Module 3: Functions, OOP, modules, Errors & Exceptions

Case Study III

Domain – E-Commerce

Business challenge/requirement
GoodsKart—largest ecommerce company of Indonesia with revenue of $2B+ acquired another ecommerce company FairDeal. FairDeal has its own IT system to maintain records of customer, sales etc. For ease of maintenance and cost savings GoodsKart is integrating customer databases of both the organizations hence customer data of FairDeal has to be converted in GoodsKart Customer Format.

Key issues
GoodsKart customer data has more fields than in FairDeal customer data. Hence FairDeal data needs to be split and stored in GoodsKart Customer Object Oriented Data Structure
Considerations
System should convert the data at run time

Business benefits
GoodsKart can eventually sunset IT systems of FairDeal and reduce IT cost by 20-30%

Approach to Solve
You have to use fundamentals of Python taught in module 3.
1. Read FairDealCustomerData.csv
2. Name field contains full name – use regular expression to separate title, first name, last name
3. Store the data in Customer Class
4. Create Custom Exception – CustomerNotAllowedException
5. Pass a customer to function "createOrder" and throw CustomerNotAllowedException in case of blacklisted value is 1

Enhancements for code
You can try these enhancements in code
1. Change function createOrder to take productname and product code as input
2. Create Class Order
Return object of type Order in case customer is eligible

To accomplish the integration of FairDeal's customer data into GoodsKart's system, we need to follow the steps outlined. Below is the Python code to achieve the desired transformation and to handle the business requirements specified.

Step-by-Step Solution

1. **Read FairDealCustomerData.csv**
2. **Use Regular Expressions to Split Name Field**
3. **Store Data in Customer Class**
4. **Create Custom Exception - `CustomerNotAllowedException`**
5. **Implement `createOrder` Function to Handle Blacklisted Customers**

```
import re
import csv

# Define the Customer class
class Customer:
    def __init__(self, title, first_name, last_name, is_blacklisted):
        self.title = title
        self.first_name = first_name
        self.last_name = last_name
        self.is_blacklisted = is_blacklisted

    def __str__(self):
        return f"{self.title} {self.first_name} {self.last_name} - Blacklisted: {self.is_blacklisted}"

# Define the Order class
class Order:
    def __init__(self, customer, product_name, product_code):
        self.customer = customer
        self.product_name = product_name
        self.product_code = product_code

    def __str__(self):
        return f"[+] --> Order for {self.customer.first_name} {self.customer.last_name}:
{self.product_name} (Code: {self.product_code})"

# Define the custom exception
class CustomerNotAllowedException(Exception):
    def __init__(self, customer):
        self.customer = customer
        super().__init__(f"[-] --> Customer {customer.first_name} {customer.last_name} is blacklisted
and not allowed to create an order.")
```

```
# Function to create order
def createOrder(customer, product_name, product_code):
    if customer.is_blacklisted:
        raise CustomerNotAllowedException(customer)
    return Order(customer, product_name, product_code)

# Function to parse title and first name using regular expressions
def parse_title_first_name(data):

    # Define the regular expression pattern
    pattern = re.compile(r'([^,]+),\s*([^,]+)\.\s*([^,]+),\s*([^,]+)')

    # Find all matches in the data
    matches = pattern.findall(data)

    # Print the matches
    if matches:
        for match in matches:
            last_name = match[0].strip()
            title = match[1].strip()
            first_name = match[2].strip()
            flag = match[3].strip()
            #print((last_name, title, first_name, flag))
            return last_name, title, first_name, flag
    else:
        print("No matches found")
        return None, None, None, None

# Read CSV and process data
customers = []

# Open the file in read mode
with open('FairDealCustomerData.csv', 'r') as file:
    # Loop through each line in the file
    for line in file:
        # Process the line (e.g., print it)
        #print(line.strip())
        last_name, title, first_name, blacklisted = parse_title_first_name(line)
        if title and first_name:
            customer = Customer(title, first_name, last_name, bool(int(blacklisted)))
            customers.append(customer)
        else:
            print(f"Skipping invalid entry: {line}")

# Example usage
for customer in customers:
    print(customer)
```

```
try:
    order = createOrder(customer, "SampleProduct", "SP123")
    print(order)
except CustomerNotAllowedException as e:
    print(e)
```

Explanation

1. **Customer Class:**
    - Contains attributes for title, first name, last name, and blacklist status.
    - String representation method to print customer details.
2. **Order Class:**
    - Contains attributes for the customer, product name, and product code.
    - String representation method to print order details.
3. **Custom Exception - `CustomerNotAllowedException`:**
    - Raised when a blacklisted customer attempts to create an order.
4. **createOrder Function:**
    - Takes customer, product name, and product code as input.
    - Checks if the customer is blacklisted and raises an exception if they are.
    - Returns an `Order` object if the customer is not blacklisted.
5. **parse_name Function:**
    - Uses a regular expression to parse the title, first name, and last name from the full name.
6. **CSV Reading and Processing:**
    - Reads the CSV file and processes each row to create `Customer` objects.
    - Uses the `parse_name` function to split the full name into title, first name, and last name.

Enhancements

- **createOrder Function:** Modified to take `product_name` and `product_code` as inputs.
- **Order Class:** Added to encapsulate order details.

This code will read the CSV file, process the data, and handle the creation of orders while managing blacklisted customers appropriately.

```
[john@squid 777_m3_datasets_v1.0]$
[john@squid 777_m3_datasets_v1.0]$ python3 mod3_use-case_III.py
Mr Owen Harris Braund - Blacklisted: True
[-] --> Customer Owen Harris Braund is blacklisted and not allowed to create an order.
Miss Laina Heikkinen - Blacklisted: False
[+] --> Order for Laina Heikkinen: SampleProduct (Code: SP123)
Mr William Henry Allen - Blacklisted: False
[+] --> Order for William Henry Allen: SampleProduct (Code: SP123)
Mr James Moran - Blacklisted: False
[+] --> Order for James Moran: SampleProduct (Code: SP123)
Mr Timothy J McCarthy - Blacklisted: False
[+] --> Order for Timothy J McCarthy: SampleProduct (Code: SP123)
Miss Elizabeth Bonnell - Blacklisted: False
[+] --> Order for Elizabeth Bonnell: SampleProduct (Code: SP123)
Mr Anders Johan Andersson - Blacklisted: False
[+] --> Order for Anders Johan Andersson: SampleProduct (Code: SP123)
Mr Charles Eugene Williams - Blacklisted: True
[-] --> Customer Charles Eugene Williams is blacklisted and not allowed to create an order.
Mrs Fatima Masselmani - Blacklisted: False
[+] --> Order for Fatima Masselmani: SampleProduct (Code: SP123)
Mr Joseph J Fynney - Blacklisted: False
[+] --> Order for Joseph J Fynney: SampleProduct (Code: SP123)
Mr Lawrence Beesley - Blacklisted: False
[+] --> Order for Lawrence Beesley: SampleProduct (Code: SP123)
Mr William Thompson Sloper - Blacklisted: False
[+] --> Order for William Thompson Sloper: SampleProduct (Code: SP123)
Mr Farred Chehab Emir - Blacklisted: False
[+] --> Order for Farred Chehab Emir: SampleProduct (Code: SP123)
Mr Lalio Todoroff - Blacklisted: False
[+] --> Order for Lalio Todoroff: SampleProduct (Code: SP123)
Don Manuel E Uruchurtu - Blacklisted: True
[-] --> Customer Manuel E Uruchurtu is blacklisted and not allowed to create an order.
Miss Mary Agatha Glynn - Blacklisted: False
[+] --> Order for Mary Agatha Glynn: SampleProduct (Code: SP123)
Mr Edward H Wheadon - Blacklisted: False
```