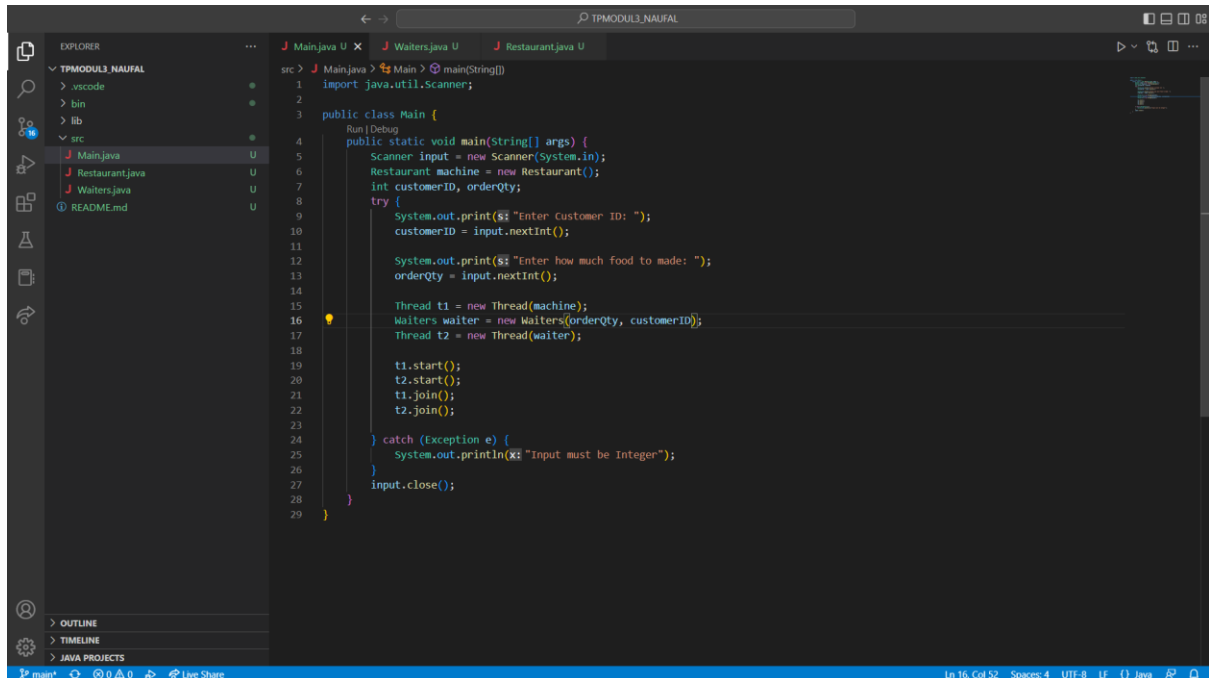


TUGAS PENDAHULUAN MODUL OOP 2022

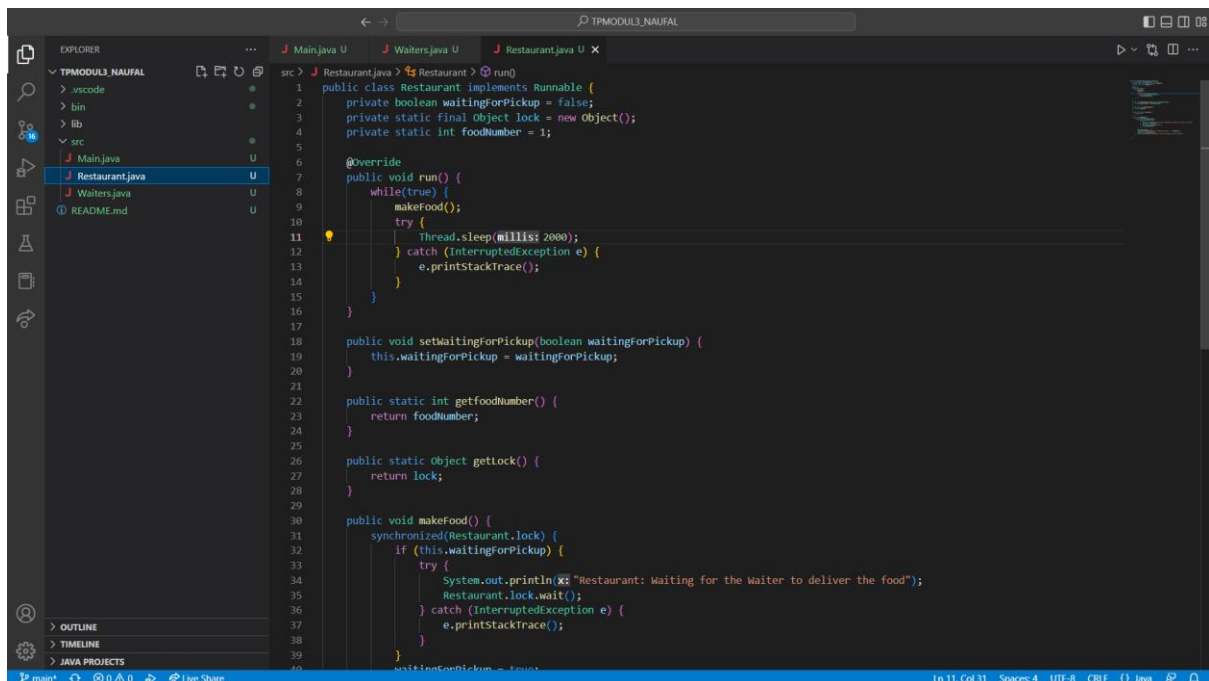
NAUFAL EKA PRASETYA

1202210109

Berikut saya lampirkan hasil pengerjaan saya dari tugas pendahuluan 3 mengenai konsep Exception and Multithreading.



```
src > J Main.java > Main > main(String[])
1 import java.util.Scanner;
2
3 public class Main {
4     Run/Debug
5     public static void main(String[] args) {
6         Scanner input = new Scanner(System.in);
7         Restaurant machine = new Restaurant();
8         int customerID, orderQty;
9         try {
10             System.out.print("Enter Customer ID: ");
11             customerID = input.nextInt();
12
13             System.out.print("Enter how much food to made: ");
14             orderQty = input.nextInt();
15
16             Thread t1 = new Thread(machine);
17             Waiters waiter = new Waiters(orderQty, customerID);
18             Thread t2 = new Thread(waiter);
19
20             t1.start();
21             t2.start();
22             t1.join();
23             t2.join();
24         } catch (Exception e) {
25             System.out.println("Input must be Integer");
26         }
27         input.close();
28     }
29 }
```



```
src > J Restaurant.java > Restaurant > run()
1 public class Restaurant implements Runnable {
2     private boolean waitingForPickup = false;
3     private static final Object lock = new Object();
4     private static int foodNumber = 1;
5
6     @Override
7     public void run() {
8         while(true) {
9             makeFood();
10            try {
11                Thread.sleep(2000);
12            } catch (InterruptedException e) {
13                e.printStackTrace();
14            }
15        }
16    }
17
18    public void setWaitingForPickup(boolean waitingForPickup) {
19        this.waitingForPickup = waitingForPickup;
20    }
21
22    public static int getFoodNumber() {
23        return foodNumber;
24    }
25
26    public static Object getLock() {
27        return lock;
28    }
29
30    public void makeFood() {
31        synchronized(Restaurant.lock) {
32            if (this.waitingForPickup) {
33                try {
34                    System.out.println("Restaurant: Waiting for the waiter to deliver the food");
35                    Restaurant.lock.wait();
36                } catch (InterruptedException e) {
37                    e.printStackTrace();
38                }
39            }
40        }
41    }
42}
```

This screenshot shows the implementation of the `Waiters` class in `Waiters.java`. The class implements the `Runnable` interface. It has private attributes `orderQty` and `customerID`, and a static attribute `foodPrice` set to 25000. The `run()` method contains a `while(true)` loop that calls `getFood()`, sleeps for 6000 milliseconds, and catches `InterruptedException`. The `orderInfo()` method prints the customer ID, number of foods, and total price. The `getFood()` method is synchronized and updates the restaurant's food count and pickup status.

```
1 public class Waiters implements Runnable {
2     private final int orderQty;
3     private final int customerID;
4     static int foodPrice = 25000;
5
6     public Waiters(int orderQty, int customerID) {
7         this.orderQty = orderQty;
8         this.customerID = customerID;
9     }
10
11     @Override
12     public void run() {
13         while(true) {
14             getFood();
15             try {
16                 Thread.sleep(6000);
17             } catch (InterruptedException e) {
18                 e.printStackTrace();
19             }
20         }
21     }
22
23     public void orderInfo() {
24         System.out.println("=====");
25         System.out.println("Customer ID: " + customerID);
26         System.out.println("Number of Food: " + orderQty);
27         System.out.println("Total Price: " + orderQty * foodPrice);
28         System.out.println("=====");
29     }
30
31     public void getFood() {
32         synchronized(Restaurant.getLock()) {
33             System.out.println("Waiter: Food is ready to deliver");
34             Restaurant restaurant = new Restaurant();
35             restaurant.setWaitingForPickup(waitingForPickup: false);
36
37             if (Restaurant.getFoodNumber() == this.orderQty + 1) {
38                 orderInfo();
39             }
40         }
41     }
42 }
```

This screenshot shows the `Main` class in `Main.java`, which uses a `Scanner` to take user input for customer ID and food quantity. The terminal output shows the program's execution with user inputs of 1234 and 2, resulting in a total price of 50000.

```
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         Restaurant machine = new Restaurant();
7         int customerID, orderQty;
8         try {
9             System.out.print("Enter Customer ID: ");
10            customerID = input.nextInt();
11
12            System.out.print("Enter how much food to made: ");
13            orderQty = input.nextInt();
14        }
15    }
16 }
```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! <https://aka.ms/PSWindows>

PS C:\Users\vaufa\Documents\Visual Code\Praktikum Pemrograman Berorientasi Objek\OOP-CACA-NAUFAL-1202210109\TPMODUL3_NAUFA\ > .\Main.exe
Enter Customer ID: 1234
Enter how much food to made: 2
Restaurant: Making Food number 1
Restaurant: Telling the waiter to get the food
Waiter: Food is ready to deliver
Waiter: Tell the Restaurant to make another Food
Restaurant: Waiting for the waiter to deliver the food
Waiter: Food is ready to deliver
Waiter: Tell the Restaurant to make another Food
Restaurant: Making Food number 2
Restaurant: Telling the waiter to get the food
Restaurant: Waiting for the waiter to deliver the food
Waiter: Food is ready to deliver
=====
Customer ID: 1234
Number of Foods: 2
Total Price: 50000
=====