Name: Keerthika Nagarajan **Superset ID:** 5370583

College: Saveetha Engineering College

Assignment-Courier Management System

TASK 1: Control Flow Statements

 # TASK 1.1 - Check order status

```
def check_order_status(self, tracking_number: str) -> str:
    conn = self._get_connection()
    if not conn:
        return "Database connection failed"

    try:
        cursor = conn.cursor(dictionary=True)
        cursor.execute("SELECT Status FROM Courier WHERE TrackingNumber = %s", (tracking_number,))
        result = cursor.fetchone()
        return result['Status'] if result else "Not Found"
    except Error as e:
        raise TrackingNumberNotFoundException(tracking_number)
    finally:
        conn.close()
```

```
Enter your choice (0-18): 1
Enter tracking number: TRK123456

=== Order Status ===
Status: Delivered
```

 # TASK 1.2 - Categorize parcel

```
def categorize_parcel(self, weight: float) -> str:
    if weight < 1:
        return "Light"
    elif weight < 5:
        return "Medium"
    return "Heavy"
```

```
Enter your choice (0-18): 2
Enter weight (kg): 4

=== Parcel Category ===
Category: Medium
```

 # TASK 1.3 - User authentication

```
def authenticate_user(self, email: str, password: str, is_employee: bool = False) -> bool:
  conn = self._get_connection()
  if not conn:
    return False

  try:
    cursor = conn.cursor(dictionary=True)
    table = "Employee" if is_employee else "User"
    cursor.execute(f"SELECT * FROM {table} WHERE Email = %s AND Password = %s", (email,
password))
    return cursor.fetchone() is not None
  finally:
    conn.close()
```

```
Enter your choice (0-18): 3
Enter email: john.smith@email.com
Enter password: jsmith123
Employee login? (y/n): n

=== Authentication ===
Login successful as customer
```

 # TASK 1.4 - Assign courier

```
def assign_courier(self, courier_id: int, employee_id: int) -> bool:
  conn = self._get_connection()
  if not conn:
    return False

  try:
    cursor = conn.cursor()
    # Verify employee exists
    cursor.execute("SELECT * FROM Employee WHERE EmployeeID = %s", (employee_id,))
    if not cursor.fetchone():
      raise InvalidEmployeeIdException(employee_id)

    query = """
    INSERT INTO CourierEmployeeAssignment
    (CourierID, EmployeeID, AssignmentDate, Status)
    VALUES (%s, %s, NOW(), 'Assigned')
    """
    cursor.execute(query, (courier_id, employee_id))
    conn.commit()
    return cursor.rowcount > 0
  except Error as e:
    print(f"Assignment error: {e}")
```

```
    return False
  finally:
    conn.close()
```

```
Enter your choice (0-18): 4
Enter courier ID: 5
Enter employee ID: 3

=== Courier Assignment ===
Courier assigned successfully
```

TASK 2: Loops and Iteration

 # TASK 2.1 - Display customer orders

```
def get_customer_orders(self, user_id: int) -> List[Dict]:
  conn = self._get_connection()
  if not conn:
    return []


  try:
    cursor = conn.cursor(dictionary=True)
    query = """
    SELECT c.* FROM Courier c
    JOIN Orders o ON c.OrderID = o.OrderID
    WHERE o.UserID = %s
    """
    cursor.execute(query, (user_id,))
    return cursor.fetchall()
  finally:
    conn.close()
```

```
Enter your choice (0-18): 5
Enter customer ID: 2

=== Customer Orders ===
1. TRK789012 - In Transit (5.00kg)
```

 # TASK 2.2 - Track courier location

```
def get_tracking_history(self, courier_id: int) -> List[Dict]:
  conn = self._get_connection()
  if not conn:
    return []

  try:
    cursor = conn.cursor(dictionary=True)
    cursor.execute("""
      SELECT * FROM TrackingHistory
```

```
        WHERE CourierID = %s
        ORDER BY UpdateTime
        """, (courier_id,))
      return cursor.fetchall()
    finally:
      conn.close()
```

```
Enter your choice (0-18): 6
Enter courier ID: 1

=== Tracking History ===
2023-06-01 14:00:00: Processing
2023-06-02 08:30:00: In Transit
2023-06-03 10:30:00: Delivered
```

TASK 3: Arrays and Data Structures

 # TASK 3.1 - Tracking history array

```
def get_tracking_array(self, courier_id: int) -> List[str]:
    history = self.get_tracking_history(courier_id)
    return [f"{h['UpdateTime']}: {h['Status']}" for h in history]
```

```
Enter your choice (0-18): 7
Enter courier ID: 2

=== Tracking History Array ===
1. 2023-06-02 12:30:00: Processing
2. 2023-06-03 09:15:00: In Transit
```

 # TASK 3.2 - Find nearest courier

```
def find_nearest_courier(self, location_id: int) -> List[Dict]:
    conn = self._get_connection()
    if not conn:
        return []

    try:
        cursor = conn.cursor(dictionary=True)
        query = """
        SELECT e.* FROM Employee e
        JOIN EmployeeLocationAssignment ela ON e.EmployeeID = ela.EmployeeID
        WHERE ela.LocationID = %s AND e.Role = 'Driver' AND e.Status = 'Active'
        """
        cursor.execute(query, (location_id,))
        return cursor.fetchall()
    finally:
        conn.close()
```

```
Enter your choice (0-18): 8
Enter location ID: 1

=== Available Couriers ===
1. Driver 1 (ID: 3)
```

TASK 4: Strings and Functions

 # TASK 4.2 - Customer data validation

```
def validate_data(self, data: str, field_type: str) -> bool:
    if field_type == "name":
        return bool(re.match(r"^[A-Z][a-z]+( [A-Z][a-z]+)*$", data))
    elif field_type == "phone":
        return bool(re.match(r"^\d{3}-\d{3}-\d{4}$", data))
    elif field_type == "address":
        return bool(re.match(r"^[\w\s,.-]+$", data))
    return False
```

```
Enter your choice (0-18): 9
Enter data: Emily Davis
Field type (name/phone/address): name

=== Validation Result ===
Valid
```

 # TASK 4.3 - Address formatting

```
def format_address(self, street: str, city: str, state: str, zip_code: str) -> str:
    return f"{street.title()}, {city.title()}, {state.upper()} {zip_code}"
```

```
Enter your choice (0-18): 10
Enter street: 123 main st
Enter city: anytown
Enter state: ca
Enter zip code: 90210

=== Formatted Address ===
123 Main St, Anytown, CA 90210
```

 # TASK 4.4 - Order confirmation email

```
def generate_confirmation_email(self, order_id: int) -> str:
    conn = self._get_connection()
    if not conn:
        return "Database connection failed"

    try:
        cursor = conn.cursor(dictionary=True)
```

```
    query = """
    SELECT u.Name, o.OrderID, c.ReceiverAddress, o.EstimatedDelivery, c.TrackingNumber
    FROM Orders o
    JOIN User u ON o.UserID = u.UserID
    JOIN Courier c ON o.OrderID = c.OrderID
    WHERE o.OrderID = %s
    """
    cursor.execute(query, (order_id,))
    result = cursor.fetchone()

    if result:
      return f"""
      Order Confirmation for {result['Name']}
      Order #: {result['OrderID']}
      Delivery Address: {result['ReceiverAddress']}
      Estimated Delivery: {result['EstimatedDelivery']}
      Tracking Number: {result['TrackingNumber']}
      """
    return "Order not found"
  finally:
    conn.close()
```

```
Enter your choice (0-18): 11
Enter order ID: 2

=== Order Confirmation ===

                Order Confirmation for Sarah Johnson
                Order #: 2
                Delivery Address: 321 Elm St, Anywhere, USA
                Estimated Delivery: 2023-06-03
                Tracking Number: TRK789012
```

 # TASK 4.5 - Shipping cost calculation

```
def calculate_shipping_cost(self, source: str, destination: str, weight: float) -> float:
  # Simplified calculation - in real system would use distance API
  base_cost = 5.0
  distance_cost = len(source + destination) * 0.1  # Fake distance calculation
  weight_cost = weight * 0.5
  return round(base_cost + distance_cost + weight_cost, 2)
```

```
Enter your choice (0-18): 12
Enter source location: New York
Enter destination: Los Angeles
Enter weight (kg): 2.5

=== Shipping Cost ===
Estimated cost: $8.15
```

 # TASK 4.6 - Password generator

```
def generate_password(self, length: int = 12) -> str:
    chars = string.ascii_letters + string.digits + string.punctuation
    while True:
        pwd = ''.join(random.choice(chars) for _ in range(length))
        if (any(c.islower() for c in pwd) and
                any(c.isupper() for c in pwd) and
                any(c.isdigit() for c in pwd) and
                any(c in string.punctuation for c in pwd)):
            return pwd
```

```
Enter your choice (0-18): 13
Length (8-20): 12

=== Generated Password ===
a-C)$tK4.V*%
```

 # TASK 4.7 - Find similar addresses

```
def find_similar_addresses(self, search_term: str) -> List[str]:
    conn = self._get_connection()
    if not conn:
        return []

    try:
        cursor = conn.cursor(dictionary=True)
        query = """
        SELECT DISTINCT ReceiverAddress FROM Courier
        WHERE ReceiverAddress LIKE %s
        LIMIT 5
        """
        cursor.execute(query, (f"%{search_term}%",))
        return [r['ReceiverAddress'] for r in cursor.fetchall()]
    finally:
        conn.close()
```

```
Enter your choice (0-18): 14
Enter address search term: SomeWhere

=== Similar Addresses ===
1. 987 Cedar Ln, Somewhere, USA
2. 456 Oak Ave, Somewhere, USA
```

TASK 5: OOP

 # TASK 5 - OOP Demonstration

```
def create_user(self, name: str, email: str, contact_number: str = "000-000-0000",
            address: str = "Not provided") -> User:
    conn = self._get_connection()
    if not conn:
        return None

    try:
        cursor = conn.cursor()
        password = self.generate_password()
        cursor.execute(
            "INSERT INTO User (Name, Email, Password, ContactNumber, Address) VALUES (%s, %s, %s,
%s, %s)",
            (name, email, password, contact_number, address)
        )
        conn.commit()
        return User(cursor.lastrowid, name, email, password, contact_number, address)
    except Error as e:
        print(f"Database error: {e}")
        return None
    finally:
        conn.close()
```

```
Enter your choice (0-18): 15
Enter user name: Lewis Hamilton
Enter email: lewis44@gmail.com

=== User Created ===
New user: User(ID: 7, Name: Lewis Hamilton)
```

TASK 8: Collections

 # TASK 8 – Courier Added

```
def add_courier_to_collection(self, courier: Courier):
    self.company.couriers.append(courier)
```

```
Enter your choice (0-18): 16
Enter sender name: John Smith
Enter receiver name: Alice Brown
Enter weight (kg): 2.5

=== Courier Added ===
Added to collection: Courier(ID: 0, Tracking: TRK1000, Status: Processing, Weight: 2.5kg)
```

 # TASK 8 – Collection Stats

```
def get_collection_stats(self):
  return {
    'couriers': len(self.company.couriers),
    'employees': len(self.company.employees),
    'locations': len(self.company.locations)
  }
```

```
=== Collection Stats ===
Couriers: 1
Employees: 0
Locations: 0
```

TASK 9: Database Reports

 # TASK 9 - Revenue Report

```
def generate_revenue_report(self, start_date=None, end_date=None):
  """Generate revenue report between specified dates"""
  conn = self._get_connection()
  if not conn:
    return

  try:
    cursor = conn.cursor(dictionary=True)

    # Handle date inputs
    if not start_date or not end_date:
      # Default to last 30 days if dates not provided
      end_date = datetime.now().strftime('%Y-%m-%d')
      start_date = (datetime.now() - timedelta(days=30)).strftime('%Y-%m-%d')
      print(f"\nUsing default date range: {start_date} to {end_date}")

    # Build query
    query = """
    SELECT
      DATE(PaymentDate) AS date,
      COUNT(*) AS shipments,
      SUM(Amount) AS revenue,
```

```
        AVG(Amount) AS avg_order
    FROM Payment
    WHERE PaymentDate BETWEEN %s AND %s
    GROUP BY DATE(PaymentDate)
    ORDER BY date DESC
    """

    # Execute with date parameters
    cursor.execute(query, (
        f"{start_date} 00:00:00",
        f"{end_date} 23:59:59"
    ))

    results = cursor.fetchall()

    if results:
        total_shipments = sum(row['shipments'] for row in results)
        total_revenue = sum(row['revenue'] for row in results)

        print(f"\nREVENUE REPORT ({start_date} to {end_date})")
        print(tabulate(results, headers="keys", tablefmt="grid"))
        print(f"\nTOTAL: {total_shipments} shipments, ${total_revenue:.2f} revenue")
    else:
        print(f"\nNo payment records found between {start_date} and {end_date}")

except Error as e:
    print(f"\nDatabase error: {e}")
finally:
    conn.close()
```

```
Enter date range (YYYY-MM-DD) or leave blank for last 30 days
Start date: 2023-06-01
End date: 2023-06-30

REVENUE REPORT (2023-06-01 to 2023-06-30)
+------------+-------------+-----------+-------------+
| date       |   shipments |   revenue |   avg_order |
+============+=============+===========+=============+
| 2023-06-06 |           1 |     59.99 |       59.99 |
+------------+-------------+-----------+-------------+
| 2023-06-05 |           1 |    120    |      120    |
+------------+-------------+-----------+-------------+
| 2023-06-04 |           1 |     89.99 |       89.99 |
+------------+-------------+-----------+-------------+
| 2023-06-03 |           1 |     15.99 |       15.99 |
+------------+-------------+-----------+-------------+
| 2023-06-02 |           1 |     45.5  |       45.5  |
+------------+-------------+-----------+-------------+
| 2023-06-01 |           1 |     25.99 |       25.99 |
+------------+-------------+-----------+-------------+

TOTAL: 6 shipments, $357.46 revenue
```