Exercise 1: Bank Account Class

Objective:
Create a BankAccount class to model a simple bank account with deposit and withdrawal functionalities.

Tasks:

1. **Define the Class:**
    - Create a class called BankAccount.
    - Add attributes for the account number, account holder's name, and current balance.
2. **Implement Methods:**
    - **deposit(amount)**: Increase the balance by the given amount.
    - **withdraw(amount)**: Decrease the balance by the given amount, ensuring that the account does not go into overdraft. If there isn't enough money, print an error message.
    - **display_balance()**: Print the current balance.
    - *(Optional)* **transfer(target_account, amount)**: Transfer money from one account to another if sufficient funds exist.
3. **Test Your Class:**
   Write a script to create one or more BankAccount objects, perform various operations (deposit, withdraw, transfer), and display the results.

Exercise 2: Employee Management with Inheritance

Objective:
Build a simple employee management system using class inheritance.

Tasks:

1. **Base Class Employee:**
    - Define an Employee class with attributes: name, employee_id, and salary.
    - Create a method **display_info()** that prints the basic details of the employee.
2. **Derived Classes:**
    - **Manager Class:** Inherit from Employee and add an attribute department. Override the display_info() method to include the department.
    - **Engineer Class:** Inherit from Employee and add an attribute specialization. Override the display_info() method to include the specialization.
3. **Testing:**
   Create instances of both Manager and Engineer, then call their display_info() methods to see the differences.

Exercise 3: Geometric Shapes Hierarchy

Objective:
Design a hierarchy of classes to represent geometric shapes.

Tasks:

1. **Abstract Base Class Shape:**
    - Define an abstract class Shape (using the abc module) with an abstract method area().
    - *(Optional)* Also include an abstract method perimeter().
2. **Subclasses:**
    - **Circle:**
        - Attribute: radius.
        - Implement area() as π * radius².
        - Implement perimeter() as 2 * π * radius.
    - **Rectangle:**
        - Attributes: width and height.
        - Implement area() as width * height.
        - Implement perimeter() as 2 * (width + height).
    - **Triangle:**
        - Either use base and height (for area calculation) or three sides (to calculate perimeter) depending on your design.
        - Implement the area method (for example, using 0.5 * base * height if you choose that approach).
3. **Testing:**
   Instantiate each shape, compute its area (and perimeter, if implemented), and print the results.

Exercise 4: Singly Linked List Implementation

Objective:
Implement a basic singly linked list using classes.

Tasks:

1. **Define a Node Class:**
    - Attributes: data and next (pointer to the next node).

2. **Define a LinkedList Class:**
    - Include methods to:
        - **append(data)**: Add a new node with the given data at the end.
        - **insert(position, data)**: Insert a new node at a specified position.
        - **delete(data)**: Remove the first node that contains the specified data.
        - **display()**: Print all elements in the list.
3. **Testing:**
   Create a linked list, perform a series of operations (append, insert, delete), and display the list after each operation.

Exercise 5: Shopping Cart System

Objective:
Develop a simple shopping cart application using classes.

Tasks:

1. **Product Class:**
    - Create a Product class with attributes such as name, price, and quantity.
2. **ShoppingCart Class:**
    - Create a ShoppingCart class to manage products.
    - Implement methods:
        - **add_product(product)**: Add a product to the cart.
        - **remove_product(product_name)**: Remove a product based on its name.
        - **total_price()**: Calculate and return the total price of all products in the cart.
        - *(Optional)* **display_cart()**: List all products in the cart.
3. **Testing:**
   Write a script that simulates a shopping session:
    - Create several Product instances.
    - Add and remove products from the ShoppingCart.
    - Display the cart contents and total price.

Exercise 6: Student Grade Management System

Objective:
Create a simple system to manage student enrollments and grades. Build the following classes:

- **Student**: Represents a student with an ID, name, and a list of enrollments.
- **Course**: Represents a course with a code and title.
- **Enrollment**: Represents the relationship between a student and a course, including a grade.

Tasks:

1. **Student Class:**
    - Include attributes for the student ID, name, and a list to hold Enrollment objects.
    - Implement methods to enroll in a course, record a grade for a course, calculate GPA, and display student info.
2. **Course Class:**
    - Include attributes for the course code and title.
    - Maintain a list of enrollments.
3. **Enrollment Class:**
    - Tie a student to a course and include an attribute for the grade (initially unset).
4. **Testing:**
   Create instances of students and courses. Enroll students in courses, record grades, and display student information including calculated GPA.

Exercise 7: Game Inventory System

Objective:
Design a simple inventory system for a game. Create a hierarchy for items and an inventory to manage them.

Tasks:

1. **Item Class:**
    - Create a base Item class with attributes such as name and weight.
2. **Derived Classes:**
    - **Weapon**: Inherit from Item and add an attribute for damage.
    - **Potion**: Inherit from Item and add an attribute for effect.
3. **Inventory Class:**

- Create an Inventory class that manages a list of items.
- Implement methods to add an item, remove an item by name, and display the inventory.

4. **Testing:**
   Create several item instances (weapons and potions), add them to the inventory, and perform operations like displaying and removing items.

Exercise 8: Parking Lot System

Objective:
Develop a simulation of a parking lot using classes.

Tasks:

1. **Car Class:**
   - Create a Car class with attributes such as license_plate and model.
2. **ParkingSpot Class:**
   - Create a ParkingSpot class representing a single parking spot.
   - Include an attribute to hold a Car object (if parked) and methods to park a car and free the spot.
3. **ParkingLot Class:**
   - Create a ParkingLot class to manage multiple parking spots.
   - Implement methods to park a car (finding the first available spot), remove a car (using the license plate), and display the status of the parking lot.
4. **Testing:**
   Simulate parking and removing cars from the parking lot and display the current status.