Problem 1: Bank Account Create a class representing a bank account with attributes like account number, account holder name, and balance. Implement methods to deposit and withdraw money from the account.

```
class BankAccount:
    def __init__(self, account_number, account_holder_name, balance):
        self.account_number = account_number
        self.account_holder_name = account_holder_name
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited {amount} successfully. New balance: {self.balance}")

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
            print(f"Withdrawn {amount} successfully. New balance: {self.balance}")
        else:
            print("Insufficient balance")
```

Problem 2: Employee Management Create a class representing an employee with attributes like employee ID, name, and salary. Implement methods to calculate the yearly bonus and display employee details.

```
class Employee:
    def __init__(self, employee_id, name, salary):
        self.employee_id = employee_id
        self.name = name
        self.salary = salary

    def calculate_bonus(self):
        return 0.1 * self.salary  # 10% of salary

    def display_details(self):
        print(f"Employee ID: {self.employee_id}")
        print(f"Name: {self.name}")
        print(f"Salary: {self.salary}")
        print(f"Yearly Bonus: {self.calculate_bonus()}")
```

Problem 3: Vehicle Rental Create a class representing a vehicle rental system. Implement methods to rent a vehicle, return a vehicle, and display available vehicles.

```
class VehicleRental:
    def __init__(self):
        self.available_vehicles = ["Car", "Bike", "Truck"]
```

```
    def rent_vehicle(self, vehicle):
        if vehicle in self.available_vehicles:
            self.available_vehicles.remove(vehicle)
            print(f"{vehicle} rented successfully")
        else:
            print(f"{vehicle} not available for rent")

    def return_vehicle(self, vehicle):
        self.available_vehicles.append(vehicle)
        print(f"{vehicle} returned successfully")

    def display_available_vehicles(self):
        print("Available vehicles:", self.available_vehicles)
```

Problem 4: Library Catalog Create classes representing a library and a book. Implement methods to add books to the library, borrow books, and display available books.

```
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author
        self.available = True

class Library:
    def __init__(self):
        self.books = []

    def add_book(self, book):
        self.books.append(book)

    def borrow_book(self, title):
        for book in self.books:
            if book.title == title and book.available:
                book.available = False
                print(f"{title} borrowed successfully")
                return
        print(f"{title} not available for borrowing")

    def display_available_books(self):
        print("Available books:")
        for book in self.books:
            if book.available:
                print(f"{book.title} by {book.author}")
```

Problem 5: Product Inventory Create classes representing a product and an inventory system. Implement methods to add products to the inventory, update product quantity, and display available products.

```
class Product:
```

```
    def __init__(self, name, price, quantity):
        self.name = name
        self.price = price
        self.quantity = quantity

class Inventory:
    def __init__(self):
        self.products = []

    def add_product(self, product):
        self.products.append(product)

    def update_quantity(self, name, quantity):
        for product in self.products:
            if product.name == name:
                product.quantity += quantity
                print(f"Quantity of {name} updated successfully. New quantity: {product.quantity}")
                return
        print(f"{name} not found in inventory")

    def display_available_products(self):
        print("Available products:")
        for product in self.products:
            print(f"{product.name}: {product.quantity}")
```

Problem 6: Shape Calculation Create a class representing a shape with attributes like length, width, and height. Implement methods to calculate the area and perimeter of the shape.

```
class Shape:
    def __init__(self, length, width, height):
        self.length = length
        self.width = width
        self.height = height

    def calculate_area(self):
        raise NotImplementedError("Subclass must implement abstract method")

    def calculate_perimeter(self):
        raise NotImplementedError("Subclass must implement abstract method")

class Rectangle(Shape):
    def calculate_area(self):
        return self.length * self.width

    def calculate_perimeter(self):
        return 2 * (self.length + self.width)

class Triangle(Shape):
    def calculate_area(self):
        return 0.5 * self
```

Problem 7: Student Management Create a class representing a student with attributes like student ID, name, and grades. Implement methods to calculate the average grade and display student details.

```
class Student:
    def __init__(self, student_id, name, grades):
        self.student_id = student_id
        self.name = name
        self.grades = grades

    def calculate_average_grade(self):
        return sum(self.grades) / len(self.grades)

    def display_details(self):
        print(f"Student ID: {self.student_id}")
        print(f"Name: {self.name}")
        print(f"Grades: {self.grades}")
        print(f"Average Grade: {self.calculate_average_grade()}")
```

Problem 8: Email Management Create a class representing an email with attributes like sender, recipient, and subject. Implement methods to send an email and display email details.

```
class Email:
    def __init__(self, sender, recipient, subject):
        self.sender = sender
        self.recipient = recipient
        self.subject = subject

    def send_email(self):
        print(f"Email sent from {self.sender} to {self.recipient} with subject '{self.subject}'")

    def display_details(self):
        print("Email Details:")
        print(f"Sender: {self.sender}")
        print(f"Recipient: {self.recipient}")
        print(f"Subject: {self.subject}")
```

Problem 9: Social Media Profile Create a class representing a social media profile with attributes like username and posts. Implement methods to add posts, display posts, and search for posts by keyword.

```
class SocialMediaProfile:
    def __init__(self, username):
        self.username = username
        self.posts = []

    def add_post(self, post):
        self.posts.append(post)

    def display_posts(self):
```

```
        print(f"Posts by {self.username}:")
        for post in self.posts:
            print(post)

    def search_posts(self, keyword):
        print(f"Posts containing '{keyword}':")
        for post in self.posts:
            if keyword in post:
                print(post)
```

Problem 10: ToDo List Create a class representing a ToDo list with attributes like tasks and due dates. Implement methods to add tasks, mark tasks as completed, and display pending tasks.

```
class ToDoList:
    def __init__(self):
        self.tasks = {}

    def add_task(self, task, due_date):
        self.tasks[task] = due_date

    def mark_task_completed(self, task):
        if task in self.tasks:
            del self.tasks[task]
            print(f"{task} marked as completed")
        else:
            print(f"{task} not found in ToDo list")

    def display_pending_tasks(self):
        print("Pending tasks:")
        for task, due_date in self.tasks.items():
            print(f"{task} (Due Date: {due_date})")
```